СЕРИЯ

МИРОВАЯ ЭЛЕКТРОНИКА

СЕМЕЙСТВО МИКРОКОНТРОЛЛЕРОВ MSP430x2xx

Архитектура
Программирование
Разработка приложений

ДОДЭКА

Москва
Издательский дом «Додэка-XXI»
2010

УДК 621.316.544.1(035.5)
ББК 32.844.1-04я2
С30

Данное издание подготовлено к печати по заказу компании «КОМПЭЛ». Название оригинального документа компании Texas Instruments — «MSP430x2xx Family User's Guide».

С30 Семейство микроконтроллеров MSP430x2xx. Архитектура, программирование, разработка приложений / пер. с англ. Евстифеева А. В. — М. : Додэка-XXI, 2010. — 544 с.: ил. — (Серия «Мировая электроника»). — ISBN 978-5-94120-221-8

Настоящая книга посвящена однокристальным микроконтроллерам со сверхнизким потреблением семейства MSP430x2xx компании Texas Instruments. Данное руководство представляет собой перевод документа «MSP430x2xx Family User's Guide», выпущенного компанией в 2008 году.

В руководстве подробно рассмотрена архитектура ЦПУ MSP430 и MSP430x, описаны система команд и поддерживаемые режимы адресации. Помимо этого, в книге детально описываются различные периферийные модули, реализованные в микроконтроллерах семейства: таймеры, порты ввода/вывода, модули АЦП и ЦАП, модули последовательных интерфейсов USI/USCI и прочие, а также аналоговые модули, такие как модуль операционного усилителя и модуль аналогового компаратора.

Это руководство безо всякого преувеличения можно считать настольной книгой инженера-разработчика, занимающегося проектированием устройств на микроконтроллерах семейства MSP430x2xx. Кроме того, полнота и ясность изложения материала позволяет рекомендовать данную книгу студентам соответствующих специальностей и подготовленным радиолюбителям.

УДК 621.316.544.1(035.5)
ББК 32.844.1-04я2

Все права защищены, никакая часть этого издания не может быть воспроизведена в любой форме или любыми средствами, электронными или механическими, включая фотографирование, ксерокопирование или иные средства копирования или сохранения информации, без письменного разрешения издательства.

ISBN 978-5-94120-221-8 © Издательский дом «Додэка-XXI», 2010

СОДЕРЖАНИЕ

Предисловие 11

Глава 1. Введение 13

1.1. Архитектура 13
1.2. Гибкая система тактирования 14
1.3. Внутрисхемная эмуляция 14
1.4. Адресное пространство 15
1.4.1. Флэш/ПЗУ 16
1.4.2. ОЗУ 16
1.4.3. Периферийные устройства 16
1.4.4. Регистры специальных функций (SFR) 16
1.4.5. Организация памяти 16
1.5. Расширенные возможности семейства MSP430x2xx 17

Глава 2. Сброс, прерывания и режимы работы 19

2.1. Сброс и инициализация системы 19
2.1.1. Сброс по снижению напряжения питания (BOR) 20
2.1.2. Состояние устройства после сброса 21
2.2. Прерывания 21
2.2.1. Немаскируемые прерывания (NMI) 22
2.2.2. Маскируемые прерывания 25
2.2.3. Обработка прерывания 25
2.2.4. Векторы прерываний 27
2.3. Режимы работы 29
2.3.1. Вход в режимы пониженного энергопотребления и выход из них 30
2.4. Принципы программирования устройств с низким энергопотреблением 31
2.5. Подключение неиспользуемых выводов 32

Глава 3. 16-битное RISC ЦПУ MSP430 33

3.1. Введение в ЦПУ 33
3.2. Регистры ЦПУ 35
3.2.1. Счётчик команд (PC) 35
3.2.2. Указатель стека (SP) 35

3.2.3. Регистр состояния (SR) 36
3.2.4. Регистры генератора констант CG1 и CG2 37
3.2.5. Регистры общего назначения R4...R15 38
3.3. Режимы адресации. 39
3.3.1. Регистровый режим адресации 39
3.3.2. Индексный режим адресации 40
3.3.3. Относительный режим адресации 41
3.3.4. Абсолютный режим адресации 42
3.3.5. Косвенный регистровый режим адресации 43
3.3.6. Косвенный регистровый режим адресации с автоинкрементом 44
3.3.7. Непосредственный режим адресации 45
3.4. Система команд 45
3.4.1. Команды с двумя операндами (формат I) 46
3.4.2. Команды с одним операндом (формат II) 46
3.4.3. Команды перехода 48
3.4.4. Время выполнения и размер команд 90
3.4.5. Описание набора команд 92

Глава 4. 16-битное RISC ЦПУ MSP430X 94

4.1. Введение в ЦПУ 94
4.2. Прерывания 96
4.3. Регистры ЦПУ 96
4.3.1. Счётчик команд (PC) 96
4.3.2. Указатель стека (SP) 97
4.3.3. Регистр состояния (SR) 99
4.3.4. Регистры генератора констант CG1 и CG2 100
4.3.5. Регистры общего назначения R4...R15 101
4.4. Режимы адресации. 103
4.4.1. Регистровый режим 104
4.4.2. Индексный режим адресации 105
4.4.3. Относительный режим адресации 110
4.4.4. Абсолютный режим адресации 114
4.4.5. Косвенный регистровый режим адресации 117
4.4.6. Косвенный регистровый режим адресации с автоинкрементом 118
4.4.7. Непосредственный режим адресации 119
4.5. Команды MSP430 и MSP430X 121
4.5.1. Команды MSP430 121
4.5.2. Команды MSP430X 126
4.6. Описание набора команд 137
4.6.1. Подробные описания расширенных команд 137
4.6.2. Команды MSP430 139
4.6.3. Расширенные команды 184
4.6.4. Адресные команды 220

Глава 5. Модуль синхронизации Basic Clock Module+ 230

5.1. Введение 230

5.2. Функционирование модуля синхронизации 231
5.2.1. Возможности модуля синхронизации и приложения с низким энергопотреблением 231
5.2.2. Встроенный низкочастотный генератор со сверхнизким потреблением 233
5.2.3. Генератор LFXT1 233
5.2.4. Генератор XT2 234
5.2.5. Генератор с цифровым управлением (DCO) 235
5.2.6. Модулятор DCO 236
5.2.7. Отказоустойчивая работа модуля синхронизации 237
5.2.8. Синхронизация тактовых сигналов 239
5.3. Регистры модуля синхронизации 239

Глава 6. Контроллер DMA 244

6.1. Введение 244
6.2. Функционирование контроллера DMA 246
6.2.1. Режимы адресации контроллера DMA 246
6.2.2. Режимы пересылки контроллера DMA 247
6.2.3. Инициация передачи данных с использованием DMA 250
6.2.4. Прерывание DMA-пересылок 254
6.2.5. Приоритеты каналов DMA 254
6.2.6. Длительность DMA-пересылки 254
6.2.7. Функционирование DMA и прерывания 255
6.2.8. Прерывания контроллера DMA 255
6.2.9. Использование модуля USCI_B в режиме I2C с контроллером DMA 257
6.2.10. Использование модуля ADC12 с контроллером DMA 257
6.2.11. Использование модуля DAC12 с контроллером DMA 257
6.2.12. Запись в флэш-память с использованием контроллера DMA 258
6.3. Регистры контроллера DMA 258

Глава 7. Контроллер флэш-памяти 265

7.1. Введение 265
7.2. Сегментная организация флэш-памяти 265
7.2.1. Сегмент A 267
7.3. Функционирование флэш-памяти 267
7.3.1. Тактовый генератор контроллера флэш-памяти 268
7.3.2. Стирание флэш-памяти 269
7.3.3. Запись в флэш-память 271
7.3.4. Обращение к флэш-памяти во время записи или стирания 277
7.3.5. Останов циклов записи или стирания 278
7.3.6. Режим чтения при граничных условиях 278
7.3.7. Конфигурирование контроллера флэш-памяти и организация доступа к нему 279
7.3.8. Прерывания контроллера флэш-памяти 279
7.3.9. Программирование флэш-памяти 279
7.4. Регистры контроллера флэш-памяти 281

Глава 8. Цифровые порты ввода/вывода 285

8.1. Введение 285
8.2. Функционирование цифровых портов ввода/вывода 285
8.2.1. Регистр данных входа PxIN 285
8.2.2. Регистр данных выхода PxOUT 286
8.2.3. Регистр направления PxDIR 286
8.2.4. Регистр включения подтягивающих резисторов PxREN 286
8.2.5. Регистры выбора функции PxSEL и PxSEL2 286
8.2.6. Прерывания от портов P1 и P2 287
8.2.7. Конфигурация неиспользуемых выводов портов 288
8.3. Регистры цифровых портов ввода/вывода 288

Глава 9. Супервизор напряжения питания 290

9.1. Введение 290
9.2. Функционирование супервизора 290
9.2.1. Конфигурирование супервизора 290
9.2.2. Функционирование компаратора супервизора 291
9.2.3. Изменение битов VLDx 292
9.2.4. Рабочий диапазон супервизора 292
9.3. Регистры супервизора 293

Глава 10. Сторожевой таймер 295

10.1. Введение 295
10.2. Функционирование сторожевого таймера 296
10.2.1. Счётчик сторожевого таймера 297
10.2.2. Режим сторожевого таймера 297
10.2.3. Режим интервального таймера 297
10.2.4. Прерывания сторожевого таймера 298
10.2.5. Отказоустойчивое тактирование сторожевого таймера 298
10.2.6. Функционирование в режимах пониженного энергопотребления 299
10.2.7. Примеры кода 299
10.3. Регистры сторожевого таймера 299

Глава 11. Аппаратный умножитель 302

11.1. Введение 302
11.2. Функционирование аппаратного умножителя 302
11.2.1. Регистры операндов 303
11.2.2. Регистры результата 304
11.2.3. Примеры кода 305
11.2.4. Косвенная адресация RESLO 305
11.2.5. Использование прерываний 306
11.3. Регистры аппаратного умножителя 306

Глава 12. Таймер A 307

12.1. Введение 307
12.2. Функционирование Таймера A 307

12.2.1. 16-битный таймер/счётчик . . . 308
12.2.2. Запуск таймера . . . 309
12.2.3. Управление режимом работы таймера . . . 309
12.2.4. Блоки захвата/сравнения . . . 314
12.2.5. Модуль вывода . . . 316
12.2.6. Прерывания Таймера А . . . 319
12.3. Регистры Таймера А . . . 321

Глава 13. Таймер В . . . 326

13.1. Введение . . . 326
13.1.1. Сходства и различия с Таймером А . . . 326
13.2. Функционирование Таймера В . . . 328
13.2.1. 16-битный таймер/счётчик . . . 328
13.2.2. Запуск таймера . . . 329
13.2.3. Управление режимом работы таймера . . . 329
13.2.4. Блоки захвата/сравнения . . . 334
13.2.5. Модуль вывода . . . 337
13.2.6. Прерывания Таймера В . . . 340
13.3. Регистры Таймера В . . . 343

Глава 14. Универсальный последовательный интерфейс . . . 348

14.1. Введение . . . 348
14.2. Функционирование модуля USI . . . 351
14.2.1. Инициализация модуля USI . . . 351
14.2.2. Генерация тактового сигнала USI . . . 351
14.2.3. Режим SPI . . . 352
14.2.4. Режим I^2C . . . 354
14.3. Регистры модуля USI . . . 358

Глава 15. Универсальный последовательный коммуникационный интерфейс: режим UART . . . 362

15.1. Введение . . . 362
15.2. Введение в модуль USCI: режим UART . . . 362
15.3. Функционирование модуля USCI: режим UART . . . 363
15.3.1. Инициализация и сброс модуля USCI . . . 363
15.3.2. Формат символа . . . 365
15.3.3. Форматы асинхронного обмена . . . 365
15.3.4. Автоматическое определение скорости передачи . . . 368
15.3.5. Кодирование и декодирование сигналов IrDA . . . 370
15.3.6. Автоматическое обнаружение ошибок . . . 371
15.3.7. Разрешение приёма USCI . . . 372
15.3.8. Разрешение передачи USCI . . . 373
15.3.9. Контроллер скорости передачи UART . . . 374
15.3.10. Установка скорости обмена . . . 376
15.3.11. Синхронизация при передаче . . . 377
15.3.12. Синхронизация при приёме . . . 378

15.3.13. Типовые скорости обмена и величины ошибок 379
15.3.14. Использование модуля USCI в режиме UART совместно с режимами пониженного энергопотребления 382
15.3.15. Прерывания модуля USCI 382
15.4. Регистры модуля USCI: режим UART 383

Глава 16. Универсальный последовательный коммуникационный интерфейс: режим SPI 392

16.1. Введение 392
16.2. Введение в модуль USCI: режим SPI 392
16.3. Функционирование модуля USCI: режим SPI 394
16.3.1. Инициализация и сброс модуля USCI 394
16.3.2. Формат символа 395
16.3.3. Режим ведущего 395
16.3.4. Режим ведомого 396
16.3.5. Разрешение обмена по интерфейсу SPI 397
16.3.6. Управление тактовым сигналом 397
16.3.7. Использование режима SPI совместно с режимами пониженного энергопотребления 398
16.3.8. Прерывания в режиме SPI 399
16.4. Регистры модуля USCI: режим SPI 401

Глава 17. Универсальный последовательный коммуникационный интерфейс: режим I^2C 408

17.1. Введение 408
17.2. Введение в модуль USCI: режим I^2C 408
17.3. Функционирование модуля USCI: режим I^2C 409
17.3.1. Инициализация и сброс модуля USCI 409
17.3.2. Передача данных по шине I^2C 411
17.3.3. Режимы адресации I^2C 412
17.3.4. Режимы работы модуля I^2C 413
17.3.5. Генерация и синхронизация тактового сигнала I^2C 421
17.3.6. Использование модуля USCI в режиме I^2C совместно с режимами пониженного энергопотребления 425
17.3.7. Прерывания в режиме I^2C 425
17.4. Регистры модуля USCI: режим I^2C 427

Глава 18. Модуль операционного усилителя OA 435

18.1. Введение 435
18.2. Функционирование модуля OA 435
18.2.1. Операционный усилитель 437
18.2.2. Входы модуля OA 437
18.2.3. Выход модуля OA и организация обратной связи 437
18.2.4. Конфигурация модуля OA 438
18.3. Регистры модулей OA 444

Глава 19. Модуль аналогового компаратора Comparator_A+ 447

19.1. Введение 447
19.2. Функционирование модуля Comparator_A+ 447
19.2.1. Компаратор 447
19.2.2. Входные аналоговые ключи 448
19.2.3. Ключ замыкания входов 449
19.2.4. Выходной фильтр 450
19.2.5. Генератор опорного напряжения 450
19.2.6. Компаратор и регистр отключения порта CAPD 451
19.2.7. Прерывания компаратора 451
19.2.8. Использование компаратора для измерения сопротивления 452
19.3. Регистры модуля Comparator_A+ 454

Глава 20. Модуль 10-битного АЦП ADC10 456

20.1. Введение 456
20.2. Функционирование модуля ADC10 458
20.2.1. Ядро 10-битного АЦП 458
20.2.2. Входы модуля ADC10 и мультиплексор 459
20.2.3. Генератор опорного напряжения 459
20.2.4. Автоматическое отключение 460
20.2.5. Синхронизация выборки и преобразования 461
20.2.6. Режимы преобразования 462
20.2.7. Контроллер передачи данных модуля ADC10 467
20.2.8. Использование встроенного датчика температуры 472
20.2.9. Заземление и борьба с помехами при использовании модуля ADC10 473
20.2.10. Прерывания модуля ADC10 474
20.3. Регистры модуля ADC10 475

Глава 21. Модуль 12-битного АЦП ADC12 481

21.1. Введение 481
21.2. Функционирование модуля ADC12 482
21.2.1. Ядро 12-битного АЦП 482
21.2.2. Входы модуля ADC12 и мультиплексор 484
21.2.3. Генератор опорного напряжения 485
21.2.4. Синхронизация выборки и преобразования 485
21.2.5. Сохранение результатов преобразования 488
21.2.6. Режимы преобразования 488
21.2.7. Использование встроенного датчика температуры 493
21.2.8. Заземление и борьба с помехами при использовании модуля ADC12 494
21.2.9. Прерывания модуля ADC12 495
21.3. Регистры модуля ADC12 497

Глава 22. Структура TLV 504

22.1. Введение 504
22.2. Поддерживаемые теги 504
22.2.1. Структура TLV калибровочных значений DCO 506

22.2.2. Структура TLV калибровочных значений модуля ADC12 506
22.3. Проверка целостности содержимого сегмента A 509
22.4. Анализ содержимого сегмента A 510

Глава 23. Модуль 12-битного ЦАП DAC12 511

23.1. Введение 511
23.2. Функционирование модуля ADC12 513
23.2.1. Ядро 12-битного ЦАП 513
23.2.2. Опорное напряжение модуля DAC12 514
23.2.3. Обновление состояния выхода модуля ADC12 514
23.2.4. Формат содержимого DAC12_xDAT 514
23.2.5. Калибровка смещения выходного усилителя модуля DAC12 515
23.2.6. Группирование нескольких модулей DAC12 517
23.2.7. Прерывания модуля DAC12 518
23.3. Регистры модуля DAC12 518

Глава 24. Модуль 16-битного АЦП SD16_A 521

24.1. Введение 521
24.2. Функционирование модуля SD16_A 523
24.2.1. Ядро АЦП 523
24.2.2. Диапазон входного аналогового сигнала и усилитель с программируемым коэффициентом усиления (PGA) 523
24.2.3. Генератор опорного напряжения 523
24.2.4. Автоматическое отключение 524
24.2.5. Выбор входного канала 524
24.2.6. Параметры аналогового входа 525
24.2.7. Цифровой фильтр 526
24.2.8. Регистр данных SD16MEM0 530
24.2.9. Режимы преобразования 531
24.2.10. Использование встроенного датчика температуры 532
24.2.11. Обработка прерываний 533
24.3. Регистры модуля SD16_A 534

Глава 25. Встроенный модуль эмуляции EEM 539

25.1. Введение 539
25.2. Функциональные узлы модуля EEM 541
25.2.1. Триггеры 541
25.2.2. Секвенсор триггеров 541
25.2.3. Внутренний буфер трассировки 542
25.2.4. Управление тактовыми сигналами 542
25.3. Конфигурации модуля EEM 542

ПРЕДИСЛОВИЕ

Об этой книге

В настоящей книге рассматриваются модули и периферийные устройства микроконтроллеров семейства MSP430x2xx. В каждой главе приводится обобщённый обзор отдельного модуля или периферийного устройства. Не все модели семейства обладают полным набором функций и возможностей того или иного модуля или периферийного устройства. Кроме того, в различных семействах микроконтроллеров одни и те же модули и периферийные устройства могут быть реализованы по-разному. Некоторые модули в отдельных моделях или во всём семействе могут быть реализованы не в полном объёме.

Назначение выводов, подключение внутренних сигналов и рабочие параметры отличаются от устройства к устройству. Для получения более точной информации пользователю необходимо изучить справочную документацию на конкретный микроконтроллер.

Дополнительная документация

Дополнительную информацию по рассматриваемой теме можно найти на сайте http://www.ti.com/msp430.

Принятые обозначения

Примеры кода набраны `особым шрифтом`.

Глоссарий

ACLK (Auxiliary CLocK) — вспомогательный тактовый сигнал
ADC (Analog-to-Digital Converter) — аналого-цифровой преобразователь, АЦП
BOR (Brown-Out Reset) — сброс по снижению напряжения питания
BSL (BootStrap Loader) — начальный загрузчик
CPU (Central Processing Unit) — центральный процессор, ЦПУ
DAC (Digital-to-Analog Converter) — цифро-аналоговый преобразователь, ЦАП
DCO (Digitally Controlled Oscillator) — генератор с цифровым управлением
dst (Destination) — получатель
FLL (Frequency Locked Loop) — система автоматической подстройки частоты
GIE (General Interrupt Enable) — общее разрешение прерываний

INT(N/2) (Integer portion of N/2) — целая часть от N/2
I/O (Input/Output) — ввод/вывод
ISR (Interrupt Service Routine) — процедура обработки прерывания
LSB (Least-Significant Bit) — младший значащий бит
LSD (Least-Significant Digit) — младший значащий разряд
LPM (Low-Power Mode) — режим пониженного энергопотребления
MAB (Memory Address Bus) — шина адреса
MCLK (Master CLocK) — основной тактовый сигнал
MDB (Memory Data Bus) — шина данных
MSB (Most-Significant Bit) — старший значащий бит
MSD (Most-Significant Digit) — старший значащий разряд
NMI (Non-Maskable Interrupt) — немаскируемое прерывание
PC (Program Counter) — счётчик команд
POR (Power-On Reset) — сброс по включению питания
PUC (Power-Up Clear) — очистка по включению питания
RAM (Random Access Memory) — оперативное запоминающее устройство, ОЗУ
SCG (System Clock Generator) — генератор системного тактового сигнала
SFR (Special Function Register) — регистр специальных функций
SMCLK (Sub-system Master CLocK) — дополнительный тактовый сигнал
SP (Stack Pointer) — указатель стека
SR (Status Register) — регистр состояния
src (Source) — источник
TOS (Top-of-Stack) — вершина стека
WDT (WatchDog Timer) — сторожевой таймер

Соглашения по обозначению битов регистров

Формат всех регистров приводится с ключом, обозначающим доступность каждого отдельного бита и его исходное состояние.

Доступность бита регистра и его начальное состояние

Ключ	Доступность бита
rw	Чтение/запись
r	Только чтение
r0	Читается как 0
r1	Читается как 1
w	Только запись
w0	Записывается как 0
w1	Записывается как 1
(w)	Бит в регистре не реализован; запись 1 приводит к формированию импульса. Всегда читается как 0
h0	Сбрасывается аппаратно
h1	Устанавливается аппаратно
–0, –1	Состояние после сигнала PUC
–(0), –(1)	Состояние после сигнала POR

ГЛАВА 1

ВВЕДЕНИЕ

В этой главе описывается архитектура MSP430.

1.1. Архитектура

Микроконтроллеры семейства MSP430 имеют фон-неймановскую архитектуру (**Рис. 1.1**) и содержат 16-битное RISC ЦПУ, периферийные модули, а также гибкую систему тактирования, объединённые общими шинами адреса (MAB) и данных (MDB). Сочетание современного ЦПУ и отображаемых в памяти аналоговых и цифровых периферийных модулей делает семейство MSP430 пригодным для работы в приложениях, связанных с обработкой смешанных сигналов.

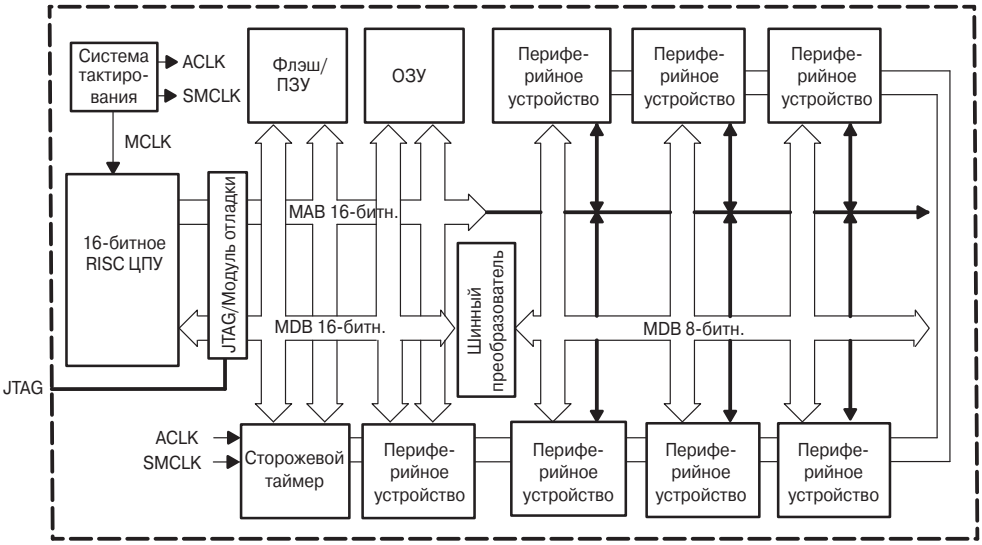


Рис. 1.1. Архитектура MSP430.

Отличительные характеристики микроконтроллеров семейства MSP430x2xx:

- Архитектура со сверхнизким потреблением, позволяющая увеличить время работы при питании от батарей:
  – ток сохранения содержимого ОЗУ — не более 0.1 мкА;
  – ток потребления в режиме часов реального времени — не более 0.8 мкА;
  – ток потребления в активном режиме — 250 мкА/MIPS.
- Высокоэффективная аналоговая подсистема, позволяющая выполнять точные измерения:
  – таймеры, управляемые компаратором, для измерения сопротивления резистивных элементов.
- 16-битное RISC ЦПУ:
  – большой регистровый файл устраняет ограничения рабочего регистра;
  – произведённое по меньшему техпроцессу ядро позволяет снизить потребление и уменьшает стоимость кристалла;
  – оптимизировано для современных языков программирования высокого уровня;
  – набор команд состоит всего из 27 инструкций; поддерживается 7 режимов адресации;
  – векторная система прерываний с расширенными возможностями.
- Флэш-память с возможностью внутрисхемного программирования позволяет гибко изменять программный код (в том числе, во время эксплуатации), а также производить сохранение данных.

1.2. Гибкая система тактирования

Система тактирования разработана специально для применения в устройствах с батарейным питанием. Низкочастотный вспомогательный тактовый сигнал ACLK формируется обычным «часовым» кварцем частотой 32 кГц. Сигнал ACLK может использоваться для периодического «пробуждения» часов реального времени, работающих в фоновом режиме. Встроенный высокочастотный генератор с цифровым управлением (DCO) может формировать основной тактовый сигнал (MCLK), используемый ЦПУ и быстродействующими периферийными модулями. Время выхода на режим этого генератора составляет менее 2 мкс при частоте 1 МГц. Решения на базе микроконтроллеров MSP430 эффективно используют высокопроизводительное 16-битное RISC ЦПУ в течение очень коротких интервалов времени:

- низкочастотный вспомогательный тактовый сигнал используется для реализации режима ожидания со сверхнизким потреблением;
- высокочастотный основной тактовый сигнал используется для эффективной обработки сигналов.

1.3. Внутрисхемная эмуляция

В составе микроконтроллеров имеется специальный модуль внутрисхемной эмуляции, доступ к которому осуществляется по интерфейсу JTAG без использования дополнительных системных ресурсов.

Преимущества внутрисхемной эмуляции:

- создание и отладка кода программы с возможностью его выполнения в реальном времени;
- поддержка точек останова и выполнения программы в пошаговом режиме;
- отлаживаемый объект имеет те же характеристики, что и конечное устройство;
- сохраняется целостность смешанных сигналов благодаря отсутствию взаимного влияния проводов.

1.4. Адресное пространство

Семейство MSP430 имеет фон-неймановскую архитектуру с единым адресным пространством, которое разделено между регистрами специальных функций (SFR), периферийными устройствами, ОЗУ и флэш-памятью в соответствии с **Рис. 1.2**. Подробное распределение памяти для конкретной модели микроконтроллера можно узнать из соответствующей документации. Обращение к исполняемому коду всегда выполняется по чётным адресам. Доступ к данным может осуществляться как побайтно, так и пословно. В настоящее время общий объём адресуемой памяти составляет 128 КБ.

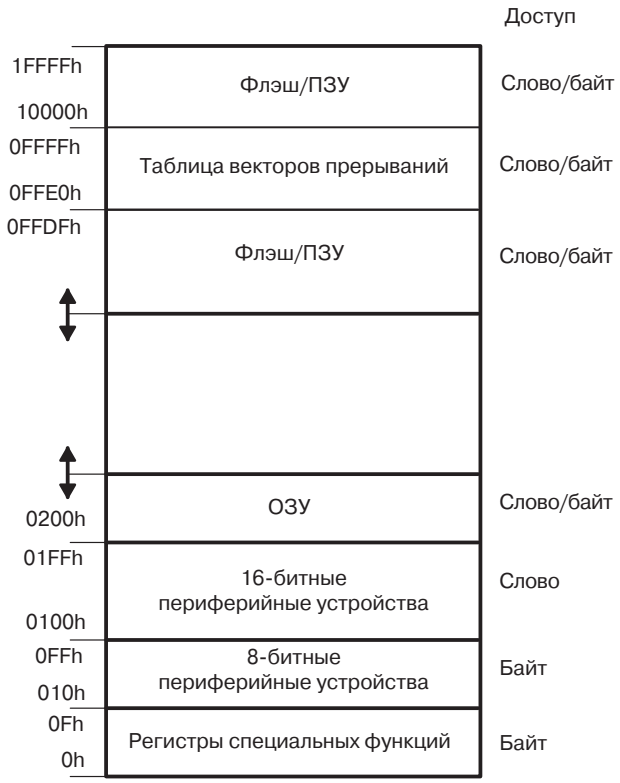


Рис. 1.2. Карта памяти.

1.4.1. Флэш/ПЗУ

Начальный адрес области флэш/ПЗУ зависит от объёма этой памяти и отличается для разных устройств. Конечный адрес области флэш/ПЗУ всегда равен 0x1FFFF. Флэш-память может использоваться как для хранения кода, так и для хранения данных. Двухбайтные и однобайтные данные (или таблицы данных) могут располагаться во флэш-памяти и использоваться непосредственно оттуда, без предварительного копирования в ОЗУ.

Таблица векторов прерываний занимает верхние 16 слов нижней области памяти размером 64 КБ. При этом вектор прерывания с наивысшим приоритетом располагается в последнем слове области по адресу 0x1FFFF.

1.4.2. ОЗУ

Область ОЗУ начинается с адреса 0200h. Конечный адрес области зависит от объёма ОЗУ и меняется от модели к модели. ОЗУ может использоваться как для хранения данных, так и для хранения программного кода.

1.4.3. Периферийные устройства

Регистры периферийных модулей (устройств) располагаются в общем адресном пространстве. Область адресов от 0100h до 01FFh зарезервирована для 16-битных периферийных модулей. Для обращения к таким устройствам необходимо использовать команды, оперирующие двухбайтными значениями. При использовании команд, работающих с однобайтными значениями, допускаются обращения только к чётным адресам памяти, при этом старший байт результата всегда будет равен нулю.

Область адресов от 010h до 0FFh зарезервирована для 8-битных периферийных модулей. Для обращения к этим устройствам необходимо использовать команды, оперирующие байтами. Если для чтения из такого модуля использовать команду, оперирующую словами, то содержимое старшего байта результата будет неопределённым. При записи в 8-битный модуль двухбайтного значения, в регистр устройства будет записан только младший байт.

1.4.4. Регистры специальных функций (SFR)

Некоторые функции периферийных устройств конфигурируются посредством регистров специальных функций. Эти 8-битные регистры располагаются в младших 16 байт адресного пространства. Для обращения к указанным регистрам можно использовать только команды, оперирующие байтами. Назначение отдельных битов регистров специальных функций описано в документации на конкретные модели.

1.4.5. Организация памяти

Однобайтные значения располагаются по чётным или нечётным адресам. Двухбайтные значения располагаются только по чётным адресам, как показано

на **Рис. 1.3**. При использовании команд, оперирующих словами, обращаться можно только к чётным адресам памяти. Младший байт двухбайтного значения всегда располагается по чётному адресу, а старший байт — по следующему нечётному адресу. Например, если слово данных расположено по адресу xxx4h, то младший байт значения находится по адресу xxx4h, а старший байт значения — по адресу xxx5h.

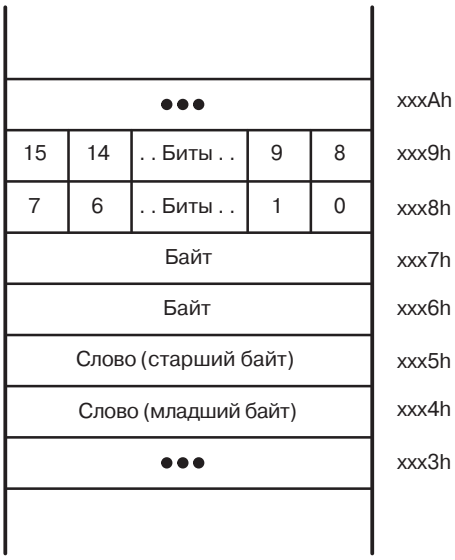


Рис. 1.3. Биты, байты и слова в памяти с побайтовой организацией.

1.5. Расширенные возможности семейства MSP430x2xx

В **Табл. 1.1** перечислены различные усовершенствования, появившиеся в микроконтроллерах семейства MSP430x2xx. Эти усовершенствования описываются в последующих главах книги или же, если улучшение касается параметров устройства, приводятся в документации на конкретные модели.

Таблица 1.1. **Усовершенствования семейства MSP430x2xx**

Подсистема	Усовершенствование
Сброс	Сброс по снижению напряжения питания реализован во всех моделях MSP430x2xx. В регистр IFG1 добавлены флаги PORIFG и RETIFG, показывающие причину сброса. Выборка команды из диапазона адресов 0x0000...0x01FF вызывает сброс устройства
Сторожевой таймер	Во всех моделях MSP430x2xx реализован модуль усовершенствованного сторожевого таймера (WDT+). Гарантируется бесперебойная генерация тактового сигнала для этого таймера

Таблица 1.1. **Усовершенствования семейства MSP430x2xx** *(продолжение)*

Подсистема	Усовершенствование
Базовая система синхронизации	Генератор LFXT1 в режиме LF имеет встроенные нагрузочные конденсаторы с программируемой ёмкостью. Генератор LFXT1 в режиме HF позволяет использовать кварцевые резонаторы частотой до 16 МГц. Генератор LFXT1 в режиме LF имеет схему обнаружения сбоев. Выводы XIN и XOUT в 20- и 28-выводных моделях имеют дополнительные функции. Некоторые модели не поддерживают использование внешнего R_{OSC} с DCO. В этом случае младший бит регистра DCSCTL2 всегда должен быть сброшен. За подробной информацией обращайтесь к документации на конкретный микроконтроллер. Значительно увеличена рабочая частота DCO. Значительно улучшена температурная стабильность DCO
Флэш-память	Информационная секция памяти содержит 4 сегмента по 64 байт. Сегмент A индивидуально блокируется битом LOCKA. Вся информация может быть защищена от группового стирания битом LOCKA. Стирание сегмента может приостанавливаться на время прерывания. Обновление флэш-памяти может быть отменено прерыванием. Напряжение программирования флэш-памяти снижено до 2.2 В. Уменьшена длительность операций записи/стирания. Сбой в работе тактового генератора отменяет обновление флэш-памяти
Цифровые порты ввода/вывода	Все порты имеют встроенные подтягивающие (pullup/pulldown) резисторы. В 20- и 28-выводных моделях добавлены линии ввода/вывода P2.6 и P2.7 в качестве альтернативных функций выводов XIN и XOUT. Если используется кварцевый резонатор, биты P2SELx для этих линий порта всегда должны быть установлены
Компаратор A	Количество входов, подключаемых к компаратору, увеличено за счет использования нового входного мультиплексора
Энергопотребление	Типовой потребляемый ток в режиме LPM3 снижен почти на 50% при напряжении питания 3 В. Значительно уменьшено время выхода DCO на режим
Рабочая частота	Максимальная рабочая частота увеличена до 16 МГц при напряжении питания 3.3 В
Начальный загрузчик	Некорректный пароль приводит к стиранию всей флэш-памяти. Усложнена процедура входа в загрузчик для предотвращения случайного стирания флэш-памяти

ГЛАВА 2

СБРОС, ПРЕРЫВАНИЯ И РЕЖИМЫ РАБОТЫ

В этой главе описываются подсистема сброса, прерывания и режимы работы MSP430x2xx.

2.1. Сброс и инициализация системы

Схема сброса, показанная на **Рис. 2.1**, формирует сигналы сброса (POR) и очистки (PUC) по включению питания. Генерация данных сигналов вызывается различными событиями, при этом исходное состояние устройства зависит от того, какой из сигналов был сгенерирован.

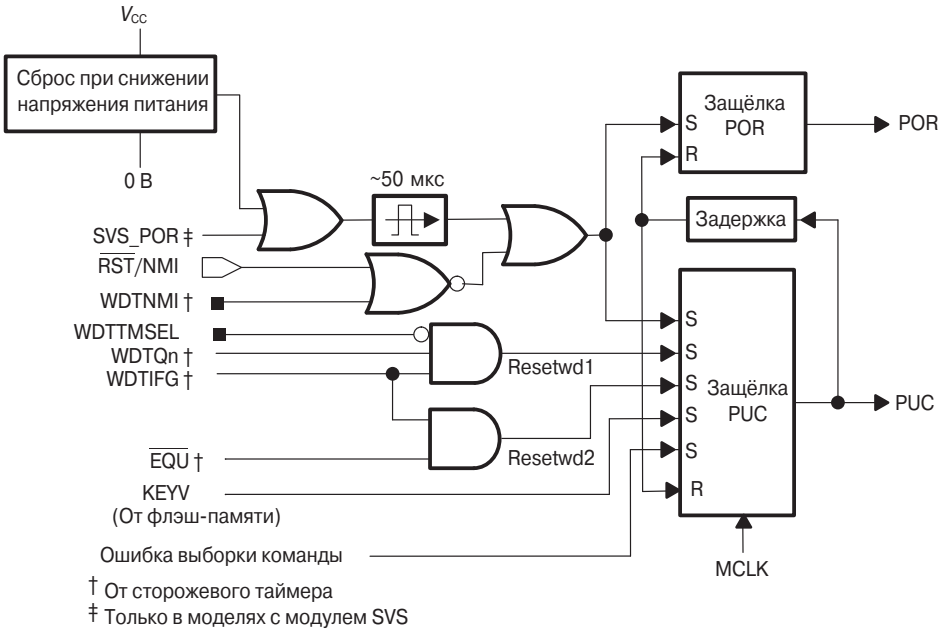


Рис. 2.1. Подсистема сброса.

Сигнал POR является сигналом сброса устройства. Этот сигнал генерируется при наступлении любого из следующих событий:

- включение устройства;
- сигнал НИЗКОГО уровня на выводе $\overline{\text{RST}}$/NMI, если последний сконфигурирован как вход сброса;
- низкий уровень напряжения питания при PORON = 1.

Сигнал PUC всегда генерируется при появлении сигнала POR, однако генерация последнего никак не связана с сигналом PUC. Сигнал PUC генерируется при наступлении любого из следующих событий:

- генерация сигнала POR;
- тайм-аут сторожевого таймера (только в соответствующем режиме);
- обращение к сторожевому таймеру с неверным ключом;
- обращение к контроллеру флэш-памяти с неверным ключом;
- попытка выборки команды из памяти в диапазоне адресов 0h...01FFh.

2.1.1. Сброс по снижению напряжения питания (BOR)

Модуль BOR отслеживает изменения напряжения питания на выводе V_{CC}. Этот модуль сбрасывает устройство, вызывая генерацию сигнала POR при подаче и снятии напряжения питания. Диаграммы работы модуля BOR показаны на **Рис. 2.2**.

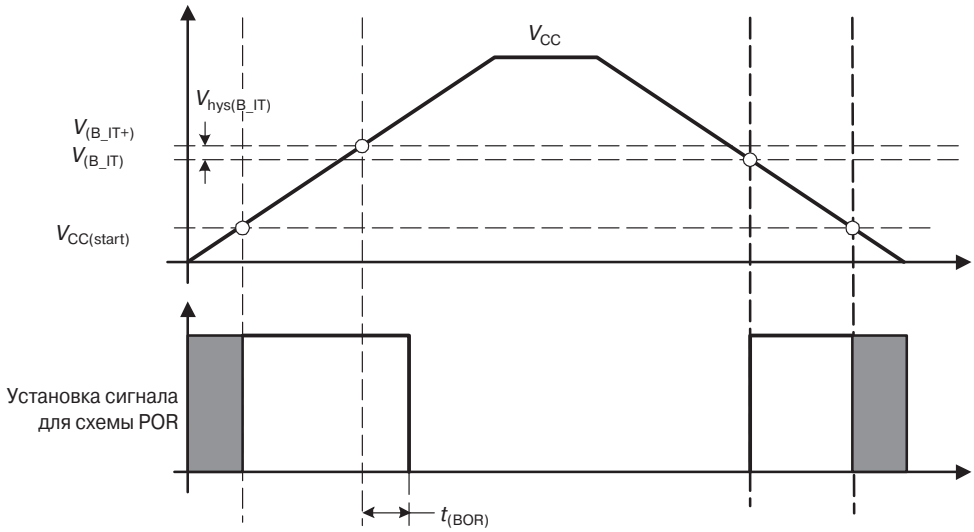


Рис. 2.2. Временные диаграммы работы модуля BOR.

Сигнал POR становится активным при переходе напряжением V_{CC} порогового значения $V_{CC(start)}$. Сигнал сброса удерживается в активном состоянии до тех пор, пока напряжение V_{CC} не превысит значения $V_{(B_IT+)}$ и не истечёт время $t_{(BOR)}$. Из-за наличия гистерезиса $V_{hys(B_IT-)}$ повторная активация сигнала сброса модулем BOR происходит при снижении напряжения питания ниже уровня $V_{(B_IT-)}$.

Поскольку величина $V_{(B_IT-)}$ намного больше уровня срабатывания схемы POR (V_{min}), модуль BOR обеспечивает сброс устройства при таких неисправностях источника питания, при которых его напряжение не опускается ниже уровня V_{min}. Точные значения параметров модуля BOR приводятся в документации на конкретные модели.

2.1.2. Состояние устройства после сброса

После снятия сигнала POR микроконтроллер переходит в следующее состояние:

- вывод $\overline{RST}$/NMI конфигурируется как вход сброса;
- все линии портов ввода/вывода конфигурируются как входы в соответствии с описанием, приведенным в главе 8 «Цифровые порты ввода/вывода»;
- прочие периферийные модули и регистры инициализируются так, как описано в соответствующих главах книги;
- регистр состояния (SR) сбрасывается;
- сторожевой таймер включается в сторожевом режиме;
- в счётчик команд (PC) загружается значение, находящееся по адресу вектора сброса (0FFFEh). Если по этому адресу записано 0FFFFh, то устройство отключается для минимизации потребления.

Инициализация программы

После сброса устройства пользовательская программа должна инициализировать его в соответствии с требованиями конкретного приложения. Процесс инициализации должен включать следующие этапы:

- инициализация указателя стека SP (как правило, указатель устанавливается на вершину ОЗУ);
- инициализация сторожевого таймера в соответствии с требованиями приложения;
- конфигурирование периферийных модулей в соответствии с требованиями приложения.

Дополнительно можно проверить состояние флагов сторожевого таймера, сбоя тактового генератора и флэш-памяти для определения причины сброса.

2.2. Прерывания

Приоритеты прерываний фиксированы и зависят от местонахождения конкретного модуля в цепочке, как показано на **Рис. 2.3**. Чем ближе расположен модуль к ЦПУ, тем выше приоритет его прерывания. Приоритеты определяют порядок обработки прерываний при одновременной генерации нескольких запросов.

В MSP430 имеется три типа прерываний:

- сброс системы;
- немаскируемые (NMI);
- маскируемые.

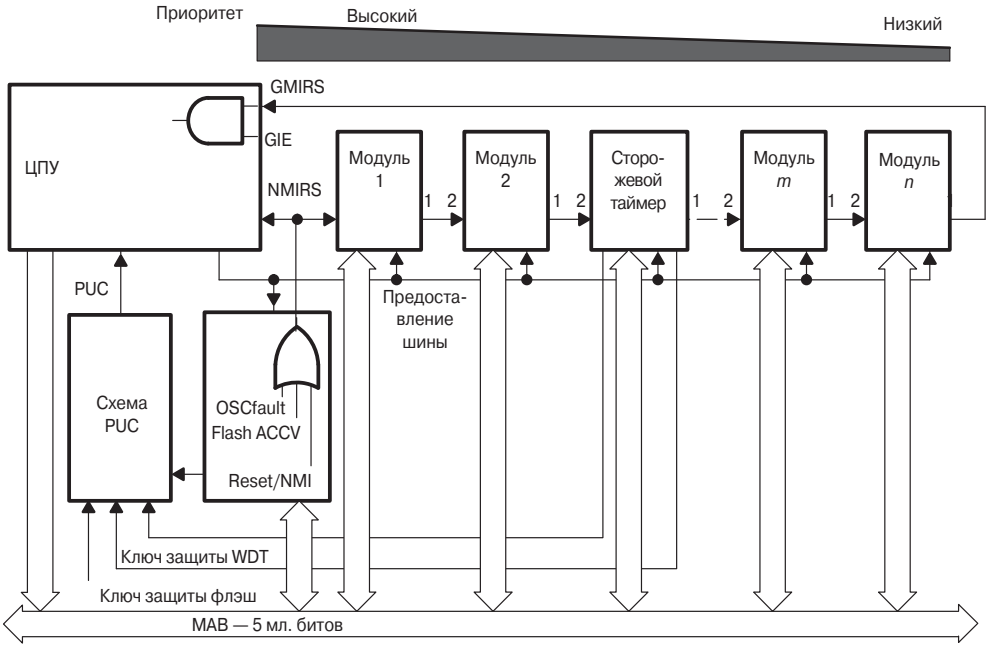


Рис. 2.3. Приоритеты прерываний.

2.2.1. Немаскируемые прерывания (NMI)

Немаскируемые прерывания NMI не маскируются битом общего разрешения прерываний (GIE), однако могут быть по отдельности разрешены/запрещены при помощи индивидуальных битов разрешения прерывания (NMIE, ACCVIE, OFIE). При возникновении немаскируемого прерывания все биты разрешения этого прерывания автоматически сбрасываются. Выполнение программы продолжается с адреса, содержащегося в векторе немаскируемого прерывания, 0FFFCh. Для повторного разрешения прерывания пользовательская программа должна снова установить требуемые биты. Блок-схема источников немаскируемого прерывания приведена на **Рис. 2.4**.

Немаскируемое прерывание может быть вызвано тремя событиями:

- появление активного фронта на выводе $\overline{\text{RST}}$/NMI при конфигурации последнего в режиме NMI;
- возникновение неисправности тактового генератора;
- нарушение доступа к флэш-памяти.

Вывод $\overline{\text{RST}}$/NMI

При включении микроконтроллера вывод $\overline{\text{RST}}$/NMI конфигурируется как вход аппаратного сброса. Назначение этого вывода задаётся в регистре управления сторожевого таймера WDTCTL. Если вывод $\overline{\text{RST}}$/NMI используется в качестве входа сброса, то ЦПУ будет удерживаться в состоянии сброса до тех пор, пока на этом выводе будет присутствовать сигнал НИЗКОГО уровня. При подаче на

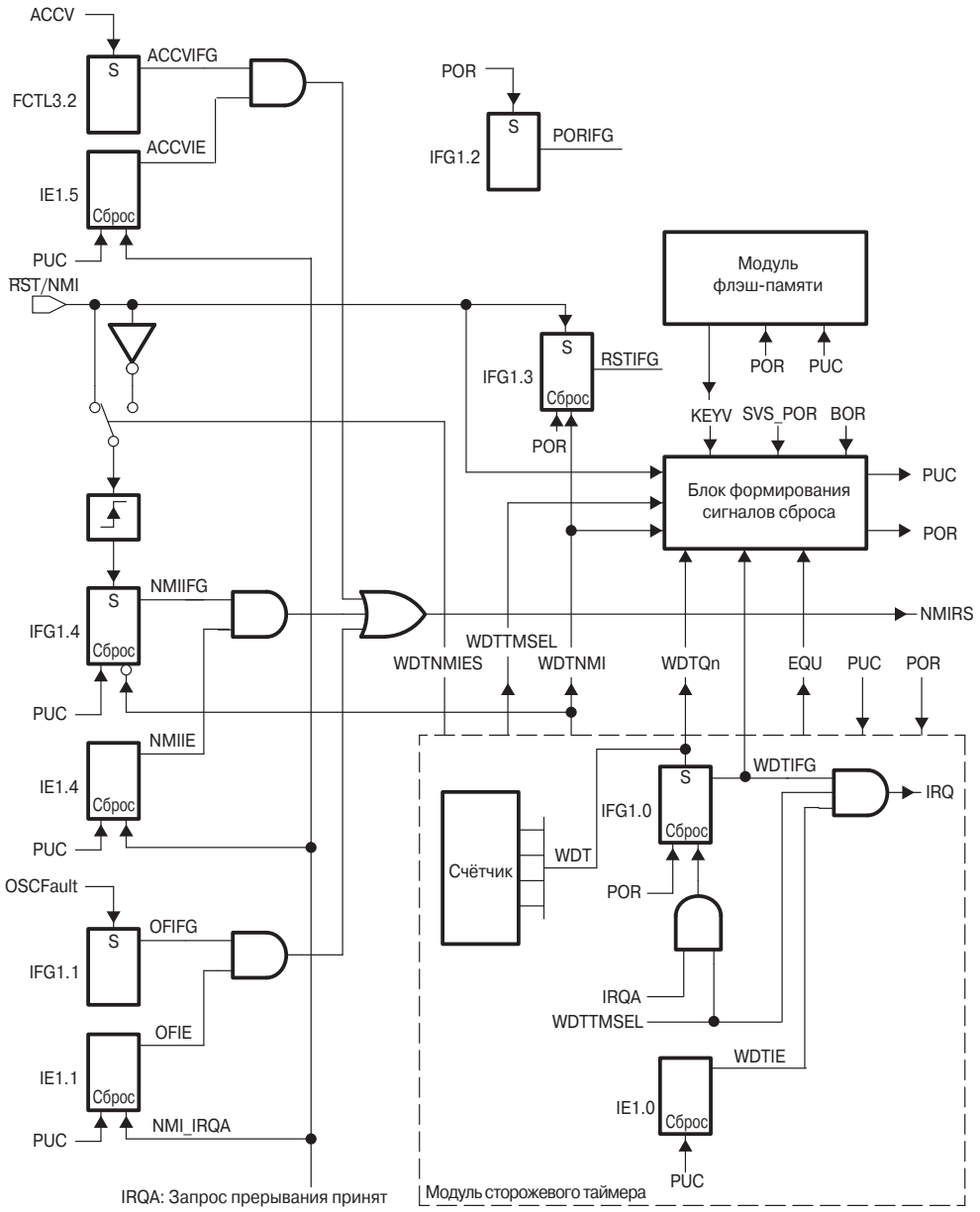


Рис. 2.4. Блок-схема источников немаскируемого прерывания.

вход сигнала ВЫСОКОГО уровня ЦПУ начинает выполнять программу с адреса, хранящегося в векторе сброса (0FFFEh). Одновременно с этим устанавливается флаг RSTIF.

Если вывод $\overline{\text{RST}}$/NMI сконфигурирован в программе как вход немаскируемого прерывания, то появление на этом выводе активного фронта (задаётся битом

WDTNMIES) при установленном бите NMIIE вызывает генерацию немаскируемого прерывания. Также устанавливается флаг NMIIFG.

Примечание 1. Удержание вывода $\overline{\text{RST}}$/NMI в состоянии лог. 0
При использовании вывода в режиме NMI сигнал, вызывающий генерацию немаскируемого прерывания, не должен удерживаться в состоянии лог. 0. Если в то время, пока сигнал NMI имеет НИЗКИЙ уровень, другое событие вызовет появление сигнала PUC, то устройство перейдёт в состояние сброса, поскольку сигнал PUC переключит вывод $\overline{\text{RST}}$/NMI в режим входа сброса.

Примечание 2. Изменение бита WDTNMIES
Если вывод $\overline{\text{RST}}$/NMI находится в режиме NMI, то изменение бита WDTNMIES может привести к генерации немаскируемого прерывания в зависимости от уровня, присутствующего на выводе. Если же бит выбора фронта немаскируемого прерывания изменяется до переключения вывода в режим NMI, то генерации прерывания не произойдёт.

Нарушение доступа к флэш-памяти

При нарушении доступа к флэш-памяти устанавливается флаг ACCVIFG. Генерация немаскируемого прерывания при возникновении такой ситуации разрешается установкой бита ACCVIE. В процедуре обработки немаскируемого прерывания можно проверить флаг ACCVIFG, чтобы определить, было ли прерывание вызвано именно нарушением доступа к флэш-памяти.

Неисправность тактового генератора

Сигнал неисправности генератора позволяет предотвратить ошибки, связанные с неправильным функционированием кварцевого генератора. Генерация немаскируемого прерывания при обнаружении неисправности генератора разрешается установкой бита OFIE. В процедуре обработки немаскируемого прерывания можно проверить флаг OFIFG, чтобы определить, было ли прерывание вызвано именно сбоем в работе генератора.

Сигнал о неисправности генератора может быть вызван появлением сигнала сброса PUC, поскольку последний переводит генератор LFXT1 из режима HF в режим LF. Сигнал PUC также отключает генератор XT2.

Пример процедуры обработки немаскируемого прерывания

Немаскируемое прерывание может генерироваться разными источниками. При возникновении прерывания биты разрешения NMIIE, OFIE и ACCVIE автоматически сбрасываются. Пользовательская процедура обработки немаскируемого прерывания сбрасывает флаги прерывания и повторно разрешает генерацию прерывания от требуемых источников в соответствии с требованиями приложения, как показано на **Рис. 2.5**.

Примечание. Разрешение NMI-прерывания с помощью битов ACCVIE, NMIIE и OFIE
Чтобы предотвратить возникновение вложенных немаскируемых прерываний, биты разрешения ACCVIE, NMIIE и OFIE не должны устанавливаться в процедуре обработки немаскируемого прерывания.

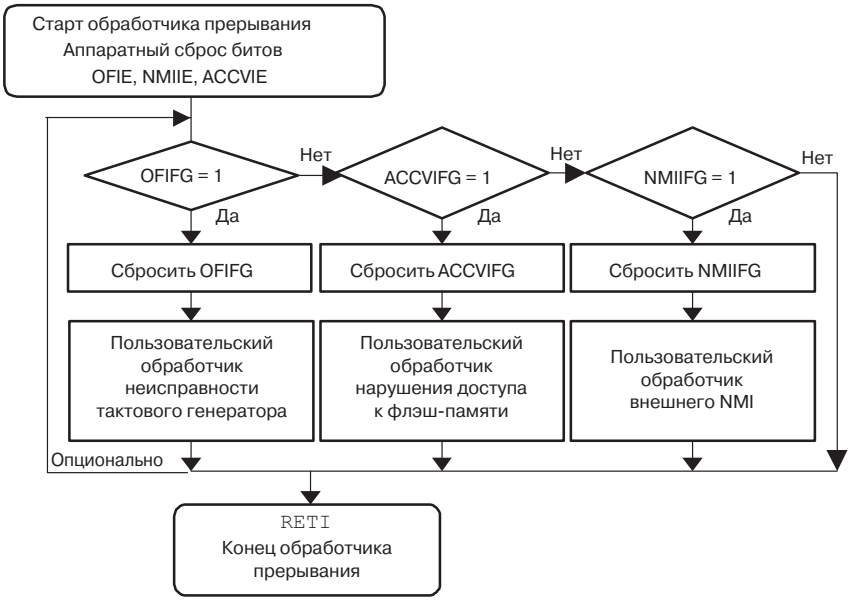


Рис. 2.5. Алгоритм обработки немаскируемого прерывания.

2.2.2. Маскируемые прерывания

Маскируемые прерывания генерируются периферийными устройствами, имеющими такую возможность. В том числе, маскируемое прерывание может генерироваться по переполнению сторожевого таймера при работе последнего в режиме интервального таймера. Прерывания от каждого из источников могут быть запрещены с помощью индивидуальных битов разрешения прерываний. Кроме того, все маскируемые прерывания могут быть запрещены с помощью бита общего разрешения прерываний GIE регистра состояния (SR).

Прерывания от каждого периферийного модуля рассматриваются в соответствующих главах книги.

2.2.3. Обработка прерывания

При возникновении запроса прерывания от периферийного устройства, если установлены бит разрешения прерывания от этого устройства и бит общего разрешения прерываний GIE, вызывается процедура обработки прерывания. Для вызова обработчика немаскируемого прерывания достаточно установленного индивидуального бита разрешения конкретного прерывания.

Принятие запроса прерывания

Задержка обработки прерывания, т.е. время с момента принятия запроса прерывания до начала выполнения первой команды процедуры обработки прерывания (**Рис. 2.6**), составляет 5 (MSP430X) или 6 (MSP430) тактов ЦПУ.

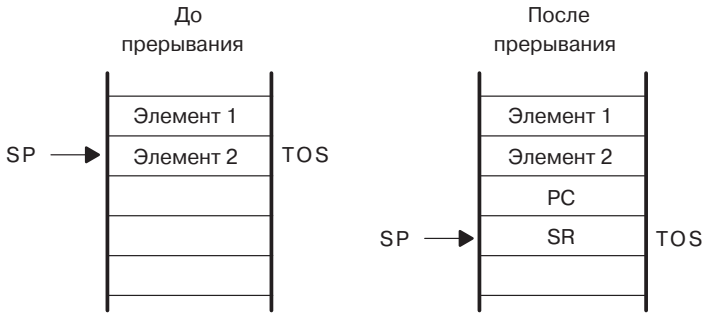


Рис. 2.6. Обработка запроса на прерывание.

Обработка запроса на прерывание производится в следующей последовательности:

1. Ожидается завершение команды, исполняемой в данный момент.
2. Содержимое счётчика команд PC, указывающего на следующую команду, помещается в стек.
3. Содержимое регистра состояния SR помещается в стек.
4. Если за время выполнения последней команды было сформировано несколько запросов на прерывание, то выбирается прерывание с наибольшим приоритетом.
5. Если прерывание имеет один источник, то флаг прерывания автоматически сбрасывается. Если прерывание может генерироваться несколькими источниками, то флаги прерывания остаются установленными для последующей обработки в программе.
6. Регистр состояния SR очищается. В результате процессор переходит из режима пониженного потребления в активный режим. Поскольку бит GIE сбрасывается, последующие прерывания запрещаются.
7. Содержимое вектора прерывания загружается в счётчик команд PC и начинается выполнение процедуры обработки прерывания, расположенной по этому адресу.

Возврат из прерывания

Процедура обработки прерывания всегда завершается командой:

`RETI` (возврат из процедуры обработки прерывания)

Для возврата из прерывания требуется 5 (MSP430) или 3 (MSP430X) такта ЦПУ, необходимых для выполнения следующих действий (**Рис. 2.7**):

1. Восстановление содержимого регистра SR из стека. В результате вступают в действие все предыдущие установки битов GIE, CPUOFF и пр., независимо от их установок, использовавшихся в процедуре обработки прерывания.
2. Содержимое счётчика команд PC извлекается из стека, и выполнение программы продолжается с того места, где она была прервана.

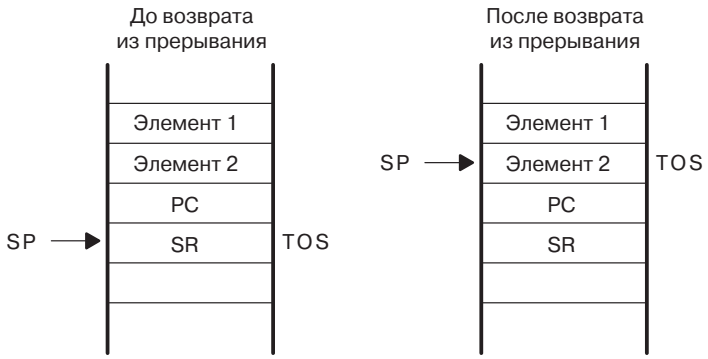


Рис. 2.7. Возврат из прерывания.

Вложенные прерывания

Вложенные прерывания разрешаются установкой бита GIE в процедуре обработки прерывания. При этом любое прерывание, возникшее во время выполнения процедуры обработки прерывания, прервёт её выполнение, независимо от приоритетов обслуживаемого и нового прерываний.

2.2.4. Векторы прерываний

Векторы прерываний и вектор сброса располагаются в диапазоне адресов 0FFFh...0FFC0h, как показано в **Табл. 2.1**. Каждый вектор программируется пользователем посредством записи в него 16-битного адреса соответствующей процедуры обработки прерывания. Полный перечень векторов прерывания приводится в справочной документации на конкретные модели.

Рекомендуется предусматривать процедуры обработки прерываний для всех прерываний, реализованных в конкретном микроконтроллере. При этом все неиспользуемые в программе вектора могут указывать на пустой обработчик прерывания, содержащий единственную команду `RETI`.

Незадействованные вектора прерываний при необходимости можно использовать для размещения программного кода.

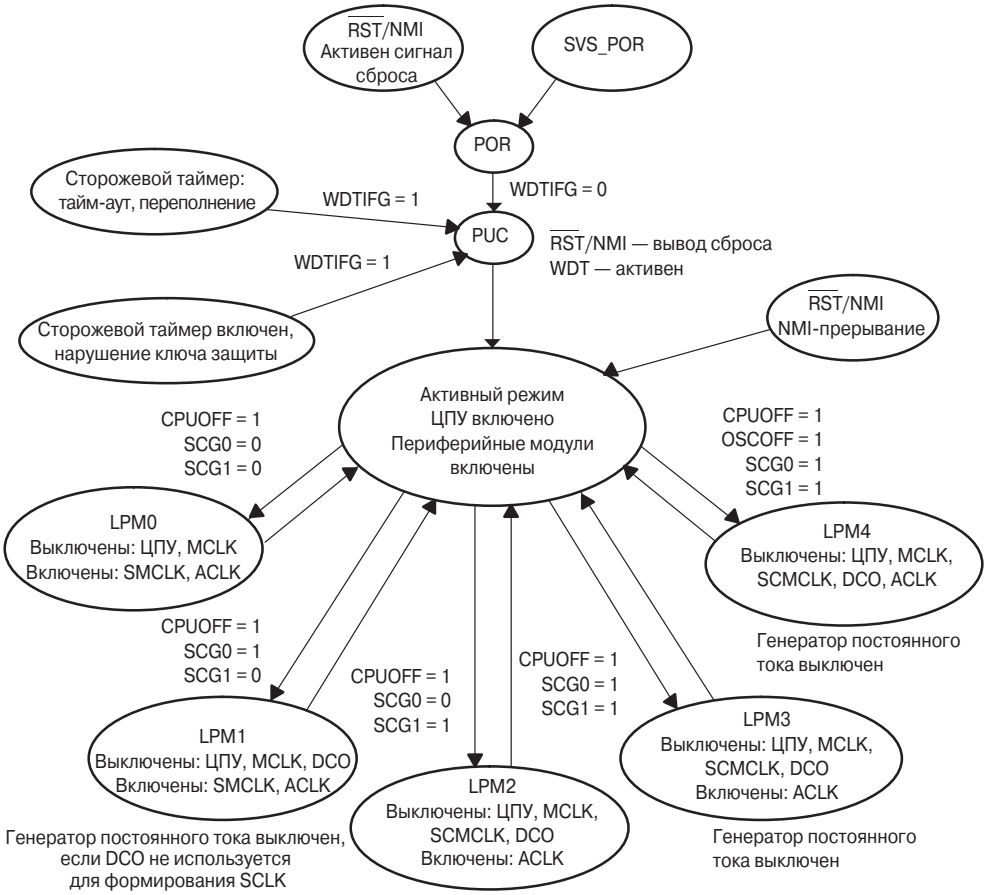
Некоторые биты включения периферийных модулей, биты разрешения прерываний и флаги прерываний находятся в регистрах специальных функций (SFR). Эти регистры являются 8-битными и располагаются в младших адресах адресного пространства. Обращаться к регистрам специальных функций необходимо с помощью команд, работающих с однобайтными операндами. Конфигурация области SFR приводится в справочной документации на конкретные модели.

Таблица 2.1. **Источники, флаги и векторы прерываний**

Источник прерывания	Флаг прерывания	Системное прерывание	Адрес	Приоритет
Включение питания, внешний сброс, сторожевой таймер, пароль флэш-памяти, выборка команды по некорректному адресу	PORIFG RSTIFG WDTIFG KEYV	Сброс	0FFFEh	31, высший
NMI-прерывание, неисправность генератора, нарушение доступа к флэш-памяти	NMIIFG	Немаскируемое	0FFFCh	30
	OFIFG	Немаскируемое		
	ACCVIFG	Немаскируемое		
Определяется устройством			0FFFAh	29
Определяется устройством			0FFF8h	28
Определяется устройством			0FFF6h	27
Сторожевой таймер	WDTIFG	Маскируемое	0FFF4h	26
Определяется устройством			0FFF2h	25
Определяется устройством			0FFF0h	24
Определяется устройством			0FFEEh	23
Определяется устройством			0FFECh	22
Определяется устройством			0FFEAh	21
Определяется устройством			0FFE8h	20
Определяется устройством			0FFE6h	19
Определяется устройством			0FFE4h	18
Определяется устройством			0FFE2h	17
Определяется устройством			0FFE0h	16
Определяется устройством			0FFDEh	15
Определяется устройством			0FFDCh	14
Определяется устройством			0FFDAh	13
Определяется устройством			0FFD8h	12
Определяется устройством			0FFD6h	11
Определяется устройством			0FFD4h	10
Определяется устройством			0FFD2h	9
Определяется устройством			0FFD0h	8
Определяется устройством			0FFCEh	7
Определяется устройством			0FFCCh	6
Определяется устройством			0FFCAh	5
Определяется устройством			0FFC8h	4
Определяется устройством			0FFC6h	3
Определяется устройством			0FFC4h	2
Определяется устройством			0FFC2h	1
Определяется устройством			0FFC0h	0, низший

2.3. Режимы работы

Семейство MSP430 было разработано для приложений со сверхнизким потреблением, поэтому микроконтроллеры семейства имеют различные режимы работы, показанные на **Рис. 2.8**.



SCG1	SCG0	OSCOFF	CPUOFF	Режим	Состояние ЦПУ и тактовых сигналов
0	0	0	0	Активный	ЦПУ активно, все разрешённые тактовые сигналы активны
0	0	0	1	LPM0	ЦПУ, MCLK выключены SMCLK, ACLK активны
0	1	0	1	LPM1	ЦПУ, MCLK выключены. Генератор постоянного тока выключен, если DCO не используется для формирования SMCLK. ACLK активен
1	0	0	1	LPM2	CPU, MCLK, SMCLK, DCO выключены. Генератор постоянного тока не выключается. ACLK активен
1	1	0	1	LPM3	CPU, MCLK, SMCLK, DCO выключены. Генератор постоянного тока выключен. ACLK активен
1	1	1	1	LPM4	ЦПУ и все тактовые сигналы выключены

Рис. 2.8. Режимы работы основной системы тактирования MSP430x2xx.

При реализации этих режимов учитывались следующие требования:

- сверхнизкое потребление;
- скорость и пропускная способность;
- минимизация тока потребления отдельных периферийных модулей.

Типовые токи потребления микроконтроллеров семейства MSP430 в различных режимах работы показаны на **Рис. 2.9**.

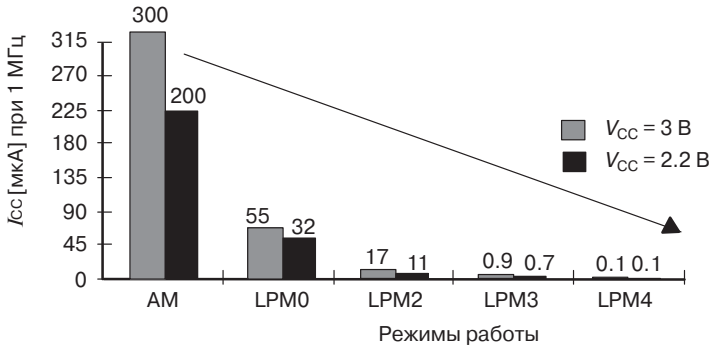


Рис. 2.9. Типовое потребление моделей MSP430F21x1 в различных режимах.

Режимы пониженного энергопотребления 0...4 конфигурируются с помощью битов CPUOFF, OSCOFF, SCG0 и SCG1 регистра состояния SR. Преимущество размещения битов управления режимом работы в регистре состояния состоит в том, что текущий режим работы сохраняется в стеке на время выполнения процедуры обработки прерывания. Если в обработчике сохранённое значение SR не изменялось, то после завершения процедуры обработки прерывания предыдущий режим работы восстанавливается. Выполнение программы может продолжиться и в другом режиме, если процедура обработки прерывания изменит значение регистра состояния, находящееся в стеке. Обращение к битам управления режимом работы и к стеку может производиться с помощью команды любого типа.

При установке любого из битов управления режимом микроконтроллер сразу же переключается в выбранный режим. При отключении любого тактового сигнала также блокируется работа использующих его периферийных устройств. Периферийные устройства могут отключаться и по отдельности с помощью соответствующих регистров управления. Состояние портов ввода/вывода и содержимое ОЗУ и регистров при изменении режима работы остаются неизменными. Выход из режима пониженного потребления возможен по любому разрешённому прерыванию.

2.3.1. Вход в режимы пониженного энергопотребления и выход из них

Выход из любого режима пониженного потребления производится при появлении какого-либо разрешённого прерывания.

При этом выполняются следующие действия:

- Вход в процедуру обработки прерывания:
  - счётчик команд PC и регистр состояния SR сохраняются в стеке;
  - биты CPUOFF, SCG1 и OSCOFF автоматически сбрасываются.

- Варианты возврата из процедуры обработки прерывания:
  – исходное значение регистра SR извлекается из стека, восстанавливая предыдущий режим работы;
  – биты регистра SR, сохранённые в стеке, могут быть изменены в процедуре обработки прерывания. В этом случае при выполнении команды `RETI` микроконтроллер переключится в другой режим работы.

```
; Пример входа в режим LPM0
  BIS   #GIE+CPUOFF,SR           ; Вход в режим LPM0
;  ...                           ; В этом месте программа останавливается
;
; Выход из режима LPM0 в обработчике прерывания
  BIC   #CPUOFF,0(SP)            ; Выход из режима LPM0 по команде RETI
  RETI

; Пример входа в режим LPM3
  BIS   #GIE+CPUOFF+SCG1+SCG0,SR ; Вход в режим LPM3
;  ...                           ; В этом месте программа останавливается
;
; Выход из режима LPM3 в обработчике прерывания
  BIC   #CPUOFF+SCG1+SCG0,0(SR)  ; Выход из режима LPM3 по команде RETI
  RETI
```

2.4. Принципы программирования устройств с низким энергопотреблением

В большинстве случаев наиболее действенным методом снижения энергопотребления является максимизация времени пребывания микроконтроллера в режиме LPM3. Типовое потребление тока в этом режиме составляет менее 2 мкА при включенной схеме часов реального времени и полностью активированной системе прерываний. Для формирования тактового сигнала ACLK используется «часовой» кварц частотой 32 кГц, а ЦПУ тактируется от DCO (выключенного бóльшую часть времени), время выхода на режим которого составляет 6 мкс.

- Используйте прерывания для «пробуждения» процессора и управления ходом выполнения программы.
- Включайте периферийные устройства только по необходимости.
- Используйте вместо программно реализуемых функций встроенные периферийные модули с низким энергопотреблением. Например, Таймер A и Таймер B могут генерировать сигнал с ШИМ и осуществлять захват внешних сигналов автоматически, без использования ресурсов ЦПУ.
- Используйте вычисляемые переходы и быстрые табличные вычисления вместо опроса флагов и длительных программных вычислений.
- Постарайтесь избегать частых вызовов подпрограмм, чтобы снизить накладные расходы.
- В длинных подпрограммах для хранения данных следует использовать регистры ЦПУ, обращение к которым производится за один такт.

2.5. Подключение неиспользуемых выводов

Правильное подключение всех неиспользуемых выводов приведено в **Табл. 2.2**.

Таблица 2.2. **Подключение неиспользуемых выводов**

Вывод	Потенциал	Примечание
AV_{CC}	DV_{CC}	
AV_{SS}	DV_{SS}	
V_{REF+}	Свободный	
Ve_{REF+}	DV_{SS}	
V_{REF-}/Ve_{REF-}	DV_{SS}	
XIN	DV_{CC}	
XOUT	Свободный	
XT2IN	DV_{SS}	
XT2OUT	Свободный	
Px.0...Px.7	Свободный	Работает в качестве линии порта ввода/вывода. Сконфигурирован как выход или как вход с включённой подтяжкой
$\overline{RST}$/NMI	DV_{CC} или V_{CC}	Подтягивающий резистор 47 кОм и конденсатор 10 нФ (2.2 нФ*) на землю
Test	Свободный	Модели 20xx, 21xx и 22xx
TDO	Свободный	
TDI	Свободный	
TMS	Свободный	
TCK	Свободный	

* Ёмкость конденсатора должна быть не более 2.2 нФ при использовании отладочного интерфейса в режиме Spy-By-Wire или в режиме 4-проводного JTAG с такими инструментальными средствами от TI, как эмуляторы FET или программаторы GANG.

ГЛАВА 3

16-БИТНОЕ RISC ЦПУ MSP430

В этой главе описывается ЦПУ MSP430, режимы адресации и набор команд.

3.1. Введение в ЦПУ

ЦПУ MSP430 обладает рядом возможностей, специально предназначенных для поддержки современных методов программирования, таких как вычисляемые переходы, табличные вычисления, а также использование языков высокого уровня, в частности, языка Си. Центральный процессор может адресовать память во всём диапазоне адресов без разбиения её на страницы.

ЦПУ MSP430 имеет следующие особенности:

- RISC-архитектура, поддерживающая 27 команд и 7 режимов адресации;
- ортогональная архитектура — с каждой из команд может использоваться любой режим адресации;
- полная доступность регистров, включая счётчик команд, регистры состояния и указатель стека;
- однотактные регистровые операции;
- большой 16-битный регистровый файл, уменьшающий количество обращений к памяти;
- 16-битная шина адреса, обеспечивающая прямой доступ и ветвление во всём диапазоне адресов;
- 16-битная шина данных, позволяющая напрямую оперировать 2-байтными значениями;
- генератор констант формирует шесть наиболее часто используемых значений, уменьшая размер кода;
- прямой обмен данными между ячейками памяти без промежуточной записи в регистр;
- одно- и двухбайтные адресация и форматы команд.

Блок-схема ЦПУ приведена на **Рис. 3.1**.

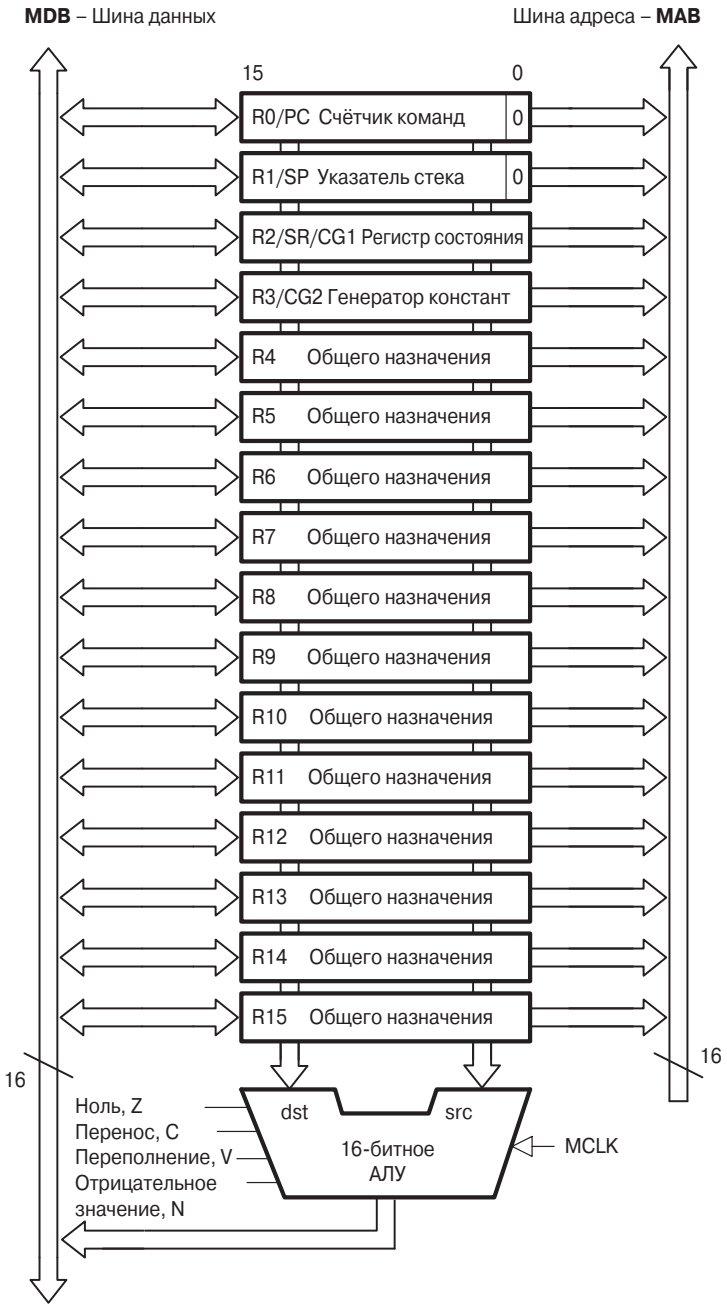


Рис. 3.1. Блок-схема ЦПУ.

3.2. Регистры ЦПУ

Центральный процессор содержит шестнадцать 16-битных регистров. Регистры R0, R1, R2 и R3 имеют специальные функции. Регистры с R4 по R15 являются рабочими регистрами общего назначения.

3.2.1. Счётчик команд (PC)

16-битный счётчик команд (PC/R0) указывает на следующую команду, которая будет выполняться. Каждая команда занимает в памяти чётное число байтов (два, четыре или шесть), и на это же значение инкрементируется счётчик команд. Выборка команд осуществляется пословно, при этом счётчик команд указывает на чётные адреса. Формат счётчика команд PC приведён на **Рис. 3.2**.

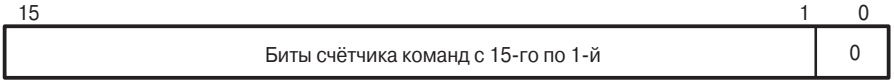


Рис. 3.2. Счётчик команд.

Счётчик команд может быть адресован всеми командами с использованием любого режима адресации:

```
MOV #LABEL,PC     ; Переход к адресу LABEL
MOV LABEL,PC      ; Переход к адресу, содержащемуся в LABEL
MOV @R14,PC       ; Косвенный переход по адресу, содержащемуся в R14
```

3.2.2. Указатель стека (SP)

Указатель стека (SP/R1) используется ЦПУ для сохранения адресов возврата из подпрограмм и прерываний. При этом он изменяется по преддекрементной/постинкрементной схеме. Кроме того, указатель стека может использоваться со всеми командами и любыми режимами адресации. Формат указателя стека SP приведён на **Рис. 3.3**. Указатель стека инициализируется (устанавливается на заданную ячейку ОЗУ) пользователем и выравнивается по чётным адресам.

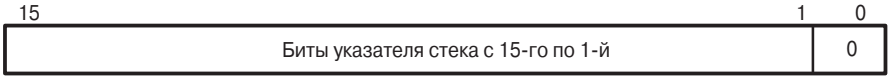


Рис. 3.3. Указатель стека.

Использование стека проиллюстрировано на **Рис. 3.4**.

```
MOV    2(SP),R6      ; Элемент стека I2 -> R6
MOV    R7,0(SP)      ; Перезаписать значение на вершине стека (TOS)
                     ; содержимым R7
PUSH   #0123h        ; Поместить число 0123h на вершину стека (TOS)
POP    R8            ; R8 = 0123h
```

К особым случаям относится использование указателя стека SP в качестве аргумента команд `PUSH` и `POP`. Эти ситуации показаны и объяснены на **Рис. 3.5**.

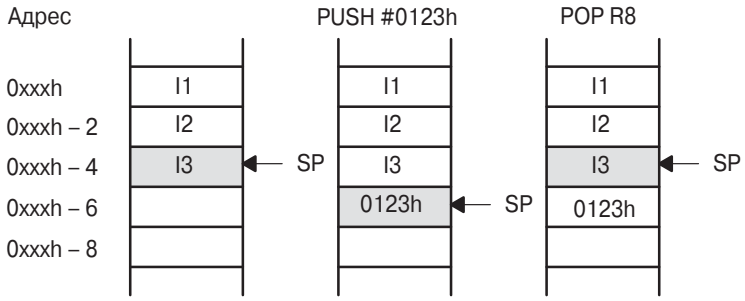


Рис. 3.4. Использование стека.

Указатель стека изменяется после выполнения команды `PUSH SP`

Указатель стека не изменяется после выполнения команды `POP SP`. Эта команда помещает SP_1 в указатель стека ($SP_2 = SP_1$).

Рис. 3.5. Выполнение последовательности команд `PUSH SP` — `POP SP`.

3.2.3. Регистр состояния (SR)

Использовать регистр состояния (SR/R2) в качестве регистра-источника или регистра-приёмника могут только команды, оперирующие двухбайтными значениями и только при использовании режима регистровой адресации. Прочие комбинации режимов адресации используются для поддержки генератора констант. Формат регистра состояния SR приведён на **Рис. 3.6**.

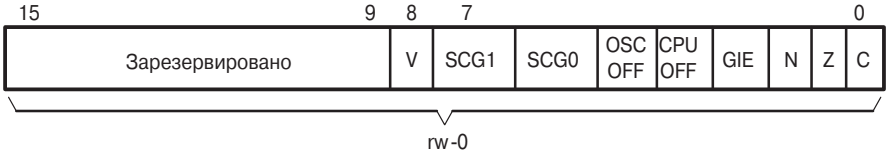


Рис. 3.6. Формат регистра состояния.

Назначение битов регистра SR описано в **Табл. 3.1**.

Таблица 3.1. **Биты регистра состояния**

Бит	Описание	
V	Флаг переполнения. Этот бит устанавливается, если результат арифметической операции выходит за границы допустимых значений для знаковых величин	
	`ADD(.B)`, `ADDC(.B)`	Устанавливается, когда: Положительное + Положительное = Отрицательное, Отрицательное + Отрицательное = Положительное, в противном случае сбрасывается
	`SUB(.B)`, `SUBC(.B)`, `CMP(.B)`	Устанавливается, когда: Положительное – Отрицательное = Отрицательное, Отрицательное – Положительное = Положительное, в противном случае сбрасывается
SCG1	Системный тактовый генератор 1. Когда этот бит установлен, сигнал SMCLK отключён	
SCG0	Системный тактовый генератор 0. Когда этот бит установлен, генератор DCO выключен, если сигнал DCOCLK не используется для формирования сигналов MCLK или SMCLK	
OSCOFF	Выключение генератора. Когда этот бит установлен, кварцевый генератор LFXT1 выключен, если сигнал LFXT1CLK не используется для формирования сигналов MCLK или SMCLK	
CPUOFF	Выключение ЦПУ. Когда этот бит установлен, ЦПУ выключено	
GIE	Общий бит разрешения прерываний. Когда этот бит установлен, маскируемые прерывания разрешены. После сброса все маскируемые прерывания запрещены	
N	Флаг отрицательного результата. Этот бит устанавливается, если результат арифметической операции отрицателен, и сбрасывается, если результат положителен	
	Операции со словами	Бит N равен значению 15-го бита результата
	Операции с байтами	Бит N равен значению 7-го бита результата
Z	Флаг нуля. Этот бит устанавливается, если результат арифметической операции равен нулю, и сбрасывается в противном случае	
C	Флаг переноса. Этот бит устанавливается, если при выполнении арифметической операции возникает перенос, и сбрасывается, если переноса не возникает	

3.2.4. Регистры генератора констант CG1 и CG2

С помощью регистров генератора констант R2 и R3 генерируются шесть часто используемых значений. Выбор конкретной константы осуществляется изменением режима адресации регистра-источника (As) в соответствии с **Табл. 3.2**.

Таблица 3.2. **Значения, формируемые генератором констант**

Регистр	As	Константа	Примечания
R2	00	— — — — —	Регистровый режим адресации
R2	01	(0)	Абсолютный режим адресации
R2	10	00004h	+4, операции над битами
R2	11	00008h	+8, операции над битами
R3	00	00000h	0, операции над словами
R3	01	00001h	+1
R3	10	00002h	+2, операции над битами
R3	11	0FFFFh	–1, операции над словами

Преимущества генератора констант:

- не требуются особые команды;
- не требуется дополнительное слово памяти программ для хранения константы;
- не требуется обращаться к памяти программ для загрузки константы.

Ассемблер автоматически использует генератор констант, если любая из шести указанных констант служит в качестве непосредственного операнда-источника. Если регистры R2 и R3 используются в режиме генерации констант, их нельзя адресовать явно — они выступают только в качестве источников.

Генератор констант — расширение набора команд

Система команд MSP430 содержит всего 27 инструкций. Однако наличие генератора констант позволяет ассемблеру MSP430 использовать 24 дополнительных (эмулируемых) команд. К примеру, однооперандная команда

```
CLR         dst
```

эмулируется двухоперандной командой, занимающей в памяти столько же места:

```
MOV         R3,dst
```

в которой ассемблер заменил константу `#0` обращением к регистру R3 с As = 00.

Команда

```
INC         dst
```

заменяется командой

```
ADD         0(R3),dst
```

3.2.5. Регистры общего назначения R4…R15

Двенадцать регистров с R4 по R15 являются регистрами общего назначения. Любой из этих регистров может использоваться в качестве регистра данных, указателя или индексного значения. Все эти регистры доступны для команд, работающих как с однобайтными, так и с двухбайтными операндами (**Рис. 3.7**).

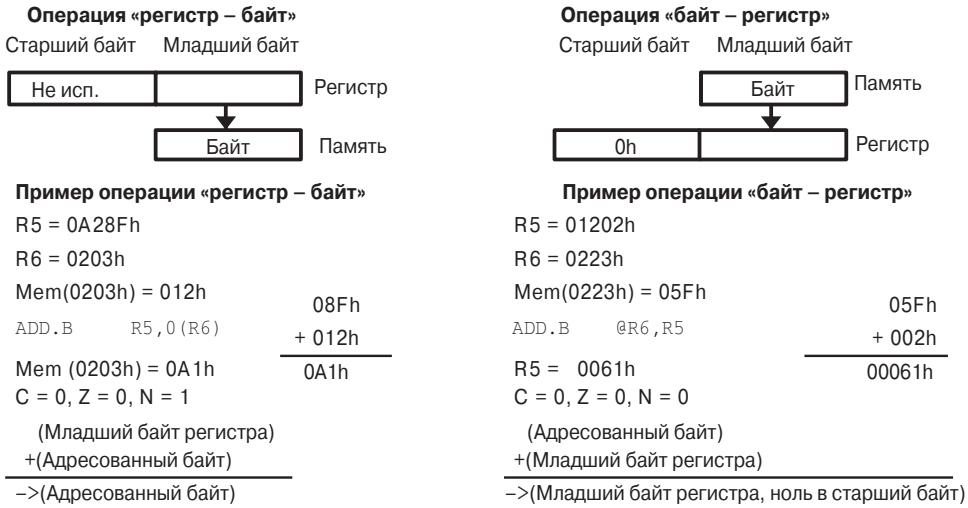


Рис. 3.7. Операции «регистр — байт» и «байт — регистр».

3.3. Режимы адресации

ЦПУ поддерживает семь режимов адресации для операндов-источников и четыре режима — для операндов-приёмников. Эти режимы позволяют адресовать любые ячейки в пределах адресного пространства. В **Табл. 3.3** приводятся значения битов режима As (для операнда-источника) и Ad (для операнда-приёмника), соответствующие тому или иному режиму адресации.

Таблица 3.3. **Режимы адресации операндов**

As/ Ad	Режим адресации	Синтаксис	Описание
00/0	Регистровый	Rn	Содержимое регистра является операндом
01/1	Индексный	X(Rn)	(Rn + X) указывает на операнд. Значение X содержится в следующем слове
01/1	Относительный	ADDR	(PC + X) указывает на операнд. Значение X содержится в следующем слове. Используется индексный режим адресации X(PC)
01/1	Абсолютный	&ADDR	Слово, следующее за командой, содержит абсолютный адрес. Значение X содержится в следующем слове. Используется индексный режим адресации X(SR)
10/-	Косвенный регистровый	@Rn	Rn используется в качестве указателя на операнд
11/-	Косвенный регистровый с автоинкрементом	@Rn+	Rn используется в качестве указателя на операнд. После выполнения операции значение регистра Rn увеличивается на 1 для команд с однобайтными операндами (.B) и на 2 для команд с двухбайтными операндами (.W)
11/-	Непосредственный	#N	Слово, следующее за командой, содержит значение константы N. Используется косвенный режим адресации с автоинкрементом @PC+

Более подробно эти режимы адресации рассматриваются в последующих разделах. В большинстве примеров для источника и приёмника используется один и тот же режим адресации, однако в командах можно использовать любые допустимые комбинации режимов адресации источника и приёмника.

Примечание. Использование меток EDE, TONI, TOM и LEO
В документации на семейство MSP430 аббревиатуры EDE, TONI, TOM и LEO используются в качестве обычных меток. Они не имеют никакого специального значения.

3.3.1. Регистровый режим адресации

Регистровый режим адресации описан в **Табл. 3.4**.

Таблица 3.4. **Описание регистрового режима адресации**

Ассемблерный код	Содержимое ПЗУ
`MOV R10,R11`	`MOV R10,R11`
Длина	Одно или два слова
Операция	Пересылка содержимого регистра R10 в регистр R11. Содержимое R10 не изменяется
Комментарий	Допускается для источника и приёмника
Пример	`MOV R10,R11`

Таблица 3.4. **Описание регистрового режима адресации** *(продолжение)*

	До операции		После операции
R10	0A023h	R10	0A023h
R11	0FA15h	R11	0A023h
PC	PC_{old}	PC	PC_{old}+ 2

Примечание. Данные в регистрах

Обращаться к регистрам можно как с помощью команд, оперирующих байтами, так и с помощью команд, оперирующих словами. При использовании байт-ориентированных команд старший байт результата всегда будет содержать ноль. Биты состояния изменяются в соответствии с результатом операции над байтами.

3.3.2. Индексный режим адресации

Индексный режим адресации описан в **Табл. 3.5**.

Таблица 3.5. **Описание индексного режима адресации**

Ассемблерный код		Содержимое ПЗУ
`MOV 2(R5),6(R6)`		`MOV X(R5),Y(R6)` X = 2, Y = 6
Длина	Два или три слова	
Операция	Пересылка значения с исходного адреса (содержимое R5 + 2) по адресу назначения (содержимое R6 + 6). Регистр-источник (R5) и регистр-приёмник (R6) не изменяются. При использовании индексного режима адресации счётчик команд автоматически инкрементируется таким образом, чтобы выполнение программы продолжилось со следующей команды	
Комментарий	Допускается для источника и приёмника	
Пример	`MOV 2(R5),6(R6)`	

До операции

Адресное пространство		Регистры	
0FF16h	00006h	R5	01080h
0FF14h	00002h	R6	0108Ch
0FF12h	04596h	PC	
01094h	0xxxxh		
01092h	05555h		
01090h	0xxxxh		
01084h	0xxxxh		
01082h	01234h		
01080h	0xxxxh		

0108Ch
+0006h
01092h

01080h
+0002h
01082h

После операции

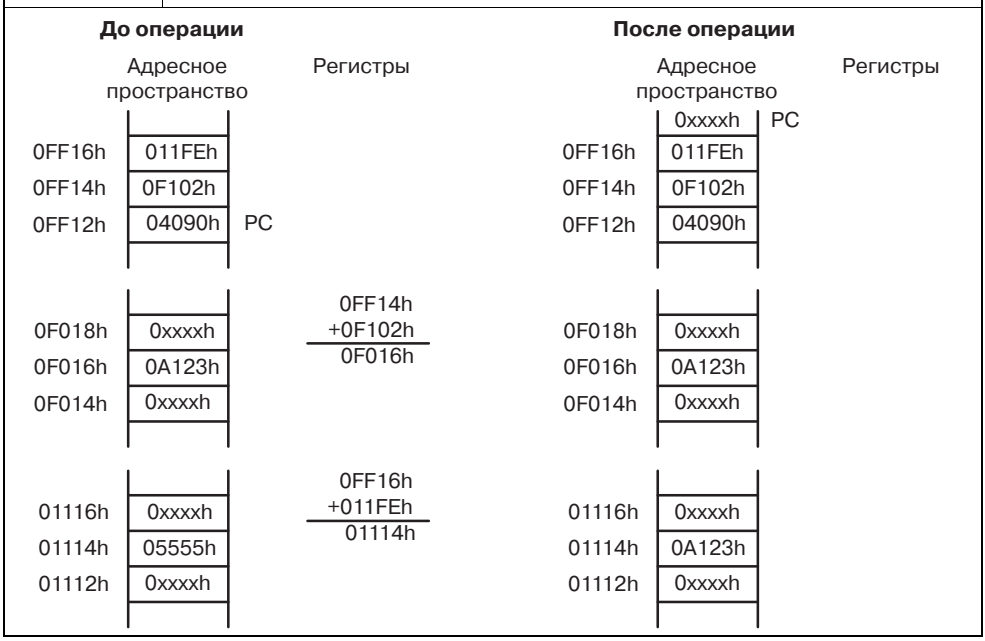
Адресное пространство			Регистры	
	0xxxxh	PC		
0FF16h	00006h		R5	01080h
0FF14h	00002h		R6	0108Ch
0FF12h	04596h			
01094h	0xxxxh			
01092h	01234h			
01090h	0xxxxh			
01084h	0xxxxh			
01082h	01234h			
01080h	0xxxxh			

3.3.3. Относительный режим адресации

Относительный режим адресации описан в **Табл. 3.6**.

Таблица 3.6. **Описание относительного режима адресации**

Ассемблерный код		Содержимое ПЗУ
`MOV EDE,TONI`		`MOV X(PC),Y(PC)` X = EDE - PC Y = TONI - PC
Длина	Два или три слова	
Операция	Пересылка значения с исходного адреса EDE (содержимое PC + X) по адресу назначения TONI (содержимое PC + Y). Слова, расположенные после команды, содержат разности между PC и адресами источника или приёмника. Ассемблер автоматически вычисляет смещения X и Y и вставляет их в код. При использовании относительного режима адресации счётчик команд автоматически инкрементируется таким образом, чтобы выполнение программы продолжилось со следующей команды	
Комментарий	Допускается для источника и приёмника	
Пример	`MOV EDE,TONI ; Адрес источника EDE = 0F016h` `; Адрес приёмника TONI = 01114h`	



3.3.4. Абсолютный режим адресации

Абсолютный режим адресации описан в **Табл. 3.7**.

Таблица 3.7. **Описание абсолютного режима адресации**

Ассемблерный код	Содержимое ПЗУ
`MOV  &EDE,&TONI`	`MOV  X(0),Y(0)` X = EDE Y = TONI

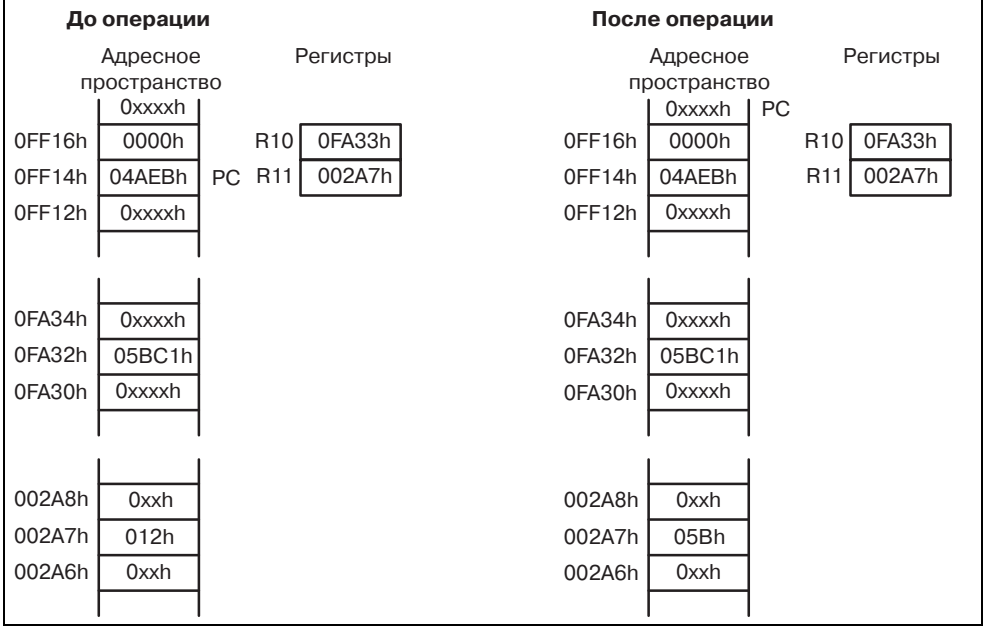
Длина	Два или три слова
Операция	Пересылка значения с исходного адреса EDE по адресу назначения TONI. Слова, расположенные после команды, содержат абсолютные значения адресов. При использовании абсолютного режима адресации счётчик команд автоматически инкрементируется таким образом, чтобы выполнение программы продолжилось со следующей команды
Комментарий	Допускается для источника и приёмника
Пример	`MOV  &EDE,&TONI  ; Адрес источника EDE = 0F016h` `; Адрес приёмника TONI = 01114h`

3.3.5. Косвенный регистровый режим адресации

Косвенный регистровый режим адресации описан в **Табл. 3.8**.

Таблица 3.8. **Описание косвенного регистрового режима адресации**

Ассемблерный код		**Содержимое ПЗУ**
`MOV @R10,0(R11)`		`MOV @R10,0(R11)`
Длина	Одно или два слова	
Операция	Пересылка значения с исходного адреса (содержится в R10) по адресу назначения (содержится в R11). Содержимое регистров не изменяется	
Комментарий	Допускается только для операнда-источника. Для операнда-приёмника используется 0(Rd)	
Пример	`MOV @R10,0(R11)`	

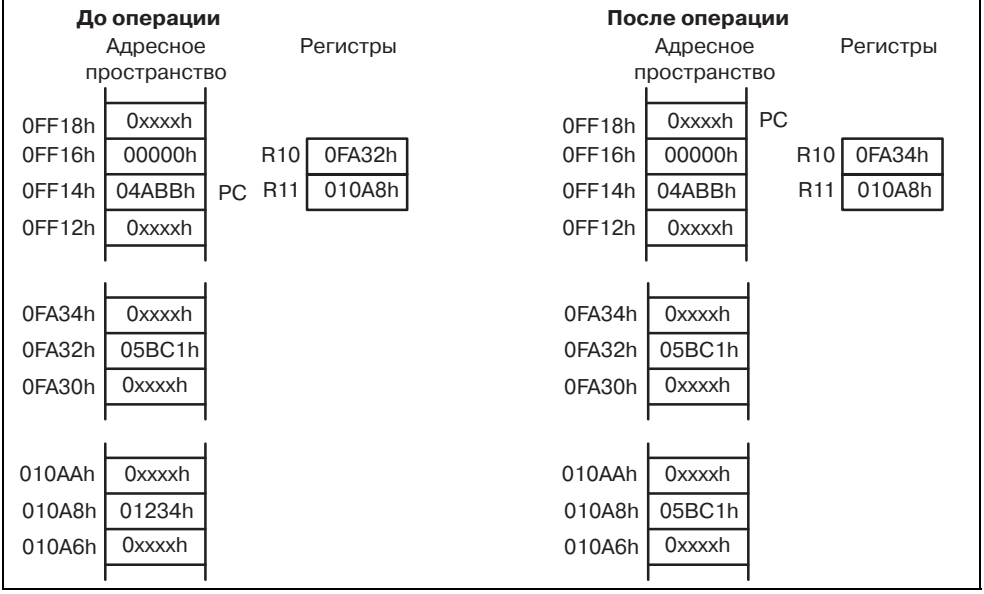


3.3.6. Косвенный регистровый режим адресации с автоинкрементом

Косвенный регистровый режим адресации с автоинкерментом описан в **Табл. 3.9**.

Таблица 3.9. **Описание косвенного регистрового режима адресации с автоинкрементом**

Ассемблерный код		Содержимое ПЗУ
`MOV @R10+,0(R11)`		`MOV @R10+,0(R11)`
Длина	Одно или два слова.	
Операция	Пересылка значения с исходного адреса (содержится в R10) по адресу назначения (содержится в R11). После выборки пересылаемого значения содержимое R10 увеличивается на 1 (при однобайтной операции) или на 2 (при двухбайтной операции), указывая на следующий адрес. Эта команда полезна при реализации табличных вычислений	
Комментарий	Допускается только для операнда-источника. Для операнда-приёмника используется 0(Rd) и добавляется вторая команда `INC Rd`	
Пример	`MOV @R10+,0(R11)`	



Автоинкрементирование содержимого регистра производится после выборки операнда из памяти (**Рис. 3.8**).

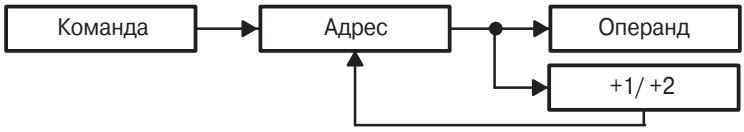


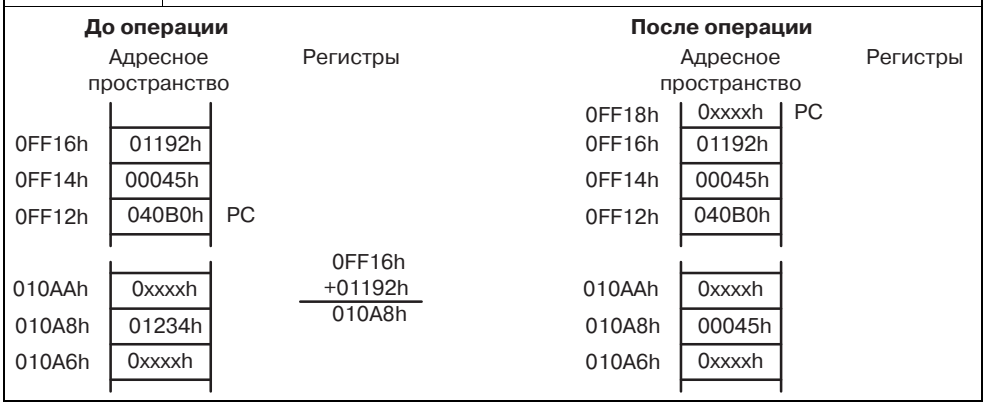
Рис. 3.8. Выборка операнда.

3.3.7. Непосредственный режим адресации

Непосредственный режим адресации описан в **Табл. 3.10**.

Таблица 3.10. **Описание непосредственного режима адресации**

Ассемблерный код		Содержимое ПЗУ
`MOV #45h,TONI`		`MOV @PC+,X(PC)` 45 X = TONI – PC
Длина	Два или три слова. На одно слово меньше при использовании генератора констант	
Операция	Пересылка константы, находящейся в слове, следующем за командой, по адресу назначения TONI. После выборки пересылаемого значения счётчик команд указывает на слово, расположенное после команды, и значение пересылается по указанному адресу	
Комментарий	Допускается только для операнда-источника	
Пример	`MOV #45h,TONI`	



3.4. Система команд

В общей сложности набор команд ЦПУ MSP430 включает в себя 27 команд ядра и 24 эмулируемых команды. Команды ядра — это команды, которые имеют уникальные коды операций, декодируемые ЦПУ. Эмулируемые команды облегчают создание и чтение кода, но не имеют собственных кодов операций, а автоматически заменяются ассемблером на эквивалентные команды ядра. Использование эмулируемых команд не приводит к увеличению размера кода или снижению производительности.

Существует три формата команд ядра:

- команды с двумя операндами;
- команды с одним операндом;
- команды перехода.

Все одно- и двухоперандные команды могут работать как с однобайтными, так и с двухбайтными значениями, используя расширения .B и .W соответственно. Команды, оперирующие байтами, используются для работы с однобайтными данными или доступа к 8-битным периферийным модулям. Команды, оперирующие словами, используются для работы с двухбайтными данными или доступа к

16-битным периферийным модулям. Если расширение команды не указано, то она использует 2-байтные операнды.

Источник и приёмник в команде определяются следующими полями:

src	Операнд-источник, определяемый битами As и S-reg
dst	Операнд-приёмник, определяемый битами Ad и D-reg
As	Биты, определяющие режим адресации источника (src)
S-reg	Рабочий регистр, используемый в качестве источника (src)
Ad	Биты, определяющие режим адресации приёмника (dst)
D-reg	Рабочий регистр, используемый в качестве приёмника (dst)
B/W	Одно- или двухбайтная операция: 0 — двухбайтная операция 1 — однобайтная операция

Примечание. Адрес назначения
Адрес назначения может быть любым в пределах адресного пространства. Тем не менее, при использовании команды, изменяющей содержимое приёмника, необходимо убедиться, что адрес назначения доступен для записи. К примеру, адрес, находящийся в диапазоне адресов масочного ПЗУ, будет корректным адресом назначения, однако его содержимое не может быть изменено, поэтому результат выполнения команды будет утерян.

3.4.1. Команды с двумя операндами (формат I)

Формат команды с двумя операндами представлен на **Рис. 3.9**.

15 14 13 12	11 10 9 8	7	6	5 4	3 2 1 0
Код операции	S-Reg	Ad	B/W	As	D-Reg

Рис. 3.9. Формат команды с двумя операндами.

Перечень и краткое описание команд с двумя операндами приведены в **Табл. 3.11**.

Примечание. Команды `CMP` и `SUB`
Команды `CMP` и `SUB` идентичны, за исключением возможности сохранения результата. Это справедливо и для команд `BIT` и `AND`.

3.4.2. Команды с одним операндом (формат II)

Формат команды с одним операндом представлен на **Рис. 3.10**.

15 14 13 12 11 10 9 8 7	6	5 4	3 2 1 0
Код операции	B/W	Ad	D/S-Reg

Рис. 3.10. Формат команды с одним операндом.

Таблица 3.11. **Команды с двумя операндами**

Мнемоника	S-reg, D-reg	Операция	Биты состояния			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst – src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (BCD-арифметика)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	*
BIC(.B)	src,dst	not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	*
AND(.B)	src,dst	src .and. dst → dst	0	*	*	*

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

Перечень и краткое описание команд с одним операндом приведены в **Табл. 3.12**.

Таблица 3.12. **Команды с одним операндом**

Мнемоника	S-reg, D-reg	Операция	Биты состояния			
			V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP – 2 → SP, src → @SP	–	–	–	–
SWPB	dst	Перестановка байтов местами	–	–	–	–
CALL	dst	SP – 2 → SP, PC+2 → @SP dst → PC	–	–	–	–
RETI	dst	TOS → SR, SP + 2 → SP TOS → PC,SP + 2 → SP	*	*	*	*
SXT	dst	Бит 7 → Бит 8.....Бит 15	0	*	*	*

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

С командой CALL можно использовать любые режимы адресации. При использовании относительного (ADDRESS), непосредственного (#N), абсолютного (&EDE) или индексного x(RN) режимов адресации значение адреса содержится в слове, расположенном после слова команды.

3.4.3. Команды перехода

Формат команд перехода представлен на **Рис. 3.11**.

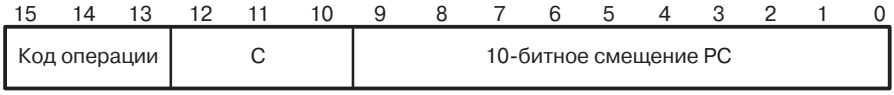


Рис. 3.11. Формат команды перехода.

Перечень и краткое описание команд перехода приведены в **Табл. 3.13**.

Таблица 3.13. **Команды перехода**

Мнемоника	S-reg, D-reg	Операция
JEQ/JZ	Метка	Переход к метке, если бит нуля (Z) установлен
JNE/JNZ	Метка	Переход к метке, если бит нуля (Z) сброшен
JC	Метка	Переход к метке, если бит переноса (C) установлен
JNC	Метка	Переход к метке, если бит переноса (C) сброшен
JN	Метка	Переход к метке, если бит отрицательного значения (N) установлен
JGE	Метка	Переход к метке, если (N .XOR. V) = 0
JL	Метка	Переход к метке, если (N .XOR. V) = 1
JMP	Метка	Безусловный переход к метке

Команды условного перехода осуществляют относительный переход по заданному смещению и не влияют на биты состояния ЦПУ. Переход может осуществляться в пределах от –511 до +512 слов относительно текущего значения PC. Величина смещения интерпретируется как 10-битное значение со знаком, которое удваивается и прибавляется к содержимому счётчика команд:

$$PC_{нов} = PC_{стар} + 2 + PC_{смещ} \times 2.$$

* ADC[.W], * ADC.B
Сложение переноса с операндом

Синтаксис	`ADC dst` или `ADC.W dst` `ADC.B dst`
Операция	dst + C → dst
Эмуляция	`ADDC #0,dst` `ADDC.B #0,dst`
Описание	Бит переноса (C) прибавляется к операнду-приёмнику. Предыдущее содержимое операнда теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если dst изменился с 0FFFFh до 0000, иначе сбрасывается. Устанавливается, если dst изменился с 0FFh до 00, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	16-битный счётчик, на который указывает R13, прибавляется к 32-битному счётчику, на который указывает R12.
	`ADD @R13,0(R12) ; Складываем младшие слова`
	`ADC 2(R12) ; Прибавляем перенос к старшему слову`

Пример 2	8-битный счётчик, на который указывает R13, прибавляется к 16-битному счётчику, на который указывает R12.
	`ADD.B @R13,0(R12) ; Складываем младшие байты`
	`ADC.B 1(R12) ; Прибавляем перенос к старшему байту`

ADD[.W], ADD.B
Сложение двух операндов

Синтаксис	`ADD  src,dst` или `ADD.W  src,dst` `ADD.B  src,dst`
Операция	src + dst → dst
Описание	Операнд-источник прибавляется к операнду-приёмнику. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются.

Пример 1	Регистр R5 увеличивается на 10. В случае переноса осуществляется переход к метке TONI.
	`ADD  #10,R5`
	`JC  TONI  ; Был перенос`
	`.....  ; Нет переноса`

Пример 2	Регистр R5 увеличивается на 10. В случае переноса осуществляется переход к метке TONI.
	`ADD.B  #10,R5  ; Прибавляем 10 к младшему байту регистра R5`
	`JC  TONI  ; Перенос произойдёт, если (R5) ≥ 246 [0Ah+0F6h]`
	`.....  ; Нет переноса`

ADDC[.W], ADDC.B
Сложение двух операндов с учётом переноса

Синтаксис	`ADDC src,dst` или `ADDC.W src,dst` `ADDC.B src,dst`
Операция	src + dst + C → dst
Описание	Операнд-источник и бит переноса (C) прибавляются к операнду-приёмнику. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из старшего бита результата, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	32-битный счётчик, на который указывает R13, прибавляется к 32-битному счётчику, смещённому на 11 слов (20/2 + 2/2) относительно адреса в R13.
	`ADD @R13+,20(R13) ; Складываем младшие слова`
	`ADDC @R13+,20(R13) ; Складываем старшие слова, учитывая перенос`

Пример 2	24-битный счётчик, на который указывает R13, прибавляется к 24-битному счётчику, смещённому на 11 байт относительно адреса в R13.
	`ADD.B @R13+,10(R13) ; Складываем младшие байты`
	`ADDC.B @R13+,20(R13) ; Складываем средние байты, учитывая перенос`
	`ADDC.B @R13+,20(R13) ; Складываем старшие байты, учитывая перенос`

AND[.W], AND.B
«Логическое И» двух операндов

Синтаксис	`AND src,dst` или `AND.W src,dst` `AND.B src,dst`
Операция	src .AND. dst → dst
Описание	Выполняется операция «Логическое И» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник
Биты состояния	N: Устанавливается, если MSB результата равен 1, иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (= .NOT. Zero). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра R5 используется в качестве битовой маски (#0AA55h) для слова, адресованного TOM. Если результат равен нулю, выполняется переход к метке `TONI`.
	`MOV #0AA55h,R5 ; Загружаем маску в R5`
	`AND R5,TOM ; Накладываем маску на слово TOM`
	`JZ TONI ;`
	`..... ; Результат не равен нулю`
	`;`
	`;`
	`; или`
	`;`
	`;`
	`AND #0AA55h,TOM`
	`JZ TONI`

Пример 2	Биты маски #0A5h логически перемножаются с младшим байтом TOM. Если результат равен нулю, выполняется переход к метке `TONI`.
	`AND.B #0A5h,TOM ; Накладываем маску #0A5h на младший байт TOM`
	`JZ TONI ;`
	`..... ; Результат не равен нулю`

BIC[.W], BIC.B
Очистка битов операнда

Синтаксис	`BIC  src,dst` или `BIC.W  src,dst` `BIC.B  src,dst`
Операция	.NOT. src .AND. dst → dst
Описание	Выполняется операция «Логическое И» между инвертированным значением операнда-источника и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Сбрасываются шесть старших битов слова LEO, расположенного в ОЗУ. `BIC  #0FC00h,LEO  ; Сбрасываем 6 старших битов по адресу LEO`
Пример 2	Сбрасываются пять старших битов байта LEO, расположенного в ОЗУ. `BIC.B  #0F8h,LEO  ; Сбрасываем 5 старших битов по адресу LEO`

BIS[.W], BIS.B
Установка битов операнда

Синтаксис	`BIS  src,dst` или `BIS.W  src,dst` `BIS.B  src,dst`
Операция	src .OR. dst → dst
Описание	Выполняется операция «Логическое ИЛИ» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Устанавливаются шесть младших битов слова TOM, расположенного в ОЗУ. `BIS  #003Fh,TOM  ; Устанавливаем 6 младших битов по адресу TOM`
Пример 2	Устанавливаются три старших бита байта TOM, расположенного в ОЗУ. `BIS.B  #0E0h,TOM  ; Устанавливаем 3 старших бита по адресу TOM`

BIT[.W], BIT.B
Проверка битов операнда

Синтаксис	`BIT src,dst` или `BIT.W src,dst` `BIT.B src,dst`
Операция	src .AND. dst
Описание	Выполняется операция «Логическое И» между операндом-источником и операндом-приёмником. Результат операции влияет только на биты состояния. Операнды не изменяются
Биты состояния	N: Устанавливается, если MSB результата равен 1, иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Если бит 9 регистра R8 установлен, выполняется переход к метке `TOM`.
	`BIT #0200h,R8 ; Бит 9 регистра R8 установлен?`
	`JNZ TOM ; Да, переходим к TOM`
	`..... ; Нет, продолжаем`

Пример 2	Если бит 3 регистра R8 установлен, выполняется переход к метке `TOM`.
	`BIT.B #8,R8`
	`JC TOM`

Пример 3	Проверяется бит, принятый по последовательному каналу (RCV). Поскольку при проверке одного бита с использованием команды `BIT` состояние флага переноса равно состоянию проверяемого бита, этот флаг используется в следующей команде. Получаемые данные побитно загружаются в регистр REGBUF.
	`; Обмен по последовательному каналу, первым передаётся младший бит (LSB)`
	`; xxxx xxxx xxxx xxxx`
	`BIT.B #RCV,RCCTL ; Принятый бит -> бит переноса`
	`RRC RECBUF ; Бит переноса -> старший бит RECBUF`
	`; cxxx xxxx`
	`..... ; повторяем две предыдущие команды`
	`..... ; восемь раз`
	`; cccc cccc`
	`; ^ ^`
	`; MSB LSB`
	`; Обмен по последовательному каналу, первым передаётся старший бит (MSB)`
	`BIT.B #RCV,RCCTL ; Принятый бит -> бит переноса`
	`RLC.B RECBUF ; Бит переноса -> младший бит RECBUF`
	`; xxxx xxxc`
	`..... ; Повторяем две предыдущие команды`
	`..... ; восемь раз`
	`; cccc cccc`
	`; ^ ^`
	`; MSB LSB`

* BR[.W], BRANCH
Переход по заданному адресу

Синтаксис	`BR dst`
Операция	dst → PC
Эмуляция	`MOV dst,PC`
Описание	Выполняется безусловный переход по любому адресу в пределах 64 КБ. Для операнда могут использоваться любые режимы адресации. В команде используется двухбайтный операнд
Биты состояния	Биты состояния не изменяются
Пример	Приведены примеры для всех режимов адресации.

```
BR   #EXEC   ; Переход к метке EXEC или заданному адресу
             ; (например, #0A4h)
             ; Команда ядра - MOV  @PC+,PC

BR   EXEC    ; Переход по адресу, находящемуся в EXEC
             ; Команда ядра - MOV  X(PC),PC
             ; Косвенная адресация

BR   &EXEC   ; Переход по адресу, находящемуся в ячейке памяти
             ; с абсолютным адресом EXEC
             ; Команда ядра - MOV  X(0),PC
             ; Косвенная адресация

BR   R5      ; Переход по адресу, содержащемуся в регистре R5
             ; Команда ядра - MOV  R5,PC
             ; Косвенная адресация по содержимому R5

BR   @R5     ; Переход по адресу, содержащемуся в слове памяти,
             ; адресуемом регистром R5
             ; Команда ядра - MOV  @R5,PC
             ; Косвенная адресация по косвенному содержимому R5

BR   @R5+    ; Переход по адресу, содержащемуся в слове памяти,
             ; адресуемом регистром R5, с последующим
             ; инкрементированием содержимого R5
             ; Команда ядра - MOV  @R5+,PC
             ; Косвенная адресация по косвенному содержимому R5
             ; с автоинкрементом

BR   X(R5)   ; Переход по адресу, содержащемуся в слове памяти
             ; с адресом R5 + X (например, обращение к таблице
             ; адресов, расположенной начиная с адреса X).
             ; X может быть адресом или меткой.
             ; Команда ядра - MOV  X(R5),PC
             ; Косвенная адресация по косвенному содержимому R5 + X
```

CALL
Вызов подпрограммы

Синтаксис	`CALL dst`
Операция	dst → tmp (dst вычисляется и запоминается) SP – 2 → SP PC → @SP (PC сохраняется в стеке) tmp → PC (dst загружается в PC)
Описание	Осуществляется вызов подпрограммы, расположенной по любому адресу в пределах 64 КБ. Адрес возврата (адрес следующей команды) сохраняется в стеке. В команде используется 2-байтный операнд
Биты состояния	Биты состояния не изменяются

Пример	Приведены примеры для всех режимов адресации.
	`CALL #EXEC ; Вызов с использованием метки EXEC`
	`; или непосредственного адреса (например, #0A4h)`
	`; SP-2 → SP, PC+2 → @SP, @PC+ → PC`
	
	`CALL EXEC ; Вызов п/п по адресу, находящемуся в EXEC`
	`; SP-2 → SP, PC+2 → @SP, X(PC) → PC`
	`; Косвенная адресация`
	
	`CALL &EXEC ; Вызов п/п по адресу, находящемуся в ячейке`
	`; с абсолютным адресом EXEC`
	`; SP-2 → SP, PC+2 → @SP, X(0) → PC`
	`; Косвенная адресация`
	
	`CALL R5 ; Вызов п/п по адресу, содержащемуся в регистре R5`
	`; SP-2 → SP, PC+2 → @SP, R5 → PC`
	`; Косвенная адресация по содержимому R5`
	
	`CALL @R5 ; Вызов п/п по адресу, содержащемуся в слове`
	`; памяти, адресуемом регистром R5`
	`; SP-2 → SP, PC+2 → @SP, @R5 → PC`
	`; Косвенная адресация по косвенному содержимому R5`
	
	`CALL @R5+ ; Вызов п/п по адресу, содержащемуся в слове`
	`; памяти, адресуемом регистром R5, с последующим`
	`; инкрементированием содержимого R5`
	`; SP-2 → SP, PC+2 → @SP, @R5+ → PC`
	`; Косвенная адресация по косвенному содержимому R5`
	`; с автоинкрементом`
	
	`CALL X(R5) ; Вызов п/п по адресу, содержащемуся в слове`
	`; памяти с адресом R5 + X (например, обращение`
	`; к таблице адресов, расположенной начиная`
	`; с адреса X). X может быть адресом или меткой.`
	`; SP-2 → SP, PC+2 → @SP, X(R5) → PC`
	`; Косвенная адресация по косвенному содержимому R5 + X`

* CLR[.W], * CLR.B
Очистка операнда

Синтаксис	`CLR dst` или `CLR.W dst` `CLR.B dst`
Операция	0 → dst
Эмуляция	`MOV #0,dst` `MOV.B #0,dst`
Описание	Операнд-приёмник обнуляется
Биты состояния	Биты состояния не изменяются
Пример 1	Обнуляется слово TONI в ОЗУ. `CLR TONI ; 0 -> TONI`
Пример 2	Обнуляется регистр R5. `CLR R5`
Пример 3	Обнуляется байт TONI, расположенный в ОЗУ. `CLR.B TONI ; 0 -> TONI`

* CLRC
Очистка бита переноса

Синтаксис	`CLRC`
Операция	0 → C
Эмуляция	`BIC #1,SR`
Описание	Бит переноса (C) сбрасывается. В команде используются 2-байтные операнды
Биты состояния	N: Не изменяется. Z: Не изменяется. C: Сбрасывается. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	16-битный счётчик, на который указывает R13, прибавляется к 32-битному счётчику, на который указывает R12.
	`CLRC ; C=0: начальное значение`
	`DADD @R13,0(R12) ; Прибавляем 16-битный счётчик к младшему`
	`; слову 32-битного счётчика`
	`DADC 2(R12) ; Прибавляем перенос к старшему слову`
	`; 32-битного счётчика`

* CLRN
Очистка бита отрицательного значения

Синтаксис	`CLRN`
Операция	0 → N или (.NOT. src .AND. dst → dst)
Эмуляция	`BIC  #4,SR`
Описание	Константа 04h инвертируется (0FFFBh) и логически перемножается (AND) с содержимым регистра состояния SR. Результат помещается в регистр состояния. В команде используются 2-байтные операнды
Биты состояния	N: Сбрасывается. Z: Не изменяется. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Сбрасывается флаг отрицательного значения в регистре состояния. Таким образом, исключается обработка отрицательных чисел в вызываемой подпрограмме.
	`  CLRN`
	`  CALL  SUBR`
	`  ......`
	`SUBR  JN  SUBRET  ; Если входной результат отрицательный,`
	`  ......  ; то ничего не делаем и выходим`
	`SUBRET  RET`

* CLRZ
Очистка бита нуля

Синтаксис	`CLRZ`
Операция	0 → Z или (.NOT. src .AND. dst → dst)
Эмуляция	`BIC  #2,SR`
Описание	Константа 02h инвертируется (0FFFDh) и логически перемножается (AND) с содержимым регистра состояния SR. Результат помещается в регистр состояния. В команде используются 2-байтные операнды
Биты состояния	N: Не изменяется. Z: Сбрасывается. C: Не изменяется. V: Не изменяется.
Биты реж.	OSCOFF, CPUOFF и GIE не изменяются
Пример	Сбрасывается флаг нуля в регистре состояния. `CLRZ`

CMP[.W], CMP.B
Сравнение двух операндов

Синтаксис	`CMP src,dst` или `CMP.W src,dst` `CMP.B src,dst`
Операция	dst + .NOT.src + 1 или (dst – src)
Описание	Операнд-источник вычитается из операнда приёмника. Для выполнения этой операции обратный код операнда-источника плюс 1 складывается с операндом-приёмником. Операнды не изменяются, результат операции не сохраняется — операция влияет только на биты состояния
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src <= dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если был перенос из MSB результата, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Сравнивается содержимое регистров R5 и R6. Если они равны, выполнение программы продолжается с метки `EQUAL`.
	`CMP R5,R6 ; R5 = R6?`
	`JEQ EQUAL ; Да, переходим`

Пример 2	Сравниваются два блока данных в ОЗУ. Если их содержимое различается, то производится переход к метке `ERROR`.
	`MOV #NUM,R5 ; Количество сравниваемых слов`
	`MOV #BLOCK1,R6 ; Начальный адрес 1-го блока -> R6`
	`MOV #BLOCK2,R7 ; Начальный адрес 2-го блока -> R7`
	`L$1 CMP @R6+,0(R7) ; 2-байтные значения равны?`
	`; R6 инкрементируется`
	`JNZ ERROR ; Нет, переходим к ERROR`
	`INCD R7 ; Инкрементируем R7`
	`DEC R5 ; Все элементы сравнили?`
	`JNZ L$1 ; Нет, продолжаем сравнение`

Пример 3	Сравниваются два байта в ОЗУ, расположенные по адресам EDE и TONI. Если они равны, то выполнение программы продолжается с метки `EQUAL`.
	`CMP.B EDE,TONI ; MEM(EDE) = MEM(TONI)?`
	`JEQ EQUAL ; Да, переходим`

* DADC[.W], * DADC.B
Сложение переноса с операндом (BCD-арифметика)

Синтаксис	`DADC dst` или `DADC.W dst` `DADC.B dst`
Операция	dst + C → dst (BCD-арифметика)
Эмуляция	`DADD #0,dst` `DADD.B #0,dst`
Описание	Бит переноса (C) прибавляется к операнду-приёмнику по правилам двоично-десятичной арифметики
Биты состояния	N: Устанавливается, если MSB результата равен 1, иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если dst изменился с 9999h до 0000, иначе сбрасывается. Устанавливается, если dst изменился с 99h до 00, иначе сбрасывается. V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	4-разрядное двоично-десятичное число, находящееся в R5, прибавляется к 8-разрядному числу, на которое указывает R8.
	`CLRC ; Сбрасываем бит переноса, задавая начальные`
	`; условия для следующей команды`
	`DADD R5,0(R8) ; Складываем младшие разряды + C`
	`DADC 2(R8) ; Прибавляем перенос к старшим разрядам`

Пример 2	2-разрядное двоично-десятичное число, находящееся в R5 прибавляется к 4-разрядному числу, на которое указывает R8.
	`CLRC ; Сбрасываем бит переноса, задавая начальные`
	`; условия для следующей команды`
	`DADD.B R5,0(R8) ; Складываем младшие разряды + C`
	`DADC.B 1(R8) ; Прибавляем перенос к старшим разрядам`

DADD[.W], DADD.B
Сложение двух операндов с учётом переноса (BCD-арифметика)

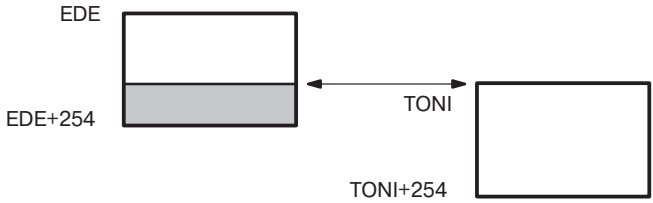
Синтаксис	`DADD src,dst` или `DADD.W src,dst` `DADD.B src,dst`
Операция	src + dst + C → dst (BCD-арифметика)
Описание	Операнд-источник и бит переноса (C) прибавляются к операнду-приёмнику по правилам двоично-десятичной арифметики. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется. Для не BCD-чисел результат операции не определён
Биты состояния	N: Устанавливается, если MSB результата равен 1, иначе сбрасывается Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат больше 9999, иначе сбрасывается. Устанавливается, если результат больше 99, иначе сбрасывается. V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	8-разрядное двоично-десятичное число, находящееся в R5 и R6, прибавляется к 8-разрядному числу, находящемуся в R3 и R4 (старшие разряды — в R6 и R4).
	`CLRC ; Сбрасываем бит переноса`
	`DADD R5,R3 ; Складываем младшие разряды`
	`DADD R6,R4 ; Складываем старшие разряды с учётом переноса`
	`JC OVERFLOW ; При переполнении переходим`
	`; к обработчику ошибок`

Пример 2	2-разрядный двоично-десятичный счётчик, расположенный в байте CNT, инкрементируется на 1.
	`CLRC ; Сбрасываем бит переноса`
	`DADD.B #1,CNT ; Инкрементируем BCD-счётчик`
	или
	`SETC`
	`DADD.B #0,CNT ; ≡ DADC.B CNT`

* DEC[.W], * DEC.B
Декрементирование операнда

Синтаксис	`DEC dst` или `DEC.W dst` `DEC.B dst`
Операция	dst – 1 → dst
Эмуляция	`SUB #1,dst` `SUB.B #1,dst`
Описание	Значение операнда-приёмника уменьшается на 1. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 1, иначе сбрасывается. C: Сбрасывается, если dst содержал 0, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается. Устанавливается, если исходное значение dst было 08000h, иначе сбрасывается. Устанавливается, если исходное значение dst было 080h, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Содержимое регистра R10 уменьшается на 1.
	`DEC R10 ; Декрементируем R10`
	`; Копируем 255-байтный блок с начальным адресом EDE в ОЗУ,`
	`; начиная с адреса TONI. Исходный и конечный блоки не должны`
	`; перекрываться: адрес TONI должен находиться вне диапазона`
	`; EDE...EDE + 0FEh`
	`;`
	`MOV #EDE,R6`
	`MOV #255,R10`
	`L$1 MOV.B @R6+,TONI-EDE-1(R6)`
	`DEC R10`
	`JNZ L$1`
	`; Не используйте данную процедуру для копирования`
	`; перекрывающихся блоков (Рис. 3.12)`
	 
	
	Рис. 3.12. Перекрытие блоков.

* DECD[.W], * DECD.B
Уменьшение операнда на 2

Синтаксис	`DECD dst` или `DECD.W dst` `DECD.B dst`
Операция	dst – 2 → dst
Эмуляция	`SUB #2,dst` `SUB.B #2,dst`
Описание	Значение операнда-приёмника уменьшается на 2. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 2, иначе сбрасывается. C: Сбрасывается, если dst содержал 0 или 1, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается. Устанавливается, если начальное значение dst было 08001h или 08000h, иначе сбрасывается. Устанавливается, если начальное значение dst было 081h или 080h, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра R10 уменьшается на 2.
	`DEC R10 ; Декрементируем R10`
	`; Копируем блок из 255 слов с начальным адресом EDE в другое`
	`; место, начиная с адреса TONI. Исходный и конечный блоки`
	`; не должны перекрываться: адрес TONI должен находиться вне`
	`; диапазона EDE...EDE + 01FCh`
	`;`
	`MOV #EDE,R6`
	`MOV #510,R10`
	`L$1 MOV @R6+,TONI-EDE-2(R6)`
	`DECD R10`
	`JNZ L$1`

Пример 2	Содержимое ячейки памяти с адресом LEO уменьшается на 2.
	`DECD.B LEO ; Декрементируем MEM(LEO)`
	Содержимое байта STATUS уменьшается на 2.
	`DECD.B STATUS`

* DINT
Общее запрещение прерываний

Синтаксис	`DINT`
Операция	0 → GIE или (0FFF7h .AND. SR → SR / .NOT.src .AND. dst → dst)
Эмуляция	`BIC #8,SR`
Описание	Запрещаются все прерывания. Константа 08h инвертируется и логически умножается (AND) на содержимое регистра состояния SR. Результат сохраняется в регистре состояния
Биты состояния	Биты состояния не изменяются
Биты режима	GIE сбрасывается, OSCOFF и CPUOFF не изменяются

Пример	Бит общего разрешения прерываний регистра SR сбрасывается для осуществления атомарного копирования 32-битного счётчика. Это гарантирует, что состояние счётчика не будет изменено в процессе копирования каким-либо прерыванием.
	`DINT ; Запрещаем генерацию прерываний`
	`NOP`
	`MOV COUNTHI,R5 ; Копируем счётчик`
	`MOV COUNTLO,R6`
	`EINT ; Разрешаем прерывания`

Примечание. Запрещение прерываний
При необходимости исключить прерывание какой-либо секции кода, между командой `DINT` и началом этой секции должна располагаться как минимум одна команда. Обычно для этой цели после команды `DINT` ставят команду `NOP`.

* EINT
Общее разрешение прерываний

Синтаксис	`EINT`
Операция	1 → GIE или (0008h .OR. SR → SR / src .OR. dst → dst)
Эмуляция	`BIS #8,SR`
Описание	Разрешаются все прерывания. Константа 08h логически складывается (OR) с содержимым регистра состояния SR. Результат сохраняется в регистре SR
Биты состояния	Биты состояния не изменяются
Биты режима	GIE устанавливается, OSCOFF и CPUOFF не изменяются

Пример	Бит общего разрешения прерываний (GIE) регистра состояния устанавливается.
	`; Обработчик прерываний от линий порта ввода/вывода P1.2...P1.7`
	`; P1IN - адрес регистра, из которого считывается состояние битов`
	`; порта. P1IFG - адрес регистра, в котором хранятся флаги`
	`; прерываний по линиям порта.`
	`PUSH.B &P1IN ; Сбрасываем только взведённые флаги`
	`BIC.B @SP,&P1IFG ; Маску для флагов прерываний`
	`; сохраняем в стеке`
	`EINT ; Разрешаем прочие прерывания`
	`BIT #Mask,@SP`
	`JEQ MaskOK ; Установленные флаги совпадают`
	`; с маской - переходим`
	`.....`
	`MaskOK BIC #Mask,@SP`
	`.....`
	`INCD SP ; Служебная операция, обратная`
	`; команде PUSH, расположенной в`
	`; начале обработчика. Корректирует`
	`; значение указателя стека`
	`RETI`

Примечание. Разрешение прерываний

Команда, следующая за командой разрешения прерываний (`EINT`), выполняется всегда, даже если на момент разрешения прерываний имеются отложенные запросы прерываний.

* INC[.W], * INC.B
Инкрементирование операнда

Синтаксис	`INC dst` или `INC.W dst` `INC.B dst`
Операция	dst + 1 → dst
Эмуляция	`ADD #1,dst` `ADD.B #1,dst`
Описание	Значение операнда-приёмника увеличивается на 1. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFFh, иначе сбрасывается. Устанавливается, если dst содержал 07Fh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Байт состояния процесса STATUS инкрементируется. Когда он становится равным 11, выполняется переход к метке `OVFL`.
	`INC.B STATUS`
	`CMP.B #11,STATUS`
	`JEQ OVFL`

* INCD[.W], * INCD.B
Увеличение операнда на 2

Синтаксис	`INCD dst` или `INCD.W dst` `INCD.B dst`
Операция	dst + 2 → dst
Эмуляция	`ADD #2,dst` `ADD.B #2,dst`
Описание	Значение операнда-приёмника увеличивается на 2. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFEh, иначе сбрасывается. Устанавливается, если dst содержал 0FEh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFEh или 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FEh или 0FFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFEh или 07FFFh, иначе сбрасывается. Устанавливается, если dst содержал 07FEh или 07Fh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Удаление значения с вершины стека без использования регистров.
	`.....`
	`PUSH R5 ; В R5 результат вычислений, сохранённый в`
	`; системном стеке`
	`INCD SP ; Удаляем значение с вершины стека, модифицируя`
	`; указатель стека`
	`; Не используйте INCD.B, так как содержимое SP`
	`; всегда выровнено по границе слова`
	`RET`

* INV[.W], * INV.B
Инвертирование операнда

Синтаксис	`INV dst` или `INV.W dst` `INV.B dst`
Операция	.NOT.dst → dst
Эмуляция	`XOR #0FFFFh,dst` `XOR.B #0FFh,dst`
Описание	Значение операнда-приёмника инвертируется. Предыдущее содержимое теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. C: Устанавливается, если результат не равен нулю, иначе сбрасывается (= .NOT. Zero). V: Устанавливается, если операнд имел отрицательное значение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Вычисляется дополнительный код содержимого регистра R5.
	`MOV #00AEh,R5 ; R5 = 000AEh`
	`INV R5 ; Инвертируем R5 R5 = 0FF51h`
	`INC R5 ; В R5 - доп. код R5 = 0FF52h`

Пример 2	Вычисляется дополнительный код байта LEO в ОЗУ.
	`MOV.B #00AEh,LEO ; MEM(LEO) = 0AEh`
	`INV.B LEO ; Инвертируем LEO MEM(LEO) = 051h`
	`INC.B LEO ; В LEO - доп. код MEM(LEO) = 052h`

* JC Переход[.W], если бит переноса установлен
* JHS Переход[.W], если «выше или равно»

Синтаксис	`JC label` `JHS label`
Операция	Если C = 1, то PC + 2 × смещение → PC. Если C = 0, то выполняется следующая команда.
Описание	Проверяется бит переноса (C) регистра состояния. Если он установлен, то смещение (10-битное число со знаком), находящееся в младших битах слова команды, прибавляется к счётчику команд. Если бит C сброшен, то выполняется команда, следующая за командой перехода. Команда `JC` (переход по «перенос»/«выше или равно») используется при сравнении чисел без знака (от 0 до 65535)
Биты состояния	Биты состояния не изменяются

Пример 1	Сигнал P1IN.1 используется для управления ходом выполнения программы.
	`BIT.B #02h,&P1IN ; Состояние бита порта -> бит переноса C`
	`JC PROGA ; Если C = 1, выполняем блок A программы`
	`..... ; C = 0, продолжаем выполнение отсюда`

Пример 2	Содержимое R5 сравнивается с числом 15. Если содержимое регистра больше или равно константе, выполняется переход к метке `LABEL`.
	`CMP #15,R5`
	`JHS LABEL ; Переходим, если R5 >= 15`
	`..... ; Остаёмся здесь, если R5 < 15`

JEQ Переход[.W], если «равно»
JZ Переход[.W], если ноль

Синтаксис	`JEQ label` `JZ label`
Операция	Если Z = 1, то PC + 2 × смещение → PC. Если Z = 0, то выполняется следующая команда.
Описание	Проверяется бит нуля (Z) регистра состояния. Если он установлен, то смещение (10-битное число со знаком), находящееся в младших битах слова команды, прибавляется к счётчику команд. Если бит Z сброшен, то выполняется команда, следующая за командой перехода
Биты состояния	Биты состояния не изменяются

Пример 1	Если содержимое R7 равно нулю, то выполняется переход к метке `TONI`.
	`TST R7 ; Проверяем R7`
	`JZ TONI ; Если ноль, то переходим`

Пример 2	Если содержимое R6 равно значению заданного элемента таблицы, то выполняется переход к метке `LEO`.
	`CMP R5,Table(R5) ; Сравниваем содержимое R6 с элементом`
	`; таблицы по адресу`
	`; (базовый адрес + содержимое R5)`
	`JEQ LEO ; Переходим, если значения равны`
	`..... ; Не равны, продолжаем выполнение программы`

Пример 3	Если содержимое R5 равно нулю, то выполняется переход к метке `LABEL`.
	`TST R5`
	`JZ LABEL`
	`.....`

JGE Переход[.W], если «больше или равно»

Синтаксис	`JGE label`
Операция	Если (N .XOR. V) = 0, то PC + 2 × смещение → PC. Если (N .XOR. V) = 1, то выполняется следующая команда.
Описание	Проверяются биты переполнения (V) и отрицательного значения (N) регистра состояния. Если установлены или сброшены оба бита, то смещение (10-битное число со знаком), находящееся в младших битах слова команды, прибавляется к счётчику команд. Если установлен только один бит, то выполняется команда, следующая за командой перехода. Эта команда используется при сравнении чисел со знаком
Биты состояния	Биты состояния не изменяются

Пример	Если содержимое регистра R6 больше или равно значению, адресуемому регистром R7, то выполняется переход к метке `EDE`.
	`CMP @R7,R6 ; R6 ≥ (R7)? - сравниваем числа со знаком`
	`JGE EDE ; Да, R6 ≥ (R7)`
	`..... ; Нет, продолжаем выполнение программы`

JL Переход[.W], если «меньше»

Синтаксис	`JL label`
Операция	Если (N .XOR. V) = 1, то PC + 2 × смещение → PC. Если (N .XOR. V) = 0, то выполняется следующая команда.
Описание	Проверяются биты переполнения (V) и отрицательного значения (N) регистра состояния. Если установлен только один из битов, то смещение (10-битное число со знаком), находящееся в младших битах слова команды, прибавляется к счётчику команд. Если установлены или сброшены оба бита, то выполняется команда, следующая за командой перехода. Эта команда используется при сравнении чисел со знаком
Биты состояния	Биты состояния не изменяются

Пример	Если содержимое регистра R6 меньше значения, адресуемого регистром R7, то выполняется переход к метке `EDE`.
	`CMP @R7,R6 ; R6 < (R7)? - сравниваем числа со знаком`
	`JL EDE ; Да, R6 < (R7)`
	`..... ; Нет, продолжаем выполнение программы`

JMP Безусловный переход

Синтаксис	`JMP  label`
Операция	PC + 2 × смещение → PC
Описание	Смещение (10-битное число со знаком), хранящееся в младших битах слова команды, прибавляется к счётчику команд
Биты состояния	Биты состояния не изменяются
Совет	Эта команда длиной в одно слово заменяет команду `BRANCH` при необходимости перехода в пределах от –511 до +512 слов относительно текущего значения счётчика команд

JN Переход[.W], если отрицательное значение

Синтаксис	`JN  label`
Операция	Если N = 1, то PC + 2 × смещение → PC. Если N = 0, то выполняется следующая команда.
Описание	Проверяется бит отрицательного значения (N) регистра состояния. Если он установлен, то смещение (10-битное число со знаком), находящееся в младших битах слова команды, прибавляется к счётчику команд. Если бит N сброшен, то выполняется команда, следующая за командой перехода
Биты состояния	Биты состояния не изменяются

Пример	Результат вычислений, сохранённый в регистре R5, вычитается из COUNT. Если результат отрицательный, то COUNT необходимо обнулить и продолжить выполнение программы с другого места.
	`SUB  R5,COUNT  ; COUNT - R5 -> COUNT`
	`JN  L$1  ; Если результат отрицателен, переходим к L$1`
	`.....  ; Продолжаем работать с COUNT ≥ 0`
	`.....`
	`.....`
	`.....`
	`L$1  CLR  COUNT`
	`.....`
	`.....`
	`.....`

JNC Переход[.W], если нет переноса
JLO Переход[.W], если «ниже»

Синтаксис	`JNC label` `JLO label`
Операция	Если C = 0, то PC + 2 × смещение → PC. Если C = 1, то выполняется следующая команда.
Описание	Проверяется бит переноса (C) регистра состояния. Если он сброшен, то смещение (10-битное число со знаком), находящееся в младших битах слова команды, прибавляется к счётчику команд. Если бит C установлен, то выполняется команда, следующая за командой перехода. Команда `JNC` (переход по «нет переноса»/ «ниже») используется при сравнении чисел без знака (от 0 до 65535)
Биты состояния	Биты состояния не изменяются

Пример 1	Содержимое R6 прибавляется к BUFFER. При переполнении выполняется переход к обработчику ошибок ERROR.
	`ADD R6,BUFFER ; BUFFER + R6 -> BUFFER`
	`JNC CONT ; Нет переноса, переходим к CONT`
	`ERROR ..... ; Начало обработчика ошибок`
	`.....`
	`.....`
	`.....`
	`CONT ..... ; Продолжаем выполнение программы`
	`.....`
	`.....`

Пример 2	Если байт STATUS равен 0 или 1, то выполняется переход к метке `STL2`.
	`CMP.B #2,STATUS`
	`JLO STL2 ; STATUS < 2`
	`..... ; STATUS ≥ 2, продолжаем выполнение программы`

JNE Переход[.W], если «не равно»
JNZ Переход[.W], если не ноль

Синтаксис	`JNE label` `JNZ label`
Операция	Если Z = 0, то PC + 2 × смещение → PC. Если Z = 1, то выполняется следующая команда.
Описание	Проверяется бит нуля (Z) регистра состояния. Если он сброшен, смещение (10-битное число со знаком), то находящееся в младших битах слова команды, прибавляется к счётчику команд. Если бит Z установлен, то выполняется команда, следующая за командой перехода
Биты состояния	Биты состояния не изменяются

Пример	Если содержимое R7 не равно R8, выполняется переход к метке `TONI`.
	`CMP R7,R8 ; Сравниваем R7 и R8`
	`JNE TONI ; Если не равны, то переходим`
	`..... ; Если равны, то продолжаем`

MOV[.W], MOV.B
Пересылка операнда

Синтаксис	`MOV  src,dst` или `MOV.W  src,dst` `MOV.B  src,dst`
Операция	src → dst
Описание	Операнд-источник пересылается в операнд-приёмник. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое 2-байтной таблицы EDE копируется в таблицу TOM. Размер таблиц — 020h элементов.
	`MOV #EDE,R10 ; Инициализируем указатель`
	`MOV #020h,R9 ; Инициализируем счетчик`
	`Loop MOV @R10+,TOM-EDE-2(R10) ; Используем указатель в R10`
	`; для обеих таблиц`
	`DEC R9 ; Декрементируем счетчик`
	`JNZ Loop ; Счётчик ≠ 0, продолжаем`
	`; копирование`
	`..... ; Копирование завершено`
	`.....`
	`.....`

Пример 2	Содержимое 1-байтной таблицы EDE копируется в таблицу TOM. Размер таблиц — 020h элементов.
	`MOV #EDE,R10 ; Инициализируем указатель`
	`MOV #020h,R9 ; Инициализируем счетчик`
	`Loop MOV.B @R10+,TOM-EDE-1(R10) ; Используем указатель в R10`
	`; для обеих таблиц`
	`DEC R9 ; Декрементируем счетчик`
	`JNZ Loop ; Счётчик ≠ 0, продолжаем`
	`; копирование`
	`..... ; Копирование завершено`
	`.....`
	`.....`

* NOP
Нет операции

Синтаксис	NOP
Операция	Нет
Эмуляция	MOV #0,R3
Описание	Не выполняется никаких операций. Команда может использоваться для замены рабочих команд при отладке программы или для формирования задержек
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
	Команда NOP используется, главным образом, в двух случаях: • Для заполнения 1, 2 или 3 слов в памяти. • Для корректировки времени выполнения программы.

Примечание. Эмуляция команды NOP

Действие команды NOP можно эмулировать с помощью других команд, получая при этом разные длительности задержек и размер кода. Ниже представлено несколько примеров:

```
MOV   #0,R3       ; 1 такт, 1 слово
MOV   0(R4),0(R4) ; 6 тактов, 3 слова
MOV   @R4,0(R4)   ; 5 тактов, 2 слова
BIC   #0,EDE(R4)  ; 4 такта, 2 слова
JMP   $+2         ; 2 такта, 1 слово
BIC   #0,R5       ; 1 такт, 1 слово
```

Однако при использовании этих примеров нужно соблюдать осторожность, чтобы избежать непредсказуемых результатов. Например, если использовать команду MOV 0(R4),0(R4) при R4 = 120h, то произойдет нарушение защиты сторожевого таймера (адрес 120h), поскольку ключ защиты не был указан.

* POP[.W], * POP.B
Извлечение операнда из стека

Синтаксис	`POP dst` или `POP.W dst` `POP.B dst`
Операция	@SP → temp SP + 2 → SP temp → dst
Эмуляция	`MOV @SP+,dst` или `MOV.W @SP+,dst` `MOV.B @SP+,dst`
Описание	Элемент стека, адресуемый указателем стека (TOS), извлекается в операнд-приёмник. После этого указатель стека увеличивается на 2
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое R7 и регистра состояния восстанавливаются из стека.
	`POP R7 ; Извлекаем R7`
	`POP SR ; Извлекаем регистр состояния`

Пример 2	Содержимое байта LEO в ОЗУ восстанавливается из стека. `POP.B LEO ; Копируем младший байт из стека в LEO`

Пример 3	Содержимое R7 восстанавливается из стека.
	`POP.B R7 ; Копируем младший байт из стека в R7`
	`; Старший байт R7 = 0`

Пример 4	Содержимое ячейки памяти, на которую указывает R7, и регистра состояния восстанавливается из стека.
	`POP.B 0(R7) ; Копируем младший байт с вершины стека в ОЗУ`
	`; по адресу, хранящемуся в регистре R7`
	`POP SR ; Загружаем последнее слово из стека в регистр SR`

Примечание. Указатель системного стека
Указатель системного стека (SP) всегда увеличивается на 2, независимо от разрядности (байт/слово) операнда команды.

PUSH[.W], PUSH.B
Сохранение операнда в стеке

Синтаксис	`PUSH src` или `PUSH.W src` `PUSH.B src`
Операция	SP – 2 → SP src → @SP
Описание	Указатель стека уменьшается на 2, после чего содержимое операнда-источника помещается в ОЗУ по адресу, определяемому указателем стека (TOS)
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра состояния и регистра R8 сохраняется в стеке.
	`PUSH SR ; Сохраняем регистр состояния`
	`PUSH R8 ; Сохраняем регистр R8`

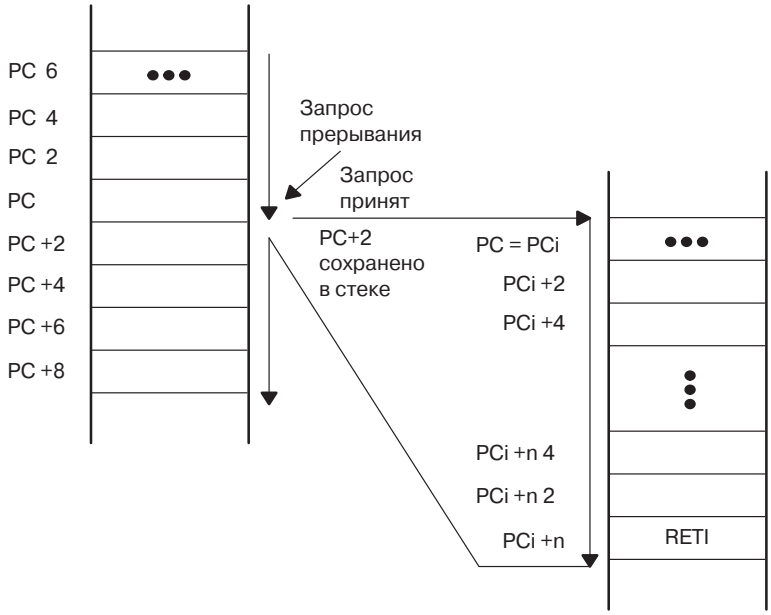
Пример 2	Содержимое регистра TCDAT периферийного модуля сохраняется в стеке.
	`PUSH.B &TCDAT ; Сохраняем в стеке данные 8-битного`
	`; периферийного модуля`

Примечание. Указатель системного стека
Указатель системного стека (SP) всегда уменьшается на 2, независимо от разрядности (байт/слово) операнда команды.

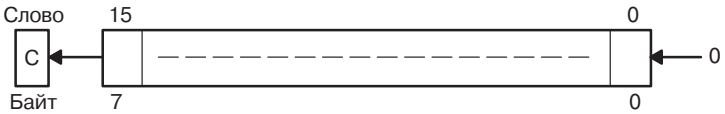
* RET
Возврат из подпрограммы

Синтаксис	`RET`
Операция	@SP → PC SP + 2 → SP
Эмуляция	`MOV @SP+,PC`
Описание	Адрес возврата, помещённый в стек при выполнении команды `CALL`, загружается в счётчик команд. Выполнение программы продолжается с команды, следующей за командой вызова подпрограммы
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

RETI
Возврат из прерывания

Синтаксис	`RETI`
Операция	TOS → SR SP + 2 → SP TOS → PC SP + 2 → SP
Описание	Содержимое регистра состояния восстанавливается таким, каким оно было при входе в процедуру обработки прерывания. Для этого значение, расположенное на вершине стека (TOS), копируется в регистр состояния. Затем указатель стека увеличивается на 2. Содержимое счётчика команд восстанавливается таким, каким оно было на момент принятия запроса прерывания. Для этого значение, расположенное на вершине стека (TOS), копируется в счётчик команд. Затем указатель стека увеличивается на 2
Биты состояния	N: Восстанавливается из стека. Z: Восстанавливается из стека. C: Восстанавливается из стека. V: Восстанавливается из стека.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	На **Рис. 3.13** показан процесс прерывания основной программы.  ***Рис. 3.13.*** Прерывание основной программы.

* RLA[.W], * RLA.B
Арифметический сдвиг влево

Синтаксис	`RLA dst` или `RLA.W dst` `RLA.B dst`
Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← 0
Эмуляция	`ADD dst,dst` `ADD.B dst,dst`
Описание	Содержимое операнда-приёмника сдвигается влево на один бит, как показано на **Рис. 3.14**. Старший бит содержимого (MSB) помещается в бит переноса (C), а младший бит (LSB) обнуляется. По сути, команда `RLA` выполняет операцию знакового умножения на 2. Если перед выполнением операции значение операнда было 04000h ≤ dst < 0C000h (040h ≤ dst < 0C0h), то возникает переполнение: знак результата отличается от знака операнда.  ***Рис. 3.14.*** Операнд-приёмник — арифметический сдвиг влево.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB. V: Устанавливается, если произошло переполнение: начальное значение 04000h ≤ dst < 0C000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 040h ≤ dst < 0C0h; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Содержимое регистра R7 умножается на 2. `RLA R7 ; Сдвигаем R7 на 1 бит влево (×2)`

Пример 2	Младший байт регистра R7 умножается на 4.
	`RLA.B R7 ; Сдвигаем влево младший байт R7 (×2)`
	`RLA.B R7 ; Сдвигаем влево младший байт R7 (×4)`

Примечание. Замена команды `RLA`

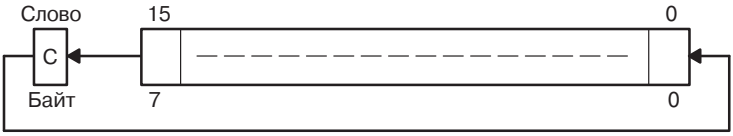
Ассемблер не распознаёт команды:

`RLA @R5+`, `RLA.B @R5+` или `RLA(.B) @R5`

Они должны быть заменены командами:

`ADD @R5+, -2(R5)` `ADD.B @R5+,-1(R5)` или `ADD(.B) @R5,0(R5)`

* RLC[.W], * RLC.B
Сдвиг влево через перенос

Синтаксис	`RLC dst` или `RLC.W dst` `RLC.B dst`
Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← C
Эмуляция	`ADDC dst,dst` `ADDC.B dst,dst`
Описание	Содержимое операнда-приёмника сдвигается влево на один бит, как показано на **Рис. 3.15**. Бит переноса (C) вдвигается в младший бит (LSB) операнда, а старший бит операнда (MSB) выдвигается в бит переноса.  ***Рис. 3.15.*** Операнд-приёмник — сдвиг влево через перенос.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB. V: Устанавливается, если произошло переполнение: начальное значение 04000h ≤ dst < 0C000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 040h ≤ dst < 0C0h; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	Содержимое регистра R5 сдвигается на 1 бит влево. `RLC R5 ; (R5 × 2) + C -> R5`

Пример	Значение бита порта P1IN.1 вдвигается в младший бит регистра R5.
	`BIT.B #2,&P1IN ; Информация -> бит переноса`
	`RLC R5 ; C = P1IN.1 -> младший бит R5`

Пример	Содержимое байта ОЗУ с адресом LEO сдвигается на 1 бит влево. `RLC.B LEO ; MEM(LEO) × 2 + C -> MEM(LEO)`

Примечание. Замена команды `RLC`

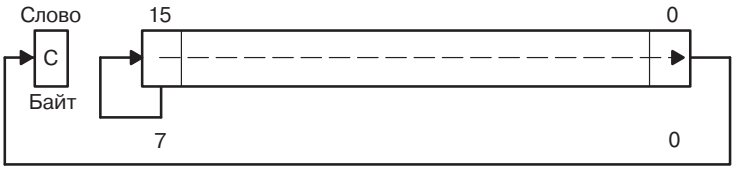
Ассемблер не распознаёт команды:

`RLC @R5+`, `RLC.B @R5+` или `RLC(.B) @R5`

Они должны быть заменены командами:

`ADDC @R5+, -2(R5)` `ADDC.B @R5+,-1(R5)` или `ADDC(.B) @R5,0(R5)`

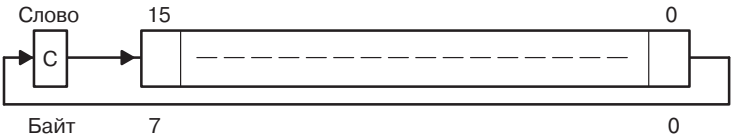
RRA[.W], RRA.B
Арифметический сдвиг вправо

Синтаксис	`RRA dst` или `RRA.W dst` `RRA.B dst`
Операция	MSB → MSB, MSB → MSB–1, ... LSB+1 → LSB, LSB → 0
Описание	Содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 3.16**. Старший бит содержимого (MSB) вдвигается в бит MSB-1 (значение MSB остаётся неизменным), а бит LSB+1 вдвигается в младший бит (LSB). По сути, команда `RLA` выполняет операцию знакового деления на 2.  ***Рис. 3.16.*** Операнд-приёмник — арифметический сдвиг вправо.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Содержимое регистра R5 сдвигается вправо на один бит, состояние старшего бита не изменяется. Эта операция эквивалентна операции деления на 2. `RRA R5 ; R5/2-> R5`

Пример 2	Содержимое регистра R5 умножается на 0.75.
	`PUSH R5 ; Временно сохраняем R5 в стеке`
	`RRA R5 ; R5 × 0.5 -> R5`
	`ADD @SP+,R5 ; R5 × 0.5 + R5 = 1.5 × R5 -> R5`
	`RRA R5 ; (1.5 × R5) × 0.5 = 0.75 × R5 ->R5`

Пример 3	Младший байт регистра R5 сдвигается вправо на один бит, состояние старшего бита не изменяется. Эта операция эквивалентна операции деления на 2.
	`RRA.B R5 ; R5/2-> R5: операция только над младшим байтом`
	`; Старший байт R5 обнуляется`
	`PUSH.B R5 ; R5 × 0.5 -> TOS`
	`RRA.B @SP ; TOS × 0.5 = 0.5 × R5 × 0.5 =0.25 × R5 -> TOS`
	`ADD.B @SP+,R5 ; R5 × 0.5 + R5 × 0.25 = 0.75 × R5 -> R5`

RRC[.W], RRC.B
Сдвиг вправо через перенос

Синтаксис	`RRC  dst` или `RRC.W  dst` `RRC.B  dst`
Операция	C → MSB → MSB–1 ... LSB+1 → LSB → C
Описание	Содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 3.17**. Бит переноса (C) вдвигается в старший бит (MSB) операнда, а младший бит операнда (LSB) выдвигается в бит переноса (C).  ***Рис. 3.17.*** Операнд-приёмник — сдвиг вправо через перенос.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра R5 сдвигается на 1 бит вправо. В MSB загружается 1.
	`SETC  ; Инициализируем бит переноса`
	`RRC  R5  ; R5/2 + 8000h -> R5`

Пример 2	Содержимое регистра R5 сдвигается на 1 бит вправо. В MSB загружается 1.
	`SETC  ; Инициализируем бит переноса`
	`RRC.B  R5  ; R5/2 + 80h -> R5 (используется младший байт R5)`

* SBC[.W], * SBC.B
Вычитание заёма из операнда

Синтаксис	`SBC dst` или `SBC.W dst` `SBC.B dst`
Операция	dst + 0FFFFh + C → dst dst + 0FFh + C → dst
Эмуляция	`SUBC #0,dst` `SUBC.B #0,dst`
Описание	Бит переноса (C), уменьшенный на 1, прибавляется к операнду-приёмнику. Предыдущее содержимое операнда теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	16-битный счётчик, на который указывает R13, вычитается из 32-битного счётчика, на который указывает R12.
	`SUB @R13,0(R12) ; Вычитаем младшие слова`
	`SBC 2(R12) ; Вычитаем заём из старшего слова`

Пример 2	8-битный счётчик, на который указывает R13, вычитается из 16-битного счётчика, на который указывает R12.
	`SUB.B @R13,0(R12) ; Вычитаем младшие байты`
	`SBC.B 1(R12) ; Вычитаем заём из старшего байта`

Примечание. Признак заёма

В качестве признака заёма используется инверсное значение бита переноса:

Заём	Бит C
Есть	0
Нет	1

* SETC
Установка бита переноса

Синтаксис	`SETC`
Операция	1 → C
Эмуляция	`BIS  #1,SR`
Описание	Устанавливается бит переноса C
Биты состояния	N: Не изменяется. Z: Не изменяется. C: Устанавливается. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Эмуляция BCD-вычитания.
	Содержимое R5 вычитается из R6 по правилам BCD-арифметики. Полагаем, что R5 = #03987h, а R6 = #04137h.
	`DSUB  ADD  #06666h,R5  ; Преобразуем значения разрядов R5`
	`  ; от 0-9 к 6-0Fh.`
	`  ; R5 = 03987h + 06666h = 09FEDh`
	`  INV  R5  ; Инвертируем промежуточный результат`
	`  ; R5 = .NOT. R5 = 06012h`
	`  SETC  ; Инициализируем бит переноса`
	`  DADD  R5,R6  ; Эмулируем вычитание сложением:`
	`  ; (010000h - R5 - 1)`
	`  ; R6 = R6 + R5 + 1`
	`  ; R6 = 0150h`

* SETN
Установка бита отрицательного значения

Синтаксис	`SETN`
Операция	1 → N
Эмуляция	`BIS  #4,SR`
Описание	Устанавливается бит отрицательного значения N
Биты состояния	N: Устанавливается. Z: Не изменяется. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

* SETZ
Установка бита нуля

Синтаксис	`SETZ`
Операция	1 → Z
Эмуляция	`BIS #2,SR`
Описание	Устанавливается бит нуля Z
Биты состояния	N: Не изменяется. Z: Устанавливается. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

SUB[.W], SUB.B
Вычитание двух операндов

Синтаксис	`SUB src,dst` или `SUB.W src,dst` `SUB.B src,dst`
Операция	dst + .NOT.src + 1 → dst или [(dst – src → dst)]
Описание	Операнд-источник вычитается из операнда-приёмника путём прибавления к последнему обратного кода операнда-источника плюс единица. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	16-битный счётчик, на который указывает R13, вычитается из 32-битного счётчика, на который указывает R12.
	`SUB @R13,0(R12) ; Вычитаем младшие слова`
	`SBC 2(R12) ; Вычитаем перенос из старшего слова`

Пример 2	8-битный счётчик, на который указывает R13, вычитается из 16-битного счётчика, на который указывает R12.
	`SUB.B @R13,0(R12) ; Вычитаем младшие байты`
	`SBC.B 1(R12) ; Вычитаем перенос из старшего байта`

Примечание. Признак заёма

В качестве признака заёма используется инверсное значение бита переноса:

Заём	Бит C
Есть	0
Нет	1

SUBC[.W], SBB[.W], SUBC.B, SBB.B
Вычитание двух операндов с учётом заёма

Синтаксис	`SUBC src,dst` или `SUBC.W src,dst`
	`SBB src,dst` или `SBB.W src,dst`
	`SUBC.B src,dst` или `SBB.B src,dst`

Операция	dst + .NOT.src + C → dst или (dst – src – 1 + C → dst)
Описание	Операнд-источник вычитается из операнда-приёмника путём прибавления к последнему обратного кода операнда-источника и значения бита переноса. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Вычитаются две мантиссы (24-битные) чисел с плавающей точкой. Младшие байты находятся в R13 и R10, старшие байты — в R12 и R9.
	`SUB.W R13,R10 ; 16-битная часть, младшие биты`
	`SUBC.B R12,R9 ; 8-битная часть, старшие биты`

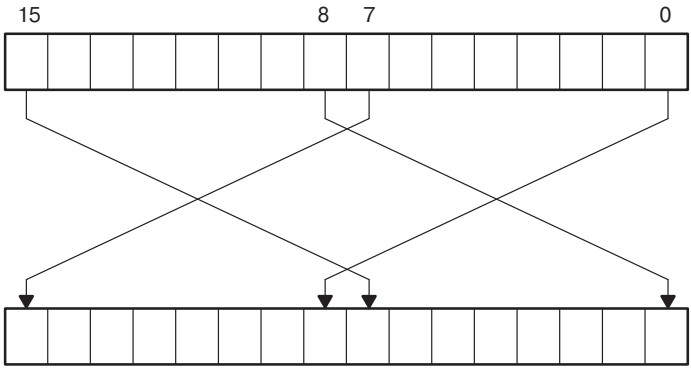
Пример 2	16-битный счетчик, адресуемый R13, вычитается из 16-битного счётчика, находящегося в регистрах R10 и R11.
	`SUB.B @R13+,R10 ; Вычитаем младшие байты без заёма`
	`SUBC.B @R13,R11 ; Вычитаем старшие байты, учитывая заём`
	`..... ; от предыдущей операции`

Примечание. Признак заёма

В качестве признака заёма используется инверсное значение бита переноса:

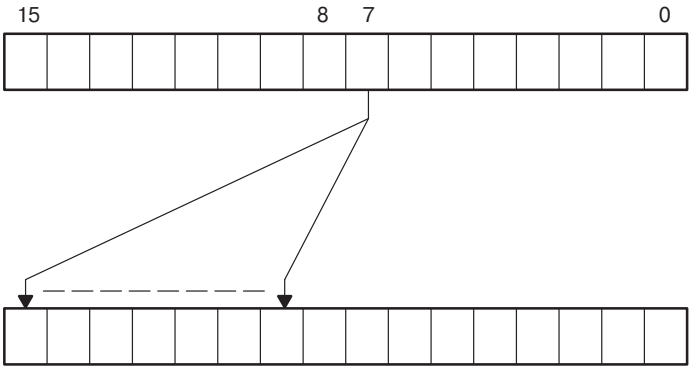
Заём	Перенос
Есть	0
Нет	1

SWPB
Перестановка байтов

Синтаксис	`SWP  dst`
Операция	Биты 15...8 ↔ биты 7...0
Описание	Старший и младший байты операнда-приёмника меняются местами, как показано на **Рис. 3.18**.  ***Рис. 3.18.*** Перестановка байтов операнда-приёмника.
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	`MOV  #040BFh,R7  ; 0100000010111111 -> R7` `SWPB  R7  ; B R7 - 1011111101000000`

Пример 2	Содержимое регистра R5 умножается на 256. Результат сохраняется в регистрах R5 и R4.
	`SWPB  R5  ; Переставляем байты в R4`
	`MOV  R5,R4  ; Копируем новое значение R4`
	`BIC  #0FF00h,R5  ; Корректируем результат`
	`BIC  #00FFh,R4  ; Корректируем результат`

SXT
Расширение знака

Синтаксис	`SXT  dst`
Операция	Биты 7 → биты 8...15
Описание	Знаковый бит младшего байта слова копируется во все биты старшего байта, как показано на **Рис. 3.19**.  ***Рис. 3.19.*** Расширение знака операнда-приёмника.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (= .NOT.Zero). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	В регистр R7 загружается содержимое 8-битного регистра P1IN. С помощью операции расширения знака в биты 8...15 заносится значение 7-го бита.
	`MOV.B  &P1IN,R7  ; P1IN = 080h:  .... .... 1000 0000`
	`SXT  R7  ; R7 = 0FF80h:  1111 1111 1000 0000`

* TST[.W], * TST.B
Проверка операнда (на ноль)

Синтаксис	`TST dst` или `TST.W dst` `TST.B dst`
Операция	dst + 0FFFFh + 1 dst + 0FFh + 1
Эмуляция	`CMP #0,dst` `CMP.B #0,dst`
Описание	Операнд-приёмник сравнивается с нулём и в соответствии с результатом изменяются биты состояния. Сам операнд остаётся неизменным
Биты состояния	N: Устанавливается, если операнд отрицателен, сбрасывается — если положителен. Z: Устанавливается, если операнд равен нулю, иначе сбрасывается. C: Устанавливается. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Проверяется содержимое регистра R7. Если оно отрицательное, то выполнение программы продолжается с метки `R7NEG`, если положительное и не равно нулю — с метки `R7POS`.
	`TST R7 ; Проверяем R7`
	`JN R7NEG ; R7 < 0`
	`JZ R7ZERO ; R7 = 0`
	`R7POS ..... ; R7 > 0`
	`R7NEG ..... ; R7 < 0`
	`R7ZERO ..... ; R7 = 0`

Пример 2	Проверяется младший байт регистра R7. Если он отрицателен, то выполнение программы продолжается с метки `R7NEG`, если положительное и не равно нулю — с метки `R7POS`.
	`TST.B R7 ; Проверяем R7`
	`JN R7NEG ; R7 < 0`
	`JZ R7ZERO ; R7 = 0`
	`R7POS ..... ; R7 > 0`
	`R7NEG ..... ; R7 < 0`
	`R7ZERO ..... ; R7 = 0`

XOR[.W], XOR.B
«Исключающее ИЛИ» двух операндов

Синтаксис	`XOR  src,dst` или `XOR.W  src,dst` `XOR.B  src,dst`
Операция	src .XOR. dst → dst
Описание	Выполняется операция «Исключающее ИЛИ» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется
Биты состояния	N: Устанавливается, если MSB результата равен 1, иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (= .NOT.Zero). V: Устанавливается, если оба операнда отрицательные.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Установленные биты регистра R6 изменяют состояние соответствующих битов слова TONI в ОЗУ. `XOR  R6,TONI  ; Изменяем биты TONI в соответствии с R6`
Пример 2	Установленные биты регистра R6 изменяют состояние соответствующих битов байта TONI в ОЗУ. `XOR.B  R6,TONI  ; Изменяем биты TONI в соответствии с R6`

Пример 3	Сбрасываются в 0 те биты младшего байта R7, которые отличаются от соответствующих битов 1-байтной переменной `EDE`.
	`XOR.B  EDE,R7  ; Устанавливаем отличающиеся биты в 1`
	`INV.B  R7  ; Инвертируем младший байт, старший байт = 0`

3.4.4. Время выполнения и размер команд

Число тактов ЦПУ, требующееся для выполнения той или иной команды, зависит от формата команды и от используемого режима адресации — но не от собственно инструкции. Приведённые далее цифры относятся к тактовому сигналу MCLK.

Сброс и прерывания

Число тактов ЦПУ, требуемое для обслуживания сброса и прерываний, указано в **Табл. 3.14**.

Таблица 3.14. **Обслуживание сброса и прерываний**

Действие	Число тактов	Длина команды
Возврат из прерывания (`RETI`)	5	1
Принятие прерывания	6	–
Сброс от сторожевого таймера	4	–
Аппаратный сброс ($\overline{RST}$/NMI)	4	–

Время выполнения и длина команд формата II (с одним операндом)

Длины всех команд формата II и время их выполнения для всех возможных режимов адресации приведены в **Табл. 3.15**.

Таблица 3.15. **Время выполнения и длина команд формата II**

Режим адресации	Число тактов			Длина команды	Пример
	`RRA, RRC, SWPB, SXT`	`PUSH`	`CALL`		
`Rn`	1	3	4	1	`SWPB R5`
`@Rn`	3	4	4	1	`RRC @R9`
`@Rn+`	3	5	5	1	`SWPB @R10+`
`#N`	См. примеч.	4	5	2	`CALL #0F000h`
`X(Rn)`	4	5	5	2	`CALL 2(R7)`
`EDE`	4	5	5	2	`PUSH EDE`
`&EDE`	4	5	5	2	`SXT &EDE`

Примечание. Непосредственный режим адресации в командах формата II
Не используйте непосредственный режим адресации с командами `RRA`, `RRC`, `SWPB` и `SXT`. Использование непосредственной адресации для этих команд приведёт к непредсказуемому поведению программы.

Время выполнения и длина команд формата III (команды перехода)

Все команды перехода занимают одно слово в памяти и выполняются за два такта, независимо от того, был или не был выполнен переход.

Время выполнения и длина команд формата I (с двумя операндами)

Длина команд формата I и время их выполнения для всех возможных режимов адресации приведены в **Табл. 3.16**.

Таблица 3.16. **Время выполнения и длина команд формата I**

Режим адресации		Число тактов	Длина команды	Пример
Источник	Приёмник			
Rn	Rm	1	1	MOV R5,R8
	PC	2	1	BR R9
	x(Rm)	4	2	ADD R5,4(R6)
	EDE	4	2	XOR R8,EDE
	&EDE	4	2	MOV R5,&EDE
@Rn	Rm	2	1	AND @R4,R5
	PC	2	1	BR @R8
	x(Rm)	5	2	XOR @R5,8(R6)
	EDE	5	2	MOV @R5,EDE
	&EDE	5	2	XOR @R5,&EDE
@Rn+	Rm	2	1	ADD @R5+,R6
	PC	3	1	BR @R9+
	x(Rm)	5	2	XOR @R5+,8(R6)
	EDE	5	2	MOV @R9+,EDE
	&EDE	5	2	MOV @R9+,&EDE
#N	Rm	2	2	MOV #20,R9
	PC	3	2	BR #2AEh
	x(Rm)	5	3	MOV #0300h,0(SP)
	EDE	5	3	ADD #33,EDE
	&EDE	5	3	ADD #33,&EDE
x(Rn)	Rm	3	2	MOV 2(R5),R7
	PC	3	2	BR 2(R6)
	TONI	6	3	MOV 4(R7),TONI
	x(Rm)	6	3	ADD 4(R4),6(R9)
	&TONI	6	3	MOV 2(R4),&TONI
EDE	Rm	3	2	AND EDE,R6
	PC	3	2	BR EDE
	TONI	6	3	CMP EDE,TONI
	x(Rm)	6	3	MOV EDE,0(SP)
	&TONI	6	3	MOV EDE,&TONI
&EDE	Rm	3	2	MOV &EDE,R8
	PC	3	2	BR &EDE
	TONI	6	3	MOV &EDE,TONI
	x(Rm)	6	3	MOV &EDE,0(SP)
	&TONI	6	3	MOV &EDE,&TONI

3.4.5. Описание набора команд

Карта команд изображена на **Рис. 3.20**. Описание всех команд набора в сжатом виде приведено в **Табл. 3.17**.

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

Рис. 3.20. Карта команд ядра.

Таблица 3.17. **Набор команд MSP430**

Мнемоника	Описание	Операция	V	N	Z	C
ADC(.B)* dst	Сложение бита переноса с операндом	dst + C → dst	•	•	•	•
ADD(.B) src,dst	Сложение двух операндов	src + dst → dst	•	•	•	•
ADDC(.B) src,dst	Сложение двух операндов с учётом бита переноса	src + dst + C → dst	•	•	•	•
AND(.B) src,dst	«Логическое И» двух операндов	src .and. dst → dst	0	•	•	•
BIC(.B) src,dst	Очистка битов операнда	.not.src .and. dst → dst	–	–	–	–
BIS(.B) src,dst	Установка битов операнда	src .or. dst → dst	–	–	–	–
BIT(.B) src,dst	Проверка битов операнда	src .and. dst	0	•	•	•
BR* dst	Переход по заданному адресу	dst → PC	–	–	–	–
CALL dst	Вызов подпрограммы	PC+2 → stack, dst → PC	–	–	–	–
CLR(.B)* dst	Очистка операнда	0 → dst	–	–	–	–
CLRC*	Очистка бита C	0 → C	–	–	–	0
CLRN*	Очистка бита N	0 → N	–	0	–	–
CLRZ*	Очистка бита Z	0 → Z	–	–	0	–
CMP(.B) src,dst	Сравнение двух операндов	dst – src	•	•	•	•

Таблица 3.17. **Набор команд MSP430** *(продолжение)*

Мнемоника	Описание	Операция	V	N	Z	C
DADC(.B)* dst	Сложение бита переноса с операндом (BCD-арифметика)	dst + C → dst (BCD)	•	•	•	•
DADD(.B) src,dst	Сложение двух операндов с учётом переноса (BCD-арифметика)	src + dst + C → dst (BCD)	•	•	•	•
DEC(.B)* dst	Декремент операнда	dst – 1 → dst	•	•	•	•
DECD(.B)* dst	Уменьшение операнда на 2	dst – 2 → dst	•	•	•	•
DINT*	Запрещение прерываний	0 → GIE	–	–	–	–
EINT*	Разрешение прерываний	1 → GIE	–	–	–	–
INC(.B)* dst	Инкремент операнда	dst +1 → dst	•	•	•	•
INCD(.B)* dst	Увеличение операнда на 2	dst+2 → dst	•	•	•	•
INV(.B)* dst	Инвертирование операнда	.not.dst → dst	•	•	•	•
JC/JHS label	Переход, если бит C установлен / Переход, если «выше или равно»		–	–	–	–
JEQ/JZ label	Переход, если «равно» / Переход, если бит Z установлен		–	–	–	–
JGE label	Переход, если «больше или равно»		–	–	–	–
JL label	Переход, если «меньше»		–	–	–	–
JMP label	Безусловный переход	PC + 2 × смещение → PC	–	–	–	–
JN label	Переход, если бит N установлен		–	–	–	–
JNC/JLO label	Переход, если бит C не установлен / Переход, если «ниже»		–	–	–	–
JNE/JNZ label	Переход, если «не равно» / Переход, если бит Z не установлен		–	–	–	–
MOV(.B) src,dst	Пересылка операнда	src → dst	–	–	–	–
NOP*	Нет операции		–	–	–	–
POP(.B)* dst	Извлечение элемента из стека	@SP → dst, SP+2 → SP	–	–	–	–
PUSH(.B) src	Загрузка операнда в стек	SP – 2 → SP, src → @SP	–	–	–	–
RET*	Возврат из подпрограммы	@SP → PC, SP + 2 → SP	–	–	–	–
RETI	Возврат из прерывания		•	•	•	•
RLA(.B)* dst	Арифметический сдвиг влево		•	•	•	•
RLC(.B)* dst	Сдвиг влево через перенос		•	•	•	•
RRA(.B) dst	Арифметический сдвиг вправо		0	•	•	•
RRC(.B) dst	Сдвиг вправо через перенос		•	•	•	•
SBC(.B)* dst	Вычитание заёма из операнда	dst + 0FFFFh + C → dst	•	•	•	•
SETC*	Установка бита C	1 → C	–	–	–	1
SETN*	Установка бита N	1 → N	–	1	–	–
SETZ*	Установка бита Z	1 → Z	–	–	1	–
SUB(.B) src,dst	Вычитание операндов	dst + .not.src + 1 → dst	•	•	•	•
SUBC(.B) src,dst	Вычитание операндов с учётом заёма	dst + .not.src + C → dst	•	•	•	•
SWPB dst	Обмен байтов		–	–	–	–
SXT dst	Расширение знака		0	•	•	•
TST(.B)* dst	Проверка операнда (на ноль)	dst + 0FFFFh + 1	0	•	•	•
XOR(.B) src,dst	«Исключающее ИЛИ» операндов	src .xor. dst → dst	•	•	•	•

* Эмулируемые команды
• Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

ГЛАВА 4

16-БИТНОЕ RISC ЦПУ MSP430X

В этой главе описывается усовершенствованное ЦПУ MSP430X, позволяющее адресовать до 1 МБ памяти, его режимы адресации и набор команд. ЦПУ MSP430X реализовано во всех микроконтроллерах семейства, имеющих адресное пространство более 64 КБ.

4.1. Введение в ЦПУ

ЦПУ MSP430X обладает рядом возможностей, специально предназначенных для поддержки современных методов программирования, таких как вычисляемые переходы, табличные вычисления, а также использование языков высокого уровня, в частности языка Си. Центральный процессор MSP430X может адресовать до 1 МБ памяти без разбиения её на страницы. Кроме того, в ряде случаев ЦПУ MSP430X требуется меньше времени на обслуживание прерываний и выполнение команд, чем MSP430. ЦПУ MSP430X полностью обратно совместимо с ЦПУ MSP430.

ЦПУ MSP430X имеет следующие особенности:

- RISC-архитектура;
- ортогональная архитектура;
- полная доступность регистров, включая счётчик команд, регистры состояния и указатель стека;
- однотактные регистровые операции;
- большой регистровый файл, уменьшающий количество обращений к памяти;
- 20-битная шина адреса, обеспечивающая прямой доступ и ветвление во всём диапазоне адресов;
- 16-битная шина данных, позволяющая напрямую оперировать 2-байтными значениями;
- генератор констант формирует шесть наиболее часто используемых значений, уменьшая размер кода;
- прямой обмен данными между ячейками памяти без промежуточной записи в регистр;
- однобайтная, двухбайтная и 20-битная адресация.

Блок-схема ЦПУ MSP430X приведена на **Рис. 4.1**.

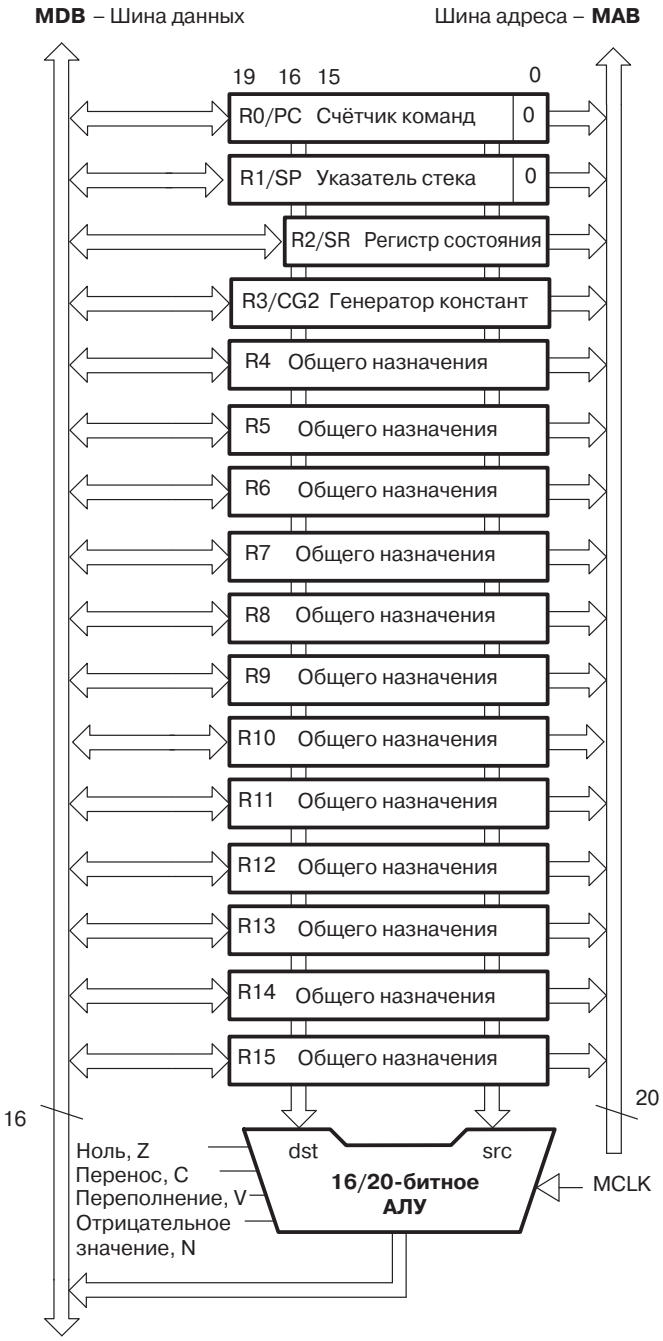


Рис. 4.1. Блок-схема ЦПУ MSP430X.

4.2. Прерывания

В ЦПУ MSP430X реализована такая же структура прерываний, что и в ЦПУ MSP430:

- векторные прерывания, не требующие программного опроса;
- таблица векторов располагается начиная с адреса 0FFFEh в сторону младших адресов.

Работа системы прерываний обоих ЦПУ описана во 2-й главе настоящей книги (раздел 2.2 «Прерывания»). В таблице векторов прерываний содержатся 16-битные адреса, указывающие на нижнюю область памяти объёмом 64 КБ. Это означает, что точки входа во все процедуры обработки прерываний должны находиться в этом диапазоне адресов — даже в моделях с ЦПУ MSP430X.

Во время прерывания счётчик команд PC и регистр состояния SR сохраняются в стеке, как показано на **Рис. 4.2**. Для эффективного сохранения 20-битного значения счётчика команд биты 19:16 этого регистра автоматически добавляются к содержимому регистра SR, сохранённому в стеке. При выполнении команды `RETI` восстанавливается полное 20-битное значение счётчика команд, обеспечивая возврат из обработчика прерывания по любому адресу из доступного диапазона адресов.

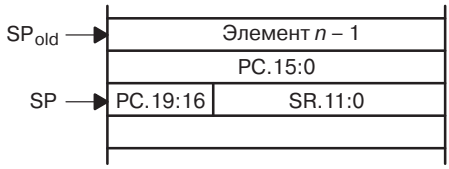


Рис. 4.2. Сохранение счётчика команд в стеке во время прерывания.

4.3. Регистры ЦПУ

Центральный процессор содержит шестнадцать регистров R0...R15. Регистры R0, R1, R2 и R3 имеют специальные функции. Регистры с R4 по R15 являются рабочими регистрами общего назначения.

4.3.1. Счётчик команд (PC)

20-битный счётчик команд (PC/R0) указывает на следующую команду, которая будет выполняться. Каждая команда занимает в памяти чётное число байтов (два, четыре, шесть или восемь), и на это же значение инкрементируется счётчик команд. Выборка команд осуществляется пословно, при этом счётчик команд указывает на чётные адреса. Формат счётчика команд PC приведён на **Рис. 4.3**.

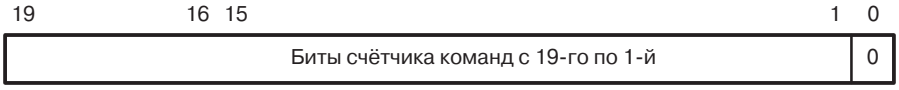


Рис. 4.3. Счётчик команд (PC).

Счётчик команд может быть адресован всеми командами с использованием любого режима адресации:

```
MOV.W  #LABEL,PC    ; Переход к адресу LABEL (младшие 64 КБ)
MOVA   #LABEL,PC    ; Переход к адресу LABEL (вся адресуемая память (1 Мб))
MOV.W  LABEL,PC     ; Переход к адресу, содержащемуся в LABEL (младшие 64 КБ)
MOV.W  @R14,PC      ; Косвенный переход по адресу, содержащемуся
                    ; в R14 (младшие 64 КБ)
ADDA   #4,PC        ; Пропуск двух слов (вся адресуемая память (1 МБ))
```

Команды BR и CALL обнуляют четыре старших бита счётчика команд. То есть эти команды позволяют выполнять переход только в пределах младших 64 КБ адресов. Адреса, расположенные вне этого диапазона, могут быть достигнуты только с помощью команд BRA или CALLA. Кроме того, любая команда, непосредственно модифицирующая PC, делает это в соответствии с используемым режимом адресации. Например, команда MOV.W #value,PC сбросит 4 старших бита счётчика команд, поскольку имеет суффикс .W.

Счётчик команд автоматически сохраняется в стеке при выполнении команд CALL или CALLA, а также при переходе к процедуре обработки прерывания. Содержимое стека с адресом возврата после выполнения команды CALLA показано на **Рис. 4.4**. Команда CALL сохраняет только биты 15:0 счётчика команд.

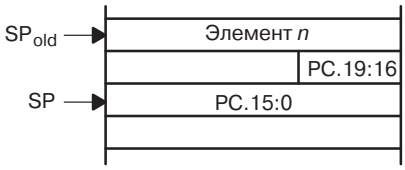


Рис. 4.4. Сохранение счётчика команд в стеке командой CALLA.

Команда RETA восстанавливает биты 19:0 счётчика команд и увеличивает на 4 указатель стека. Команда RET восстанавливает биты 15:0 счётчика команд и увеличивает указатель стека на 2.

4.3.2. Указатель стека (SP)

20-битный указатель стека (SP/R1) используется ЦПУ для сохранения адресов возврата из подпрограмм и прерываний. При этом он изменяется по преддекрементной/постинкрементной схеме. Кроме того, указатель стека может использоваться со всеми командами и любыми режимами адресации. Формат указателя стека SP приведён на **Рис. 4.5**. Указатель стека инициализируется (устанавливается на заданную ячейку ОЗУ) пользователем и выравнивается по чётным адресам.

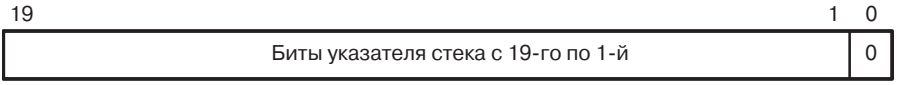


Рис. 4.5. Указатель стека.

Использование стека проиллюстрировано на **Рис. 4.6**. На **Рис. 4.7** показано использование стека при сохранении 20-битных адресов.

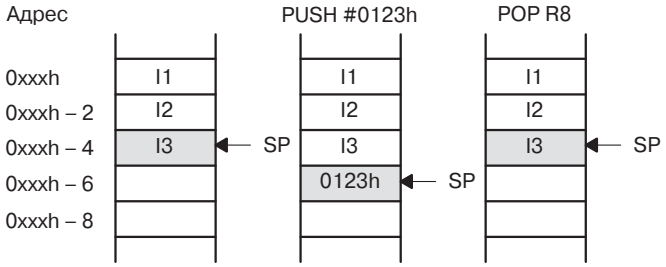


Рис. 4.6. Использование стека.

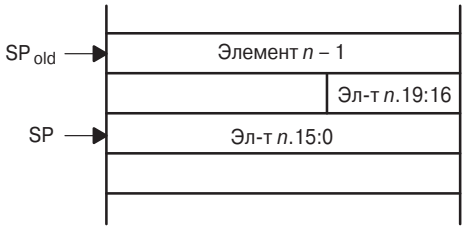


Рис. 4.7. Результат выполнения команды `PUSHX.A`.

```
MOV.W  2(SP),R6      ; Элемент стека I2 -> R6
MOV.W  R7,0(SP)      ; Перезаписать значение на вершине стека (TOS)
                     ; содержимым R7
PUSH   #0123h        ; Поместить число 0123h в стек
POP    R8            ; R8 = 0123h
```

К особым случаям относится использование указателя стека SP в качестве аргумента команд `PUSH` и `POP`. Эти ситуации показаны и объяснены на **Рис. 4.8**.

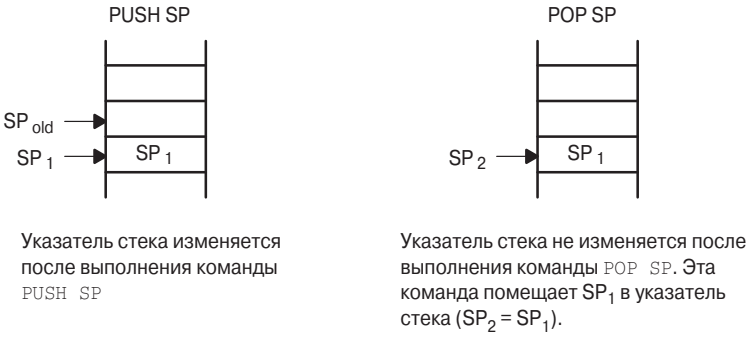


Рис. 4.8. Выполнение последовательности команд `PUSH SP` — `POP SP`.

4.3.3. Регистр состояния (SR)

16-битный регистр состояния (SR/R2) может выступать в качестве регистра-источника или регистра-приёмника только в командах, оперирующих двухбайтными значениями, и только при использовании режима регистровой адресации. Прочие комбинации режимов адресации используются для поддержки генератора констант. Формат регистра состояния SR приведён на **Рис. 4.9**. Нельзя записывать в регистр SR 20-байтные значения — результат этой операции будет непредсказуемым.

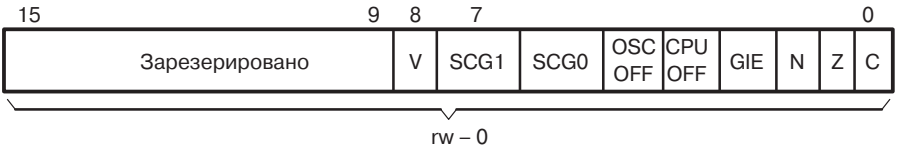


Рис. 4.9. Формат регистра состояния.

Назначение битов регистра SR описано в **Табл. 4.1**.

Таблица 4.1. **Биты регистра состояния**

Бит	Описание	
V	Флаг переполнения. Этот бит устанавливается, если результат арифметической операции выходит за границы допустимых значений для знаковых величин	
	`ADD(.B), ADDX(.B,.A), ADDC(.B), ADDCX(.B,.A), ADDA`	Устанавливается, когда: Положительное + Положительное = Отрицательное, Отрицательное + Отрицательное = Положительное, в противном случае сбрасывается
	`SUB(.B), SUBX(.B,.A), SUBC(.B),SUBCX(.B,.A), SUBA, CMP(.B), CMPX(.B,.A), CMPA`	Устанавливается, когда: Положительное – Отрицательное = Отрицательное, Отрицательное – Положительное = Положительное, в противном случае сбрасывается
SCG1	Системный тактовый генератор 1. Когда этот бит установлен, генератор DCO выключен, если сигнал DCOCLK не используется для формирования сигналов MCLK или SMCLK	
SCG0	Системный тактовый генератор 0. Когда этот бит установлен, выключается схема автоподстройки частоты модуля тактирования FLL+	
OSCOFF	Выключение генератора. Когда этот бит установлен, кварцевый генератор LFXT1 выключен, если сигнал LFXT1CLK не используется для формирования сигналов MCLK или SMCLK	
CPUOFF	Выключение ЦПУ. Когда этот бит установлен, ЦПУ выключено	
GIE	Общий бит разрешения прерываний. Когда этот бит установлен, маскируемые прерывания разрешены. После сброса все маскируемые прерывания запрещены	
N	Флаг отрицательного результата. Этот бит устанавливается, если результат арифметической операции отрицателен, и сбрасывается, если результат положителен	
Z	Флаг нуля. Этот бит устанавливается, если результат арифметической операции равен нулю, и сбрасывается в противном случае	
C	Флаг переноса. Этот бит устанавливается, если при выполнении арифметической операции возникает перенос, и сбрасывается, если переноса не возникает	

4.3.4. Регистры генератора констант CG1 и CG2

С помощью регистров генератора констант R2 и R3 генерируются шесть часто используемых значений. Выбор конкретной константы осуществляется изменением режима адресации регистра-источника (As) в соответствии с **Табл. 4.2**.

Таблица 4.2. **Значения, формируемые генератором констант**

Регистр	As	Константа	Примечания
R2	00	—	Регистровый режим адресации
R2	01	(0)	Абсолютный режим адресации
R2	10	00004h	+4, операции над битами
R2	11	00008h	+8, операции над битами
R3	00	00000h	0, операции над словами
R3	01	00001h	+1
R3	10	00002h	+2, операции над битами
R3	11	FFh, FFFFh, FFFFFh	–1, операции над словами

Преимущества генератора констант:

- не требуются особые команды;
- не требуется дополнительное слово памяти программ для хранения константы;
- не требуется обращаться к памяти программ для загрузки константы.

Ассемблер автоматически использует генератор констант, если любая из шести указанных констант служит в качестве непосредственного операнда-источника. Если регистры R2 и R3 используются в режиме генерации констант, их нельзя адресовать явно — они выступают только в качестве источников.

Генератор констант — расширение набора команд

Система команд MSP430 содержит всего 27 инструкций. Однако наличие генератора констант позволяет ассемблеру MSP430 использовать 24 дополнительных (эмулируемых) команд. К примеру, однооперандная команда:

```
CLR     dst
```

эмулируется двухоперандной командой, занимающей в памяти столько же места:

```
MOV     R3,dst
```

в которой ассемблер заменил константу #0 обращением к регистру R3 с As = 00.

Команда

```
INC     dst
```

заменяется командой

```
ADD     0(R3),dst
```

4.3.5. Регистры общего назначения R4…R15

Двенадцать регистров с R4 по R15 могут содержать 8-, 16- и 20-битные значения. При записи в регистр однобайтного значения сбрасываются биты 19:8 регистра. При записи 2-байтного значения — сбрасываются биты 19:16 регистра. Единственным исключением является команда `SXT`, которая выполняет расширение знака на все биты 20-битного регистра.

На следующих рисунках показано, каким образом осуществляется манипулирование 8-, 16- и 20-битными данными. Обратите внимание, что при использовании регистра в качестве операнда-приёмника команды, работающей с одно- и двухбайтными операндами, неиспользуемые старшие биты регистра всегда сбрасываются.

На **Рис. 4.10** показано оперирование 8-битными значениями (команды с суффиксом .B). В первом примере операндом-источником является регистр, а операндом-приёмником — байт в памяти. Во втором примере изображена обратная ситуация.

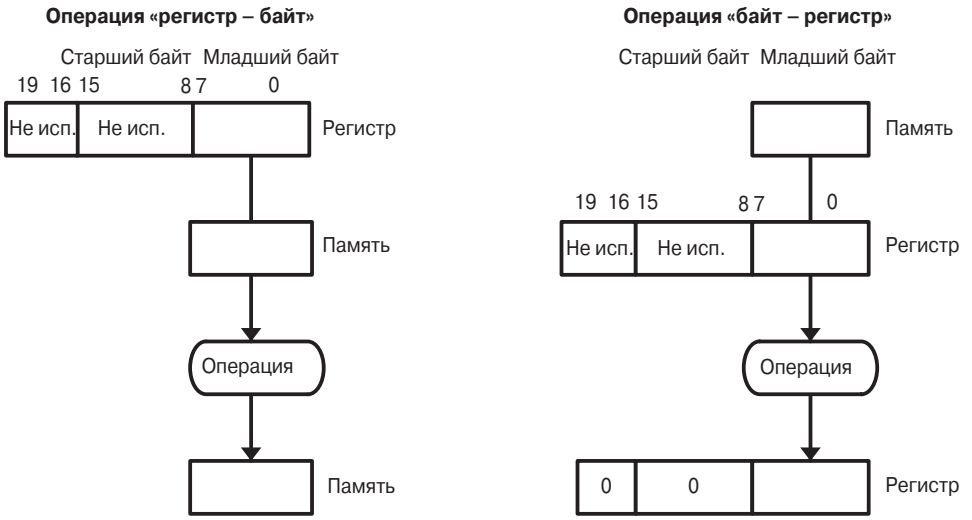


Рис. 4.10. Операции «регистр — память» и «память — регистр» с 8-битными данными.

На **Рис. 4.11** и **Рис. 4.12** показано оперирование 16-битными значениями (команды с суффиксом .W). На первом рисунке операндом-источником является регистр, а операндом-приёмником — слово в памяти. На втором рисунке изображена обратная ситуация.

На **Рис. 4.13** и **Рис. 4.14** показано оперирование 20-битными значениями (команды с суффиксом .A). На первом рисунке операндом-источником является регистр, а операндом-приёмником — 20-битное значение в памяти. На втором рисунке изображена обратная ситуация.

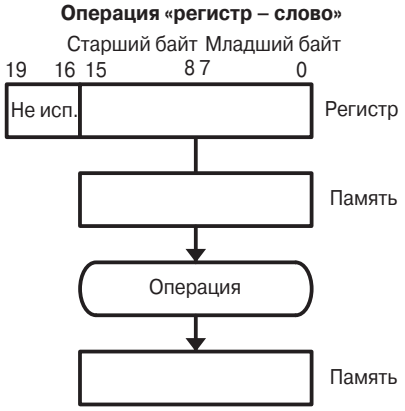


Рис. 4.11. Операция «регистр — память» с 16-битными данными.

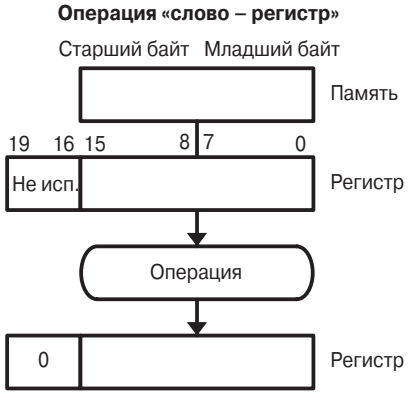


Рис. 4.12. Операция «память — регистр» с 16-битными данными.

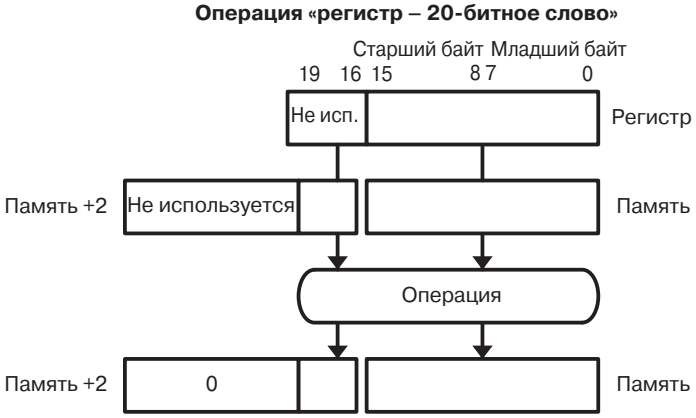


Рис. 4.13. Операция «регистр — память» с 20-битными данными.

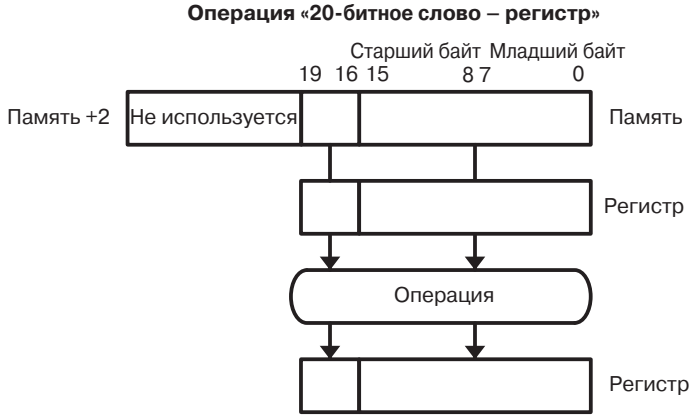


Рис. 4.14. Операция «память — регистр» с 20-битными данными.

4.4. Режимы адресации

ЦПУ MSP430X поддерживает семь режимов адресации для операндов-источников и четыре режима — для операндов-приёмников с использованием 16- или 20-битных адресов. Команды MSP430 и MSP430X могут применяться в пределах всего адресного пространства размером 1 МБ.

В **Табл. 4.3** приводятся значения битов режима As (для операнда-источника) и Ad (для операнда-приёмника), соответствующие тому или иному режиму адресации.

Таблица 4.3. **Режимы адресации операндов**

As/Ad	Режим адресации	Синтаксис	Описание
00/0	Регистровый	Rn	Содержимое регистра является операндом
01/1	Индексный	X(Rn)	(Rn + X) указывает на операнд. Значение X содержится в следующем слове
01/1	Относительный	ADDR	(PC + X) указывает на операнд. Значение X содержится в следующем слове. Используется индексный режим адресации X(PC)
01/1	Абсолютный	&ADDR	Слово, следующее за командой, содержит абсолютный адрес. Значение X содержится в следующем слове. Используется индексный режим адресации X(SR)
10/—	Косвенный регистровый	@Rn	Rn используется в качестве указателя на операнд.
11/—	Косвенный регистровый с автоинкрементом	@Rn+	Rn используется в качестве указателя на операнд. После выполнения операции значение регистра Rn увеличивается на 1 для команд с суффиксом .B, на 2 для команд с суффиксом .W и на 4 для команд с суффиксом .A
11/—	Непосредственный	#N	Слово, следующее за командой, содержит значение константы N. Используется косвенный режим адресации с автоинкрементом @PC+

Более подробно эти режимы адресации рассматриваются в последующих разделах. В большинстве примеров для источника и приёмника используется один и тот же режим адресации, однако в командах можно использовать любые допустимые комбинации режимов адресации источника и приёмника.

Примечание. Использование меток EDE, TONI, TOM и LEO

В документации на семейство MSP430 аббревиатуры EDE, TONI, TOM и LEO используются в качестве обычных меток. Они не имеют никакого специального значения.

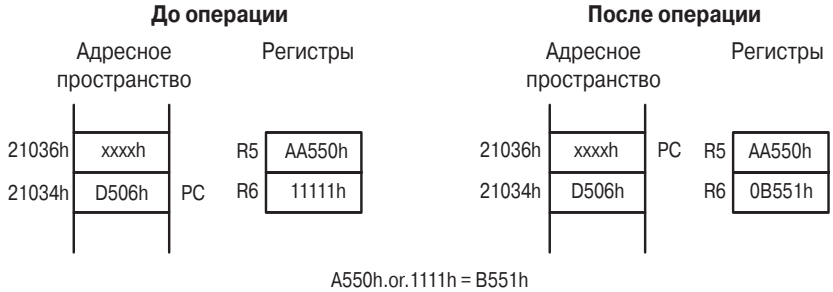
4.4.1. Регистровый режим

Операция:	Операндом является 8-, 16- или 20-битное содержимое указанного регистра ЦПУ.
Длина:	Одно, два или три слова.
Комментарий:	Допускается для источника и приёмника.
Операция с байтами:	При однобайтной операции из регистра-источника Rsrc считывается только 8 младших битов, а результат операции помещается в 8 младших битов регистра-приёмника Rdst. Биты 19:8 регистра Rdst сбрасываются. Регистр Rsrc не изменяется.
Операция с двухбайтными словами:	При двухбайтной операции из регистра-источника Rsrc считывается только 16 младших битов, а результат операции помещается в 16 младших битов регистра-приёмника Rdst. Биты 19:16 регистра Rdst сбрасываются. Регистр Rsrc не изменяется.
Операция с 20-битными словами:	При операции с 20-битными словами из регистра-источника Rsrc считываются все 20 бит, а результат операции помещается в регистр-приёмник Rdst. Регистр Rsrc не изменяется.
Команда `SXT`:	Команда `SXT` является единственным исключением — знак младшего байта, находящийся в 7-м бите регистра, копируется и в биты 19:8 регистра Rdst.

Пример 1

```
BIS.W   R5,R6
```

Эта команда выполняет операцию «Логическое ИЛИ» между 16-битным значением, находящемся в регистре R5, и 16-битным содержимым регистра R6. Биты 19:16 регистра R6 сбрасываются.

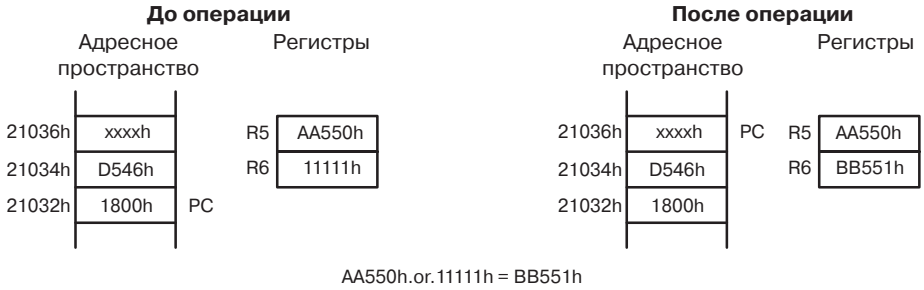


Пример 2

```
BISX.A   R5,R6
```

Эта команда выполняет операцию «Логическое ИЛИ» между 20-битным значением, находящемся в регистре R5, и 20-битным значением, находящемся в регистре R6.

Слово расширения содержит бит A/L для поддержки 20-битных данных. Слово команды соответствует однобайтной операции, при этом биты A/L:B/W = 01. Результат выполнения команды будет следующим:



4.4.2. Индексный режим адресации

При использовании индексного режима адрес операнда вычисляется как сумма смещения (число со знаком) и содержимого регистра ЦПУ. Существует три разновидности индексного режима:

- индексный режим при адресации нижней области памяти объёмом 64 КБ;
- индексный режим при адресации командами MSP430 памяти за пределами нижней 64-КБ области;
- индексный режим в командах MSP430X.

Индексный режим адресации нижней 64-КБ области памяти

Если регистр ЦПУ Rn указывает на адрес в нижней 64-КБ области памяти, то биты 19:16 итогового адреса сбрасываются после сложения содержимого регистра ЦПУ с индексом, представляющим собой 16-битное число со знаком. Это означает, что итоговый адрес всегда будет находиться в пределах нижней 64-КБ области, не выходя за её границы. Данный режим адресации позволяет обращаться к ОЗУ и периферийным устройствам микроконтроллера, а также использовать без всякой модификации существующее ПО, написанное для ЦПУ MSP430 (**Рис. 4.15**).

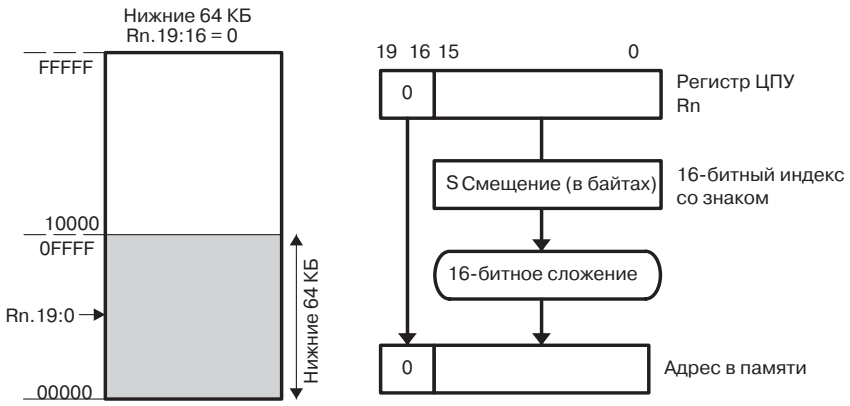


Рис. 4.15. Индексный режим адресации в нижней 64-КБ области памяти.

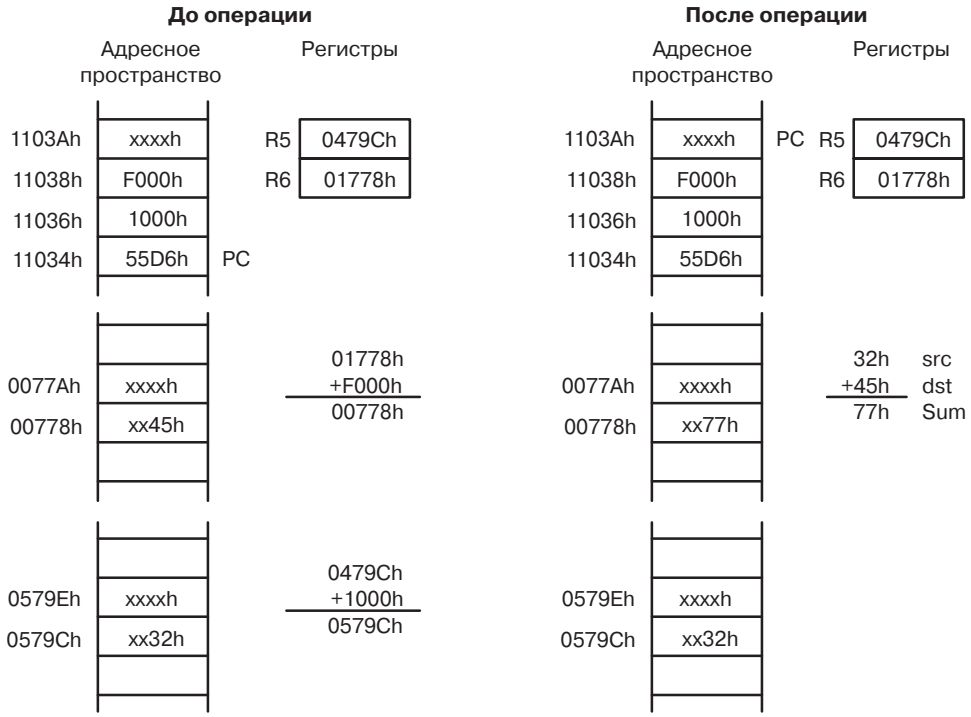
Длина:	Два или три слова.
Операция:	Индекс (смещение), представляющий собой 16-битное число со знаком и находящийся в следующем после команды слове, прибавляется к содержимому регистра ЦПУ Rn. Биты 19:16 результата обнуляются, формируя 16-битный адрес из диапазона 00000h...0FFFFh. Операндом является содержимое ячейки памяти по данному адресу.
Комментарий:	Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение и вставляет его в итоговый код.

Пример

```
ADD.B   1000h(R5),0F000h(R6)
```

Эта команда выполняет сложение 8-битного значения операнда-источника, расположенного по адресу 1000h(R5), с содержимым операнда-приёмника, расположенного по адресу 0F000h(R6). Результат сложения сохраняется в операнде-приёмнике. Оба операнда расположены в нижней 64-КБ области памяти, поскольку биты 19:16 регистров R5 и R6 сброшены.

Источник:	Байт с адресом R5 + 1000h, равным 0479Ch + 1000h = 0579Ch (после усечения до 16-битного значения).
Приёмник:	Байт с адресом R6 + F000h, равным 01778h + F000h = 00778h (после усечения до 16-битного значения).



Команды MSP430 и индексный режим адресации верхней области памяти

Если регистр ЦПУ Rn указывает на адрес, расположенный за пределами нижней 64-КБ области памяти, то для вычисления адреса операнда используются биты 19:16 регистра. Таким образом, операнд может быть расположен в диапазоне адресов Rn ±32 КБ, поскольку индекс X является 16-битным числом со знаком. При этом итоговый адрес может выйти за пределы адресного пространства ЦПУ и в результате отобразиться на нижнюю 64-КБ область памяти (см. **Рис. 4.16** и **Рис. 4.17**).

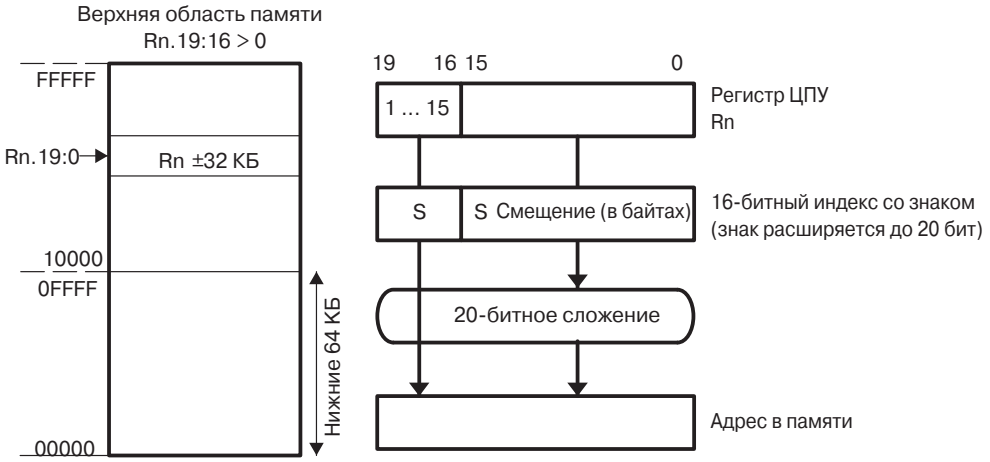


Рис. 4.16. Индексный режим адресации верхней области памяти.

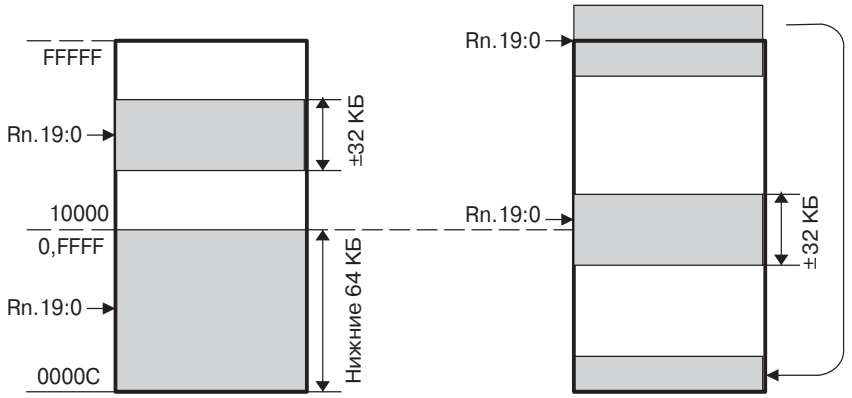


Рис. 4.17. Выход за границы адресного пространства при индексном режиме адресации.

Длина: Два или три слова.

Операция: Индекс (смещение), представляющий собой 16-битное число со знаком и находящийся в следующем после команды слове, прибавляется к 20-битному содержимому регистра ЦПУ Rn. В результате формируется 20-битный адрес из диапазона 0...FFFFFh. Операндом является содержимое ячейки памяти по данному адресу.

Комментарий: Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение и вставляет его в итоговый код.

Пример

```
ADD.W   8346h(R5),2100h(R6)
```

Эта команда выполняет сложение 16-битных значений операнда-источника и операнда-приёмника, расположенных по адресам, указанным в команде. 16-битный результат сложения сохраняется в операнде-приёмнике. Оба операнда могут быть расположены по любому адресу из допустимого диапазона.

Источник: Слово с адресом R5 + 8346h. Отрицательное 16-битное смещение 8346h преобразуется к 20-битному значению, в результате чего адрес операнда получается равным 23456h + F8346h = 1B79Ch.

Приёмник: Слово с адресом R6 + 2100h, равным 15678h + 2100h = 17778h.

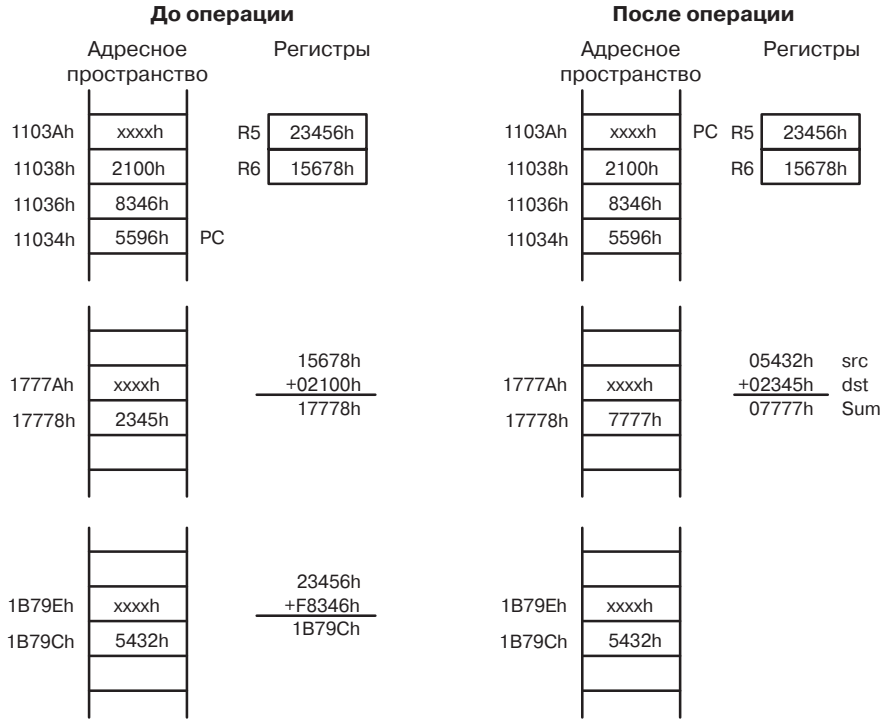


Рис. 4.18. Пример использования индексного режима адресации.

Команды MSP430X и индексный режим адресации

При использовании индексного режима адресации с командами MSP430X, операнды могут быть расположены по любому адресу в диапазоне Rn ± 19 бит.

Длина:	Три или четыре слова.
Операция:	Адрес операнда вычисляется как сумма 20-битного содержимого регистра ЦПУ и 20-битного индекса. Четыре старших бита индекса хранятся в слове расширения, а остальные 16 битов — в слове, расположенном после команды. Содержимое регистра не изменяется.
Комментарий:	Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение и вставляет его в итоговый код.

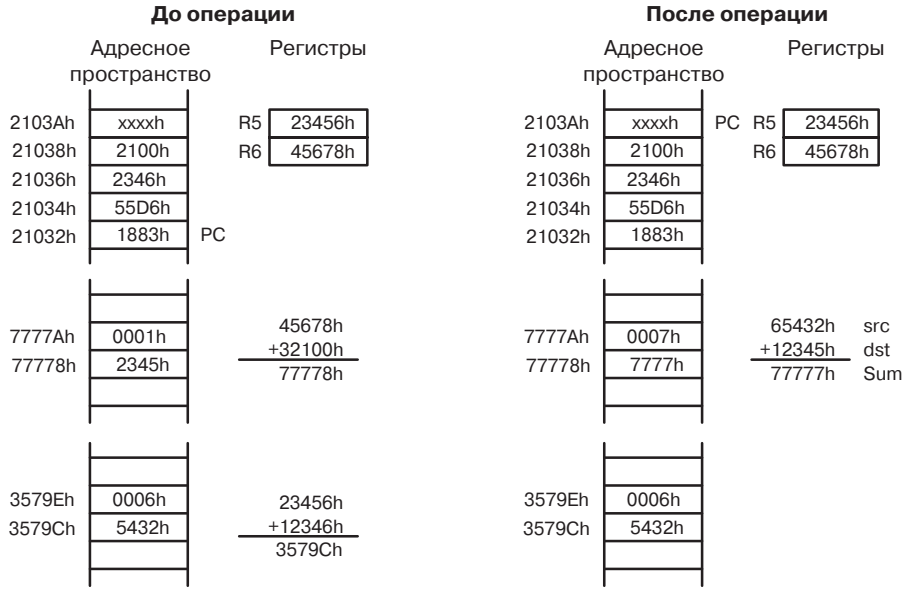
Пример

```
ADDX.A   12346h(R5),32100h(R6)
```

Эта команда выполняет сложение 20-битных значений операнда-источника и операнда-приёмника, расположенных по адресам, указанным в команде. Результат сложения сохраняется в операнде-приёмнике.

Источник:	Два слова, первое из которых расположено по адресу R5 + 12346h. То есть адрес операнда равен 23456h + 12346h = 3579Ch.
Приёмник:	Два слова, первое из которых расположено по адресу R6 + 32100h. То есть адрес операнда равен 45678h + 32100h = 77778h.

В слове расширения содержатся старшие биты значений смещения источника и приёмника и бит A/L, определяющий разрядность данных. Поскольку используются 20-битные значения, слово команды соответствует однобайтной операции, при этом биты A/L:B/W = 01.



4.4.3. Относительный режим адресации

При использовании относительного режима адрес операнда вычисляется как сумма смещения (число со знаком) и текущего значения счётчика команд. Существует три разновидности относительного режима:

- относительный режим при адресации нижней области памяти объёмом 64 КБ;
- относительный режим при адресации командами MSP430 памяти за пределами нижней 64-КБ области;
- относительный режим в командах MSP430X.

Относительный режим адресации в нижней 64-КБ области памяти

Если счётчик команд PC указывает на адрес в нижней 64-КБ области памяти, то биты 19:16 итогового адреса сбрасываются после сложения содержимого PC с индексом, представляющим собой 16-битное число со знаком. Это означает, что итоговый адрес всегда будет находиться в пределах нижней 64-КБ области, не выходя за её границы. Данный режим адресации позволяет обращаться к ОЗУ и периферийным устройствам микроконтроллера, а также использовать без всякой модификации существующее ПО, написанное для ЦПУ MSP430 (**Рис. 4.19**).

Операция:	Индекс (смещение), представляющий собой 16-битное число со знаком и находящийся в следующем после команды слове, прибавляется к текущему значению счётчика команд PC. Биты 19:16 результата обнуляются, формируя 16-битный адрес из диапазона 00000h...0FFFFh. Операндом является содержимое ячейки памяти по данному адресу.
Длина:	Два или три слова.
Комментарий:	Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение и вставляет его в итоговый код.

Пример

```
ADD.B   EDE,TONI
```

Эта команда выполняет сложение 8-битного значения операнда-источника EDE с содержимым операнда-приёмника TONI. Результат сложения сохраняется в операнде-приёмнике. Оба операнда расположены в нижней 64-КБ области памяти.

Источник:	Для адресации байта EDE, расположенного по адресу 0579Ch, используется выражение PC + 04766h. Значение смещения определяется выражением 0579Ch – 01036h = 04766h, где 01036h — адрес, по которому хранится смещение в данном примере.
Приёмник:	Для адресации байта TONI, расположенного по адресу 00778h, используется выражение PC + F740h. Значение смещения определяется выражением 00778h – 01038h = FF740h (усечено до 16 бит), где 01038h — адрес, по которому хранится смещение в данном примере.

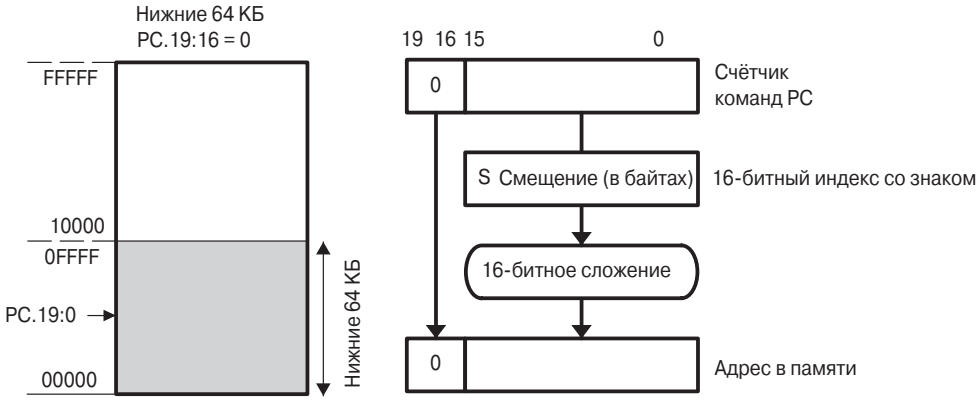
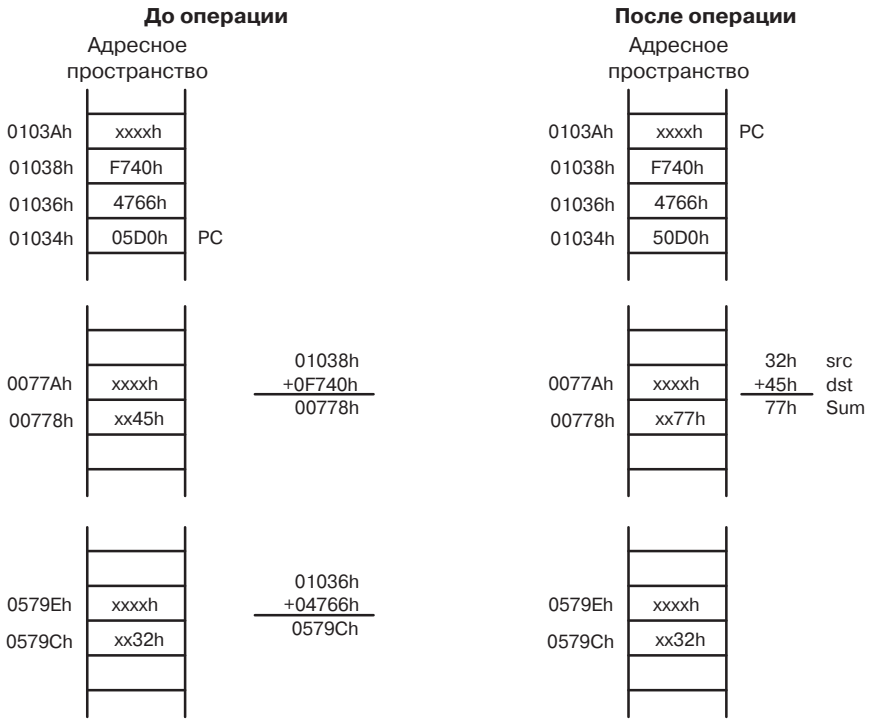


Рис. 4.19. Относительный режим адресации в нижней 64-КБ области памяти.



Команды MSP430 и относительный режим адресации верхней области памяти

Если счётчик команд указывает на адрес, расположенный за пределами нижней 64-КБ области памяти, то для вычисления адреса операнда используются биты 19:16 этого регистра. Таким образом, операнд может быть расположен в диапазоне адресов PC ±32 КБ, поскольку индекс X является 16-битным числом со знаком. При этом итоговый адрес может выйти за пределы адресного пространства ЦПУ и в результате отобразиться на нижнюю 64-КБ область памяти (см. **Рис. 4.20** и **Рис. 4.21**).

Длина: Два или три слова.

Операция: Индекс (смещение), представляющий 16-битное число со знаком и находящийся в следующем после команды слове, прибавляется к 20-битному содержимому счётчика команд PC. В результате формируется 20-битный адрес из диапазона 0...FFFFFh. Операндом является содержимое ячейки памяти по данному адресу.

Комментарий: Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение и вставляет его в итоговый код.

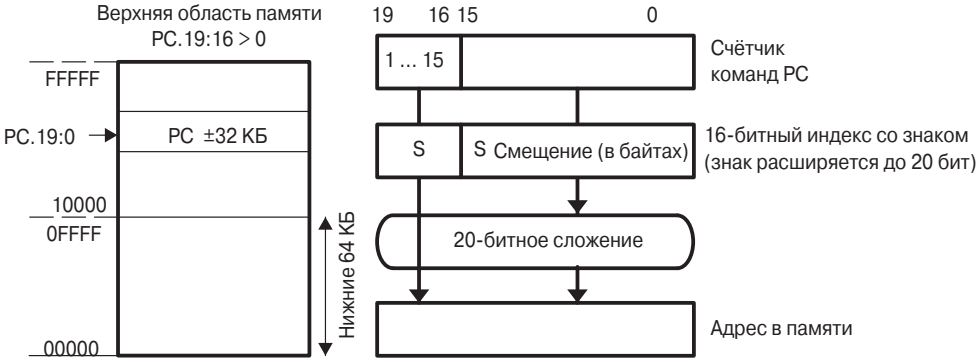


Рис. 4.20. Относительный режим адресации верхней области памяти.

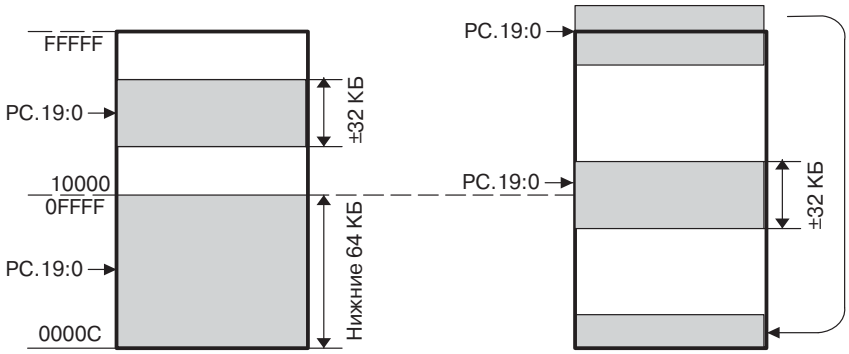


Рис. 4.21. Выход за границы адресного пространства при использовании относительного режима адресации.

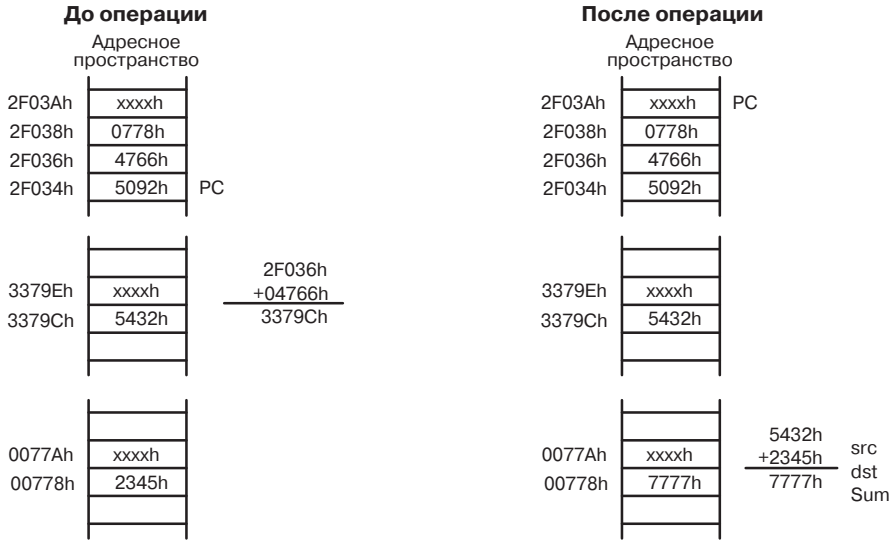
Пример

```
ADD.W   EDE,&TONI
```

Эта команда выполняет сложение 16-битных значений операнда-источника EDE и операнда-приёмника TONI. 16-битный результат сложения сохраняется в операнде-приёмнике. В данном примере команда расположена по адресу 2F034h.

Источник: Для адресации слова EDE, расположенного по адресу 3379Ch, используется выражение PC + 04766h. Значение смещения определяется выражением 3379Ch – 2F036h = 04766h, где 2F036h — адрес, по которому хранится смещение в данном примере.

Приёмник: Для адресации слова TONI используется его абсолютный адрес 00778h.



Команды MSP430X и относительный режим адресации

При использовании относительного режима адресации с командами MSP430X, операнды могут быть расположены по любому адресу в диапазоне PC ± 19 бит.

Длина: Три или четыре слова.

Операция: Адрес операнда вычисляется как сумма 20-битного значения счётчика команд и 20-битного индекса. Четыре старших бита индекса хранятся в слове расширения, а остальные 16 битов — в слове, расположенном после команды.

Комментарий: Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение и вставляет его в итоговый код.

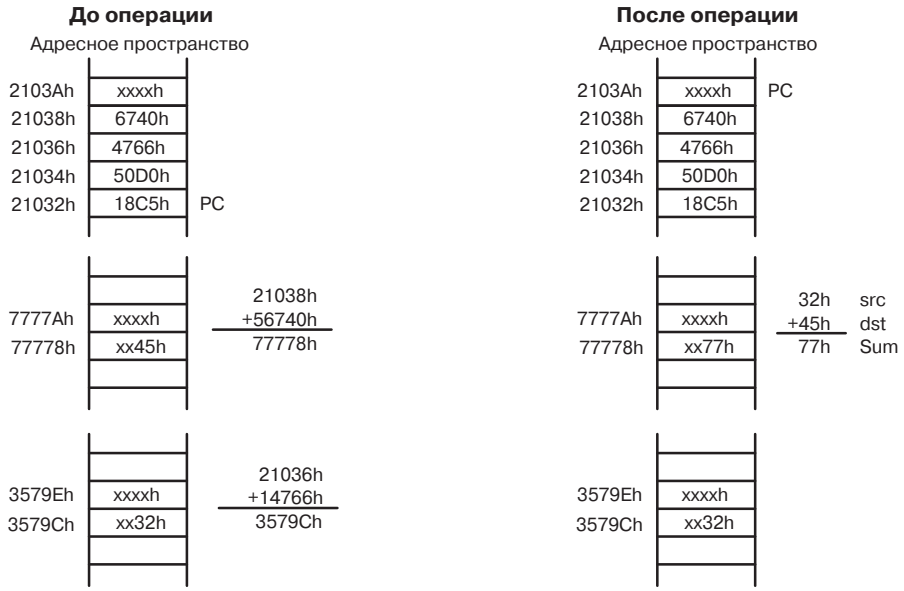
Пример

```
ADDX.B   EDE,TONI
```

Эта команда выполняет сложение 8-битных значений операнда-источника EDE и операнда-приёмника TONI. Результат сложения сохраняется в операнде-приёмнике.

Источник: Для адресации байта EDE, расположенного по адресу 3579Ch, используется выражение PC + 14766h. Значение смещения определяется выражением 3579Ch – 21036h = 14766h, где 21036h — адрес, по которому хранится смещение в данном примере.

Приёмник: Для адресации байта TONI, расположенного по адресу 77778h, используется выражение PC + 56740h. Значение смещения определяется выражением 77778h – 21038h = 56740h, где 21038h — адрес, по которому хранится смещение в данном примере.



4.4.4. Абсолютный режим адресации

При использовании абсолютного режима в качестве адреса операнда используется значение, расположенное после слова команды. Существует две разновидности абсолютного режима:

- абсолютный режим при адресации нижней области памяти объёмом 64 КБ;
- абсолютный режим в командах MSP430X.

Абсолютный режим адресации в нижней 64-КБ области памяти

Если абсолютный режим адресации используется с командой MSP430, то абсолютный адрес представляет собой 16-битное значение и, соответственно, указывает на адрес в нижней 64-КБ области памяти. Адрес вычисляется как смещение от нуля и сохраняется в следующем после команды слове. Этот режим адресации позволяет обращаться к ОЗУ и периферийным устройствам микроконтроллера, а также использовать без всякой модификации существующее ПО, написанное для ЦПУ MSP430.

Длина: Два или три слова.

Операция: Операндом является содержимое ячейки памяти по заданному адресу.

Комментарий: Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение от 0-го адреса и вставляет его в итоговый код.

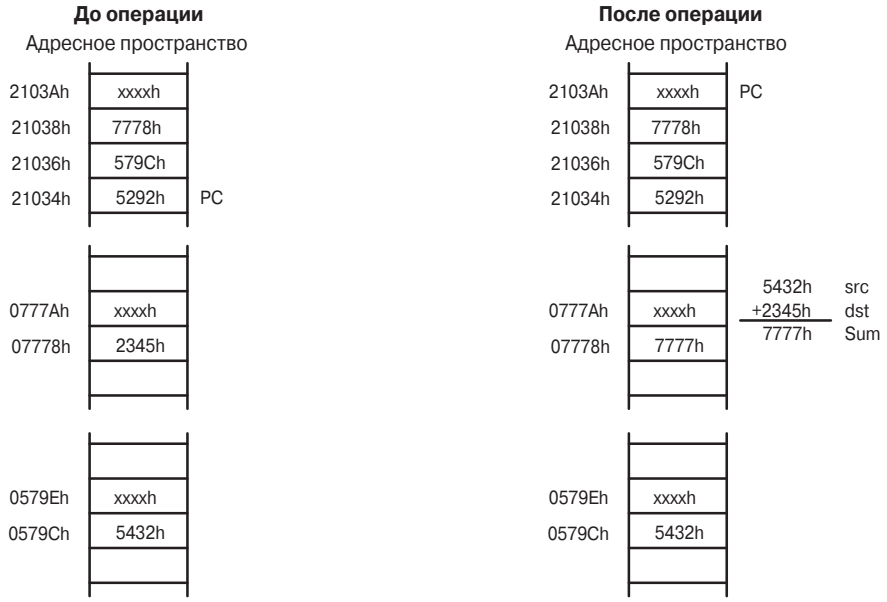
Пример

```
ADD.W   &EDE,&TONI
```

Эта команда выполняет сложение 16-битных значений операнда-источника и операнда-приёмника, расположенных по абсолютным адресам EDE и TONI соответственно. Результат сложения сохраняется в операнде-приёмнике.

Источник: Слово, расположенное в памяти по адресу EDE.

Приёмник: Слово, расположенное в памяти по адресу TONI.



Команды MSP430X и абсолютный режим адресации

При использовании абсолютного режима адресации с командой MSP430X, адрес представляет собой 20-битное значение и, соответственно, позволяет адресовать любую ячейку памяти в пределах всего адресного пространства. Адрес вычисляется как смещение от нуля. Четыре старших бита значения хранятся в слове расширения, а остальные 16 бит — в слове, расположенном после команды.

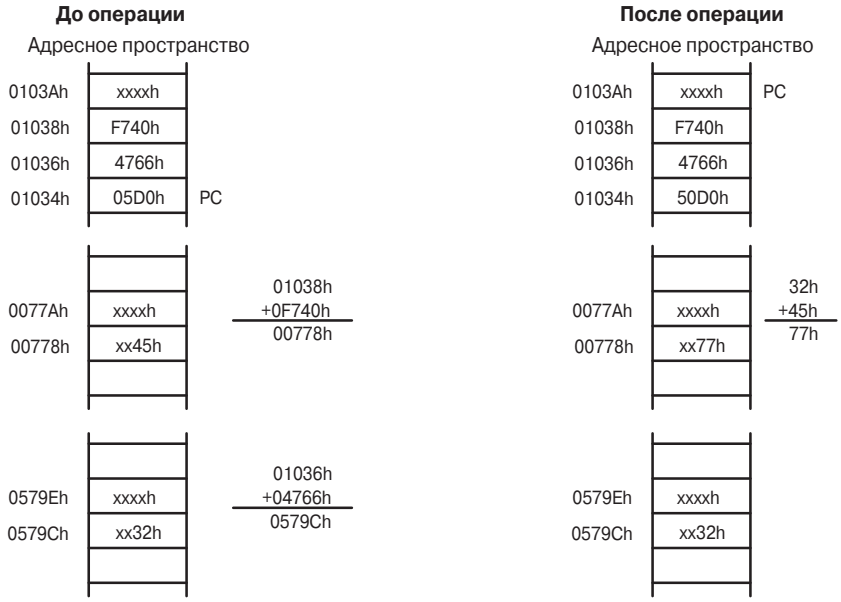
Длина:	Три или четыре слова.
Операция:	Операндом является содержимое ячейки памяти по заданному адресу.
Комментарий:	Допускается для источника и приёмника. Ассемблер автоматически вычисляет смещение от 0-го адреса и вставляет его в итоговый код.

Пример

```
ADDX.A   &EDE,&TONI
```

Эта команда выполняет сложение 20-битных значений операнда-источника и операнда-приёмника, расположенных по абсолютным адресам EDE и TONI соответственно. Результат сложения сохраняется в операнде-приёмнике.

Источник:	Два слова, расположенных в памяти, начиная с адреса EDE.
Приёмник:	Два слова, расположенных в памяти, начиная с адреса TONI.



4.4.5. Косвенный регистровый режим адресации

При использовании косвенного регистрового режима в качестве адреса операнда используется содержимое регистра ЦПУ Rsrc. В этом режиме всегда используются 20-битные адреса.

Длина: Один, два или три слова.

Операция: Операндом является содержимое ячейки памяти по заданному адресу. Содержимое регистра-источника Rsrc не изменяется.

Комментарий: Допускается только для источника. Для операнда-приёмника подставляется 0(Rdst).

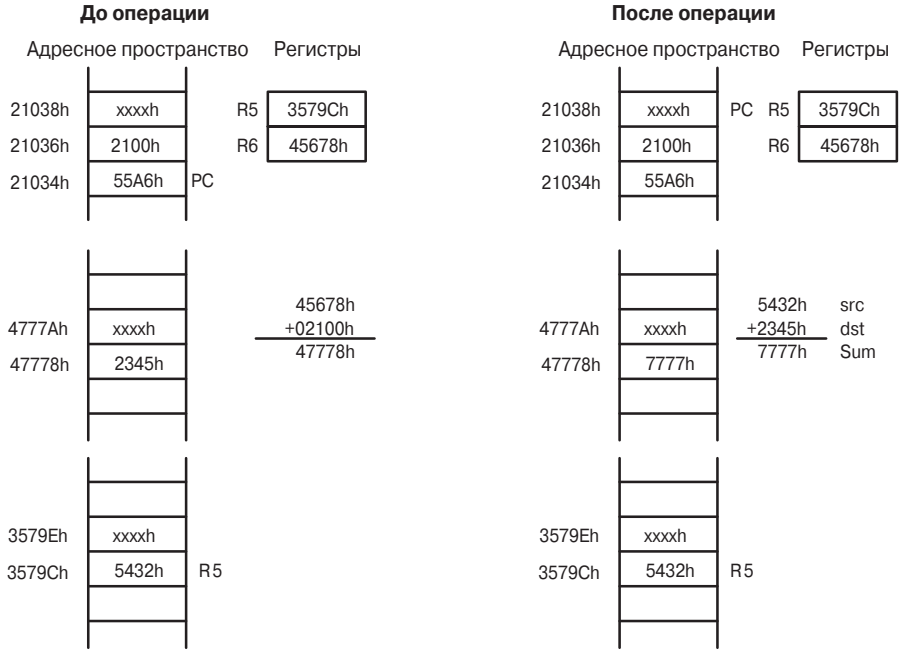
Пример

```
ADDX.W   @R5,2100h(R6)
```

Эта команда выполняет сложение 16-битных значений, расположенных по указанным адресам. Результат сложения сохраняется в операнде-приёмнике.

Источник: Слово, адресуемое регистром R5. В данном примере в регистре содержится адрес 3579Ch.

Приёмник: Слово с адресом R6 + 2100h, равным 45678h + 2100h = 47778h.



4.4.6. Косвенный регистровый режим адресации с автоинкрементом

При использовании косвенного регистрового режима с автоинкрементом в качестве адреса операнда используется содержимое регистра ЦПУ Rsrc. После выборки операнда-источника содержимое регистра Rsrc автоматически увеличивается на 1, 2 или 4 при использовании 8-, 16- и 20-битных операндов соответственно. Если один и тот же регистр задействуется как источник и как приёмник, то для адресации операнда-приёмника используется содержимое регистра после инкрементирования. В косвенном регистровом режиме с автоинкрементом всегда применяются 20-битные адреса.

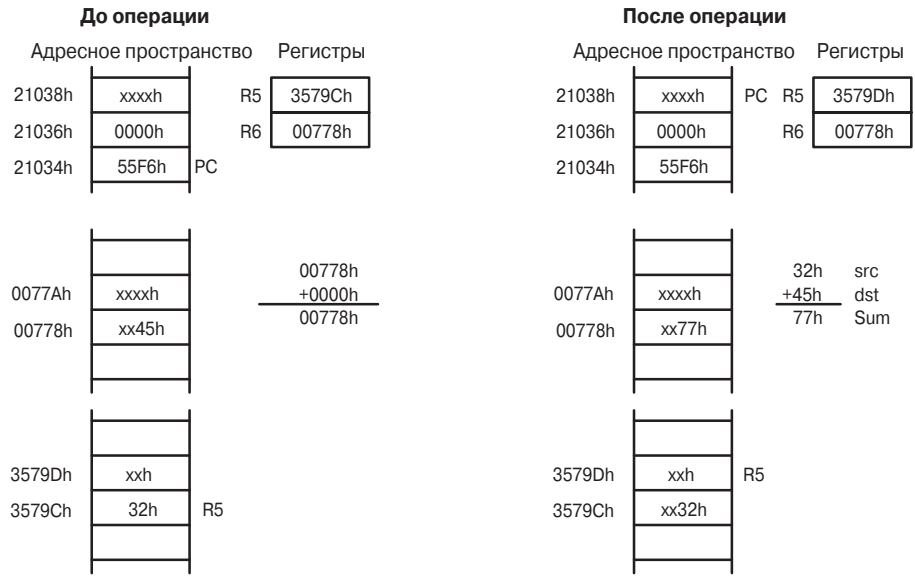
Длина:	Один, два или три слова.
Операция:	Операндом является содержимое ячейки памяти по заданному адресу.
Комментарий:	Допускается только для источника.

Пример

```
ADD.B   @R5+,0(R6)
```

Эта команда выполняет сложение 8-битных значений, расположенных по указанным адресам. Результат сложения сохраняется в операнде-приёмнике.

Источник:	Байт, адресуемый регистром R5. В данном примере в регистре содержится адрес 3579Ch.
Приёмник:	Байт с адресом R6 + 0h, равным в данном примере 00778h.



4.4.7. Непосредственный режим адресации

Непосредственный режим позволяет использовать в качестве операндов константы, которые при этом располагаются в памяти сразу после команды. Обращение к счётчику команд осуществляется с использованием косвенного режима адресации с автоинкрементом. Счётчик указывает на непосредственное значение, находящееся в следующем слове. После выборки этого значения PC увеличивается на 2 независимо от разрядности операнда. Существует две разновидности непосредственного режима:

- 8- или 16-битные константы в командах MSP430;
- 20-битные константы в командах MSP430X.

Команды MSP430 и непосредственный режим адресации

При использовании непосредственного режима адресации в команде MSP430, константа представляет собой 8- или 16-битное число и располагается в слове памяти, следующем за командой.

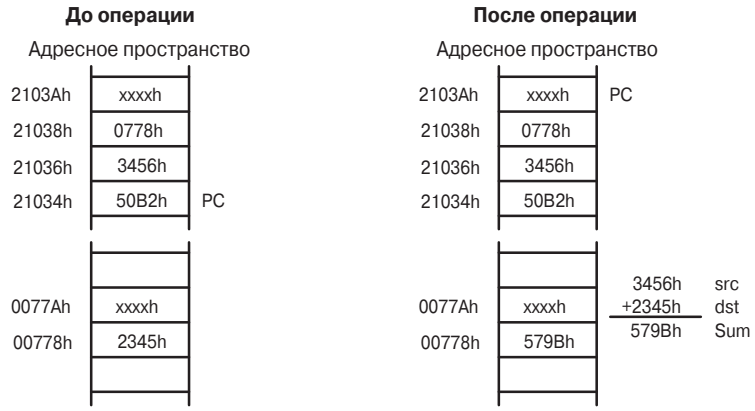
Длина:	Два или три слова. На одно слово меньше, если в качестве непосредственного операнда может использоваться значение, формируемое генератором констант.
Операция:	Операнд-источник, являющийся 16-битной константой, используется вместе с 16-битным операндом-приёмником.
Комментарий:	Допускается только для источника.

Пример

```
ADD   #3456h,&TONI
```

Эта команда прибавляет 16-битное число 3456h к содержимому операнда-приёмника с адресом TONI.

Источник:	16-битная константа 3456h.
Приёмник:	Слово, расположенное по адресу TONI.



Команды MSP430X и непосредственный режим адресации

При использовании непосредственного режима адресации в команде MSP430X, константа представляет собой 20-битное число. Старшие четыре бита значения хранятся в слове расширения, а остальные 16 бит — в слове, расположенном после команды.

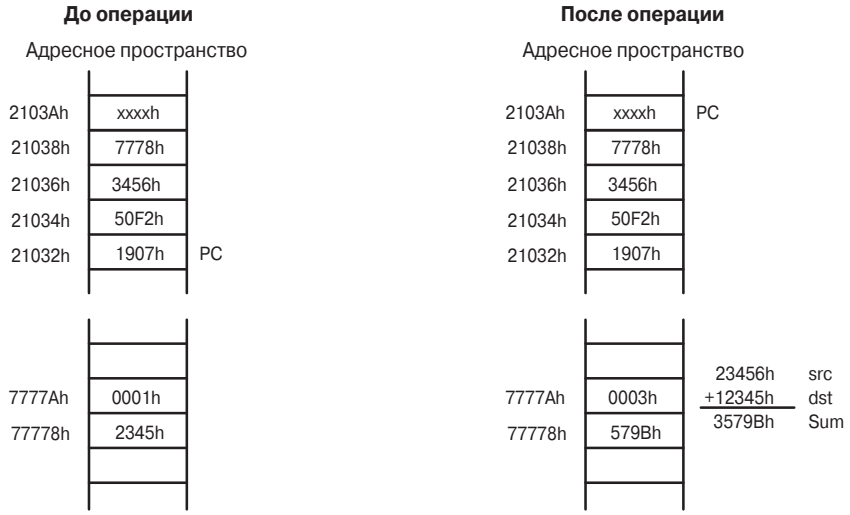
Длина:	Три или четыре слова. На одно слово меньше, если в качестве непосредственного операнда может использоваться значение, формируемое генератором констант.
Операция:	Операнд-источник, являющийся 20-битной константой, используется вместе с 20-битным операндом-приёмником.
Комментарий:	Допускается только для источника.

Пример

```
ADDX.A   #23456h,&TONI
```

Эта команда прибавляет 20-битное число 23456h к содержимому операнда-приёмника с адресом TONI.

Источник:	20-битная константа 23456h.
Приёмник:	Слово, расположенное по адресу TONI.



4.5. Команды MSP430 и MSP430X

Под командами MSP430 понимаются 27 команд ядра ЦПУ MSP430. Эти команды применяются в пределах всего адресного пространства в 1 МБ, за исключением ситуаций, когда их возможностей оказывается недостаточно. Соответственно, команды MSP430X используются в тех случаях, когда размер адреса операнда или его содержимого превышает 16 бит, что делает невозможным применение команд MSP430.

Возможны три подхода к применению команд MSP430 и MSP430X:

- Использовать только команды MSP430. Единственное исключение — команды `CALLA` и `RETA`. Этот вариант возможен при выполнении двух простых условий:
  - все константы, переменные, массивы, таблицы и данные располагаются в нижней области памяти размером 64 КБ. Это позволит использовать для доступа ко всем данным команды MSP430 с 16-битной адресацией. Не требуются указатели для хранения 20-битных адресов;
  - все локальные константы, используемые в подпрограммах, располагаются сразу же после кода подпрограммы. Это позволит задействовать относительный режим адресации с 16-битным смещением для обращения к памяти в диапазоне адресов PC ±32 КБ.
- Использовать только команды MSP430X. Недостатки такого подхода — меньшая скорость выполнения программы из-за использования дополнительных тактов ЦПУ и увеличенный размер кода из-за наличия у всех двухоперандных команд обязательного слова расширения.
- Использовать команды обоих типов, в зависимости от ситуации.

Описание команд MSP430 и MSP430X приводится в следующих подразделах.

4.5.1. Команды MSP430

Команды MSP430 могут использоваться независимо от того, где располагается программа — в нижней 64-КБ области памяти или за её пределами. Единственным исключением являются команды `CALL` и `RET`, область действия которых ограничена младшими 64 КБ памяти. Чтобы подпрограммы можно было размещать в пределах всего адресного пространства, в набор команд ЦПУ MSP430X были добавлены две команды: `CALLA` и `RETA`.

Команды с двумя операндами (формат I)

Формат команды MSP430 с двумя операндами приведён на **Рис. 4.22**. На этом же рисунке показаны дополнительные слова, определяющие источник и приёмник для индексного, относительного, абсолютного и непосредственного режимов адресации. Краткое описание всех 12 двухоперандных команд MSP430 приведено в **Табл. 4.4**.

Рис. 4.22. Формат команды MSP430 с двумя операндами.

Таблица 4.4. **Команды MSP430 с двумя операндами**

Мнемоника	S-reg, D-reg	Операция	Биты состояния			
			V	N	Z	C
MOV(.B)	src,dst	src → dst	–	–	–	–
ADD(.B)	src,dst	src + dst → dst	*	*	*	*
ADDC(.B)	src,dst	src + dst + C → dst	*	*	*	*
SUB(.B)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBC(.B)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMP(.B)	src,dst	dst – src	*	*	*	*
DADD(.B)	src,dst	src + dst + C → dst (BCD-арифметика)	*	*	*	*
BIT(.B)	src,dst	src .and. dst	0	*	*	$\overline{z}$
BIC(.B)	src,dst	not.src .and. dst → dst	–	–	–	–
BIS(.B)	src,dst	src .or. dst → dst	–	–	–	–
XOR(.B)	src,dst	src .xor. dst → dst	*	*	*	$\overline{z}$
AND(.B)	src,dst	src .and. dst → dst	0	*	*	$\overline{z}$

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

Команды с одним операндом (формат II)

Формат команд MSP430 с одним операндом (за исключением команды RETI) приведён на **Рис. 4.23**. На этом же рисунке показано дополнительное слово, определяющее операнд-приёмник для индексного, относительного, абсолютного и непосредственного режимов адресации. Краткое описание всех семи одноопе-рандных команд MSP430 приведено в **Табл. 4.5**.

Рис. 4.23. Формат команды MSP430 с одним операндом.

Таблица 4.5. **Команды MSP430 с одним операндом**

Мнемоника	S-reg, D-reg	Операция	Биты состояния			
			V	N	Z	C
RRC(.B)	dst	C → MSB →.....LSB → C	*	*	*	*
RRA(.B)	dst	MSB → MSB →....LSB → C	0	*	*	*
PUSH(.B)	src	SP – 2 →SP, src → @SP	–	–	–	–
SWPB	dst	Перестановка байтов местами	–	–	–	–
CALL	dst	Вызов подпрограммы в нижней 64-КБ области памяти	–	–	–	–
RETI	dst	TOS → SR, SP + 2 → SP TOS → PC,SP + 2 → SP	*	*	*	*
SXT	dst	Бит 7 → Бит 8.....Бит 15	0	*	*	$\overline{z}$

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

Команды перехода

Формат команд перехода MSP430 и MSP430X приведён на **Рис. 4.24**. Величина смещения в словах (10-битное число со знаком) умножается на два, расширяется до 20-битного значения и прибавляется к 20-битному счётчику команд. Это позволяет выполнять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика в пределах всего адресного пространства. Команды перехода не влияют на биты состояния. Краткое описание всех восьми команд перехода приведено в **Табл. 4.6**.

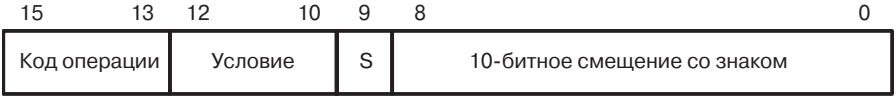


Рис. 4.24. Формат команд перехода.

Таблица 4.6. **Команды перехода**

Мнемоника	S-reg, D-reg	Операция
JEQ/JZ	Метка	Переход к метке, если бит нуля (Z) установлен
JNE/JNZ	Метка	Переход к метке, если бит нуля (Z) сброшен
JC	Метка	Переход к метке, если бит переноса (C) установлен
JNC	Метка	Переход к метке, если бит переноса (C) сброшен
JN	Метка	Переход к метке, если бит отрицательного значения (N) установлен
JGE	Метка	Переход к метке, если (N .XOR. V) = 0
JL	Метка	Переход к метке, если (N .XOR. V) = 1
JMP	Метка	Безусловный переход к метке

Эмулируемые команды

Эмулируемые команды — это команды, которые облегчают создание и чтение кода, но не имеют собственных кодов операций. Ассемблер автоматически заме-

няет такие команды эквивалентными командами ядра. Использование эмулируемых команд не приводит к увеличению размера кода или снижению производительности. Полный список эмулируемых команд приведён в **Табл. 4.7**.

Таблица 4.7. **Эмулируемые команды**

Команда	Описание	Эмуляция	V	N	Z	C
ADC(.B) dst	Сложение бита переноса с операндом	ADDC(.B) #0,dst	*	*	*	*
BR dst	Переход по заданному адресу	MOV dst,PC	–	–	–	–
CLR(.B) dst	Очистка операнда	MOV(.B) #0,dst	–	–	–	–
CLRC	Очистка бита переноса	BIC #1,SR	–	–	–	0
CLRN	Очистка бита отрицательного значения	BIC #4,SR	–	0	–	–
CLRZ	Очистка бита нуля	BIC #2,SR	–	–	0	–
DADC(.B) dst	Сложение переноса с операндом (BCD-арифметика)	DADD(.B) #0,dst	*	*	*	*
DEC(.B) dst	Декрементирование операнда	SUB(.B) #1,dst	*	*	*	*
DECD(.B) dst	Уменьшение операнда на 2	SUB(.B) #2,dst	*	*	*	*
DINT	Общее запрещение прерываний	BIC #8,SR	–	–	–	–
EINT	Общее разрешение прерываний	BIS #8,SR	–	–	–	–
INC(.B) dst	Инкрементирование операнда	ADD(.B) #1,dst	*	*	*	*
INCD(.B) dst	Увеличение операнда на 2	ADD(.B) #2,dst	*	*	*	*
INV(.B) dst	Инвертирование операнда	XOR(.B) #-1,dst	*	*	*	*
NOP	Нет операции	MOV R3,R3	–	–	–	–
POP dst	Извлечение операнда из стека	MOV @SP+,dst	–	–	–	–
RET	Возврат из подпрограммы	MOV @SP+,PC	–	–	–	–
RLA(.B) dst	Арифметический сдвиг влево	ADD(.B) dst,dst	*	*	*	*
RLC(.B) dst	Сдвиг влево через перенос	ADDC(.B) dst,dst	*	*	*	*
SBC(.B) dst	Вычитание заёма из операнда	SUBC(.B) #0,dst	*	*	*	*
SETC	Установка бита переноса	BIS #1,SR	–	–	–	1
SETN	Установка бита отрицательного значения	BIS #4,SR	–	1	–	–
SETZ	Установка бита нуля	BIS #2,SR	–	–	1	–
TST(.B) dst	Проверка операнда (на ноль)	CMP(.B) #0,dst	0	*	*	1

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

Время выполнения и размер команд MSP430

Число тактов ЦПУ, требующееся для выполнения той или иной команды, зависит от формата команды и от используемого режима адресации — но не от собственно инструкции. Приведённые далее цифры относятся к тактовому сигналу MCLK.

Сброс, подпрограммы и прерывания

Число тактов ЦПУ, требуемое для обслуживания сброса, прерываний и подпрограмм, указано в **Табл. 4.8**.

Таблица 4.8. **Обслуживание сброса, прерываний и подпрограмм**

Действие	Число тактов	Длина команды
Возврат из прерывания (`RETI`)	3*	1
Возврат из подпрограммы (`RET`)	3	1
Принятие прерывания (число тактов между появлением запроса и выполнением 1-й команды обработчика)	5**	–
Сброс от сторожевого таймера	4	–
Аппаратный сброс ($\overline{RST}$/NMI)	4	–

* В ЦПУ MSP430 — 5.
** В ЦПУ MSP430 — 6.

Время выполнения и длина команд формата II (с одним операндом)

Длина команд MSP430 формата II и время их выполнения для всех возможных режимов адресации приведены в **Табл. 4.9**.

Таблица 4.9. **Время выполнения и длина команд MSP430 формата II**

Режим адресации	Количество тактов			Длина команды	Пример
	`RRA, RRC, SWPB, SXT`	`PUSH`	`CALL`		
`Rn`	1	3	3*	1	`SWPB R5`
`@Rn`	3	3*	4	1	`RRC @R9`
`@Rn+`	3	3*	4**	1	`SWPB @R10+`
`#N`	Не применимо	3*	4**	2	`CALL #LABEL`
`X(Rn)`	4	4**	4**	2	`CALL 2(R7)`
`EDE`	4	4**	4**	2	`PUSH EDE`
`&EDE`	4	4**	4**	2	`SXT &EDE`

* В ЦПУ MSP430 — 4.
** В ЦПУ MSP430 — 5. Также время выполнения равно 5 тактам для индексного режима адресации, если Rn = SP.

Время выполнения и длина команд формата III (команды перехода)

Все команды перехода занимают одно слово в памяти и выполняются за два такта, независимо от того, был или не был выполнен переход.

Время выполнения и длина команд формата I (с двумя операндами)

Длина команд MSP430 формата I и время их выполнения для всех возможных режимов адресации приведены в **Табл. 4.10**.

Таблица 4.10. **Время выполнения и длина команд MSP430 формата I**

Режим адресации		Число тактов	Длина команды	Пример
Источник	**Приёмник**			
Rn	Rm	1	1	MOV R5,R8
	PC	2	1	BR R9
	x(Rm)	4*	2	ADD R5,4(R6)
	EDE	4*	2	XOR R8,EDE
	&EDE	4*	2	MOV R5,&EDE
@Rn	Rm	2	1	AND @R4,R5
	PC	2	1	BR @R8
	x(Rm)	5*	2	XOR @R5,8(R6)
	EDE	5*	2	MOV @R5,EDE
	&EDE	5*	2	XOR @R5,&EDE
@Rn+	Rm	2	1	ADD @R5+,R6
	PC	3	1	BR @R9+
	x(Rm)	5*	2	XOR @R5+,8(R6)
	EDE	5*	2	MOV @R9+,EDE
	&EDE	5*	2	MOV @R9+,&EDE
#N	Rm	2	2	MOV #20,R9
	PC	3	2	BR #2AEh
	x(Rm)	5*	3	MOV #0300h,0(SP)
	EDE	5*	3	ADD #33,EDE
	&EDE	5*	3	ADD #33,&EDE
x(Rn)	Rm	3	2	MOV 2(R5),R7
	PC	3	2	BR 2(R6)
	TONI	6*	3	MOV 4(R7),TONI
	x(Rm)	6*	3	ADD 4(R4),6(R9)
	&TONI	6*	3	MOV 2(R4),&TONI
EDE	Rm	3	2	AND EDE,R6
	PC	3	2	BR EDE
	TONI	6*	3	CMP EDE,TONI
	x(Rm)	6*	3	MOV EDE,0(SP)
	&TONI	6*	3	MOV EDE,&TONI
&EDE	Rm	3	2	MOV &EDE,R8
	PC	3	2	BR &EDE
	TONI	6*	3	MOV &EDE,TONI
	x(Rm)	6*	3	MOV &EDE,0(SP)
	&TONI	6*	3	MOV &EDE,&TONI

* Для команд MOV, BIT и CMP требуется на один такт меньше.

4.5.2. Команды MSP430X

Расширенный набор команд MSP430X позволяет ЦПУ MSP430X в полной мере использовать доступное адресное пространство, определяемое 20-битной шиной адреса. Большинство команд MSP430X требуют ещё одного слова для хранения кода команды, так называемого слова расширения. Некоторым из команд расширенного набора дополнительного слова не требуется — это будет отдельно указано в описании таких команд. Все адреса, смещения и непосредственные значения, которым предшествует слово расширения, являются 20-битными.

Предусмотрены слова расширения двух типов:

- для регистрового режима адресации в командах формата II (оба операнда) и формата I;
- для всех остальных сочетаний режимов адресации.

Слово расширения для регистрового режима адресации

Формат слова расширения для регистрового режима адресации приведён на **Рис. 4.25**, а назначение отдельных его битов описано в **Табл. 4.11**. Пример команды, использующей слово расширения такого формата, показан далее на **Рис. 4.27**.

15 … 12	11	10 … 9	8	7	6	5	4	3 … 0
0001	1	00	ZC	#	A/L	0	0	(*n* – 1)/Rn

Рис. 4.25. Слово расширения для регистрового режима адресации.

Таблица 4.11. **Биты слова расширения для регистрового режима адресации**

Бит	Описание
15:11	Код операции слова расширения. Для слова расширения зарезервированы значения с 1800h до 1FFFh
10:9	Зарезервировано
ZC	Нулевой бит переноса: 0 — исполняемая команда использует текущее значение бита переноса C. 1 — исполняемая команда использует бит переноса, как если бы он был сброшен. Новое значение бита переноса будет определяться результатом последней операции после выполнения команды.
#	Бит повтора: 0 — число повторений команды задаётся битами 3:0 слова расширения. 1 — число повторений команды определяется значением четырёх младших битов Rn. См. описание битов 3:0.
A/L	Бит разрядности данных. Вместе с битом B/W команды MSP430 определяет размер данных, используемых в команде: A/L B/W Размер 0 0 Зарезервировано 0 1 20-битные значения 1 0 16-битные значения 1 1 8-битные значения
5:4	Зарезервировано
3:0	Число повторений: # = 0 — эти биты определяют число повторений *n* (в битах содержится значение *n* – 1) # = 1 — эти биты определяют регистр ЦПУ, биты 3:0 которого содержат число повторов. В битах Rn.3:0 содержится значение *n* – 1.

Слово расширения для других режимов адресации

Формат слова расширения для остальных режимов адресации приведён на **Рис. 4.26**, а назначение отдельных битов описано в **Табл. 4.12**. Пример команды, использующей слово расширения такого формата, показан на **Рис. 4.28**.

15 … 12	11	10 … 7	6	5	4	3 … 0
0 0 0 1	1	Биты 19:16 источника	A/L	0	0	Биты 19:16 приёмника

Рис. 4.26. Слово расширения для режимов адресации, отличных от регистрового.

Таблица 4.12. **Биты слова расширения для режимов адресации, отличных от регистрового**

Бит	Описание
15:11	Код операции слова расширения. Для слова расширения зарезервированы значения с 1800h до 1FFFh
Биты 19:16 источника	Старшие 4 бита 20-битного значения источника. В зависимости от режима адресации операнда-источника в этих битах может содержаться 4 старших бита непосредственного значения, смещения или абсолютного адреса
A/L	Бит разрядности данных. Вместе с битом B/W команды MSP430 определяет размер данных, используемых в команде: A/L B/W Размер 0 0 Зарезервировано 0 1 20-битные значения 1 0 16-битные значения 1 1 8-битные значения
5:4	Зарезервировано
Биты 19:16 приёмника	Старшие 4 бита 20-битного значения приёмника. В зависимости от режима адресации операнда-приёмника в этих битах может содержаться 4 старших бита смещения или абсолютного адреса

Примечание. Значения битов B/W и A/L для команд `SWPBX` и `SXTX`

Для команд `SWPBX` и `SXTX` используются следующие установки битов B/W и A/L:

A/L	B/W	
0	0	`SWPBX.A`, `SXTX.A`
0	1	неприменимо
1	0	`SWPBX.W`, `SXTX.W`
1	1	неприменимо

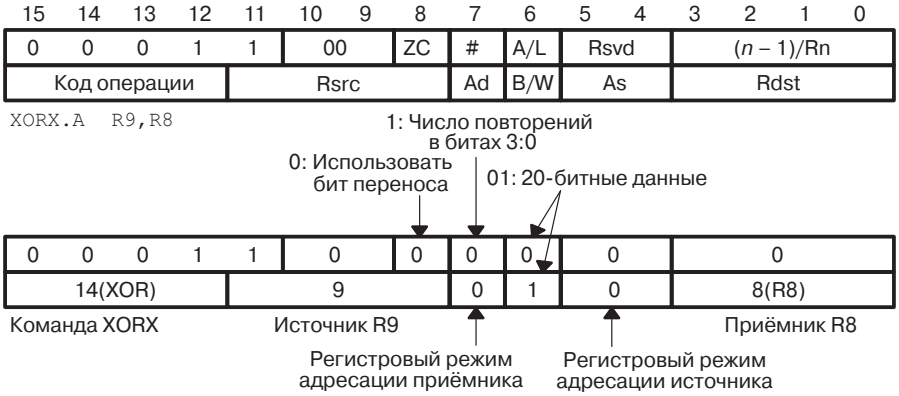


Рис. 4.27. Пример расширенной команды, использующей регистровый режим адресации.

Рис. 4.28. Пример расширенной команды, использующей непосредственный и индексный режимы адресации.

Расширенные команды с двумя операндами (формат I)

Все команды с двумя операндами имеют расширенные версии, перечисленные в **Табл. 4.13**.

Таблица 4.13. **Команды MSP430X с двумя операндами**

Мнемоника	Операнды	Операция	Биты состояния			
			V	N	Z	C
MOVX(.B,.A)	src,dst	src → dst	–	–	–	–
ADDX(.B,.A)	src,dst	src + dst → dst	*	*	*	*
ADDCX(.B,.A)	src,dst	src + dst + C → dst	*	*	*	*
SUBX(.B,.A)	src,dst	dst + .not.src + 1 → dst	*	*	*	*
SUBCX(.B,.A)	src,dst	dst + .not.src + C → dst	*	*	*	*
CMPX(.B,.A)	src,dst	dst – src	*	*	*	*
DADDX(.B,.A)	src,dst	src + dst + C → dst (BCD-арифметика)	*	*	*	*
BITX(.B,.A)	src,dst	src .and. dst	0	*	*	$\overline{z}$
BICX(.B,.A)	src,dst	not.src .and. dst → dst	–	–	–	–
BISX(.B,.A)	src,dst	src .or. dst → dst	–	–	–	–
XORX(.B,.A)	src,dst	src .xor. dst → dst	*	*	*	$\overline{z}$
ANDX(.B,.A)	src,dst	src .and. dst → dst	0	*	*	$\overline{z}$

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

Возможные форматы расширенных команд с двумя операндами показаны на **Рис. 4.29**.

Если 20-битный адрес операнда-источника или операнда-приёмника расположен не в регистре ЦПУ, а в памяти, то для хранения этого адреса используется два слова, как показано на **Рис. 4.30**.

15	14	13	12	11	10	9	8	7	6	5	4	3 … 0
0	0	0	1	1	0	0	ZC	#	A/L	0	0	*n* – 1/Rn
Код операции				src				0	B/W	0	0	dst

0	0	0	1	1	src.19:16			A/L	0	0	0	0	0	0
Код операции				src			Ad	B/W	As		dst			
src.15:0														

0	0	0	1	1	0	0	0	0	A/L	0	0	dst.19:16
Код операции				src			Ad	B/W	As		dst	
dst.15:0												

0	0	0	1	1	src.19:16			A/L	0	0	dst.19:16	
Код операции				src			Ad	B/W	As		dst	
src.15:0												
dst.15:0												

Рис. 4.29. Форматы расширенных команд с двумя операндами.

	15 14 13 12 11 10 9 8 7 6 5 4	3 2 1 0
Адрес + 2	0 ……0	19:16
Адрес	Биты 15:0 операнда	

Рис. 4.30. Хранение 20-битных адресов в памяти.

Расширенные команды с одним операндом (формат II)

Расширенные команды формата II перечислены в **Табл. 4.14**.

Таблица 4.14. **Расширенные команды с одним операндом**

Мнемоника	Операнды	Операция	*n*	Биты состояния			
				V	N	Z	C
CALLA	dst	Косвенный вызов подпрограммы (20-битный адрес)		–	–	–	–
POPM.A	#n,Rdst	Извлечение *n* 20-битных регистров из стека	1...16	–	–	–	–
POPM.W	#n,Rsrc	Извлечение *n* 16-битных регистров из стека	1...16	–	–	–	–
PUSHM.A	#n,Rsrc	Загрузка *n* 20-битных регистров в стек	1...16	–	–	–	–
PUSHM.W	#n,Rdst	Загрузка *n* 16-битных регистров в стек	1...16	–	–	–	–
PUSHX(.B,.A)	src	Загрузка 8/16/20-битного операнда в стек		–	–	–	–
RRCM(.A)	#n,Rdst	Сдвиг операнда вправо на *n* битов через перенос (16/20-битный регистр)	1...4	0	*	*	*
RRUM(.A)	#n,Rdst	Логический сдвиг операнда вправо на *n* битов (16/20-битный регистр)	1...4	0	*	*	*
RRAM(.A)	#n,Rdst	Арифметический сдвиг операнда вправо на *n* битов (16/20-битный регистр)	1...4	*	*	*	*
RLAM(.A)	#n,Rdst	Арифметический сдвиг операнда влево на *n* битов (16/20-битный регистр)	1...4	*	*	*	*
RRCX(.B,.A)	dst	Сдвиг операнда вправо через перенос (8/16/20-битный операнд)	1	0	*	*	*
RRUX(.B,.A)	dst	Логический сдвиг операнда вправо (8/16/20-битный операнд)	1	0	*	*	*
RRAX(.B,.A)	dst	Арифметический сдвиг операнда вправо (8/16/20-битный операнд)	1	*	*	*	*
SWPBX(.A)	dst	Перестановка байтов местами	1	–	–	–	–
SXTX(.A)	Rdst	Бит 7 → Бит 8.....Бит 19	1	0	*	*	*
SXTX(.A)	dst	Бит 7 → Бит 8.....MSB	1	0	*	*	*

* Влияет на бит состояния
– Не влияет на бит состояния
0 Бит состояния сбрасывается
1 Бит состояния устанавливается

Все три возможных формата расширенных команд с одним операндом показаны на **Рис. 4.31**.

Однооперандные команды MSP430X с нестандартным форматом

На **Рис. 4.32...4.35** показаны однооперандные команды, формат которых отличается от общепринятого.

15	14	13	12	11	10	9	8	7	6	5	4	3 … 0
0	0	0	1	1	0	0	ZC	#	A/L	0	0	*n*–1/Rn
Код операции									B/W	0	0	dst

0	0	0	1	1	0	0	0	0	A/L	0	0	0	0	0	0
Код операции									B/W	1	x	dst			

0	0	0	1	1	0	0	0	0	A/L	0	0	dst.19:16
Код операции									B/W	x	1	dst
dst.15:0												

Рис. 4.31. Формат расширенной команды с одним операндом.

15 … 8	7 … 4	3 … 0
Код операции	*n*–1	Rdst – *n*+1

Рис. 4.32. Формат команд `PUSHM`/`POPM`.

15 … 12	11 … 10	9 … 4	3 … 0
C	*n*–1	Код операции	Rdst

Рис. 4.33. Формат команд `RRCM`, `RRAM`, `RRUM` и `RLAM`.

15 … 12	11 … 8	7 … 4	3 … 0
C	Rsrc	Код операции	0(PC)

C	#imm/abs19:16	Код операции	0(PC)
#imm15:0 / &abs15:0			

C	Rsrc	Код операции	0(PC)
Индекс 15:0			

Рис. 4.34. Формат команды `BRA`.

15 … 4	3 … 0
Код операции	Rdst

Код операции	Rdst
Индекс15:0	

Код операции	#imm/ix/abs19:16
#imm15:0 / индекс15:0 / &abs15:0	

Рис. 4.35. Формат команды `CALLA`.

Расширенные эмулируемые команды

Расширенные эмулируемые команды образуются за счёт совместного использования генератора констант и расширенных команд ядра. Полный список эмулируемых команд приведён в **Табл. 4.15**.

Таблица 4.15. **Расширенные эмулируемые команды**

Команда		Описание	Эмуляция	
ADCX(.B,.A)	dst	Сложение бита переноса с операндом	ADDCX(.B,.A)	#0,dst
BRA	dst	Переход по заданному адресу	MOVA	dst,PC
RETA		Возврат из подпрограммы	MOVA	@SP+,PC
CLRA	Rdst	Очистка регистра Rdst	MOV	#0,Rdst
CLRX(.B,.A)	dst	Очистка операнда	MOVX(.B,.A)	#0,dst
DADCX(.B,.A)	dst	Сложение переноса с операндом (BCD-арифметика)	DADDX(.B,.A)	#0,dst
DECX(.B,.A)	dst	Декрементирование операнда	SUBX(.B,.A)	#1,dst
DECDA	Rdst	Уменьшение регистра Rdst на 2	SUBA	#2,Rdst
DECDX(.B,.A)	dst	Уменьшение операнда на 2	SUBX(.B,.A)	#2,dst
INCX(.B,.A)	dst	Инкрементирование операнда	ADDX(.B,.A)	#1,dst
INCDA	Rdst	Увеличение регистра Rdst на 2	ADDA	#2,Rdst
INCDX(.B,.A)	dst	Увеличение операнда на 2	ADDX(.B,.A)	#2,dst
INVX(.B,.A)	dst	Инвертирование операнда	XORX(.B,.A)	#-1,dst
RLAX(.B,.A)	dst	Арифметический сдвиг влево	ADDX(.B,.A)	dst,dst
RLCX(.B,.A)	dst	Сдвиг влево через перенос	ADDCX(.B,.A)	dst,dst
SBCX(.B,.A)	dst	Вычитание заёма из операнда	SUBCX(.B,.A)	#0,dst
TSTA	Rdst	Проверка регистра Rdst (на ноль)	CMPA	#0,Rdst
TSTX(.B,.A)	dst	Проверка операнда (на ноль)	CMPX(.B,.A)	#0,dst
POPX	dst	Извлечение операнда из стека	MOVX(.B,.A)	@SP+,dst

Адресные команды MSP430X

Адресные команды MSP430X — это такие команды, которые способны работать с 20-битными операндами, однако ограничены в использовании ряда режимов адресации. Все адресные команды, за исключением команды MOVA, могут задействовать только регистровый и непосредственный режимы адресации (**Табл. 4.16**). Это ограничение избавляет от необходимости использовать дополнительное слово расширения, что увеличивает плотность кода и скорость выполнения программы. Адресные команды следует использовать каждый раз, когда требуется применение команды MSP430X с соответствующим режимом адресации.

Таблица 4.16. **Адресные команды MSP430X**

Мнемоника	Операнды	Операция	Биты состояния			
			V	N	Z	C
ADDA	Rsrc,Rdst #imm20,Rdst	Сложение операнда-источника с регистром-приёмником	*	*	*	*
MOVA	Rsrc,Rdst #imm20,Rdst z16(Rsrc),Rdst EDE,Rdst &abs20,Rdst @Rsrc,Rdst @Rsrc+,Rdst Rsrc,z16(Rdst) Rsrc,&abs20	Пересылка операнда-источника в операнд-приёмник	–	–	–	–
CMPA	Rsrc,Rdst #imm20,Rdst	Сравнение операнда-источника с регистром-приёмником	*	*	*	*
SUBA	Rsrc,Rdst #imm20,Rdst	Вычитание операнда-источника из регистра-приёмника	*	*	*	*

* Влияет на бит состояния
– Не влияет на бит состояния

Время выполнения и длина команд MSP430X

Число тактов ЦПУ, требующееся для выполнения той или иной команды, зависит от формата команды и от используемого режима адресации — но не от собственно инструкции. Приведённые далее цифры относятся к тактовому сигналу MCLK.

Время выполнения и длина команд формата II (с одним операндом)

Длина команд MSP430X формата II и время их выполнения для всех возможных режимов адресации приведены в **Табл. 4.17**.

Таблица 4.17. **Время выполнения и длина команд MSP430X формата II**

	Число тактов / длина команды (слов)						
Команда	Rn	@Rn	@Rn+	#N	X(Rn)	EDE	&EDE
RRAM	*n*/1	–	–	–	–	–	–
RRCM	*n*/1	–	–	–	–	–	–
RRUM	*n*/1	–	–	–	–	–	–
RLAM	*n*/1	–	–	–	–	–	–
PUSHM	2+*n*/1	–	–	–	–	–	–
PUSHM.A	2+2*n*/1	–	–	–	–	–	–
POPM	2+*n*/1	–	–	–	–	–	–
POPM.A	2+2*n*/1	–	–	–	–	–	–
CALLA	4/1	5/1	5/1	4/2	6*/2	6/2	6/2
RRAX(.B)	1+*n*/2	4/2	4/2	–	5/3	5/3	5/3
RRAX.A	1+*n*/2	6/2	6/2	–	7/3	7/3	7/3
RRCX(.B)	1+*n*/2	4/2	4/2	–	5/3	5/3	5/3
RRCX.A	1+*n*/2	6/2	6/2	–	7/3	7/3	7/3
PUSHX(.B)	4/2	4/2	4/2	4/3	5*/3	5/3	5/3
PUSHX.A	5/2	6/2	6/2	6/3	7*/3	7/3	7/3
POPX(.B)	3/2	–	–	–	5/3	5/3	5/3
POPX.A	4/2	–	–	–	7/3	7/3	7/3

* Увеличивается на один такт, если Rn = SP.

Время выполнения и длина команд формата I (с двумя операндами)

Длина команд MSP430X формата I и время их выполнения для всех возможных режимов адресации приведены в **Табл. 4.18**.

Таблица 4.18. **Время выполнения и длина команд MSP430X формата I**

Режим адресации		Число тактов		Длина команды	Примеры
Источник	Приёмник	.B/.W	.A	.B/.W/.A	
Rn	Rm*	2	2	2	BITX.B R5,R8
	PC	3	3	2	ADDX R9,PC
	X(Rm)	5**	7***	3	ANDX.A R5,4(R6)
	EDE	5**	7***	3	XORX R8,EDE
	&EDE	5**	7***	3	BITX.W R5,&EDE
@Rn	Rm	3	4	2	BITX @R5,R8
	PC	3	4	2	ADDX @R9,PC
	X(Rm)	6**	9***	3	ANDX.A @R5,4(R6)
	EDE	6**	9***	3	XORX @R8,EDE
	&EDE	6**	9***	3	BITX.B @R5,&EDE
@Rn+	Rm	3	4	2	BITX @R5+,R8
	PC	4	5	2	ADDX.A @R9+,PC
	X(Rm)	6**	9***	3	ANDX @R5+,4(R6)
	EDE	6**	9***	3	XORX.B @R8+,EDE
	&EDE	6**	9***	3	BITX @R5+,&EDE
#N	Rm	3	3	3	BITX #20,R8
	PC****	4	4	3	ADDX.A #FE000h,PC
	X(Rm)	6**	8***	4	ANDX #1234,4(R6)
	EDE	6**	8***	4	XORX #A5A5h,EDE
	&EDE	6**	8***	4	BITX.B #12,&EDE
X(Rn)	Rm	4	5	3	BITX 2(R5),R8
	PC****	5	6	3	SUBX.A 2(R6),PC
	X(Rm)	7**	10***	4	ANDX 4(R7),4(R6)
	EDE	7**	10***	4	XORX.B 2(R6),EDE
	&EDE	7**	10***	4	BITX 8(SP),&EDE
EDE	Rm	4	5	3	BITX.B EDE,R8
	PC****	5	6	3	ADDX.A EDE,PC
	X(Rm)	7**	10***	4	ANDX EDE,4(R6)
	EDE	7**	10***	4	ANDX EDE,TONI
	&TONI	7**	10***	4	BITX EDE,&TONI
&EDE	Rm	4	5	3	BITX &EDE,R8
	PC****	5	6	3	ADDX.A &EDE,PC
	X(Rm)	7**	10***	4	ANDX.B &EDE,4(R6)
	TONI	7**	10***	4	XORX &EDE,TONI
	&TONI	7**	10***	4	BITX &EDE,&TONI

* Многократно выполняющиеся команды требуют n+1 тактов, где n – число повторений команды.
** Для команд MOV, BIT и CMP — на один такт меньше.
*** Для команд MOV, BIT и CMP — на два такта меньше.
**** Для команд MOV, ADD и SUB — на один такт меньше.

Время выполнения и длина адресных команд

Длина адресных команд MSP430X и время их выполнения для всех возможных режимов адресации приведены в **Табл. 4.19**.

Таблица 4.19. **Время выполнения и длина адресных команд**

Режим адресации		Число тактов		Длина команды		Пример
Источник	Приёмник	MOVA BRA	CMPA ADDA SUBA	MOVA	CMPA ADDA SUBA	
Rn	Rn	1	1	1	1	CMPA R5,R8
	PC	2	2	1	1	SUBA R9,PC
	x(Rm)	4	–	2	–	MOVA R5,4(R6)
	EDE	4	–	2	–	MOVA R8,EDE
	&EDE	4	–	2	–	MOVA R5,&EDE
@Rn	Rm	3	–	1	–	MOVA @R5,R8
	PC	3	–	1	–	MOVA @R9,PC
@Rn+	Rm	3	–	1	–	MOVA @R5+,R8
	PC	3	–	1	–	MOVA @R9+,PC
#N	Rm	2	3	2	2	CMPA #20,R8
	PC	3	3	2	2	SUBA #FE000h,PC
x(Rn)	Rm	4	–	2	–	MOVA 2(R5),R8
	PC	4	–	2	–	MOVA 2(R6),PC
EDE	Rm	4	–	2	–	MOVA EDE,R8
	PC	4	–	2	–	MOVA EDE,PC
&EDE	Rm	4	–	2	–	MOVA &EDE,R8
	PC	4	–	2	–	MOVA &EDE,PC

4.6. Описание набора команд

Ниже показана карта команд MSP430X.

	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx	MOVA, CMPA, ADDA, SUBA, RRCM, RRAM, RLAM, RRUM															
10xx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI	CALLA		
14xx	PUSHM.A, POPM.A, PUSHM.W, POPM.W															
18xx 1Cxx	Слово расширения для команд формата I и II															
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

4.6.1. Подробные описания расширенных команд

Ниже в **Табл. 4.20**, **4.21** и **4.22** приведены подробные (на уровне битов) описания команд MSP430X.

Таблица 4.20. **Команды, оперирующие стеком**

Команда	Идентификатор команды												Приёмник				Синтаксис
	15			12	11			8	7	6	5	4	3			0	
RETI	0	0	0	1	0	0	1	1	0	0	0	0	0	0	0	0	RETI
CALLA	0	0	0	1	0	0	1	1	0	1	0	0	dst				CALLA Rdst
	0	0	0	1	0	0	1	1	0	1	0	1	dst				CALLA x(Rdst)
	x.15:0																
	0	0	0	1	0	0	1	1	0	1	1	0	dst				CALLA @Rdst
	0	0	0	1	0	0	1	1	0	1	1	1	dst				CALLA @Rdst+
	0	0	0	1	0	0	1	1	1	0	0	0	&abs.19:16				CALLA &abs20
	&abs.15:0																
	0	0	0	1	0	0	1	1	1	0	0	1	x.19:16				CALLA EDE
	x.15:0																CALLA x(PC)
	0	0	0	1	0	0	1	1	1	0	0	1	imm.19:16				CALLA #imm20
	imm.15:0																
Зарезервировано	0	0	0	1	0	0	1	1	1	0	1	0	x	x	x	x	
Зарезервировано	0	0	0	1	0	0	1	1	1	1	x	x	x	x	x	x	
PUSHM.A	0	0	0	1	0	1	0	0	*n*–1				dst				PUSHM.A #n,Rdst
PUSHM.W	0	0	0	1	0	1	0	1	*n*–1				dst				PUSHM.W #n,Rdst
POPM.A	0	0	0	1	0	1	1	0	*n*–1				dst–*n*+1				POPM.A #n,Rdst
POPM.W	0	0	0	1	0	1	1	1	*n*–1				dst–*n*+1				POPM.W #n,Rdst

Таблица 4.21. **Команды пересылки и арифметических операций**

Команда	Группа команд				Источник или data.15:0				Идентификатор команды				Приёмник				Синтаксис
	15			12	11			8	7			4	3			0	
MOVA	0	0	0	0	src				0	0	0	0	dst				MOVA @Rsrc,Rdst
	0	0	0	0	src				0	0	0	1	dst				MOVA @Rsrc+,Rdst
	0	0	0	0	&abs.19:16				0	0	1	0	dst				MOVA &abs20,Rdst
	&abs.15:0																
	0	0	0	0	src				0	0	1	1	dst				MOVA x(Rsrc),Rdst
	x.15:0																±15-битное смещение x
	0	0	0	0	src				0	1	1	0	dst				MOVA Rsrc,&abs20
	&abs.15:0																
	0	0	0	0	src				0	1	1	1	dst				MOVA Rsrc,x(Rdst)
	x.15:0																±15-битное смещение x
	0	0	0	0	imm.19:16				1	0	0	0	dst				MOVA #imm20,Rdst
	imm.15:0																
CMPA	0	0	0	0	imm.19:16				1	0	0	1	dst				CMPA #imm20,Rdst
	imm.15:0																
ADDA	0	0	0	0	imm.19:16				1	0	1	0	dst				ADDA #imm20,Rdst
	imm.15:0																
SUBA	0	0	0	0	imm.19:16				1	0	1	1	dst				SUBA #imm20,Rdst
	imm.15:0																
MOVA	0	0	0	0	src				1	1	0	0	dst				MOVA Rsrc,Rdst
CMPA	0	0	0	0	src				1	1	0	1	dst				CMPA Rsrc,Rdst
ADDA	0	0	0	0	src				1	1	1	0	dst				ADDA Rsrc,Rdst
SUBA	0	0	0	0	src				1	1	1	1	dst				SUBA Rsrc,Rdst

Таблица 4.22. **Команды сдвига**

Команда	Группа команд				Повт.		ID команды		Идентификатор команды				Приёмник				Синтаксис
	15			12	11	10	9	8	7			4	3			0	
RRCM.A	0	0	0	0	*n*–1		0	0	0	1	0	0	dst				RRCM.A #n,Rdst
RRAM.A	0	0	0	0	*n*–1		0	1	0	1	0	0	dst				RRAM.A #n,Rdst
RLAM.A	0	0	0	0	*n*–1		1	0	0	1	0	0	dst				RLAM.A #n,Rdst
RRUM.A	0	0	0	0	*n*–1		1	1	0	1	0	0	dst				RRUM.A #n,Rdst
RRCM.W	0	0	0	0	*n*–1		0	0	0	1	0	1	dst				RRCM.W #n,Rdst
RRAM.W	0	0	0	0	*n*–1		0	1	0	1	0	1	dst				RRAM.W #n,Rdst
RLAM.W	0	0	0	0	*n*–1		1	0	0	1	0	1	dst				RLAM.W #n,Rdst
RRUM.W	0	0	0	0	*n*–1		1	1	0	1	0	1	dst				RRUM.W #n,Rdst

4.6.2. Команды MSP430

Далее приводятся подробные описания для всех команд MSP430.

* ADC[.W] Сложение переноса с операндом
* ADC.B Сложение переноса с 1-байтным операндом

Синтаксис	`ADC dst` или `ADC.W dst` `ADC.B dst`
Операция	dst + C → dst
Эмуляция	`ADDC #0,dst` `ADDC.B #0,dst`
Описание	Бит переноса (C) прибавляется к операнду-приёмнику. Предыдущее содержимое операнда теряется
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если dst изменился с 0FFFFh до 0000, иначе сбрасывается. Устанавливается, если dst изменился с 0FFh до 00, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	16-битный счётчик, на который указывает R13, прибавляется к 32-битному счётчику, на который указывает R12.
	`ADD @R13,0(R12) ; Складываем младшие слова`
	`ADC 2(R12) ; Прибавляем перенос к старшему слову`

Пример 2	8-битный счётчик, на который указывает R13, прибавляется к 16-битному счётчику, на который указывает R12.
	`ADD.B @R13,0(R12) ; Складываем младшие байты`
	`ADC.B 1(R12) ; Прибавляем перенос к старшему байту`

ADD[.W] Сложение двух 2-байтных операндов
ADD.B Сложение двух 1-байтных операндов

Синтаксис	`ADD src,dst` или `ADD.W src,dst` `ADD.B src,dst`
Операция	src + dst → dst
Описание	Операнд-источник прибавляется к операнду-приёмнику. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос, иначе сбрасывается. V: Устанавливается, если результат операции над двумя положительными операндами отрицателен или если результат операции над двумя отрицательными операндами положителен; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	К 16-битному счётчику CNTR, расположенному в нижней 64-КБ области памяти, прибавляется десять. `ADD.W #10,&CNTR ; Прибавляем 10 к 16-битному счётчику`

Пример 2	Слово из таблицы, адресуемое регистром R5 (в регистре — 20-битный адрес), складывается с содержимым регистра R6. В случае переноса осуществляется переход к метке `TONI`.
	`ADD.W @R5,R6 ; Прибавляем слово к R6, R6.19:16 = 0`
	`JC TONI ; Переходим в случае переноса`
	`..... ; Нет переноса`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), складывается с содержимым регистра R6. При отсутствии переноса осуществляется переход к метке `TONI`. Указатель на таблицу автоматически инкрементируется. Биты R6.19:8 = 0.
	`ADD.B @R5+,R6 ; Прибавляем байт к R6. R5 = R5 + 1. R6: 000xxh`
	`JNC TONI ; Переходим, если нет переноса`
	`..... ; Был перенос`

ADDC[.W], ADDC.B
Сложение двух операндов с учётом переноса

Синтаксис	`ADDC src,dst` или `ADDC.W src,dst` `ADDC.B src,dst`
Операция	src + dst + C → dst
Описание	Операнд-источник и бит переноса (C) прибавляются к операнду-приёмнику. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из старшего бита результата, иначе сбрасывается. V: Устанавливается, если результат операции над двумя положительными операндами отрицателен или если результат операции над двумя отрицательными операндами положителен; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	К 16-битному счётчику CNTR, расположенному в нижней 64-КБ области памяти, прибавляется число 15 и бит переноса от предыдущей операции. `ADDC.W #15,&CNTR ; Прибавляем 15+C к 16-битному счётчику`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), и бит переноса C складываются с содержимым регистра R6. В случае переноса осуществляется переход к метке `TONI`. Биты R6.19:16 = 0.
	`ADDC.W @R5,R6 ; Прибавляем слово и C к R6`
	`JC TONI ; Переходим в случае переноса`
	`..... ; Нет переноса`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), складывается с содержимым регистра R6. При отсутствии переноса осуществляется переход к метке `TONI`. Указатель на таблицу автоматически инкрементируется. Биты R6.19:8 = 0.
	`ADDC.B @R5+,R6 ; Прибавляем байт и C к R6. R5 = R5 + 1.`
	`JNC TONI ; Переходим, если нет переноса`
	`..... ; Был перенос`

AND[.W], AND.B
«Логическое И» двух операндов

Синтаксис	`AND src,dst` или `AND.W src,dst` `AND.B src,dst`
Операция	src .and. dst → dst
Описание	Выполняется операция «Логическое И» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется
Биты состояния	N: Устанавливается, если результат отрицателен (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается. C = (.not. Z) V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра R5 используется в качестве битовой маски (#0AA55h) для слова TOM, расположенного в нижней 64-КБ области памяти. Если результат равен нулю, выполняется переход к метке `TONI`. Биты R5.19:16 = 0.
	`MOV #0AA55h,R5 ; Загружаем маску в R5`
	`AND R5,&TOM ; TOM .and. R5 -> TOM`
	`JZ TONI ; Переходим, если результат нулевой`
	`..... ; Результат не равен нулю`
	или короче
	`AND #0AA55h,&TOM ; TOM .and. AA55h -> TOM`
	`JZ TONI ; Переходим, если результат нулевой`

Пример 2	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), логически перемножается с содержимым регистра R6. После выборки байта R5 инкрементируется. Биты R6.19:8 = 0.
	`AND.B @R5+,R6 ; «Логическое И» между байтом из таблицы и R6.`
	`; R5 = R5 + 1`

BIC[.W], BIC.B
Очистка битов операнда

Синтаксис	`BIC  src,dst` или `BIC.W  src,dst` `BIC.B  src,dst`
Операция	(.not. src) .and. dst → dst
Описание	Выполняется операция «Логическое И» между инвертированным значением операнда-источника и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется
Биты состояния	N: Не изменяется Z: Не изменяется C: Не изменяется V: Не изменяется
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Сбрасываются биты 15:14 регистра R5 (16-битное значение). Биты R5.19:16 = 0. `BIC  #0C000h,R5  ; Сбрасываем биты R5.19:14`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для очистки битов регистра R7. Биты R7.19:16 = 0
	`BIC.W  @R5,R7  ; Очищаем те биты в R7,`
	`; которые установлены в @R5`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), используется для очистки битов порта Port1.
	`BIC.B  @R5,&P1OUT  ; Очищаем биты порта 1,`
	`; которые установлены в @R5`

BIS[.W], BIS.B
Установка битов операнда

Синтаксис	`BIS src,dst` или `BIS.W src,dst` `BIS.B src,dst`
Операция	src .or. dst → dst
Описание	Выполняется операция «Логическое ИЛИ» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется
Биты состояния	N: Не изменяется Z: Не изменяется C: Не изменяется V: Не изменяется
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Устанавливаются биты 15 и 13 регистра R5 (16-битное значение). Биты R5.19:16 = 0. `BIS #0A000h,R5 ; Устанавливаем биты R5`
Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для установки битов регистра R7. Биты R7.19:16 = 0 `BIS.W @R5,R7 ; Устанавливаем биты R7`
Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), используется для установки битов порта Port1. Затем R5 инкрементируется. `BIS.B @R5+,&P1OUT ; Устанавливаем биты порта 1, R5 = R5 + 1`

BIT[.W], BIT.B
Проверка битов операнда

Синтаксис	`BIT  src,dst` или `BIT.W  src,dst` `BIT.B  src,dst`
Операция	src .and. dst
Описание	Выполняется операция «Логическое И» между операндом-источником и операндом-приёмником. Результат операции влияет только на биты состояния регистра SR. Режим регистровой адресации: биты регистра Rdst.19:16 и Rdst.19:8 (.W и .B соответственно) не сбрасываются!
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается. C = (.not. Z) V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Если установлен бит 15 или 14 регистра R5 (16-битное значение) или оба этих бита, то выполняется переход к метке `TONI`. Биты R5.19:16 не изменяются.
	`BIT  #C000h,R5  ; Проверяем биты R5.15:14`
	`JNZ  TONI  ; Хотя бы один бит установлен`
	`.....  ; Оба бита сброшены`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для проверки битов регистра R7. Если установлен хотя бы один бит, то выполняется переход к метке `TONI`. Биты R7.19:16 не изменяются.
	`BIT.W  @R5,R7  ; Проверяем биты R7`
	`JC  TONI  ; Хотя бы один бит установлен`
	`.....  ; Все биты сброшены`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), используется для проверки битов порта Port1. Если все биты сброшены, то выполняется переход к метке `TONI`. Затем указатель устанавливается на следующий байт таблицы.
	`BIT.B  @R5+,&P1OUT  ; Проверяем биты порта 1, R5 = R5 + 1`
	`JNC  TONI  ; Все требуемые биты сброшены`
	`.....  ; Хотя бы один бит установлен`

* BR[.W], BRANCH
Переход в пределах нижней 64-КБ области памяти

Синтаксис	`BR dst`
Операция	dst → PC
Эмуляция	`MOV dst,PC`
Описание	Выполняется безусловный переход по любому адресу в пределах нижней 64-КБ области памяти. Для операнда могут использоваться любые режимы адресации. В команде используются 2-байтные операнды
Биты состояния	Биты состояния не изменяются

Пример	Приведены примеры для всех режимов адресации.
	`BR #EXEC ; Переход к метке EXEC или заданному адресу`
	`; (например, #0A4h)`
	`; Команда ядра - MOV @PC+,PC`
	`BR EXEC ; Переход по адресу, находящемуся в EXEC`
	`; Команда ядра - MOV X(PC),PC`
	`; Косвенная адресация`
	`BR &EXEC ; Переход по адресу, находящемуся в ячейке памяти`
	`; с абсолютным адресом EXEC`
	`; Команда ядра - MOV X(0),PC`
	`; Косвенная адресация`
	`BR R5 ; Переход по адресу, содержащемуся в регистре R5`
	`; Команда ядра - MOV R5,PC`
	`; Косвенная адресация по содержимому R5`
	`BR @R5 ; Переход по адресу, содержащемуся в слове памяти,`
	`; адресуемом регистром R5`
	`; Команда ядра - MOV @R5,PC`
	`; Косвенная адресация по косвенному содержимому R5`
	`BR @R5+ ; Переход по адресу, содержащемуся в слове памяти,`
	`; адресуемом регистром R5, с последующим`
	`; инкрементированием содержимого R5`
	`; Команда ядра - MOV @R5+,PC`
	`; Косвенная адресация по косвенному содержимому R5`
	`; с автоинкрементом`
	`BR X(R5) ; Переход по адресу, содержащемуся в слове памяти`
	`; с адресом R5 + X (например, обращение к таблице`
	`; адресов, расположенной начиная с адреса X).`
	`; X может быть адресом или меткой.`
	`; Команда ядра - MOV X(R5),PC`
	`; Косвенная адресация по косвенному`
	`; содержимому R5 + X`

CALL
Вызов подпрограммы из нижней 64-КБ области памяти

Синтаксис	`CALL dst`
Операция	Dst → tmp 16-битное dst вычисляется и запоминается SP – 2 → SP PC → @SP PC сохраняется в стеке tmp → PC dst загружается в PC
Описание	Осуществляется вызов подпрограммы, расположенной по любому адресу в пределах 64 КБ. Могут использоваться все семь режимов адресации. В команде используется 2-байтный операнд. Возврат из подпрограммы осуществляется по команде `RET`.
Биты состояния	Биты состояния не изменяются. Биты PC.19:16 — сбрасываются (адрес в пределах нижней 64 КБ области)
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Приведены примеры для всех режимов адресации.
	Непосредственная адресация: вызов подпрограммы по метке (нижние 64 КБ) или по адресу.
	`CALL #EXEC ; Стартовый адрес - EXEC`
	`CALL #0AA04h ; Стартовый адрес - #0AA04h`
	Относительная адресация: вызов подпрограммы по 16-битному адресу, находящемуся в EXEC. EXEC находится по адресу (PC + X), где X – в диапазоне PC ±32 КБ.
	`CALL EXEC ; Стартовый адрес - @EXEC. z16(PC)`
	Абсолютная адресация: вызов подпрограммы по 16-битному адресу, находящемуся в ячейке с абсолютным адресом EXEC (нижние 64 КБ).
	`CALL &EXEC ; Стартовый адрес - @EXEC`
	Регистровая адресация: вызов подпрограммы по 16-битному адресу, находящемуся в регистре R5 (R5.15:0).
	`CALL R5 ; Стартовый адрес - в регистре R5`
	Косвенная адресация: вызов подпрограммы по 16-битному адресу, находящемуся в слове памяти, адресуемом регистром R5 (20-битный адрес).
	`CALL @R5 ; Стартовый адрес - @R5`
	Косвенная адресация с автоинкрементом: вызов подпрограммы по 16-битному адресу, находящемуся в слове памяти, адресуемом регистром R5 (20-битный адрес), с последующим увеличением содержимого R5 на 2.
	`CALL @R5+ ; Стартовый адрес - @R5, R5 = R5 + 2`
	Индексная адресация: вызов подпрограммы по 16-битному адресу, находящемуся в слове памяти с адресом (R5 + X), например обращение к таблице адресов, расположенной начиная с адреса X. Адрес находится в пределах нижней 64-КБ области. X — в диапазоне ±32 КБ.
	`CALL X(R5) ; Стартовый адрес - @(R5 + X). z16(R5)`

* CLR[.W], * CLR.B
Очистка операнда

Синтаксис	`CLR dst` или `CLR.W dst` `CLR.B dst`
Операция	0 → dst
Эмуляция	`MOV #0,dst` `MOV.B #0,dst`
Описание	Операнд-приёмник обнуляется
Биты состояния	Биты состояния не изменяются
Пример 1	Обнуляется слово TONI в ОЗУ. `CLR TONI ; 0 -> TONI`
Пример 2	Обнуляется регистр R5. `CLR R5`
Пример 3	Обнуляется байт TONI, расположенный в ОЗУ. `CLR.B TONI ; 0 -> TONI`

* CLRC
Очистка бита переноса

Синтаксис	`CLRC`
Операция	0 → C
Эмуляция	`BIC #1,SR`
Описание	Бит переноса (C) сбрасывается. В команде используются 2-байтные операнды
Биты состояния	N: Не изменяется. Z: Не изменяется. C: Сбрасывается. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	16-битный счётчик, на который указывает R13, прибавляется к 32-битному счётчику, на который указывает R12.
	`CLRC ; C=0: начальное значение`
	`DADD @R13,0(R12) ; Прибавляем 16-битный счётчик к младшему`
	`; слову 32-битного счётчика`
	`DADC 2(R12) ; Прибавляем перенос к старшему слову`
	`; 32-битного счётчика`

* CLRN
Очистка бита отрицательного значения

Синтаксис	`CLRN`
Операция	0 → N или (.NOT. src .AND. dst → dst)
Эмуляция	`BIC #4,SR`
Описание	Константа 04h инвертируется (0FFFBh) и логически перемножается (AND) с содержимым регистра состояния SR. Результат помещается в регистр состояния. В команде используются 2-байтные операнды
Биты состояния	N: Сбрасывается. Z: Не изменяется. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Сбрасывается флаг отрицательного значения в регистре состояния. Таким образом, исключается обработка отрицательных чисел в вызываемой подпрограмме.
	`CLRN`
	`CALL SUBR`
	`.....`
	`SUBR JN SUBRET ; Если входной результат отрицательный,`
	`..... ; то ничего не делаем и выходим`
	`SUBRET RET`

* CLRZ
Очистка бита нуля

Синтаксис	`CLRZ`
Операция	0 → Z или (.NOT. src .AND. dst → dst)
Эмуляция	`BIC #2,SR`
Описание	Константа 02h инвертируется (0FFFDh) и логически перемножается (AND) с содержимым регистра состояния SR. Результат помещается в регистр состояния. В команде используются 2-байтные операнды
Биты состояния	N: Не изменяется. Z: Сбрасывается. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	Сбрасывается флаг нуля в регистре состояния. `CLRZ`

CMP[.W], CMP.B
Сравнение двух операндов

Синтаксис	`CMP src,dst` или `CMP.W src,dst` `CMP.B src,dst`
Операция	.not.src + 1 + dst или (dst – src)
Описание	Операнд-источник вычитается из операнда приёмника. Для выполнения этой операции обратный код операнда-источника плюс 1 складывается с операндом-приёмником. Операция влияет только на биты состояния регистра SR
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src <= dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если был перенос из MSB результата, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда-источника из положительного операнда-приёмника отрицателен или результат вычитания положительного операнда-источника из отрицательного операнда-приёмника положителен, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Слово EDE сравнивается с 16-битной константой 1800h. В случае равенства выполняется переход к метке `TONI`. Слово EDE расположено в диапазоне PC ±32 КБ.
	`CMP #01800h,EDE ; Сравниваем EDE с 1800h`
	`JEQ TONI ; EDE = 1800h`
	`...`

Пример 2	Слово из таблицы с адресом (R5 + 10) сравнивается содержимым регистра R7 (20-битный адрес). Если число в R7 меньше табличного значения (сравниваются 16-битные числа со знаком), то выполняется переход к метке `TONI`. Биты R7.19:16 не изменяются. Адрес операнда-источника — 20-битный.
	`CMP.W 10(R5),R7 ; Сравниваем два числа со знаком`
	`JL TONI ; R7 < 10(R5)`
	`... ; R7 >= 10(R5)`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), сравнивается с выходным состоянием порта Port1. Если значения равны, то выполняется переход к метке `TONI`. Затем указатель устанавливается на следующий байт таблицы.
	`CMP.B @R5+,&P1OUT ; Сравниваем биты порта с таблицей,`
	`; R5 = R5 + 1`
	`JEQ TONI ; Значения одинаковы`
	`... ; Значения разные`

* DADC[.W], * DADC.B
Сложение переноса с операндом (BCD-арифметика)

Синтаксис	`DADC dst` или `DADC.W dst` `DADC.B dst`
Операция	dst + C → dst (BCD-арифметика)
Эмуляция	`DADD #0,dst` `DADD.B #0,dst`
Описание	Бит переноса (C) прибавляется к операнду-приёмнику по правилам двоично-десятичной арифметики
Биты состояния	N: Устанавливается, если MSB результата равен 1, иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если dst изменился с 9999h до 0000, иначе сбрасывается. Устанавливается, если dst изменился с 99h до 00, иначе сбрасывается. V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	4-разрядное двоично-десятичное число, находящееся в R5, прибавляется к 8-разрядному числу, на которое указывает R8.
	`CLRC ; Сбрасываем бит переноса, задавая начальные`
	`; условия для следующей команды`
	`DADD R5,0(R8) ; Складываем младшие разряды + C`
	`DADC 2(R8) ; Прибавляем перенос к старшим разрядам`

Пример 2	2-разрядное двоично-десятичное число, находящееся в R5, прибавляется к 4-разрядному числу, на которое указывает R8.
	`CLRC ; Сбрасываем бит переноса, задавая начальные`
	`; условия для следующей команды`
	`DADD.B R5,0(R8) ; Складываем младшие разряды + C`
	`DADC.B 1(R8) ; Прибавляем перенос к старшим разрядам`

DADD[.W], DADD.B
Сложение двух операндов с учётом переноса (BCD-арифметика)

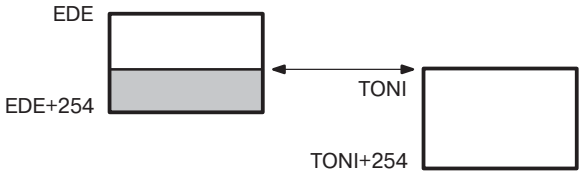
Синтаксис	`DADD src,dst` или `DADD.W src,dst` `DADD.B src,dst`
Операция	src + dst + C → dst (BCD-арифметика)
Описание	Операнд-источник и бит переноса (C) прибавляются к операнду-приёмнику по правилам двоично-десятичной арифметики. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется. Для не BCD-чисел результат операции не определён
Биты состояния	N: Устанавливается, если MSB результата равен 1 (слово > 7999h, байт > 79h), иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат слишком велик (слово > 9999h, байт > 99h), иначе сбрасывается. V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Двоично-десятичное число 10 прибавляется к 16-битному BCD-счётчику DECCNTR. `DADD #10h,&DECCNTR ; Прибавляем 10 к 4-разрядному BCD-счётчику`

Пример 2	8-разрядное двоично-десятичное число, расположенное в ОЗУ по адресам BCD и BCD+2, прибавляется к 8-разрядному числу, находящемуся в R4 и R5 (старшие разряды — BCD+2 и R5).
	`CLRC ; Сбрасываем бит переноса`
	`DADD.W &BCD,R4 ; Складываем младшие разряды. R4.19:16 = 0`
	`DADD.W &BCD+2,R5 ; Складываем старшие разряды с учётом`
	`; переноса. R5.19:16 = 0`
	`JC OVERFLOW ; Результат > 9999’9999, переходим`
	`; к обработчику ошибок`
	`... ; Результат нормальный`

Пример 3	2-разрядное двоично-десятичное число, расположенное в слове BCD, прибавляется к 2-разрядному числу, находящемуся в R4.
	`CLRC ; Сбрасываем бит переноса`
	`DADD.B &BCD,R4 ; Складываем BCD и R4. R4: 00ddh`

* DEC[.W], * DEC.B
Декрементирование операнда

Синтаксис	`DEC dst` или `DEC.W dst` `DEC.B dst`
Операция	dst – 1 → dst
Эмуляция	`SUB #1,dst` `SUB.B #1,dst`
Описание	Значение операнда-приёмника уменьшается на 1. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 1, иначе сбрасывается. C: Сбрасывается, если dst содержал 0, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается. Устанавливается, если исходное значение dst было 08000h, иначе сбрасывается. Устанавливается, если исходное значение dst было 080h, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Содержимое регистра R10 уменьшается на 1.
	`DEC R10 ; Декрементируем R10`
	`; Копируем 255-байтный блок с начальным адресом EDE в ОЗУ,`
	`; начиная с адреса TONI. Исходный и конечный блоки не должны`
	`; перекрываться: адрес TONI должен находиться вне ; диапазона`
	`; EDE...EDE + 0FEh`
	`;`
	`MOV #EDE,R6`
	`MOV #255,R10`
	`L$1 MOV.B @R6+,TONI-EDE-1(R6)`
	`DEC R10`
	`JNZ L$1`
	Не используйте данную процедуру для копирования перекрывающихся блоков (**Рис. 4.36**).
	 
	
	Рис. 4.36. Перекрытие блоков.

* DECD[.W], * DECD.B
Уменьшение операнда на 2

Синтаксис	`DECD dst` или `DECD.W dst` `DECD.B dst`
Операция	dst – 2 → dst
Эмуляция	`SUB #2,dst` `SUB.B #2,dst`
Описание	Значение операнда-приёмника уменьшается на 2. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 2, иначе сбрасывается. C: Сбрасывается, если dst содержал 0 или 1, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается. Устанавливается, если начальное значение dst было 08001h или 08000h, иначе сбрасывается. Устанавливается, если начальное значение dst было 081h или 080h, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра R10 уменьшается на 2.
	`DECD R10 ; R10 = R10 - 2`
	`; Копируем блок из 255 слов с начальным адресом EDE`
	`; в другое место, начиная с адреса TONI.`
	`; Исходный и конечный блоки не должны перекрываться:`
	`: адрес TONI должен находиться вне диапазона EDE...EDE + 01FCh`
	`;`
	`MOV #EDE,R6`
	`MOV #510,R10`
	`L$1 MOV @R6+,TONI-EDE-2(R6)`
	`DECD R10`
	`JNZ L$1`

Пример 2	Содержимое ячейки памяти с адресом LEO уменьшается на 2.
	`DECD.B LEO ; Декрементируем MEM(LEO)`
	Содержимое байта STATUS уменьшается на 2.
	`DECD.B STATUS`

* DINT
Общее запрещение прерываний

Синтаксис	`DINT`
Операция	0 → GIE или (0FFF7h .AND. SR → SR / .NOT.src .AND. dst → dst)
Эмуляция	`BIC #8,SR`
Описание	Запрещаются все прерывания. Константа 08h инвертируется и логически умножается (AND) с содержимым регистра состояния SR. Результат сохраняется в регистре состояния
Биты состояния	Биты состояния не изменяются
Биты режима	GIE сбрасывается, OSCOFF и CPUOFF не изменяются

Пример	Бит общего разрешения прерываний регистра SR сбрасывается для осуществления атомарного копирования 32-битного счётчика. Это гарантирует, что состояние счётчика не будет изменено в процессе копирования каким-либо прерыванием.
	`DINT ; Запрещаем генерацию прерываний`
	`NOP`
	`MOV COUNTHI,R5 ; Копируем счётчик`
	`MOV COUNTLO,R6`
	`EINT ; Разрешаем прерывания`

Примечание. Запрещение прерываний
При необходимости исключить прерывание какой-либо секции кода, между командой `DINT` и началом этой секции должна располагаться как минимум одна команда. Обычно для этой цели после команды `DINT` ставят команду `NOP`.

* EINT
Общее разрешение прерываний

Синтаксис	`EINT`
Операция	1 → GIE или (0008h .OR. SR → SR / src .OR. dst → dst)
Эмуляция	`BIS  #8,SR`
Описание	Разрешаются все прерывания. Константа 08h логически складывается (OR) с содержимым регистра состояния SR. Результат сохраняется в регистре SR
Биты состояния	Биты состояния не изменяются
Биты режима	GIE устанавливается, OSCOFF и CPUOFF не изменяются
Пример	Бит общего разрешения прерываний (GIE) регистра состояния устанавливается.

```
; Обработчик прерываний от линий порта ввода/вывода P1.2...P1.7
; P1IN - адрес регистра, из которого читается состояние битов порта.
; P1IFG - адрес регистра, в котором хранятся флаги прерываний по линиям порта.
        PUSH.B  &P1IN          ; Сбрасываем только взведённые флаги
        BIC.B   @SP,&P1IFG     ; Маску для флагов прерываний
                               ; сохраняем в стеке
        EINT                   ; Разрешаем прочие прерывания
        BIT     #Mask,@SP
        JEQ     MaskOK         ; Установленные флаги совпадают
                               ; с маской - переходим
        ......
MaskOK  BIC     #Mask,@SP
        ......
        INCD    SP             ; Служебная операция, обратная
                               ; команде PUSH, расположенной в
                               ; начале обработчика. Корректирует
                               ; значение указателя стека
         RETI
```

Примечание. Разрешение прерываний

Команда, следующая за командой разрешения прерываний (`EINT`), всегда выполняется, даже если на момент разрешения прерываний имеются отложенные запросы прерываний.

* INC[.W], * INC.B
Инкрементирование операнда

Синтаксис	`INC dst` или `INC.W dst` `INC.B dst`
Операция	dst + 1 → dst
Эмуляция	`ADD #1,dst` `ADD.B #1,dst`
Описание	Значение операнда-приёмника увеличивается на 1. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFFh, иначе сбрасывается. Устанавливается, если dst содержал 07Fh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Байт состояния процесса STATUS инкрементируется. Когда он становится равным 11, выполняется переход к метке `OVFL`.
	`INC.B STATUS`
	`CMP.B #11,STATUS`
	`JEQ OVFL`

* INCD[.W], * INCD.B
Увеличение операнда на 2

Синтаксис	`INCD dst` или `INCD.W dst` `INCD.B dst`
Операция	dst + 2 → dst
Эмуляция	`ADD #2,dst` `ADD.B #2,dst`
Описание	Значение операнда-приёмника увеличивается на 2. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFEh, иначе сбрасывается. Устанавливается, если dst содержал 0FEh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFEh или 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FEh или 0FFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFEh или 07FFFh, иначе сбрасывается. Устанавливается, если dst содержал 07FEh или 07Fh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Удаление значения с вершины стека без использования регистров.
	`.....`
	`PUSH R5 ; В R5 результат вычислений, сохранённый в`
	`; системном стеке`
	`INCD SP ; Удаляем значение с вершины стека, модифицируя`
	`; указатель стека`
	`; Не используйте INCD.B, так как содержимое SP`
	`; всегда выровнено по границе слова`
	`RET`

Пример 2	Байт, расположенный на вершине стека, увеличиваем на 2. `INCD.B 0(SP) ; Увеличиваем на 2 верхний байт в стеке`

* INV[.W], * INV.B
Инвертирование операнда

Синтаксис	`INV dst` или `INV.W dst` `INV.B dst`
Операция	.NOT.dst → dst
Эмуляция	`XOR #0FFFFh,dst` `XOR.B #0FFh,dst`
Описание	Значение операнда-приёмника инвертируется. Предыдущее содержимое теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. C: Устанавливается, если результат не равен нулю, иначе сбрасывается (= .NOT. Zero). V: Устанавливается, если операнд имел отрицательное значение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Вычисляется дополнительный код содержимого регистра R5.
	`MOV #00AEh,R5 ; R5 = 000AEh`
	`INV R5 ; Инвертируем R5 R5 = 0FF51h`
	`INC R5 ; В R5 - доп. код R5 = 0FF52h`

Пример 2	Вычисляется дополнительный код байта LEO в ОЗУ.
	`MOV.B #00AEh,LEO ; MEM(LEO) = 0AEh`
	`INV.B LEO ; Инвертируем LEO MEM(LEO) = 051h`
	`INC.B LEO ; В LEO - доп. код MEM(LEO) = 052h`

* JC Переход[.W], если бит переноса установлен
* JHS Переход, если «выше или равно» (беззнаковое сравнение)

Синтаксис	`JC  label` `JHS  label`
Операция	Если C = 1, то PC + (2 × смещение) → PC. Если C = 0, то выполняется следующая команда.
Описание	Проверяется бит переноса (C) регистра состояния. Если он установлен, то смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если бит C сброшен, то выполняется команда, следующая за командой перехода. Команда `JC` используется для проверки бита переноса C. Команда `JHS` используется для сравнения чисел без знака.
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Сигнал P1IN.1 используется для управления ходом выполнения программы.
	`BIT.B  #02h,&P1IN  ; Состояние бита порта -> бит переноса C`
	`JC  Label1  ; Если C = 1, переходим к Label1`
	`...  ; C = 0, продолжаем выполнение`

Пример 2	Если R5 ≥ R6 (беззнаковое содержимое), то выполнение программы продолжается с метки `Label2`.
	`CMP  R6,R5  ; R5 >= R6? Признак - бит C`
	`JHS  Label2  ; Да, C = 1`
	`...  ; Нет, R5 < R6. Продолжаем`

JEQ Переход[.W], если «равно»
JZ Переход, если ноль

Синтаксис	`JEQ label` `JZ label`
Операция	Если Z = 1, то PC + (2 × смещение) → PC. Если Z = 0, то выполняется следующая команда.
Описание	Проверяется бит нуля (Z) регистра состояния. Если он установлен, смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если бит Z сброшен, то выполняется команда, следующая за командой перехода. Команда `JZ` используется для проверки бита нуля Z. Команда `JEQ` используется для сравнения операндов
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Сигнал P2IN.0 используется для управления ходом выполнения программы.
	`BIT.B #1,&P2IN ; Бит 0 порта 2 сброшен?`
	`JZ Label1 ; Да, переходим к Label1`
	`... ; Нет, продолжаем выполнение`

Пример 2	Если R5 = 15000h (20-битное число), то выполнение программы продолжается с метки `Label2`.
	`CMPA #15000h,R5 ; R5 = 15000h? Признак - бит Z`
	`JEQ Label2 ; Да, R5 = 15000h, Z = 1`
	`... ; Нет, R5 ≠ 15000h. Остаёмся здесь`

Пример 3	Содержимое регистра R7 (20-битный счётчик) инкрементируется. Если оно становится равным нулю, то выполняется переход к метке `Label4`.
	`ADDA #1,R7 ; Инкрементируем R7`
	`JZ Label4 ; Достигли нуля? Переходим к Label4`
	`... ; R7 ≠ 0. Остаёмся здесь`

JGE Переход[.W], если «больше или равно» (сравнение со знаком)

Синтаксис	`JGE label`
Операция	Если (N .xor. V) = 0, то PC + (2 × смещение) → PC. Если (N .xor. V) = 1, то выполняется следующая команда.
Описание	Проверяются биты переполнения (V) и отрицательного значения (N) регистра состояния. Если установлены или сброшены оба бита, то смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если установлен только один бит, то выполняется команда, следующая за командой перехода. Эта команда используется при сравнении чисел со знаком. Команда `JGE` используется для сравнения операндов со знаком: даже в случае некорректного результата, вызванного переполнением, результат сравнения будет корректным. **Примечание.** Команда `JGE` эмулирует нереализованную команду `JP` (переход, если результат положителен) при использовании после команд `AND`, `BIT`, `RRA`, `SXTX` и `TST`. Эти команды сбрасывают бит V регистра состояния
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Если байт EDE (нижние 64 КБ) содержит положительное число, то выполняется переход к метке `Label1`. Этот пример может выполняться в любом месте адресного пространства.
	`TST.B &EDE ; EDE > 0? V <- 0`
	`JGE Label1 ; Да, JGE эмулирует команду JP`
	`... ; Нет, 80h <= EDE <= FFh`

Пример 2	Если содержимое регистра R6 больше или равно содержимому ячейки памяти, адресуемой регистром R7, то выполняется переход к метке `Label5`. Сравниваются числа со знаком. Данные и программа могут располагаться в любом месте адресного пространства.
	`CMP @R7,R6 ; R6 >= @R7?`
	`JGE Label5 ; Да, переходим к Label5`
	`... ; Нет, остаёмся здесь`

Пример 3	Если R5 ≥ 12345h (операнд со знаком), то выполнение программы продолжается с метки `Label2`. Программа может располагаться в любом месте адресного пространства.
	`CMPA #12345h,R5 ; R5 >= 12345h`
	`JGE Label2 ; Да, 12344h < R5 <=7FFFFh`
	`... ; Нет, 80000h <= R5 < 12345h`

JL Переход, если «меньше» (сравнение со знаком)

Синтаксис	`JL label`
Операция	Если (N .xor. V) = 1, то PC + (2 × смещение) → PC. Если (N .xor. V) = 0, то выполняется следующая команда.
Описание	Проверяются биты переполнения (V) и отрицательного значения (N) регистра состояния. Если установлен только один из битов, то смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от −511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если установлены или сброшены оба бита, то выполняется команда, следующая за командой перехода. Команда `JL` используется для сравнения операндов со знаком: даже в случае некорректного результата, вызванного переполнением, результат сравнения будет корректным
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Если содержимое байта EDE меньше, чем байта TONI, то выполняется переход к метке `Label1`. Байт EDE находится в пределах области PC ±32 КБ.
	`CMP.B &EDE,TONI ; EDE < TONI`
	`JL Label1 ; Да`
	`... ; Нет, TONI <= EDE`

Пример 2	Если содержимое регистра R6 меньше содержимого ячейки памяти, адресуемой регистром R7 (20-битный адрес), то выполняется переход к метке `Label5`. Данные и программа могут располагаться в любом месте адресного пространства.
	`CMP @R7,R6 ; R6 < @R7?`
	`JL Label5 ; Да, переходим к Label5`
	`... ; Нет, остаёмся здесь`

Пример 3	Если R5 < 12345h (операнд со знаком), то выполнение программы продолжается с метки `Label2`. Данные и программа могут располагаться в любом месте адресного пространства.
	`CMPA #12345h,R5 ; R5 < 12345h`
	`JL Label5 ; Да, 80000h <= R5 < 12345h`
	`... ; Нет, 12344h < R5 <=7FFFFh`

JMP Безусловный переход

Синтаксис	`JMP  label`
Операция	PC + (2 × смещение) → PC
Описание	Смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Данная команда может использоваться вместо команды `BR` или `BRA` для реализации перехода в пределах от –511 до +512 слов относительно счётчика команд
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	В байт STATUS заносится число 10, после чего выполняется переход к метке `MAINLOOP`. Байт STATUS располагается в нижней 64-КБ области памяти, а программа может располагаться в любом месте адресного пространства.
	`MOV.B  #10,&STATUS  ; STATUS = 10`
	`JMP  MAINLOOP  ; Переходим к основному циклу`

Пример 2	Считывается содержимое вектора прерывания Timer_A3, которое затем используется для перехода к требуемому обработчику. Программа может располагаться в любом месте адресного пространства, однако точки входа в обработчики прерываний должны располагаться в нижней 64-КБ области памяти.
	`ADD  &TAIV,PC  ; Прибавляем содержимое вектора к PC`
	`RETI  ; Нет прерываний Timer_A`
	`JMP  IHCCR1  ; Было прерывание от 1-го блока таймера`
	`JMP  IHCCR2  ; Было прерывание от 2-го блока таймера`
	`RETI  ; Нет обрабатываемых прерываний, выходим`

JN Переход, если отрицательное значение

Синтаксис	`JN  label`
Операция	Если N = 1, то PC + (2 × смещение) → PC. Если N = 0, то выполняется следующая команда.
Описание	Проверяется бит отрицательного значения (N) регистра состояния. Если он установлен, то смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если бит N сброшен, то выполняется команда, следующая за командой перехода
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Проверяется содержимое байта COUNT. Если оно отрицательно, то выполнение программы продолжается с метки `Label0`. Байт COUNT располагается в нижней 64-КБ области памяти, а программа может располагаться в любом месте адресного пространства.
	`TST.B  &COUNT  ; COUNT < 0?`
	`JN  L$1  ; Да, переходим к Label0`
	`...  ; COUNT ≥ 0`

Пример 2	Регистр R6 вычитается из R5. Если результат отрицателен, то выполняется переход к метке `Label2`. Программа может располагаться в любом месте адресного пространства.
	`SUB  R6,R5  ; R5 - R6 -> R5`
	`JN  Label2  ; R5 < 0: R6 > R5 (N = 1)`
	`...  ; R5 ≥ 0, остаёмся здесь`

Пример 3	Регистр R7 (20-битный счётчик) декрементируется. Если его значение становится отрицательным, то выполнение программы продолжается с метки `Label4`. Программа может располагаться в любом месте адресного пространства.
	`SUBA  #1,R7  ; Декрементируем R7`
	`JN  Label4  ; R7 < 0, переходим к Label4`
	`...  ; R7 ≥ 0, остаёмся здесь`

JNC Переход, если нет переноса
JLO Переход, если «ниже» (сравнение без знака)

Синтаксис	`JNC label` `JLO label`
Операция	Если C = 0, то PC + (2 × смещение) → PC. Если C = 1, то выполняется следующая команда.
Описание	Проверяется бит переноса (C) регистра состояния. Если он сброшен, то смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если бит C установлен, то выполняется команда, следующая за командой перехода. Команда `JNC` используется для проверки бита переноса C. Команда `JLO` используется для сравнения чисел без знака
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Если EDE <15, выполняется переход к метке `Label2`. Сравниваются числа без знака. Байт EDE располагается в нижней 64-КБ области памяти, а программа может располагаться в любом месте адресного пространства.
	`CMP.B #15,&EDE ; EDE < 15? Признак -> бит C`
	`JLO Label2 ; Да, EDE < 15. C = 0`
	`... ; Нет, EDE ≥ 15. Остаёмся здесь`

Пример 2	Слово TONI прибавляется к содержимому регистра R5. При отсутствии переноса выполнение программы продолжается с метки `Label0`. Адрес TONI находится в пределах PC ±32 КБ.
	`ADD TONI,R5 ; TONI + R5 -> R5. Перенос -> C`
	`JNC Label0 ; Нет переноса`
	`... ; Был перенос, остаёмся здесь`

JNZ Переход, если не ноль
JNE Переход, если «не равно»

Синтаксис	`JNZ label` `JNE label`
Операция	Если Z = 0, то PC + (2 × смещение) → PC. Если Z = 1, то выполняется следующая команда.
Описание	Проверяется бит нуля (Z) регистра состояния. Если он сброшен, то смещение (10-битное число со знаком), содержащееся в слове команды, умножается на 2, расширяется до 20-битного значения и прибавляется к счётчику команд. Это позволяет осуществлять переход в диапазоне от –511 до +512 слов относительно текущего значения счётчика команд в пределах всего адресного пространства. Если бит Z установлен, то выполняется команда, следующая за командой перехода. Команда `JNZ` используется для проверки бита нуля Z. Команда `JNE` используется для сравнения операндов
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Проверяется значение байта STATUS. Если он не равен нулю, то выполняется переход к метке `Label3`. Адрес байта STATUS находится в диапазоне PC ±32 КБ.
	`TST.B STATUS ; STATUS = 0?`
	`JNZ Label3 ; Нет, переходим к Label3`
	`... ; Да, остаёмся здесь`

Пример 2	Если слово EDE не равно 1500, то выполняется переход к метке `Label2`. Слово EDE располагается в нижней 64-КБ области памяти, а программа может располагаться в любом месте адресного пространства.
	`CMP #1500,&EDE ; EDE = 1500? Признак -> бит Z`
	`JNE Label2 ; Нет, EDE ≠ 1500.`
	`... ; Да, EDE = 1500. Продолжаем`

Пример 3	Регистр R7 (20-битный счётчик) декрементируется. Если он не равен нулю, то выполняется переход к метке `Label4`. Программа может располагаться в любом месте адресного пространства.
	`SUBA #1,R7 ; Декрементируем R7`
	`JNZ Label4 ; R7 ≠ 0, переходим к Label4`
	`... ; R7 = 0, остаёмся здесь`

MOV[.W], MOV.B
Пересылка операнда

Синтаксис	`MOV src,dst` или `MOV.W src,dst` `MOV.B src,dst`
Операция	src → dst
Описание	Операнд-источник пересылается в операнд-приёмник. Содержимое операнда-источника не изменяется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	16-битная константа 1800h загружается в слово EDE (нижняя 64-КБ область). `MOV #01800h,&EDE ; 1800h -> EDE`

Пример 2	Содержимое таблицы EDE (2-байтные элементы, 16-битная адресация) копируется в таблицу TOM. Размер таблиц — 030h слов. Обе таблицы располагаются в нижней 64-КБ области памяти.
	`MOV #EDE,R10 ; Инициализируем указатель`
	`Loop MOV @R10+,TOM-EDE-2(R10) ; Используем указатель в R10`
	`; для обеих таблиц. R10 + 2`
	`CMP #EDE+60h,R10 ; Достигли конца таблицы?`
	`JLO Loop ; Ещё нет`
	`... ; Копирование завершено`

Пример 3	Содержимое таблицы EDE (1-байтные элементы, 16-битная адресация) копируется в таблицу TOM. Размер таблиц — 020h байтов. Обе таблицы могут располагаться в любом месте адресного пространства, однако должны находиться в пределах R10 ± 32 КБ.
	`MOVA #EDE,R10 ; Инициализируем указатель`
	`MOV #020h,R9 ; Инициализируем счетчик`
	`Loop MOV.B @R10+,TOM-EDE-1(R10) ; Используем указатель в R10`
	`; для обеих таблиц. R10 + 1`
	`DEC R9 ; Декрементируем счётчик`
	`JNZ Loop ; Счётчик ≠ 0, продолжаем`
	`; копирование`
	`... ; Копирование завершено`

* NOP Нет операции

Синтаксис	`NOP`
Операция	Нет
Эмуляция	`MOV #0,R3`
Описание	Не выполняется никаких операций. Команда может использоваться для замены рабочих команд при отладке программы или для формирования задержек
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

* POP[.W], * POP.B
Извлечение операнда из стека

Синтаксис	`POP dst` или `POP.W dst` `POP.B dst`
Операция	@SP → temp SP + 2 → SP temp → dst
Эмуляция	`MOV @SP+,dst` или `MOV.W @SP+,dst` `MOV.B @SP+,dst`
Описание	Элемент стека, адресуемый указателем стека (TOS), извлекается в операнд-приёмник. После этого указатель стека увеличивается на 2
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое R7 и регистра состояния восстанавливаются из стека.
	`POP R7 ; Извлекаем R7`
	`POP SR ; Извлекаем регистр состояния`

Пример 2	Содержимое байта LEO в ОЗУ восстанавливается из стека. `POP.B LEO ; Копируем младший байт из стека в LEO`

Пример 3	Содержимое R7 восстанавливается из стека.
	`POP.B R7 ; Копируем младший байт из стека в R7`
	`; Старший байт R7 = 0`

Пример 4	Содержимое ячейки памяти, на которую указывает R7, и регистра состояния восстанавливается из стека.
	`POP.B 0(R7) ; Копируем младший байт с вершины стека в ОЗУ`
	`; по адресу, хранящемуся в регистре R7`
	`POP SR ; Загружаем последнее слово из стека в регистр SR`

Примечание. Указатель системного стека
Указатель системного стека (SP) всегда увеличивается на 2, независимо от разрядности (байт/слово) операнда команды.

PUSH[.W], PUSH.B
Сохранение операнда в стек

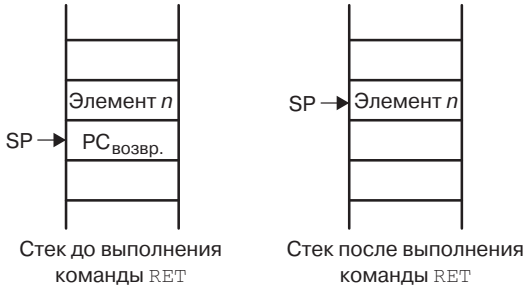
Синтаксис	`PUSH src` или `PUSH.W src` `PUSH.B src`
Операция	SP – 2 → SP src → @SP
Описание	20-битный указатель стека SP уменьшается на 2, после чего содержимое операнда-источника помещается в ОЗУ по адресу, определяемому указателем. При сохранении 1-байтного значения оно помещается в младший байт слова памяти, содержимое старшего слова при этом не изменяется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистров R9 и R10 сохраняется в стеке.
	`PUSH R9 ; Сохраняем регистр R9`
	`PUSH R10 ; Сохраняем регистр R10`

Пример 2	Содержимое двух байтов EDE и TONI сохраняется в стеке. Адреса обоих байтов находятся в диапазоне PC ±32 КБ.
	`PUSH.B EDE ; Сохраняем EDE`
	`PUSH.B TONI ; Сохраняем TONI`

* RET Возврат из подпрограммы

Синтаксис	RET
Операция	@SP → PC.15:0; PC.19:16 ← 0 SP + 2 → SP
Эмуляция	MOV @SP+,PC
Описание	16-битный адрес возврата (нижние 64 КБ), помещённый в стек при выполнении команды CALL, загружается в счётчик команд. Выполнение программы продолжается с команды, следующей за командой вызова подпрограммы. Четыре старших бита PC.19:16 сбрасываются
Биты состояния	Биты состояния не изменяются. PC.19:16 сбрасываются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

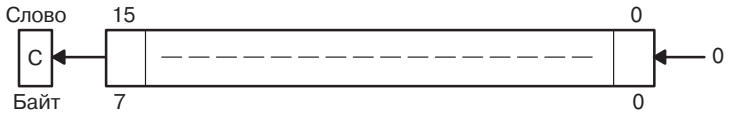
Пример	Вызывается подпрограмма SUBR, расположенная в нижней 64-КБ области памяти.
	`CALL #SUBR ; Вызываем подпрограмму по адресу SUBR`
	`... ; Из подпрограммы возвращаемся сюда`
	`SUBR PUSH R14 ; Сохраняем R14 (16-битное значение)`
	`... ; Тело подпрограммы`
	`POP R14 ; Восстанавливаем R14`
	`RET ; Возвращаемся в нижние 64 КБ`
	 
	
	Рис. 4.37. Состояние стека после выполнения команды RET.

RETI Возврат из прерывания

Синтаксис	`RETI`
Операция	@SP → SR.15:0 Восстанавливаем регистр состояния SR и биты PC.19:16 SP + 2 → SP TOS → PC.15:0 Восстанавливаем счётчик команд PC.15:0 SP + 2 → SP
Описание	Содержимое регистра состояния восстанавливается таким, каким оно было при входе в процедуру обработки прерывания. Одновременно восстанавливается содержимое старших битов счётчика команд PC.19:16. Затем указатель стека увеличивается на 2. 20-битное содержимое счётчика команд восстанавливается таким, каким оно было на момент принятия запроса прерывания (биты PC.19:16 — из того же элемента стека, что и регистр состояния, биты PC.15:0 — из следующего элемента). Затем указатель стека увеличивается на 2
Биты состояния	N: Восстанавливается из стека. Z: Восстанавливается из стека. C: Восстанавливается из стека. V: Восстанавливается из стека.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Подпрограмма обработки прерывания располагается в нижней 64-КБ области. В стеке сохраняется 20-битный адрес возврата.
	`INTRPT PUSHM.A #2,R14 ; Сохраняем R14 и R13`
	`; (20-битные данные)`
	`... ; Тело обработчика прерывания`
	`POPM.A #2,R14 ; Восстанавливаем R14 и R13`
	`; (20-битные данные)`
	`RETI ; Возвращаемся к 20-битному адресу в`
	`; любом месте адресного пространства`

* RLA[.W], * RLA.B
Арифметический сдвиг влево операнда

Синтаксис	`RLA dst` или `RLA.W dst` `RLA.B dst`
Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← 0
Эмуляция	`ADD dst,dst` `ADD.B dst,dst`
Описание	Содержимое операнда-приёмника сдвигается влево на один бит, как показано на **Рис. 4.38**. Старший бит содержимого помещается в бит переноса (C), а младший бит обнуляется. По сути, команда `RLA` выполняет операцию знакового умножения на 2. Если перед выполнением операции значение операнда было 04000h ≤ dst < 0C000h (040h ≤ dst < 0C0h), то возникает переполнение: знак результата отличается от знака операнда.  ***Рис. 4.38.*** Операнд-приёмник — арифметический сдвиг влево.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB. V: Устанавливается, если произошло переполнение: начальное значение 04000h ≤ dst < 0C000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 040h ≤ dst < 0C0h; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Содержимое регистра R7 умножается на 2. `RLA R7 ; Сдвигаем R7 на 1 бит влево (×2)`

Пример 2	Младший байт регистра R7 умножается на 4.
	`RLA.B R7 ; Сдвигаем влево младший байт R7 (×2)`
	`RLA.B R7 ; Сдвигаем влево младший байт R7 (×4)`

Примечание. Замена команды `RLA`

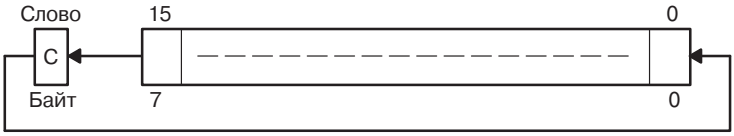
Ассемблер не распознаёт команды:

`RLA @R5+, RLA.B @R5+` или `RLA(.B) @R5`

Они должны быть заменены командами:

`ADD @R5+,-2(R5) ADD.B @R5+,-1(R5)` или `ADD(.B) @R5,0(R5)`

* RLC[.W], * RLC.B
Сдвиг операнда влево через перенос

Синтаксис	`RLC dst` или `RLC.W dst` `RLC.B dst`
Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← C
Эмуляция	`ADDC dst,dst` `ADDC.B dst,dst`
Описание	Содержимое операнда-приёмника сдвигается влево на один бит, как показано на **Рис. 4.39**. Бит переноса (C) вдвигается в младший бит (LSB) операнда, а старший бит операнда (MSB) выдвигается в бит переноса.  ***Рис. 4.39.*** Операнд-приёмник — сдвиг влево через перенос.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB. V: Устанавливается, если произошло переполнение: начальное значение 04000h ≤ dst < 0C000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 040h ≤ dst < 0C0h; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Содержимое регистра R5 сдвигается на 1 бит влево. `RLC R5 ; (R5 × 2) + C -> R5`

Пример 2	Значение бита порта P1IN.1 вдвигается в младший бит регистра R5.
	`BIT.B #2,&P1IN ; Информация -> бит переноса`
	`RLC R5 ; C=P1IN.1 -> младший бит R5`

Пример 3	Содержимое байта ОЗУ LEO сдвигается на 1 бит влево. `RLC.B LEO ; MEM(LEO) × 2 + C -> MEM(LEO)`

Примечание. Замена команды `RLC`

Ассемблер не распознаёт команды:

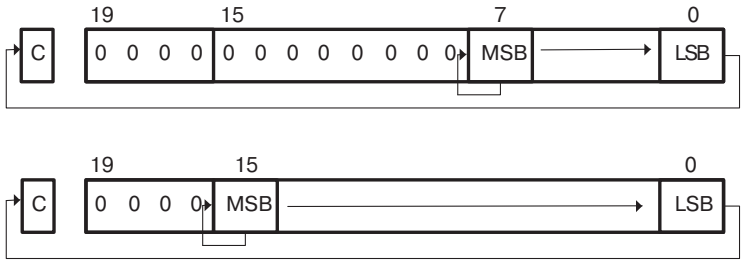
`RLC @R5+`, `RLC.B @R5+` или `RLC(.B) @R5`

Они должны быть заменены командами:

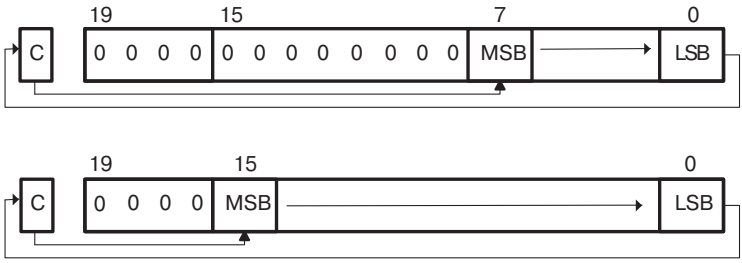
`ADDC @R5+,-2(R5)`, `ADDC.B @R5+,-1(R5)`

или `ADDC(.B) @R5,0(R5)`

RRA[.W], RRA.B
Арифметический сдвиг операнда вправо

Синтаксис	`RRA dst` или `RRA.W dst` `RRA.B dst`
Операция	MSB → MSB, MSB → MSB–1, ... LSB+1 → LSB, LSB → 0
Описание	Содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.40**. Старший бит содержимого (MSB) вдвигается в бит MSB–1 (значение MSB остаётся неизменным), а бит LSB+1 вдвигается в младший бит (LSB). По сути, команда `RLA` выполняет операцию знакового деления на 2.  ***Рис. 4.40.*** Арифметический сдвиг вправо: команды `RRA.B` и `RRA.W`.
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	16-битное число со знаком, находящееся в регистре R5, сдвигается вправо на один бит. `RRA R5 ; R5/2 -> R5`
Пример 2	Содержимое байта EDE сдвигается вправо на один бит. `RRA.B EDE ; EDE/2 -> EDE`

RRC[.W], RRC.B
Сдвиг вправо через перенос операнда

Синтаксис	`RRC dst` или `RRC.W dst` `RRC.B dst`
Операция	C → MSB → MSB–1 ... LSB+1 → LSB → C
Описание	Содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.41**. Бит переноса (C) вдвигается в старший бит (MSB) операнда, а младший бит операнда (LSB) выдвигается в бит переноса (C).  ***Рис. 4.41.*** Сдвиг вправо через перенос: команды `RRC.B` и `RRC.W`.
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Содержимое слова EDE сдвигается на 1 бит вправо. В MSB загружается 1.
	`SETC ; Инициализируем бит переноса`
	`RRC EDE ; EDE = EDE >> 1 + 8000h`

* SBC[.W], * SBC.B
Вычитание заёма из операнда

Синтаксис	`SBC dst` или `SBC.W dst` `SBC.B dst`
Операция	dst + 0FFFFh + C → dst dst + 0FFh + C → dst
Эмуляция	`SUBC #0,dst` `SUBC.B #0,dst`
Описание	Бит переноса (C), уменьшенный на 1, прибавляется к операнду-приёмнику. Предыдущее содержимое операнда теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	16-битный счётчик, на который указывает R13, вычитается из 32-битного счётчика, на который указывает R12.
	`SUB @R13,0(R12) ; Вычитаем младшие слова`
	`SBC 2(R12) ; Вычитаем заём из старшего слова`

Пример 2	8-битный счётчик, на который указывает R13, вычитается из 16-битного счётчика, на который указывает R12.
	`SUB.B @R13,0(R12) ; Вычитаем младшие байты`
	`SBC.B 1(R12) ; Вычитаем заём из старшего байта`

Примечание. Признак заёма

В качестве признака заёма используется инверсное значение бита переноса:

Заём	Бит C
Есть	0
Нет	1

* SETC Установка бита переноса

Синтаксис	`SETC`
Операция	1 → C
Эмуляция	`BIS #1,SR`
Описание	Устанавливается бит переноса C
Биты состояния	N: Не изменяется. Z: Не изменяется. C: Устанавливается. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Эмуляция BCD-вычитания:
	Регистр R5 вычитается из R6 по правилам BCD-арифметики. Полагаем, что R5 = #03987h, а R6 = #04137h.
	`DSUB ADD #06666h,R5 ; Преобразуем значения разрядов R5`
	`; от 0-9 к 6-0Fh.`
	`; R5 = 03987h + 06666h = 09FEDh`
	`INV R5 ; Инвертируем промежуточный результат`
	`; R5 = .NOT. R5 = 06012h`
	`SETC ; Инициализируем бит переноса`
	`DADD R5,R6 ; Эмулируем вычитание сложением:`
	`; (010000h - R5 - 1)`
	`; R6 = R6 + R5 + 1`
	`; R6 = 0150h`

* SETN Установка бита отрицательного значения

Синтаксис	`SETN`
Операция	1 → N
Эмуляция	`BIS #4,SR`
Описание	Устанавливается бит отрицательного значения N
Биты состояния	N: Устанавливается. Z: Не изменяется. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

* SETZ Установка бита нуля

Синтаксис	`SETZ`
Операция	1 → Z
Эмуляция	`BIS #2,SR`
Описание	Устанавливается бит нуля Z
Биты состояния	N: Не изменяется. Z: Устанавливается. C: Не изменяется. V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

SUB[.W], SUB.B
Вычитание двух операндов

Синтаксис	`SUB src,dst` или `SUB.W src,dst` `SUB.B src,dst`
Операция	(.not.src) + 1 + dst → dst или dst – src → dst
Описание	Операнд-источник вычитается из операнда-приёмника путём прибавления к последнему обратного кода операнда-источника плюс единица. Содержимое операнда-источника не изменяется. Результат сохраняется в операнде-приёмнике
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src ≤ dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда из положительного отрицателен или если результат вычитания положительного операнда из отрицательного положителен; иначе сбрасывается (нет переполнения).
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	16-битная константа 7654h вычитается из слова EDE. `SUB #7654h,&EDE ; EDE = EDE - 7654h`

Пример 2	Элемент таблицы, адресуемый регистром R5 (20-битный адрес), вычитается из регистра R7. Если результат равен нулю, то выполняется переход к метке `TONI`. Содержимое регистра R5 увеличивается на 2. Биты R7.19:16 = 0.
	`SUB @R5+,R7 ; Вычитаем элемент таблицы из R7, R5 + 2`
	`JZ TONI ; R7 = @R5 перед вычитанием`
	`... ; R7 <> @R5 перед вычитанием`

Пример 3	Байт CNT вычитается из байта, на который указывает R12. Адрес CNT лежит в пределах PC ±32 КБ. Регистр R12 может указывать на любую ячейку адресного пространства. `SUB.B CNT,0(R12) ; Вычитаем CNT из @R12`

SUBC[.W], SUBC.B
Вычитание двух операндов с учётом заёма

Синтаксис	`SUBC src,dst` или `SUBC.W src,dst` `SUBC.B src,dst`
Операция	(.not.src) + C + dst → dst или dst – (src – 1) + C → dst
Описание	Операнд-источник вычитается из операнда-приёмника путём прибавления к последнему обратного кода операнда-источника и значения бита переноса. Содержимое операнда-источника не изменяется. Результат сохраняется в операнде-приёмнике. Команда используется для обработки 32-, 48- и 64-битных значений
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда из положительного отрицателен или если результат вычитания положительного операнда из отрицательного положителен; иначе сбрасывается (нет переполнения).
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	16-битная константа 7654h вычитается из регистра R5 с учётом переноса от предыдущей команды. Биты R5.19:16 = 0. `SUBC.W #7654h,R5 ; R5 = R5 - 7654h - C`

Пример 2	48-битное число (3 слова), адресуемое регистром R5 (20-битный адрес), вычитается из 48-битного счётчика, адресуемого регистром R7. После завершения операции R5 указывает на следующее 48-битное число. Регистр R7 может адресовать любую ячейку в пределах адресного пространства.
	`SUB @R5+,0(R7) ; Вычитаем младшие слова. R5 + 2`
	`SUBC @R5+,2(R7) ; Вычитаем средние слова с учётом C. R5 + 2`
	`SUBC @R5+,4(R7) ; Вычитаем старшие слова с учётом C. R5 + 2`

Пример 3	Байт CNT вычитается из байта, на который указывает R12, с учётом переноса от предыдущей операции. Адрес CNT лежит в нижней 64-КБ области памяти. `SUBC.B &CNT,0(R12) ; Вычитаем CNT из @R12`

SWPB Перестановка байтов

Синтаксис	`SWP dst`
Операция	dst.15...8 ↔ dst.7...0
Описание	Старший и младший байты операнда-приёмника меняются местами. При использовании регистрового режима адресации биты 19:16 регистра сбрасываются
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	Старший и младший байты слова EDE (нижние 64 КБ) меняются местами.

```
MOV   #1234h,&EDE   ; 1234h -> EDE
SWP   &EDE          ; 3412h -> EDE
```

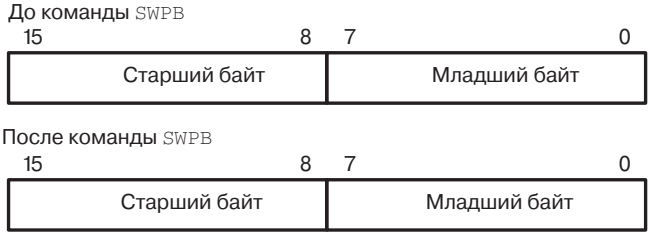


Рис. 4.42. Перестановка байтов слова в памяти.

Рис. 4.43. Перестановка байтов в регистре.

SXT Расширение знака

Синтаксис	`SXT dst`
Операция	dst.7 → dst.15:8 dst.7 → dst.19:8 (при использовании регистрового режима адресации)
Описание	Регистровый режим: знаковый бит младшего байта слова копируется в биты 19:8 регистра. Если Rdst.7 = 0, то после операции Rdst.19:8 = 000h. Если Rdst.7 = 1, то после операции Rdst.19:8 = FFFh. Прочие режимы: знаковый бит младшего байта слова копируется во все биты старшего байта операнда. Если dst.7 = 0, то после операции старший байт = 00h. Если dst.7 = 1, то после операции старший байт = FFh.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (C = .not.Z). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	8-битное число со знаком в EDE (нижние 64 КБ) приводится к 16-битному виду и складывается с 16-битным значением в регистре R7.
	`MOV.B &EDE,R5 ; EDE -> R5.00XXh`
	`SXT R5 ; Копируем знак младшего байта в биты R5.19:8`
	`ADD R5,R7 ; Складываем 16-битные значения`

Пример 2	8-битное число со знаком в EDE (PC ±32 КБ) приводится к 20-битному виду и складывается с 20-битным значением в регистре R7.
	`MOV.B EDE,R5 ; EDE -> R5.00XXh`
	`SXT R5 ; Копируем знак младшего байта в биты R5.19:8`
	`ADDA R5,R7 ; Складываем 20-битные значения`

* TST[.W], * TST.B
Проверка операнда (на ноль)

Синтаксис	`TST dst` или `TST.W dst` `TST.B dst`
Операция	dst + 0FFFFh + 1 dst + 0FFh + 1
Эмуляция	`CMP #0,dst` `CMP.B #0,dst`
Описание	Операнд-приёмник сравнивается с нулём и в соответствии с результатом изменяются биты состояния. Сам операнд остаётся неизменным
Биты состояния	N: Устанавливается, если операнд отрицателен, сбрасывается — если положителен. Z: Устанавливается, если операнд равен нулю, иначе сбрасывается. C: Устанавливается. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Проверяется содержимое регистра R7. Если оно отрицательное, то выполнение программы продолжается с метки `R7NEG`, если положительное и не равно нулю — с метки `R7POS`
	`TST R7 ; Проверяем R7`
	`JN R7NEG ; R7 < 0`
	`JZ R7ZERO ; R7 = 0`
	`R7POS ..... ; R7 > 0`
	`R7NEG ..... ; R7 < 0`
	`R7ZERO ..... ; R7 = 0`

Пример 2	Проверяется младший байт регистра R7. Если он отрицателен, то выполнение программы продолжается с метки `R7NEG`, если положителен и не равен нулю — с метки `R7POS`.
	`TST.B R7 ; Проверяем R7`
	`JN R7NEG ; R7 < 0`
	`JZ R7ZERO ; R7 = 0`
	`R7POS ..... ; R7 > 0`
	`R7NEG ..... ; R7 < 0`
	`R7ZERO ..... ; R7 = 0`

XOR[.W], XOR.B
«Исключающее ИЛИ» двух операндов

Синтаксис	`XOR src,dst` или `XOR.W src,dst` `XOR.B src,dst`
Операция	src .xor. dst → dst
Описание	Выполняется операция «Исключающее ИЛИ» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицателен (MSB = 1), иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (C = .not.Z). V: Устанавливается, если оба операнда отрицательные.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Установленные биты слова TONI (20-битное число) изменяют состояние соответствующих битов слова CNTR (16-битное число). Оба операнда расположены в нижней 64-КБ области. `XOR &TONI,&CNTR ; Переключаем биты CNTR`
Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для изменения состояния битов регистра R6. Биты R6.19:6 = 0. `XOR @R5,R6 ; Переключаем биты R6`

Пример 3	Сбрасываются в 0 те биты младшего байта R7, которые отличаются от соответствующих битов 1-байтовой переменной EDE. Биты R7.19:8 = 0. Адрес EDE лежит в диапазоне PC ±32 КБ.
	`XOR.B EDE,R7 ; Устанавливаем отличающиеся биты в 1`
	`INV.B R7 ; Инвертируем младший байт,`
	`; старший байт = 0`

4.6.3. Расширенные команды

Расширенный набор команд MSP430X позволяет этому ЦПУ в полной мере использовать доступное адресное пространство, определяемое 20-битной шиной адреса. Большинство команд MSP430X требуют ещё одного слова для хранения кода команды, так называемого слова расширения. Все адреса, смещения и непосредственные значения, которым предшествует слово расширения, являются 20-битными. Ниже приведены подробные описания всех расширенных команд MSP430X. Если команда не требует дополнительного слова, это явно указывается в её описании.

* ADCX.A, * ADCX[.W], * ADCX.B
Сложение переноса операндом

Синтаксис	`ADCX.A dst`
	`ADCX dst` или `ADXC.W dst`
	`ADCX.B dst`

Операция	dst + C → dst

Эмуляция	`ADDCX.A #0,dst`
	`ADDCX #0,dst`
	`ADDCX.B #0,dst`

Описание	Бит переноса (C) прибавляется к операнду-приёмнику. Предыдущее содержимое операнда теряется
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если dst изменился с 0FFFFh до 0000, иначе сбрасывается. Устанавливается, если dst изменился с 0FFh до 00, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	40-битный счётчик, на который указывают регистры R12 и R13, инкрементируется.
	`INCX.A @R12 ; Инкрементируем младшие 20 бит`
	`ADCX.A @R13 ; Прибавляем перенос к старшим 20 бит`

ADDX.A, ADDX[.W], ADDX.B
Сложение двух операндов

Синтаксис	`ADDX.A src,dst`
	`ADDX src,dst` или `ADDX.W src,dst`
	`ADDX.B src,dst`

Операция	src + dst → dst
Описание	Операнд-источник прибавляется к операнду-приёмнику. Предыдущее содержимое операнда-приёмника теряется. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос, иначе сбрасывается. V: Устанавливается, если результат операции над двумя положительными операндами отрицателен или если результат операции над двумя отрицательными операндами положителен; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	К 20-битному счётчику CNTR, расположенному в двух словах ОЗУ CNTR (младшие биты) и CNTR+2 (старшие биты), прибавляется десять. `ADDX.A #10,CNTR ; Прибавляем 10 к 20-битному счётчику`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), складывается с содержимым регистра R6. В случае переноса осуществляется переход к метке `TONI`.
	`ADDX.W @R5,R6 ; Прибавляем слово к R6, R6.19:16 = 0`
	`JC TONI ; Переходим в случае переноса`
	`... ; Нет переноса`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), складывается с содержимым регистра R6. При отсутствии переноса осуществляется переход к метке `TONI`. Указатель на таблицу автоматически инкрементируется.
	`ADDX.B @R5+,R6 ; Прибавляем байт к R6. R5 + 1. R6: 000xxh`
	`JNC TONI ; Переходим, если нет переноса`
	`... ; Был перенос`

Замечание. Для увеличения плотности кода и скорости выполнения программы используйте команду `ADDA` в следующих случаях: `ADDX.A Rsrc,Rdst` или `ADDX.A #imm20,Rdst`	

ADDCX.A, ADDCX[.W], ADDCX.B
Сложение двух операндов с учётом переноса

Синтаксис	`ADDCX.A src,dst`
	`ADDCX src,dst` или `ADDCX.W src,dst`
	`ADDCX.B src,dst`

Операция	src + dst + C → dst
Описание	Операнд-источник и бит переноса (C) прибавляются к операнду-приёмнику. Предыдущее содержимое операнда-приёмника теряется. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из старшего бита результата, иначе сбрасывается. V: Устанавливается, если результат операции над двумя положительными операндами отрицателен или если результат операции над двумя отрицательными операндами положителен; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	К 20-битному счётчику CNTR, расположенному в двух словах ОЗУ, прибавляется число 15 и бит переноса от предыдущей операции. `ADDCX.W #15,&CNTR ; Прибавляем 15+C к 20-битному счётчику`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), и бит переноса C складываются с содержимым регистра R6. В случае переноса осуществляется переход к метке `TONI`.
	`ADDCX.W @R5,R6 ; Прибавляем слово и C к R6`
	`JC TONI ; Переходим в случае переноса`
	`... ; Нет переноса`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), складывается с содержимым регистра R6. При отсутствии переноса осуществляется переход к метке `TONI`. Указатель на таблицу автоматически инкрементируется.
	`ADDCX.B @R5+,R6 ; Прибавляем байт и C к R6. R5 = R5 + 1.`
	`JNC TONI ; Переходим, если нет переноса`
	`... ; Был перенос`

ANDX.A, ANDX[.W], ANDX.B
«Логическое И» двух операндов

Синтаксис	`ANDX.A src,dst`
	`ANDX src,dst` или `AND.W src,dst`
	`ANDX.B src,dst`

Операция	src .and. dst → dst
Описание	Выполняется операция «Логическое И» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Устанавливается, если результат отрицателен (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается. C = (.not. Z) V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Содержимое регистра R5 (20-битное значение) используется в качестве битовой маски (AAA55h) для 20-битного слова TOM, расположенного в двух словах ОЗУ. Если результат равен нулю, то выполняется переход к метке `TONI`.
	`MOVA #AAA55h,R5 ; Загружаем 20-битную маску в R5`
	`ANDX.A R5,TOM ; TOM .and. R5 -> TOM`
	`JZ TONI ; Переходим, если результат нулевой`
	`... ; Результат не равен нулю`
	или короче
	`ANDX.A #AAA55h,TOM ; TOM .and. AAA55h -> TOM`
	`JZ TONI ; Переходим, если результат нулевой`

Пример 2	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), логически перемножается с содержимым регистра R6. После выборки байта R5 инкрементируется. Биты R6.19:8 = 0.
	`ANDX.B @R5+,R6 ; «Логическое И» между байтом из таблицы`
	`; и R6. R5 + 1.`

BICX.A, BICX[.W], BICX.B
Очистка битов операнда

Синтаксис	`BICX.A src,dst`
	`BICX src,dst` или `BICX.W src,dst`
	`BICX.B src,dst`

Операция	(.not. src) .and. dst → dst
Описание	Выполняется операция «Логическое И» между инвертированным значением операнда-источника и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Не изменяется. Z: Не изменяется. C: Не изменяется . V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Сбрасываются биты 19:15 регистра R5 (20-битное значение). `BICX.A #0F8000h,R5 ; Сбрасываем биты R5.19:15`
Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для очистки битов регистра R7. Биты R7.19:16 = 0. `BIXC.W @R5,R7 ; Сбрасываем биты в R7`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), используется для очистки битов порта Port1.
	`BICX.B @R5,&P1OUT ; Очищаем биты порта 1,`
	`; которые установлены в @R5`

BISX.A, BISX[.W], BISX.B
Установка битов операнда

Синтаксис	`BISX.A src,dst`
	`BISX src,dst` или `BISX.W src,dst`
	`BISX.B src,dst`

Операция	src .or. dst → dst
Описание	Выполняется операция «Логическое ИЛИ» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Не изменяется. Z: Не изменяется. C: Не изменяется . V: Не изменяется.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Устанавливаются биты 15 и 16 регистра R5 (20-битное значение). `BISX.A #018000h,R5 ; Устанавливаем биты R5.16:15`
Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для установки битов регистра R7. `BISX.W @R5,R7 ; Устанавливаем биты R7`
Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), используется для установки битов порта Port1. `BISX.B @R5,&P1OUT ; Устанавливаем биты порта 1`

BITX.A, BITX[.W], BITX.B
Проверка битов операнда

Синтаксис	`BITX.A src,dst`
	`BITX src,dst` или `BITX.W src,dst`
	`BITX.B src,dst`

Операция	src .and. dst
Описание	Выполняется операция «Логическое И» между операндом-источником и операндом-приёмником. Результат операции влияет только на биты состояния регистра SR. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается. C = (.not. Z) V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Если установлен бит 16 или 15 регистра R5 (20-битное значение) или оба этих бита, то выполняется переход к метке `TONI`.
	`BITX.A #018000h,R5 ; Проверяем биты R5.16:15`
	`JNZ TONI ; Хотя бы один бит установлен`
	`... ; Оба бита сброшены`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для проверки битов регистра R7. Если установлен хотя бы один бит, то выполняется переход к метке `TONI`.
	`BITX.W @R5,R7 ; Проверяем биты R7`
	`JC TONI ; Хотя бы один бит установлен`
	`... ; Все биты сброшены`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), используется для проверки битов порта Port1. Если все биты сброшены, то выполняется переход к метке `TONI`. Затем указатель устанавливается на следующий байт таблицы.
	`BITX.B @R5+,&P1OUT ; Проверяем биты порта 1, R5 = R5 + 1`
	`JNC TONI ; Все требуемые биты сброшены`
	`... ; Хотя бы один бит установлен`

* CLRX.A, * CLRX[.W], * CLRX.B
Очистка операнда

Синтаксис	`CLRX.A dst`
	`CLRX dst` или `CLRX.W dst`
	`CLRX.B dst`

Операция	0 → dst

Эмуляция	`MOVX.A #0,dst`
	`MOVX #0,dst`
	`MOVX.B #0,dst`

Описание	Операнд-приёмник обнуляется
Биты состояния	Биты состояния не изменяются
Пример	Обнуляется 20-битное слово TONI в ОЗУ. `CLRX.A TONI ; 0 -> TONI`

CMPX.A, CMPX[.W], CMPX.B
Сравнение двух операндов

Синтаксис	`CMPX.A src,dst`
	`CMPX src,dst` или `CMPX.W src,dst`
	`CMPX.B src,dst`

Операция	.not.src + 1 + dst или (dst – src)
Описание	Операнд-источник вычитается из операнда приёмника. Для выполнения этой операции обратный код операнда-источника плюс 1 складывается с операндом-приёмником. Операция влияет только на биты состояния регистра SR
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src ≤ dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если был перенос из MSB результата, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда-источника из положительного операнда-приёмника отрицателен или результат вычитания положительного операнда-источника из отрицательного операнда-приёмника положителен, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Слово EDE сравнивается с 20-битной константой 18000h. В случае равенства выполняется переход к метке `TONI`.
	`CMP #18000h,EDE ; Сравниваем EDE с 18000h`
	`JEQ TONI ; EDE = 18000h`
	`...`

Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), сравнивается с содержимым регистра R7. Если число в R7 меньше табличного значения (сравниваются 16-битные числа со знаком), то выполняется переход к метке `TONI`.
	`CMPX.W @R5,R7 ; Сравниваем два числа со знаком`
	`JL TONI ; R7 < @R5`
	`... ; R7 >= @R5`

Пример 3	Байт из таблицы, адресуемый регистром R5 (20-битный адрес), сравнивается с входным значением порта Port1. Если значения равны, то выполняется переход к метке `TONI`. Затем указатель устанавливается на следующий байт таблицы.
	`CMPX.B @R5+,&P1IN ; Сравниваем биты порта с таблицей,`
	`; R5 = R5 + 1`
	`JEQ TONI ; Значения одинаковы`
	`... ; Значения разные`

Замечание. Для увеличения плотности кода и скорости выполнения программы используйте команду `CMPA` в следующих случаях: `CMPA Rsrc,Rdst` или `CMPA #imm20,Rdst`	

* DADCX.A, * DADCX[.W], * DADCX.B
Сложение переноса с операндом (BCD-арифметика)

Синтаксис	`DADCX.B dst`
	`DADCX dst` или `DADCX.W dst`
	`DADCX.B dst`

Операция	dst + C → dst (BCD-арифметика)

Эмуляция	`DADDX.A #0,dst`
	`DADDX #0,dst`
	`DADDX.B #0,dst`

Описание	Бит переноса (C) прибавляется к операнду-приёмнику по правилам двоично-десятичной арифметики
Биты состояния	N: Устанавливается, если MSB результата равен 1 (20-битное значение > 79999h, 16-битное значение > 7999h, 8-битное значение > 79h), иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат слишком велик (20-битное значение > 99999h, 16-битное значение > 9999h, 8-битное значение > 99h), иначе сбрасывается. V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	40-битный BCD-счётчик, находящийся в регистрах R12 и R13, инкрементируется.
	`DADDX.A #1,0(R12) ; Инкрементируем младшие разряды`
	`DADCX.A 0(R13) ; Прибавляем перенос к старшим разрядам`

DADDX.A, DADDX[.W], DADDX.B
Сложение двух операндов с учётом переноса (BCD-арифметика)

Синтаксис	`DADDX.A src,dst`
	`DADDX src,dst` или `DADD.W src,dst`
	`DADDX.B src,dst`

Операция	src + dst + C → dst (BCD-арифметика)
Описание	Операнды интерпретируются как 2-, 4- и 5-разрядные (суффиксы .B, .W и .A соответственно) положительные BCD-числа. Операнд-источник и бит переноса (C) прибавляются к операнду-приёмнику по правилам двоично-десятичной арифметики. Содержимое операнда-источника не изменяется. Предыдущее содержимое операнда-приёмника теряется. Для не BCD-чисел результат операции не определён. Оба операнда могут находиться в любом месте адресного пространства
Биты состояния	N: Устанавливается, если MSB результата равен 1 (20-битное значение > 79999h, 16-битное значение > 7999h, 8-битное значение > 79h), иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат слишком велик (20-битное значение > 99999h, 16-битное значение > 9999h, 8-битное значение > 99h), иначе сбрасывается. V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Двоично-десятичное число 10 прибавляется к 20-битному BCD-счётчику DECCNTR.
	`DADDX.A #10h,&DECCNTR ; Прибавляем 10 к 5-разрядному`
	`; BCD-счётчику`

Пример 2	8-разрядное двоично-десятичное число, расположенное в ОЗУ по адресам BCD и BCD+2, прибавляется к 8-разрядному числу, находящемуся в R4 и R5 (старшие разряды — BCD+2 и R5).
	`CLRC ; Сбрасываем бит переноса`
	`DADDX.W BCD,R4 ; Складываем младшие разряды`
	`DADDX.W BCD+2,R5 ; Складываем старшие разряды с учётом переноса`
	`JC OVERFLOW ; Результат > 9999'9999, переходим`
	`; к обработчику ошибок`
	`... ; Результат нормальный`

Пример 3	2-разрядное двоично-десятичное число, расположенное в слове BCD (20-битный адрес), прибавляется к 2-разрядному числу, находящемуся в R4.
	`CLRC ; Сбрасываем бит переноса`
	`DADDX.B BCD,R4 ; Складываем BCD и R4. R4: 000ddh`

* DECX.A, * DECX[.W], * DECX.B
Декрементирование операнда

Синтаксис	`DECX.A dst`
	`DECX dst` или `DECX.W dst`
	`DECX.B dst`

Операция	dst – 1 → dst

Эмуляция	`SUBX.A #1,dst`
	`SUBX #1,dst`
	`SUBX.B #1,dst`

Описание	Значение операнда-приёмника уменьшается на 1. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 1, иначе сбрасывается. C: Сбрасывается, если dst содержал 0, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	20-битное значение TONI уменьшается на 1. `DECX.A TONI ; Декрементируем TONI`

* DECDX.A, * DECDX[.W], * DECDX.B
Уменьшение операнда на 2

Синтаксис	`DECDX.A dst`
	`DECDX dst` или `DECDX.W dst`
	`DECDX.B dst`

Операция	dst – 2 → dst

Эмуляция	`SUBX.A #2,dst`
	`SUBX #2,dst`
	`SUBX.B #2,dst`

Описание	Значение операнда-приёмника уменьшается на 2. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 2, иначе сбрасывается. C: Сбрасывается, если dst содержал 0 или 1, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	20-битное значение TONI уменьшается на 2. `DECDX.A TONI ; Уменьшаем TONI на 2`

* INCX.A, * INCX[.W], * INCX.B
Инкрементирование операнда

Синтаксис	`INCX.A dst`
	`INCX dst` или `INCX.W dst`
	`INCX.B dst`

Операция	dst + 1 → dst

Эмуляция	`ADDX.A #1,dst`
	`ADDX #1,dst`
	`ADDX.B #1,dst`

Описание	Значение операнда-приёмника увеличивается на 1. Предыдущее содержимое операнда-приёмника теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 07FFFh, иначе сбрасывается. Устанавливается, если dst содержал 07Fh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	20-битное значение TONI увеличивается на 1. `INCX.A TONI ; Инкрементируем TONI`

* INCDX.A, * INCDX[.W], * INCDX.B
Увеличение операнда на 2

Синтаксис	`INCDX.A dst`
	`INCDX dst` или `INCDX.W dst`
	`INCDX.B dst`

Операция	dst + 2 → dst

Эмуляция	`ADDX.A #2,dst`
	`ADDX #2,dst`
	`ADDX.B #2,dst`

Описание	Значение операнда-приёмника увеличивается на 2. Предыдущее содержимое операнда-приёмника теряется.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFEh, иначе сбрасывается. Устанавливается, если dst содержал 0FFFEh, иначе сбрасывается. Устанавливается, если dst содержал 0FEh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFFEh или 0FFFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFFEh или 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FEh или 0FFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFFEh или 07FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 07FFEh или 07FFFh, иначе сбрасывается. Устанавливается, если dst содержал 07FEh или 07Fh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	Байт LEO в ОЗУ увеличивается на 2. Счётчик команд указывает на верхнюю область памяти. `INCDX.B LEO ; Увеличиваем LEO на 2`

* INVX.A, * INVX[.W], * INVX.B
Инвертирование операнда

Синтаксис	`INVX.A dst`
	`INVX dst` или `INVX.W dst`
	`INVX.B dst`

Операция	.NOT.dst → dst

Эмуляция	`XORX.A #0FFFFFh,dst`
	`XORX #0FFFFh,dst`
	`XORX.B #0FFh,dst`

Описание	Значение операнда-приёмника инвертируется. Предыдущее содержимое теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFFFh, иначе сбрасывается. Устанавливается, если dst содержал 0FFh, иначе сбрасывается. C: Устанавливается, если результат не равен нулю, иначе сбрасывается (C = .NOT. Z). V: Устанавливается, если операнд имел отрицательное значение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Вычисляется дополнительный код 20-битного содержимого регистра R5.
	`INVX.A R5 ; Инвертируем R5`
	`INCX.A R5 ; В R5 - дополнительный код`

Пример 2	Вычисляется дополнительный код байта LEO в ОЗУ. Счётчик команд указывает на верхнюю область памяти.
	`INVX.B LEO ; Инвертируем LEO`
	`INCX.B LEO ; В LEO - дополнительный код`

MOVX.A, MOVX[.W], MOVX.B
Пересылка операнда

Синтаксис	`MOVX.A src,dst`
	`MOVX src,dst` или `MOVX.W src,dst`
	`MOVX.B src,dst`

Операция	src → dst
Описание	Операнд-источник пересылается в операнд-приёмник. Содержимое операнда-источника не изменяется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	20-битная константа 18000h загружается в слово EDE. `MOVX.A #018000h,&EDE ; 18000h -> EDE`

Пример 2	Содержимое таблицы EDE (2-байтные элементы, 20-битная адресация) копируется в таблицу TOM. Размер таблиц — 030h слов.
	`MOVA #EDE,R10 ; Инициализируем указатель`
	`Loop MOVX.W @R10+,TOM-EDE-2(R10) ; Используем указатель в R10`
	`; для обеих таблиц. R10 + 2`
	`CMPA #EDE+60h,R10 ; Достигли конца таблицы?`
	`JLO Loop ; Ещё нет`
	`... ; Копирование завершено`

Пример 3	Содержимое таблицы EDE (1-байтные элементы, 20-битная адресация) копируется в таблицу TOM. Размер таблиц — 020h байтов.
	`MOVA #EDE,R10 ; Инициализируем указатель`
	`MOV #020h,R9 ; Инициализируем счетчик`
	`Loop MOVX.B @R10+,TOM-EDE-1(R10) ; Используем указатель в R10`
	`; для обеих таблиц. R10 + 1`
	`DEC R9 ; Декрементируем счетчик`
	`JNZ Loop ; Счётчик ≠ 0, продолжаем`
	`; копирование`
	`... ; Копирование завершено`

Из 28 возможных сочетаний режимов адресации операндов в десяти случаях вместо команды `MOVX.A` можно использовать команду `MOVA`. При этом экономится два слова памяти и несколько тактов. Речь идёт о сочетаниях:

```
MOVX.A Rsrc,Rdst      MOVA Rsrc,Rdst      ; Регистровый/Регистровый
MOVX.A #imm20,Rdst    MOVA #imm20,Rdst    ; Непосредственный/Регистровый
MOVX.A &abs20,Rdst    MOVA &abs20,Rdst    ; Абсолютный/Регистровый
MOVX.A @Rsrc,Rdst     MOVA @Rsrc,Rdst     ; Косвенный/Регистровый
MOVX.A @Rsrc+,Rdst    MOVA @Rsrc+,Rdst    ; Косвенный с автоинкр./Регистровый
MOVX.A Rsrc,&abs20    MOVA Rsrc,&abs20    ; Регистровый/Абсолютный
```

Следующие четыре замены применимы только в том случае, если для адресации операндов можно использовать 16-битные смещения.

```
MOVX.A z20(Rsrc),Rdst    MOVA z16(Rsrc),Rdst     ; Индексный/Регистровый
MOVX.A Rsrc,z20(Rdst)    MOVA Rsrc,z16(Rdst)     ; Регистровый/Индексный
MOVX.A symb20,Rdst       MOVA symb16,Rdst        ; Относительный/Регистровый
MOVX.A Rsrc,symb20       MOVA Rsrc,symb16        ; Регистровый/Относительный
```

POPM.A, POPM[.W]
Извлечение *n* регистров ЦПУ из стека

Синтаксис	`POPM.A #n,Rdst` $1 \le n \le 16$ `POPM.W #n,Rdst` или `POPM #n,Rdst` $1 \le n \le 16$
Операция	`POPM.A`: Восстанавливает из стека 20-битные значения заданных регистров ЦПУ. Для каждого значения, извлечённого из стека, указатель стека SP увеличивается на 4 (2 слова на регистр). `POPM.W`: Восстанавливает из стека 16-битные значения заданных регистров ЦПУ. Для каждого значения, извлечённого из стека, указатель стека SP увеличивается на 2 (одно слово на регистр).
Описание	`POPM.A`: Содержимое стека (20-битные значения) пересылается в регистры ЦПУ, начиная с регистра (Rdst – n + 1). По окончании операции указатель стека увеличивается на ($n \times 4$). `POPM.W`: Содержимое стека (16-битные значения) пересылается в регистры ЦПУ, начиная с регистра (Rdst – n + 1). По окончании операции указатель стека увеличивается на ($n \times 2$). Старшие биты (Rdst.19:16) регистров сбрасываются. **Примечание.** Команда не использует слово расширения.
Биты состояния	Биты состояния не изменяются, если регистр SR не является операндом
Биты режима	OSCOFF, CPUOFF и GIE не изменяются, если регистр SR не является операндом
Пример 1	20-битное содержимое регистров R9...R13 восстанавливается из стека. `POPM.A #5,R13 ; Извлекаем R9, R10, R11, R12, R13`
Пример 2	16-битное содержимое регистров R9...R13 восстанавливается из стека. `POPM.W #5,R13 ; Извлекаем R9, R10, R11, R12, R13`

PUSHM.A, PUSHM[.W]
Сохранение *n* регистров ЦПУ в стеке

Синтаксис	`PUSHM.A #n,Rdst` $1 \le n \le 16$ `PUSHM.W #n,Rdst` или `PUSHM #n,Rdst` $1 \le n \le 16$
Операция	`PUSHM.A`: Сохраняет 20-битные значения заданных регистров ЦПУ в стеке. Для каждого значения, сохранённого в стеке, указатель стека SP уменьшается на 4 (2 слова на регистр). Старшие биты сохраняются первыми (по старшему адресу). `POPM.W`: Сохраняет 16-битные значения заданных регистров ЦПУ в стеке. Для каждого значения, сохранённого в стеке, указатель стека SP уменьшается на 2 (одно слово на регистр).
Описание	`PUSHM.A`: Содержимое *n* регистров ЦПУ, начиная с регистра (Rdst – *n* + 1), сохраняется в стеке. По окончании операции указатель стека уменьшается на ($n \times 4$). Содержимое сохраняемых регистров (Rn.19:0) остаётся неизменным. `PUSHM.W`: Содержимое *n* регистров ЦПУ, начиная с регистра (Rdst – *n* + 1), сохраняется в стеке. По окончании операции указатель стека уменьшается на ($n \times 2$). Содержимое сохраняемых регистров (Rn.19:0) остаётся неизменным. **Примечание.** Команда не использует слово расширения.
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	20-битное содержимое регистров R9...R13 сохраняется в стеке. `PUSHM.A #5,R13 ; Сохраняем R9, R10, R11, R12, R13`
Пример 2	16-битное содержимое регистров R9...R13 сохраняется в стеке. `PUSHM.W #5,R13 ; Сохраняем R9, R10, R11, R12, R13`

* POPX.A, * POPX[.W], * POPX.B
Извлечение операнда из стека

Синтаксис	`POPX.A dst`
	`POPX dst` или `POPX.W dst`
	`POPX.B dst`

Операция	Извлекает 8/16/20-битное значение из стека в операнд-приёмник. Используется 20-битная адресация. Указатель стека SP увеличивается на 2 (8- и 16-битные операнды) или 4 (20-битный операнд)

Эмуляция	`MOVX.A @SP+,dst`
	`MOVX @SP+,dst`
	`MOVX.B @SP+,dst`

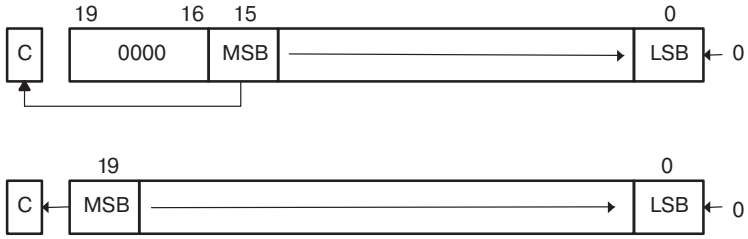
Описание	Последний элемент стека (TOS) извлекается в операнд-приёмник. Допускается использование регистрового, индексного, относительного и абсолютного режимов адресации. После этого указатель стека увеличивается на 2 или 4
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	16-битное число извлекается из стека по адресу EDE. `POPX.W &EDE ; Пишем слово по адресу EDE`
Пример 2	20-битное число извлекается из стека в регистр R9. `POPX.A R9 ; Пишем 20-битное число в R9`

PUSHX.A, PUSHX[.W], PUSHX.B
Сохранение операнда в стеке

Синтаксис	`PUSHX.A src`
	`PUSHX src` или `PUSHX.W src`
	`PUSHX.B src`

Операция	Сохраняет 8/16/20-битное значение операнда-источника в стеке. Используется 20-битная адресация. Перед записью значения указатель стека SP уменьшается на 2 (8- и 16-битные операнды) или 4 (20-битный операнд)
Описание	20-битный указатель стека SP уменьшается на 2 (8- и 16-битные операнды) или 4 (20-битный операнд), после чего содержимое операнда-источника помещается в ОЗУ по адресу, определяемому указателем. Для адресации операнда можно использовать любой из семи режимов
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Байт, находящийся по адресу EDE (20-битное значение), сохраняется в стеке. `PUSHX.B &EDE ; Сохраняем байт с адресом EDE`
Пример 2	Содержимое регистра R9 (20-битное значение) сохраняется в стеке. `PUSHX.A R9 ; Сохраняем 20-битное содержимое R9`

RLAM.A, RLAM[.W]
Арифметический сдвиг влево содержимого регистра ЦПУ

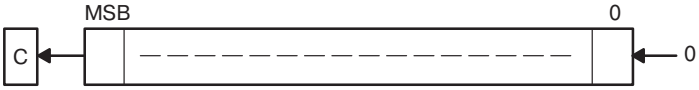
Синтаксис	`RLAM.A #n,Rdst` (1 ≤ n ≤ 4) `RLAM.W #n,Rdst` или `RLAM #n,Rdst` (1 ≤ n ≤ 4)
Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← 0
Описание	Содержимое операнда-приёмника сдвигается влево на один, два, три или четыре бита, как показано на **Рис. 4.44**. По сути, команда `RLAM` выполняет операцию умножения (знакового и беззнакового) на 2, 4, 8 или 16. Двухбайтная модификация команды (`RLAM.W`) сбрасывает биты Rdst.19:16.  ***Рис. 4.44.*** Арифметический сдвиг влево: команды `RLAM[.W]` и `RLAM.A`. **Примечание.** Команда не использует слово расширения.
Биты состояния	N: Устанавливается, если результат отрицательный .A: Rdst.19 = 1, сбрасывается, если Rdst.19 = 0; .W: Rdst.15 = 1, сбрасывается, если Rdst.15 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB (n = 1), MSB–1 (n = 2), MSB–2 (n = 3), MSB–3 (n = 4). V: Не определён.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	20-битное содержимое регистра R5 сдвигается влево на три бита. Эта операция эквивалентна умножению на 8. `RLAM.A #3,R5 ; R5 = R5 × 8`

* RLAX.A, * RLAX[.W], * RLAX.B
Арифметический сдвиг операнда влево

Синтаксис	`RLAX.A dst`
	`RLAX dst` или `RLAX.W dst`
	`RLAX.B dst`

Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← 0

Эмуляция	`ADDX.W dst,dst`
	`ADDX dst,dst`
	`ADDX.B dst,dst`

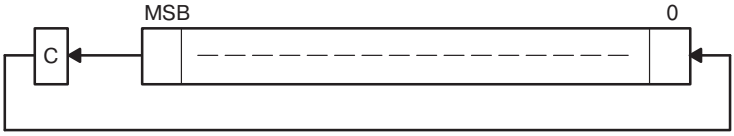
Описание	Содержимое операнда-приёмника сдвигается влево на один бит, как показано на **Рис. 4.45**. Старший бит операнда помещается в бит переноса (C), а младший бит обнуляется. По сути, команда `RLA` выполняет операцию знакового умножения на 2.  ***Рис. 4.45.*** Операнд-приёмник — арифметический сдвиг влево.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB. V: Устанавливается, если произошло переполнение: начальное значение 040000h ≤ dst < 0C0000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 04000h ≤ dst < 0C000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 040h ≤ dst < 0C0h; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	20-битное содержимое регистра R7 умножается на 2. `RLAX.A R7 ; Сдвигаем R7 на 1 бит влево (×2)`

* RLCX.A, * RLCX[.W], * RLCX.B
Сдвиг операнда влево через перенос

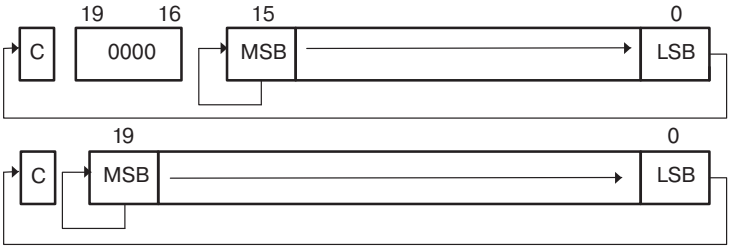
Синтаксис	`RLCX.A dst`
	`RLCX dst` или `RLCX.W dst`
	`RLCX.B dst`

Операция	C ← MSB ← MSB–1 ... LSB+1 ← LSB ← C

Эмуляция	`ADDCX.A dst,dst`
	`ADDCX dst,dst`
	`ADDCX.B dst,dst`

Описание	Содержимое операнда-приёмника сдвигается влево на один бит, как показано на **Рис. 4.46**. Бит переноса (C) вдвигается в младший бит (LSB) операнда, а старший бит операнда (MSB) выдвигается в бит переноса.  ***Рис. 4.46.*** Операнд-приёмник — сдвиг влево через перенос.
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из MSB. V: Устанавливается, если произошло переполнение: начальное значение 040000h ≤ dst < 0C0000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 04000h ≤ dst < 0C000h; иначе сбрасывается. Устанавливается, если произошло переполнение: начальное значение 040h ≤ dst < 0C0h; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	20-битное содержимое регистра R5 сдвигается на 1 бит влево. `RLCX.A R5 ; (R5 × 2) + C -> R5`
Пример 2	Содержимое байта ОЗУ LEO сдвигается на 1 бит влево. `RLCX.B LEO ; MEM(LEO) × 2 + C -> MEM(LEO)`

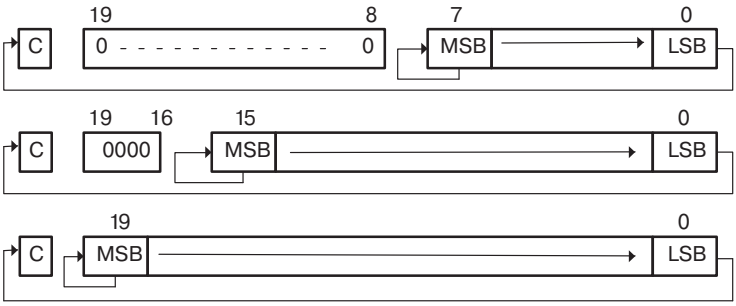
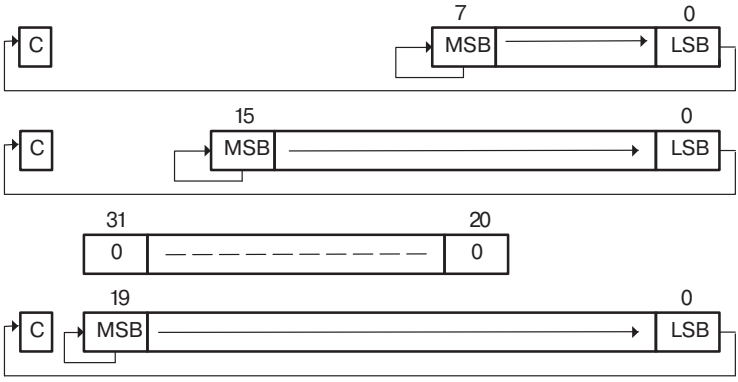
RRAM.A, RRAM[.W]
Арифметический сдвиг вправо содержимого регистра ЦПУ

Синтаксис	`RRAM.A #n,Rdst` ($1 \le n \le 4$) `RRAM.W #n,Rdst` или `RRAM #n,Rdst` ($1 \le n \le 4$)
Операция	MSB → MSB, MSB → MSB–1, ... LSB+1 → LSB, LSB → 0
Описание	Содержимое операнда-приёмника сдвигается вправо на на один, два, три или четыре бита, как показано на **Рис. 4.47**. Состояние MSB (знак числа) остаётся неизменным. По сути, команда `RLA` выполняет операцию знакового деления на 2, 4, 8 и 16. Старший бит содержимого (MSB) выдвигается в бит MSB–1 (значение MSB остаётся неизменным). Бит LSB+1 выдвигается в младший бит (LSB), а бит LSB выдвигается в бит переноса C. Двухбайтная модификация команды (`RRAM.W`) сбрасывает биты Rdst.19:16.  ***Рис. 4.47.*** Арифметический сдвиг вправо: команды `RRAM[.W]` и `RRAM.A`. **Примечание.** Команда не использует слово расширения.
Биты состояния	N: Устанавливается, если результат отрицательный .A: Rdst.19 = 1, сбрасывается, если Rdst.19 = 0; .W: Rdst.15 = 1, сбрасывается, если Rdst.15 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB ($n = 1$), LSB+1 ($n = 2$), LSB+2 ($n = 3$), LSB+3 ($n = 4$). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	20-битное число со знаком, находящееся в регистре R5, сдвигается вправо на два бита. `RRAM.A #2,R5 ; R5/4 -> R5`

Пример 2	20-битное число со знаком, находящееся в регистре R15, умножается на 0.75.
	`PUSHM.A R15 ; Временно сохраняем R15 в стеке`
	`RRAM.A R15 ; R15 × 0.5 -> R15`
	`ADDX.A @SP+,R15 ; R15 × 0.5 + R15 = 1.5 × R15 -> R15`
	`RRAM.A R15 ; (1.5 × R15) × 0.5 = 0.75 × R15 ->R15`

* RRAX.A, * RRAX[.W], * RRAX.B
Арифметический сдвиг операнда вправо

Синтаксис	`RRAX.A Rdst`
	`RRAX.W Rdst`
	`RRAX Rdst`
	`RRAX.B Rdst`
	
	`RRAX.A dst`
	`RRAX.W dst` или `RRAX dst`
	`RRAX.B dst`

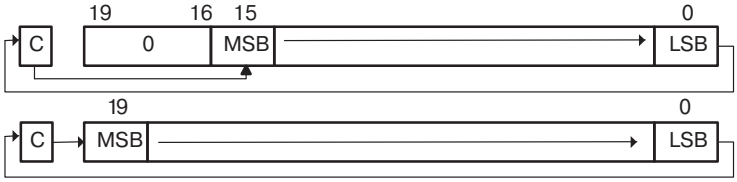
Операция	MSB → MSB, MSB → MSB–1, ... LSB+1 → LSB, LSB → 0
Описание	Регистровый режим адресации: содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.48**. Состояние MSB (знак числа) остаётся неизменным. Команда `RRAX.W` сбрасывает биты Rdst.19:16, команда `RRAX.B` сбрасывает биты Rdst.19:8. Старший бит остаётся неизменным, а младший бит (LSB) выдвигается в бит переноса C. По сути, команда `RRAX` выполняет операцию знакового деления на 2.  ***Рис. 4.48.*** Арифметический сдвиг вправо: команда `RRAX(.B,.A)`. Регистровая адресация. Прочие режимы адресации: содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.49**. Состояние MSB (знак числа) остаётся неизменным, а младший бит (LSB) выдвигается в бит переноса C. По сути, команда `RRAX` выполняет операцию знакового деления на 2. Все режимы адресации, за исключением непосредственного режима, могут использоваться в пределах всего адресного пространства.  ***Рис. 4.49.*** Арифметический сдвиг вправо: команда `RRAX(.B,.A)`. Не регистровая адресация.

Биты состояния	N: Устанавливается, если результат отрицательный .A: dst.19 = 1, сбрасывается, если dst.19 = 0; .W: dst.15 = 1, сбрасывается, если dst.15 = 0; .B: dst.7 = 1, сбрасывается, если dst.7 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	20-битное число со знаком, находящееся в регистре R5, сдвигается вправо на четыре бита.
	`RPT #4`
	`RRAX.A R5 ; R5/16 -> R5`

Пример 2	Содержимое байта EDE умножается на 0.5. `RRAX.B &EDE ; EDE/2 -> EDE`

RRCM.A, RRCM[.W]
Сдвиг вправо через перенос содержимого регистра ЦПУ

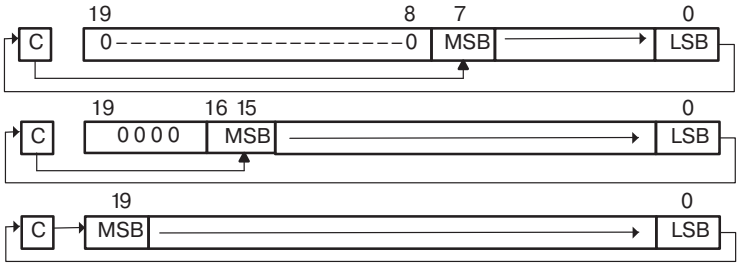
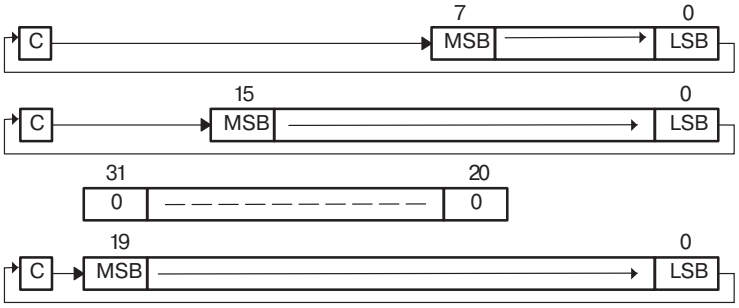
Синтаксис	`RRAM.A #n,Rdst` ($1 \le n \le 4$) `RRAM.W #n,Rdst` или `RRAM #n,Rdst` ($1 \le n \le 4$)
Операция	MSB → MSB, MSB → MSB–1, ... LSB+1 → LSB, LSB → 0
Описание	Содержимое операнда-приёмника сдвигается вправо на на один, два, три или четыре бита, как показано на **Рис. 4.50**. Бит переноса (C) вдвигается в старший бит (MSB) операнда, а младший бит операнда (LSB) выдвигается в бит переноса (C). Двухбайтная модификация команды (`RRAM.W`) сбрасывает биты Rdst.19:16.  ***Рис. 4.50.*** Сдвиг вправо через перенос: команды `RRCM[.W]` и `RRCM.A`. **Примечание.** Команда не использует слово расширения.
Биты состояния	N: Устанавливается, если результат отрицательный .A: Rdst.19 = 1, сбрасывается, если Rdst.19 = 0; .W: Rdst.15 = 1, сбрасывается, если Rdst.15 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB ($n = 1$), LSB+1 ($n = 2$), LSB+2 ($n = 3$), LSB+3 ($n = 4$). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	20-битное число со знаком, находящееся в регистре R5, сдвигается вправо на три бита. В бит MSB-2 загружается 1.
	`SETC ; Инициализируем бит переноса`
	`RRCM.A #3,R5 ; R5 = R5 >> 3 + 20000h`

Пример 2	Двухбайтное содержимое регистра R6 сдвигается вправо на два бита. В MSB загружается LSB. В бит MSB–1 загружается значение бита переноса. `RRCM.W #2,R6 ; R6 = R6 >> 2. R6.19:16 = 0`

RRCX.A, RRCX[.W], RRCX.B
Сдвиг операнда вправо через перенос

Синтаксис	`RRCX.A Rdst`
	`RRCX.W Rdst`
	`RRCX Rdst`
	`RRCX.B Rdst`
	
	`RRCX.A dst`
	`RRCX.W dst` или `RRCX dst`
	`RRCX.B dst`

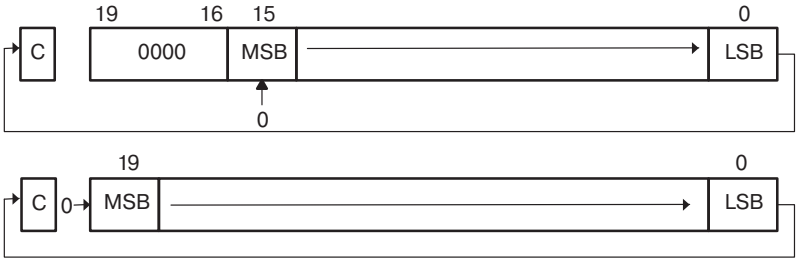
Операция	C → MSB → MSB–1 ... LSB+1 → LSB → C
Описание	Регистровый режим адресации: содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.51**. Команда `RRCX.W` сбрасывает биты Rdst.19:16, команда `RRCX.B` сбрасывает биты Rdst.19:8. Бит переноса (C) вдвигается в MSB операнда, а LSB операнда выдвигается в бит переноса. Прочие режимы адресации: содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.52**. Бит переноса (C) вдвигается в MSB операнда, а LSB операнда выдвигается в бит переноса. Все режимы адресации, за исключением непосредственного режима, могут использоваться в пределах всего адресного пространства.  ***Рис. 4.51.*** Сдвиг вправо через перенос: команда `RRCX(.B,.A)`. Регистровая адресация.  ***Рис. 4.52.*** Сдвиг вправо через перенос: команда `RRCX(.B,.A)`. Не регистровая адресация.
Биты состояния	N: Устанавливается, если результат отрицательный .A: dst.19 = 1, сбрасывается, если dst.19 = 0; .W: dst.15 = 1, сбрасывается, если dst.15 = 0; .B: dst.7 = 1, сбрасывается, если dst.7 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.

Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	20-битное значение по адресу EDE сдвигается на 1 бит вправо. В MSB загружается 1.
	`SETC ; Инициализируем бит переноса`
	`RRCX.A EDE ; EDE = EDE >> 1 + 80000h`

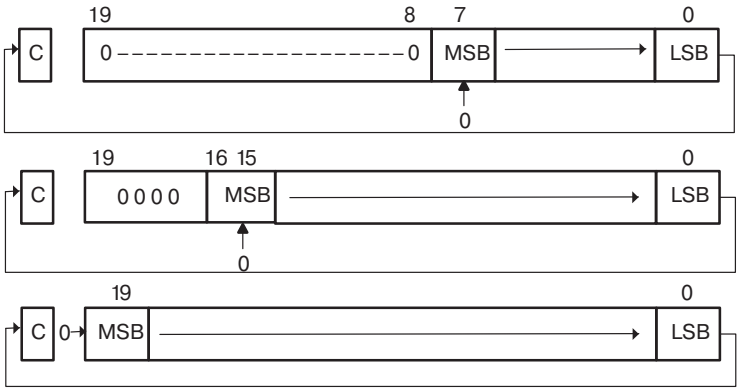
Пример 2	Слово, находящееся в регистре R6, сдвигается на 12 битов вправо.
	`RPT #12`
	`RRCX.W R6 ; R6 = R6 >> 12. Биты R6.19:16 = 0`

RRUM.A, RRUM[.W]
Сдвиг вправо беззнакового содержимого регистра ЦПУ

Синтаксис	`RRUM.A #n,Rdst` ($1 \le n \le 4$) `RRUM.W #n,Rdst` или `RRUM #n,Rdst` ($1 \le n \le 4$)
Операция	0 → MSB → MSB → MSB–1 → ... LSB+1 → LSB, LSB → C
Описание	Содержимое операнда-приёмника сдвигается вправо на на один, два, три или четыре бита, как показано на **Рис. 4.53**. В MSB операнда вдвигается 0, а младший бит операнда (LSB) выдвигается в бит переноса C. По сути, команда `RRUM` выполняет операцию беззнакового деления на 2, 4, 6 или 16. Двухбайтная модификация команды (`RRUM.W`) сбрасывает биты Rdst.19:16.  ***Рис. 4.53.*** Сдвиг вправо беззнакового операнда: команды `RRUM[.W]` и `RRUM.A`. **Примечание.** Команда не использует слово расширения.
Биты состояния	N: Устанавливается, если результат отрицательный .A: Rdst.19 = 1, сбрасывается, если Rdst.19 = 0; .W: Rdst.15 = 1, сбрасывается, если Rdst.15 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB ($n = 1$), LSB+1 ($n = 2$), LSB+2 ($n = 3$), LSB+3 ($n = 4$). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Беззнаковое 20-битное число, находящееся в регистре R5, делится на 16. `RRUM.A #4,R5 ; R5 = R5 >> 4, R5/16`
Пример 2	Двухбайтное содержимое регистра R6 сдвигается вправо на один бит. Старший бит значения (R6.15) обнуляется. `RRUM.W #1,R6 ; R6 = R6/2, R6.19:15 = 0`

* RRUX.A, * RRUX[.W], * RRUX.B
Сдвиг вправо беззнакового операнда

Синтаксис	`RRUX.A  Rdst`
	`RRUX.W  Rdst`
	`RRUX  Rdst`
	`RRUX.B  Rdst`

Операция	C = 0 → MSB → MSB–1 → ... LSB+1 → LSB → 0
Описание	С командой `RRUX` может использоваться только регистровый режим адресации: содержимое операнда-приёмника сдвигается вправо на один бит, как показано на **Рис. 4.54**. Команда `RRUX.W` сбрасывает биты Rdst.19:16, команда `RRUX.B` сбрасывает биты Rdst.19:8. В старший бит загружается 0, а младший бит (LSB) выдвигается в бит переноса C.  ***Рис. 4.54.*** Сдвиг вправо беззнакового операнда: команда `RRUX(.B,.A)`. Регистровая адресация.
Биты состояния	N: Устанавливается, если результат отрицательный .A: dst.19 = 1, сбрасывается, если dst.19 = 0; .W: dst.15 = 1, сбрасывается, если dst.15 = 0; .B: dst.7 = 1, сбрасывается, если dst.7 = 0. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Загружается из LSB. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Слово, находящееся в регистре R6, сдвигается на 12 битов вправо.
	`RPT  #12`
	`RRГX.W  R6  ; R6 = R6 >> 12. Биты R6.19:16 = 0`

* SBCX.A, * SBCX[.W], * SBCX.B
Вычитание заёма из операнда

Синтаксис	`SBCX.A dst`
	`SBCX dst` или `SBCX.W dst`
	`SBCX.B dst`

Операция	dst + 0FFFFFh + C → dst dst + 0FFFFh + C → dst dst + 0FFh + C → dst

Эмуляция	`SUBCX.B #0,dst`
	`SUBCX #0,dst`
	`SUBCX.B #0,dst`

Описание	Бит переноса (C), уменьшенный на 1, прибавляется к операнду-приёмнику. Предыдущее содержимое операнда теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	8-битный счётчик, на который указывает R13, вычитается из 16-битного счётчика, на который указывает R12.
	`SUBX.B @R13,0(R12) ; Вычитаем младшие байты`
	`SBCX.B 1(R12) ; Вычитаем заём из старшего байта`

Примечание. Признак заёма
В качестве признака заёма используется инверсное значение бита переноса:

Заём	Бит C
Есть	0
Нет	1

SUBX.A, SUBX[.W], SUBX.B
Вычитание двух операндов

Синтаксис	`SUBX.A src,dst`
	`SUBX src,dst` или `SUBX.W src,dst`
	`SUBX.B src,dst`

Операция	(.not.src) + 1 + dst → dst или dst – src → dst
Описание	Операнд-источник вычитается из операнда-приёмника путём прибавления к последнему обратного кода операнда-источника плюс единица. Содержимое операнда-источника не изменяется. Результат сохраняется в операнде-приёмнике. Оба операнда могут располагаться в пределах всего адресного пространства
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src ≤ dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда из положительного отрицателен или если результат вычитания положительного операнда из отрицательного положителен; иначе сбрасывается (нет переполнения).
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	20-битная константа 87654h вычитается из слова EDE (младшие биты) и EDE+2 (старшие биты). `SUBX.A #87654h,EDE ; Вычитаем 87654h из EDE+2:EDE`

Пример 2	Элемент таблицы, адресуемый регистром R5 (20-битный адрес), вычитается из регистра R7. Если результат равен нулю, то выполняется переход к метке `TONI`. Содержимое регистра R5 увеличивается на 2. Биты R7.19:16 = 0.
	`SUBX.W @R5+,R7 ; Вычитаем элемент таблицы из R7, R5 + 2`
	`JZ TONI ; R7 = @R5 перед вычитанием`
	`... ; R7 <> @R5 перед вычитанием`

Пример 3	Байт CNT вычитается из байта, на который указывает R12. Адрес CNT лежит в пределах PC ±512 КБ. Регистр R12 может указывать на любую ячейку адресного пространства. `SUBX.B CNT,0(R12) ; Вычитаем CNT из @R12`
Замечание. Для увеличения плотности кода и скорости выполнения программы используйте команду `SUBA` в следующих случаях: `SUBX.A Rsrc,Rdst` или `SUBX.A #imm20,Rdst`	

SUBCX.A, SUBCX[.W], SUBCX.B
Вычитание двух операндов с учётом переноса

Синтаксис	`SUBCX.A src,dst`
	`SUBCX src,dst` или `SUBCX.W src,dst`
	`SUBCX.B src,dst`

Операция	(.not.src) + C + dst → dst или dst – (src – 1) + C → dst
Описание	Операнд-источник вычитается из операнда-приёмника путём прибавления к последнему обратного кода операнда-источника и значения бита переноса. Содержимое операнда-источника не изменяется. Результат сохраняется в операнде-приёмнике. Оба операнда могут располагаться в пределах всего адресного пространства
Биты состояния	N: Устанавливается, если результат отрицательный (MSB = 1), сбрасывается — если положительный (MSB = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из MSB результата, иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда из положительного отрицателен или если результат вычитания положительного операнда из отрицательного положителен; иначе сбрасывается (нет переполнения).
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	20-битная константа 87654h вычитается из регистра R5 с учётом переноса от предыдущей команды. `SUBCX.A #87654h,R5 ; R5 = R5 - 87654h - C`

Пример 2	48-битное число (3 слова), адресуемое регистром R5 (20-битный адрес), вычитается из 48-битного счётчика, адресуемого регистром R7. После операции R5 указывает на следующее 48-битное число.
	`SUBX.W @R5+,0(R7) ; Вычитаем младшие слова. R5 + 2`
	`SUBCX.W @R5+,2(R7) ; Вычитаем средние слова с учётом C. R5 + 2`
	`SUBCX.W @R5+,4(R7) ; Вычитаем старшие слова с учётом C. R5 + 2`

Пример 3	Байт CNT вычитается из байта, на который указывает R12, с учётом переноса от предыдущей операции. Используется 20-битная адресация. `SUBCX.B &CNT,0(R12) ; Вычитаем CNT из @R12`

SWPBX.A Перестановка байтов младшего слова
SWPBX[.W] Перестановка байтов двухбайтного слова

Синтаксис	`SWPBX.A dst` `SWPBX.W dst` или `SWPBX dst`
Операция	dst.15...8 ↔ dst.7...0
Описание	Регистровый режим адресации: биты Rn.15:8 и Rn.7:0 меняются местами. При использовании команды с суффиксом .A состояние битов Rdst.19:16 не изменяется. При использовании команды с суффиксом .W биты Rdst.19:16 сбрасываются. Прочие режимы адресации: при использовании команды с суффиксом .A биты 31:20 по адресу операнда-приёмника сбрасываются, биты 19:16 остаются неизменными, а биты 15:8 и биты 7:0 меняются местами. При использовании команды с суффиксом .W биты 15:8 и биты 7:0 адресованного слова меняются местами.

Рис. 4.55. Перестановка байтов: команда `SWPBX.A`, регистровая адресация.

Рис. 4.56. Перестановка байтов: команда `SWPBX.A`, операнд в ОЗУ.

<table>
<tr><td>Описание</td><td>

Рис. 4.57. Перестановка байтов: команда SWPBX[.W], регистровая адресация.

Рис. 4.58. Перестановка байтов: команда SWPBX[.W], операнд в ОЗУ.
</td></tr>
<tr><td>Биты состояния</td><td>Биты состояния не изменяются</td></tr>
<tr><td>Биты режима</td><td>OSCOFF, CPUOFF и GIE не изменяются</td></tr>
<tr><td>Пример 1</td><td>Меняются местами байты 20-битного слова EDE в ОЗУ.
<pre>MOVX.A #23456h,&EDE ; 23456h -> EDE
SWPBX.A EDE   ; 25634h -> EDE</pre></td></tr>
<tr><td>Пример 2</td><td>Старший и младший байты содержимого регистра R5 меняются местами.
<pre>MOVX.A #23456h,R5   ; 23456h -> R5
SWPBX.W R5   ; 05634h -> R5</pre></td></tr>
</table>

SXTX.A Расширение знака до 20-битного значения
SXTX[.W] Расширение знака до 16-битного значения

Синтаксис	`SXTX.A dst` `SXTX.W dst` или `SXTX dst`
Операция	dst.7 → dst.15:8 dst.7 → dst.19:8 (при использовании регистрового режима адресации)

Описание	Регистровый режим адресации: знаковый бит младшего байта операнда (Rdst.7) в биты 19:8 регистра.
	Прочие режимы:
	`SXTX.A`: знаковый бит младшего байта слова (dst.7) копируется в биты dst.19:8 операнда. Биты dst.32:20 сбрасываются.
	`SXTX[.W]`: знаковый бит младшего байта слова (dst.7) копируется в биты dst.15:8 операнда.
	`SXTX.A Rdst`
	19 16 15 8 7 6 0
	S
	`SXTX.A dst`
	31 20 19 16 15 8 7 6 0
	0 ..... 0 S
	Рис. 4.59. Расширение знака: команда `SXTX.A`.
	`SXTX[.W] Rdst`
	19 16 15 8 7 6 0
	S
	`SXTX[.W] dst`
	15 8 7 6 0
	S
	Рис. 4.60. Расширение знака: команда `SXTX[.W]`.

Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (C = .not.Z). V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	8-битное число со знаком, находящееся в битах EDE.7:0, расширяется до 20 бит. Биты 32:20, расположенные в EDE+2, сбрасываются. `SXTX.A &EDE ; Расширяем знак EDE -> EDE+2:EDE`

* TSTX.A, * TSTX[.W], * TSTX.B
Проверка операнда (на ноль)

Синтаксис	`TSTX.A dst`
	`TSTX dst` или `TSTX.W dst`
	`TSTX.B dst`

Операция	dst + 0FFFFFh + 1 dst + 0FFFFh + 1 dst + 0FFh + 1

Эмуляция	`CMPX.A #0,dst`
	`CMPX #0,dst`
	`CMPX.B #0,dst`

Описание	Операнд-приёмник сравнивается с нулём и в соответствии с результатом изменяются биты состояния. Сам операнд остаётся неизменным
Биты состояния	N: Устанавливается, если операнд отрицателен, сбрасывается — если положителен. Z: Устанавливается, если операнд равен нулю, иначе сбрасывается. C: Устанавливается. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Проверяется содержимое регистра R7. Если оно отрицательное, то выполнение программы продолжается с метки `R7NEG`, если положительное и не равно нулю — с метки `R7POS`.
	`TST R7 ; Проверяем R7`
	`JN R7NEG ; R7 < 0`
	`JZ R7ZERO ; R7 = 0`
	`R7POS ..... ; R7 > 0`
	`R7NEG ..... ; R7 < 0`
	`R7ZERO ..... ; R7 = 0`

Пример 2	Проверяется байт LEO в ОЗУ; счётчик команд указывает на верхнюю область памяти. Если значение байта отрицательно, то выполнение программы продолжается с метки `LEONEG`, если положительно и не равно нулю — с метки `LEOPOS`.
	`TSTX.B LEO ; Проверяем LEO`
	`JN LEONEG ; LEO < 0`
	`JZ LEOZERO ; LEO = 0`
	`LEOPOS ..... ; LEO > 0`
	`LEONEG ..... ; LEO < 0`
	`LEOZERO ..... ; LEO = 0`

XORX.A, XORX[.W], XORX.B
«Исключающее ИЛИ» двух операндов

Синтаксис	`XORX.A src,dst`
	`XORX src,dst` или `XORX.W src,dst`
	`XORX.B src,dst`

Операция	src .xor. dst → dst
Описание	Выполняется операция «Исключающее ИЛИ» между операндом-источником и операндом-приёмником. Результат помещается в операнд-приёмник. Операнд-источник не изменяется. Предыдущее содержимое операнда-приёмника теряется. Оба операнда могут располагаться в пределах всего адресного пространства
Биты состояния	N: Устанавливается, если результат отрицателен (MSB = 1), иначе сбрасывается. Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если результат ненулевой, иначе сбрасывается (C = .not.Z). V: Устанавливается, если оба операнда отрицательные.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример 1	Состояние битов 20-битного слова CNTR изменяется в соответствии со значением 20-битного слова TONI (20-битный адрес). `XORX.A TONI,&CNTR ; Переключаем биты CNTR`
Пример 2	Слово из таблицы, адресуемое регистром R5 (20-битный адрес), используется для изменения состояния битов регистра R6. Биты R6.19:6 = 0. `XORX.W @R5,R6 ; Переключаем биты R6. R6.19:16 = 0`

Пример 3	Сбрасываются в 0 те биты младшего байта R7, которые отличаются от соответствующих битов 1-байтовой переменной EDE (20-битный адрес).
	`XORX.B EDE,R7 ; Устанавливаем отличающиеся биты в 1`
	`INV.B R7 ; Инвертируем младший байт. R7.19:8 = 0`

4.6.4. Адресные команды

Адресные команды MSP430X — это такие команды, которые способны работать с 20-битными операндами, однако ограничены в использовании ряда режимов адресации. Все адресные команды, за исключением команды `MOVA`, могут задействовать только регистровый и непосредственный режимы адресации (см. **Табл. 4.16**). Данное ограничение избавляет от необходимости использовать дополнительное слово расширения, что увеличивает плотность кода и скорость выполнения программы. Далее в настоящем разделе приведены подробные описания всех адресных команд.

ADDA Сложение 20-битного операнда с 20-битным регистром

Синтаксис	`ADDA Rsrc,Rdst` `ADDA #imm20,Rdst`
Операция	src + Rdst → Rdst
Описание	20-битный операнд-источник прибавляется к 20-битному содержимому регистра-приёмника. Предыдущее содержимое регистра теряется. Операнд-источник не изменяется
Биты состояния	N: Устанавливается, если результат отрицательный (Rdst.19 = 1), сбрасывается — если положительный (Rdst.19 = 0). Z: Устанавливается, если результат нулевой, иначе сбрасывается. C: Устанавливается, если произошёл перенос из 20-битного рещультата, иначе сбрасывается. V: Устанавливается, если результат операции над двумя положительными операндами отрицателен или если результат операции над двумя отрицательными операндами положителен; иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Содержимое регистра R5 увеличивается на 0A4320h. В случае переноса осуществляется переход к метке `TONI`.
	`ADDA #0A4320h,R56 ; Прибавляем A4320h к 20-битному R5`
	`JC TONI ; Переходим в случае переноса`
	`... ; Переноса не было`

* BRA Безусловный переход

Синтаксис	`BRA dst`
Операция	dst → PC
Эмуляция	`MOVA dst,PC`
Описание	Выполняется безусловный переход по любому адресу в пределах всего адресного пространства. Для операнда могут использоваться любой из семи режимов адресации. В команде используются 20-битные операнды. Если адрес назначения расположен в ячейке памяти X, то он содержится в двух соседних словах: X (младшие биты) и X+2 (старшие биты).
Биты состояния	Биты состояния не изменяются

Пример	Приведены примеры для всех режимов адресации.
	Непосредственная адресация: переход к метке `EDE`, находящейся в любом месте 20-битного адресного пространства или непосредственно по заданному адресу.
	`BRA #EDE ; MOVA #imm20,PC`
	`BRA #01AA04h`
	Относительная адресация: переход по 20-битному адресу, значение которого хранится в словах EXEC (младшие биты) и EXEC+2 (старшие биты). EXEC располагается по адресу (PC + X), где X находится в диапазоне ±32 КБ. Косвенная адресация.
	`BRA EXEC ; MOVA z16(PC),PC`
	Примечание. Если для записи смещения требуется больше 16 бит, можно использовать 20-битное смещение со следующей командой:
	`MOVX.A EXEC,PC ; 1 МБ, 20-битное смещение`

Пример	Абсолютная адресация: переход по 20-битному адресу, значение которого хранится в словах EXEC (младшие биты) и EXEC+2 (старшие биты). Косвенная адресация.
	`BRA  &EXEC  ; MOVA  &abs20,PC`
	
	Регистровая адресация: переход по 20-битному адресу, находящемуся в регистре R5. Косвенная адресация по содержимому R5.
	`BRA  R5  ; MOVA  R5,PC`
	
	Косвенная адресация: переход по 20-битному адресу, находящемуся в слове памяти, адресуемом регистром R5 (младшие биты адреса). Старшие биты адреса хранятся в слове с адресом (R5 + 2). Косвенная адресация по косвенному содержимому R5.
	`BRA  @R5  ; MOVA  @R5,PC`
	
	Косвенная адресация с автоинкрементом: переход по 20-битному адресу, находящемуся в слове памяти, адресуемом регистром R5, с последующим увеличением содержимого R5 на 4. Косвенная адресация по косвенному содержимому R5.
	`BRA  @R5+  ; MOVA  @R5+,PC. R5+4`
	
	Индексная адресация: переход по 20-битному адресу, находящемуся в слове памяти с адресом (R5 + X) например обращение к таблице адресов, расположенной, начиная с адреса X. По адресу (R5 + X) располагаются младшие биты искомого адреса, а по адресу (R5 + X +2) — старшие. Смещение X находится в диапазоне ±32 КБ. Косвенная адресация по косвенному содержимому R5 + X.
	`BRA  X(R5)  ; MOVA  z16(R5),PC`
	
	Примечание. Если для записи смещения X требуется больше 16 бит, можно использовать 20-битное смещение со следующей командой:
	`MOVX.A  X(R5),PC  ; 1Мб, 20-битное смещение`

CALLA Вызов подпрограммы

Синтаксис	`CALLA dst`
Операция	dst → tmp (20-битное dst вычисляется и запоминается) SP – 2 → SP PC.19:16 → @SP (Старшие биты PC сохраняются в стеке) SP – 2 → SP PC.15:0 → @SP (Младшие биты PC сохраняются в стеке) tmp → PC (dst загружается в PC)
Описание	Осуществляется вызов подпрограммы, расположенной по любому адресу в пределах всего адресного пространства. Могут использоваться любые режимы адресации. В команде используется 20-битный операнд. Если адрес назначения расположен в ячейке памяти X, то он содержится в двух соседних словах: X (младшие биты) и X+2 (старшие биты). Для хранения адреса возврата используются 2 элемента стека. Возврат из подпрограммы осуществляется по команде `RETA`
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Приведены примеры для всех режимов адресации.
	
	Непосредственная адресация: вызов подпрограммы по метке `EXEC` или по непосредственному адресу.
	`CALLA #EXEC ; Стартовый адрес - EXEC`
	`CALLA #01AA04h ; Стартовый адрес - 01AA04h`
	
	Относительная адресация: вызов подпрограммы по 20-битному адресу, значение которого хранится в словах EXEC (младшие биты) и EXEC+2 (старшие биты). EXEC располагается по адресу (PC + X), где X находится в диапазоне ±32 КБ. Косвенная адресация.
	`CALLA EXEC ; Стартовый адрес - @EXEC. z16(PC)`
	
	Абсолютная адресация: вызов подпрограммы по 20-битному адресу, значение которого хранится в словах EXEC (младшие биты) и EXEC+2 (старшие биты). Косвенная адресация.
	`CALLA &EXEC ; Стартовый адрес - @EXEC`
	
	Регистровая адресация: вызов подпрограммы по 20-битному адресу, находящемуся в регистре R5. Косвенная адресация по содержимому R5.
	`CALLA R5 ; Стартовый адрес - в регистре R5`
	
	Косвенная адресация: вызов подпрограммы по 16-битному адресу, находящемуся в слове памяти, адресуемом регистром R5 (20-битный адрес). Косвенная адресация по косвенному содержимому R5.
	`CALLA @R5 ; Стартовый адрес - @R5`
	Косвенная адресация с автоинкрементом: вызов подпрограммы по 20-битному адресу, находящемуся в слове памяти, адресуемом регистром R5 (20-битный адрес), с последующим увеличением содержимого R5 на 4. Косвенная адресация по косвенному содержимому R5.
	`CALLA @R5+ ; Стартовый адрес - @R5, R5 = R5 + 4`

Пример	Индексная адресация: вызов подпрограммы по 20-битному адресу, находящемуся в слове памяти с адресом (R5 + X), например обращение к таблице адресов, расположенной, начиная с адреса X. По адресу (R5 + X) располагаются младшие биты искомого адреса, а по адресу (R5 + X +2) — старшие. Смещение X находится в диапазоне ±32 КБ. Косвенная адресация по косвенному содержимому R5 + X. `CALLA  X(R5)  ; Стартовый адрес - @(R5 + X). z16(R5)`

* CLRA Очистка 20-битного регистра

Синтаксис	`CLRA  Rdst`
Операция	0 → Rdst
Эмуляция	`MOVA  #0,Rdst`
Описание	Регистр-приёмник обнуляется
Биты сост.	Биты состояния не изменяются
Пример	Обнуляются все 20 бит регистра R10. `CLRA  R10  ; 0 -> R10`

CMPA Сравнение 20-байтного операнда с 20-битным регистром

Синтаксис	`CMPA  Rsrc,Rdst` `CMPA  #imm20,Rdst`
Операция	.not.src + 1 + Rdst или (Rdst – src)
Описание	20-битный операнд-источник вычитается из регистра-приёмника. Для выполнения этой операции к содержимому регистра прибавляется обратный код операнда-источника плюс 1. Операция влияет только на биты состояния регистра SR
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src ≤ dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если был перенос из MSB результата, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда из положительного отрицателен или результат вычитания положительного операнда из отрицательного положителен, иначе сбрасывается (нет переполнения).
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	Сравнивается 20-битная константа и регистр R6. В случае равенства выполняется переход к метке `EQUAL`.
	`CMPA  #12345h,R6  ; Сравниваем R6 с 12345h`
	`JEQ  EQUAL  ; R5 = 12345h`

Пример 2	Сравниваются 20-битные значения, находящиеся в регистрах R5 и R6. Если R5 больше или равен R6, выполняется переход к метке `GRE`.
	`CMPA  R5,R6  ; Сравниваем R6 с R5 (R5 - R6)`
	`JGE  GRE  ; R5 >= R6`
	`...  ; R5 < R6`

* DECDA Уменьшение 20-битного регистра на 2

Синтаксис	`DECDA Rdst`
Операция	Rdst – 2 → Rdst
Эмуляция	`SUBA #2,Rdst`
Описание	Значение регистра-приёмника уменьшается на 2. Предыдущее содержимое регистра теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если Rdst содержал 2, иначе сбрасывается. C: Сбрасывается, если Rdst содержал 0 или 1, иначе устанавливается. V: Устанавливается, если произошло переполнение, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	Содержимое регистра R5 уменьшается на 2. `DECDA R5 ; R5 = R5 - 2`

* INCDA Увеличение 20-битного регистра на 2

Синтаксис	`INCDA Rdst`
Операция	Rdst + 2 → dst
Эмуляция	`ADDA #2,dst`
Описание	Значение регистра-приёмника увеличивается на 2. Предыдущее содержимое регистра теряется
Биты состояния	N: Устанавливается, если результат отрицательный, сбрасывается — если положительный. Z: Устанавливается, если dst содержал 0FFFFEh, иначе сбрасывается. C: Устанавливается, если dst содержал 0FFFFEh или 0FFFFFh, иначе сбрасывается. V: Устанавливается, если dst содержал 07FFFEh или 07FFFFh, иначе сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются
Пример	Содержимое регистра R5 увеличивается на 2. `INCDA R5 ; R5 = R5 + 2`

MOVA Пересылка 20-битного операнда

Синтаксис	`MOVA Rsrc,Rdst`
	`MOVA #imm20,Rdst`
	`MOVA z16(Rsrc),Rdst`
	`MOVA EDE,Rdst`
	`MOVA &abs20,Rdst`
	`MOVA @Rsrc,Rdst`
	`MOVA @Rsrc+,Rdst`
	`MOVA Rsrc,z16(Rdst)`
	`MOVA Rsrc,&abs20`

Операция	src → Rdst Rsrc → dst
Описание	20-битный операнд-источник пересылается в операнд-приёмник. Содержимое операнда-источника не изменяется. Предыдущие значение операнда-приёмника теряется
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Копируем содержимое регистра R9 в регистр R8.
	`MOVA  R9,R8  ; R9 -> R8`
	
	Загружаем 20-битную константу 12345h в регистр R12.
	`MOVA  #12345h,R12  ; 12345h -> R12`
	
	Копируем 20-битное значение, адресуемое (R9 + 100h) в регистр R8. Младшие биты операнда-источника располагаются по адресу (R9 + 100h), а старшие — по адресу (R9 + 102h).
	`MOVA  100h(R9),R8  ; Смещение ±32 КБ. Пересылается два слова`
	
	Копируем 20-битное значение, расположенное по абсолютному 20-битному адресу EDE (младшие биты) и EDE+2 (старшие биты), в регистр R12.
	`MOVA  &EDE,R12  ; &EDE -> R12. Пересылается два слова`
	
	Копируем 20-битное значение, расположенное по 20-битному адресу EDE (младшие биты) и EDE+2 (старшие биты), в регистр R12. Величина смещения относительно PC находится в диапазоне ±32 КБ.
	`MOVA  EDE,R12  ; EDE -> R12. Пересылается два слова`
	
	Копируем 20-битное значение, на которое указывает регистр R9 (20-битный адрес), в регистр R8. Младшие биты операнда-источника располагаются по адресу @R9, а старшие — по адресу (@R9+2).
	`MOVA  @R9,R8  ; @R9 -> R8. Пересылается два слова`
	
	Копируем 20-битное значение, на которое указывает регистр R9 (20-битный адрес), в регистр R8. Затем регистр R9 увеличивается на 4. Младшие биты операнда-источника располагаются по адресу @R9, а старшие — по адресу (@R9+2).
	`MOVA  @R9+,R8  ; @R9 -> R8. Пересылается два слова`
	
	Копируем 20-битное содержимое R8 по адресу (R9 + 100h). Младшие биты операнда-приёмника располагаются по адресу (R9 + 100h), а старшие — по адресу (R9 + 102h).
	`MOVA  R8,100h(R9)  ; Смещение ±32 КБ. Пересылается два слова`
	
	Копируем 20-битное содержимое R13 в память по абсолютному 20-битному адресу EDE (младшие биты) и EDE+2 (старшие биты).
	`MOVA  R13,&EDE  ; R13 -> &EDE. Пересылается два слова`
	
	Копируем 20-битное содержимое R13 в память по 20-битному адресу EDE (младшие биты) и EDE+2 (старшие биты). Величина смещения относительно PC находится в диапазоне ±32 КБ.
	`MOVA  R12,EDE  ; R12 -> EDE. Пересылается два слова`

* RETA Возврат из подпрограммы

Синтаксис	RET
Операция	@SP → PC.15:0 (Восстанавливаем младшие биты сохранённого PC) SP + 2 → SP @SP → PC.19:16 (Восстанавливаем старшие биты сохранённого PC) SP + 2 → SP
Эмуляция	MOVA @SP+,PC
Описание	20-битный адрес возврата, помещённый в стек при выполнении команды CALLA, загружается в счётчик команд. Выполнение программы продолжается с команды, следующей за командой вызова подпрограммы. Биты регистра состояния SR.11:0 не изменяются, что позволяет использовать эти биты для передачи информации
Биты состояния	Биты состояния не изменяются
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Вызывается подпрограмма SUBR, которая может располагаться в любом месте адресного пространства.
	`CALLA #SUBR ; Вызываем подпрограмму по адресу SUBR`
	`... ; Из подпрограммы возвращаемся сюда`
	`SUBR PUSHM.A #2,R14 ; Сохраняем R14 и R13`
	`; (20-битные значения)`
	`... ; Тело подпрограммы`
	`POPM.A #2,R14 ; Восстанавливаем R13 и R14`
	`RETA ; Возвращаемся`

SUBA Вычитание 20-битного операнда из 20-битного регистра

Синтаксис	`SUBA Rsrc,Rdst` `SUBA #imm20,Rdst`
Операция	(.not.src) + 1 + Rdst → Rdst или Rdst – src → Rdst
Описание	20-битный операнд-источник вычитается из 20-битного регистра-приёмника путём прибавления к последнему обратного кода операнда плюс единица. Содержимое операнда-источника не изменяется. Результат сохраняется в регистре-приёмнике
Биты состояния	N: Устанавливается, если результат отрицательный (src > dst), сбрасывается — если положительный (src ≤ dst). Z: Устанавливается, если результат нулевой (src = dst), иначе сбрасывается (src ≠ dst). C: Устанавливается, если произошёл перенос из MSB результата (Rdst.19), иначе сбрасывается. Устанавливается, если не было заёма, иначе сбрасывается. V: Устанавливается, если результат вычитания отрицательного операнда из положительного отрицателен или если результат вычитания положительного операнда из отрицательного положителен; иначе сбрасывается (нет переполнения).
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример 1	20-битное число, находящееся в регистре R5, вычитается из R6. При возникновении переноса выполняется переход к метке `TONI`.
	`SUBA R5,R6 ; R6 - R5 -> R6`
	`JC TONI ; Перенос`
	`... ; Переноса не было`

Пример 2	Элемент таблицы, адресуемый регистром R5 (20-битный адрес), вычитается из регистра R7. Если результат равен нулю, то выполняется переход к метке `TONI`. Содержимое регистра R5 увеличивается на 2. Биты R7.19:16 = 0.
	`SUBX.W @R5+,R7 ; Вычитаем элемент таблицы из R7, R5 + 2`
	`JZ TONI ; R7 = @R5 перед вычитанием`
	`... ; R7 <> @R5 перед вычитанием`

* TSTA Проверка 20-битного регистра (на ноль)

Синтаксис	`TSTA  Rdst`
Операция	Rdst + 0FFFFFh + 1
Эмуляция	`CMPA  #0,Rdst`
Описание	Содержимое регистра-приёмника сравнивается с нулём и в соответствии с результатом изменяются биты состояния. Сам операнд остаётся неизменным
Биты состояния	N: Устанавливается, если операнд отрицателен, сбрасывается — если положителен. Z: Устанавливается, если операнд равен нулю, иначе сбрасывается. C: Устанавливается. V: Сбрасывается.
Биты режима	OSCOFF, CPUOFF и GIE не изменяются

Пример	Проверяется 20-битное значение в регистре R7. Если оно отрицательное, то выполнение программы продолжается с метки `R7NEG`, если положительное и не равно нулю — с метки `R7POS`.
	`TSTA R7 ; Проверяем R7`
	`JN R7NEG ; R7 < 0`
	`JZ R7ZERO ; R7 = 0`
	`R7POS ..... ; R7 > 0`
	`R7NEG ..... ; R7 < 0`
	`R7ZERO ..... ; R7 = 0`

ГЛАВА 5

МОДУЛЬ СИНХРОНИЗАЦИИ BASIC CLOCK MODULE+

В этой главе описывается функционирование модуля синхронизации Basic Clock Module+, который отвечает за формирование тактовых сигналов микроконтроллеров семейства MSP430x2xx.

5.1. Введение

Модуль синхронизации Basic Clock Module+ позволяет снизить стоимость конечной системы и обеспечивает сверхнизкое потребление устройства. Используя один из трёх тактовых сигналов, формируемых модулем, пользователь может добиться оптимального соотношения производительности и энергопотребления. Модуль синхронизации может быть программно сконфигурирован для работы без использования дополнительных внешних элементов, с одним внешним резистором, с одним или двумя внешними кварцевыми или керамическими резонаторами.

Модуль синхронизации содержит три или четыре источника тактового сигнала:

- LFXT1CLK — низкочастотный/высокочастотный генератор, который может работать с «часовым» кварцевым резонатором или внешним сигналом частотой 32 768 Гц или же с обычными кварцевыми/керамическими резонаторами или внешним сигналом синхронизации частотой от 400 кГц до 16 МГц.
- XT2CLK — опциональный высокочастотный генератор, который может работать с обычными кварцевыми/керамическими резонаторами или внешним сигналом синхронизации частотой от 400 кГц до 16 МГц.
- DCOCLK — встроенный генератор с цифровым управлением (DCO).
- VLOCLK — встроенный низкочастотный генератор с очень низким потреблением, работающий на частоте 12 кГц.

Модуль синхронизации формирует три тактовых сигнала:

- ACLK — вспомогательный тактовый сигнал. Источник выбирается программно: LFXT1CLK или VLOCLK. Посредством делителя частота сигнала от выбранного источника уменьшается в 1, 2, 4 или 8 раз. Сигнал ACLK может программно назначаться для отдельных периферийных модулей.
- MCLK — основной тактовый сигнал. Источник выбирается программно: LFXT1CLK, VLOCLK, XT2CLK (если имеется в конкретной модели) или

DCOCLK. Посредством делителя частота сигнала от выбранного источника уменьшается в 1, 2, 4 или 8 раз. Сигнал MCLK используется для тактирования ЦПУ и системы.

- SMCLK — дополнительный тактовый сигнал. Источник выбирается программно: LFXT1CLK, VLOCLK, XT2CLK (если имеется в конкретной модели) или DCOCLK. Посредством делителя частота сигнала от выбранного источника уменьшается в 1, 2, 4 или 8 раз. Сигнал SMCLK может программно назначаться для отдельных периферийных модулей.

Блок-схема модуля синхронизации приведена на **Рис. 5.1**.

5.2. Функционирование модуля синхронизации

После сигнала PUC (очистка при включении питания) тактовые сигналы MCLK и SMCLK формируются из DCOCLK, имеющем частоту около 1.1 МГц (точное значение приводится в документации на конкретные модели). Сигнал ACLK формируется из LFXT1CLK, при этом генератор LFXT1 работает в режиме LF с использованием встроенных конденсаторов (нагрузочная ёмкость — 6 пФ).

Биты управления SCG0, SCG1, OSCOFF и CPUOFF регистра состояния определяют режимы работы ядра MSP430 и включают или отключают отдельные узлы модуля синхронизации (см. главу 2 «Сброс, прерывания и режимы работы»). Конфигурирование модуля осуществляется с помощью регистров DCOCTL, BCSCTL1, BCSCTL2 и BCSCTL3.

Изменение конфигурации модуля Base Clock Module+ может быть выполнено программно в любой момент времени:

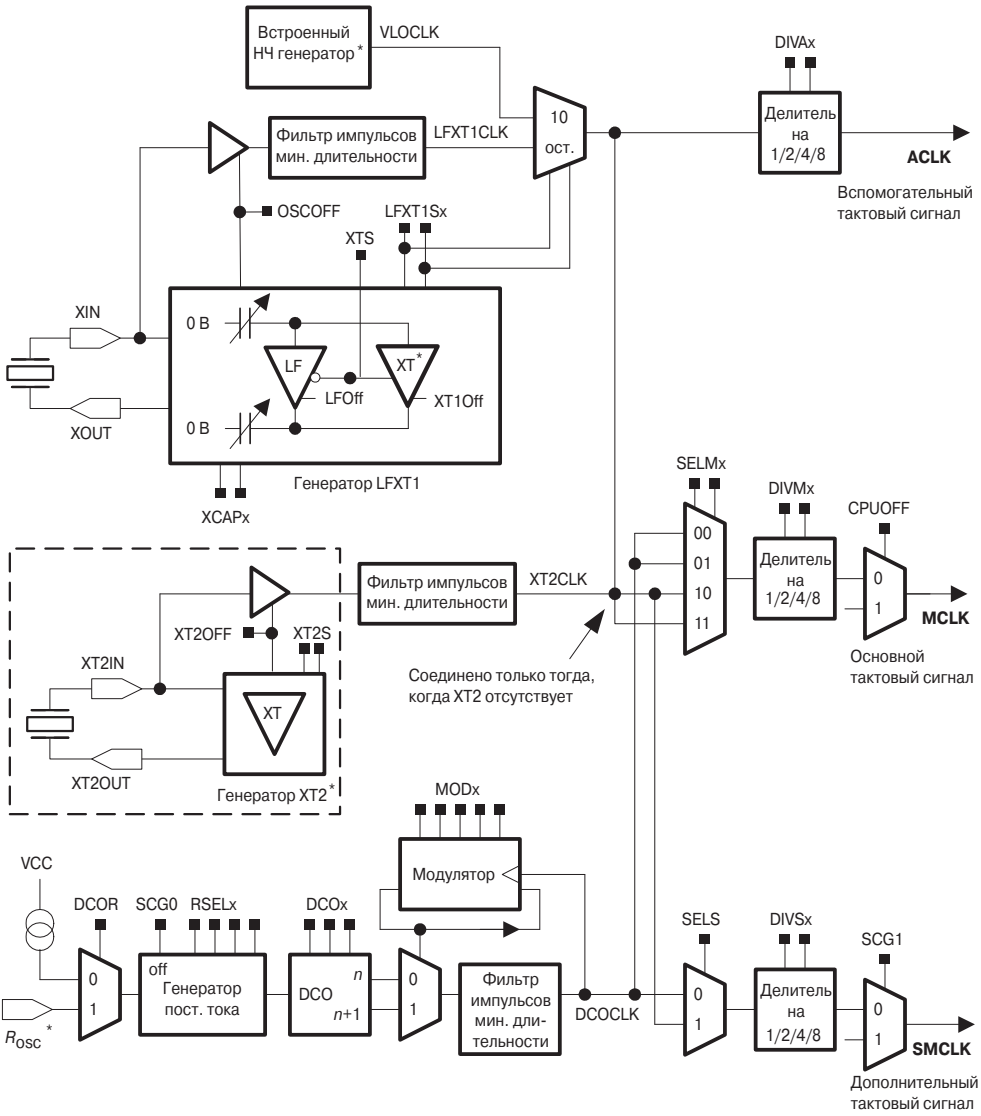
```
BIS.B #RSEL2+RSEL1+RSEL0,&BCSCTL1 ; Выбираем 7-й диапазон
BIS.B #DCO2+DCO1+DCO0,&DCOCTL     ; Выбираем максимальную частоту DCO
```

5.2.1. Возможности модуля синхронизации и приложения с низким энергопотреблением

К системе синхронизации устройств с батарейным питанием часто предъявляются противоречивые требования:

- наличие тактового сигнала низкой частоты для снижения энергопотребления и реализации функций отсчёта времени;
- наличие тактового сигнала высокой частоты для обеспечения быстрой реакции на события и быстрой обработки данных;
- обеспечение стабильности частоты тактового сигнала при изменении температуры и напряжения питания.

Модуль синхронизации Basic Clock Module+ разработан с учётом перечисленных требований и позволяет пользователю выбирать любой из трёх доступных тактовых сигналов: ACLK, MCLK и SMCLK. Для достижения оптимальной производительности при незначительном потреблении в качестве источника сигнала ACLK может использоваться экономичный «часовой» кварц частотой 32 768 Гц, при этом обеспечивается стабильный синхросигнал и малое потребление в режиме ожидания. Если точность формирования временных интервалов не критична, то в качестве источника сигнала ACLK может использоваться встроенный низкочас-



***Различные исполнения модуля синхронизации**

Не во всех моделях семейства MSP430x2xx модуль синхронизации реализован полностью.

MSP430x20xx — LFXT1 не поддерживает режим HS, XT2 отсутствует, использование внешнего R_{OSC} не поддерживается.

MSP430x21x1 — встроенный НЧ генератор отсутствует, XT2 отсутствует, использование внешнего R_{OSC} не поддерживается.

MSP430x21x2 — XT2 отсутствует.

MSP430x22xx, MSP430x23x0 — XT2 отсутствует.

Рис. 5.1. Блок-схема модуля синхронизации Basic Clock Module+.

тотный генератор. Сигнал MCLK может формироваться встроенным DCO, который в случае необходимости активируется обработчиками прерываний при наступлении соответствующих событий. Сигнал SMCLK может формироваться кварцевым генератором с внешним резонатором либо DCO в зависимости от требований, предъявляемых периферийными модулями. Гибкая система тактирования и наличие делителей тактовых сигналов обеспечивают возможность точной настройки модуля синхронизации в соответствии с требованиями конкретного приложения.

5.2.2. Встроенный низкочастотный генератор со сверхнизким потреблением

Встроенный НЧ генератор со сверхнизким потреблением (VLO) работает на частоте 12 кГц (прочие параметры приводятся в документации на конкретные модели) без внешнего кварцевого резонатора. Использование VCOCLK в качестве источника тактового сигнала задаётся битами LFXT1Sx = 10 при XTS = 0. Бит OSCOFF отключает генератор для перехода в режим пониженного энергопотребления LPM4. При использовании генератора VLO кварцевый генератор LFXT1 отключается, снижая тем самым потребление микроконтроллера. Если генератор VLO не используется, то его ток потребления равен нулю.

5.2.3. Генератор LFXT1

Генератор LFXT1 имеет очень малое потребление в режиме LF (XT = 0) при использовании «часового» кварца частотой 32 768 Гц. Кварцевый резонатор подключается к выводам XIN и XOUT микроконтроллера без использования дополнительных элементов. Величина нагрузочной ёмкости для кварцевого резонатора при работе генератора LFXT1 в режиме LF задаётся программно конфигурируемыми битами XCAPx. Эта ёмкость может принимать значения из ряда: 1 пФ, 6 пФ, 10 пФ или 12.5 пФ. При необходимости допускается подключение внешних конденсаторов.

В режиме HF (XTS = 1, XCAPx = 00) генератор LFXT1 поддерживает использование высокочастотных кварцевых и керамических резонаторов. Высокочастотный резонатор подключается к выводам XIN и XOUT, при этом на обоих выводах должны присутствовать внешние нагрузочные конденсаторы. Ёмкость данных конденсаторов выбирается в соответствии со спецификацией используемого резонатора. Биты LFXT1Sx определяют рабочий диапазон частот при работе генератора LFXT1 в режиме HF.

В любом из режимов генератор LFXT1 может работать с внешним тактовым сигналом, подаваемым на вход XIN (LFXT1Sx = 11, OSCOFF = 0, XCAPx = 00). При использовании внешнего сигнала его частота должна соответствовать спецификации микроконтроллера для выбранного режима. Если частота внешнего сигнала окажется ниже минимально допустимой величины, то может быть установлен бит LFXT1OF, предотвращающий тактирование ЦПУ сигналом LFXT1CLK.

Если LFXT1CLK не используется в качестве источника тактового сигнала SMCLK или MCLK, то генератор LFXT1 может быть отключён установкой бита OSCOFF, как показано на **Рис. 5.2**.

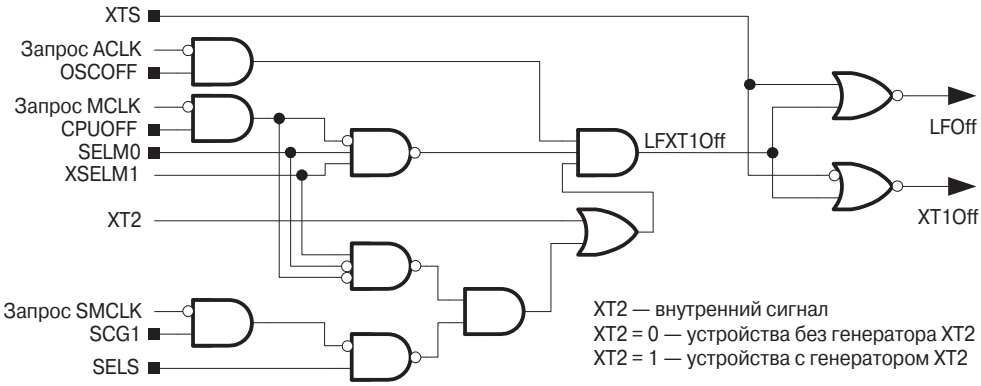


Рис. 5.2. Сигналы отключения генератора LFXT1.

Примечание. Особенности генератора LFXT1

Время запуска низкочастотных кварцевых резонаторов часто составляет несколько сотен миллисекунд.

Генераторы с очень малым потреблением, такие как генератор LFXT1 в режиме LF, необходимо защищать от помех со стороны других источников. Резонатор должен быть размещён как можно ближе к микроконтроллеру, корпус резонатора — заземлён, а дорожки, связывающие резонатор с микроконтроллером, — защищены «земляными» цепями.

5.2.4. Генератор XT2

В некоторых моделях имеется второй кварцевый генератор, XT2. Этот генератор формирует сигнал XT2CLK, а его параметры идентичны параметрам генератора LFXT1 в режиме HF. Рабочий диапазон частот генератора XT2 задаётся битами XT2Sx. Если сигнал XT2CLK не используется в качестве источника тактового сигнала SMCLK или MCLK, то генератор XT2 может быть отключён установкой бита XT2OFF, как показано на **Рис. 5.3**.

Генератор XT2 может работать с внешним тактовым сигналом, подаваемым на вход XT2IN (XT2Sx = 11 и XT2OFF = 0). При использовании внешнего сигнала его частота должна соответствовать спецификации генератора для конкретной модели. Если частота внешнего сигнала окажется ниже минимально допустимой величины, то может быть установлен бит XT2OF, предотвращающий тактирование ЦПУ сигналом XT2CLK.

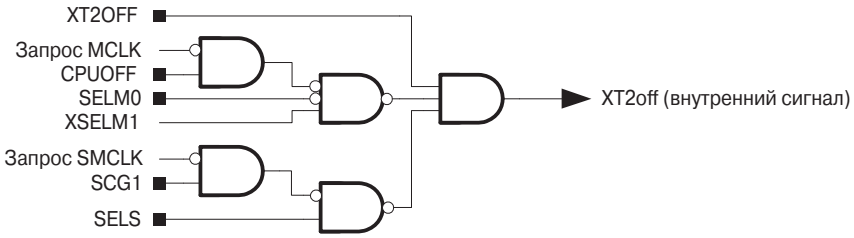


Рис. 5.3. Сигналы отключения генератора XT2.

5.2.5. Генератор с цифровым управлением (DCO)

Генератор DCO представляет собой встроенный генератор с цифровым управлением. Частота DCO может подстраиваться программно с помощью битов DCOx, MODx и RSELx.

Отключение DCO

Если DCO не используется для формирования тактового сигнала SMCLK или MCLK в активном режиме микроконтроллера, то он может быть отключён установкой бита SCG0, как показано на **Рис. 5.4**.

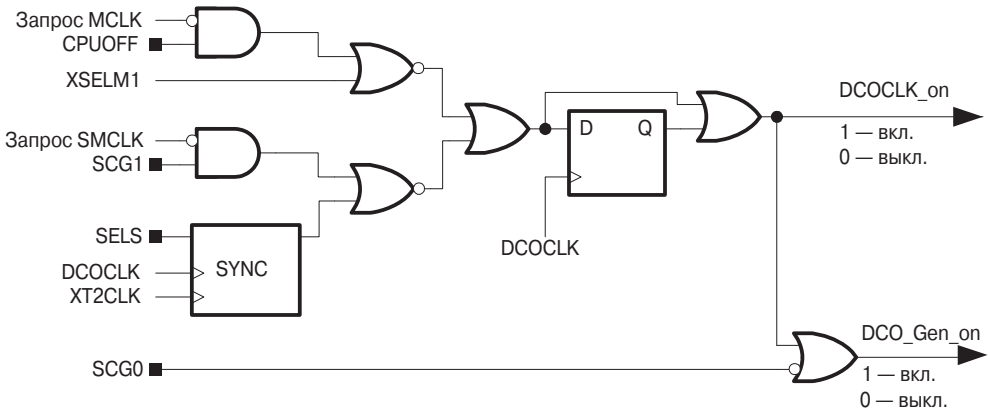


Рис. 5.4. Управление работой DCO.

Подстройка частоты DCO

После сигнала PUC в биты управления DCO загружаются такие значения (RSELx = 7, DCOx = 3), которые обеспечивают запуск DCO на средней частоте рабочего диапазона. При этом тактовые сигналы MCLK и SMCLK формируются из сигнала DCOCLK. Поскольку ЦПУ тактируется сигналом MCLK, а DCO имеет весьма малое время запуска, программа начинает выполняться менее чем через 2 мкс после появления сигнала PUC. Типовые значения диапазонов и дискретных интервалов для битов RSELx и DCOx показаны на **Рис. 5.5**.

Частота сигнала DCOCLK устанавливается следующим образом:

- Четыре бита RSELx определяют один из шестнадцати рабочих диапазонов DCO. Значения частот для этих диапазонов приводятся в документации на конкретные модели.
- Три бита DCOx делят выбранный диапазон на 8 дискретных значений, отличающихся друг от друга примерно на 10%.
- Пять битов MODx определяют частоту переключений с текущей частоты, заданной битами DCOx, на следующую частоту, соответствующую значению DCOx+1. При DCOx = 07h содержимое битов MODx игнорируется, поскольку в этом случае DCO уже работает на максимально возможной для выбранного диапазона RSELx частоте.

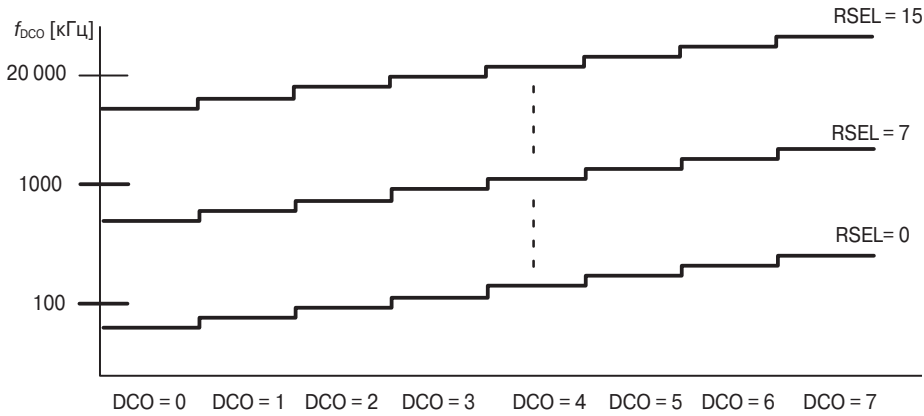


Рис. 5.5. Частота DCO и значения битов DCOx и RSELx.

Все микроконтроллеры семейства MSP430F2xx содержат в сегменте A информационной секции флэш-памяти значения регистров DCOCTL и BCSCTL1, откалиброванные для конкретных частот. Чтобы использовать калиброванные значения, их копируют из флэш-памяти в соответствующие регистры. Эти значения определяют состояния битов DCOx, MODx, RSELx и сбрасывают все прочие биты регистров, за исключением бита XT2OFF, который устанавливается в 1. Остальные биты регистра BCSCTL1 могут быть индивидуально установлены или сброшены командами `BIS.B` или `BIC.B`.

```
; Установка DCO на частоту 1 МГц:
MOV.B &CALBC1_1MHZ,&BCSCTL1   ; Устанавливаем диапазон
MOV.B &CALDCO_1MHZ,&DCOCTL    ; Устанавливаем частоту DCO и параметры
                              ; модуляции
```

Использование внешнего резистора (R_{OSC}) с DCO

Некоторые модели микроконтроллеров MSP430F2xx позволяют задавать ток DCO при помощи внешнего резистора R_{OSC}, подключённого к DV_{CC}. Этот режим включается при DCOR = 1. В данном режиме DCO имеет такие же характеристики, как и в моделях семейства MSP430x1xx, при этом биты RSELx могут принимать значения от 0 до 7 (бит RSEL3 игнорируется). Такая опция позволяет реализовать дополнительный метод регулирования частоты DCO посредством изменения сопротивления резистора. Параметры DCO для данного режима приводятся в документации на конкретные модели.

5.2.6. Модулятор DCO

Модулятор смешивает два сигнала DCO с частотами f_{DCO} и f_{DCO+1}, формируя сигнал с эффективной частотой, находящейся в диапазоне от f_{DCO} до f_{DCO+1}. При этом происходит распределение энергии тактового сигнала, что уменьшает уровень электромагнитных помех. Модулятор смешивает сигналы на протяжении

32 тактов DCOCLK и конфигурируется при помощи битов MODx. При MODx = 0 модулятор выключен.

Смешивание сигналов осуществляется в соответствии с формулой:

$$t = (32 - \mathrm{MODx}) \times t_{\mathrm{DCO}} + \mathrm{MODx} \times t_{\mathrm{DCO+1}}.$$

Поскольку f_{DCO} меньше эффективной частоты, а f_{DCO} — больше, погрешность эффективной частоты в сумме получается равной нулю. Накопления погрешности не происходит. Работа модулятора иллюстрируется на **Рис. 5.6**.

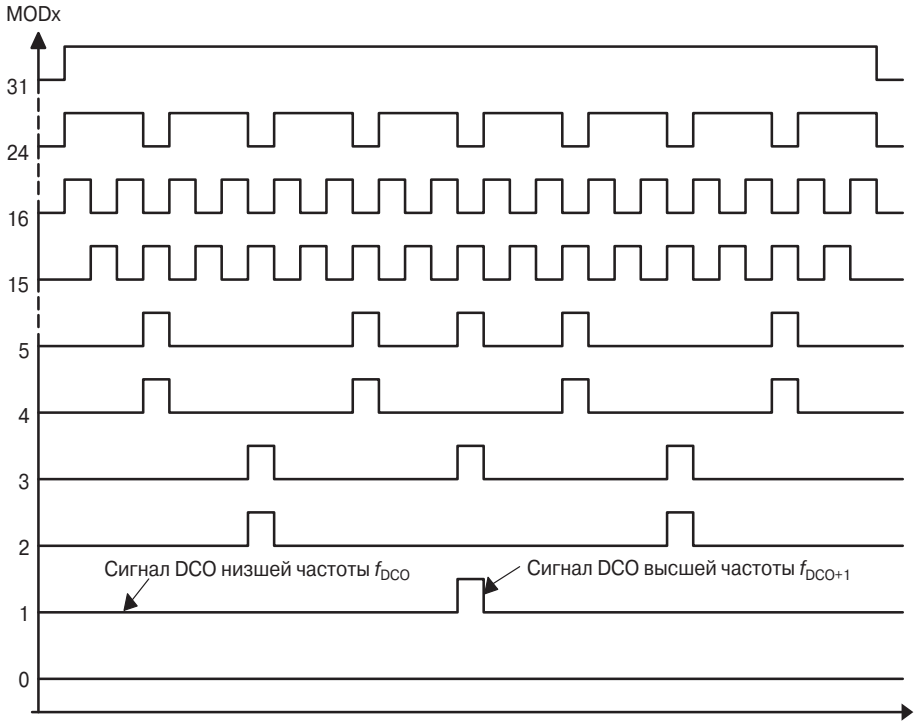


Рис. 5.6. Сигналы модулятора.

Конфигурирование модулятора и управление DCO осуществляется программно. Сигнал DCOCLK можно сравнить со стабильным сигналом известной частоты, после чего подстроить его частоту, корректируя значения битов DCOx, RSELx и MODx. Рекомендации по использованию DCO и примеры программного кода доступны на сайте производителя http://www.msp430.com.

5.2.7. Отказоустойчивая работа модуля синхронизации

Модуль синхронизации Basic Clock Module+ позволяет обеспечить работоспособность микроконтроллера при неисправности тактового генератора. При этом распознаются неисправности генераторов LFXT1 и XT2, как показано на **Рис. 5.7**.

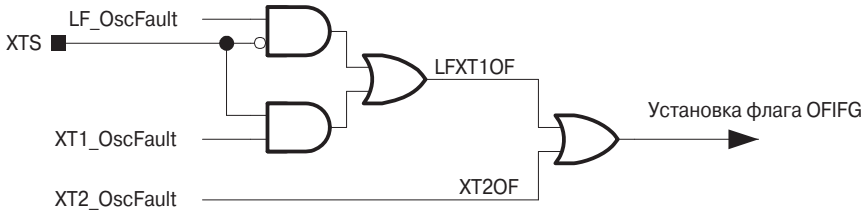


Рис. 5.7. Логика обработки сигналов о неисправности генераторов.

Отслеживаются следующие ситуации:

- неисправность низкочастотного генератора LFXT1 в режиме LF (LFXT1OF);
- неисправность низкочастотного генератора LFXT1 в режиме HF (LFXT1OF);
- неисправность высокочастотного генератора XT2 (XT2OF).

Биты неисправности кварцевых генераторов LFXT1OF и XT2OF устанавливаются, если соответствующий генератор включён, но работает некорректно. Эти биты остаются в установленном состоянии до устранения причины, вызвавшей отказ генератора, и автоматически сбрасываются, если включённый генератор работает нормально.

Флаг неисправности генератора OFIFG устанавливается и защёлкивается в момент сброса по включению питания или при обнаружении неисправности генератора (LFXT1OF или XT2OF). При установленном бите OFIFG тактовый сигнал MCLK формируется из сигнала DCO и, если установлен бит OFIE, генерируется запрос NMI-прерывания. При переходе к обработчику прерывания бит OFIE сбрасывается автоматически. Флаг OFIFG необходимо сбрасывать программно. Конкретный источник неисправности можно определить, проверив состояния отдельных битов неисправности.

Если обнаружена неисправность кварцевого генератора, использующегося для формирования тактового сигнала MCLK, то этот сигнал автоматически переключается на DCO. При этом состояние битов SELMx не изменяется — данная ситуация должна обрабатываться пользовательской программой.

Использование кварцевого генератора для формирования MCLK

После сброса по включению питания (PUC) тактовый сигнал MCLK формируется из сигнала DCOCLK. При необходимости для формирования MCLK можно использовать генератор LFXT1 или XT2.

Для переключения тактового сигнала MCLK с DCOCLK на сигнал от кварцевого генератора (LFXT1CLK или XT2CLK) необходимо выполнить следующие действия:

1. Включить генератор и выбрать требуемый режим работы.
2. Сбросить флаг OFIFG.
3. Подождать не менее 50 мкс.
4. Проверить флаг OFIFG — если он установлен, то повторить этапы 1...4.

```
; Выбираем для MCLK генератор LFXT1 (режим HF)
   BIC.W #OSCOFF,SR            ; Включаем генератор
   BIS.B #XTS,&BCSCTL1         ; Режим HF
   MOV.B #LFXT1S0,&BCSCTL3     ; Резонатор 1-3 МГц
L1 BIC.B #OFIFG,&IFG1          ; Сбрасываем OFIFG
   MOV.W #0FFh,R15             ; Формируем задержку
L2 DEC.W R15                   ;
   JNZ   L2                    ;
   BIT.B #OFIFG,&IFG1          ; Проверяем OFIFG
   JNZ   L1                    ; При необходимости проверку повторяем
   BIS.B #SELM1+SELM0,&BCSCTL2 ; Выбираем LFXT1CLK
```

5.2.8. Синхронизация тактовых сигналов

При переключении тактового сигнала MCLK или SMCLK с одного источника на другой осуществляется синхронизация (**Рис. 5.8**), позволяющая избежать возникновения состояний «гонки» сигналов:

1. Тактовый сигнал продолжает формироваться из текущего опорного сигнала до появления следующего нарастающего фронта.
2. Тактовый сигнал удерживается в состоянии ВЫСОКОГО уровня до появления нарастающего фронта сигнала от нового источника.
3. Выбирается новый источник, и формирование тактового сигнала продолжается в нормальном режиме.

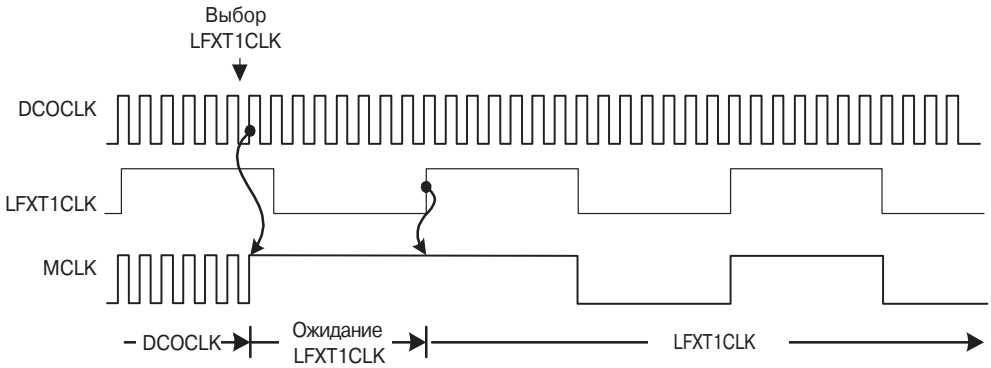


Рис. 5.8. Переключение тактового сигнала MCLK с DCOCLK на LFXT1CLK.

5.3. Регистры модуля синхронизации

Список регистров модуля Basic Clock Module+ приведён в **Табл. 5.1**.

Таблица 5.1. **Регистры модуля Basic Clock Module+**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления DCO	DCOCTL	Чтение/запись	056h	060h после PUC
Регистр управления 1 модуля синхронизации	BCSCTL1	Чтение/запись	057h	087h после POR*
Регистр управления 2 модуля синхронизации	BCSCTL2	Чтение/запись	058h	Сбрасывается после PUC
Регистр управления 3 модуля синхронизации	BCSCTL3	Чтение/запись	053h	005h после PUC
Регистр разрешения прерываний 1	IE1	Чтение/запись	000h	Сбрасывается после PUC
Регистр флагов прерываний 1	IFG1	Чтение/запись	002h	Сбрасывается после PUC
* Некоторые биты регистра также инициализируются при PUC. См. описание регистра.				

DCOCTL, регистр управления DCO

7	6	5	4	3	2	1	0
DCOx			**MODx**				
rw–0	rw–1	rw–1	rw–0	rw–0	rw–0	rw–0	rw–0

DCOx Биты 7...5 Выбор частоты DCO. Эти биты определяют одно из восьми дискретных значений частоты в пределах выбранного диапазона, задаваемого битами RSELx.

MODx Биты 4...0 Настройка модулятора. Эти биты определяют, как часто сигнал частотой f_{DCO+1} будет использоваться в течение периода, равного 32 тактам DCOCLK. В оставшееся время (MOD – 32) будет использоваться сигнал частотой f_{DCO}. Эти биты не используются при DCOx = 7.

BCSCTL1, регистр управления 1 модуля синхронизации

7	6	5	4	3	2	1	0
XT2OFF	**XTS***	**DIVAx**		**RSELx**			
rw–(1)	rw–(0)	rw–(0)	rw–(0)	rw–0	rw–1	rw–1	rw–1

*XTS = 1 не поддерживается в микроконтроллерах MSP430x20xx.

XT2OFF Бит 7 Выключение XT2. Этот бит управляет работой генератора XT2.
- 0 XT2 включен
- 1 XT2 выключен и не используется для формирования MCLK или SMCLK

XTS Бит 6 Режим LFXT1. Этот бит определяет режим работы генератора LFXT1.
- 0 Низкочастотный режим (LF)
- 1 Высокочастотный режим (HF)

DIVAx Биты 5...4 Делитель для ACLK.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

RSELx Биты 3...0 Выбор диапазона. Доступно 16 частотных диапазонов. Нижнему диапазону соответствует значение RSELx = 0. Бит RSEL3 игнорируется при DCOR = 1.

BCSCTL2, регистр управления 2 модуля синхронизации

7	6	5	4	3	2	1	0
SELMx		**DIVMx**		**SELS**	**DIVSx**		**DCOR***
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

*Не используется в моделях MSP430x20xx и MSP430x21x.

SELMx Биты 7...6 Выбор MCLK. Эти биты определяют источник тактового сигнала MCLK.
- 00 DCOCLK
- 01 DCOCLK
- 10 XT2CLK, если генератор XT2 реализован в микроконтроллере. LFXT1CLK или VLOCLK, если генератор XT2 отсутствует
- 11 LFXT1CLK или VLOCLK

DIVMx Биты 5...4 Делитель для MCLK.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

SELS Бит 3 Выбор SMCLK.
- 0 DCOCLK
- 1 XT2CLK, если генератор XT2 реализован в микроконтроллере. LFXT1CLK или VLOCLK, если генератор XT2 отсутствует

DIVSx Биты 2...1 Делитель для SMCLK.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

DCOR Бит 0 Выбор резистора DCO.
- 0 Внутренний резистор
- 1 Внешний резистор

BCSCTL3, регистр управления 3 модуля синхронизации

7	6	5	4	3	2	1	0
XT2Sx		**LFXT1Sx**		**XCAPx**		**XT2OF***	**LFXT1OF**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–1	r0	r–(1)

*Не используется в моделях MSP430x2xx, MSP430x21xx и MSP430x22xx.

XT2Sx Биты 7...6 Выбор диапазона XT2. Эти биты определяют частотный диапазон генератора XT2.
- 00 Кварцевый/керамический резонатор 0.4...1 МГц
- 01 Кварцевый/керамический резонатор 1...3 МГц
- 10 Кварцевый/керамический резонатор 3...16 МГц
- 11 Внешний сигнал частотой 0.4...16 МГц

LFXT1Sx Биты 5...4 Выбор низкочастотного источника тактового сигнала и выбор диапазона LFXT1. При XTS = 0 эти биты определяют источник тактового сигнала (LFXT1 или VLO). При XTS = 1 данные биты определяют частотный диапазон генератора LFXT1.

XTS = 0:
- 00 Кварцевый резонатор 32 768 Гц в генераторе LFXT1
- 01 Зарезервировано
- 10 VLOCLK
- 11 Внешний сигнал синхронизации

XTS = 1 (не применимо для моделей MSP430x20xx):
- 00 Кварцевый/керамический резонатор 0.4...1 МГц
- 01 Кварцевый/керамический резонатор 1...3 МГц
- 10 Кварцевый/керамический резонатор 3...16 МГц
- 11 Внешний сигнал частотой 0.4...16 МГц

XCAPx Биты 3...2 Выбор нагрузочной ёмкости. При XTS = 0 эти биты определяют величину нагрузочной ёмкости для резонатора, подключаемого к генератору LFXT1. Если XTS = 1 или LFXT1Sx = 11, то биты XCAPx должны быть равны 00.
- 00 ~ 1 пФ
- 01 ~ 6 пФ
- 10 ~ 10 пФ
- 11 ~ 12.5 пФ

XT2OF Бит 1 Неисправность генератора XT2.
- 0 Генератор функционирует нормально
- 1 Обнаружен сбой генератора

LFXT1OF Бит 0 Неисправность генератора LFXT1.
- 0 Генератор функционирует нормально
- 1 Обнаружен сбой генератора

IE1, регистр разрешения прерываний 1

7	6	5	4	3	2	1	0
						OFIE	
						rw–0	

	Биты 7...2	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
OFIE	Бит 1	Разрешение прерывания при неисправности генератора. Этот бит разрешает прерывание OFIFG. Поскольку прочие биты регистра IE1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Прерывание запрещено 1 Прерывание разрешено
	Бит 0	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

IFG1, регистр флагов прерываний 1

7	6	5	4	3	2	1	0
						OFIFG	
						rw–1	

	Биты 7...2	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
OFIFG	Бит 1	Флаг прерывания при неисправности генератора. Поскольку прочие биты регистра IFG1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Не было запроса прерывания 1 Есть запрос прерывания
	Бит 0	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

ГЛАВА 6

КОНТРОЛЛЕР DMA

Контроллер DMA позволяет осуществлять пересылку данных без участия ЦПУ. В этой главе описывается функционирование контроллера DMA, реализованного в микроконтроллерах семейства MSP430x2xx.

6.1. Введение

Контроллер прямого доступа к памяти (Direct Memory Access — DMA) осуществляет пересылку данных без использования ЦПУ с одного адреса на другой в пределах всего адресного пространства. К примеру, контроллер DMA может пересылать результаты преобразования из буфера модуля ADC12 в ОЗУ.

Модели, в которых содержится контроллер DMA, могут иметь один, два или три канала прямого доступа к памяти. В связи с этим некоторые возможности контроллера, описанные в данной главе, доступны не во всех микроконтроллерах семейства.

Использование контроллера DMA может увеличить пропускную способность периферийных модулей. Кроме того, применение контроллера может уменьшить потребление устройства, позволяя осуществлять передачу данных к/от периферийных модулей, не выводя ЦПУ из режима пониженного энергопотребления.

Контроллер DMA имеет следующие особенности:

- до трёх независимых каналов передачи данных;
- конфигурируемые приоритеты каналов DMA;
- для пересылки одного значения требуется всего два такта MCLK;
- поддерживается пересылка между адресатами одинаковой (байт, слово) и разной разрядности;
- размер блока — до 65 535 байт или слов;
- большое число источников, инициирующих пересылку данных;
- инициация пересылки по фронту или по уровню;
- четыре режима адресации;
- пересылка одиночных значений, блоков или групп блоков.

Блок-схема контроллера DMA приведена на **Рис. 6.1**.

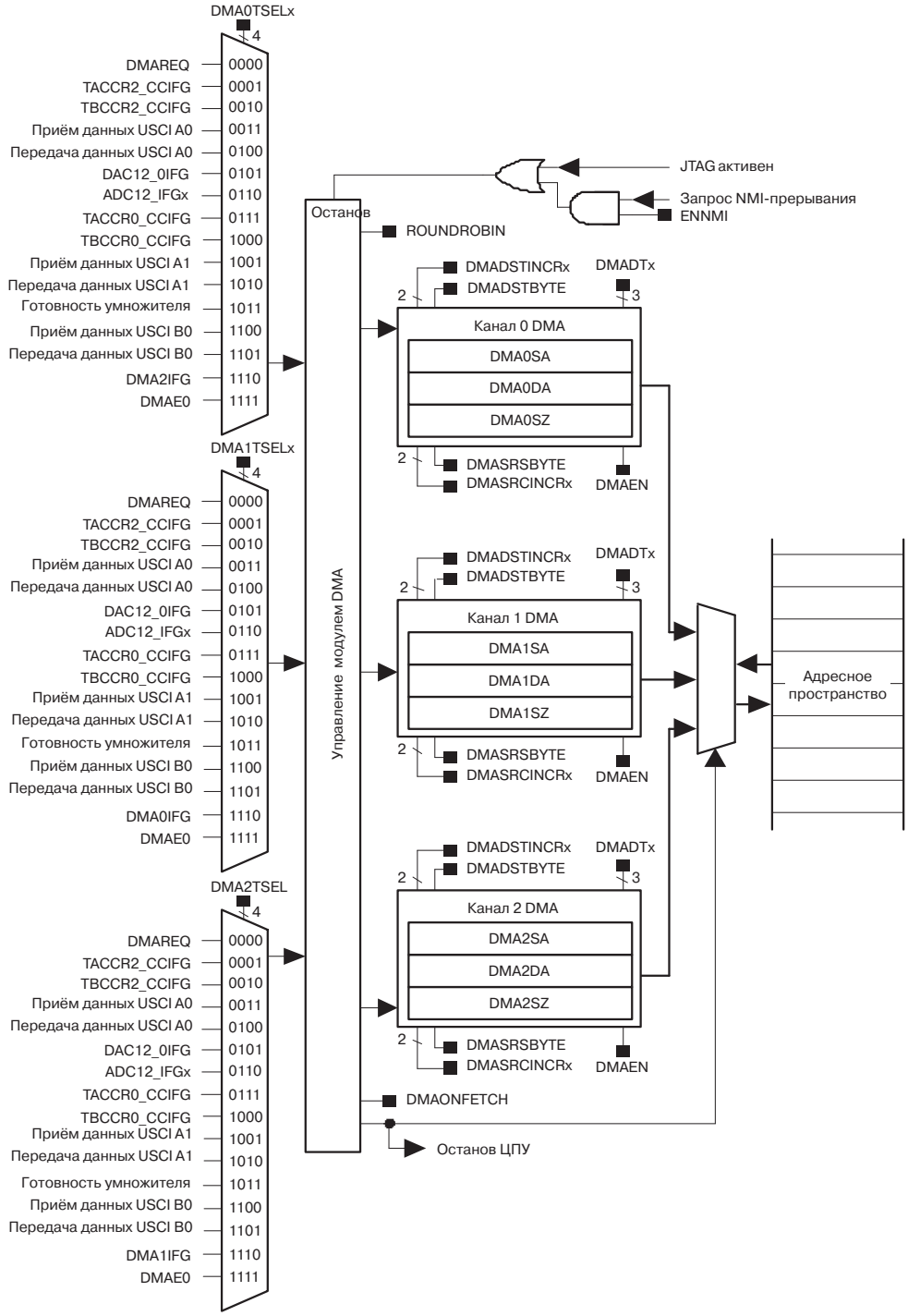


Рис. 6.1. Блок-схема контроллера DMA.

6.2. Функционирование контроллера DMA

Конфигурирование контроллера DMA осуществляется пользовательской программой. Настройка контроллера DMA и его функционирование рассматриваются в следующих подразделах.

6.2.1. Режимы адресации контроллера DMA

Контроллер DMA поддерживает четыре режима адресации. Режим адресации задаётся независимо для каждого канала DMA. К примеру, канал 0 может осуществлять пересылку между двумя фиксированными адресами, в то время как канал 1 используется для пересылки данных между двумя блоками адресов. Доступны следующие режимы адресации (**Рис. 6.2**):

- фиксированный адрес → фиксированный адрес;
- фиксированный адрес → блок адресов;
- блок адресов → фиксированный адрес;
- блок адресов → блок адресов.

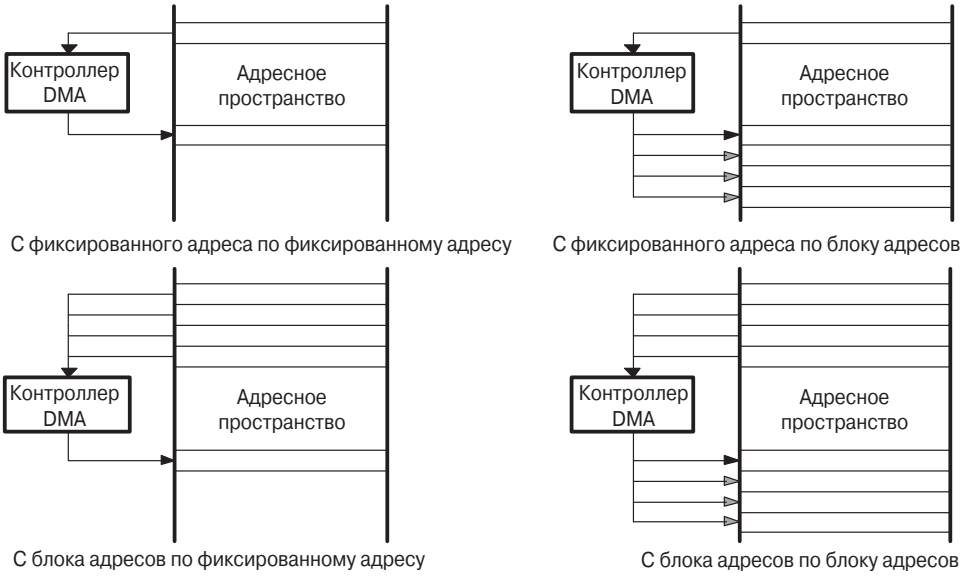


Рис. 6.2. Режимы адресации DMA.

Режимы адресации задаются при помощи битов DMASRCINCRx и DMADSTINCRx. Биты DMASRCINCRx определяют, будет ли адрес источника инкрементироваться, декрементироваться или оставаться неизменным после очередной пересылки. Биты DMADSTINCRx определяют то же самое для адреса назначения.

Пересылка может осуществляться между адресатами одинаковой (байт—байт, слово—слово) и разной (байт—слово, слово—байт) разрядности. При пересылке двухбайтного значения в однобайтное передаётся только младший байт источника. При пересылке однобайтного значения в двухбайтное старший байт получателя сбрасывается.

6.2.2. Режимы пересылки контроллера DMA

Контроллер DMA поддерживает шесть режимов пересылки данных, которые определяются битами DMADTx в соответствии с **Табл. 6.1**. Режим пересылки задаётся индивидуально для каждого канала DMA. Так, канал 0 может использовать режим пересылки одиночных значений, одновременно с этим канал 1 может работать в режиме пересылки группы блоков, а канал 2 — в режиме циклической пересылки блока данных.

Разрядность пересылаемых данных определяется битами DSTBYTE (приёмник) и SRCBYTE (источник) регистра DMAxCTL. Как источник, так и приёмник может быть однобайтным либо двухбайтным. Пересылка может осуществляться между однобайтными адресатами, между двухбайтными адресатами, а также между адресатами разной разрядности.

Таблица 6.1. **Режимы пересылки DMA**

DMADTx	Режим пересылки	Описание
000	Пересылка одиночных значений	Каждая пересылка требует отдельного запуска. Бит DMAEN автоматически сбрасывается после осуществления DMAxSZ-пересылок
001	Пересылка блоков	Весь блок пересылается по одному сигналу запуска. Бит DMAEN автоматически сбрасывается после пересылки блока
010, 011	Пакетная пересылка блоков	ЦПУ периодически активируется во время пересылки блока. Бит DMAEN автоматически сбрасывается после пересылки блока
100	Циклическая пересылка одиночных значений	Каждая пересылка требует отдельного запуска. Бит DMAEN остаётся установленным
101	Циклическая пересылка блоков	Весь блок пересылается по одному сигналу запуска. Бит DMAEN остаётся установленным
110, 111	Циклическая пересылка группы блоков	ЦПУ периодически активируется во время пересылки блока. Бит DMAEN остаётся установленным

Пересылка одиночных значений

В режиме пересылки одиночных значений пересылка каждого байта/слова запускается индивидуально. Диаграмма состояний для этого режима показана на **Рис. 6.3**.

Регистр DMAxSZ используется для задания числа пересылок. Биты DMADSTINCRx и DMASRCINCRx определяют, будет ли адрес приёмника и источника инкрементироваться либо декрементироваться после каждой пересылки. Если DMAxSZ = 0, то пересылка не производится.

Содержимое регистров DMAxSA, DMAxDA и DMAxSZ копируется во временные регистры. Временно сохранённые значения DMAxSA и DMAxDA инкрементируются или декрементируются после каждой пересылки. Регистр DMAxSZ декрементируется после каждой пересылки. При достижении нулевого значения он перезагружается из соответствующего временного регистра. Одновременно устанавливается флаг DMAIFG соответствующего канала. При DMADTx = 0 бит DMAEN сбрасывается автоматически при достижении регистром DMAxSZ нулевого значения и должен быть установлен повторно для выполнения следующей пересылки.

В режиме циклической передачи одиночных значений контроллер DMA автоматически не отключается (DMAEN = 1), и пересылка осуществляется каждый раз при появлении события, инициирующего запуск.

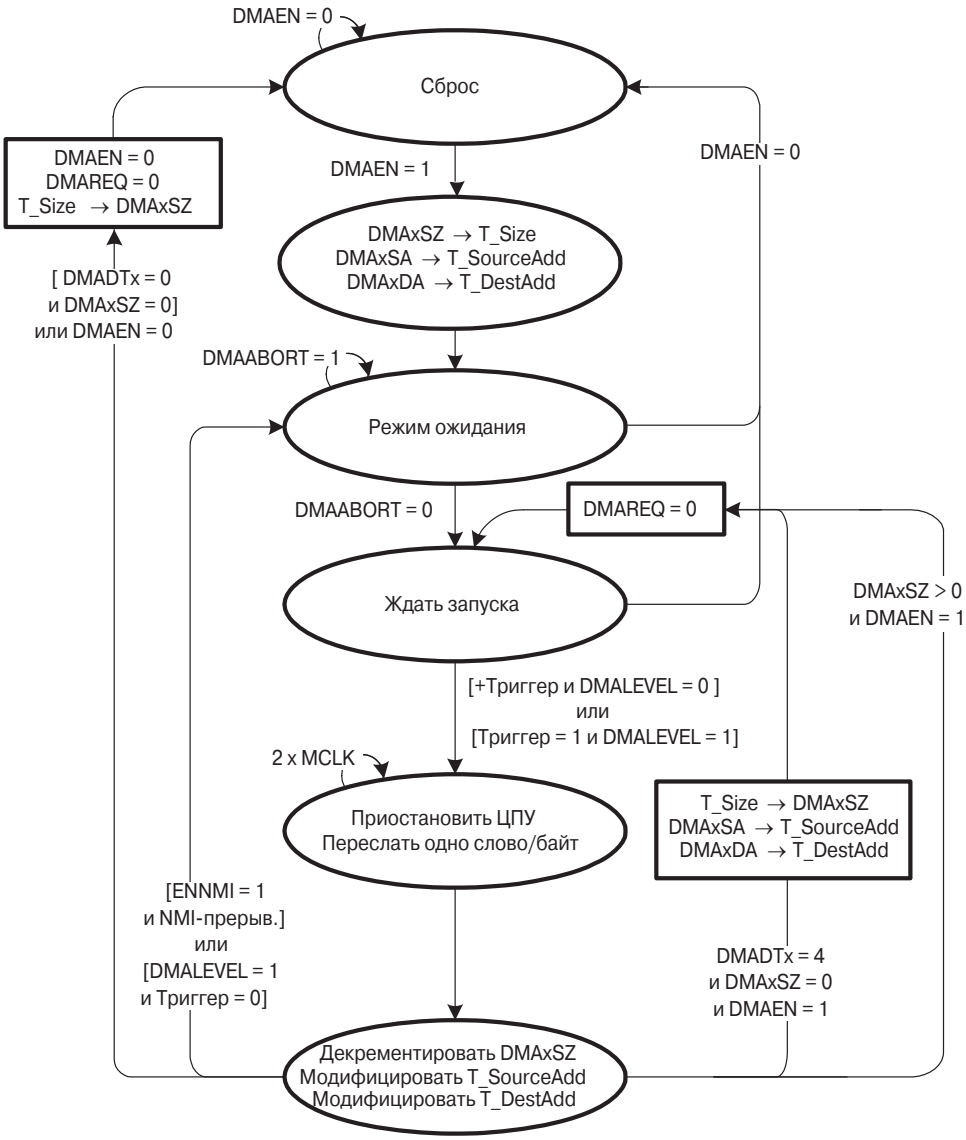


Рис. 6.3. Диаграмма состояний контроллера DMA для режима пересылки одиночных значений.

Пересылка блоков

В режиме пересылки блоков при каждом запуске выполняется пересылка целого блока данных. При DMADTx = 1 бит DMAEN сбрасывается автоматически при завершении пересылки блока и должен быть установлен повторно, чтобы можно было снова запустить пересылку данных. После начала пересылки блока и до момента её завершения повторные сигналы запуска игнорируются. Диаграмма состояний для этого режима приведена на **Рис. 6.4**.

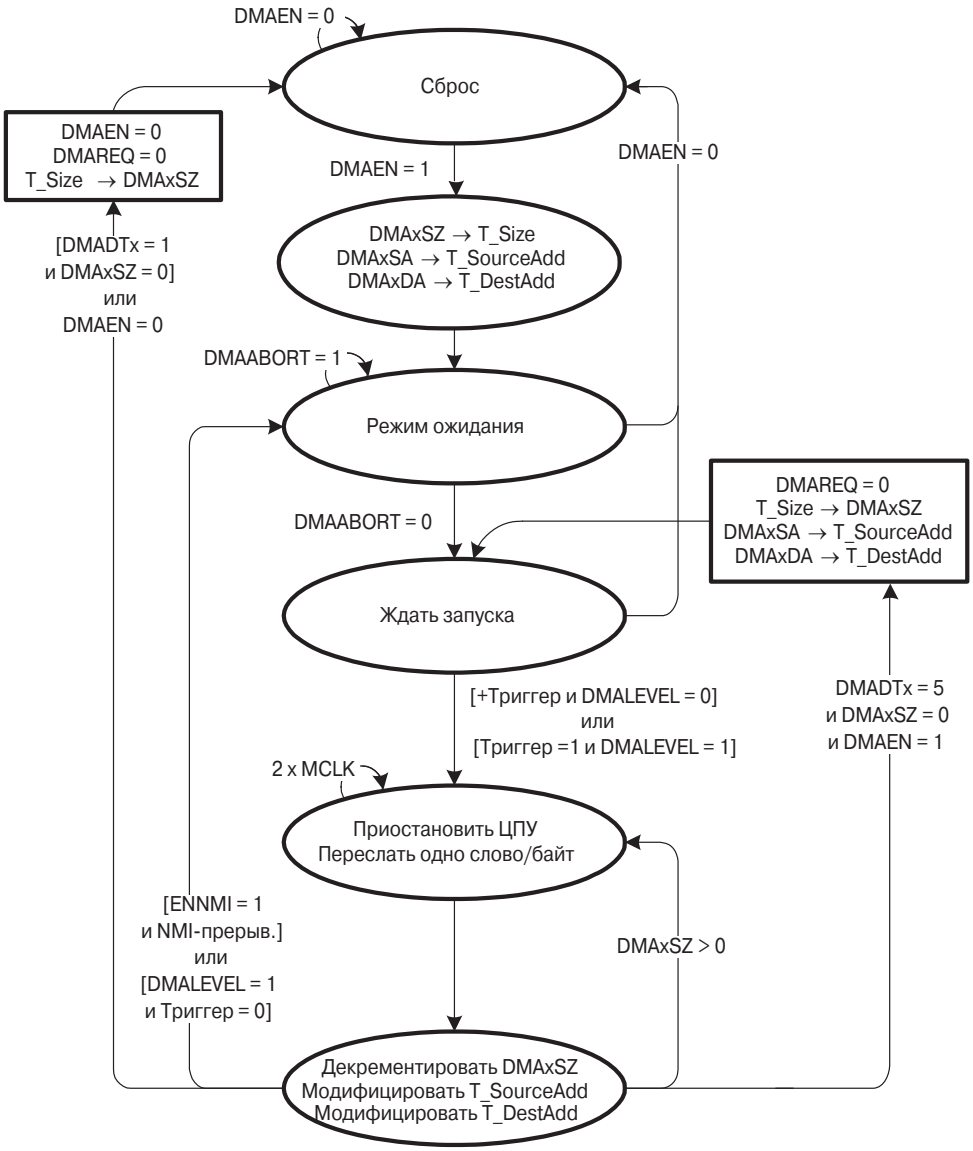


Рис. 6.4. Диаграмма состояний контроллера DMA для режима пересылки блоков.

Регистр DMAxSZ используется для задания размера блока. Биты DMADSTINCRx и DMASRCINCRx определяют, будет ли адрес приёмника и источника инкрементироваться либо декрементироваться после каждой пересылки. Если DMAxSZ = 0, то пересылка не производится.

Содержимое регистров DMAxSA, DMAxDA и DMAxSZ копируется во временные регистры. Временно сохранённые значения DMAxSA и DMAxDA инкрементируются или декрементируются после пересылки каждого элемента блока. Регистр DMAxSZ декрементируется после пересылки каждого элемента блока,

показывая тем самым число элементов, которые осталось переслать. При достижении нулевого значения регистр DMAxSZ перезагружается из соответствующего временного регистра. Одновременно устанавливается флаг DMAIFG соответствующего канала.

На время пересылки блока работа ЦПУ приостанавливается. Для пересылки одного блока данных требуется 2 × MCLK × DMAxSZ тактов. После завершения пересылки блока работа ЦПУ возобновляется с его предыдущего состояния.

В режиме циклической пересылки блоков бит DMAEN после завершения пересылки блока остаётся в установленном состоянии. Поэтому первое же событие, инициирующее пересылку, вызовет повторную пересылку блока.

Пакетная пересылка блоков

В режиме пакетной пересылки блоков пересылка содержимого блока чередуется с активностью ЦПУ. Каждый раз после пересылки очередной порции из четырех байтов/слов ЦПУ включается на 2 такта MCLK, т.е. ЦПУ работает на 20% мощности. После завершения пересылки блока ЦПУ возобновляет работу в нормальном режиме, а бит DMAEN сбрасывается. Этот бит должен быть установлен повторно, чтобы можно было снова запустить пересылку данных. После начала пересылки блока и до момента её завершения повторные сигналы запуска игнорируются. Диаграмма состояний для этого режима приведена на **Рис. 6.5**.

Регистр DMAxSZ используется для задания размера блока. Биты DMADSTINCRx и DMASRCINCRx определяют, будет ли адрес приёмника и источника инкрементироваться либо декрементироваться после каждой пересылки. Если DMAxSZ = 0, то пересылка не производится.

Содержимое регистров DMAxSA, DMAxDA и DMAxSZ копируется во временные регистры. Временно сохранённые значения DMAxSA и DMAxDA инкрементируются или декрементируются после пересылки каждого элемента блока. Регистр DMAxSZ декрементируется после пересылки каждого элемента блока, показывая тем самым число элементов, которые осталось переслать. При достижении нулевого значения регистр DMAxSZ перезагружается из соответствующего временного регистра. Одновременно устанавливается флаг DMAIFG соответствующего канала.

В режиме циклической пакетной пересылки блоков бит DMAEN после завершения пересылки блока остаётся в установленном состоянии. Более того, для инициации новой пакетной пересылки не требуется никаких дополнительных событий — повторная пересылка блока начинается сразу же после завершения текущей пересылки. Процесс пересылки данных может быть остановлен либо сбросом бита DMAEN, либо возникновением немаскируемого прерывания (при установленном бите ENNMI). В режиме циклической пакетной пересылки блоков ЦПУ работает на 20% мощности до тех пор, пока процесс пересылки данных не будет остановлен.

6.2.3. Инициация передачи данных с использованием DMA

Выбор источника, инициирующего пересылку данных, осуществляется с помощью битов DMAxTSELx независимо для каждого канала DMA (**Табл. 6.2**).

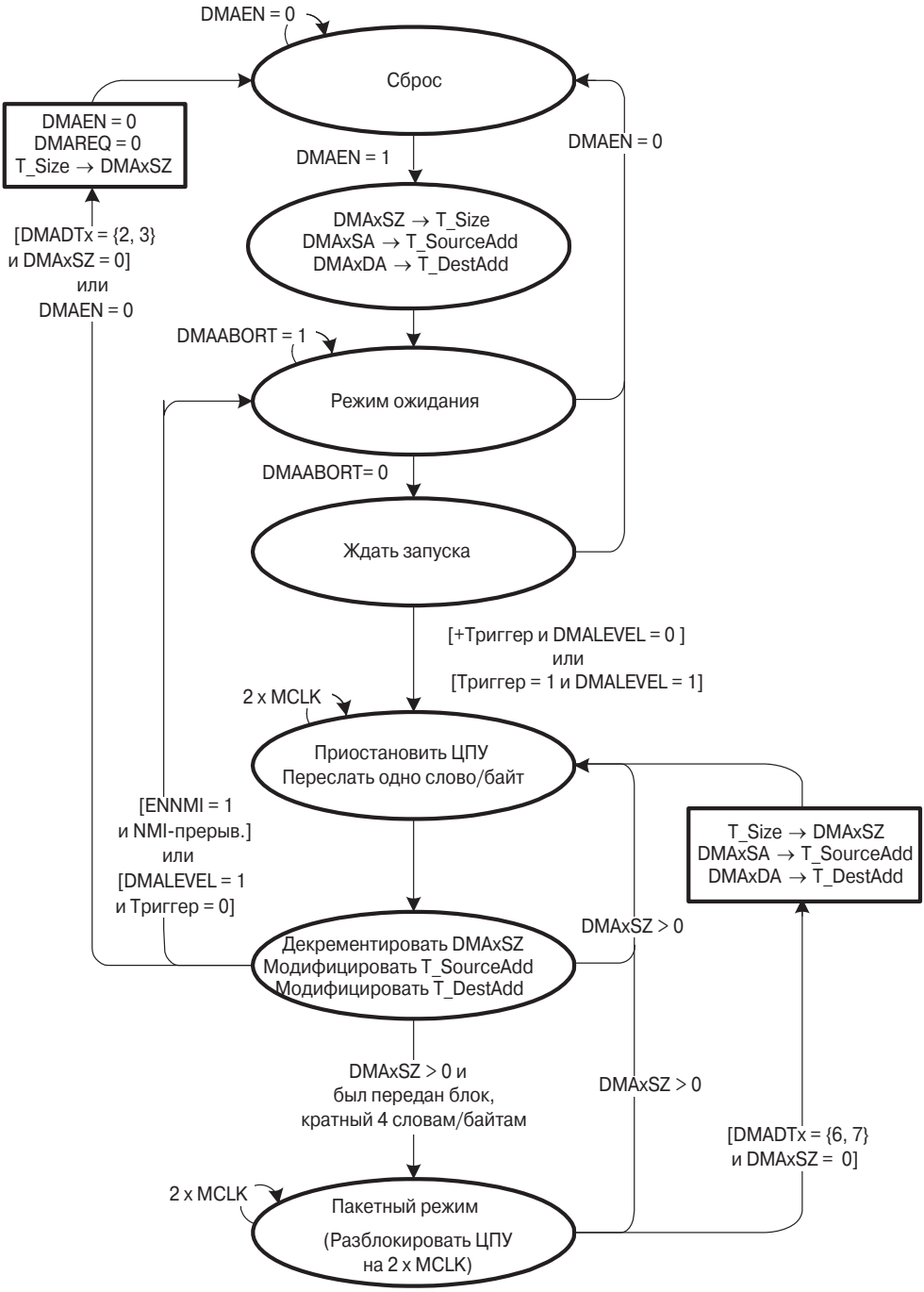


Рис. 6.5. Диаграмма состояний контроллера DMA для режима пакетной пересылки блоков.

Таблица 6.2. **Источники запуска для контроллера DMA**

DMAxTSELx	Описание
0000	Пересылка инициируется установкой бита DMAREQ. После начала пересылки бит DMAREQ автоматически сбрасывается
0001	Пересылка инициируется установкой флага CCIFG регистра TACCR2. После начала пересылки флаг CCIFG автоматически сбрасывается. Если бит CCIE регистра TACCR2 установлен, то установка флага CCIFG не приведёт к инициации пересылки
0010	Пересылка инициируется установкой флага CCIFG регистра TBCCR2. После начала пересылки флаг CCIFG автоматически сбрасывается. Если бит CCIE регистра TBCCR2 установлен, то установка флага CCIFG не приведёт к инициации пересылки
0011	Пересылка инициируется при приёме нового значения по последовательному интерфейсу. Модели с модулем USCI_A0: пересылка запускается при приёме модулем USCI_A0 нового значения. После начала пересылки флаг UCA0RXIFG автоматически сбрасывается. Если бит UCA0RXIE установлен, то установка флага UCA0RXIFG не приведёт к инициации пересылки
0100	Пересылка инициируется при готовности последовательного интерфейса к передаче нового значения. Модели с модулем USCI_A0: пересылка запускается при готовности модуля USCI_A0 к передаче нового значения. После начала пересылки флаг UCA0TXIFG автоматически сбрасывается. Если бит UCA0TXIE установлен, то установка флага UCA0TXIFG не приведёт к инициации пересылки
0101	Пересылка инициируется установкой флага DAC12IFG регистра DAC12_0CTL. После начала пересылки флаг DAC12IFG автоматически сбрасывается. Если бит DAC12IE регистра DAC12_0CTL установлен, то установка флага DAC12IFG не приведёт к инициации пересылки
0110	Пересылка инициируется установкой флага ADC12IFGx. В режиме одноканального преобразования установка соответствующего флага ADC12IFGx инициирует пересылку. В режиме последовательного преобразования нескольких каналов пересылка инициируется установкой флага ADC12IFGx при обработке последнего канала. Пересылка данных начинается после завершения преобразования и установки флага ADC12IFGx. Программная установка флага ADC12IFGx не приведёт к инициации пересылки. Все флаги ADC12IFGx автоматически сбрасываются при обращении контроллера DMA к соответствующим регистрам ADC12MEMx
0111	Пересылка инициируется установкой флага CCIFG регистра TACCR0. После начала пересылки флаг CCIFG автоматически сбрасывается. Если бит CCIE регистра TACCR0 установлен, то установка флага CCIFG не приведёт к инициации пересылки
1000	Пересылка инициируется установкой флага CCIFG регистра TBCCR0. После начала пересылки флаг CCIFG автоматически сбрасывается. Если бит CCIE регистра TBCCR0 установлен, то установка флага CCIFG не приведёт к инициации пересылки
1001	Пересылка инициируется при установке флага UCA1RXIFG. После начала пересылки этот флаг автоматически сбрасывается. Если бит URXIE1 установлен, то установка флага UCA1RXIFG не приведёт к инициации пересылки
1010	Пересылка инициируется при установке флага UCA1TXIFG. После начала пересылки этот флаг автоматически сбрасывается. Если бит UTXIE1 установлен, то установка флага UCA1TXIFG не приведёт к инициации пересылки
1011	Пересылка инициируется при готовности аппаратного умножителя к обработке нового операнда
1100	Пересылка не инициируется. Модели с модулем USCI_B0: пересылка запускается при приёме модулем USCI_B0 нового значения. После начала пересылки флаг UCB0RXIFG автоматически сбрасывается. Если бит UCB0RXIE установлен, то установка флага UCB0RXIFG не приведёт к инициации пересылки
1101	Пересылка не инициируется. Модели с модулем USCI_B0: пересылка запускается при готовности модуля USCI_B0 к передаче нового значения. После начала пересылки флаг UCB0TXIFG автоматически сбрасывается. Если бит UCB0TXIE установлен, то установка флага UCB0TXIFG не приведёт к инициации пересылки
1110	Пересылка инициируется установкой флага DMAxIFG. Флаг DMA0IFG запускает пересылку по 0-му, флаг DMA1IFG — по 1-му, а флаг DMA2IFG — по 2-му каналу. Флаги DMAxIFG после начала пересылки не сбрасываются автоматически
1111	Пересылка инициируется внешним сигналом, подаваемым на вход DMAE0

Содержимое битов DMAxTSELx должно изменяться только при сброшенном бите DMAEN регистра DMACTLx. В противном случае может произойти несанкционированный запуск пересылки данных.

На момент выбора источника запуска соответствующее событие должно быть неактивным, в противном случае пересылка данных запущена не будет. Например, если выбран запуск по изменению бита CCIFG регистра TACCR2 и этот бит уже установлен, то пересылка не будет запущена до тех пор, пока данный бит не будет установлен повторно.

Запуск по фронту

При DMALEVEL = 0 запуск пересылки производится по нарастающему фронту соответствующего сигнала. В режиме пересылки одиночных значений каждая пересылка требует отдельного запуска. В режиме пересылки блоков (обычной или пакетной) для пересылки всего блока данных достаточно одного запуска.

Запуск по уровню

При DMALEVEL = 1 запуск пересылки производится по уровню соответствующего сигнала. Для корректной работы контроллера DMA запуск по уровню можно использовать только в том случае, если в качестве сигнала запуска используется внешний сигнал DMAE0. При этом повторные пересылки будут инициироваться до тех пор, пока этот сигнал имеет ВЫСОКИЙ уровень при установленном бите DMAEN.

Сигнал запуска должен удерживаться в состоянии ВЫСОКОГО уровня до завершения пересылки блока. Если во время пересылки блока (как в обычном, так и в пакетном режиме) сигнал перейдёт в состояние НИЗКОГО уровня, то работа контроллера DMA будет приостановлена, и он будет находиться в этом состоянии до повторного появления ВЫСОКОГО уровня или до тех пор, пока регистры контроллера не будут изменены программно. Если к моменту повторного появления ВЫСОКОГО уровня сигнала регистры контроллера DMA изменены не были, то пересылка возобновится с того же самого места, в котором была прервана.

При DMALEVEL = 1 рекомендуется выбирать режимы пересылки, которым соответствуют значения DMADTx = {0, 1, 2, 3}, поскольку в этих режимах бит DMAEN автоматически сбрасывается после пересылки требуемых данных.

Прерывание команд ЦПУ при использовании DMA

Бит DMAONFETCH определяет поведение ЦПУ во время пересылки данных с использованием DMA. При DMAONFETCH = 0 останов ЦПУ и запуск пересылки производятся сразу же после появления заданного события. При DMAONFETCH = 1 центральный процессор сначала завершает выполнение текущей команды и только после этого контроллер DMA останавливает ЦПУ и начинает пересылку данных.

Примечание. Бит DMAONFETCH и запись в флэш-память
Если контроллер DMA используется для записи в флэш-память, то бит DMAONFETCH обязательно должен быть установлен. В противном случае результат операции может быть непредсказуемым.

6.2.4. Прерывание DMA-пересылок

Остановить текущую пересылку данных можно двумя способами:

- если установлен бит ENNMI регистра DMACTL1, то процесс пересылки данных (независимо от режима пересылки) может быть остановлен при появлении немаскируемого прерывания;
- пакетная пересылка блока данных может быть остановлена сбросом бита DMAEN.

6.2.5. Приоритеты каналов DMA

По умолчанию приоритеты каналов DMA располагаются в следующем порядке: DMA0 – DMA1 – DMA2. Если одновременно происходят или ожидают обработки два или три события, запускающие пересылку данных, то сначала завершает пересылку (одиночного значения, блока или блока в пакетном режиме) канал с наивысшим приоритетом, затем канал со средним приоритетом и только потом — канал с наименьшим приоритетом. Если сигнал запуска для канала с более высоким приоритетом появится во время пересылки данных по другому каналу, то текущая пересылка прервана не будет. Пересылка данных по каналу с более высоким приоритетом начнётся только после завершения текущей пересылки.

Приоритеты каналов DMA определяются битом ROUNDROBIN. Если бит ROUNDROBIN установлен, то каналу, завершившему пересылку данных, назначается наименьший приоритет. Порядок приоритетов каналов всегда остаётся неизменным, DMA0 – DMA1 – DMA2, например:

Приоритеты каналов DMA	Завершилась пересылка по каналу	Новые приоритеты каналов DMA
DMA0 – DMA1 – DMA2	DMA1	DMA2 – DMA0 – DMA1
DMA2 – DMA0 – DMA1	DMA2	DMA0 – DMA1 – DMA2
DMA0 – DMA1 – DMA2	DMA0	DMA1 – DMA2 – DMA0

Если бит ROUNDROBIN сброшен, то каналы имеют приоритеты, назначаемые по умолчанию.

6.2.6. Длительность DMA-пересылки

Перед пересылкой одиночного значения или пересылкой целого блока (в обычном или пакетном режиме) контроллеру DMA для синхронизации требуется один или два такта сигнала MCLK. Для пересылки каждого одно/двухбайтного значения требуется два такта MCLK после синхронизации и один такт после завершения пересылки. Поскольку для тактирования контроллера DMA используется сигнал MCLK, продолжительность операций DMA зависит от режима работы MSP430 и конфигурации модуля синхронизации.

Если источник тактового сигнала MCLK активен, а ЦПУ выключен, то контроллер DMA будет использовать данный источник, не включая ЦПУ. Если же источник MCLK выключен, то контроллер DMA разрешит формирование MCLK из сигнала DCOCLK на время пересылки одного значения или блока данных. При этом ЦПУ так и остаётся в выключенном состоянии, а сигнал MCLK отключается после завершения пересылки. Максимальные значения длительности одной DMA-пересылки для всех режимов работы микроконтроллера приведены в **Табл. 6.3**.

Таблица 6.3. **Максимальное время пересылки одного значения с использованием DMA**

Режим работы ЦПУ	Источник тактового сигнала	Максимальное время пересылки
Активный режим	MCLK = DCOCLK	4 такта MCLK
Активный режим	MCLK = LFXT1CLK	4 такта MCLK
Режим пониженного потребления LPM0/1	MCLK = DCOCLK	5 тактов MCLK
Режим пониженного потребления LPM3/4	MCLK = DCOCLK	5 тактов MCLK + 6 мкс*
Режим пониженного потребления LPM0/1	MCLK = LFXT1CLK	5 тактов MCLK
Режим пониженного потребления LPM3	MCLK = LFXT1CLK	5 тактов MCLK
Режим пониженного потребления LPM4	MCLK = LFXT1CLK	5 тактов MCLK + 6 мкс*

* Дополнительные 6 мкс требуются для запуска DCO. Это значение параметра $t_{(LPMx)}$, приводимого в справочной документации.

6.2.7. Функционирование DMA и прерывания

Пересылка данных с использованием DMA не прерывается системными прерываниями. Обслуживание всех системных прерываний производится после завершения пересылки. Работа контроллера DMA может быть прервана только немаскируемым прерыванием, если установлен бит ENNMI.

В то же время любые DMA-пересылки прерывают выполнение процедур обработки системных прерываний. Если прерывание какой-либо процедуры обработки прерывания или же обычной процедуры недопустимо, то перед выполнением такой процедуры контроллер DMA необходимо выключать.

6.2.8. Прерывания контроллера DMA

Каждому каналу DMA назначен свой флаг прерывания DMAIFG. Данный флаг устанавливается в любом режиме, когда содержимое соответствующего регистра DMAxSZ уменьшается до нуля. Если при этом установлены соответствующий бит DMAIE и бит GIE, то генерируется запрос прерывания.

Все флаги DMAIFG привязаны к единственному вектору прерывания контроллера DMA, а в ряде моделей этот вектор может использоваться ещё и совместно с другими модулями. Для получения более подробной информации обращайтесь к документации на конкретный микроконтроллер. При использовании таких микроконтроллеров программа должна проверять состояние флага DMAIFG и флагов прерываний соответствующих модулей для определения источника прерывания. Флаг DMAIFG не сбрасывается автоматически — это необходимо делать программно.

Кроме того, в некоторых моделях имеется регистр DMAIV. Каждый флаг DMAIFG имеет свой приоритет (DMA0IFG — наивысший), и все они назначены единственному вектору прерывания. При генерации прерывания с наивысшим приоритетом в регистр DMAIV заносится определённое значение. Впоследствии это значение можно использовать для автоматического перехода к соответствующей секции программы. Запрещённые прерывания DMA не влияют на содержимое регистра DMAIV.

При любом обращении к регистру DMAIV как для чтения, так и для записи, флаг прерывания с наивысшим приоритетом, ожидающего обработки, автоматически сбрасывается. Если имеется ещё один установленный флаг, то сразу же после обработки текущего прерывания будет сгенерировано новое прерывание. Предположим, например, что канал DMA0 имеет наивысший приоритет. Если на момент обращения к регистру DMAIV в процедуре обработки прерывания были установлены флаги DMA0IFG и DMA2IFG, то автоматически будет сброшен флаг DMA0IFG. После выполнения команды `RETI` процедуры обработки прерывания флаг DMA2IFG вызовет генерацию нового прерывания.

Пример использования регистра DMAIV

В следующем примере показано рекомендуемое использование регистра DMAIV и приведена информация о накладных расходах на обработку прерывания. Содержимое регистра DMAUV прибавляется к PC для автоматического перехода к соответствующей процедуре.

Значения у правых границ строк показывают число тактов ЦПУ, требуемое для выполнения каждой команды. Накладные расходы на обработку различных источников прерывания включают в себя задержку обслуживания прерывания и время, требуемое для возврата из прерывания, но не время, необходимое для выполнения собственно задачи.

```
; Обработчик прерывания для DMA0IFG, DMA1IFG, DMA2IFG                Тактов
DMA_HND  ...              ; Задержка обработки прерывания              6
         ADD &DMAIV,PC    ; Прибавляем смещение в таблице переходов    3
         RETI             ; Вектор 0: Нет прерывания                   5
         JMP DMA0_HND     ; Вектор 2: DMA канал 0                      2
         JMP DMA1_HND     ; Вектор 4: DMA канал 1                      2
         JMP DMA2_HND     ; Вектор 6: DMA канал 2                      2
         RETI             ; Вектор 8: Зарезервировано                  5
         RETI             ; Вектор 10: Зарезервировано                 5
         RETI             ; Вектор 12: Зарезервировано                 5
         RETI             ; Вектор 14: Зарезервировано                 5
DMA2_HND                  ; Вектор 6: DMA канал 2
         ...              ; Здесь начало задачи
         RETI             ; Возвращаемся в основную программу          5
DMA1_HND                  ; Вектор 4: DMA канал 1
         ...              ; Здесь начало задачи
         RETI             ; Возвращаемся в основную программу          5
DMA0_HND                  ; Вектор 2: DMA канал 0
         ...              ; Здесь начало задачи
         RETI             ; Возвращаемся в основную программу          5
```

6.2.9. Использование модуля USCI_B в режиме I²C с контроллером DMA

Модуль USCI_B в режиме I²C обеспечивает два источника запуска для контроллера DMA. Модуль может инициировать пересылку при получении новых данных по шине I²C и при необходимости отправки данных.

Пересылка запускается при установке флага UCB0RXIFG. Этот флаг сбрасывается автоматически после подтверждения пересылки контроллером DMA. Если установлен бит UCB0RXIE, то установка флага UCB0RXIFG не приведёт к запуску пересылки.

Пересылка запускается при установке флага UCB0TXIFG. Этот флаг сбрасывается автоматически после подтверждения пересылки контроллером DMA. Если установлен бит UCB0TXIE, то установка флага UCB0TXIFG не приведёт к запуску пересылки.

6.2.10. Использование модуля ADC12 с контроллером DMA

Микроконтроллеры семейства MSP430, имеющие встроенный контроллер DMA, могут автоматически копировать данные из регистра ADC12MEMx по другому адресу в памяти. Пересылка данных с использованием DMA осуществляется без вмешательства ЦПУ и независимо от всех режимов пониженного энергопотребления. Контроллер DMA увеличивает пропускную способность модуля ADC12 и расширяет возможности устройств с низким потреблением, позволяя не включать ЦПУ во время пересылки данных.

Для запуска DMA-пересылок может использоваться любой из флагов ADC12IFGx. Если биты CONSEQx = {0, 2}, то инициировать пересылку может флаг ADC12IFGx, который соответствует регистру ADC12MEMx, используемому для хранения результата преобразования. Если биты CONSEQx = {1, 3}, то инициировать пересылку может флаг ADC12IFGx, который соответствует последнему регистру ADC12MEMx в последовательности. Каждый из флагов ADC12IFGx сбрасывается автоматически при обращении контроллера DMA к соответствующему регистру ADC12MEMx.

6.2.11. Использование модуля DAC12 с контроллером DMA

Микроконтроллеры семейства MSP430, имеющие встроенный контроллер DMA, могут автоматически загружать данные в регистр DAC12_xDAT. Пересылка данных с использованием DMA осуществляется без вмешательства ЦПУ и независимо от всех режимов пониженного энергопотребления. Контроллер DMA увеличивает пропускную способность модуля DAC12 и расширяет возможности устройств с низким потреблением, позволяя не включать ЦПУ во время пересылки данных.

Совместное использование контроллера DMA с модулем DAC12 может оказаться полезным в приложениях, требующих генерации периодических сигналов. В частности, в случае генерации синусоидального сигнала значения функции синуса часто хранятся в виде таблицы. Контроллер DMA может автоматически с за-

данной периодичностью пересылать данные из таблицы в модуль DAC12, формируя синусоидальный сигнал вообще без использования ЦПУ. Флаг DAC12IFG регистра DAC12_xCTL сбрасывается автоматически при обращении контроллера DMA к регистру DAC12_xDAT.

6.2.12. Запись в флэш-память с использованием контроллера DMA

Микроконтроллеры семейства MSP430, имеющие встроенный контроллер DMA, могут автоматически пересылать данные в флэш-память. Пересылка данных с использованием DMA осуществляется без вмешательства ЦПУ и независимо от всех режимов пониженного энергопотребления. Контроллер DMA осуществляет пересылку одно- и двухбайтных значений в флэш-память. Управление процессом записи осуществляется контроллером флэш-памяти. Для успешной записи контроллер флэш-памяти должен быть настроен до начала пересылки данных, а сама флэш-память должна быть незанятой. Более подробно процесс настройки контроллера флэш-памяти на запись описан в главе 7 «Контроллер флэш-памяти».

6.3. Регистры контроллера DMA

Список регистров контроллера DMA приведён в **Табл. 6.4**.

Таблица 6.4. **Регистры контроллера DMA**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 контроллера DMA	DMACTL0	Чтение/запись	0122h	Сбрасывается после POR
Регистр управления 1 контроллера DMA	DMACTL1	Чтение/запись	0124h	Сбрасывается после POR
Регистр вектора прерывания контроллера DMA	DMAIV	Только чтение	0126h	Сбрасывается после POR
Регистр управления 0-го канала DMA	DMA0CTL	Чтение/запись	01D0h	Сбрасывается после POR
Регистр адреса источника 0-го канала DMA	DMA0SA	Чтение/запись	01D2h	Не изменяется
Регистр адреса приёмника 0-го канала DMA	DMA0DA	Чтение/запись	01D6h	Не изменяется
Регистр размера пересылки 0-го канала DMA	DMA0SZ	Чтение/запись	01DAh	Не изменяется
Регистр управления 1-го канала DMA	DMA1CTL	Чтение/запись	01DCh	Сбрасывается после POR
Регистр адреса источника 1-го канала DMA	DMA1SA	Чтение/запись	01DEh	Не изменяется
Регистр адреса приёмника 1-го канала DMA	DMA1DA	Чтение/запись	01E2h	Не изменяется
Регистр размера пересылки 1-го канала DMA	DMA1SZ	Чтение/запись	01E6h	Не изменяется
Регистр управления 2-го канала DMA	DMA2CTL	Чтение/запись	01E8h	Сбрасывается после POR
Регистр адреса источника 2-го канала DMA	DMA2SA	Чтение/запись	01EAh	Не изменяется
Регистр адреса приёмника 2-го канала DMA	DMA2DA	Чтение/запись	01EEh	Не изменяется
Регистр размера пересылки 2-го канала DMA	DMA2SZ	Чтение/запись	01F2h	Не изменяется

DMACTL0, регистр управления 0 контроллера DMA

15	14	13	12	11	10	9	8
Reserved				**DMA2TSELx**			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DMA1TSELx				**DMA0TSELx**			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Reserved Биты 15...12 Зарезервированы

DMA2TSELx Биты 11...8 Выбор источника DMA. Эти биты определяют источник, инициирующий DMA-пересылку.

- 0000 Бит DMAREQ (программный запуск)
- 0001 Бит CCIFG регистра TACCR2
- 0010 Бит CCIFG регистра TBCCR2
- 0011 Бит UCA0RXIFG (приняты данные по последовательному каналу)
- 0100 Бит UCA0TXIFG (готовы данные для передачи по последовательному каналу)
- 0101 Бит DAC12IFG регистра DAC12_0CTL
- 0110 Бит ADC12IFGx регистра ADC12IFG
- 0111 Бит CCIFG регистра TACCR0
- 1000 Бит CCIFG регистра TBCCR0
- 1001 Бит UCA1RXIFG (приняты данные по последовательному каналу)
- 1010 Бит UCA1TXIFG (готовы данные для передачи по последовательному каналу)
- 1011 Готовность аппаратного умножителя
- 1100 Бит UCB0RXIFG (приняты данные по последовательному каналу)
- 1101 Бит UCB0TXIFG (готовы данные для передачи по последовательному каналу)
- 1110 Бит DMA0IFG инициирует пересылку по 0-му каналу DMA
  Бит DMA1IFG инициирует пересылку по 1-му каналу DMA
  Бит DMA2IFG инициирует пересылку по 2-му каналу DMA
- 1111 Внешний сигнал DMAE0

DMA1TSELx Биты 7...4 То же, что и для DMA2TSELx

DMA2TSELx Биты 3...0 То же, что и для DMA2TSELx

DMACTL1, регистр управления 1 контроллера DMA

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**0**	**DMA ONFETCH**	**ROUND ROBIN**	**ENNMI**
r0	r0	r0	r0	r0	rw–(0)	rw–(0)	rw–(0)

	Биты 15...3	Зарезервированы. Только для чтения. Всегда читаются как 0.
DMA ONFETCH	Бит 2	Запуск DMA во время выборки команды. 0 Пересылка запускается немедленно 1 Пересылка запускается при выборке следующей после появления сигнала запуска команды
ROUND ROBIN	Бит 1	Циклическое назначение приоритетов. Этот бит разрешает назначение приоритетов каналов DMA по циклическому алгоритму. 0 Приоритеты каналов располагаются в порядке DMA0 – DMA1 – DMA2 1 Приоритеты каналов DMA изменяются после каждой пересылки
ENNMI	Бит 0	Разрешение NMI. Этот бит разрешает прерывание DMA-пересылки немаскируемым прерыванием. Если этот бит установлен, то при возникновении немаскируемого прерывания текущая пересылка корректно завершается, последующие пересылки останавливаются, и устанавливается бит DMAABORT. 0 NMI не прерывает DMA-пересылку. 1 NMI прерывает DMA-пересылку.

DMAxCTL, регистр управления канала x DMA

15	14	13	12	11	10	9	8
Reserved	**DMADTx**			**DMADSTINRx**		**DMASRCINCRx**	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DMA DSTBYTE	**DMA SRCBYTE**	**DMALEVEL**	**DMAEN**	**DMAIFG**	**DMAIE**	**DMA ABORT**	**DMAREQ**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Reserved	Бит 15	Зарезервирован.
DMADTx	Биты 14...12	Режим пересылки DMA. 000 Пересылка одиночных значений 001 Пересылка блоков 010 Пакетная пересылка блоков 011 Пакетная пересылка блоков 100 Циклическая пересылка одиночных значений 101 Циклическая пересылка блоков 110 Циклическая пакетная пересылка блоков 111 Циклическая пакетная пересылка блоков

DMA DSTINRx	Биты 11...10	Инкремент конечного адреса. Эти биты разрешают автоматическое декрементирование/инкрементирование адреса приёмника после пересылки очередного байта или слова. При DMADSTBYTE = 1 адрес приёмника увеличивается/уменьшается на единицу. При DMADSTBYTE = 0 адрес приёмника увеличивается/уменьшается на два. Регистр DMAxDA копируется во временный регистр, содержимое которого инкрементируется или декрементируется. Сам регистр DMAxDA не изменяется. 00 Адрес приёмника не изменяется 01 Адрес приёмника не изменяется 10 Адрес приёмника декрементируется 11 Адрес приёмника инкрементируется
DMA SRCINCRx	Биты 9...8	Инкремент исходного адреса. Эти биты разрешают автоматическое декрементирование/инкрементирование адреса источника после пересылки очередного байта или слова. При DMADSTBYTE = 1 адрес источника увеличивается/уменьшается на единицу. При DMADSTBYTE = 0 адрес источника увеличивается/уменьшается на два. Регистр DMAxSA копируется во временный регистр, содержимое которого инкрементируется или декрементируется. Сам регистр DMAxSA не изменяется. 00 Адрес источника не изменяется 01 Адрес источника не изменяется 10 Адрес источника декрементируется 11 Адрес источника инкрементируется
DMA DSTBYTE	Бит 7	Формат приёмника DMA. Этот бит определяет разрядность приёмника. 0 Слово 1 Байт
DMA SRCBYTE	Бит 6	Формат источника DMA. Этот бит определяет разрядность источника. 0 Слово 1 Байт
DMALEVEL	Бит 5	Уровень DMA. Этот бит определяет характер сигнала запуска. 0 Запуск по фронту (нарастающий фронт) 1 Запуск по уровню (ВЫСОКИЙ уровень)
DMAEN	Бит 4	Разрешение DMA. 0 Выключен 1 Включен
DMAIFG	Бит 3	Флаг прерывания DMA. 0 Нет прерывания 1 Есть прерывание
DMAIE	Бит 2	Разрешение прерывания DMA 0 Запрещено 1 Разрешено
DMA ABORT	Бит 1	Аварийный останов DMA. Этот бит показывает, была ли DMA-пересылка прервана немаскируемым прерыванием. 0 DMA-пересылка не прервана 1 DMA-пересылка была прервана NMI
DMAREQ	Бит 0	Запрос DMA. Программный запуск DMA. Бит DMAREQ сбрасывается автоматически. 0 Не запускать DMA 1 Запустить DMA

DMAxSA, регистр адреса источника канала x DMA

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved				DMAxSAx			
r0	r0	r0	r0	rw	rw	rw	rw

15	14	13	12	11	10	9	8
DMAxSAx							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
DMAxSAx							
rw	rw	rw	rw	rw	rw	rw	rw

DMAxSAx Биты 15...0 Адрес источника DMA. Регистр адреса источника указывает на адрес исходного значения для режима одиночных пересылок или на адрес первого значения исходного блока данных для режимов пересылки блоков.

В моделях, содержащих не более 64 КБ адресуемой памяти, в регистре DMAxSAx используется одно слово. При записи в старшее слово с использованием команд, оперирующих 2-байтными операндами, это слово автоматически обнуляется.

В моделях, имеющих более 64 КБ адресуемой памяти, для хранения адреса источника используется дополнительное слово. Биты 15...4 этого слова зарезервированы и всегда читаются как 0. При операциях записи в регистр DMAxSAx с использованием команд, оперирующих 2-байтными операндами, это дополнительное слово автоматически обнуляется. При чтении дополнительного слова с использованием команд, оперирующих 2-байтными операндами, всегда возвращается нулевое значение.

DMAxDA, регистр адреса приёмника канала x DMA

15	14	13	12	11	10	9	8
Reserved							
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
Reserved				**DMAxDAx**			
r0	r0	r0	r0	rw	rw	rw	rw

15	14	13	12	11	10	9	8
DMAxDAx							
rw	rw	rw	rw	rw	rw	rw	rw

7	6	5	4	3	2	1	0
DMAxDAx							
rw	rw	rw	rw	rw	rw	rw	rw

DMAxDAx Биты 15...0 Адрес приёмника DMA. Регистр адреса приёмника указывает на итоговый адрес значения для режима одиночных пересылок или на адрес первого значения итогового блока данных для режимов пересылки блоков.

В моделях, содержащих не более 64 КБ адресуемой памяти, для регистра DMAxDAx используется одно слово. При записи в старшее слово с использованием команд, оперирующих 2-байтными операндами, это слово автоматически обнуляется.

В моделях, имеющих более 64 КБ адресуемой памяти, для хранения адреса источника используется дополнительное слово. Биты 15...4 этого слова зарезервированы и всегда читаются как 0. При операциях записи в регистр DMAxDAx с использованием команд, оперирующих 2-байтными операндами, это дополнительное слово автоматически обнуляется. При чтении дополнительного слова с использованием команд, оперирующих 2-байтными операндами, всегда возвращается нулевое значение.

DMAxSZ, регистр размера пересылки канала x DMA

15	14	13	12	11	10	9	8
DMAxSZx							
rw	rw	rw	rw	rw	rw	rw	rw
7	6	5	4	3	2	1	0
DMAxSZx							
rw	rw	rw	rw	rw	rw	rw	rw

DMAxSZx Биты 15...0 Размер пересылки DMA. Этот регистр определяет количество байтов/слов в пересылаемом блоке. Содержимое регистра DMAxSZx декрементируется после пересылки каждого слова или байта. При достижении нулевого значения в регистр DMAxSZ автоматически загружается его исходное содержимое.

00000h Пересылка запрещена
00001h Необходимо переслать один байт или слово
00002h Необходимо переслать два байта или слова
:
0FFFFh Необходимо переслать 65535 байтов или слов

DMAIV, регистр вектора прерывания контроллера DMA

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**DMAIVx**			**0**
r0	r0	r0	r0	r-(0)	r-(0)	r-(0)	r0

DMAIVx Биты 15...0 Значение вектора прерывания DMA

Содержимое DMAIV	Источник прерывания	Флаг прерывания	Приоритет прерывания
00h	Нет прерывания	—	
02h	Канал 0 DMA	DMA0IFG	Высший
04h	Канал 1 DMA	DMA1IFG	
06h	Канал 2 DMA	DMA2IFG	
08h	Зарезервировано	—	
0Ah	Зарезервировано	—	
0Ch	Зарезервировано	—	
0Fh	Зарезервировано	—	Низший

ГЛАВА 7

КОНТРОЛЛЕР ФЛЭШ-ПАМЯТИ

В этой главе описывается функционирование контроллера флэш-памяти микроконтроллеров семейства MSP430x2xx.

7.1. Введение

В микроконтроллерах семейства MSP430 флэш-память может адресоваться и записываться побитно, побайтно или пословно. Модуль флэш-памяти имеет собственный контроллер, управляющий операциями программирования и стирания памяти. Контроллер имеет четыре регистра, тактовый генератор, а также генератор напряжения, формирующий напряжения, необходимые для выполнения операций записи и стирания.

Контроллер флэш-памяти имеет следующие особенности:

- встроенный генератор напряжения программирования;
- побитовое, побайтовое или пословное программирование;
- сверхмалое потребление;
- поддерживается стирание сегментов и общее стирание;
- два режима чтения при граничных условиях (опционально, см. документацию на конкретную модель).

Блок-схема контроллера флэш-памяти приведена на **Рис. 7.1**.

Примечание. Минимальное значение V_{CC} при записи и стирании флэш-памяти
Минимальное значение напряжения питания V_{CC} во время операций записи или стирания флэш-памяти составляет 2.2 В. Если в процессе записи или стирания флэш-памяти напряжение V_{CC} снизится ниже 2.2 В, результат операции будет непредсказуем.

7.2. Сегментная организация флэш-памяти

В микроконтроллерах MSP430 вся флэш-память поделена на сегменты. Записывать в флэш-память можно отдельные биты, байты или же слова, однако наименьшей единицей флэш-памяти, которую можно стереть, является сегмент.

Кроме того, флэш-память содержит две секции — основную и информационную. Обе секции функционируют совершенно одинаково. И программный код, и данные могут располагаться в обеих секциях. Различие между этими секциями заключается в разных значениях размера сегмента и физических адресов.

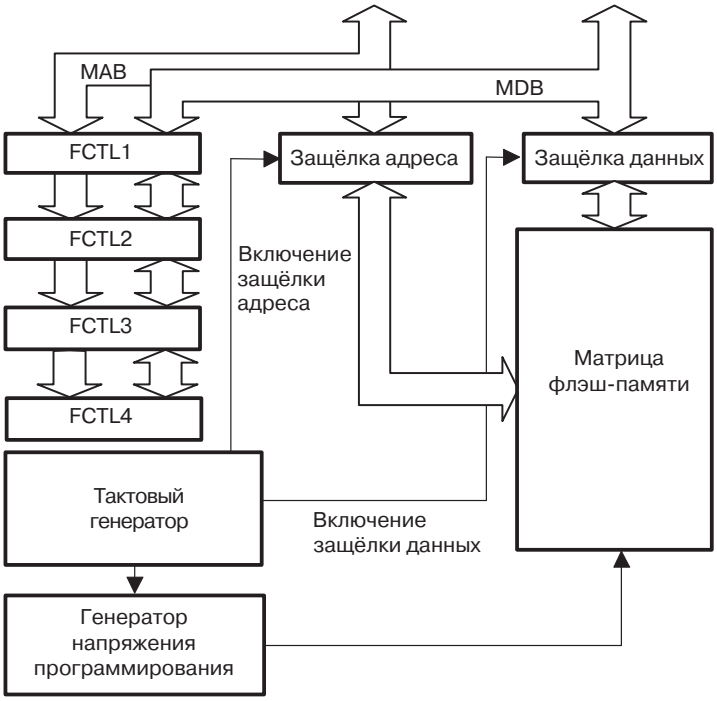


Рис. 7.1. Блок-схема контроллера флэш-памяти.

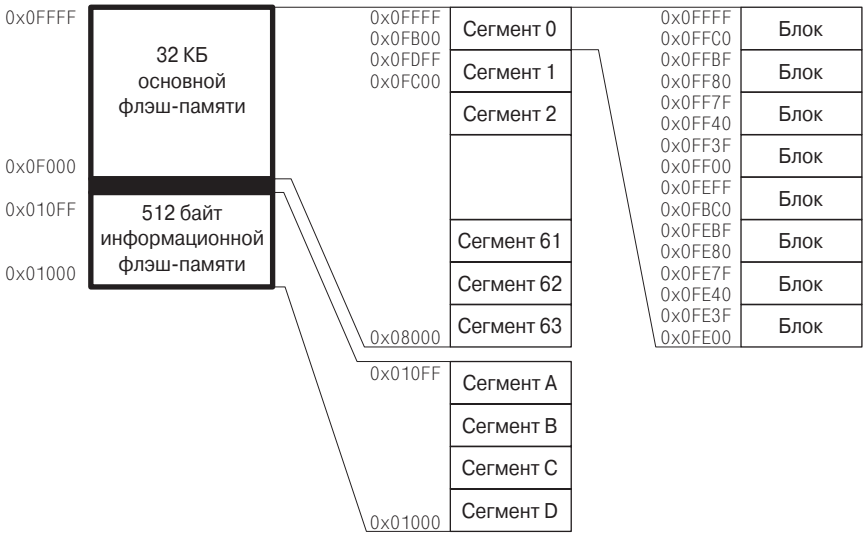


Рис. 7.2. Сегментная организация флэш-памяти (для модуля объёмом 32 КБ).

Информационная секция содержит четыре 64-байтных сегмента, в то время как основная секция содержит два или более сегментов размером 512 байт. Полная карта памяти конкретного устройства приводится в его справочной документации.

Сегменты, в свою очередь, поделены на блоки.

На **Рис. 7.2** показана сегментная организация модуля флэш-памяти объёмом 32 КБ, который имеет 8 основных и четыре информационных сегмента.

7.2.1. Сегмент А

Сегмент А информационной секции флэш-памяти может быть заблокирован независимо от остальных секций посредством бита LOCKA. При LOCKA = 1 запись или стирание сегмента А запрещены, а всё содержимое информационной секции защищено от стирания при выполнении операции общего стирания или при программировании микроконтроллера. При LOCKA = 0 запись и стирание сегмента А осуществляются наравне с любыми другими секциями флэш-памяти, а содержимое информационной секции при выполнении общего стирания или при программировании микроконтроллера стирается.

Состояние бита LOCKA изменяется записью в него 1. Запись 0 в бит LOCKA не изменяет его состояния. Это позволяет использовать без изменений существующие процедуры программирования флэш-памяти.

```
; Разблокировать сегмент А
   BIT  #LOCKA,&FCTL3            ; Проверяем LOCKA
   JZ   SEGA_UNLOCKED            ; Уже разблокировано?
   MOV  #FWKEY+LOCKA,&FCTL3      ; Нет, разблокируем сегмент
SEGA_UNLOCKED                    ; Да, продолжаем
; Сегмент А разблокирован

; Заблокировать сегмент А
   BIT  #LOCKA,&FCTL3            ; Проверяем LOCKA
   JNZ  SEGALOCKED               ; Уже заблокировано?
   MOV  #FWKEY+LOCKA,&FCTL3      ; Нет, блокируем сегмент
SEGA_LOCKED                      ; Да, продолжаем
; Сегмент А заблокирован
```

7.3. Функционирование флэш-памяти

По умолчанию флэш-память находится в режиме для чтения. В этом режиме стирание или запись флэш-памяти заблокированы, а тактовый генератор и генератор напряжения выключены — память работает подобно ПЗУ.

Флэш-память семейства MSP430 поддерживает внутрисхемное программирование (ISP) и не требует дополнительного источника напряжения. Центральный процессор может осуществлять программирование собственной флэш-памяти. Посредством битов BLKWRT, WRT, MERAS и ERASE могут быть выбраны следующие режимы записи/стирания флэш-памяти:

- запись байта/слова;
- запись блока;

- стирание сегмента;
- общее стирание (стирание всех сегментов основной секции памяти);
- полное стирание (стирание всей памяти).

Чтение или запись флэш-памяти во время её программирования или стирания запрещены. Если во время записи или стирания памяти требуется выполнение программы, то исполняемый код должен располагаться в ОЗУ. Процесс изменения флэш-памяти может быть инициирован как из флэш-памяти, так и из ОЗУ.

7.3.1. Тактовый генератор контроллера флэш-памяти

Операции записи и стирания управляются тактовым генератором контроллера, блок-схема которого приведена на **Рис. 7.3**. Рабочая частота тактового генератора f_{FTG} должна находиться в диапазоне от ~257 кГц до ~476 кГц (см. документацию на конкретный микроконтроллер).

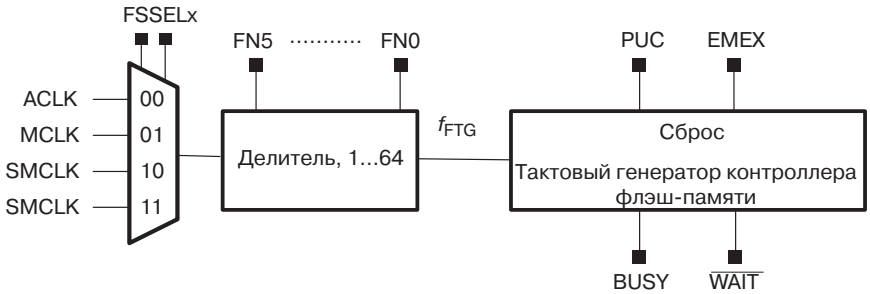


Рис. 7.3. Блок-схема тактового генератора контроллера флэш-памяти.

Выбор опорного сигнала тактового генератора контроллера флэш-памяти

Тактовый генератор контроллера флэш-памяти может тактироваться от любого из тактовых сигналов ACLK, SMCLK или MCLK. Из выбранного сигнала посредством делителя, коэффициент деления которого задаётся битами FNx, формируется тактовый сигнал контроллера f_{FTG}. Если во время операции записи или стирания частота этого сигнала будет отличаться от указанной в спецификации, то результат операции может оказаться непредсказуемым или же рабочие параметры флэш-памяти могут выйти за пределы, гарантирующие надёжную работу.

При обнаружении неисправности тактового генератора во время операции записи или стирания данная операция прерывается и устанавливается флаг FAIL. Результат операции при этом будет непредсказуемым.

Во время выполнения операции записи или стирания выбранный источник тактового сигнала нельзя выключить, переведя MSP430 в режим пониженного энергопотребления. Этот источник останется в активном состоянии до завершения операции, после чего будет выключен.

7.3.2. Стирание флэш-памяти

При стирании биты флэш-памяти устанавливаются в 1. Состояние каждого отдельного бита при программировании может быть изменено с 1 на 0, однако для изменения его состояния с 0 на 1 требуется цикл стирания. Наименьшим элементом флэш-памяти, допускающим независимое стирание, является сегмент. Предусмотрено три режима стирания, которые задаются битами ERASE и MERAS в соответствии с **Табл. 7.1**.

Таблица 7.1. **Режимы стирания**

MERAS	ERASE	Режим стирания
0	1	Стирание сегмента
1	0	Общее стирание (стирание всех сегментов основной секции флэш-памяти)
1	1	LOCKA = 0 — стирание основной и информационной секций флэш-памяти LOCKA = 1 — стирание только основной секции флэш-памяти

Любой цикл стирания инициируется фиктивной записью по адресу, расположенному в пределах стираемого участка памяти. Фиктивная запись запускает тактовый генератор контроллера и процесс стирания. Временная диаграмма цикла стирания приведена на **Рис. 7.4**. Бит BUSY устанавливается сразу же после выполнения фиктивной записи и остаётся в этом состоянии до окончания цикла стирания. Биты BUSY, MERAS и ERASE автоматически сбрасываются после завершения цикла стирания. Значения временных параметров цикла стирания не зависят от объёма флэш-памяти, имеющейся в устройстве. Длительность однотипных циклов стирания одинакова для всех микроконтроллеров семейства.

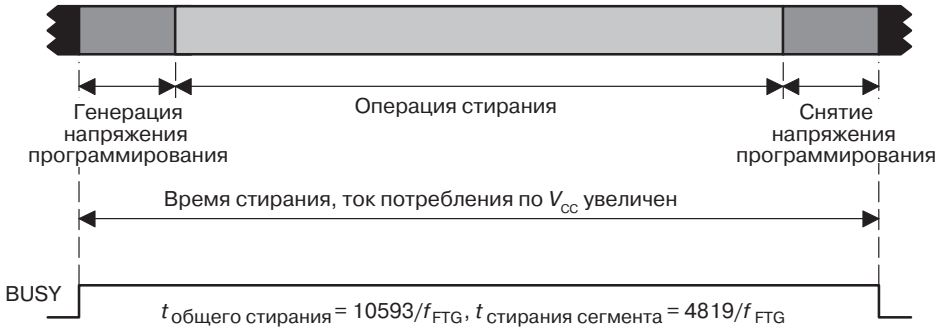


Рис. 7.4. Временная диаграмма цикла стирания.

Фиктивная запись по адресу, находящемуся вне стираемого участка памяти, не инициирует цикл стирания, не изменяет состояние флэш-памяти и никак не влияет на флаги. Такая операция фиктивной записи просто игнорируется.

Запуск процедуры стирания из программы, расположенной в флэш-памяти

Любой цикл стирания может быть инициирован из программы, расположенной как в флэш-памяти, так и в ОЗУ. При запуске операции стирания сегмента из

программы, расположенной в флэш-памяти, все временные параметры определяются контроллером флэш-памяти, а работа ЦПУ приостанавливается до окончания операции. После завершения цикла стирания центральный процессор возобновляет выполнение программы, начиная с команды, следующей за командой фиктивной записи.

При инициации цикла стирания из программы, расположенной в флэш-памяти, возможно стирание фрагмента кода программы, который должен был выполняться после стирания. При возникновении такой ситуации поведение ЦПУ после завершения операции стирания будет непредсказуемым.

Последовательность операций, требуемая для запуска цикла стирания из программы, находящейся в флэш-памяти, показана на **Рис. 7.5**.

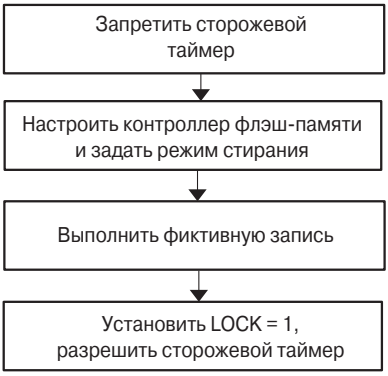


Рис. 7.5. Запуск цикла стирания из флэш-памяти.

```
; Стирание сегмента из флэш-памяти. 514 кГц < SMCLK < 952 кГц
; Предполагается, что ACCVIE = NMIIE = OFIE = 0.
  MOV #WDTPW+WDTHOLD,&WDTCTL      ; Запрещаем WDT
  MOV #FWKEY+FSSEL1+FN0,&FCTL2    ; SMCLK/2
  MOV #FWKEY,&FCTL3               ; Сбрасываем LOCK
  MOV #FWKEY+ERASE,&FCTL1         ; Разрешаем стирание сегмента
  CLR &0FC10h                     ; Фиктивная запись, стираем сегмент 1
  MOV #FWKEY+LOCK,&FCTL3          ; Выполнено, устанавливаем LOCK
  ...                             ; Повторно разрешаем WDT?
```

Запуск процедуры стирания из программы, расположенной в ОЗУ

Любой цикл стирания может быть инициирован из программы, расположенной в ОЗУ. В этом случае ЦПУ не останавливается, а продолжает исполнять код программы из ОЗУ. Обращение ЦПУ по любому адресу флэш-памяти допускается только после завершения цикла стирания, что определяется по состоянию бита BUSY. Попытка обращения к флэш-памяти при BUSY = 1 приведёт к нарушению доступа с последующей установкой флага ACCVIFG, а результат стирания окажется неопределённым.

Последовательность операций, требуемая для запуска цикла стирания из программы, находящейся в ОЗУ, показана на **Рис. 7.6**.

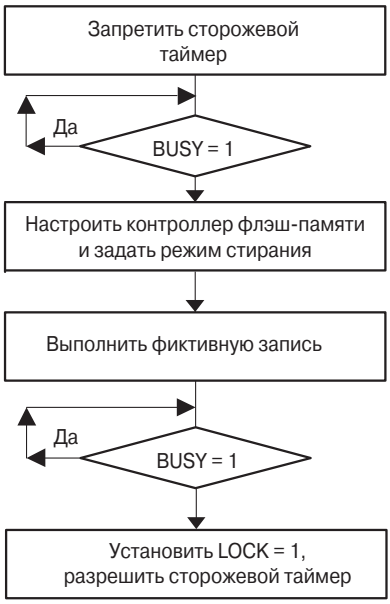


Рис. 7.6. Запуск цикла стирания из ОЗУ.

```
; Стирание сегмента из ОЗУ. 514 кГц < SMCLK < 952 кГц
; Предполагается, что ACCVIE = NMIIE = OFIE = 0.
     MOV #WDTPW+WDTHOLD,&WDTCTL     ; Запрещаем WDT
  L1 BIT #BUSY,&FCTL3               ; Проверяем BUSY
     JNZ L1                         ; Ждём освобождения контроллера
     MOV #FWKEY+FSSEL1+FN0,&FCTL2   ; SMCLK/2
     MOV #FWKEY,&FCTL3              ; Сбрасываем LOCK
     MOV #FWKEY+ERASE,&FCTL1        ; Разрешаем стирание
     CLR &0FC10h                    ; Фиктивная запись, стираем сегмент 1
  L2 BIT #BUSY,&FCTL3               ; Проверяем BUSY
     JNZ L2                         ; Ждём завершения операции
     MOV #FWKEY+LOCK,&FCTL3         ; Выполнено, устанавливаем LOCK
     ...                            ; Повторно разрешаем WDT?
```

7.3.3. Запись в флэш-память

Возможные режимы записи задаются битами WRT и BLKWRT в соответствии с **Табл. 7.2**.

Таблица 7.2. **Режимы записи**

BLKWRT	WRT	Режим записи
0	1	Запись байта/слова
1	1	Блочная запись

В обоих режимах запись производится последовательным выполнением отдельных команд записи, однако режим блочной записи позволяет выполнять данную операцию почти в два раза быстрее, нежели режим записи отдельных значений, поскольку в этом режиме генератор напряжения остаётся включенным до завершения записи всего блока. Для изменения содержимого флэш-памяти как в режиме записи слова/байта, так и в режиме блочной записи можно использовать любые команды, модифицирующие операнд-приёмник. Причём после каждого стирания не следует писать в одно и то же слово флэш-памяти (младший + старший байты) больше двух раз — это может вызвать повреждение памяти.

Бит BUSY устанавливается в начале операции записи и сбрасывается после её завершения. Если операция записи инициирована программой, находящейся в ОЗУ, то процессор не должен обращаться к флэш-памяти до сброса бита BUSY. Попытка обращения к флэш-памяти при BUSY = 1 приведёт к нарушению доступа с последующей установкой флага ACCVIFG, а результат записи окажется неопределённым.

Запись байта/слова

Операция записи отдельного байта/слова может быть инициирована из программы, расположенной как в флэш-памяти, так и в ОЗУ. При запуске операции записи из программы, расположенной в флэш-памяти, все временные параметры определяются контроллером флэш-памяти, а работа ЦПУ приостанавливается до окончания операции. После завершения цикла стирания центральный процессор возобновляет выполнение программы, начиная с команды, следующей за командой записи. Временная диаграмма операции записи байта/слова приведена на **Рис. 7.7**.

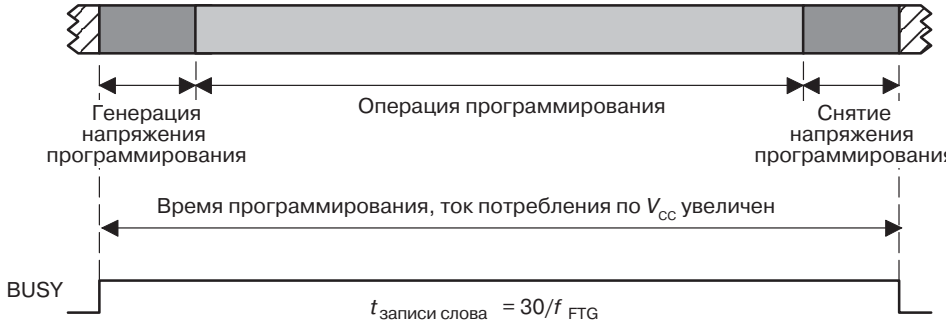


Рис. 7.7. Временная диаграмма операции записи байта/слова.

При запуске операции записи из программы, расположенной в ОЗУ, процессор не останавливается, а продолжает исполнять код программы. Обращение ЦПУ к флэш-памяти допускается только после завершения цикла записи, что определяется по состоянию бита BUSY. Попытка обращения к флэш-памяти при BUSY = 1 приведёт к нарушению доступа с последующей установкой флага ACCVIFG, а результат записи окажется неопределённым.

В режиме записи байта/слова формируемое внутренним генератором напряжение подаётся на весь 64-байтный блок в течение 27 тактов f_{FTG} из 30 при каждой операции записи. Длительности этих интервалов, во время которых при каждой операции записи на блок памяти подаётся напряжение программирования, суммируются. Для любого блока это суммарное время не должно превысить значения t_{CPT}, указанного в спецификации на конкретный микроконтроллер. Если же общее время записи достигает указанного значения, то перед выполнением последующих операций записи по любым адресам в пределах этого блока его необходимо стереть.

Запуск процедуры записи байта/слова из программы, расположенной в флэш-памяти

Последовательность операций, требуемая для запуска процедуры записи байта/слова из программы, находящейся в флэш-памяти, приведёна на **Рис. 7.8**.

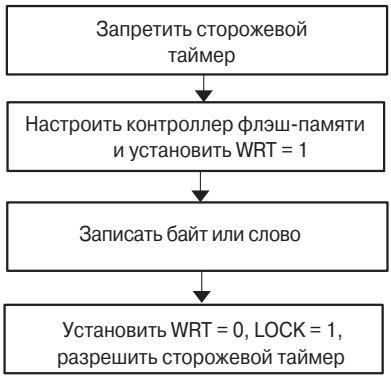


Рис. 7.8. Запись байта/слова, инициируемая из флэш-памяти.

```
; Запись байта/слова из флэш-памяти. 514 кГц < SMCLK < 952 кГц
; Предполагается, что блок 0FF1Eh уже стёрт
; Предполагается, что ACCVIE = NMIIE = OFIE = 0.
  MOV #WDTPW+WDTHOLD,&WDTCTL     ; Запрещаем WDT
  MOV #FWKEY+FSSEL1+FN0,&FCTL2   ; SMCLK/2
  MOV #FWKEY,&FCTL3              ; Сбрасываем LOCK
  MOV #FWKEY+WRT,&FCTL1          ; Разрешаем запись
  MOV #0123h,&0FF1Eh             ; 0123h -> 0FF1Eh
  MOV #FWKEY,&FCTL1              ; Готово. Сбрасываем WRT
  MOV #FWKEY+LOCK,&FCTL3         ; Устанавливаем LOCK
  ...                            ; Повторно разрешаем WDT?
```

Запуск процедуры записи байта/слова из программы, расположенной в ОЗУ

Последовательность операций, треьуемая для запуска процедуры записи байта/слова из программы, находящейся в ОЗУ, приведёна на **Рис. 7.9**.

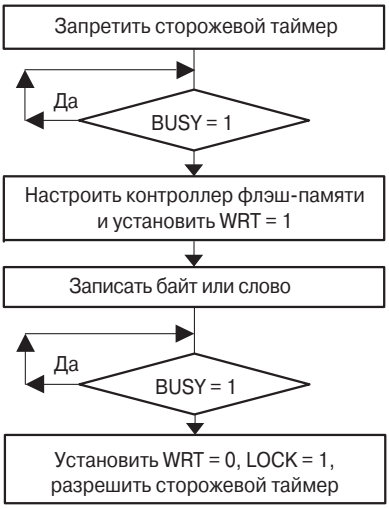


Рис. 7.9. Запись байта/слова, инициируемая из ОЗУ.

```
; Запись байта/слова из ОЗУ. 514 кГц < SMCLK < 952 кГц
; Предполагается, что блок 0FF1Eh уже стёрт
; Предполагается, что ACCVIE = NMIIE = OFIE = 0.
   MOV #WDTPW+WDTHOLD,&WDTCTL     ; Запрещаем WDT
L1 BIT #BUSY,&FCTL3               ; Проверяем BUSY
   JNZ L1                         ; Ждём освобождения контроллера
   MOV #FWKEY+FSSEL1+FN0,&FCTL2   ; SMCLK/2
   MOV #FWKEY,&FCTL3              ; Сбрасываем LOCK
   MOV #FWKEY+WRT,&FCTL1          ; Разрешаем запись
   MOV #0123h,&0FF1Eh             ; 0123h -> 0FF1Eh
L2 BIT #BUSY,&FCTL3               ; Проверяем BUSY
   JNZ L2                         ; Ждём освобождения контроллера
   MOV #FWKEY,&FCTL1              ; Сбрасываем WRT
   MOV #FWKEY+LOCK,&FCTL3         ; Устанавливаем LOCK
   ...                            ; Повторно разрешаем WDT?
```

Блочная запись

Режим блочной записи может использоваться для ускорения процесса записи в флэш-память при необходимости программирования большого числа подряд идущих байтов или слов. Напряжение программирования не снимается с флэш-памяти до окончания записи 64-байтного блока. Суммарное время программирования любого блока не должно превышать значения t_{CPT} для данного микроконтроллера.

Блочная запись не может быть инициирована из флэш-памяти — только из ОЗУ. Бит BUSY сбрасывается только после завершения записи всего блока. Между записью отдельных байтов или слов блока необходимо проверять бит WAIT — запись очередного байта или слова разрешается только при WAIT = 1. При последовательной записи нескольких блоков бит BLKWRT необходимо сбрасывать после окончания записи каждого из них. Повторная установка бита BLKWRT,

разрешающая запись следующего блока, может быть произведена через определённое время, требуемое для восстановления флэш-памяти (t_{end}). Бит BUSY сбрасывается после окончания записи очередного блока, информируя о возможности записи следующего блока. Временная диаграмма операции блочной записи показана на **Рис. 7.10**.

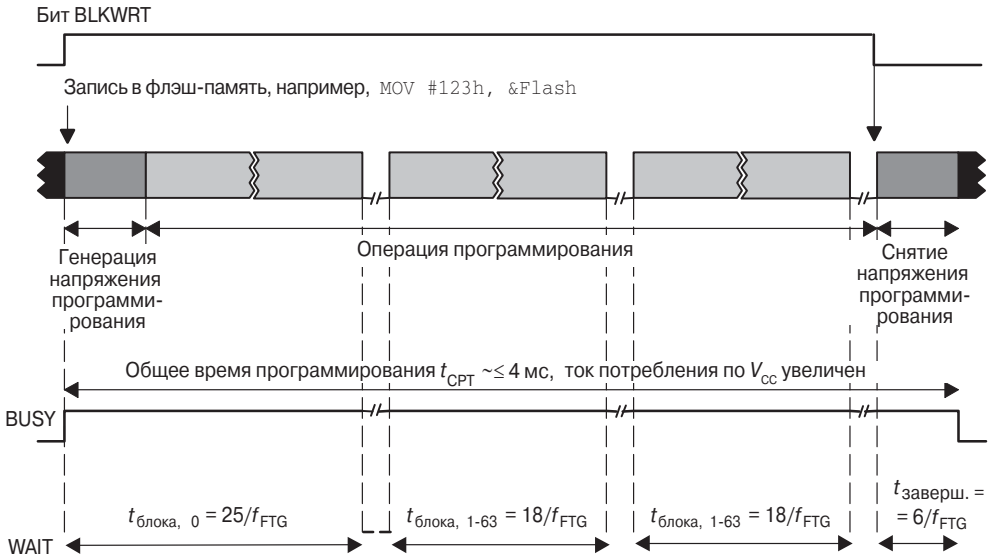


Рис. 7.10. Временная диаграмма операции блочной записи.

Использование режима блочной записи

Последовательность операций, требуемая для выполнения записи в блочном режиме, приведена на **Рис. 7.11**.

```
; Записываем блок, начиная с адреса 0F000h.
; Программа должна исполняться из ОЗУ, предполагается,
; что флэш-память уже стёрта.
; 514 кГц < SMCLK < 952 кГц
; Предполагается, что ACCVIE = NMIIE = OFIE = 0.
   MOV   #32,R5                    ; Используем как счётчик записи
   MOV   #0F000h,R6                ; Загружаем указатель
   MOV   #WDTPW+WDTHOLD,&WDTCTL    ; Запрещаем WDT
L1 BIT   #BUSY,&FCTL3              ; Проверяем BUSY
   JNZ   L1                        ; Ждём готовности контроллера
   MOV   #FWKEY+FSSEL1+FN0,&FCTL2  ; SMCLK/2
   MOV   #FWKEY,&FCTL3             ; Сбрасываем LOCK
   MOV   #FWKEY+BLKWRT+WRT,&FCTL1  ; Разрешаем блочную запись
L2 MOV   Write_Value,0(R6)         ; Пишем по адресу
L3 BIT   #WAIT,&FCTL3              ; Проверяем WAIT
   JZ    L3                        ; Будем ждать, пока WAIT=0
   INCD  R6                        ; Указываем на следующее слово
   DEC   R5                        ; Декрементируем счётчик записи
```

```
   JNZ   L2                        ; Конец блока?
   MOV   #FWKEY,&FCTL1             ; Сбрасываем WRT,BLKWRT
L4 BIT   #BUSY,&FCTL3              ; Проверяем BUSY
   JNZ   L4                        ; Ждем готовности контроллера
   MOV   #FWKEY+LOCK,&FCTL3        ; Устанавливаем LOCK
   ...                             ; Повторно разрешаем WDT
                                   ; при необходимости
```

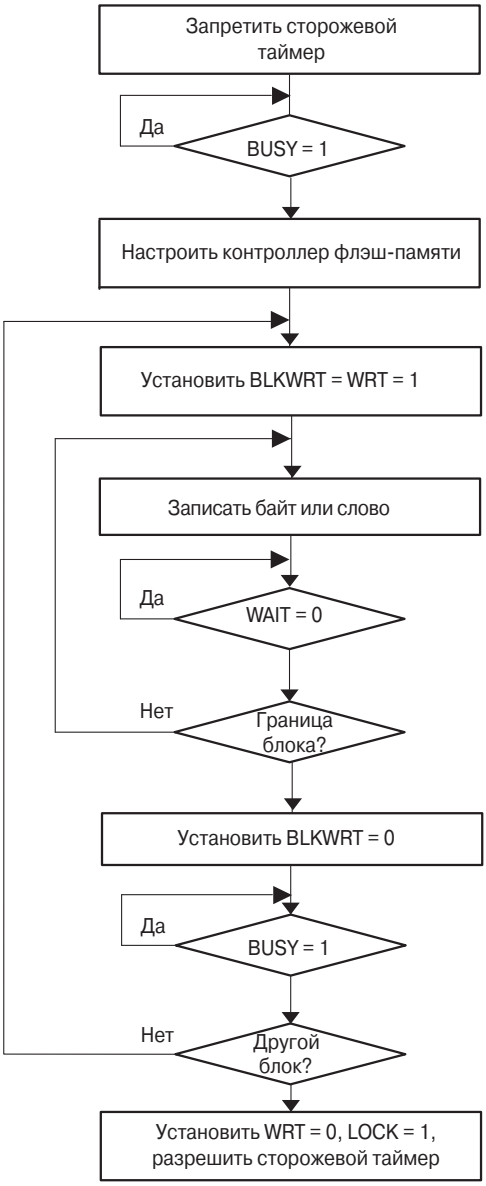


Рис. 7.11. Использование режима блочной записи.

7.3.4. Обращение к флэш-памяти во время записи или стирания

В случае инициации операции записи или стирания из ОЗУ центральный процессор не может ни читать из флэш-памяти, ни писать в неё до тех пор, пока бит BUSY = 1. В противном случае произойдёт нарушение доступа с последующей установкой флага ACCVIFG, а результат операции окажется непредсказуемым. Аналогично, флаг ACCVIFG устанавливается в случае попытки записи в флэш-память при сброшенном бите WRT. Содержимое флэш-памяти при этом не изменяется.

В случае если операция записи слова/байта или операция стирания инициирована из флэш-памяти, ЦПУ при выборке очередной инструкции получает от контроллера флэш-памяти значение 03FFFh. Это код команды `JMP PC`. В результате ЦПУ зацикливается до окончания операции с флэш-памятью. После завершения операции и сброса бита BUSY контроллер флэш-памяти возвращает ЦПУ корректное значение кода команды и выполнение программы возобновляется.

Условия доступа к флэш-памяти при BUSY = 1 перечислены в **Табл. 7.3**.

Таблица 7.3. **Доступ к флэш-памяти при BUSY = 1**

Операция с флэш-памятью	Обращение к флэш-памяти	WAIT	Результат
Стирание или запись байта/слова	Чтение	0	ACCVIFG = 0. Считывается значение 03FFFh.
	Запись	0	ACCVIFG = 1. Команда записи игнорируется.
	Выборка команды	0	ACCVIFG = 0. ЦПУ считывает код 03FFFh, соответствующий команде JMP PC.
Блочная запись	Любое	0	ACCVIFG = 1, LOCK = 1
	Чтение	1	ACCVIFG = 0, считывается значение 03FFFh
	Запись	1	ACCVIFG = 0, команда записи игнорируется
	Выборка команды	1	ACCVIFG = 1, LOCK = 1

Прерывания автоматически запрещаются на время манипуляций с флэш-памятью, если биты EEI = 0 и EEIEX = 0, а также в моделях MSP430x20xx, в которых эти биты отсутствуют. После завершения операции с флэш-памятью прерывания автоматически разрешаются. При возникновении во время работы с флэш-памятью какого-либо прерывания будет установлен соответствующий флаг, а после повторного разрешения прерываний будет сгенерирован запрос прерывания.

Если биты EEIEX = 1 и GIE = 1, то при возникновении прерывания любая операция с флэш-памятью будет немедленно прервана с последующей установкой флага FAIL. При EEI = 1, GIE = 1 и EEIEX = 0 операция стирания сегмента будет прерываться каждые 32 такта f_{FTG} для обработки отложенных прерываний. После обслуживания прерывания стирание сегмента продолжается в течение следующих 32 тактов или до завершения операции. Во время обслуживания прерывания бит BUSY остаётся в установленном состоянии, однако ЦПУ может обращаться к флэш-памяти, не вызывая нарушения доступа. Вложенные прерывания и использование команды `RETI` внутри процедур обработки прерывания не поддерживаются.

Сторожевой таймер (при использовании его в сторожевом режиме) должен быть запрещён перед запуском цикла стирания флэш-памяти. В случае сброса процесс стирания окажется прерванным, и результат операции будет непредсказуемым. После завершения цикла стирания работа сторожевого таймера может быть разрешена.

7.3.5. Останов циклов записи или стирания

Любая операция записи или стирания может быть досрочно прекращена установкой бита EMEX. Установка бита EMEX немедленно прерывает выполнение текущей операции и останавливает контроллер флэш-памяти. Любые действия с флэш-памятью прекращаются, она возвращается в режим чтения, а все биты регистра FCTL1 сбрасываются. Результат прерванной операции будет неопределённым.

7.3.6. Режим чтения при граничных условиях

Режим чтения при граничных условиях может использоваться для проверки целостности содержимого флэш-памяти. Такая возможность реализована в отдельных моделях семейства MSP430x2xx; чтобы узнать о наличии этой функции в конкретном микроконтроллере обратитесь к документации. В режиме чтения при граничных условиях можно обнаружить «слабо» запрограммированные биты флэш-памяти. Такие биты могут появиться в результате неправильной установки частоты f_{FTG} или же в результате снижения напряжения питания V_{CC} ниже минимально допустимого значения при выполнении операций стирания/программирования. Один из методов обнаружения таких ячеек памяти заключается в периодическом вычислении контрольной суммы некоторого фрагмента флэш-памяти (к примеру, сегмента) и повторении этой операции с использованием режима чтения при граничных условиях. Различие между полученными значениями может служить признаком того, что в проверяемом фрагменте флэш-памяти имеются «слабо» запрограммированные биты. Такой сегмент флэш-памяти можно восстановить. Для этого следует, выключив режим чтения при граничных условиях, скопировать содержимое сегмента в ОЗУ, затем стереть указанный сегмент и повторно записать в него данные из ОЗУ.

Процедура, выполняющая проверку целостности содержимого флэш-памяти, должна выполняться из ОЗУ. При выполнении кода из флэш-памяти режим чтения при граничных условиях автоматически выключается. Конкретные режимы чтения при граничных условиях задаются битами MRG0 и MRG1 регистра FCTL4. При установленном бите MRG1 выполняется обнаружение неустойчиво запрограммированных ячеек флэш-памяти, содержащих 1 (стёртые биты). При установленном бите MRG0 выполняется обнаружение неустойчиво запрограммированных ячеек флэш-памяти, содержащих 0 (запрограммированные биты). Одновременно может быть установлен только один из этих битов. Соответственно, полная проверка целостности содержимого флэш-памяти потребует двух проходов. При включенном режиме чтения при граничных условиях частота обращений к флэш-памяти (MCLK) должна быть ограничена значением 1 МГц (см. документацию на конкретный микроконтроллер).

7.3.7. Конфигурирование контроллера флэш-памяти и организация доступа к нему

Регистры FCTLx — это защищённые паролем 16-битные регистры, доступные для чтения и записи. Любые обращения к этим регистрам должны производиться с использованием команд, оперирующих 2-байтными операндами, причём при операциях записи в старшем байте записываемого значения должно содержаться число 0A5h. Запись в любой из регистров FCTLx значения, старший байт которого не равен 0A5h, вызовет нарушение ключа защиты с последующей установкой флага KEYV и формированием сигнала сброса системы PUC. При чтении любого из регистров FCTLx в старшем байте возвращается значение 096h.

Запись в регистр FCTL1 во время операции стирания или записи байта/слова вызывает нарушение доступа к памяти и установку флага ACCVIFG. В режиме блочной записи запись в регистр FCTL1 допускается, но только при WAIT = 1. Запись в регистр FCTL1 в режиме блочной записи при WAIT = 0 вызывает нарушение доступа к памяти и установку флага ACCVIFG.

Попытка записи в регистр FCTL2 при BUSY = 1 вызывает нарушение доступа к флэш-памяти.

Чтение регистров FCTLx при BUSY = 1 разрешается. Операция чтения регистров контроллера флэш-памяти не вызовет нарушения доступа.

7.3.8. Прерывания контроллера флэш-памяти

Контроллер флэш-памяти содержит два источника прерываний — KEYV и ACCVIFG. Флаг ACCVIFG устанавливается при нарушении доступа к флэш-памяти. Если бит ACCVIE повторно устанавливается после операции записи или стирания флэш-памяти, то установленный флаг ACCVIFG вызовет генерацию запроса прерывания. Флаг ACCVIFG является одним из источников вектора немаскируемого прерывания, поэтому для генерации запроса прерывания по флагу ACCVIFG устанавливать бит GIE не требуется. Кроме того, этот флаг можно проверять программно, чтобы отслеживать возникновение ситуации нарушения доступа. Сброс флага ACCVIFG должен осуществляться программно.

Флаг нарушения ключа защиты KEYV устанавливается при попытке записи в любой из регистров контроллера флэш-памяти с использованием некорректного ключа. При этом генерируется сигнал PUC, немедленно сбрасывающий микроконтроллер.

7.3.9. Программирование флэш-памяти

Существует три способа программирования микроконтроллеров семейства MSP430. Все эти способы позволяют программировать устройство в составе системы:

- программирование по интерфейсу JTAG;
- программирование с использованием встроенного загрузчика;
- программирование с использованием заказных решений.

Программирование флэш-памяти по интерфейсу JTAG

Микроконтроллеры семейства MSP430 допускают программирование через порт JTAG. Для интерфейса JTAG требуется четыре (в 20- и 28-выводных устройствах — пять) сигнальных линий, линия земли и, опционально, линия напряжения питания V_{CC} и линия сброса $\overline{RST}$/NMI.

Порт JTAG защищён плавкой перемычкой, при перегорании которой он полностью отключается. Перегорание перемычки необратимо — последующий доступ к устройству по интерфейсу JTAG будет невозможен. Для получения более подробной информации обратитесь к руководству по применению «Programming a Flash-Based MSP430 Using the JTAG Interface», находящемуся на сайте www.msp430.com.

Программирование флэш-памяти с использованием загрузчика (BSL)

В большинстве флэш-микроконтроллеров семейства MSP430 имеется встроенный начальный загрузчик (BootStrap Loader — BSL). Для получения более подробной информации о реализации загрузчика обратитесь к документации на конкретный микроконтроллер. Загрузчик позволяет осуществлять чтение/запись флэш-памяти или ОЗУ микроконтроллера, используя последовательный интерфейс UART. Доступ к флэш-памяти посредством загрузчика защищён 256-байтным паролем, определяемым пользователем. Более подробно использование загрузчика описано в руководстве по применению «Features of the MSP430 Bootstrap Loader», находящемуся на сайте www.ti.com/msp430.

Программирование флэш-памяти с использованием заказных решений

Способность ЦПУ MSP430 осуществлять запись в собственную флэш-память позволяет создавать специализированные средства для программирования микроконтроллеров как в автономном режиме, так и в составе системы (**Рис. 7.12**). Пользователь может использовать любой доступный интерфейс (UART, SPI и т.д.) для передачи данных в устройство. Пользовательская программа может принимать эти данные и осуществлять программирование флэш-памяти. Поскольку подобные средства разрабатываются самим пользователем, они могут наиболее полно удовлетворять требованиям конкретных приложений в части программирования, стирания и обновления флэш-памяти.

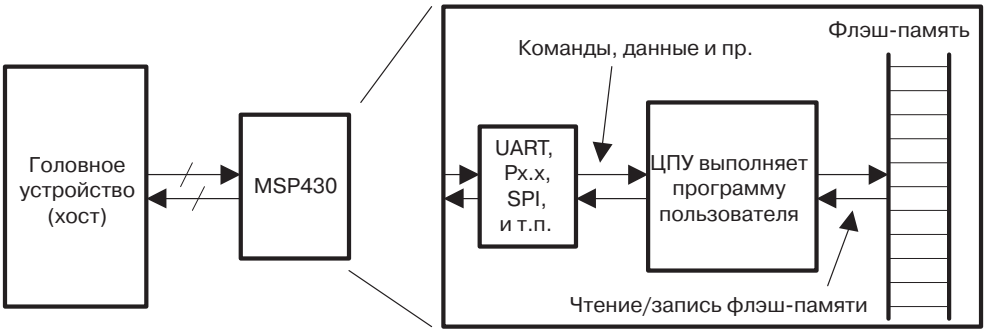


Рис. 7.12. Средство для программирования, разработанное пользователем.

7.4. Регистры контроллера флэш-памяти

Список регистров контроллера флэш-памяти приведён в **Табл. 7.4**.

Таблица 7.4. **Регистры контроллера флэш-памяти**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 1 контроллера флэш-памяти	FCTL1	Чтение/запись	0128h	09600h после PUC
Регистр управления 2 контроллера флэш-памяти	FCTL2	Чтение/запись	012Ah	09642h после PUC
Регистр управления 3 контроллера флэш-памяти	FCTL3	Чтение/запись	012Ch	09658h после PUC*
Регистр управления 4 контроллера флэш-памяти**	FCTL4	Чтение/запись	01BEh	00000h после PUC
Регистр разрешения прерываний 1	IE1	Чтение/запись	0000h	Сбрасывается после PUC
Регистр флагов прерываний 1	IFG1	Чтение/запись	0002h	Сбрасывается после PUC

* Бит KEYV сбрасывается после POR.

** Имеется не во всех моделях семейства MSP430x2xx. См. документацию на конкретный микроконтроллер.

FCTL1, регистр управления 1 контроллера флэш-памяти

15	14	13	12	11	10	9	8
FRKEY, читается как 096h FWKEY, должен записываться как 0A5h							
7	6	5	4	3	2	1	0
BLKWRT	**WRT**	**Reserved**	**EEIEX***	**EEI***	**MERAS**	**ERASE**	**Reserved**
rw–0	rw–0	r0	rw–0	rw–0	rw–0	rw–0	r–0

*Отсутствуют в микроконтроллерах MSP430x20xx.

FRKEY/FWKEY Биты 15...8 Ключ защиты FCTLx. Всегда читается как 096h. При записи должен быть равен 0A5h, в противном случае будет генерироваться сигнал PUC

BLKWRT Бит 7 Режим блочной записи. Для включения режима блочной записи также должен быть установлен бит WRT. Бит BLKWRT автоматически сбрасывается при установке бита EMEX.
- 0 Режим блочной записи выключен
- 1 Режим блочной записи включен

WRT Бит 6 Режим записи. Этот бит используется для включения любого режима записи. Бит WRT автоматически сбрасывается при установке бита EMEX.
- 0 Режим записи выключен
- 1 Режим записи включен

Reserved Бит 5 Зарезервирован. Всегда читается как 0.

EEIEX Бит 4 Аварийное завершение работы контроллера при прерывании. Установка этого бита разрешает аварийное завершение любых операций с флэш-памятью при возникновении какого-либо прерывания при условии, что бит GIE = 1. Бит EEIEX автоматически сбрасывается при установке бита EMEX.
0 Аварийное завершение при прерывании запрещено
1 Аварийное завершение при прерывании разрешено

EEI Бит 3 Разрешение прерываний во время стирания. Установка этого бита разрешает приостанавливать выполнение операции стирания сегмента для обслуживания запросов прерываний. После обработки очередного прерывания процесс стирания возобновляется.
0 Прерывания во время стирания сегмента запрещены
1 Прерывания во время стирания сегмента разрешены

MERAS Бит 2
ERASE Бит 1
Выбор режима стирания. Эти биты используются для выбора конкретного режима стирания. Биты MERAS и ERASE автоматически сбрасываются при установке бита EMEX.

MERAS	ERASE	Режим стирания
0	0	Режим стирания выключен
0	1	Стирание отдельного сегмента
1	0	Стирание всех сегментов основной секции флэш-памяти
1	1	LOCKA = 0 — стирание основной и информационной секций флэш-памяти LOCKA = 1 — стирание только основной секции флэш-памяти

Reserved Бит 0 Зарезервирован. Всегда читается как 0.

FCTL2, регистр управления 2 контроллера флэш-памяти

15	14	13	12	11	10	9	8
FRKEY, читается как 096h **FWKEY, должен записываться как 0A5h**							

7	6	5	4	3	2	1	0
FSSELx		**FNx**					
rw–0	rw–1	rw–0	rw–0	rw–0	rw–0	rw–1	rw–0

*Отсутствуют в микроконтроллерах MSP430x20xx.

FRKEY/FWKEY Биты 15...8 Ключ защиты FCTLx. Всегда читается как 096h. При записи должен быть равен 0A5h, в противном случае будет генерироваться сигнал PUC.

FSSELx Биты 7...6 Выбор источника тактового сигнала контроллера флэш-памяти.
00 ACLK
01 MCLK
10 SMCLK
11 SMCLK

FNx Биты 5...0 Делитель тактового сигнала контроллера флэш-памяти. Эти шесть битов определяют коэффициент деления опорного тактового сигнала. Коэффициент деления равен FNx + 1. К примеру, при FNx = 0 коэффициент деления равен 1. При FNx = 03Fh коэффициент деления равен 64.

FCTL3, регистр управления 3 контроллера флэш-памяти

15	14	13	12	11	10	9	8
FRKEY, читается как 096h **FWKEY, должен записываться как 0A5h**							
7	6	5	4	3	2	1	0
FAIL	**LOCKA**	**EMEX**	**LOCK**	**WAIT**	**ACCVIFG**	**KEYV**	**BUSY**
r(w)–0	r(w)–1	rw–0	rw–1	r–1	rw–0	rw–(0)	r(w)–0

FRKEY/FWKEY Биты 15...8 Ключ защиты FCTLx. Всегда читается как 096h. При записи должен быть равен 0A5h, в противном случае будет генерироваться сигнал PUC.

FAIL Бит 7 Сбой при выполнении операции. Этот бит устанавливается при обнаружении неисправности источника тактового сигнала f_{FTG} или при аварийном завершении операции с флэш-памятью в результате прерывания, когда EEIEX = 1. Бит FAIL должен сбрасываться программно.
- 0 Сбоя не было
- 1 Сбой был

LOCKA Бит 6 Блокирование сегмента A и информационной секции памяти. Запись 1 в этот бит изменяет его состояние. При записи 0 состояние бита не изменяется.
- 0 Сегмент A разблокирован; вся информационная память стирается при выполнении общего стирания.
- 1 Сегмент A заблокирован; информационная память защищена от стирания при выполнении общего стирания.

EMEX Бит 5 Экстренное завершение операции с флэш-памятью.
- 0 Не использовать экстренное завершение операции
- 1 Экстренно завершить операцию

LOCK Бит 4 Блокирование флэш-памяти. Этот бит разблокирует флэш-память для выполнения операций записи или стирания. Бит LOCK может быть установлен в любой момент времени при выполнении операции записи байта/слова или стирания, при этом выполняемая операция будет нормально завершена. В режиме блочной записи если бит LOCK устанавливается при BLKWRT = WAIT = 1, то биты BLKWRT и WAIT сбрасываются и производится нормальный выход из режима программирования.
- 0 Разблокировано
- 1 Заблокировано

WAIT Бит 3 Флаг ожидания. Этот бит показывает, что в данный момент производится запись в флэш-память.
- 0 Флэш-память не готова к записи следующего байта/слова
- 1 Флэш-память готова к записи следующего байта/слова

ACCVIFG Бит 2 Флаг прерывания при нарушении доступа.
- 0 Нет прерывания
- 1 Есть прерывание

KEYV Бит 1 Флаг нарушения ключа защиты. Этот бит показывает, что была произведена запись в какой-либо из регистров контроллера флэш-памяти с некорректным значением ключа защиты. При установке этого бита генерируется сигнал сброса PUC. Бит KEYV должен сбрасываться программно.
- 0 Был записан правильный ключ защиты FCTLx
- 1 Был записан неверный ключ защиты FCTLx

BUSY	Бит 0	Флаг занятости. Этот бит показывает состояние тактового генератора контроллера флэш-памяти. 0 Не занят 1 Занят

FCTL4, регистр управления 4 контроллера флэш-памяти

15	14	13	12	11	10	9	8
FRKEY, читается как 096h FWKEY, должен записываться как 0A5h							
7	6	5	4	3	2	1	0
Reserved		**MRG1**	**MRG0**	**Reserved**			
r–0	r–0	rw–0	rw–0	r–0	r–0	r–0	r–0

FRKEY/ FWKEY	Биты 15...8	Ключ защиты FCTLx. Всегда читается как 096h. При записи должен быть равен 0A5h, в противном случае будет генерироваться сигнал PUC.
Reserved	Биты 7...6	Зарезервированы. Всегда читаются как 0.
MRG1	Бит 5	Режим чтения 1 при граничных условиях. Этот бит включает режим чтения при граничных условиях единичных битов. Бит MRG1 сбрасывается при выполнении программы из флэш-памяти. Если оба бита MRG1 и MRG0 установлены, то бит MRG0 игнорируется. 0 Режим чтения 1 при граничных условиях выключен 1 Режим чтения 1 при граничных условиях включен
MRG0	Бит 4	Режим чтения 0 при граничных условиях. Этот бит включает режим чтения при граничных условиях нулевых битов. Бит MRG0 сбрасывается при выполнении программы из флэш-памяти. Если оба бита MRG1 и MRG0 установлены, то бит MRG0 игнорируется. 0 Режим чтения 0 при граничных условиях выключен 1 Режим чтения 0 при граничных условиях включен
Reserved	Биты 3...0	Зарезервированы. Всегда читаются как 0.

IE1, регистр разрешения прерываний 1

7	6	5	4	3	2	1	0
		ACCVIE					
		rw–0					

	Биты 7...6	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
ACCVIE	Бит 5	Разрешение прерывания при нарушении доступа к флэш-памяти. Этот бит разрешает прерывание ACCVIFG. Поскольку прочие биты регистра IE1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Прерывание запрещено 1 Прерывание разрешено
	Биты 4...0	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

ГЛАВА 8

ЦИФРОВЫЕ ПОРТЫ ВВОДА/ВЫВОДА

В этой главе описывается функционирование цифровых портов ввода/вывода микроконтроллеров семейства MSP430x2xx.

8.1. Введение

Микроконтроллеры семейства могут иметь до восьми портов ввода/вывода P1...P8. Все порты содержат по восемь выводов (линий). Каждый из выводов порта индивидуально конфигурируется в качестве входа или выхода. Запись и чтение линий ввода/вывода также может осуществляться в индивидуальном порядке.

Порты P1 и P2 поддерживают внешние прерывания. Для каждого из выводов портов P1 и P2 можно индивидуально разрешить прерывание и сконфигурировать его так, чтобы оно генерировалось по нарастающему или спадающему фронту входного сигнала. Все линии ввода/вывода порта P1 назначены одному вектору прерываний, а все линии порта P2 — другому вектору.

Цифровые порты ввода/вывода обладают следующими возможностями:

- независимые индивидуально программируемые линии ввода/вывода;
- любые комбинации входов или выходов;
- индивидуально конфигурируемые прерывания от выводов портов P1 и P2;
- раздельные регистры данных для входов и выходов;
- индивидуально конфигурируемые внутренние подтягивающие резисторы.

8.2. Функционирование цифровых портов ввода/вывода

Конфигурирование цифровых портов ввода/вывода осуществляется пользовательской программой. Настройка и функционирование цифровых портов описывается в следующих подразделах.

8.2.1. Регистр данных входа PxIN

Каждый бит регистра PxIN отражает уровень входного сигнала на соответствующем выводе порта, если этот вывод используется в качестве цифрового входа/выхода.

- Бит = 0: Входной сигнал имеет НИЗКИЙ уровень.
- Бит = 1: Входной сигнал имеет ВЫСОКИЙ уровень.

Примечание. Запись в регистры PxIN, предназначенные только для чтения
Запись в любой из регистров PxIN приводит к увеличению тока потребления на время выполнения этой операции.

8.2.2. Регистр данных выхода PxOUT

Значение каждого бита регистра PxOUT определяет состояние соответствующего вывода порта, если этот вывод сконфигурирован как цифровой выход, и внутренний подтягивающий резистор не используется.

- Бит = 0: Выходной сигнал имеет НИЗКИЙ уровень.
- Бит = 1: Выходной сигнал имеет ВЫСОКИЙ уровень.

Если используется внутренний подтягивающий резистор, то значение бита регистра PxOUT определяет тип «подтяжки» на соответствующем выводе порта.

- Бит = 0: Вывод подтягивается к общему проводу.
- Бит = 1: Вывод подтягивается к питанию.

8.2.3. Регистр направления PxDIR

Значение каждого бита регистра PxDIR определяет направление передачи данных соответствующего вывода порта, независимо от выбранной для этого вывода функции. Если вывод используется каким-либо периферийным модулем, то бит регистра PxDIR должен быть установлен в соответствии с требованиями данного модуля.

- Бит = 0: Вывод порта переключается на вход.
- Бит = 1: Вывод порта переключается на выход.

8.2.4. Регистр включения подтягивающих резисторов PxREN

Каждый бит регистра PxREN подключает или отключает внутренний подтягивающий резистор соответствующего вывода порта. Тип «подтяжки» определяется соответствующим битом регистра PxOUT.

- Бит = 0: Подтягивающий резистор отключен.
- Бит = 1: Подтягивающий резистор подключен.

8.2.5. Регистры выбора функции PxSEL и PxSEL2

Большинство выводов портов используются различными периферийными модулями. Для выяснения альтернативных функций выводов обратитесь к документации на конкретный микроконтроллер. Каждый из битов регистров PxSEL и PxSEL2 используется для выбора функции соответствующего вывода микроконтроллера: линия порта ввода/вывода или вывод периферийного модуля.

PxSEL	PxSEL2	Функция вывода
0	0	Линия порта ввода/вывода
0	1	Вывод основного периферийного модуля
1	0	Зарезервировано. См. документацию на конкретную модель
1	1	Вывод дополнительного периферийного модуля

Установка PxSELx = 1 не задаёт автоматически направление передачи данных вывода. Для определённых периферийных модулей может потребоваться конфигурирование битов PxDIRx в соответствии с требованиями модуля. Обратитесь к принципиальным схемам выводов, приведённым в документации на конкретные модели.

Примечание. Установка PxREN = 1 при PxSEL = 1
Для некоторых портов ввода/вывода микроконтроллеров MSP430F261x и MSP430F2416/7/8/9 включение подтягивающего резистора (PxREN = 1) при использовании вывода порта периферийным модулем (PxSEL = 1) не приводит к отключению выходного драйвера. Использование такой конфигурации не рекомендуется, поскольку она может вызвать нежелательное увеличение тока, протекающего через внутренний резистор. Для получения более подробной информации обратитесь к принципиальным схемам выводов, приведённым в документации на конкретные модели.

```
; Выдача ACLK на P2.0 в моделях MSP430F21x1
  BIS.B #01h,&P2SEL   ; Выбираем функцию ACLK для вывода
  BIS.B #01h,&P2DIR   ; Переключаем вывод порта на выход (обязательно)
```

Примечание. Запрещение прерываний от P1 и P2 при PxSEL = 1
Если установлен любой из битов PxSEL или PxSEL2, то генерация прерывания от соответствующего вывода запрещена. Поэтому сигналы на таких выводах не вызовут генерации внешних прерываний от P1 или P2, независимо от состояния соответствующего бита P1IE или P2IE.

Если вывод порта используется в качестве входа периферийного модуля, то входным сигналом для данного модуля является зафиксированное в регистре-защёлке значение сигнала с вывода микроконтроллера. При PxSELx = 1 внутренний входной сигнал повторяет сигнал, присутствующий на выводе. Однако при PxSELx = 0 на входе периферийного модуля сохраняется значение сигнала, которое присутствовало на выводе микроконтроллера в момент сброса бита PxSELx.

8.2.6. Прерывания от портов P1 и P2

Любой из выводов портов P1 и P2 может генерировать прерывание. Конфигурирование этой функции осуществляется с помощью регистров PxIFG, PxIE и PxIES. Все выводы порта P1 связаны с одним вектором прерывания, а все выводы порта P2 — с другим вектором. Для определения конкретного источника прерывания от портов P1 или P2 можно проверить содержимое соответствующего регистра PxIFG.

Регистры флагов прерываний P1IFG и P2IFG

Каждый из битов PxIFGx является флагом прерывания от соответствующего вывода порта и устанавливается при появлении на выводе заданного фронта сигнала. Любой из флагов PxIFGx генерирует запрос прерывания, если установлен соответствующий бит регистра PxIE и бит общего разрешения прерываний GIE. Все флаги PxIFG должны сбрасываться программно. Кроме того, любой флаг PxIFG может быть установлен вручную для программной генерации прерывания.

- Бит = 0: Не было прерывания.
- Бит = 1: Было прерывание.

Прерывания генерируются только по фронту сигнала, а не по его уровню. Если любой из флагов PxIFGx будет установлен во время выполнения подпрограммы обработки прерывания от порта Px или же после выполнения команды `RETI` данной подпрограммы, то такой флаг вызовет генерацию нового прерывания. Это гарантирует отработку всех изменений уровня сигнала.

Примечание. Флаги PxIFG при изменении регистров PxOUT или PxDIR
Запись в любой из регистров P1OUT, P1DIR, P2OUT или P2DIR может привести к установке соответствующих флагов в регистрах P1IFG или P2IFG.

Регистры выбора фронта прерывания P1IES и P2IES

Значение каждого бита регистра PxIES определяет, по какому фронту сигнала будет генерироваться прерывание от соответствующего вывода порта.

- Бит = 0: Флаг PxIFGx устанавливается по нарастающему фронту.
- Бит = 1: Флаг PxIFGx устанавливается по спадающему фронту.

Примечание. Запись в регистры PxIES
Запись в регистр P1IES или P2IES может привести к установке соответствующих флагов прерываний.

PxIESx	PxINx	PxIFGx
$0 \rightarrow 1$	0	Может быть установлен
$0 \rightarrow 1$	1	Не изменяется
$1 \rightarrow 0$	0	Не изменяется
$1 \rightarrow 0$	1	Может быть установлен

Регистры разрешения прерываний P1IE и P2IE

Каждый бит регистра PxIE разрешает генерацию прерывания при установке соответствующего флага прерывания PxIFG.

- Бит = 0: Прерывание запрещено.
- Бит = 1: Прерывание разрешено.

8.2.7. Конфигурация неиспользуемых выводов портов

Неиспользуемые выводы микроконтроллера необходимо сконфигурировать как выходы портов ввода/вывода и оставить неподключенными, чтобы избежать появления «плавающих» входов и снизить ток потребления устройства. Значение бита PxOUT для такого вывода может быть любым, поскольку вывод никуда не подключен. В качестве альтернативы, чтобы избежать появления «плавающего» входа, можно к неиспользуемому выводу подключить внутренний подтягивающий резистор, установив соответствующий бит регистра PxREN. Более подробно вопрос подключения неиспользуемых выводов рассмотрен в главе 2 «Сброс, прерывания и режимы работы».

8.3. Регистры цифровых портов ввода/вывода

Регистры цифровых портов ввода/вывода перечислены в **Табл. 8.1**.

Таблица 8.1. **Регистры цифровых портов ввода/вывода**

Порт	Регистр	Обозначение	Адрес	Тип регистра	Начальное состояние
P1	Вход	P1IN	020h	Только чтение	–
	Выход	P1OUT	021h	Чтение/запись	Не изменяется
	Направление	P1DIR	022h	Чтение/запись	Сбрасывается после PUC
	Флаг прерывания	P1IFG	023h	Чтение/запись	Сбрасывается после PUC
	Фронт прерывания	P1IES	024h	Чтение/запись	Не изменяется
	Разрешение прерывания	P1IE	025h	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P1SEL	026h	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P1SEL2	041h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P1REN	027h	Чтение/запись	Сбрасывается после PUC
P2	Данные входа	P2IN	028h	Только чтение	–
	Данные выхода	P2OUT	029h	Чтение/запись	Не изменяется
	Направление	P2DIR	02Ah	Чтение/запись	Сбрасывается после PUC
	Флаг прерывания	P2IFG	02Bh	Чтение/запись	Сбрасывается после PUC
	Фронт прерывания	P2IES	02Ch	Чтение/запись	Не изменяется
	Разрешение прерывания	P2IE	02Dh	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P2SEL	02Eh	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P2SEL2	042h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P2REN	02Fh	Чтение/запись	Сбрасывается после PUC
P3	Вход	P3IN	018h	Только чтение	–
	Выход	P3OUT	019h	Чтение/запись	Не изменяется
	Направление	P3DIR	01Ah	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P3SEL	01Bh	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P3SEL2	043h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P3REN	010h	Чтение/запись	Сбрасывается после PUC
P4	Вход	P4IN	01Ch	Только чтение	–
	Выход	P4OUT	01Dh	Чтение/запись	Не изменяется
	Направление	P4DIR	01Eh	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P4SEL	01Fh	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P4SEL2	044h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P4REN	011h	Чтение/запись	Сбрасывается после PUC
P5	Вход	P5IN	030h	Только чтение	–
	Выход	P5OUT	031h	Чтение/запись	Не изменяется
	Направление	P5DIR	032h	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P5SEL	033h	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P5SEL2	045h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P5REN	012h	Чтение/запись	Сбрасывается после PUC
P6	Вход	P6IN	034h	Только чтение	–
	Выход	P6OUT	035h	Чтение/запись	Не изменяется
	Направление	P6DIR	036h	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P6SEL	037h	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P6SEL2	046h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P6REN	013h	Чтение/запись	Сбрасывается после PUC
P7	Вход	P7IN	038h	Только чтение	–
	Выход	P7OUT	03Ah	Чтение/запись	Не изменяется
	Направление	P7DIR	03Ch	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P7SEL	03Eh	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P7SEL2	047h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P7REN	014h	Чтение/запись	Сбрасывается после PUC
P8	Вход	P8IN	039h	Только чтение	–
	Выход	P8OUT	03Bh	Чтение/запись	Не изменяется
	Направление	P8DIR	03Dh	Чтение/запись	Сбрасывается после PUC
	Выбор функции	P8SEL	03Fh	Чтение/запись	Сбрасывается после PUC
	Выбор функции 2	P8SEL2	048h	Чтение/запись	Сбрасывается после PUC
	Включение резистора	P8REN	015h	Чтение/запись	Сбрасывается после PUC

ГЛАВА 9

СУПЕРВИЗОР НАПРЯЖЕНИЯ ПИТАНИЯ

В этой главе описывается функционирование супервизора напряжения питания (SVS), реализованного в микроконтроллерах семейства MSP430x2xx.

9.1. Введение

Супервизор напряжения питания (SVS) используется для мониторинга напряжения питания AV_{CC} или внешнего напряжения. Супервизор можно сконфигурировать так, чтобы он устанавливал флаг или формировал сигнал сброса POR при снижении напряжения питания или внешнего напряжения ниже порогового значения, заданного пользователем.

Модуль SVS обладает следующими возможностями:

- мониторинг напряжения питания AV_{CC};
- опциональная генерация сигнала POR;
- доступный программно выход компаратора SVS;
- фиксируемый признак обнаружения пониженного напряжения, доступный программно;
- выбор из 14 возможных пороговых значений;
- дополнительный канал для мониторинга внешнего напряжения.

Блок-схема супервизора напряжения питания приведена на **Рис. 9.1**.

9.2. Функционирование супервизора

Супервизор отслеживает снижение напряжения питания AV_{CC} ниже заданного уровня. Модуль SVS можно сконфигурировать так, чтобы при наступлении указанного события генерировался сигнал сброса POR или устанавливался флаг. После сброса по снижению напряжения питания (BOR) супервизор находится в выключенном состоянии для уменьшения тока потребления микроконтроллера.

9.2.1. Конфигурирование супервизора

Для включения/выключения супервизора, а также для выбора одного из 14 пороговых значений ($V_{(SVS_IT-)}$), с которым будет сравниваться напряжение AV_{CC}, используются биты VLDx. Супервизор выключен при VLDx = 0 и включен при

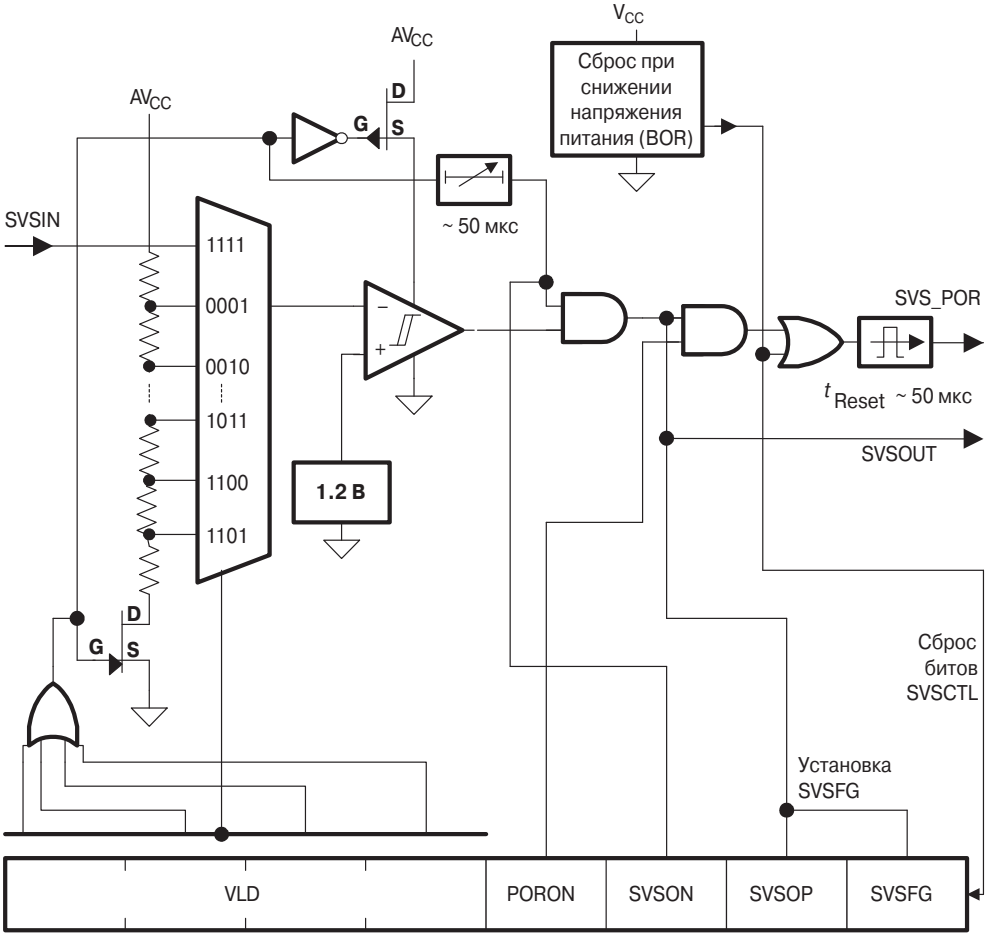


Рис. 9.1. Блок-схема супервизора напряжения питания.

VLDx > 0. Бит SVSON для включения супервизора не используется. Он только отображает его состояние и может применяться для определения, включён ли супервизор.

При LVDx = 1111 выбирается внешний канал SVSIN. Напряжение на входе SVSIN сравнивается с напряжением от внутреннего источника, составляющим приблизительно 1.2 В.

9.2.2. Функционирование компаратора супервизора

Состояние пониженного напряжения возникает при снижении AV_{CC} ниже заданного порога или при снижении внешнего напряжения ниже уровня 1.25 В. Любое из этих событий вызывает установку бита SVSFG.

Бит PORON разрешает или запрещает сброс микроконтроллера супервизором. Если PORON = 1, то при установке бита SVSFG генерируется сигнал POR.

Если PORON = 0, то снижение контролируемого напряжения ниже порогового значения вызывает только установку бита SVSFG, сигнал POR при этом не генерируется.

При установке бита SVSFG его значение фиксируется. Это даёт пользователю возможность узнать о снижении напряжения, произошедшем ранее. Бит SVSFG должен сбрасываться программно. Если на момент сброса бита SVSFG напряжение всё ещё будет ниже порогового значения, то сразу же после сброса этого бита супервизор установит его повторно.

9.2.3. Изменение битов VLDx

При изменении содержимого битов VLDx с нулевого значения на любое другое, отличное от нуля, автоматически формируется задержка $t_{d(SVSon)}$, необходимая для установления схемы супервизора. Длительность задержки $t_{d(SVSon)}$ составляет около 50 мкс. В течение этого времени бит SVSON будет сброшен, а супервизор не будет реагировать на понижение напряжения. Проверяя в программе состояние данного бита, можно определить момент окончания задержки и начала мониторинга напряжения модулем SVS. Запись в регистр SVSCTL при SVSON = 0 прервёт формирование задержки $t_{d(SVSon)}$ и немедленно переключит модуль супервизора в активный режим. В этом случае схема супервизора может не успеть установиться, что приведёт к её непредсказуемому поведению.

При изменении содержимого битов VLDx с одного ненулевого значения на другое для установления схемы супервизора требуется задержка t_{settle}. Длительность задержки t_{settle} составляет не более ~12 мкс (см. документацию на конкретный микроконтроллер). Во время этой задержки блокирование установки флага SVSFLG и/или генерации сигнала сброса устройства не производится. Переключение пороговых значений супервизора рекомендуется выполнять следующим образом:

```
; Первоначальное включение супервизора напряжения:
   MOV.B    #080h,&SVSCTL     ; Порог 2.8 В, сигнал POR не генерируется
;   ...

; Изменяем порог срабатывания супервизора
   MOV.B    #000h,&SVSCTL     ; Временно выключаем SVS
   MOV.B    #018h,&SVSCTL     ; Порог 1.9 В, сигнал POR генерируется
;   ...
```

9.2.4. Рабочий диапазон супервизора

Каждое пороговое значение модуля SVS имеет некоторый гистерезис, позволяющий уменьшить чувствительность схемы к небольшим изменениям напряжения питания в тех случаях, когда величина AV_{CC} близка к установленному порогу. Работа супервизора и его взаимодействие со схемой BOR показаны на **Рис. 9.2**.

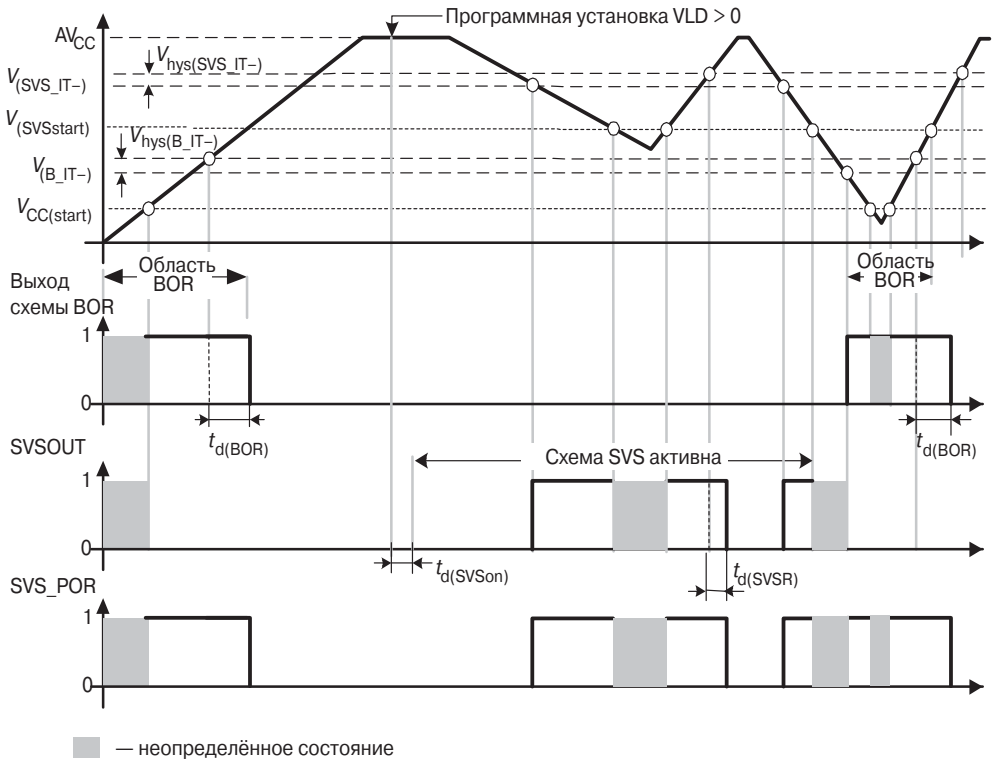


Рис. 9.2. Блок-схема супервизора напряжения питания.

9.3. Регистры супервизора

Список регистров супервизора приведён в **Табл. 9.1**.

Таблица 9.1. **Регистры супервизора**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления супервизора	SVSCTL	Чтение/запись	056h	Сбрасывается после BOR

SVSCTL, регистр управления супервизора

7	6	5	4	3	2	1	0
VLDx				**PORON**	**SVSON**	**SVSOP**	**SVSFG**
rw–0*	rw–0*	rw–0*	rw–0*	rw–0*	r*	r*	rw–0*

* Сбрасываются только при BOR, а не при POR или PUC.

VLDx Биты 7...4 Контролируемый уровень напряжения. Эти биты включают супервизор и определяют номинальное значение порогового напряжения. Прочие параметры супервизора приведены в документации на конкретные модели.

0000 Супервизор выключен
0001 1.9 В
0010 2.1 В
0011 2.2 В
0100 2.3 В
0101 2.4 В
0110 2.5 В
0111 2.65 В
1000 2.8 В
1001 2.9 В
1010 3.05 В
1011 3.2 В
1100 3.35 В
1101 3.5 В
1110 3.7 В
1111 Сравнивается напряжение на входе SVSIN с уровнем 1.25 В

PORON Бит 3 Разрешение POR. Этот бит разрешает генерирование сигнала сброса POR при установке флага SVSFG.

0 Бит SVSFG не влияет на сигнал POR
1 Бит SVSFG вызывает генерацию POR

SVSON Бит 2 Включение супервизора. Данный бит отражает состояние супервизора. Этот бит НЕ используется для включения супервизора. Модуль SVS включается установкой значения VLDx > 0.

0 Супервизор выключен
1 Супервизор включен

SVSOP Бит 1 Выход супервизора. Этот бит отражает состояние выхода компаратора модуля SVS.

0 На выходе компаратора НИЗКИЙ уровень
1 На выходе компаратора ВЫСОКИЙ уровень

SVSFG Бит 0 Флаг супервизора. Этот бит отражает наличие пониженного напряжения. Бит SVSFG остаётся установленным после выхода из данного состояния. Сбрасывается программно.

0 Состояние пониженного напряжения отсутствует
1 Состояние пониженного напряжения либо присутствует в данный момент, либо имело место раньше

ГЛАВА 10

СТОРОЖЕВОЙ ТАЙМЕР

Усовершенствованный сторожевой таймер (WDT+) представляет собой 16-битный таймер, который может использоваться в качестве сторожевого либо интервального таймера. В этой главе описывается функционирование модуля усовершенствованного сторожевого таймера, реализованного во всех микроконтроллерах семейства MSP430x2xx.

10.1. Введение

Основная функция модуля WDT+ заключается в выполнении управляемого перезапуска системы при некорректном функционировании программы. При «зависании» программы на время, превышающее заданный интервал, модуль генерирует сигнал системного сброса. Если сторожевой таймер в приложении не требуется, то модуль может использоваться в качестве обычного интервального таймера, генерирующего прерывания с заданной периодичностью.

Модуль усовершенствованного сторожевого таймера WDT+ имеет следующие особенности:

- четыре временных интервала, выбираемые программно;
- режим сторожевого таймера;
- режим интервального таймера;
- парольная защита доступа к регистру управления модуля;
- управление функционированием вывода $\overline{RST}$/NMI;
- возможность выбора источника тактового сигнала;
- возможность останова для уменьшения тока потребления;
- защита от пропадания тактового сигнала.

Блок-схема модуля WDT+ приведена на **Рис. 10.1**.

Примечание. Работа модуля WDT+ после подачи напряжения питания
После появления сигнала PUC модуль WDT+ автоматически запускается в режиме сторожевого таймера с интервалом сброса, равным 32 768 тактам сигнала DCOCLK. Пользователь должен сконфигурировать или остановить сторожевой таймер до истечения этого интервала.

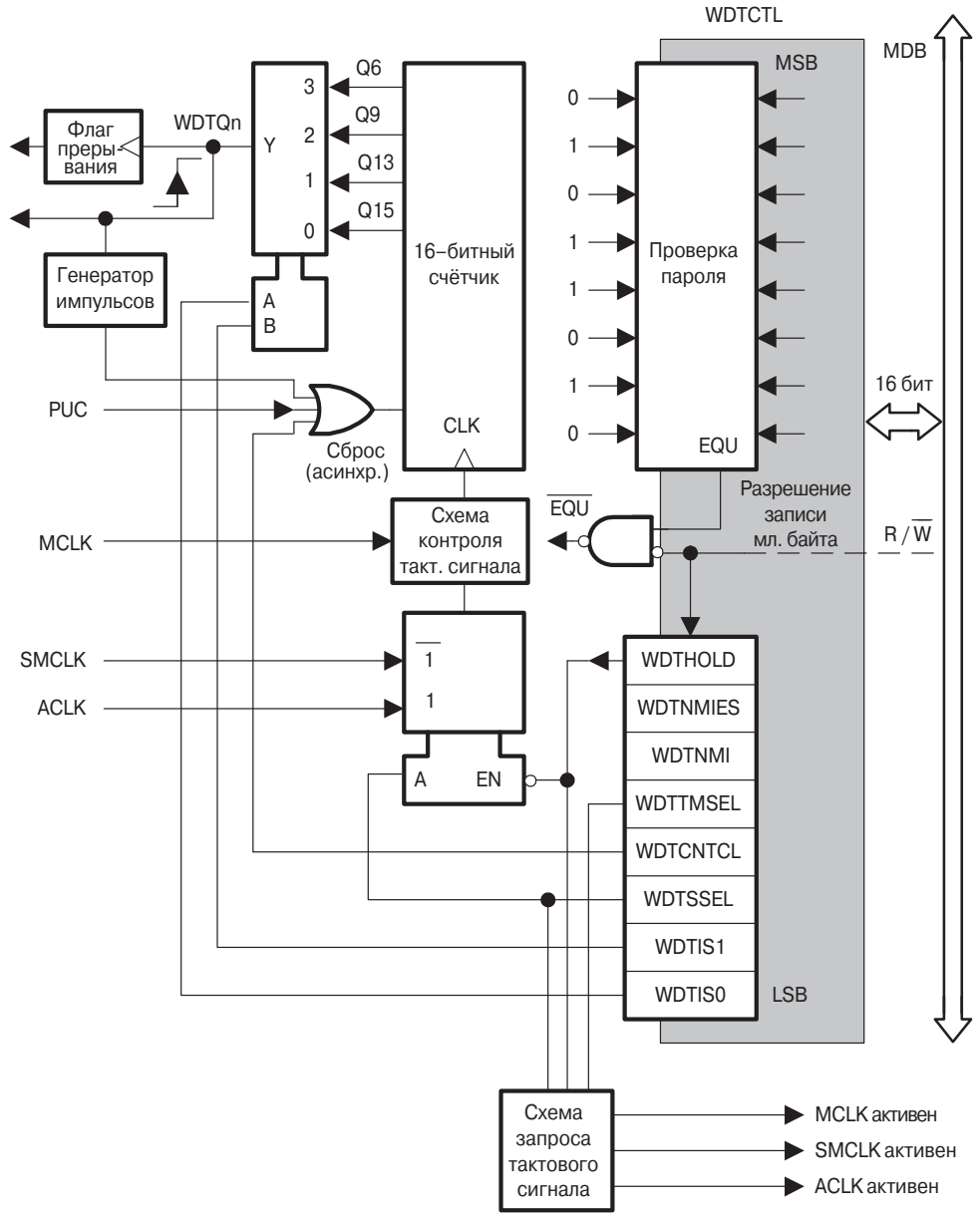


Рис. 10.1. Блок-схема модуля WDT+.

10.2. Функционирование сторожевого таймера

Модуль WDT+ может быть сконфигурирован для работы в режиме сторожевого либо интервального таймера посредством регистра WDTCTL. Кроме того, в регистре WDTCTL содержатся биты, определяющие конфигурацию вывода

$\overline{\text{RST}}$/NMI. Регистр WDTCTL — это защищённый паролем 16-битный регистр, доступный как для чтения, так и для записи. Любые обращения к данному регистру должны производиться с использованием команд, оперирующих 2-байтными операндами, причём при операциях записи в старшем байте записываемого значения должно содержаться число 0A5h. Запись в регистр WDTCTL значения, старший байт которого не равен 0A5h, вызовет нарушение ключа защиты с последующим формированием сигнала сброса PUC, независимо от режима работы модуля. При чтении регистра WDTCTL в старшем байте возвращается значение 069h. Частота тактового сигнала сторожевого таймера должна быть меньше или равна частоте системного тактового сигнала (MCLK).

10.2.1. Счётчик сторожевого таймера

Счётчик сторожевого таймера WDTCNT представляет собой 16-битный суммирующий счётчик, который напрямую из программы не доступен. Управление счётчиком, в том числе задание временных интервалов, осуществляется при помощи регистра управления WDTCTL.

Счётчик WDTCNT может тактироваться от сигналов ACLK или SMCLK. Источник тактового сигнала задаётся битом WDTSSEL.

10.2.2. Режим сторожевого таймера

После сброса по включению питания модуль WDT+ начинает работать в режиме сторожевого таймера с интервалом сброса, равным 32 768 тактам сигнала DCOCLK. Пользователь должен перенастроить, остановить или сбросить сторожевой таймер до истечения этого интервала, в противном случае сигнал сброса PUC будет сгенерирован повторно. При работе модуля в режиме сторожевого таймера сигнал сброса PUC генерируется либо при записи в регистр WDTCTL с некорректным паролем, либо по истечении выбранного интервала времени. При появлении сигнала PUC модуль WDT+ сбрасывается в исходное состояние, а вывод $\overline{\text{RST}}$/NMI конфигурируется как вход аппаратного сброса.

10.2.3. Режим интервального таймера

Установка бита WDTTMSEL в 1 переводит модуль WDT+ в режим интервального таймера. Этот режим может использоваться для периодической генерации прерываний. В режиме интервального таймера по истечении выбранного интервала устанавливается флаг WDTIFG. Сигнал сброса PUC в данном режиме не генерируется, а состояние бита разрешения прерывания WDTIE регистра WDTIFG не изменяется.

Если биты WDTIE и GIE установлены, то при установке флага WDTIFG генерируется запрос прерывания. Флаг прерывания WDTIFG сбрасывается автоматически при вызове процедуры обработки этого прерывания или может быть сброшен программно. Адрес вектора прерывания в режиме интервального таймера отличается от адреса вектора в режиме сторожевого таймера.

Примечание. Изменение состояния сторожевого таймера
Изменение интервала сторожевого таймера необходимо выполнять одновременно (в одной команде) с установкой бита WDTCNTCL, чтобы исключить несанкционированную генерацию сигнала PUC или прерывания.
Перед сменой источника тактового сигнала модуль WDT+ необходимо останавливать, чтобы предотвратить возможную установку некорректного интервала.

10.2.4. Прерывания сторожевого таймера

Для управления прерываниями модуля WDT+ используются два бита регистров специальных функций:

- флаг прерывания WDTIFG (IFG1.0);
- бит разрешения прерывания WDTIE (IE1.0).

При работе модуля WDT+ в режиме сторожевого таймера флаг WDTIFG является одним из источников вектора сброса. Этот флаг можно использовать в процедуре обработки сброса для определения причины сброса микроконтроллера. Установленный флаг WDTIFG означает, что сброс был инициирован сторожевым таймером по истечении заданного интервала времени или в результате нарушения ключа защиты. Если же флаг WDTIFG сброшен, то причина сброса была иной.

При работе модуля WDT+ в режиме интервального таймера флаг WDTIFG устанавливается по истечении выбранного интервала времени и, если установлены биты WDTIE и GIE, генерирует запрос прерывания. Вектор прерывания интервального таймера не совпадает с вектором сброса, используемого в режиме сторожевого таймера. В режиме интервального таймера флаг WDTIFG сбрасывается автоматически при обработке прерывания либо может быть сброшен программно.

10.2.5. Отказоустойчивое тактирование сторожевого таймера

Модуль WDT+ имеет функцию защиты от пропадания тактового сигнала, которая блокирует отключение источника тактового сигнала модуля при его использовании в режиме сторожевого таймера. То есть выбор источника тактового сигнала модуля WDT+ влияет на возможность использования того или иного режима пониженного энергопотребления. Например, если сторожевой таймер тактируется сигналом ACLK, то режим LPM4 будет недоступен, поскольку модуль WDT+ не позволит отключить источник ACLK. Кроме того, в случае пропадания сигнала ACLK или SMCLK, используемого для тактирования модуля, сторожевой таймер автоматически переключится на сигнал MCLK. Если сигнал MCLK формируется кварцевым генератором, то в случае неисправности последнего (или при отсутствии резонатора) модуль WDT+ активирует DCO и задействует его в качестве источника сигнала MCLK.

10.2.6. Функционирование в режимах пониженного энергопотребления

Микроконтроллеры MSP430 имеют несколько режимов пониженного энергопотребления. В разных режимах пользователю доступны различные источники тактовых сигналов. Соответственно, конфигурация модуля WDT+ определяется требованиями конкретного приложения и режимом работы модуля синхронизации микроконтроллера. Например, если пользователь предполагает задействовать режим LPM3, то модуль WDT+ не должен использоваться в режиме сторожевого таймера с тактированием от сигнала SMCLK, поскольку модуль WDT+ не позволит выключить источник этого сигнала, что приведет к повышенному потреблению в режиме LPM3. Если сторожевой таймер не используется, то с помощью бита WDTHOLD можно остановить счётчик WDTCNT, уменьшая тем самым потребление микроконтроллера.

10.2.7. Примеры кода

Любые операции записи в регистр WDTCTL должны производиться с использованием команд, оперирующих 2-байтными операндами, при этом в старшем байте записываемого значения должно содержаться число 0A5h (WDTPW).

```
; Периодический сброс активного сторожевого таймера
    MOV #WDTPW+WDTCNTCL,&WDTCTL
;
; Изменение интервала сторожевого таймера
    MOV #WDTPW+WDTCNTL+WDTSSEL,&WDTCTL
;
; Останов сторожевого таймера
    MOV #WDTPW+WDTHOLD,&WDTCTL
;
; Переключение модуля WDT+ в режим интервального таймера с периодом clock/8192
    MOV #WDTPW+WDTCNTCL+WDTTMSEL+WDTIS0,&WDTCTL
```

10.3. Регистры сторожевого таймера

Список регистров сторожевого таймера приведён в **Табл. 10.1**.

Таблица 10.1. **Регистры сторожевого таймера**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления сторожевого таймера	WDTCTL	Чтение/запись	0120h	06900h после PUC
Регистр разрешения прерываний 1	IE1	Чтение/запись	0000h	Сбрасывается после PUC
Регистр флагов прерываний 1	IFG1	Чтение/запись	0002h	Сбрасывается после PUC*
* Бит WDTIFG сбрасывается после POR.				

WDTCTL, регистр управления модуля WDT+

15	14	13	12	11	10	9	8
Читается как 069h WDTPW, должен записываться как 05Ah							

7	6	5	4	3	2	1	0
WDTHOLD	**WDTNMIES**	**WDTNMI**	**WDTTMSEL**	**WDTCNTCL**	**WDTSSEL**	**WDTISx**	
rw–0	rw–0	rw–0	rw–0	r0(w)	rw–0	rw–0	rw–0

*Отсутствуют в микроконтроллерах MSP430x20xx.

WDTPW Биты 15...8 Ключ защиты WDT+. Всегда читается как 069h. При записи должен быть равен 05Ah, в противном случае будет генерироваться сигнал PUC.

WDTHOLD Бит 7 Останов модуля WDT+. Этот бит используется для останова модуля WDT+. Установка бита WDTHOLD = 1 при неиспользуемом модуле позволяет уменьшить ток потребления микроконтроллера.
- 0 Модуль WDT+ не остановлен
- 1 Модуль WDT+ остановлен

WDTNMIES Бит 6 Выбор фронта NMI. Этот бит используется для выбора активного фронта немаскируемого прерывания при WDTNMI = 1. Изменение данного бита может вызвать генерацию немаскируемого прерывания. Чтобы избежать этого, бит WDTNMIES следует изменять при WDTIE = 1.
- 0 Немаскируемое прерывание генерируется по нарастающему фронту
- 1 Немаскируемое прерывание генерируется по спадающему фронту

WDTNMI Бит 5 Выбор NMI. С помощью этого бита задаётся режим работы вывода $\overline{\text{RST}}$/NMI микроконтроллера.
- 0 Вход аппаратного сброса
- 1 Вход немаскируемого прерывания

WDTTMSEL Бит 4 Выбор режима модуля WDT+.
- 0 Режим сторожевого таймера
- 1 Режим интервального таймера

WDTCNTCL Бит 3 Сброс счётчика модуля WDT+. Установка бита WDTCNTCL = 1 вызывает загрузку в счётчик значения 0000h. Бит WDTCNTCL сбрасывается автоматически.
- 0 Нет действия
- 1 WDTCNT = 0000h

WDTSSEL Бит 2 Выбор тактового сигнала модуля WDT+.
- 0 SMCLK
- 1 ACLK

WDTISx Биты 1...0 Выбор интервала модуля WDT+. Эти биты определяют временной интервал, по истечении которого устанавливается флаг WDTIFG и/или генерируется сигнал PUC.
- 00 Частота тактового сигнала /32768
- 01 Частота тактового сигнала /8192
- 10 Частота тактового сигнала /512
- 11 Частота тактового сигнала /64

IE1, регистр разрешения прерываний 1

7	6	5	4	3	2	1	0
			NMIIE				**WDTIE**
			rw–0				rw–0

	Биты 7...5	Эти биты могут использоваться другими модулями. См. документацию на конкретный микроконтроллер.
NMIIE	Бит 4	Разрешение немаскируемого прерывания. Этот бит разрешает немаскируемое прерывание. Поскольку прочие биты регистра IE1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Прерывание запрещено 1 Прерывание разрешено
	Биты 3...1	Эти биты могут использоваться другими модулями. См. документацию на конкретный микроконтроллер.
WDTIE	Бит 0	Разрешение прерывания модуля WDT+. Данный бит разрешает прерывание WDTIFG при работе модуля в режиме интервального таймера. При использовании режима сторожевого таймера установка этого бита не требуется. Поскольку прочие биты регистра IE1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Прерывание запрещено 1 Прерывание разрешено

IFG1, регистр флагов прерываний 1

7	6	5	4	3	2	1	0
			NMIIFG				**WDTIFG**
			rw–0				rw–(0)

	Биты 7...5	Эти биты могут использоваться другими модулями. См. документацию на конкретный микроконтроллер.
NMIIFG	Бит 4	Флаг немаскируемого прерывания. Бит NMIIFG должен сбрасываться программно. Поскольку прочие биты регистра IFG1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Не было запроса прерывания 1 Есть запрос прерывания
	Биты 3...1	Эти биты могут использоваться другими модулями. См. документацию на конкретный микроконтроллер.
WDTIFG	Бит 0	Флаг прерывания модуля WDT+. В режиме сторожевого таймера бит WDTIFG остаётся установленным до тех пор, пока не будет сброшен программно. В режиме интервального таймера бит WDTIFG сбрасывается автоматически при обслуживании прерывания или же может быть сброшен программно. Поскольку прочие биты регистра IE1 могут использоваться другими модулями, то для установки или очистки битов регистра рекомендуется вместо команд `MOV.B` или `CLR.B` применять команды `BIS.B` и `BIC.B`. 0 Не было запроса прерывания 1 Есть запрос прерывания

ГЛАВА 11

АППАРАТНЫЙ УМНОЖИТЕЛЬ

В этой главе описывается модуль аппаратного умножителя, реализованного в некоторых моделях микроконтроллеров семейства MSP430x2xx.

11.1. Введение

Аппаратный умножитель представляет собой специализированное периферийное устройство и не входит в состав ЦПУ MSP430. Соответственно, его функционирование никак не влияет на работу ЦПУ микроконтроллера. Регистры умножителя — это регистры периферийного устройства, для загрузки и чтения содержимого которых используются команды ЦПУ.

Аппаратный умножитель поддерживает:

- умножение без знака;
- умножение со знаком;
- умножение без знака с накоплением;
- умножение со знаком и с накоплением;
- разрядность 16 $\times$ 16 бит, 16 $\times$ 8 бит, 8 $\times$ 16 бит, 8 $\times$ 8 бит.

Блок-схема аппаратного умножителя приведена на **Рис. 11.1**.

11.2. Функционирование аппаратного умножителя

Аппаратный умножитель поддерживает операции умножения без знака, умножения со знаком, умножения без знака с накоплением и умножения со знаком и с накоплением. Тип операции определяется адресом, по которому располагается первый операнд.

Аппаратный умножитель содержит два 16-битных регистра операндов (OP1 и OP2) и три регистра результата (RESLO, RESHI и SUMEXT). В регистре RESLO хранится младшее слово результата, в регистре RESH — старшее слово результата, а в регистре SUMEXT содержится информация о результате. Для формирования результата требуется три такта MCLK, поэтому его можно прочитать уже с помощью команды, следующей за командой записи в регистр OP2. При использовании косвенной адресации для обращения к регистрам результата, перед их чтением необходимо вставить одну команду `NOP`.

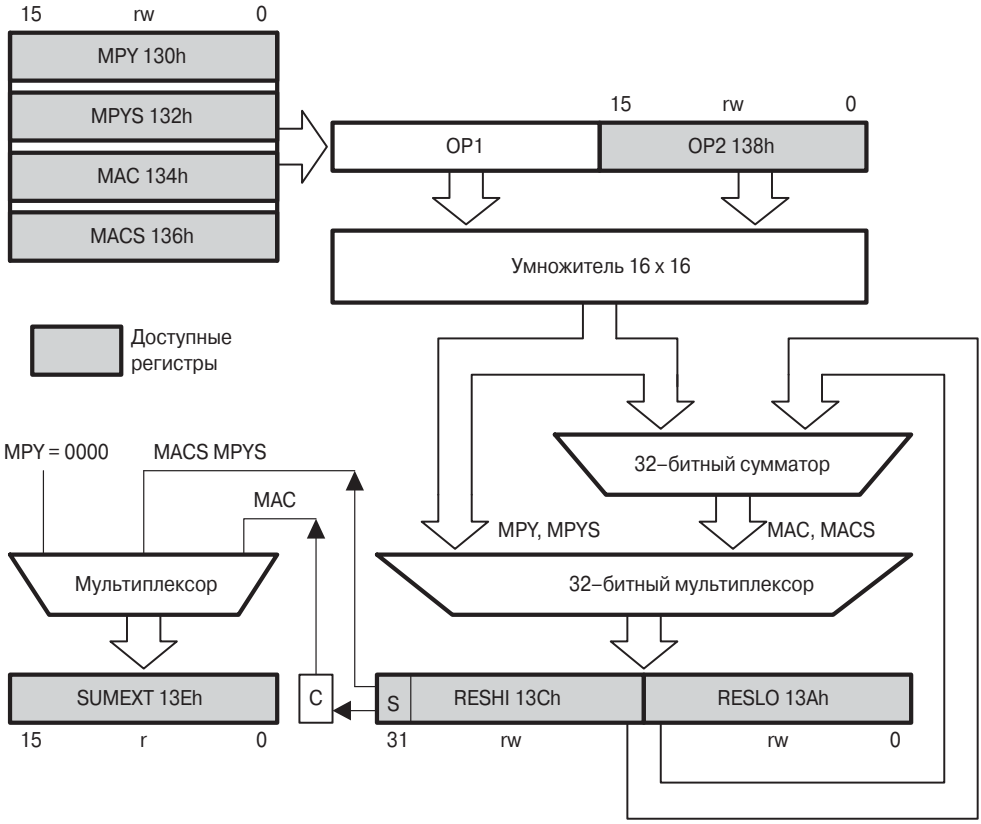


Рис. 11.1. Блок-схема аппаратного умножителя.

11.2.1. Регистры операндов

Регистр первого операнда OP1 имеет 4 адреса, указанные в **Табл. 11.1**, которые используются для выбора режима умножения.

Таблица 11.1. **Адреса регистра OP1**

Адрес OP1	Название регистра	Операция
0130h	MPY	Умножение без знака
01332h	MPYS	Умножение со знаком
0134h	MAC	Умножение без знака с накоплением
0136h	MACS	Умножение со знаком и с накоплением

Загрузка первого операнда по выбранному адресу определяет конкретный тип операции умножения. Запись второго операнда в регистр OP2 инициирует выполнение операции заданного типа над значениями, загруженными в регистры операндов OP1 и OP2. Результат умножения сохраняется в трёх регистрах результата RESLO, RESHI и SUMEXT.

Если значение, содержащееся в регистре OP1, используется в нескольких операциях умножения, то для выполнения последующих операций перезагрузка регистра OP1 не требуется.

11.2.2. Регистры результата

Младший регистр результата RESLO содержит младшие 16 бит полученного результата. Содержимое старшего регистра результата RESHI зависит от типа операции умножения (**Табл. 11.2**).

Таблица 11.2. **Содержимое регистра RESHI**

Режим	Содержимое регистра
MPY	Старшие 16 бит результата
MPYS	В старшем бите содержится знак результата. В остальных битах содержатся старшие 15 бит результата. Результат представлен в дополнительном коде
MAC	Старшие 16 бит результата
MACS	Старшие 16 бит результата. Результат представлен в дополнительном коде

Содержимое регистра SUMEXT также зависит от типа операции умножения (**Табл. 11.3**).

Таблица 11.3. **Содержимое регистра SUMEXT**

Режим	Содержимое регистра
MPY	SUMEXT всегда содержит 0000h
MPYS	SUMEXT содержит расширенный знак результата: `00000h` Результат положителен или равен нулю `0FFFFh` Результат отрицателен
MAC	SUMEXT содержит признак переноса: `00000h` Переноса не было `00001h` Был перенос
MACS	SUMEXT содержит расширенный знак результата: `00000h` Результат положителен или равен нулю `0FFFFh` Результат отрицателен

Потеря значимости и переполнение в режиме MACS

Аппаратный умножитель не может автоматически определить переполнение или потерю значимости в режиме MACS. Диапазон допустимых значений аккумулятора составляет 0...7FFF FFFFh для положительных чисел и 0FFF FFFFh...8000 0000h для отрицательных. Потеря значимости происходит, когда результат сложения двух отрицательных чисел оказывается в диапазоне положительных значений. Переполнение происходит, когда результат сложения двух положительных чисел оказывается в диапазоне отрицательных значений. В обоих случаях в регистре SUMEXT содержится знак результата и соответствующее значение: 0FFFFh при переполнении и 0000h при потере значимости. Пользовательская программа должна отслеживать такие ситуации и обрабатывать их соответствующим образом.

11.2.3. Примеры кода

Ниже приведены примеры для всех четырёх режимов работы умножителя. При перемножении 8-битных значений обращение к регистрам производится по их абсолютным адресам, поскольку ассемблер не позволяет осуществлять побайтный доступ к двухбайтным регистрам при использовании меток из стандартного файла определений.

В программе не требуется специально выполнять расширение знака операндов. Если при выполнении операций умножения со знаком для обращения к регистрам умножителя используются байтовые команды, то расширение знака однобайтного операнда производится автоматически самим модулем.

```
; 16x16 Умножение без знака
  MOV   #01234h,&MPY   ; Загружаем первый операнд
  MOV   #05678h,&OP2   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 8x8 Умножение без знака. Абсолютная адресация
  MOV.B #012h,&0130h   ; Загружаем первый операнд
  MOV.B #034h,&0138h   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 16x16 Умножение со знаком
  MOV   #01234h,&MPYS  ; Загружаем первый операнд
  MOV   #05678h,&OP2   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 8x8 Умножение со знаком. Абсолютная адресация.
  MOV.B #012h,&0132h   ; Загружаем первый операнд
  MOV.B #034h,&0138h   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 16x16 Умножение без знака с накоплением
  MOV   #01234h,&MAC   ; Загружаем первый операнд
  MOV   #05678h,&OP2   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 8x8 Умножение без знака с накоплением. Абсолютная адресация
  MOV.B #012h,&0134h   ; Загружаем первый операнд
  MOV.B #034h,&0138h   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 16x16 Умножение со знаком и с накоплением
  MOV   #01234h,&MACS  ; Загружаем первый операнд
  MOV   #05678h,&OP2   ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
; 8x8 Умножение со знаком и с накоплением. Абсолютная адресация
  MOV.B #012h,&0136h   ; Загружаем первый операнд
  MOV.B #034h,R5       ; Временное хранилище для 2-го операнда
  MOV   R5,&OP2        ; Загружаем второй операнд
; ...                  ; Обрабатываем результат
```

11.2.4. Косвенная адресация RESLO

При обращении к регистрам результата с использованием режимов косвенной адресации или косвенной адресации с автоинкрементом между загрузкой

второго операнда и чтением любого из регистров результата должна присутствовать, по крайней мере, одна команда.

```
; Использование косвенной адресации для обращения
; к регистрам результата умножителя
  MOV #RESLO,R5      ; Загружаем адрес RESLO в R5 для косвенной адресации
  MOV &OPER1,&MPY    ; Загружаем 1-й операнд
  MOV &OPER2,&OP2    ; Загружаем 2-й операнд
  NOP                ; Требуется один такт
  MOV @R5+,&xxx      ; Копируем RESLO
  MOV @R5,&xxx       ; Копируем RESHI
```

11.2.5. Использование прерываний

Если прерывание, в процедуре обработки которого используется модуль умножителя, произойдёт после записи в регистр OP1, но до записи в регистр OP2, то первоначально выбранный режим работы умножителя будет потерян, и результат операции будет неопределённым. Во избежание такой ситуации следует запрещать прерывания на время использования аппаратного умножителя или же не использовать его в процедурах обработки прерываний.

```
; Запрещение прерываний на время использования аппаратного умножителя
  DINT            ; Запрещаем прерывания
  NOP             ; Требуется для DINT
  MOV #xxh,&MPY   ; Загружаем 1-й операнд
  MOV #xxh,&OP2   ; Загружаем 2-й операнд
  EINT            ; Теперь можем разрешить прерывания
                  ; Обрабатываем результат
```

11.3. Регистры аппаратного умножителя

Список регистров аппаратного умножителя приведён в **Табл. 11.4**.

Таблица 11.4. **Регистры аппаратного умножителя**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Первый операнд — умножение без знака	MPY	Чтение/запись	0130h	Не изменяется
Первый операнд — умножение со знаком	MPYS	Чтение/запись	0132h	Не изменяется
Первый операнд — умножение без знака с накоплением	MAC	Чтение/запись	0134h	Не изменяется
Первый операнд — умножение со знаком и с накоплением	MACS	Чтение/запись	0136h	Не изменяется
Второй операнд	OP2	Чтение/запись	0138h	Не изменяется
Результат, младшее слово	RESLO	Чтение/запись	013Ah	Не определено
Результат, старшее слово	RESHI	Чтение/запись	013Ch	Не определено
Регистр расширения знака	SUMEXT	Чтение	013Eh	Не определено

ГЛАВА 12

ТАЙМЕР А

Таймер А представляет собой 16-битный таймер/счётчик, имеющий несколько регистров захвата/сравнения. В этой главе описывается работа Таймера А, реализованного во всех микроконтроллерах семейства MSP430x2xx.

12.1. Введение

Таймер А представляет собой 16-битный таймер/счётчик с тремя регистрами захвата/сравнения. Этот таймер может обеспечить несколько каналов захвата/сравнения, генерации сигналов с ШИМ и формирования временных интервалов. Кроме того, Таймер А имеет развитую поддержку прерываний. Прерывания могут генерироваться как регистром счётчика таймера (при его переполнении), так и каждым из регистров захвата/сравнения.

Таймер А имеет следующие особенности:

- асинхронный 16-битный таймер/счётчик, имеющий четыре режима работы;
- возможность выбора и конфигурирования источника тактового сигнала;
- два или три конфигурируемых регистра захвата/сравнения;
- конфигурируемые выходы с возможностью генерации ШИМ-сигналов;
- асинхронное защёлкивание входных и выходных сигналов;
- регистр вектора прерывания для быстрого декодирования всех прерываний таймера.

Блок-схема Таймера А приведена на **Рис. 12.1**.

Примечание. Использование термина «счёт»
В этой главе активно используется термин «счёт». Использование данного термина означает, что для совершения действия заданное значение должно быть достигнуто таймером в процессе счёта. Если это значение просто записать в регистр счётчика таймера, то соответствующее действие выполнено не будет.

12.2. Функционирование Таймера А

Конфигурирование модуля Таймера А осуществляется пользовательской программой. Настройка Таймера А и его функционирование рассматриваются в следующих подразделах.

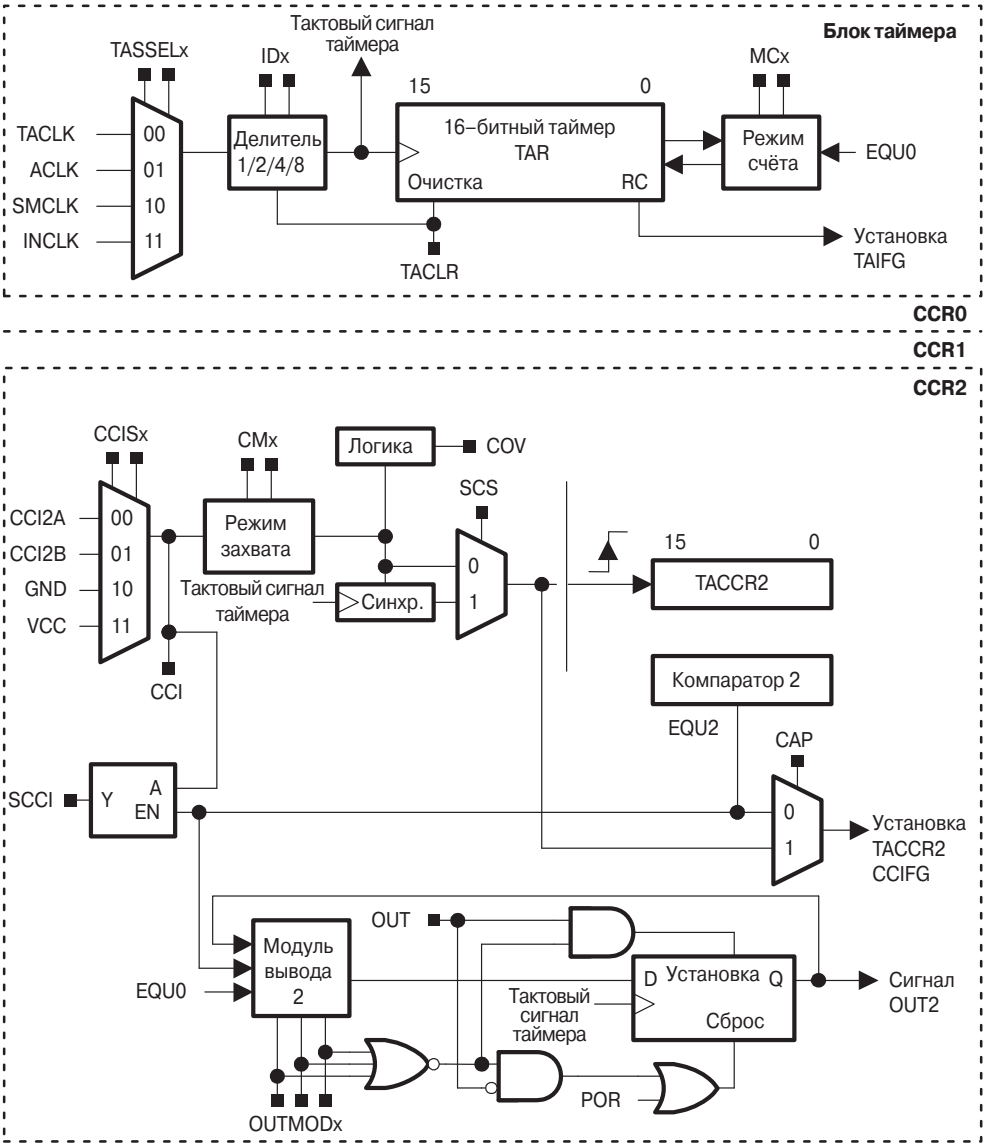


Рис. 12.1. Блок-схема Таймера A.

12.2.1. 16-битный таймер/счётчик

Регистр 16-битного таймера/счётчика TAR инкрементируется или декрементируется (в зависимости от режима работы) по нарастающему фронту импульсов тактового сигнала. Регистр TAR доступен как для чтения, так и для записи. Кроме того, при переполнении этого регистра таймер может генерировать прерывание.

Регистр TAR может быть очищен установкой бита TACLR. При установке бита TACLR также очищается делитель тактового сигнала и сбрасывается признак направления для режима реверсивного счёта.

Примечание. Изменение регистров Таймера A
Рекомендуется останавливать таймер перед изменением режима работы (это не относится к операциям с битами разрешения прерываний, флагами прерываний и битом TACLR), чтобы предотвратить его некорректное функционирование.
Если тактовый сигнал таймера не синхронен с тактовым сигналом ЦПУ, то любые операции чтения регистра TAR должны выполняться при остановленном таймере, в противном случае результат чтения будет непредсказуемым. В качестве альтернативного варианта можно выполнить несколько операций чтения регистра при работающем таймере, а затем определить корректный результат, используя мажорирование. Любые операции записи в регистр TAR вступают в силу немедленно.

Тактовый сигнал и делитель таймера

Для тактирования таймера могут использоваться системные тактовые сигналы ACLK и SMCLK или же внешние сигналы TACLK и INCLK. Источник тактового сигнала задаётся битами TASSELx регистра TACTL. Выбранный сигнал поступает на таймер через делитель, коэффициент деления которого (1, 2, 4 или 8) определяется битами IDx регистра TACTL. При установке бита TACLR делитель таймера очищается.

12.2.2. Запуск таймера

Запуск или перезапуск таймера может быть выполнен двумя способами:

- таймер осуществляет счёт, если MCx > 0 и источник тактового сигнала таймера активен;
- в режиме прямого или реверсивного счёта таймер может быть остановлен записью нуля в регистр TACCR0. Повторный запуск таймера может быть выполнен записью в регистр TACCR0 ненулевого значения. В этом случае таймер начнёт считать в прямом направлении, начиная с нуля.

12.2.3. Управление режимом работы таймера

Таймер имеет четыре режима работы, которые перечисленны в **Табл. 12.1**: останов, прямого счёта, непрерывного счёта и реверсивного счёта. Выбор рабочего режима осуществляется битами MCx.

Таблица 12.1. **Режимы работы таймера**

MCx	Режим	Описание
00	Останов	Таймер остановлен
01	Прямого счёта	Таймер циклически считает от нуля до значения, записанного в регистре TACCR0
10	Непрерывного счёта	Таймер циклически считает от нулевого значения до 0FFFFh
11	Реверсивного счёта	Таймер циклически считает от нуля до значения, записанного в регистре TACCR0, а затем в обратном направлении до нуля

Режим прямого счёта

Режим прямого счёта используется в том случае, если период таймера должен быть отличным от 0FFFFh. В этом режиме таймер циклически считает в прямом направлении до тех пор, пока его значение не станет равным содержимому регистра сравнения TACCR0, определяющего период счёта, как показано на **Рис. 12.2**. Величина периода в тактах равна TACCR0 + 1. Когда значение таймера достигает величины, записанной в регистре TACCR0, таймер сбрасывается и счёт начинается с нуля. Если в момент выбора режима прямого счёта значение таймера будет больше содержимого TACCR0, таймер сразу же начнёт счёт с нулевого значения.

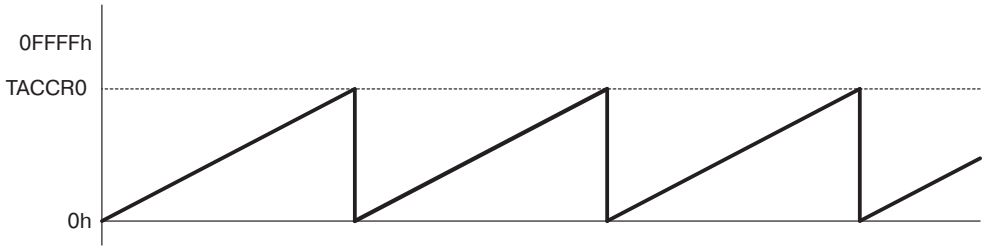


Рис. 12.2. Режим прямого счёта.

Флаг прерывания CCIFG, соответствующий регистру TACCR0, устанавливается при достижении таймером в процессе счёта значения, записанного в этом регистре. Флаг прерывания TAIFG устанавливается при счёте таймера от значения, содержащегося в регистре TACCR0, до нуля (**Рис. 12.3**).

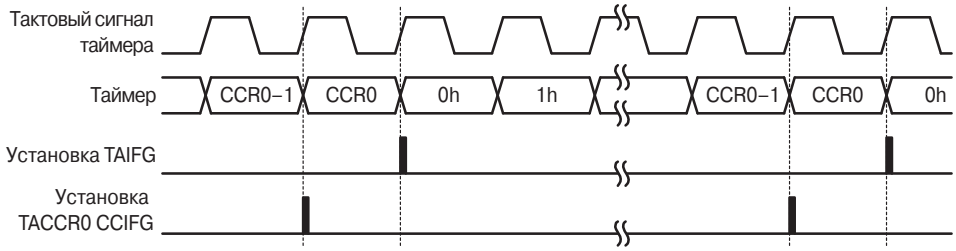


Рис. 12.3. Установка флагов прерываний в режиме прямого счёта.

Изменение регистра периода TACCR0

Если при изменении содержимого регистра TACCR0 во время работы таймера новый период окажется больше или равен предыдущему периоду или больше текущего значения таймера, то таймер продолжит счёт до достижения нового значения периода. Если новый период окажется меньше текущего значения таймера, то таймер начнёт счёт с нуля, однако перед обнулением счётчика может возникнуть дополнительный отсчёт.

Режим непрерывного счёта

В режиме непрерывного счёта таймер циклически считает в прямом направлении от нуля до значения 0FFFFh, как показано на **Рис. 12.4**. Регистр TACCR0 используется так же, как и остальные регистры захвата/сравнения.

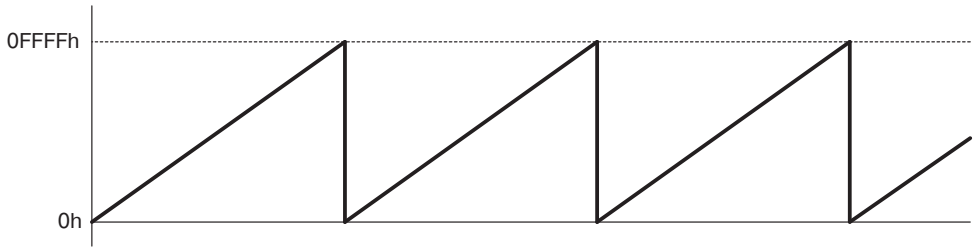


Рис. 12.4. Режим непрерывного счёта.

Флаг прерывания TAIFG устанавливается при счёте таймера от значения 0FFFFh до нуля (**Рис. 12.5**).

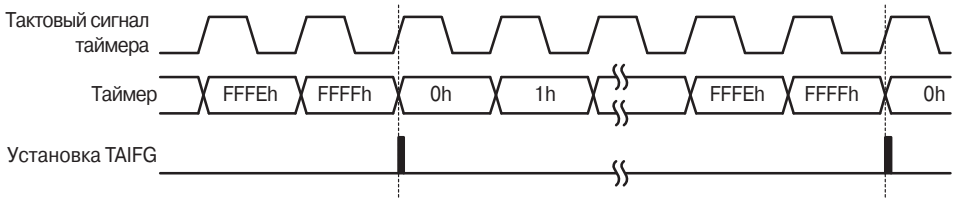


Рис. 12.5. Установка флага прерывания в режиме непрерывного счёта.

Использование режима непрерывного счёта

Режим непрерывного счёта может использоваться для формирования независимых временных интервалов и сигналов разных частот. Каждый раз при завершении очередного интервала генерируется прерывание. В процедуре обработки прерывания длительность следующего интервала прибавляется к содержимому регистра TACCRx. На **Рис. 12.6** показано формирование двух временных интервалов длительностью t_0 и t_1 (эти значения прибавляются к содержимому регистров захвата/сравнения). В данном случае формирование временных интервалов осуществляется аппаратно, и задержка обработки прерывания не сказывается на их длительности.

Временные интервалы можно формировать и в других режимах, использующих регистр TACCR0 в качестве регистра периода. Однако при этом необходимо предусмотреть обработку ситуации, когда сумма старого содержимого TACCRx и значения нового периода оказывается больше содержимого регистра периода TACCR0. В таком случае для получения корректного результата из получившейся суммы необходимо вычесть значение TACCR0 + 1.

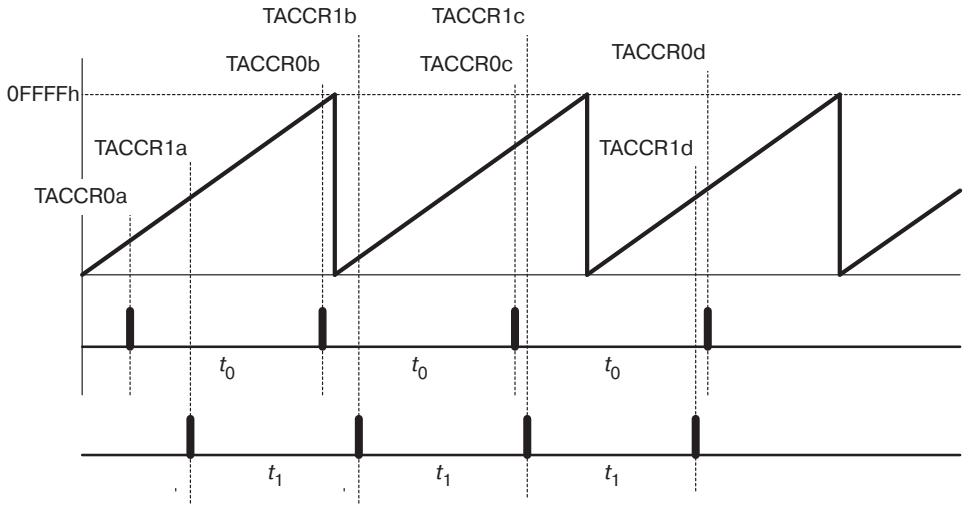


Рис. 12.6. Формирование временных интервалов в режиме непрерывного счёта.

Режим реверсивного счёта

Режим реверсивного счёта используется в случае, если период таймера должен быть отличным от 0FFFFh и если необходимо генерировать симметричные импульсы. В данном режиме таймер циклически считает в прямом направлении до тех пор, пока его значение не станет равным содержимому регистра сравнения TACCR0, а затем в обратном направлении до нуля, как показано на **Рис. 12.7**. При этом период равен удвоенному значению TACCR0.

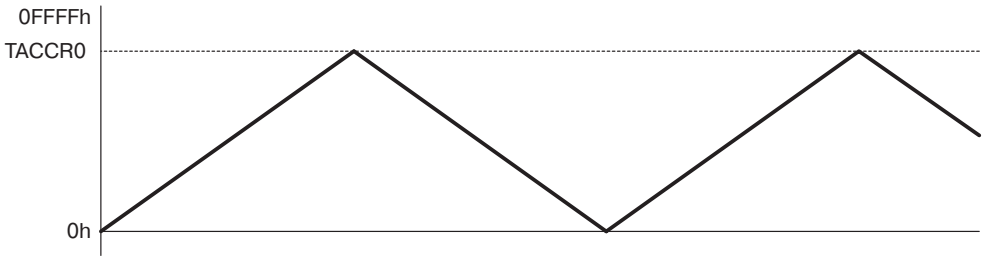


Рис. 12.7. Режим реверсивного счёта.

Направление счёта фиксируется, что позволяет после останова таймера и последующего его запуска продолжать счёт в том же направлении, что и до останова. Если этого не требуется, то для сброса признака направления необходимо записать 1 в бит TACLR. Установка бита TACLR также вызывает очистку регистра TAR и делителя тактового сигнала таймера.

В режиме реверсивного счёта флаг прерываний CCIFG для регистра TACCR0 и флаг TAIFG устанавливаются только один раз за период, причём интервал между установкой этих флагов составляет 1/2 периода таймера. Флаг CCIFG устанав-

ливается при счёте таймера от значения (TACCR0 – 1) до TACCR0, а флаг TAIFG — при счёте с 0001h до 0000h (**Рис. 12.8**).

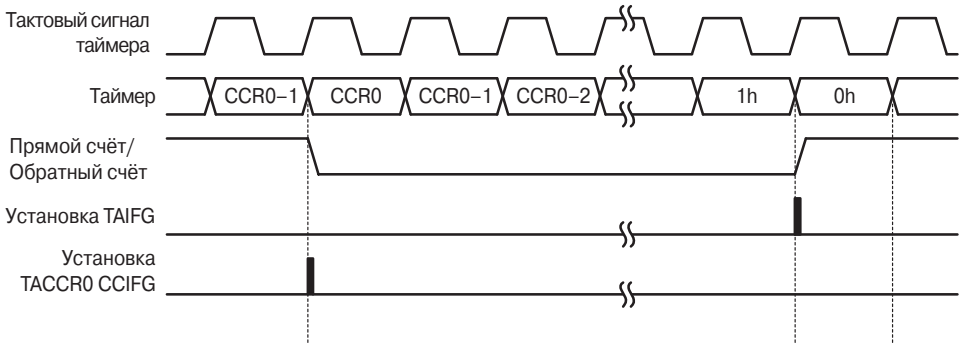


Рис. 12.8. Установка флагов прерываний в режиме реверсивного счёта.

Изменение регистра периода TACCR0

При изменении содержимого регистра TACCR0 во время счёта таймера в обратном направлении, процесс счёта не прерывается. Новое содержимое TACCR0 защёлкивается в TACL0 сразу же, однако формирование периода с новой длительностью начнётся только после достижения счётчиком нулевого значения.

Если таймер считает в прямом направлении и новое значение периода больше или равно старому значению или же больше текущего значения таймера, то смена направления счёта произойдёт после достижения таймером нового значения периода. Если новый период меньше текущего значения таймера, то таймер сразу же начинает считать в обратном направлении, однако перед сменой направления может возникнуть один дополнительный отсчёт.

Использование реверсивного режима счёта

Режим реверсивного счёта может применяться в случаях, когда требуется наличие «мёртвых зон» между выходными сигналами (см. подраздел 12.2.5 «Модуль вывода»). Так, на выводах, используемых для управления Н-мостом, не допускается одновременное появление ВЫСОКОГО уровня во избежание перегрузки элементов моста.

В примере, изображённом на **Рис. 12.9**, величина t_{dead} определяется как

$$t_{dead} = t_{timer} \times (\text{TACCR1} - \text{TACCR2}),$$

где t_{dead} — интервал времени, в течение которого оба вывода должны быть неактивными («мёртвое время»);

t_{timer} — период тактового сигнала таймера;

TACCRx — содержимое регистра захвата/сравнения x.

Регистры TACCRx не буферизованы — их содержимое обновляется в момент операции записи. Соответственно требуемая длительность «мёртвого» времени не будет выдерживаться автоматически.

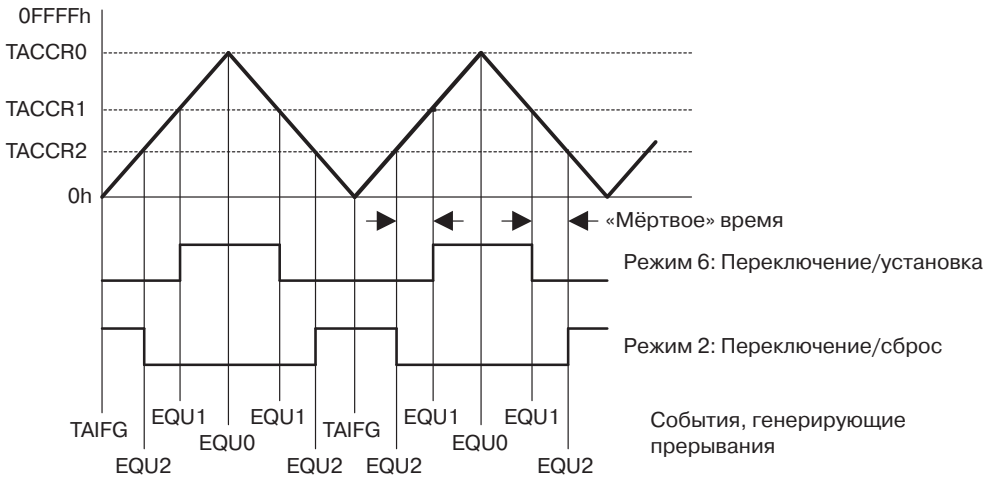


Рис. 12.9. Модуль вывода в режиме реверсивного счёта.

12.2.4. Блоки захвата/сравнения

В Таймере A в зависимости от модели микроконтроллера имеется два или три идентичных блока захвата/сравнения. Любой из блоков может использоваться для захвата значения таймера и формирования временных интервалов.

Режим захвата

Режим захвата включается при CAP = 1. Этот режим используется для регистрации временных событий. Он может быть использован для вычисления скорости или измерения временных параметров. Входы захвата CCIxA и CCIxB подключены к выводам микроконтроллера или же к его внутренним шинам. Выбор конкретного входа захвата осуществляется битами CCISx. Биты CMx определяют активный фронт сигнала захвата (нарастающий, спадающий или оба), по которому выполняется операция захвата. При выполнении этой операции:

- значение таймера копируется в соответствующий регистр TACCRx;
- устанавливается флаг прерывания CCIFG.

Уровень входного сигнала можно в любой момент времени определить по состоянию бита CCI. На входы CCIxA и CCIxB могут поступать различные сигналы, в зависимости от модели. Для получения более подробной информации обратитесь к документации на конкретный микроконтроллер.

Сигнал захвата может быть асинхронным по отношению к тактовому сигналу таймера, что приведёт к возникновению «гонок». При установленном бите SCS операция захвата синхронизируется с тактовым сигналом таймера. Рекомендуется всегда устанавливать этот бит. Синхронизация сигнала захвата показана на **Рис. 12.10**.

Для каждого регистра захвата/сравнения предусмотрена схема переполнения, служащая для индикации выполнения операции захвата до считывания результата предыдущей операции. При возникновении такой ситуации устанавливается бит COV, как показано на **Рис. 12.11**. Бит COV должен сбрасываться программно.

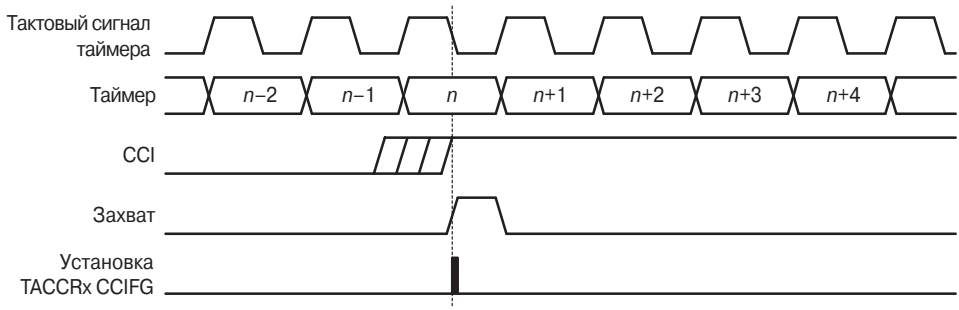


Рис. 12.10. Сигнал захвата при SCS = 1.

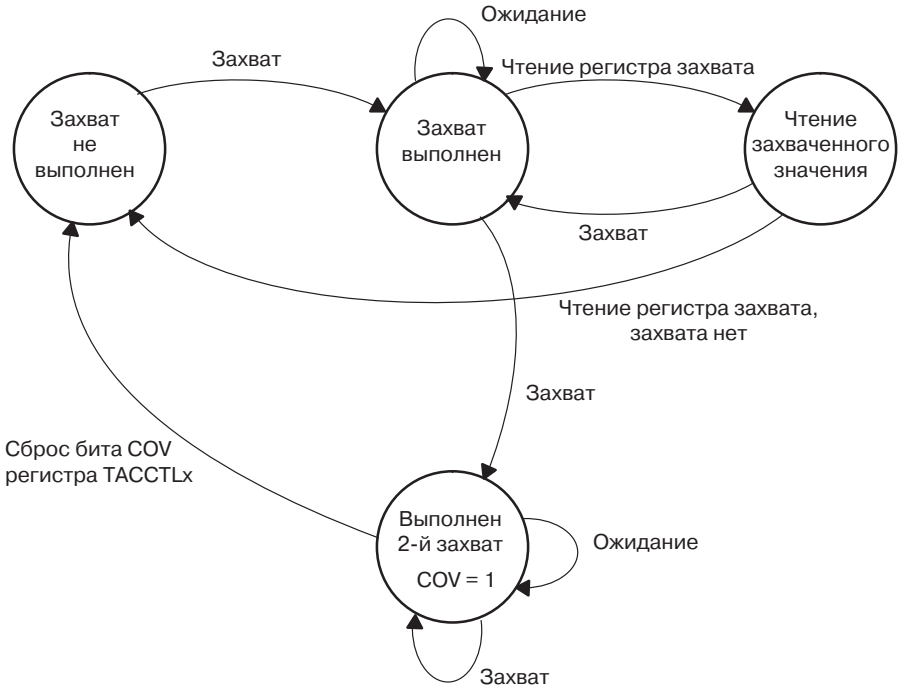


Рис. 12.11. Цикл захвата.

Захват, инициируемый программно

Захват может быть инициирован программно. Для выполнения захвата по обоим фронтам можно установить биты CMx = 11. После этого в программе достаточно установить бит CCIS1 и изменить состояние бита CCIS0 для формирования фронта сигнала захвата. Захват будет инициироваться при каждом изменении бита CCIS0:

```
MOV   #CAP+SCS+CCIS1+CM_3,&TACCTLx      ; Настроим TACCTLx
XOR   #CCIS0,&TACCTLx                   ; TACCTLx = TAR
```

Режим сравнения

Режим сравнения включается при CAP = 0. Этот режим используется для генерации сигналов с широтно-импульсной модуляцией или для формирования прерываний с заданной периодичностью. При достижении таймером в процессе счёта значения, записанного в регистре TACCRx:

- устанавливается флаг прерывания CCIFG;
- формируется внутренний сигнал EQUx = 1;
- сигнал EQUx воздействует на выход в соответствии с режимом работы модуля вывода;
- значение входного сигнала CCI защёлкивается в бите SCCI.

12.2.5. Модуль вывода

Каждый блок захвата/сравнения содержит модуль вывода. Этот модуль используется для генерации различных сигналов, в том числе сигналов с ШИМ. Каждый модуль вывода имеет восемь режимов работы, которые используются для формирования различных сигналов в соответствии с внутренними сигналами EQU0 и EQUx.

Режимы модуля вывода

Режимы работы модуля вывода определяются битами OUTMODx и перечислены в **Табл. 12.2**. Сигнал OUTx изменяется по нарастающему фронту тактового сигнала таймера во всех режимах, кроме режима 0. Режимы 2, 3, 6 и 7 не применимы к 0-му модулю вывода, поскольку в этом случае EQUx = EQU0.

Таблица 12.2. **Режимы работы модуля вывода**

OUTMODx	Режим	Описание
000	Вывод	Выходной сигнал OUTx определяется состоянием бита OUTx. Выходной сигнал изменяется сразу же после изменения бита OUTx
001	Установка	При достижении таймером в процессе счёта значения, записанного в регистре TACCRx, выход устанавливается. Это состояние сохраняется до тех пор, пока таймер не будет сброшен или пока не будет задан новый режим работы модуля вывода с последующим воздействием на выход
010	Переключение/ сброс	При достижении таймером в процессе счёта значения, записанного в регистре TACCRx, выход переключается. Выход сбрасывается при достижении таймером значения, записанного в регистре TACCR0
011	Установка/ сброс	При достижении таймером в процессе счёта значения, записанного в регистре TACCRx, выход устанавливается. Выход сбрасывается при достижении таймером значения, записанного в регистре TACCR0
100	Переключение	Выход переключается при достижении таймером в процессе *счёта* значения, записанного в регистре TACCRx. Период выходного сигнала равен удвоенному периоду таймера
101	Сброс	При достижении таймером в процессе счёта значения, записанного в регистре TACCRx, выход сбрасывается. Это состояние сохраняется до тех пор, пока не будет задан новый режим работы модуля вывода с последующим воздействием на выход
110	Переключение/ установка	При достижении таймером в процессе счёта значения, записанного в регистре TACCRx, выход переключается. Выход устанавливается при достижении таймером значения, записанного в регистре TACCR0
111	Сброс/ установка	При достижении таймером в процессе счёта значения, записанного в регистре TACCRx, выход сбрасывается. Выход устанавливается при достижении таймером значения, записанного в регистре TACCR0

Использование модуля вывода — таймер в режиме прямого счёта

Сигнал OUTx изменяется при достижении таймером в процессе счёта значения, записанного в регистре TACCRx, а также при переходе таймера от значения, записанного в регистре TACCR0, к нулевому значению. На **Рис. 12.12** показано формирование выходного сигнала в различных режимах работы модуля вывода с использованием регистров захвата/сравнения TACCR0 и TACCR1.

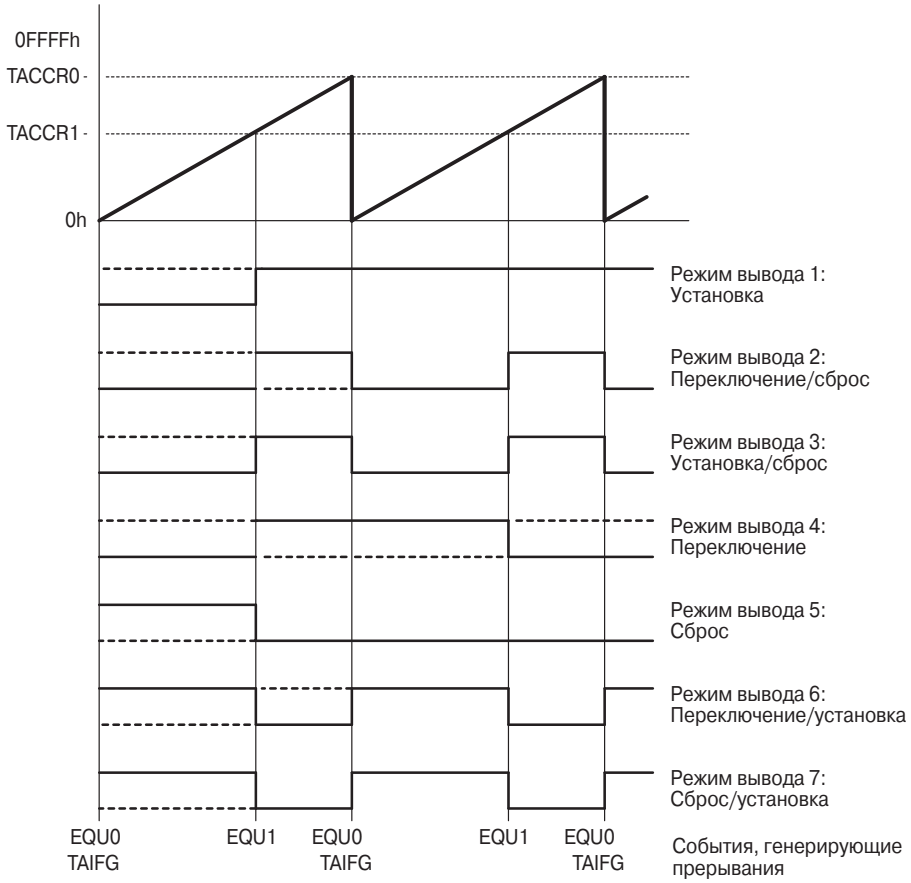


Рис. 12.12. Формирование выходного сигнала — таймер в режиме прямого счёта.

Использование модуля вывода — таймер в режиме непрерывного счёта

Сигнал OUTx изменяется при достижении таймером в процессе счёта значений, записанных в регистрах TACCRx и TACCR0. На **Рис. 12.13** показано формирование выходного сигнала в различных режимах работы модуля вывода с использованием регистров захвата/сравнения TACCR0 и TACCR1.

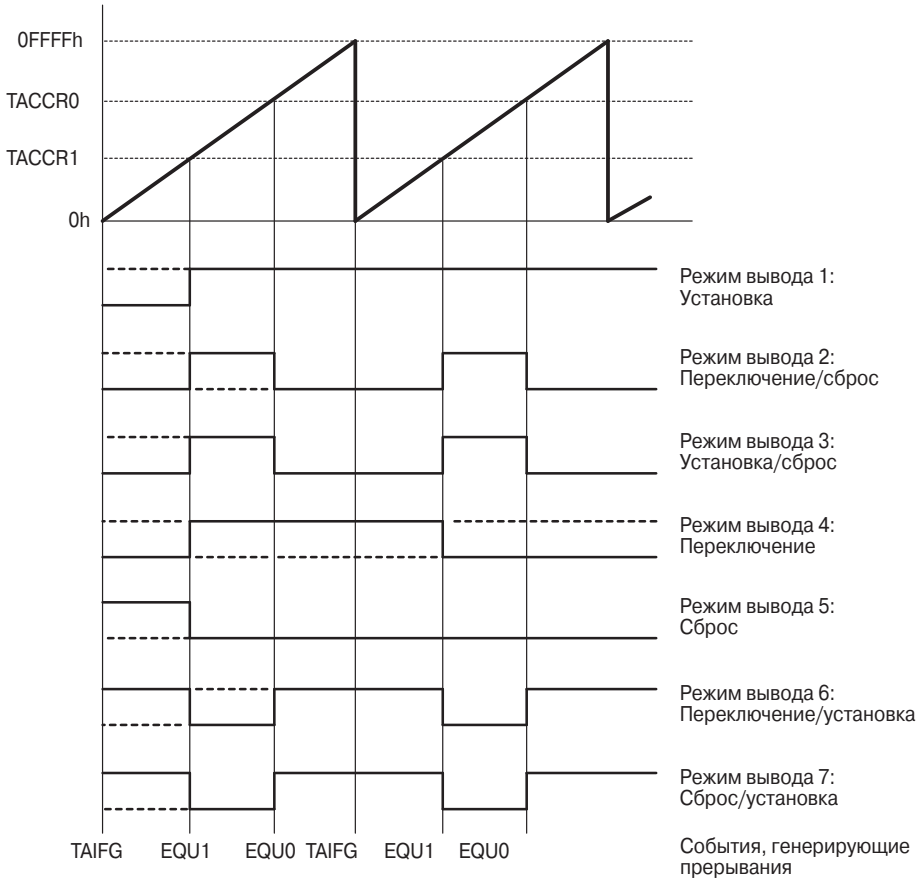


Рис. 12.13. Формирование выходного сигнала — таймер в режиме непрерывного счёта.

Использование модуля вывода — таймер в режиме реверсивного счёта

Сигнал OUTx изменяется при достижении таймером в процессе счёта значения, записанного в регистре TACCRx (вне зависимости от направления счёта), а также при достижении значения, записанного в регистре TACCR0. На **Рис. 12.14** показано формирование выходного сигнала в различных режимах работы модуля вывода с использованием регистров захвата/сравнения TACCR0 и TACCR2.

Примечание. Переключение между режимами модуля вывода
При переходе от одного режима работы модуля вывода к другому, один из битов OUTMODx в процессе переключения обязательно должен оставаться в установленном состоянии, за исключением переключения в режим 0. В противном случае из-за ошибочного декодирования элементом ИЛИ-НЕ режима 0 модуля возможно появление на выходе паразитных импульсов. Безопасный метод переключения режимов модуля заключается в использовании в качестве промежуточного состояния режима 7.

```
BIS   #OUTMOD_7,&TACCTLx  ; Устанавливаем режим 7
BIC   #OUTMODx,&TACCTLx   ; Сбрасываем требуемые биты
```

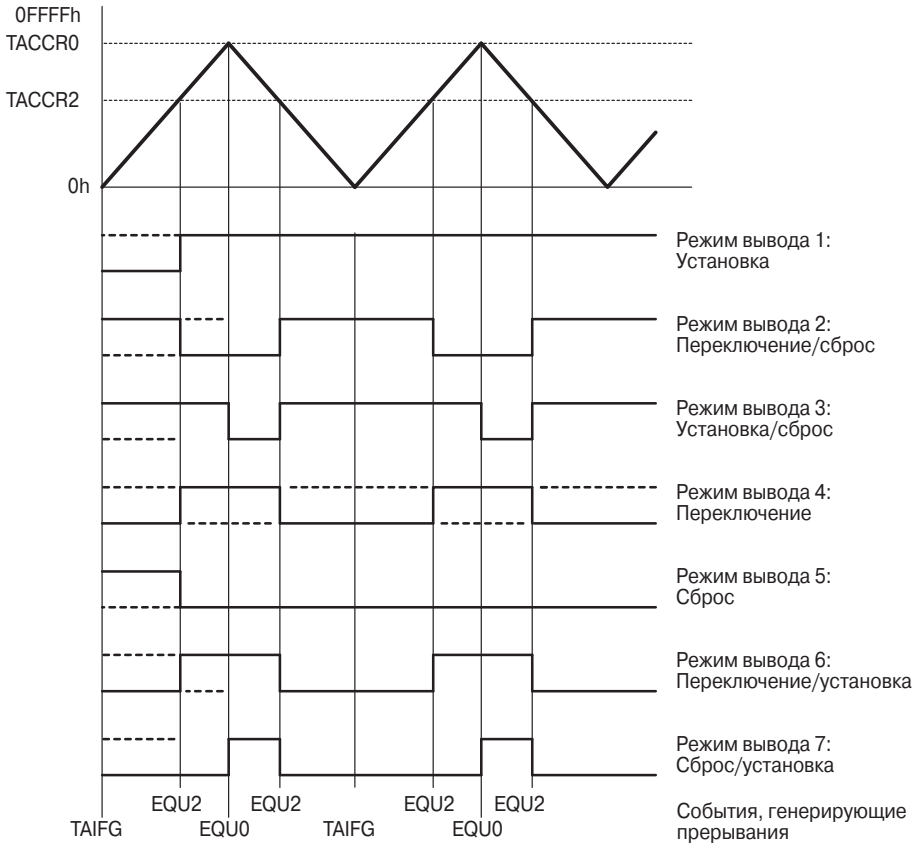


Рис. 12.14. Формирование выходного сигнала — таймер в режиме реверсивного счёта.

12.2.6. Прерывания Таймера A

С модулем 16-битного Таймера A связаны два вектора прерывания:

- вектор прерывания TACCR0 для бита CCIFG, соответствующего регистру TACCR0;
- вектор прерывания TAIV для всех остальных флагов CCIFG и флага TAIFG.

В режиме захвата любой из флагов CCIFG устанавливается в момент сохранения значения таймера в соответствующем регистре TACCRx. В режиме сравнения любой из флагов CCIFG устанавливается при достижении таймером в процессе счёта значения, записанного в соответствующем регистре TACCRx. Кроме того, любой флаг CCIFG можно установить или сбросить программно. При установке флага CCIFG, если установлен соответствующий бит CCIE и бит общего разрешения прерываний GIE, генерируется запрос прерывания.

Прерывание TACCR0

Флаг прерывания CCIFG для регистра TACCR0 соответствует прерыванию Таймера A с наивысшим приоритетом и имеет отдельный вектор прерывания, как

показано на **Рис. 12.15**. Флаг CCIFG для регистра TACCR0 автоматически сбрасывается при обслуживании запроса прерывания TACCR0.

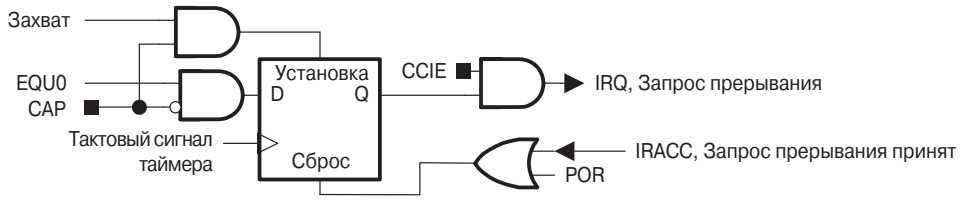


Рис. 12.15. Флаг прерывания захвата/сравнения TACCR0.

Генератор вектора прерывания TAIV

Флаги CCIFG для регистров TACCR1, TACCR2 и флаг TAIFG имеют различные приоритеты и служат источником единственного вектора прерывания. Для определения, какой из флагов вызвал генерацию прерывания, используется регистр вектора прерывания TAIV.

Разрешённое прерывание с наивысшим приоритетом формирует в регистре TAIV число (см. описание регистра). Это число можно оценить или же просто прибавить к счётчику команд для автоматического перехода к соответствующей секции программы. Запрещённые прерывания таймера не влияют на содержимое регистра TAIV.

При любом обращении к регистру TAIV как для чтения, так и для записи флаг прерывания с наивысшим приоритетом, ожидающего обработки, автоматически сбрасывается. Если имеется ещё один установленный флаг, то сразу же после обработки текущего прерывания будет сгенерировано новое прерывание. Например, если на момент обращения к регистру TAIV в процедуре обработки прерывания были установлены флаги CCIFG для регистров TACCR1 и TACCR2, то автоматически будет сброшен флаг CCIFG для регистра TACCR1. После выполнения команды `RETI` процедуры обработки прерывания, флаг CCIFG для регистра TACCR2 вызовет генерацию нового прерывания.

Пример процедуры обработки прерывания TAIV

В следующем примере показано рекомендуемое использование регистра TAIV и приведена информация о накладных расходах на обработку прерывания. Содержимое регистра TAIV прибавляется к счётчику команд для автоматического перехода к соответствующей процедуре.

Значения, расположенные по правому краю, показывают число тактов ЦПУ, требуемое для выполнения каждой команды. Накладные расходы на обработку различных источников прерывания включают в себя задержку обслуживания прерывания и время, требуемое для возврата из прерывания, но не время, необходимое для выполнения собственно задачи. Задержки обработки прерывания имеют следующие значения:

- Блок захвата/сравнения CCR0 — 11 тактов
- Блоки захвата/сравнения CCR1 и CCR2 — 16 тактов
- Переполнение таймера TAIFG — 14 тактов

```
; Обработчик прерывания для TACCR0 CCIFG.                          Тактов
CCIFG_0_HND
;        ...                ; Задержка обработки прерывания            6
        RETI                                                           5
; Обработчик прерывания для TAIFG, TACCR1 и TACCR2 CCIFG.
TA_HND  ...                 ; Задержка обработки прерывания            6
        ADD    &TAIV,PC     ; Прибавляем смещение в таблице переходов  3
        RETI                ; Вектор 0: Нет прерывания                 5
        JMP    CCIFG_1_HND  ; Вектор 2: TACCR1                         2
        JMP    CCIFG_2_HND  ; Вектор 4: TACCR2                         2
        RETI                ; Вектор 6: Зарезервировано                5
        RETI                ; Вектор 8: Зарезервировано                5

TAIFG_HND                   ; Вектор 10: Флаг TAIFG
        ...                 ; Здесь начало задачи
        RETI                                                           5

CCIFG_2_HND                 ; Вектор 4: TACCR2
        ...                 ; Здесь начало задачи
        RETI                ; Возвращаемся в основную программу        5

CCIFG_1_HND                 ; Вектор 2: TACCR1
        ...                 ; Здесь начало задачи
        RETI                ; Возвращаемся в основную программу        5
```

12.3. Регистры Таймера A

Список регистров Таймера A приведён в **Табл. 12.3**.

Таблица 12.3. **Регистры Таймера A**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления Таймера A	TACTL	Чтение/запись	0160h	Сбрасывается после POR
Регистр счётчика Таймера A	TAR	Чтение/запись	0170h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 0 Таймера A	TACCTL0	Чтение/запись	0162h	Сбрасывается после POR
Регистр захвата/сравнения 0 Таймера A	TACCR0	Чтение/запись	0172h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 1 Таймера A	TACCTL1	Чтение/запись	0164h	Сбрасывается после POR
Регистр захвата/сравнения 1 Таймера A	TACCR1	Чтение/запись	0174h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 2 Таймера A	TACCTL2*	Чтение/запись	0166h	Сбрасывается после POR
Регистр захвата/сравнения 2 Таймера A	TACCR2*	Чтение/запись	0176h	Сбрасывается после POR
Регистр вектора прерывания Таймера A	TAIV	Только чтение	012Eh	Сбрасывается после POR
* Отсутствует в моделях MSP430x20xx				

TACTL, регистр управления Таймера A

15	14	13	12	11	10	9	8
Unused						**TASSELx**	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
IDx		**MCx**		**Unused**	**TACLR**	**TAIE**	**TAIFG**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	w–(0)	rw–(0)	rw–(0)

Unused Биты 15...10 Не используются.

TASSELx Биты 9...8 Выбор источника тактового сигнала.
- 00 TACLK
- 01 ACLK
- 10 SMCLK
- 11 INCLK

IDx Биты 7...6 Коэффициент деления входного делителя. Эти биты определяют коэффициент деления задающего тактового сигнала.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

MCx Биты 5...4 Управление режимом таймера. Установка MCx = 00h при неиспользуемом таймере уменьшает общее потребление микроконтроллера.
- 00 Останов: таймер остановлен.
- 01 Прямой счёт: таймер считает от 0000h до TACCR0.
- 10 Непрерывный счёт: таймер считает от 0000h до 0FFFFh.
- 11 Реверсивный счёт: таймер считает от 0000h до TACCR0, а потом в обратном направлении до 0000h.

Unused Бит 3 Не используется.

TACLR Бит 2 Очистка таймера. Установка этого бита обнуляет регистр TAR, делитель и сбрасывает признак направления счёта. Бит TACLR автоматически сбрасывается; всегда читается как 0.

TAIE Бит 1 Разрешение прерывания таймера. Этот бит разрешает генерацию запроса прерывания TAIFG.
- 0 Прерывание запрещено
- 1 Прерывание разрешено

TAIFG Бит 0 Флаг прерывания таймера.
- 0 Прерывания не было
- 1 Прерывание было

TAR, регистр счётчика Таймера A

15	14	13	12	11	10	9	8
TARx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)
7	6	5	4	3	2	1	0
TARx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

TARx Биты 15-0 Регистр счётчика Таймера A. В этом регистре содержится текущее значение Таймера A

TACCRx, регистр захвата/сравнения x Таймера A

15	14	13	12	11	10	9	8
TACCRx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)
7	6	5	4	3	2	1	0
TACCRx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

TACCRx Биты 15...0 Регистр захвата/сравнения Таймера A.
Режим сравнения: регистр TACCRx содержит число, которое сравнивается с содержимым регистра счётчика TAR.
Режим захвата: содержимое регистра TAR копируется в регистр TACCRx в момент выполнения операции захвата.

TACCTLx, регистр управления блока захвата/сравнения

15	14	13	12	11	10	9	8
CMx		CCISx		SCS	SCCI	Unused	CAP
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	r0	rw–(0)
7	6	5	4	3	2	1	0
OUTMODx			CCIE	CCI	OUT	COV	CCIFG
rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	rw–(0)	rw–(0)	rw–(0)

CMx Биты 15...14 Режим захвата.
- 00 Нет захвата
- 01 Захват по нарастающему фронту
- 10 Захват по спадающему фронту
- 11 Захват по обоим фронтам

CCISx Биты 13...12 Выбор входа захвата/сравнения. Эти биты определяют входной сигнал блока захвата/сравнения. Соответствие сигналов каждому из входов приводится в документации на конкретные модели.
- 00 CCIxA
- 01 CCIxB
- 10 GND
- 11 V_{CC}

SCS	Бит 11	Синхронизация захвата. Этот бит используется для синхронизации сигнала захвата с тактовым сигналом таймера. 0 Асинхронный захват 1 Синхронный захват
SCCI	Бит 10	Синхронизованный вход захвата/сравнения. Входной сигнал блока захвата/сравнения фиксируется по сигналу EQUx и может быть считан при помощи этого бита.
Unused	Бит 9	Не используется. Только для чтения. Всегда читается как 0.
CAP	Бит 8	Режим работы блока захвата/сравнения. 0 Режим захвата 1 Режим сравнения
OUTMODx	Биты 7...5	Режим работы модуля вывода. Для регистра TACCR0 использование режимов 2, 3, 6 и 7 не имеет смысла, поскольку в этом случае EQUx = EQU0. 000 Состояние бита OUT 001 Установка 010 Переключение/сброс 011 Установка/сброс 100 Переключение 101 Сброс 110 Переключение/установка 111 Сброс/установка
CCIE	Бит 4	Разрешение прерывания захвата/сравнения. Этот бит разрешает генерацию запроса прерывания при установке соответствующего флага CCIFG. 0 Прерывание запрещено 1 Прерывание разрешено
CCI	Бит 3	Вход захвата/сравнения. Посредством этого бита можно определить значение входного сигнала блока захвата/сравнения.
OUT	Бит 2	Состояние выхода. При работе модуля вывода в режиме 0 этот бит напрямую управляет состоянием выхода. 0 На выходе НИЗКИЙ уровень 1 На выходе ВЫСОКИЙ уровень
COV	Бит 1	Переполнение захвата. Этот бит индицирует переполнение при операциях захвата. Бит COV должен сбрасываться программно. 0 Не было переполнения при захвате 1 Было переполнение при захвате
CCIFG	Бит 0	Флаг прерывания захвата/сравнения 0 Прерывания не было 1 Прерывание было

TAIV, регистр вектора прерывания

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**TAIVx**			**0**
r0	r0	r0	r0	r–(0)	r–(0)	r–(0)	r0

TAIVx Биты 15...0 Значение вектора прерывания таймера

Содержимое TAIV	Источник прерывания	Флаг прерывания	Приоритет прерывания
00h	Не было прерываний	—	
02h	Захват/сравнение 1	TACCR1 CCIFG	Высший
04h	Захват/сравнение 2*	TACCR2 CCIFG	
06h	Зарезервировано	—	
08h	Зарезервировано	—	
0Ah	Переполнение таймера	TAIFG	
0Ch	Зарезервировано	—	
0Eh	Зарезервировано	—	Низший

* Не реализовано в моделях MSP430x20xx.

ГЛАВА 13

ТАЙМЕР B

Таймер B представляет собой 16-битный таймер/счётчик, имеющий несколько регистров захвата/сравнения. В этой главе описывается работа Таймера B, реализованного в микроконтроллерах семейства MSP430x2xx.

13.1. Введение

Таймер B представляет собой 16-битный таймер/счётчик с тремя или семью регистрами захвата/сравнения. Этот таймер может обеспечить несколько каналов захвата/сравнения, генерации сигналов с ШИМ и формирования временных интервалов. Кроме того, Таймер B имеет развитую поддержку прерываний. Прерывания могут генерироваться как регистром счётчика таймера (при его переполнении), так и каждым из регистров захвата/сравнения.

Таймер B имеет следующие особенности:

- асинхронный 16-битный таймер/счётчик изменяемой разрядности, имеющий четыре режима работы;
- возможность выбора и конфигурирования источника тактового сигнала;
- три или семь конфигурируемых регистров захвата/сравнения;
- конфигурируемые выходы с возможностью генерации ШИМ-сигналов;
- регистры-защёлки сравнения с двойной буферизацией и синхронизуемой загрузкой;
- регистр вектора прерывания для быстрого декодирования всех прерываний таймера.

Блок-схема Таймера B приведена на **Рис. 13.1**.

Примечание. Использование термина «счёт»
В этой главе активно используется термин «счёт». Использование данного термина означает, что для совершения действия заданное значение должно быть достигнуто таймером в процессе счёта. Если это значение просто записать в регистр счётчика таймера, то соответствующее действие выполнено не будет.

13.1.1. Сходства и различия с Таймером A

Таймер B полностью идентичен Таймеру A за некоторыми исключениями:

- таймер B имеет программируемую разрядность, которая может быть равна 8, 10, 12 или 16 бит;

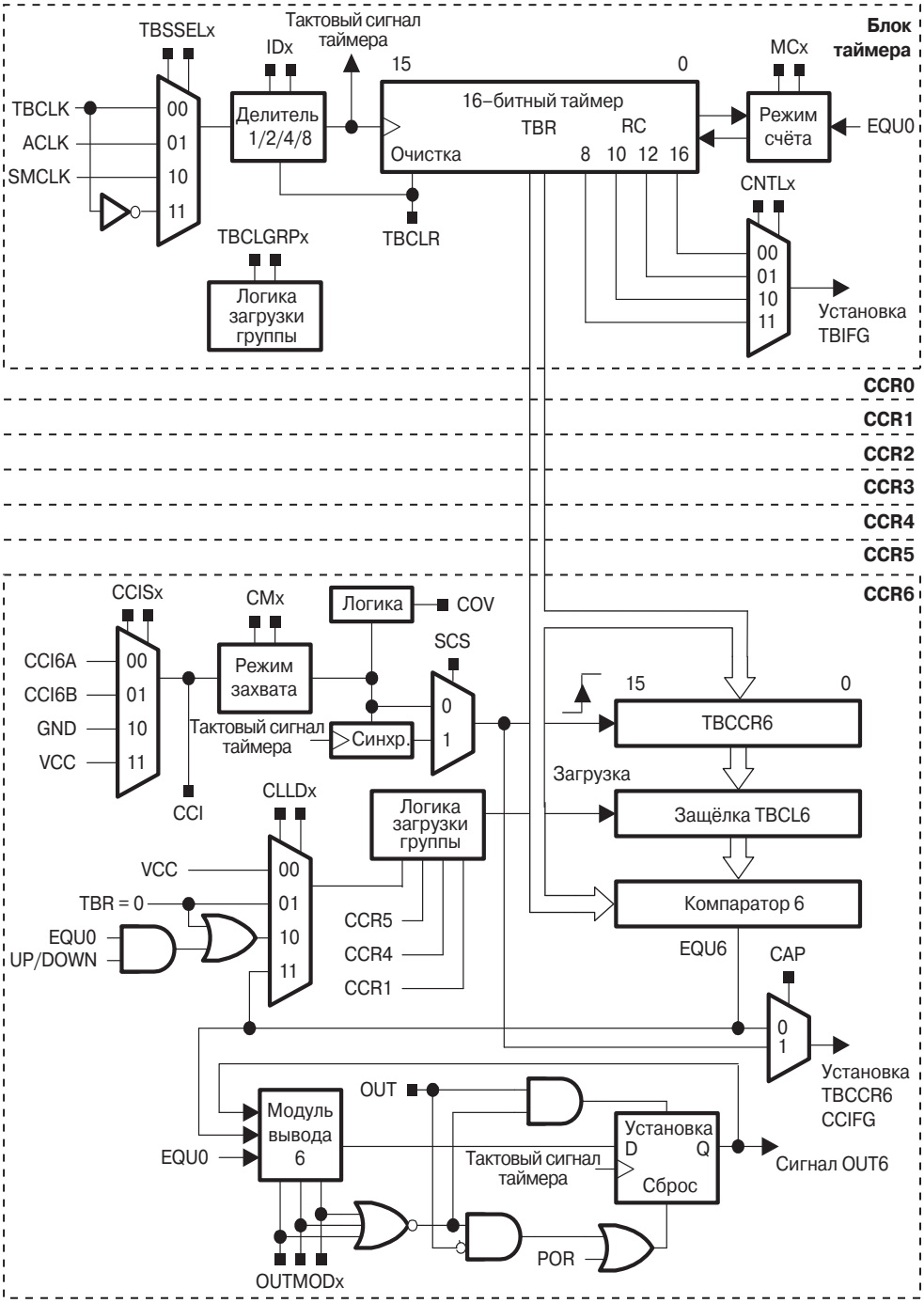


Рис. 13.1. Блок-схема Таймера B.

- регистры TBCCRx Таймера B имеют двойную буферизацию и могут объединяться друг с другом;
- все выходы Таймера B могут быть переведёны в высокоимпедансное состояние;
- функция бита SCCI в Таймере B не реализована.

13.2. Функционирование Таймера B

Конфигурирование модуля Таймера B осуществляется пользовательской программой. Настройка Таймера B и его функционирование рассматриваются в следующих подразделах.

13.2.1. 16-битный таймер/счётчик

Регистр 16-битного таймера/счётчика TBR инкрементируется или декрементируется (в зависимости от режима работы) по нарастающему фронту импульсов тактового сигнала. Регистр TBR доступен как для чтения, так и для записи. Кроме того, при переполнении этого регистра таймер может генерировать прерывание.

Регистр TBR может быть очищен установкой бита TBCLR. При установке бита TBCLR также очищается делитель тактового сигнала и сбрасывается признак направления для режима реверсивного счёта.

Примечание. Изменение регистров Таймера B
Рекомендуется останавливать таймер перед изменением режима работы (это не относится к операциям с битами разрешения прерываний, флагами прерываний и битом TBCLR), чтобы предотвратить его некорректное функционирование.
Если тактовый сигнал таймера не синхронен с тактовым сигналом ЦПУ, то любые операции чтения регистра TBR должны выполняться при остановленном таймере, в противном случае результат чтения будет непредсказуемым. В качестве альтернативного варианта можно выполнить несколько операций чтения регистра при работающем таймере, а затем определить корректный результат, используя мажорирование. Любые операции записи в регистр TBR вступают в силу немедленно.

Разрядность регистра TBR

Посредством битов CNTLx Таймер B может быть сконфигурирован как 8-, 10-, 12- или 16-битный таймер. В зависимости от разрядности таймера максимальное значение счётчика $TBR_{(max)}$ составляет 0FFh, 03FFh, 0FFFh и 0FFFFh соответственно. Данные, загружаемые в регистр TBR в 8-, 10- и 12-битном режимах, выравниваются вправо с заполнением незначащих битов нулями.

Тактовый сигнал и делитель таймера

Для тактирования таймера могут использоваться системные тактовые сигналы ACLK и SMCLK или же внешний сигнал (прямой или инвертированный), подаваемый на вывод TBCLK. Источник тактового сигнала задаётся битами TBSSELx регистра TBCTL. Выбранный сигнал поступает на таймер через делитель, коэффициент деления которого (1, 2, 4 или 8) определяется битами IDx регистра TBCTL. При установке бита TBCLR делитель таймера очищается.

13.2.2. Запуск таймера

Запуск или перезапуск таймера может быть выполнен двумя способами:

- таймер осуществляет счёт, если MCx > 0 и источник тактового сигнала таймера активен;
- в режиме прямого или реверсивного счёта таймер может быть остановлен записью нуля в защёлку TBCL0. Повторный запуск таймера может быть выполнен в результате загрузки в TBCL0 ненулевого значения. В этом случае таймер начнёт считать в прямом направлении, начиная с нуля.

13.2.3. Управление режимом работы таймера

Таймер имеет четыре режима работы, которые перечисленны в **Табл. 13.1**: останов, прямого счёта, непрерывного счёта и реверсивного счёта. Выбор рабочего режима осуществляется битами MCx.

Таблица 13.1. **Режимы работы таймера**

MCx	Режим	Описание
00	Останов	Таймер остановлен
01	Прямого счёта	Таймер циклически считает от нуля до значения, записанного в защёлке сравнения TBCL0
10	Непрерывного счёта	Таймер циклически считает от нуля, до значения, определяемого битами CNTLx
11	Реверсивного счёта	Таймер циклически считает от нуля до значения, записанного в защёлке TBCL0, а затем в обратном направлении до нуля

Режим прямого счёта

Режим прямого счёта используется в том случае, если период таймера должен быть отличным от $TBR_{(max)}$. В этом режиме таймер циклически считает в прямом направлении до тех пор, пока его значение не станет равным содержимому защёлки сравнения TBCL0, определяющему период счёта, как показано на **Рис. 13.2**. Длительность периода в тактах равна TBCL0 + 1. Когда значение таймера становится равным содержимому TBCL0, таймер сбрасывается и счёт начинается с нуля. Если в момент выбора режима прямого счёта значение таймера будет больше числа, находящегося в защёлке TBCL0, то таймер сразу же начнёт счёт с нулевого значения.

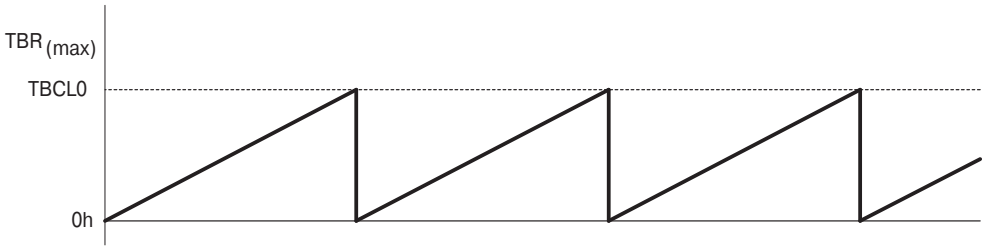


Рис. 13.2. Режим прямого счёта.

Флаг прерывания CCIFG, соответствующий регистру TBCCR0, устанавливается при достижении таймером в процессе счёта значения, хранящегося в защёлке TBCL0. Флаг прерывания TBIFG устанавливается при счёте таймера от значения, содержащегося в защёлке TBCL0, до нуля (**Рис. 13.3**).

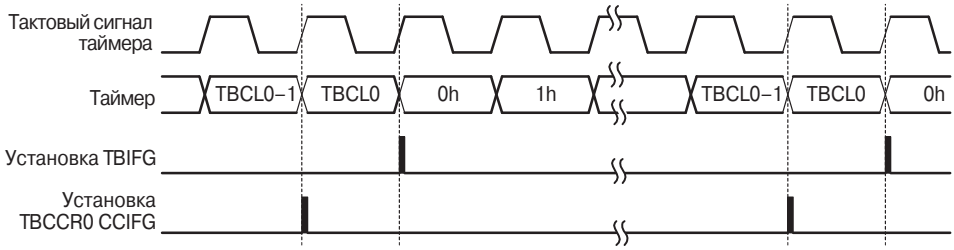


Рис. 13.3. Установка флагов прерываний в режиме прямого счёта.

Изменение регистра периода TBCL0

Если при изменении содержимого регистра-защёлки TBCL0 во время работы таймера в случае немедленной загрузки TBCL0 (биты CLLD0 = 00) новый период окажется больше или равен предыдущему периоду или больше текущего значения таймера, таймер продолжит счёт до достижения нового значения периода. Если новый период окажется меньше текущего значения таймера, то таймер начнёт счёт с нуля, однако перед обнулением счётчика может возникнуть дополнительный отсчёт.

Режим непрерывного счёта

В режиме непрерывного счёта таймер циклически считает в прямом направлении от нуля до значения $TBR_{(max)}$, как показано на **Рис. 13.4**. Защёлка сравнения TBCL0 используется так же, как и регистры-защёлки остальных блоков захвата/сравнения.

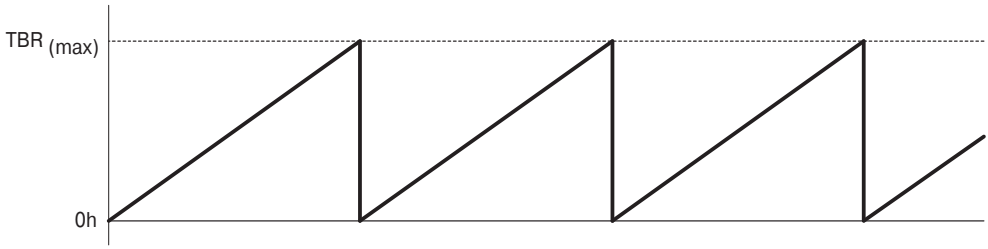


Рис. 13.4. Режим непрерывного счёта.

Флаг прерывания TBIFG устанавливается при счёте таймера от значения $TBR_{(max)}$ до нуля (**Рис. 13.5**).

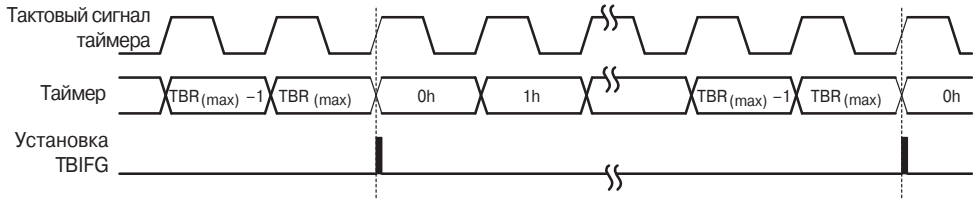


Рис. 13.5. Установка флага прерывания в режиме непрерывного счёта.

Использование режима непрерывного счёта

Режим непрерывного счёта может использоваться для формирования независимых временных интервалов и сигналов разных частот. Каждый раз при завершении очередного интервала генерируется прерывание. В процедуре обработки прерывания длительность следующего интервала прибавляется к содержимому защёлки TBCLx. На **Рис. 13.6** показано формирование двух временных интервалов длительностью t_0 и t_1 (эти значения прибавляются к содержимому регистров захвата/сравнения). В данном случае формирование временных интервалов осуществляется аппаратно, и задержка обработки прерывания не сказывается на их длительности. С помощью блоков захвата/сравнения можно формировать до 3 (модуль Timer_B3) или до 7 (модуль Timer_B7) независимых временных интервалов или сигналов с разными частотами.

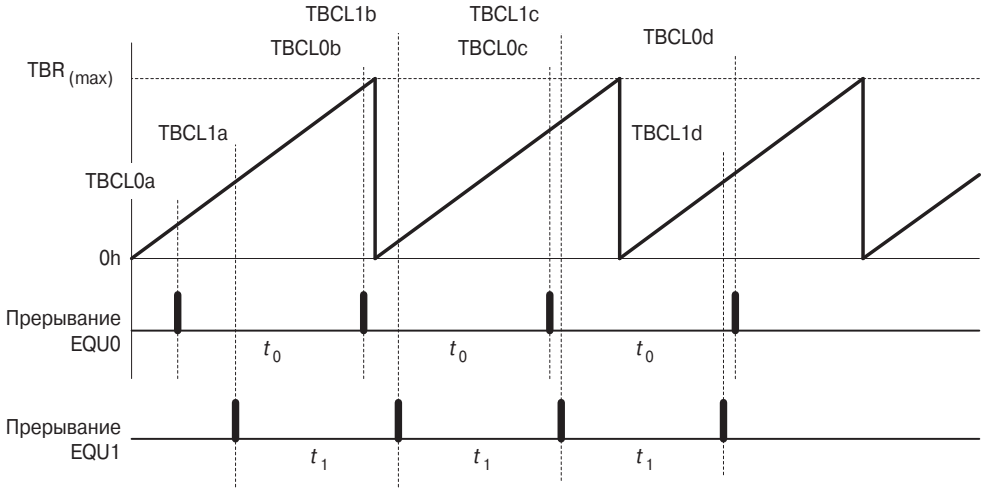


Рис. 13.6. Формирование временных интервалов в режиме непрерывного счёта.

Временные интервалы можно формировать и в других режимах, использующих регистр TBCL0 в качестве регистра периода. Однако при этом необходимо предусмотреть обработку ситуации, когда сумма старого содержимого TBCLx и нового периода оказывается больше значения TBCL0. В таком случае для получения корректного результата из получившейся суммы необходимо вычесть значение TBCL0 + 1.

Режим реверсивного счёта

Режим реверсивного счёта используется в случае, если период таймера должен быть отличным от $TBR_{(max)}$ и если необходимо генерировать симметричные импульсы. В этом режиме таймер циклически считает в прямом направлении до тех пор, пока его значение не станет равным содержимому защёлки сравнения TBCL0, а затем в обратном направлении до нуля, как показано на **Рис. 13.7**. При этом период равен удвоенному значению TBCL0.

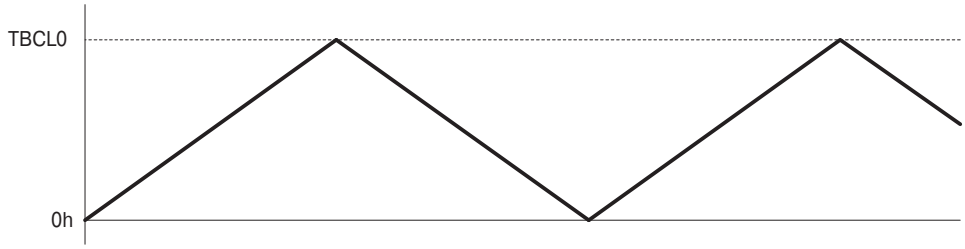


Рис. 13.7. Режим реверсивного счёта.

Направление счёта фиксируется, что позволяет после останова таймера и последующего его запуска продолжать счёт в том же направлении, что и до останова. Если этого не требуется, то для сброса признака направления необходимо записать 1 в бит TBCLR. Установка бита TBCLR также вызывает очистку регистра TBR и делителя тактового сигнала таймера.

В режиме реверсивного счёта флаг прерываний CCIFG для регистра TBCCR0 и флаг TBIFG устанавливаются только один раз за период, причём интервал между установкой этих флагов составляет 1/2 периода таймера. Флаг CCIFG устанавливается при счёте таймера от значения TBCL0 – 1 до TBCL0, а флаг TBIFG — при счёте с 0001h до 0000h (**Рис. 13.8**).

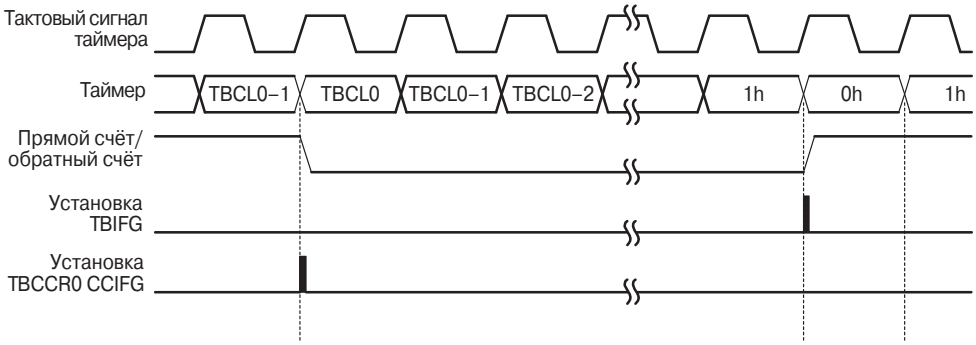


Рис. 13.8. Установка флагов прерываний в режиме реверсивного счёта.

Изменение регистра периода TBCL0

При изменении содержимого TBCL0 во время счёта таймера в обратном направлении, процесс счёта не прерывается даже в случае немедленной загрузки

TBCL0. Новое содержимое TBCCR0 защёлкивается в TBCL0 сразу же после записи, однако формирование периода с новой длительностью начнётся только после достижения счётчиком нулевого значения.

Если в момент загрузки нового значения периода в защёлку TBCL0 таймер считает в прямом направлении и загруженное значение больше или равно старому значению периода или же больше текущего значения таймера, то смена направления счёта произойдёт после достижения таймером нового значения периода. Если новый период меньше текущего значения таймера, то таймер сразу же начинает считать в обратном направлении, однако перед сменой направления может возникнуть один дополнительный отсчёт.

Использование реверсивного режима счёта

Режим реверсивного счёта может использоваться в случаях, когда требуется наличие «мёртвых зон» между выходными сигналами (см. подраздел 13.2.5 «Модуль вывода»). Так, на выводах, используемых для управления Н-мостом, не допускается одновременное появление ВЫСОКОГО уровня во избежание перегрузки элементов моста.

В примере, показанном на **Рис. 13.9**, величина t_{dead} определяется как

$$t_{dead} = t_{timer} \times (\text{TBCL1} - \text{TBCL3}),$$

где t_{dead} — интервал времени, в течение которого оба вывода должны быть неактивными («мёртвое» время);

t_{timer} — период тактового сигнала таймера;

TBCLx — содержимое защёлки сравнения x.

Возможность одновременной загрузки значений в несколько регистров-защёлок, объединённых в одну группу, гарантирует наличие «мёртвого» времени требуемой длительности.

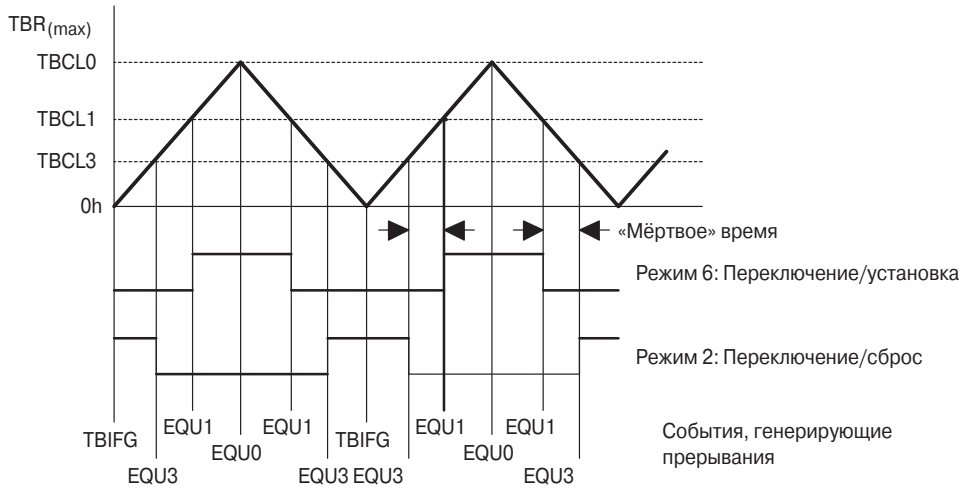


Рис. 13.9. Модуль вывода в режиме реверсивного счёта.

13.2.4. Блоки захвата/сравнения

В Таймере B в зависимости от модели микроконтроллера имеется три или семь идентичных блоков захвата/сравнения. Любой из блоков может использоваться для захвата значения таймера и формирования временных интервалов.

Режим захвата

Режим захвата включается при CAP = 1. Этот режим используется для регистрации временных событий. Он может быть использован для вычисления скорости или измерения временных параметров. Входы захвата CCIxA и CCIxB подключены к выводам микроконтроллера или же к его внутренним шинам. Выбор конкретного входа захвата осуществляется битами CCISx. Биты CMx определяют активный фронт сигнала захвата (нарастающий, спадающий или оба), по которому выполняется операция захвата. При выполнении этой операции:

- значение таймера копируется в соответствующий регистр TBCCRx;
- устанавливается флаг прерывания CCIFG.

Уровень входного сигнала можно в любой момент времени определить по состоянию бита CCI. В зависимости от модели на входы CCIxA и CCIxB могут поступать различные сигналы. Для получения более подробной информации обратитесь к документации на конкретный микроконтроллер.

Сигнал захвата может быть асинхронным по отношению к тактовому сигналу таймера, что приведёт к возникновению «гонок». При установленном бите SCS операция захвата синхронизируется с тактовым сигналом таймера. Рекомендуется всегда устанавливать этот бит. Синхронизация сигнала захвата показана на **Рис. 13.10**.

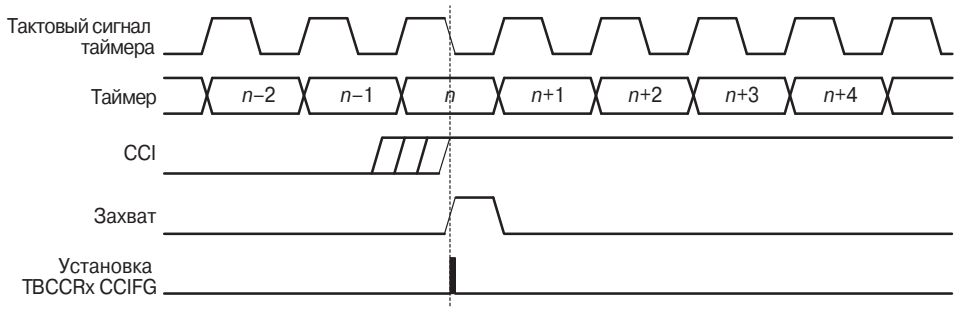


Рис. 13.10. Сигнал захвата при SCS = 1.

Для каждого регистра захвата/сравнения предусмотрена схема переполнения, служащая для индикации выполнения операции захвата до считывания результата предыдущей операции. При возникновении такой ситуации устанавливается бит COV, как показано на **Рис. 13.11**. Бит COV должен сбрасываться программно.

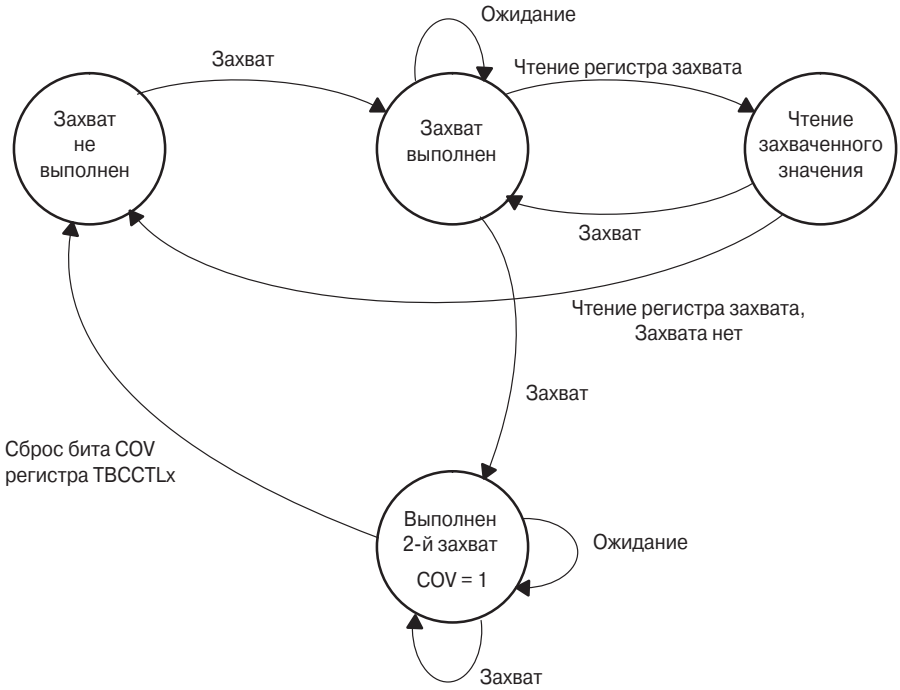


Рис. 13.11. Цикл захвата.

Захват, инициируемый программно

Захват может быть инициирован программно. Для выполнения захвата по обоим фронтам можно установить биты CMx = 11. После этого в программе достаточно установить бит CCIS1 и изменить состояние бита CCIS0 для формирования фронта сигнала захвата. Захват будет инициироваться при каждом изменении бита CCIS0:

```
MOV   #CAP+SCS+CCIS1+CM_3,&TBCCTLx     ; Настроим TBCCTLx
XOR   #CCIS0,&TBCCTLx                  ; TBCCTLx = TBR
```

Режим сравнения

Режим сравнения включается при CAP = 0. Этот режим используется для генерации сигналов с широтно-импульсной модуляцией или для формирования прерываний с заданной периодичностью. При достижении таймером в процессе счёта значения, находящегося в защёлке TBCLx:

- устанавливается флаг прерывания CCIFG;
- формируется внутренний сигнал EQUx = 1;
- сигнал EQUx воздействует на выход в соответствии с режимом работы модуля вывода.

Регистры-защёлки TBCLx

В регистрах-защёлках блоков захвата/сравнения TBCLx хранятся значения, используемые для сравнения с текущим значением таймера в режиме сравнения. Каждая из защёлок TBCLx буферирована регистром TBCCRx. Наличие буферизации даёт пользователю возможность управлять моментом обновления содержимого защёлки. Напрямую защёлки TBCLx для пользователя недоступны. Сравниваемое значение заносится в регистр TBCCRx, откуда автоматически перегружается в TBCLx. Момент пересылки содержимого TBCCRx в TBCLx задаётся пользователем при помощи битов CLLDx в соответствии с **Табл. 13.2**.

Таблица 13.2. **Варианты загрузки защёлок TBCLx**

CLLDx	Описание
00	Новое значение перегружается из TBCCRx в TBCLx одновременно с его записью в регистр TBCCRx
01	Новое значение перегружается из TBCCRx в TBCLx при достижении таймером в процессе счёта нулевого значения
10	В режимах прямого и непрерывного счёта новое значение перегружается из TBCCRx в TBCLx при достижении таймером в процессе счёта нулевого значения. В режиме реверсивного счёта новое значение перегружается из TBCCRx в TBCLx при достижении таймером в процессе счёта старого значения TBCL0 или же нулевого значения
11	Новое значение перегружается из TBCCRx в TBCLx при достижении таймером в процессе счёта старого значения TBCLx

Группирование регистров-защёлок

Для одновременного обновления содержимого нескольких регистров-защёлок они могут быть сгруппированы при помощи битов TBCLGRPx. При использовании группирования момент загрузки новых значений в защёлки группы определяется битами CCLDx, относящимися к регистру TBCCRx с наименьшим номером, за исключением варианта TBCLGRP = 3 (**Табл. 13.3**).

Таблица 13.3. **Группирование регистров-защёлок**

TBCLGRPx	Группирование	Управление загрузкой
00	Нет	Индивидуально
01	TBCL1 + TBCL2 TBCL3 + TBCL4 TBCL5 + TBCL6	TBCCR1 TBCCR3 TBCCR5
10	TBCL1 + TBCL2 + TBCL3 TBCL4 + TBCL5 + TBCL6	TBCCR1 TBCCR4
11	TBCL0 + TBCL1 + TBCL2 + TBCL3 + TBCL4 + TBCL5 + TBCL6	TBCCR1

При этом значение битов CLLDx управляющего регистра TBCCRx должно быть отличным от нуля. Если все биты CCLDx управляющего регистра TBCCRx сброшены, то обновление содержимого всех защёлок сравнения производится при записи в соответствующие регистры TBCCRx; группирование в этом случае отсутствует.

Для загрузки новых значений в регистры-защёлки при использовании группирования необходимо выполнение двух условий. Во-первых, новое значение

должно быть записано во все регистры TBCCRx группы, даже если оно совпадает со старым. Во-вторых, должно наступить соответствующее событие.

13.2.5. Модуль вывода

Каждый блок захвата/сравнения содержит модуль вывода. Этот модуль используется для генерации различных сигналов, в том числе сигналов с ШИМ. Каждый модуль вывода имеет восемь режимов работы, которые используются для формирования различных сигналов в соответствии с внутренними сигналами EQU0 и EQUx. Один из выводов микроконтроллера, имеющий альтернативную функцию TBOUTH, может быть использован для перевода выходов Таймера B в высокоимпедансное состояние. При выборе для вывода порта данной функции и подаче на этот вывод сигнала ВЫСОКОГО уровня все выходы Таймера B переключаются в состояние с высоким входным сопротивлением.

Режимы модуля вывода

Режимы работы модуля вывода определяются битами OUTMODx и перечислены в **Табл. 13.4**. Сигнал OUTx изменяется по нарастающему фронту тактового сигнала таймера во всех режимах, кроме режима 0. Режимы 2, 3, 6 и 7 не применимы к 0-му модулю вывода, поскольку в этом случае EQUx = EQU0.

Таблица 13.4. **Режимы работы модуля вывода**

OUTMODx	Режим	Описание
000	Вывод	Выходной сигнал OUTx определяется состоянием бита OUTx. Выходной сигнал изменяется сразу же после изменения бита OUTx
001	Установка	При достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx, выход устанавливается. Это состояние сохраняется до тех пор, пока таймер не будет сброшен или пока не будет задан новый режим работы модуля вывода с последующим воздействием на выход
010	Переключение/ сброс	При достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx, выход переключается. Выход сбрасывается при достижении таймером значения, записанного в защёлке TBCL0
011	Установка/ сброс	При достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx, выход устанавливается. Выход сбрасывается при достижении таймером значения, записанного в защёлке TBCL0
100	Переключение	Выход переключается при достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx. Период выходного сигнала равен удвоенному периоду таймера
101	Сброс	При достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx, выход сбрасывается. Это состояние сохраняется до тех пор, пока не будет задан новый режим работы модуля вывода с последующим воздействием на выход
110	Переключение/ установка	При достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx, выход переключается. Выход устанавливается при достижении таймером значения, записанного в защёлке TBCL0
111	Сброс/ установка	При достижении таймером в процессе *счёта* значения, записанного в защёлке TBCLx, выход сбрасывается. Выход устанавливается при достижении таймером значения, записанного в защёлке TBCL0

Использование модуля вывода — таймер в режиме прямого счёта

Сигнал OUTx изменяется при достижении таймером в процессе счёта значения, записанного в защёлке TBCLx, а также при переходе таймера от значения, записанного в защёлке TBCL0, к нулевому значению. На **Рис. 13.12** показано формирование выходного сигнала в различных режимах работы модуля вывода с использованием защёлок TBCL0 и TBCL1.

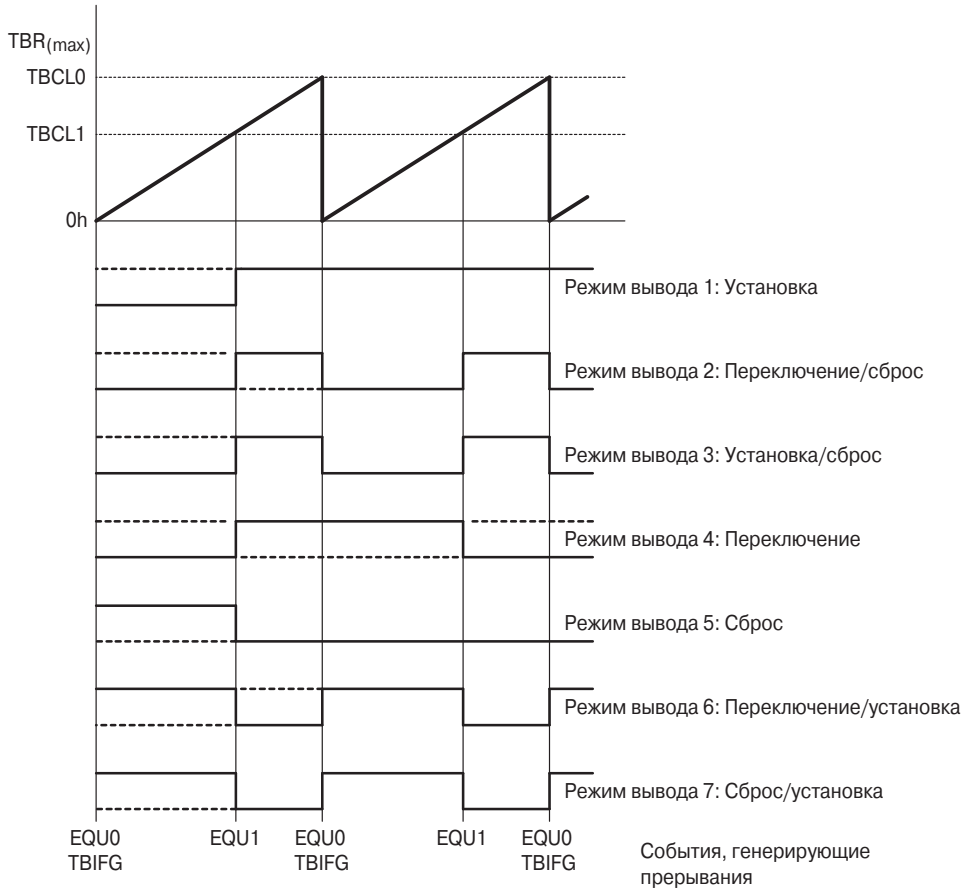


Рис. 13.12. Формирование выходного сигнала — таймер в режиме прямого счёта.

Использование модуля вывода — таймер в режиме непрерывного счёта

Сигнал OUTx изменяется при достижении таймером в процессе счёта значений, записанных в защёлках TBCLx и TBCL0. На **Рис. 13.13** показано формирование выходного сигнала в различных режимах работы модуля вывода с использованием защёлок TBCL0 и TBCL1.

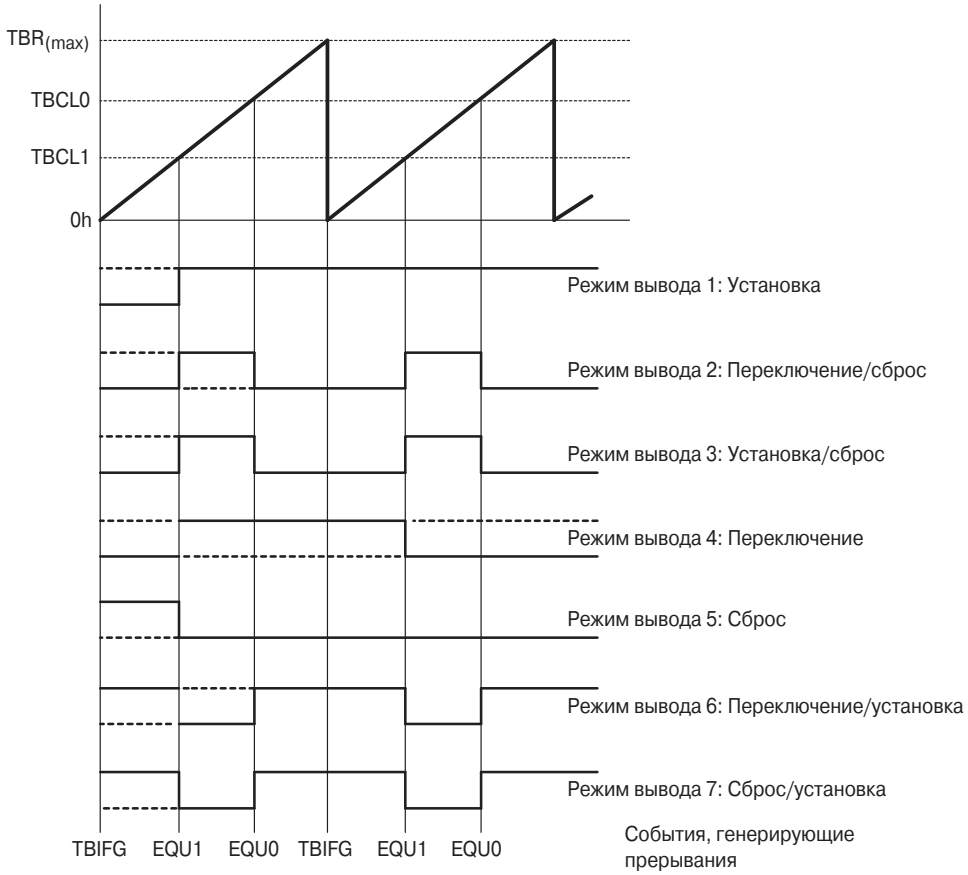


Рис. 13.13. Формирование выходного сигнала — таймер в режиме непрерывного счёта.

Использование модуля вывода — таймер в режиме реверсивного счёта

Сигнал OUTx изменяется при достижении таймером в процессе счёта значения, записанного в защёлке TBCLx (вне зависимости от направления счёта), а также при достижении значения, записанного в защёлке TBCL0. На **Рис. 13.14** показано формирование выходного сигнала в различных режимах работы модуля вывода с использованием защёлок TBCL0 и TBCL3.

Примечание. Переключение между режимами модуля вывода
При переходе от одного режима работы модуля вывода к другому, один из битов OUTMODx в процессе переключения обязательно должен оставаться в установленном состоянии, за исключением переключения в режим 0. В противном случае

из-за ошибочного декодирования элементом ИЛИ-НЕ режима 0 модуля возможно появление на выходе паразитных импульсов. Безопасный метод переключения режимов модуля заключается в использовании в качестве промежуточного состояния режима 7.

```
BIS   #OUTMOD_7,&TBCCTLx  ; Устанавливаем режим 7
BIC   #OUTMODx,&TBCCTLx   ; Сбрасываем требуемые биты
```

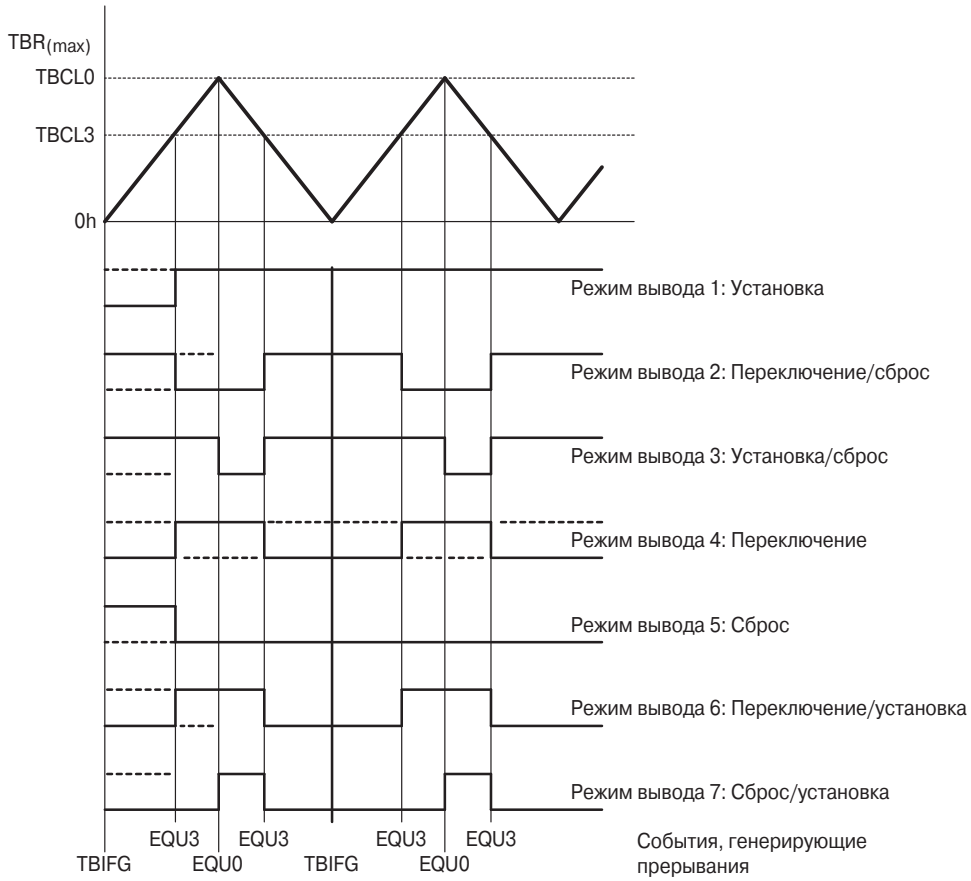


Рис. 13.14. Формирование выходного сигнала — таймер в режиме реверсивного счёта.

13.2.6. Прерывания Таймера B

С модулем 16-битного Таймера B связаны два вектора прерывания:

- вектор прерывания TBCCR0 для бита CCIFG, соответствующего регистру TBCCR0;
- вектор прерывания TBIV для всех остальных флагов CCIFG и флага TBIFG.

В режиме захвата любой из флагов CCIFG устанавливается в момент сохранения значения таймера в соответствующем регистре TBCCRx. В режиме сравнения любой из флагов CCIFG устанавливается при достижении таймером в процессе счёта значения, загруженного в соответствующую защёлку TBCLx. Кроме

того, любой флаг CCIFG можно установить или сбросить программно. При установке флага CCIFG, если установлен соответствующий бит CCIE и бит общего разрешения прерываний GIE, генерируется запрос прерывания.

Прерывание TBCCR0

Флаг прерывания CCIFG для регистра TBCCR0 соответствует прерыванию Таймера B с наивысшим приоритетом и имеет отдельный вектор прерывания, как показано на **Рис. 13.15**. Флаг CCIFG для регистра TBCCR0 автоматически сбрасывается при обслуживании запроса прерывания TBCCR0.

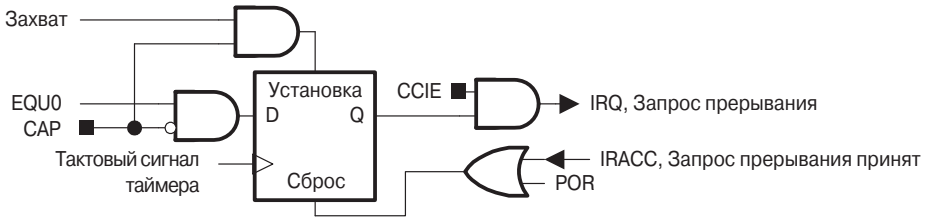


Рис. 13.15. Флаг прерывания захвата/сравнения TACCR0.

Генератор вектора прерывания TBIV

Флаги CCIFG для регистров TBCCR1...TBCCR6 и флаг TBIFG имеют различные приоритеты и служат источником единственного вектора прерывания. Для определения, какой из флагов вызвал генерацию прерывания, используется регистр вектора прерывания TBIV.

Разрешённое прерывание с наивысшим приоритетом формирует в регистре TBIV число (см. описание регистра). Это число можно оценить или же просто прибавить к счётчику команд для автоматического перехода к соответствующей секции программы. Запрещённые прерывания таймера не влияют на содержимое регистра TBIV.

При любом обращении к регистру TBIV как для чтения, так и для записи флаг прерывания с наивысшим приоритетом, ожидающего обработки, автоматически сбрасывается. Если имеется ещё один установленный флаг, то сразу же после завершения обработки текущего прерывания будет сгенерировано новое прерывание. Например, если на момент обращения к регистру TBIV в процедуре обработки прерывания были установлены флаги CCIFG для регистров TBCCR1 и TBCCR2, то будет автоматически сброшен флаг CCIFG для регистра TBCCR1. После выполнения команды `RETI` процедуры обработки прерывания флаг CCIFG для регистра TBCCR2 вызовет генерацию нового прерывания.

Пример процедуры обработки прерывания TBIV

В следующем примере показано рекомендуемое использование регистра TBIV и приведена информация о накладных расходах на обработку прерывания. Содержимое регистра TBIV прибавляется к счётчику команд для автоматического перехода к соответствующей процедуре.

Значения, расположенные по правому краю, показывают число тактов ЦПУ, требуемое для выполнения каждой команды. Накладные расходы на обработку различных источников прерывания включают в себя задержку обслуживания прерывания и время, требуемое для возврата из прерывания, но не время, необходимое для выполнения собственно задачи. Задержки обработки прерывания имеют следующие значения:

- Блок захвата/сравнения CCR0 — 11 тактов
- Блоки захвата/сравнения с CCR1 по CCR6 — 16 тактов
- Переполнение таймера TBIFG — 14 тактов

```
; Обработчик прерывания для TBCCR0 CCIFG.                          Тактов
CCIFG_0_HND
;        ...                ; Задержка обработки прерывания            6
         RETI                                                          5
; Обработчик прерывания для TBIFG, TBCCR1 и TBCCR2 CCIFG
TB_HND   ...                ; Задержка обработки прерывания            6
         ADD    &TBIV,PC    ; Прибавляем смещение в таблице переходов   3
         RETI               ; Вектор 0: Нет прерывания                 5
         JMP    CCIFG_1_HND ; Вектор 2: Модуль 1                       2
         JMP    CCIFG_2_HND ; Вектор 4: Модуль 2                       2
         RETI               ; Вектор 6
         RETI               ; Вектор 8
         RETI               ; Вектор 10
         RETI               ; Вектор 12

TBIFG_HND                   ; Вектор 14: Флаг TBIFG
         ...                ; Здесь начало задачи
         RETI                                                          5

CCIFG_2_HND                 ; Вектор 4: Модуль 2
         ...                ; Здесь начало задачи
         RETI               ; Возвращаемся в основную программу        5

; В обработчике 1-го модуля используется программное решение,
; позволяющее определить, нет ли других прерываний, ожидающих
; обработки. На это тратится 5 тактов, однако при наличии
; такого прерывания данное решение позволяет сэкономить 9 тактов
CCIFG_1_HND                 ; Вектор 2: Модуль 1
         ...                ; Здесь начало задачи
         JMP    TB_HND      ; Возвращаемся в основную программу        5
```

13.3. Регистры Таймера B

Список регистров Таймера B приведён в **Табл. 13.5**.

Таблица 13.5. **Регистры Таймера B**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления Таймера B	TBCTL	Чтение/запись	0180h	Сбрасывается после POR
Регистр счётчика Таймера B	TBR	Чтение/запись	0190h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 0 Таймера B	TBCCTL0	Чтение/запись	0182h	Сбрасывается после POR
Регистр захвата/сравнения 0 Таймера B	TBCCR0	Чтение/запись	0192h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 1 Таймера B	TBCCTL1	Чтение/запись	0184h	Сбрасывается после POR
Регистр захвата/сравнения 1 Таймера B	TBCCR1	Чтение/запись	0194h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 2 Таймера B	TBCCTL2	Чтение/запись	0186h	Сбрасывается после POR
Регистр захвата/сравнения 2 Таймера B	TBCCR2	Чтение/запись	0196h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 3 Таймера B	TBCCTL3	Чтение/запись	0188h	Сбрасывается после POR
Регистр захвата/сравнения 3 Таймера B	TBCCR3	Чтение/запись	0198h	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 4 Таймера B	TBCCTL4	Чтение/запись	018Ah	Сбрасывается после POR
Регистр захвата/сравнения 4 Таймера B	TBCCR4	Чтение/запись	019Ah	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 5 Таймера B	TBCCTL5	Чтение/запись	018Ch	Сбрасывается после POR
Регистр захвата/сравнения 5 Таймера B	TBCCR5	Чтение/запись	019Ch	Сбрасывается после POR
Регистр управления блоком захвата/сравнения 6 Таймера B	TBCCTL6	Чтение/запись	018Eh	Сбрасывается после POR
Регистр захвата/сравнения 6 Таймера B	TBCCR6	Чтение/запись	019Eh	Сбрасывается после POR
Регистр вектора прерывания Таймера B	TBIV	Только чтение	011Eh	Сбрасывается после POR

TBCTL, регистр управления Таймера B

15	14	13	12	11	10	9	8
Unused	**TBCLGRPx**		**CNTLx**		**Unused**	**TBSSELx**	
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
IDx		**MCx**		**Unused**	**TBCLR**	**TBIE**	**TBIFG**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	w–(0)	rw–(0)	rw–(0)

Unused	Бит 15	Не используется.
TBCLGRPx	Биты 14...13	Группирование TBCLx. 00 Каждая защёлка TBCLx загружается индивидуально 01 TBCL1 + TBCL2 (момент загрузки определяется битами CLLDx регистра TBCCR1) TBCL3 + TBCL4 (момент загрузки определяется битами CLLDx регистра TBCCR3) TBCL5 + TBCL6 (момент загрузки определяется битами CLLDx регистра TBCCR5) TBCL0 загружается независимо от других 10 TBCL1 + TBCL2 + TBCL3 (момент загрузки определяется битами CLLDx регистра TBCCR1) TBCL4 + TBCL5 + TBCL6 (момент загрузки определяется битами CLLDx регистра TBCCR4) TBCL0 загружается независимо от других 11 TBCL0 + TBCL1 + TBCL2 + TBCL3 + TBCL4 + TBCL5 + TBCL6 (момент загрузки определяется битами CLLDx регистра TBCCR1)
CNTLx	Биты 12...11	Разрядность счётчика. 00 16 бит, $TBR_{(max)}$ = 0FFFFh 01 12 бит, $TBR_{(max)}$ = 0FFFh 10 10 бит, $TBR_{(max)}$ = 03FFh 11 8 бит, $TBR_{(max)}$ = 0FFh
Unused	Бит 10	Не используется.
TBSSELx	Биты 9...8	Выбор источника тактового сигнала. 00 TACLK 01 ACLK 10 SMCLK 11 INCLK
IDx	Биты 7...6	Коэффициент деления входного делителя. Эти биты определяют коэффициент деления задающего тактового сигнала. 00 /1 01 /2 10 /4 11 /8
MCx	Биты 5...4	Управление режимом таймера. Установка MCx = 00h при неиспользуемом таймере уменьшает общее потребление микроконтроллера. 00 Останов: таймер остановлен 01 Прямой счёт: таймер считает от 0000h до TBCL0 10 Непрерывный счёт: таймер считает от 0000h до значения, определяемого битами CNTLx 11 Реверсивный счёт: таймер считает от 0000h до TBCL0, а потом в обратном направлении до 0000h
Unused	Бит 3	Не используется.
TBCLR	Бит 2	Очистка таймера. Установка этого бита обнуляет регистр TBR, делитель и сбрасывает признак направления счёта. Бит TBCLR автоматически сбрасывается; всегда читается как 0.
TBIE	Бит 1	Разрешение прерывания таймера. Этот бит разрешает генерацию запроса прерывания TBIFG. 0 Прерывание запрещено 1 Прерывание разрешено
TBIFG	Бит 0	Флаг прерывания таймера. 0 Прерывания не было 1 Прерывание было

TBR, регистр счётчика Таймера B

15	14	13	12	11	10	9	8
TBRx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
TBRx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

TBRx Биты 15...0 Регистр счётчика Таймера B. В этом регистре содержится текущее значение Таймера B.

TBCCRx, регистр захвата/сравнения x Таймера B

15	14	13	12	11	10	9	8
TBCCRx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
TBCCRx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

TBCCRx Биты 15...0 Регистр захвата/сравнения Таймера B.
Режим сравнения: Число для сравнения загружается в регистр TBCCRx и автоматически пересылается в защёлку TBCLx. В защёлке TBCLx содержится число, которое сравнивается с содержимым регистра счётчика TBR.
Режим захвата: Содержимое регистра TBR копируется в регистр TBCCRx в момент выполнения операции захвата.

TBCCTLx, регистр управления блока захвата/сравнения

15	14	13	12	11	10	9	8
CMx		**CCISx**		**SCS**	**CLLDx**		**CAP**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
OUTMODx			**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	r	rw–(0)	rw–(0)	rw–(0)

CMx Биты 15...14 Режим захвата.
- 00 Нет захвата
- 01 Захват по нарастающему фронту
- 10 Захват по спадающему фронту
- 11 Захват по обоим фронтам

CCISx	Биты 13...12	Выбор входа захвата/сравнения. Эти биты определяют входной сигнал блока захвата/сравнения. Соответствие сигналов каждому из входов приводится в документации на конкретные модели. 00 CCIxA 01 CCIxB 10 GND 11 V_{CC}
SCS	Бит 11	Синхронизация захвата. Этот бит используется для синхронизации сигнала захвата с тактовым сигналом таймера. 0 Асинхронный захват 1 Синхронный захват
CLLDx	Биты 10...9	Загрузка защёлки сравнения. Эти биты определяют момент загрузки значения в защёлку. 00 TBCLx загружается при записи в TBCCRx 01 TBCLx загружается, когда TBR досчитывает до 0 10 TBCLx загружается, когда TBR досчитывает до 0 (режимы прямого и непрерывного счёта) TBCLx загружается, когда TBR досчитывает до TBCL0 или до 0 (режим реверсивного счёта) 11 TBCLx загружается, когда TBR досчитывает до TBCLx
CAP	Бит 8	Режим работы блока захвата/сравнения. 0 Режим захвата 1 Режим сравнения
OUTMODx	Биты 7...5	Режим работы модуля вывода. Для регистра TBCCR0 использование режимов 2, 3, 6 и 7 не имеет смысла, поскольку в этом случае EQUx = EQU0. 000 Состояние бита OUT 001 Установка 010 Переключение/сброс 011 Установка/сброс 100 Переключение 101 Сброс 110 Переключение/установка 111 Сброс/установка
CCIE	Бит 4	Разрешение прерывания захвата/сравнения. Этот бит разрешает генерацию запроса прерывания при установке соответствующего флага CCIFG. 0 Прерывание запрещено 1 Прерывание разрешено
CCI	Бит 3	Вход захвата/сравнения. Посредством этого бита можно определить значение входного сигнала блока захвата сравнения.
OUT	Бит 2	Состояние выхода. При работе модуля вывода в режиме 0 этот бит напрямую управляет состоянием выхода. 0 На выходе НИЗКИЙ уровень 1 На выходе ВЫСОКИЙ уровень
COV	Бит 1	Переполнение захвата. Этот бит индицирует переполнение при операциях захвата. Бит COV должен сбрасываться программно. 0 Не было переполнения при захвате 1 Было переполнение при захвате

CCIFG Бит 0 Флаг прерывания захвата/сравнения.
0 Прерывания не было
1 Прерывание было

TBIV, регистр вектора прерывания

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**0**	**TBIVx**			**0**
r0	r0	r0	r0	r–(0)	r–(0)	r–(0)	r0

TBIVx Биты 15...0 Значение вектора прерывания таймера

Содержимое TBIV	Источник прерывания	Флаг прерывания	Приоритет прерывания
00h	Не было прерываний	—	
02h	Захват/сравнение 1	TBCCR1 CCIFG	Высший
04h	Захват/сравнение 2	TBCCR2 CCIFG	
06h	Захват/сравнение 3*	TBCCR3 CCIFG	
08h	Захват/сравнение 4*	TBCCR4 CCIFG	
0Ah	Захват/сравнение 5*	TBCCR5 CCIFG	
0Ch	Захват/сравнение 6*	TBCCR6 CCIFG	
0Eh	Переполнение таймера	TAIFG	Низший
* Реализовано не во всех моделях			

ГЛАВА 14

УНИВЕРСАЛЬНЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ ИНТЕРФЕЙС

Модуль универсального последовательного интерфейса (USI) позволяет организовать обмен с внешними устройствами по интерфейсам SPI и I^2C. В этой главе описывается работа модуля в обоих режимах. Модуль USI реализован в моделях MSP430x20xx семейства.

14.1. Введение

Модуль USI обеспечивает базовые функции для поддержки синхронного обмена по последовательному каналу. В наиболее упрощённом виде этот модуль представляет собой 8- или 16-битный сдвиговый регистр, который может использоваться для передачи потока битов. В сочетании с незначительным по объёму программным кодом данный регистр позволяет реализовать обмен по последовательному каналу. Кроме того, в составе модуля USI имеются функциональные узлы, облегчающие организацию обмена по интерфейсам SPI и I^2C. Также в модуле USI предусмотрена поддержка прерываний, что позволяет ещё больше уменьшить программные издержки, необходимые для реализации последовательного обмена и, таким образом, уменьшить потребление микроконтроллера в соответствии с идеологией семейства MSP430.

Модуль USI имеет следующие особенности:

- поддержка 3-проводного интерфейса SPI;
- поддержка интерфейса I^2C;
- изменяемая разрядность данных;
- работа в качестве ведомого в режиме LPM4 — внутренний тактовый сигнал не требуется;
- изменяемый порядок передачи битов;
- обнаружение состояний СТАРТ и СТОП в режиме I^2C с автоматическим контролем линии SCL;
- обнаружение потери арбитража в режиме ведущего;
- программируемый сигнал синхронизации;
- возможность выбора полярности и фазы тактового сигнала.

Блок-схема модуля USI в режиме SPI приведена на **Рис. 14.1**, а в режиме I^2C — на **Рис. 14.2**.

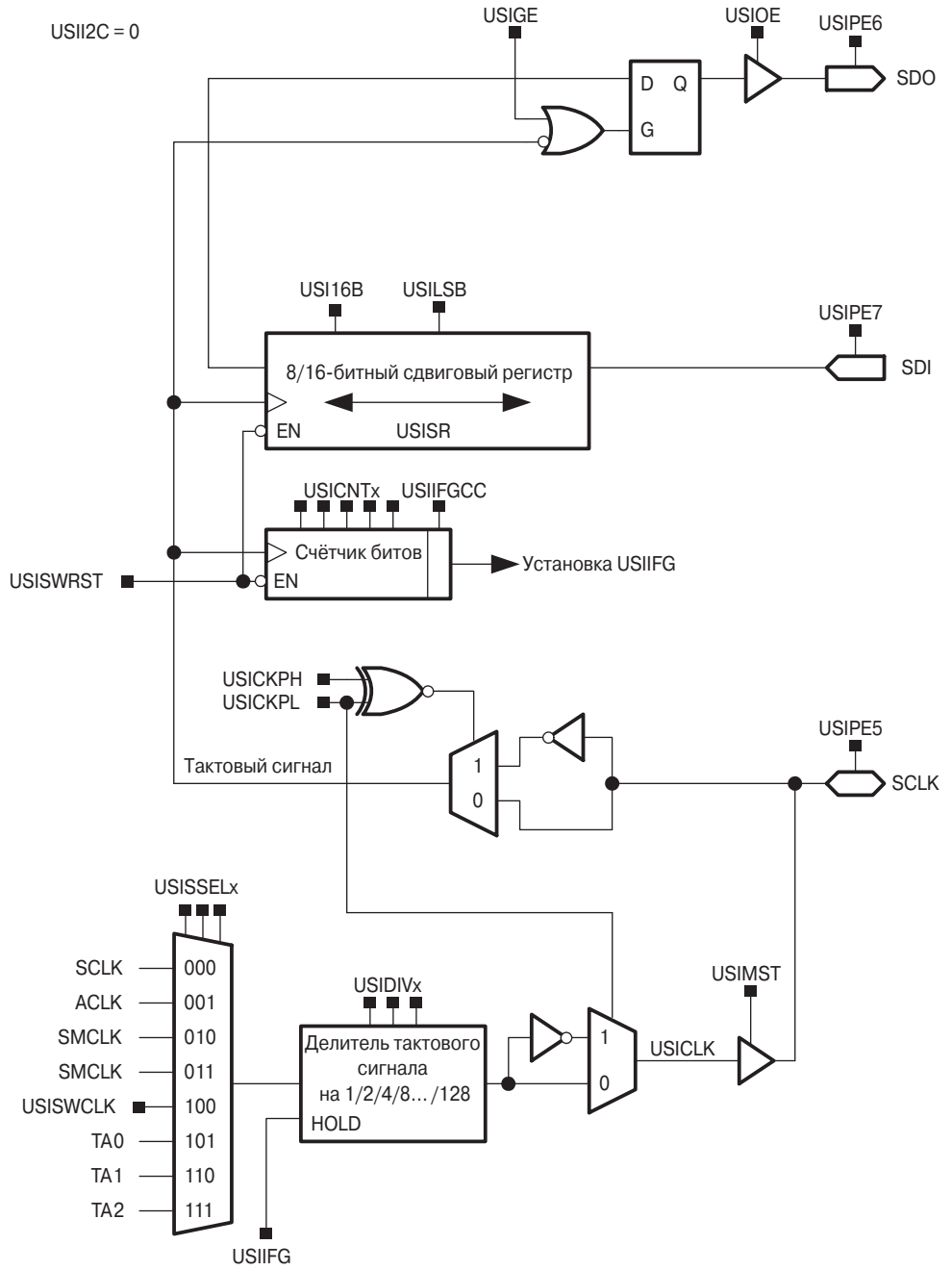


Рис. 14.1. Блок-схема модуля USI: режим SPI.

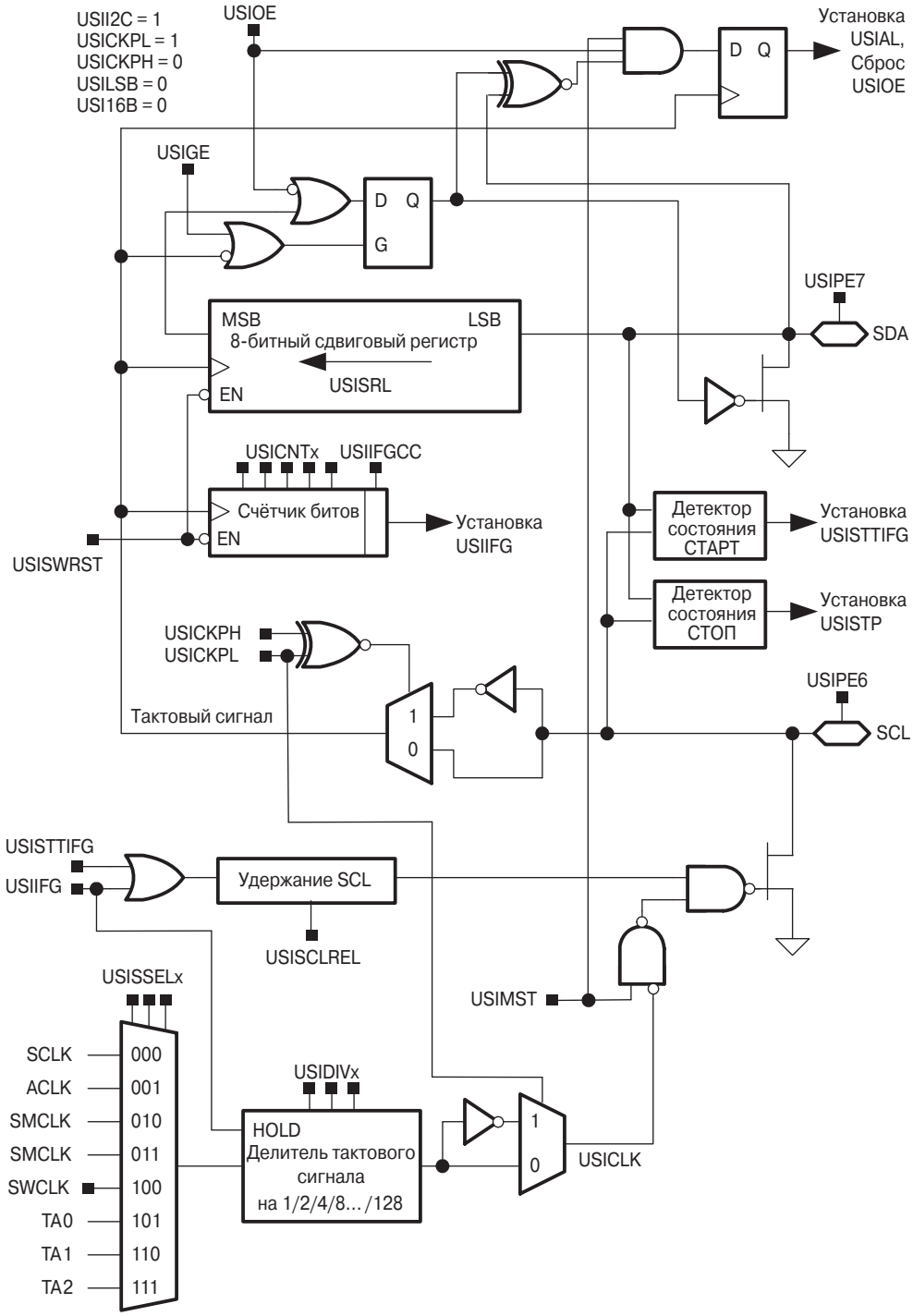


Рис. 14.2. Блок-схема модуля USI: режим I^2C.

14.2. Функционирование модуля USI

Основными элементами модуля USI являются сдвиговый регистр и счётчик битов, дополненные логическими узлами, обеспечивающими поддержку интерфейсов SPI и I^2C. Сдвиговый регистр модуля USISR напрямую доступен из программы и содержит принятые или требующие передачи данные.

Счётчик битов отсчитывает число принятых битов и устанавливает флаг прерывания модуля USI USIIFG в тот момент, когда содержимое битов USICNTx счётчика становится равным нулю — либо в результате декрементирования счётчика, либо в результате непосредственной записи нуля в биты USICNTx. Запись в биты USICNTx любого значения, отличного от нуля, вызывает сброс флага USIIFG, если бит USIIFGCC = 0. Если же бит USIIFGCC = 1, то состояние флага USIIFG не изменяется. Декрементирование содержимого битов USICNTx прекращается после достижения нулевого значения, т.е. перехода к значению FFh не происходит.

И счётчик, и сдвиговый регистр тактируются одним тактовым сигналом. По нарастающему фронту этого сигнала значение битов USICNTx декрементируется и в регистр USISR вдвигается значение, считанное с входа. Защёлка, подключённая к выходу сдвигового регистра, обеспечивает изменение выхода модуля по спадающему фронту тактового сигнала. Эта защёлка может быть отключена установкой бита USIGE. При установленном бите USIGE младший или старший бит регистра USISR, в зависимости от значения бита USILSB, будет немедленно выставлен на вывод SDO.

14.2.1. Инициализация модуля USI

При установленном бите USISWRST, отвечающем за программный сброс модуля, флаги USIIFG, USISTTIFG, USISTP и USIAL удерживаются в исходном состоянии. Тактовый сигнал на регистры USISR и USICNT не подаётся и, соответственно, их содержимое остаётся неизменным. В режиме I^2C, кроме того, модуль отключается от линии SCL, переводя её в состояние ожидания.

Для активации функций порта универсального интерфейса необходимо установить соответствующие биты USIPEx регистра управления модуля USI. При этом разрешается использование соответствующей линии порта ввода/вывода в качестве вывода модуля USI и обеспечивается поддержка функций регистров PxIN и PxIFG для данного вывода. Эта возможность позволяет программно считывать входной сигнал порта с помощью регистра PxIN, а также генерировать прерывания порта при изменении входного сигнала. В частности, так можно генерировать прерывание порта при появлении состояния СТАРТ.

14.2.2. Генерация тактового сигнала USI

Система синхронизации модуля USI содержит мультиплексор задающих тактовых сигналов, делитель, а также узел выбора полярности тактового сигнала, как показано на **Рис. 14.1** и **Рис. 14.2**.

В качестве источника тактового сигнала модуля USI могут использоваться системные тактовые сигналы ACLK или SMCLK, внешний сигнал синхронизации SCLK или же выходной сигнал любого из блоков захвата/сравнения Таймера A. Кроме того, тактовый сигнал модуля может формироваться программно с помощью бита USISWCLK при USISELx = 100.

Биты USIDIVx определяют коэффициент деления (от 2 до 128, кратный степени двойки) тактового сигнала. Формирование тактового сигнала USICLK прекращается при USIIFG = 1 или же при работе модуля в режиме ведомого.

Бит USICKPL используется для выбора полярности сигнала USICLK. При USICKPL = 0 неактивному состоянию сигнала USICLK соответствует ВЫСОКИЙ уровень, а при USICKPL = 1 — НИЗКИЙ уровень.

14.2.3. Режим SPI

Модуль USI переключается в режим SPI при USII2C = 0. Бит USICKPL определяет неактивный уровень тактового сигнала SPI, в то время как бит USICKPH определяет фронт тактового сигнала, по которому будет выставляться сигнал на вывод SDO и считываться состояние вывода SDI. Временные диаграммы для всех возможных вариантов режима SPI (8-битные данные, MSB передаётся первым) приведены на **Рис. 14.3**. Для активации линий SCLK, SDO и SDI порта должны быть установлены биты USIPE5, USIPE6 и USIPE7.

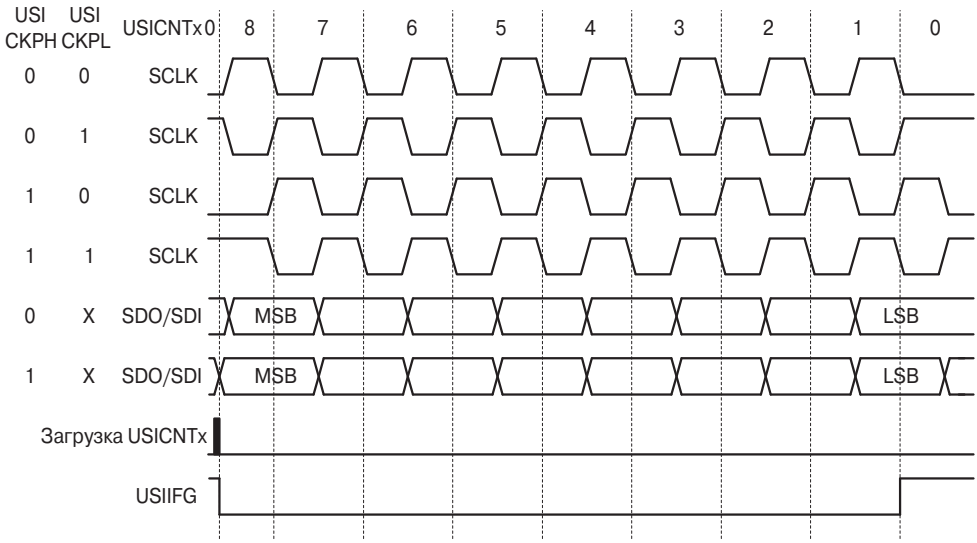


Рис. 14.3. Временные диаграммы для режима SPI.

Режим ведущего SPI

Модуль USI переводится в режим ведущего SPI установкой бита режима ведущего USIMST с одновременным сбросом бита USII2C, отвечающим за включение режима I^2C. Поскольку ведущее устройство генерирует тактовый сигнал для ведомых, необходимо также выбрать соответствующий источник тактового сиг-

нала и сконфигурировать вывод SCLK в качестве выхода. При USIPE5 = 1 вывод SCLK конфигурируется в качестве выхода автоматически.

При USIIFG = 0 и USICNTx > 0 разрешается генерация тактового сигнала и ведущий начинает принимать/передавать данные с использованием регистра USISR.

Полученные данные необходимо считать из сдвигового регистра до того, как в него будут загружены новые передаваемые данные. Как правило, программа считывает полученные данные из сдвигового регистра USISR, загружает в него новые данные и разрешает их передачу, загружая в USICNTx число битов передаваемого слова.

Режим ведомого SPI

Модуль USI переводится в режим ведомого SPI сбросом битов USIMST и USII2C. В этом режиме при USIPE5 = 1 вывод SCLK автоматически конфигурируется как вход, и модуль USI принимает тактовый сигнал от ведущего устройства.

Если модулю требуется передать данные, то их необходимо загрузить в сдвиговый регистр до появления первого фронта тактового сигнала, генерируемого ведущим. Выход модуля должен быть разрешён установкой бита USIOE. При USICKPH = 1 старший бит передаваемого слова данных появится на выводе SDO сразу же после загрузки слова в сдвиговый регистр.

Вывод SDO может быть отключён сбросом бита USIOE. Эта возможность используется неадресованными ведомыми при наличии на шине нескольких ведомых устройств.

После того как все биты будут приняты, полученные данные необходимо считать из сдвигового регистра USISR, а новые данные загрузить в него до появления следующего фронта тактового сигнала от ведущего устройства. Обычно после приёма очередного слова данных программа считывает содержимое регистра USISR, загружает в него новые данные и разрешает модулю передачу новых данных, загружая в USICNTx число битов передаваемого слова.

Сдвиговый регистр USISR

16-битный сдвиговый регистр USISR образован из двух 8-битных регистров USISRL и USISRH. Число битов регистра USISR, используемых для приёма и передачи данных, определяется битом USI16B. При USI16B = 0 используются только 8 младших битов, USISRL.

Если размер передаваемого слова меньше 8 бит, то данные необходимо загружать в регистр USISRL таким образом, чтобы неиспользуемые биты не выдвигались из сдвигового регистра. В зависимости от состояния бита USILSB данные должны быть выровнены по старшему или по младшему биту. В качестве примера на **Рис. 14.4** показана обработка 7-битных данных.

При USI16B = 1 для хранения данных используются все 16 бит регистра USISR. Если размер передаваемого слова меньше 16 бит, то данные должны быть выровнены аналогично тому, как показано на **Рис. 14.4**.

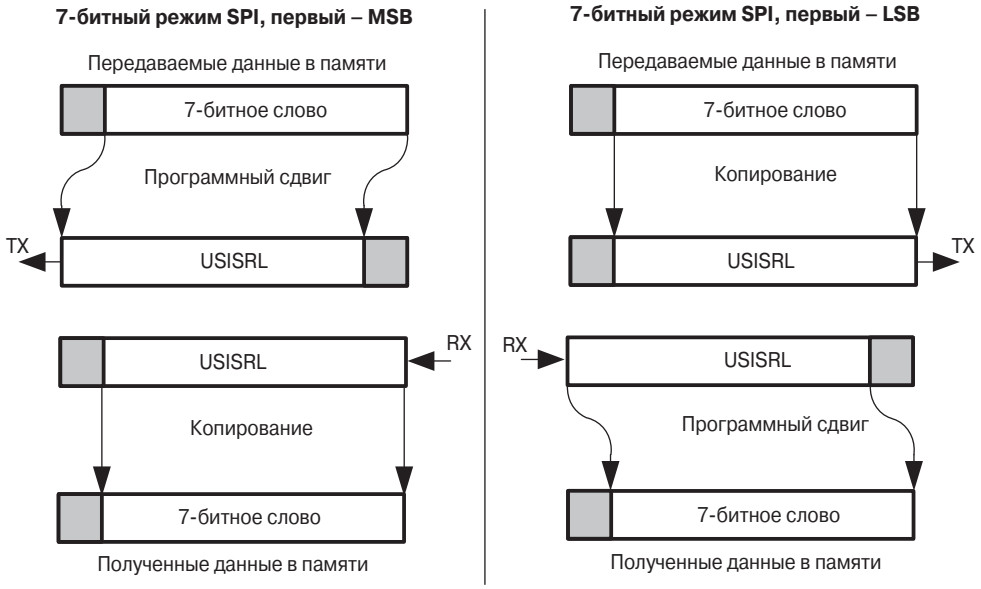


Рис. 14.4. Выравнивание данных при использовании 7-битных слов.

Прерывания в режиме SPI

Модулю USI выделен один вектор прерывания, с которым в режиме SPI связан единственный флаг прерывания USIIFG. При установленных битах USIIE и GIE установка этого флага генерирует запрос прерывания.

Флаг USIIFG устанавливается, когда содержимое счётчика USICNTx становится равным нулю в результате декрементирования либо в результате непосредственной загрузки нулевого значения. Флаг USIIFG сбрасывается при записи в биты USICNTx значения, отличного от нуля (если бит USIIFGCC = 0), или же непосредственно.

14.2.4. Режим I^{2}C

Модуль USI переключается в режим I^{2}C при USII2C = 1, USICKPL = 1 и USICKPH = 0. Для обеспечения совместимости со спецификацией I^{2}C биты USILSB и USI16B должны быть сброшены. Для активации выводов SCL и SDA модуля должны быть установлены биты USIPE6 и USIPE7.

Режим ведущего I^{2}C

Для перевода модуля USI в режим ведущего I^{2}C необходимо установить бит USIMST. В режиме ведущего тактовый сигнал формируется модулем и выдаётся на линию SCL при USIIFG = 0. При USIIFG = 1 на линии SCL постоянно присутствует ВЫСОКИЙ уровень. Поддерживается работа при наличии на шине нескольких ведущих (см. подраздел «Арбитраж» далее).

Поддержка ведомых устройств, способных удерживать на линии SCL НИЗКИЙ уровень, обеспечивается только при USIDIVx > 0. Если коэффициент деле-

ния делителя тактового сигнала модуля USI равен единице (USIDIVx = 0), то подключённые к шине ведомые устройства не должны удерживать линию SCL в состоянии НИЗКОГО уровня во время передачи данных. В противном случае связь может быть нарушена.

Режим ведомого I²C

Для перевода модуля USI в режим ведущего I²C бит USIMST должен быть сброшен. В режиме ведомого на линии SCL удерживается НИЗКИЙ уровень при USIIFG = 1 и USISTTIFG = 1 или если USICNTx = 0. Бит USISTTIFG должен сбрасываться программно после инициализации ведомого устройства, когда модуль будет готов к получению адреса ведомого, передаваемого ведущим устройством.

Передача данных в режиме I²C

В режиме передачи данные сначала загружаются в регистр USISRL. Выход модуля активируется установкой бита USIOE, а процесс передачи данных запускается записью в счётчик USICNTx числа 8. При этом сбрасывается бит USIIFG и на линии SCL начинают формироваться импульсы тактового сигнала (в режиме ведущего) или перестаёт удерживаться НИЗКИЙ уровень (в режиме ведомого). После передачи всех восьми битов флаг USIIFG устанавливается. При этом в режиме ведущего прекращается генерация тактового сигнала, а в режиме ведомого вывод SCL переводится в состояние НИЗКОГО уровня.

Для приёма бита квитирования I²C бит USIOE сбрасывается программой, а в USICNTx загружается 1. При этом сбрасывается флаг USIIFG и принимается один бит. После повторной установки флага USIIFG в младшем бите регистра USSRL содержится принятый бит квитирования, который может быть обработан в программе.

```
; Приём ACK/NACK
  BIC.B  #USIOE,&USICTL0      ; SDA - вход
  MOV.B  #01h,&USICNT         ; USICNTx = 1
TEST_USIIFG
  BIT.B  #USIIFG,&USICTL1     ; Проверяем USIIFG
  JZ     TEST_USIIFG
  BIT.B  #01h,&USISRL         ; Проверяем полученный бит квитирования
  JNZ    HANDLE_NACK          ; Обрабатываем NACK
...Иначе обрабатываем ACK
```

Приём данных в режиме I²C

В режиме приёма выход модуля должен быть отключён сбросом бита USIOE, а модуль USI должен быть подготовлен к приёму данных загрузкой в счётчик USICNTx числа 8. При этом сбрасывается флаг USIIFG и на линии SCL начинают формироваться импульсы тактового сигнала (в режиме ведущего) или перестаёт удерживаться НИЗКИЙ уровень (в режиме ведомого). После восьми импульсов тактового сигнала флаг USIIFG устанавливается. При этом в режиме ведущего прекращается генерация тактового сигнала, а в режиме ведомого вывод SCL переводится в состояние НИЗКОГО уровня.

Для передачи бита квитирования (ACK/NACK) в старший бит сдвигового регистра загружается 0 или 1, программной установкой бита USIOE активируется выход модуля USI, а в счётчик USICNTx загружается 1. После выдачи старшего бита сдвигового регистра устанавливается флаг USIIFG, после чего можно готовить модуль к приёму следующего байта данных по шине I^2C.

```
; Генерация ACK
  BIS.B  #USIOE,&USICTL0     ; SDA - выход
  MOV.B  #00h,&USISRL        ; MSB = 0
  MOV.B  #01h,&USICNT        ; USICNTx = 1
TEST_USIIFG
  BIT.B  #USIIFG,&USICTL1    ; Проверяем USIIFG
  JZ     TEST_USIIFG
...продолжение...
; Генерация NACK
  BIS.B  #USIOE,&USICTL0     ; SDA - выход
  MOV.B  #0FFh,&USISRL       ; MSB = 1
  MOV.B  #01h,&USICNT        ; USICNTx = 1
TEST_USIIFG
  BIT.B  #USIIFG,&USICTL1    ; Проверяем USIIFG
  JZ     TEST_USIIFG
... продолжение...
```

Состояние СТАРТ

Состоянию СТАРТ соответствует изменение уровня на линии SDA с ВЫСОКОГО на НИЗКИЙ при ВЫСОКОМ уровне на линии SCL. Для формирования состояния СТАРТ необходимо записать 0 в старший бит сдвигового регистра и установить биты USIGE и USIOE для отключения защёлки на выходе сдвигового регистра. При этом старший бит сдвигового регистра сразу же появляется на выводе SDA, формируя на линии НИЗКИЙ уровень. После сброса бита USIGE работа защёлки возобновляется, а НИЗКИЙ уровень удерживается на линии SDA до выдачи следующего бита данных синхронно с сигналом на линии SCL.

```
; Формирование состояния СТАРТ
  MOV.B  #000h,&USISRL             ; MSB = 0
  BIS.B  #USIGE+USIOE,&USICTL0     ; Отключаем защёлку и активируем выход SDA
  BIC.B  #USIGE,&USICTL0           ; Включаем защёлку
... продолжение...
```

Состояние СТОП

Состоянию СТОП соответствует изменение уровня на линии SDA с НИЗКОГО на ВЫСОКИЙ при ВЫСОКОМ уровне на линии SCL. Для завершения передачи бита квитирования и выдачи на линию SDA НИЗКОГО уровня, т.е. для подготовки к формированию состояния СТОП, необходимо сбросить старший бит сдвигового регистра и загрузить 1 в счётчик USICNTx. При этом на линии SCL будет сформирован отрицательный импульс, по спадающему фронту которого на линию SDA будет выставлен НИЗКИЙ уровень. Поскольку модуль работает в режиме ведущего, на линии SCL останется ВЫСОКИЙ уровень. Для формирования состояния СТОП необходимо установить старший бит сдвигового регистра,

загрузить 1 в счётчик USICNTx и установить биты USIGE и USIOE для отключения защёлки. В результате старший бит сдвигового регистра сразу же появится на выводе SDA, формируя на линии ВЫСОКИЙ уровень. Последующий сброс бита USIGE сохранит старший бит сдвигового регистра в защёлке, а сброс бита USIOE отключит выход модуля. За счёт использования внешних подтягивающих резисторов ВЫСОКИЙ уровень на линии SDA будет удерживаться до формирования состояния СТАРТ.

```
; Формирование состояния СТОП
  BIS.B  #USIOE,&USICTL0          ; SDA - выход
  MOV.B  #000H,&USISRL            ; MSB = 0
  MOV.B  #001H,&USICNT            ; USICNT = 1 для 1 такта
TEST_USIIFG
  BIT.B  #USIIFG,&USICTL1         ; Проверяем USIIFG
  JZ     TEST_USIIFG              ;
  MOV.B  #0FFH,&USISRL            ; USISRL = 1 для выдачи на SDA ВЫСОКОГО уровня
  BIS.B  #USIGE,&USICTL0          ; Отключаем защёлку
  BIC.B  #USIGE+USIOE,&USICTL     ; Включаем защёлку и отключаем выход SDA
... продолжение...
```

Освобождение линии SCL

Если модуль USI удерживает на линии SCL НИЗКИЙ уровень, то её можно освободить, не сбрасывая флага USIIFG, путём установки бита USISCLREL. Бит USISCLREL будет сброшен автоматически при обнаружении состояния СТАРТ, после чего, начиная со следующего тактового импульса, на линии SCL будет удерживаться НИЗКИЙ уровень.

В режиме ведомого этот бит используется для того, чтобы предотвратить удержание НИЗКОГО уровня на линии SCL в том случае, если устройство не адресовано ведущим. При обнаружении следующего состояния СТАРТ бит USISCLREL будет сброшен, а флаг USISTTIFG — установлен.

Арбитраж

Модуль USI может обнаруживать ситуацию потери арбитража в системах с несколькими ведущими. В процессе арбитража на шине I^2C используются данные, выставляемые на шину SDA «конкурирующими» передатчиками. Ведущий-передатчик, выставляющий на линию SDA ВЫСОКИЙ уровень, теряет арбитраж, если другой передатчик одновременно выставит на линию SDA НИЗКИЙ уровень. Потеря арбитража обнаруживается модулем USI путём сравнения значения, выставляемого на шину, и значения, считываемого с шины. Если эти значения не равны, то фиксируется потеря арбитража и устанавливается флаг USIAL. Одновременно с этим сбрасывается бит USIOE, отключая модуль от шины. Таким образом, пользовательская программа должна контролировать состояние флагов USIAL и USIIFG, переключая модуль в режим ведомого-приёмника в случае потери арбитража. Флаг USIAL должен сбрасываться программно.

Чтобы предотвратить во время арбитража генерацию тактовых сигналов более быстрыми ведущими, на линии SCL удерживается НИЗКИЙ уровень в том случае, если другой ведущий, присутствующий на шине, выставляет на линию НИЗ-

КИЙ уровень и установлен флаг USIIG или USISTTIFG или же если USICNTx = 0.

Прерывания в режиме I^2C

Модулю USI выделен один вектор прерывания, с которым в режиме I^2C связаны два флага прерывания — USIIFG и USISTTIFG. Каждый флаг имеет собственный бит разрешения прерывания USIIE и USISTTIE соответственно. При разрешённом прерывании и установленном бите GIE установка любого флага прерывания генерирует запрос прерывания.

Флаг USIIFG устанавливается, когда содержимое счётчика USICNTx становится равным нулю в результате декрементирования либо в результате непосредственной загрузки нулевого значения. Флаг USIIFG сбрасывается при записи в биты USICNTx значения, отличного от нуля (если бит USIIFGCC = 0), или же непосредственно.

Флаг USISTTIFG устанавливается при обнаружении состояния СТАРТ. Флаг USISTTIFG должен сбрасываться программно.

Обнаружение состояния СТОП сопровождается установкой флага USISTP, однако этот флаг не вызывает генерации прерывания. Флаг USISTP сбрасывается при записи в биты USICNTx значения, отличного от нуля (если бит USIIFGCC = 0), или же непосредственно.

14.3. Регистры модуля USI

Список регистров модуля USI приведён в **Табл. 14.1**.

Таблица 14.1. **Регистры модуля USI**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USI	USICTL0	Чтение/запись	078h	01h после PUC
Регистр управления 1 модуля USI	USICTL1	Чтение/запись	079h	01h после PUC
Регистр управления системой синхронизации модуля USI	USICKCTL	Чтение/запись	07Ah	Сбрасывается после PUC
Регистр счётчика битов модуля USI	USICNT	Чтение/запись	07Bh	Сбрасывается после PUC
Сдвиговый регистр модуля USI, младший байт	USISRL	Чтение/запись	07Ch	Не изменяется
Сдвиговый регистр модуля USI, старший байт	USISRH	Чтение/запись	07Dh	Не изменяется

Ко всем регистрам модуля USI можно обращаться с использованием двухбайтных команд, как показано в **Табл. 14.2**.

Таблица 14.2. **Обращение к регистрам модуля USI с использованием двухбайтных команд**

Регистр	Обозначение	Старший байт	Младший байт	Адрес
Регистр управления модуля USI	USICTL	USICTL1	USICTL0	078h
Регистр управления системой синхронизации и счётчиком модуля USI	USICCTL	USICNT	USICKCTL	07Ah
Сдвиговый регистр модуля USI	USISR	USISRH	USISRL	07Ch

USICTL0, регистр управления 0 модуля USI

7	6	5	4	3	2	1	0
USIPE7	**USIPE6**	**USIPE5**	**USILSB**	**USIMST**	**USIGE**	**USIOE**	**USISWRST**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

USIPE7 Бит 7 Активация вывода SDI/SDA модуля USI. Вход в режиме SPI, вход или выход с открытым стоком в режиме I^2C.
- 0 Функция вывода модуля USI отключена
- 1 Функция вывода модуля USI включена

USIPE6 Бит 6 Активация вывода SDO/SCL модуля USI. Выход в режиме SPI, вход или выход с открытым стоком в режиме I^2C.
- 0 Функция вывода модуля USI отключена
- 1 Функция вывода модуля USI включена

USIPE5 Бит 5 Активация вывода SCLK модуля USI. Вход в режиме ведомого SPI или I^2C, выход в режиме ведущего SPI.
- 0 Функция вывода модуля USI отключена
- 1 Функция вывода модуля USI включена

USILSB Бит 4 Порядок передачи битов. Этот бит определяет направление сдвига принимаемых и передаваемых данных.
- 0 Старший бит первый
- 1 Младший бит первый

USIMST Бит 3 Выбор режима ведущего.
- 0 Режим ведомого
- 1 Режим ведущего

USIGE Бит 2 Управление регистром-защёлкой выхода.
- 0 Включение защёлки производится синхронно с тактовым сигналом
- 1 Защёлка включена постоянно («прозрачна»)

USIOE Бит 1 Разрешение вывода данных.
- 0 Выход отключен
- 1 Выход включен

USISWRST Бит 0 Программный сброс модуля USI.
- 0 Модуль USI готов к работе
- 1 Модуль USI находится в состоянии сброса

USICTL1, регистр управления 1 модуля USI

7	6	5	4	3	2	1	0
USICKPH	**USII2C**	**USISTTIE**	**USIIE**	**USIAL**	**USISTP**	**USISTTIFG**	**USIIFG**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

USICKPH Бит 7 Фаза тактового сигнала.
- 0 Данные выставляются по первому фронту сигнала SCLK, а считываются по следующему фронту
- 1 Данные считываются по первому фронту сигнала SCLK, а выставляются по следующему фронту

USII2C Бит 6 Включение режима I^2C.
- 0 Режим I^2C выключен
- 1 Режим I^2C включён

USISTTIE Бит 5 Разрешение прерывания при обнаружении состояния СТАРТ.
0 Прерывание запрещено
1 Прерывание разрешено

USIIE Бит 4 Разрешение прерывания от счётчика битов.
0 Прерывание запрещено
1 Прерывание разрешено

USIAL Бит 3 Потеря арбитража.
0 Не было потери арбитража
1 Арбитраж потерян

USISTP Бит 2 Обнаружено состояние СТОП. Бит USISTP автоматически сбрасывается при записи в биты USICNTx значения, отличного от нуля, если USIIFGCC = 0.
0 Состояние СТОП не обнаружено
1 Состояние СТОП обнаружено

USISTTIFG Бит 1 Флаг прерывания при обнаружении состояния СТАРТ.
0 Состояние СТАРТ не обнаружено. Прерывания не было
1 Состояние СТАРТ обнаружено. Прерывание было

USIIFG Бит 0 Флаг прерывания от счётчика битов. Устанавливается при USICNTx = 0. Автоматически сбрасывается при записи в биты USICNTx значения, отличного от нуля, если USIIFGCC = 0.
0 Прерывания не было
1 Прерывание было

USICKCTL, регистр управления системой синхронизации модуля USI

7	6	5	4	3	2	1	0
USIDIVx			**USISSELx**			**USICKPL**	**USISWCLK**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

USIDIVx Биты 7...5 Коэффициент деления тактового сигнала.
000 1
001 2
010 4
011 8
100 16
101 32
110 64
111 128

USISSELx Биты 4...2 Выбор источника тактового сигнала.
000 SCLK (не используется в режиме SPI)
001 ACLK
010 SMCLK
011 SMCLK
100 Бит USISWCLK
101 TACCR0
110 TACCR1
111 TACCR2 (зарезервировано в моделях MSP430F20xx)

USICKPL Бит 1 Полярность тактового сигнала.
0 Неактивное состояние — НИЗКИЙ уровень
1 Неактивное состояние — ВЫСОКИЙ уровень

USISWCLK Бит 0 Программное формирование тактового сигнала.
0 НИЗКИЙ уровень
1 ВЫСОКИЙ уровень

USICNT, регистр счётчика битов модуля USI

7	6	5	4	3	2	1	0
USISCLREL	**USI16B**	**USIIFGCC**	**USICNTx**				
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

USISCLREL Бит 7 Освобождение линии SCL. При USISCLREL = 1 с линии SCL снимается НИЗКИЙ уровень, и она переводится в состояние ожидания. Бит USISCLREL сбрасывается при обнаружении состояния СТАРТ.
0 На линии SCL удерживается НИЗКИЙ уровень, если USIIFG = 1
1 Линия SCL освобождена

USI16B Бит 6 Выбор размера сдвигового регистра.
0 8-битный регистр. Используется младший байт USISRL.
1 16-битный регистр. Используются оба регистра USISRL и USISRH. Регистр USISR адресует все 16 бит одновременно.

USIIFGCC Бит 5 Управление сбросом флага прерывания. Когда бит USIIFGCC = 1, флаг USIIFG не будет сбрасываться автоматически при записи в счётчик USICNTx ненулевого значения.
0 Флаг USIIFG автоматически сбрасывается при обновлении USICNTx
1 Флаг USIIFG не сбрасывается автоматически

USICNTx Биты 4...0 Счётчик битов.
Поле USICNTx содержит значение, равное числу принимаемых или передаваемых битов данных.

USISRL, сдвиговый регистр модуля USI, младший байт

7	6	5	4	3	2	1	0
USISRLx							
rw	rw	rw	rw	rw	rw	rw	rw

USISRLx Биты 7...0 Содержимое младшего байта сдвигового регистра.

USISRH, сдвиговый регистр модуля USI, старший байт

7	6	5	4	3	2	1	0
USISRHx							
rw	rw	rw	rw	rw	rw	rw	rw

USISRHx Биты 7...0 Содержимое старшего байта сдвигового регистра. Игнорируются при USI16B = 0.

ГЛАВА 15

УНИВЕРСАЛЬНЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ КОММУНИКАЦИОННЫЙ ИНТЕРФЕЙС: РЕЖИМ UART

Модуль универсального последовательного коммуникационного интерфейса (USCI) поддерживает несколько режимов обмена по последовательному каналу. В этой главе описывается работа модуля в режиме асинхронного приёмопередатчика UART.

15.1. Введение

Модули универсального последовательного коммуникационного интерфейса (USCI) поддерживают несколько режимов передачи данных по последовательному каналу, при этом разные модули USCI поддерживают различные режимы. Модули USCI имеют уникальное обозначение. Так, модуль USCI_A отличается от модуля USCI_B и т.д. Если в конкретном устройстве имеется более одного модуля USCI с идентичными функциями, то в обозначение таких модулей добавляется порядковый номер. Например, если в устройстве содержится два модуля USCI_A, то они имеют обозначения USCI_A0 и USCI_A1. Чтобы узнать, какие именно модули USCI реализованы и реализованы ли вообще в конкретном микроконтроллере, обратитесь к справочной документации.

Модули USCI_A поддерживают:

- режим UART;
- формирование импульсов для обмена по протоколу IrDA;
- автоматическое определение скорости обмена для поддержки протокола LIN;
- режим SPI.

Модули USCI_B поддерживают:

- режим I^2C;
- режим SPI.

15.2. Введение в модуль USCI: режим UART

В режиме асинхронного приёмопередатчика модули USCI_Ax позволяют подключать микроконтроллеры семейства MSP430 к внешним устройствам с помощью выводов UCAxRXD и UCAxTXD. Включение режима UART осуществляется сбросом бита UCSYNC.

Режим UART имеет следующие особенности:

- 7- или 8-битные данные с контролем чётности, нечётности или без контроля;
- независимые сдвиговые регистры передачи и приёма;
- раздельные буферные регистры передачи и приёма;
- изменяемый порядок передачи и приёма битов;
- встроенная поддержка коммуникационных протоколов idle-line (неактивная линия) и address-bit (адресный бит) для многопроцессорных систем;
- обнаружение приёмником фронта старт-бита для автоматического выхода из режимов пониженного энергопотребления LPMx;
- программируемая скорость обмена с использованием модуляции для поддержки дробных значений скорости;
- флаги обнаружения и игнорирования ошибок;
- флаг обнаружения адреса;
- независимые прерывания передачи и приёма.

Блок-схема модуля USCI_Ax, сконфигурированного для работы в режиме UART, приведена на **Рис. 15.1**.

15.3. Функционирование модуля USCI: режим UART

В режиме UART модуль USCI передаёт и принимает данные с заданной скоростью асинхронно по отношению к другому устройству. Синхронизация приёма/передачи каждого символа осуществляется на основе выбранной скорости обмена модуля. Блоки передачи и приёма используют одно и то же значение скорости обмена.

15.3.1. Инициализация и сброс модуля USCI

Модуль USCI сбрасывается по сигналу PUC или в результате установки бита USCSWRST. При появлении сигнала PUC бит USCSWRST автоматически устанавливается, переводя модуль в состояние сброса. При установке бита USCSWRST сбрасываются биты UCAxRXIE, UCAxTXIE, UCAxRXIFG, UCRXERR, UCBRK, UCPE, UCOE, UCFE, UCSTOE, UCBTOE и устанавливается бит UCAxTXIFG. Сброс бита USCSWRST переводит модуль USCI в рабочий режим.

Примечание. Инициализация и реконфигурирование модуля USCI

Инициализацию и реконфигурирование модуля USCI рекомендуется выполнять в следующей последовательности:

1. Установить бит UCSWRST (`BIS.B #UCSWRST,&UCAxCTL1`).
2. Инициализировать все регистры USCI установкой UCSWRST = 1 (включая регистр UCAxCTL1).
3. Сконфигурировать порты ввода/вывода.
4. Сбросить в программе бит UCSWRST (`BIC.B #UCSWRST,&UCAxCTL1`).
5. Разрешить прерывания (при необходимости) установкой битов UCAxRXIE и/или UCAxTXIE.

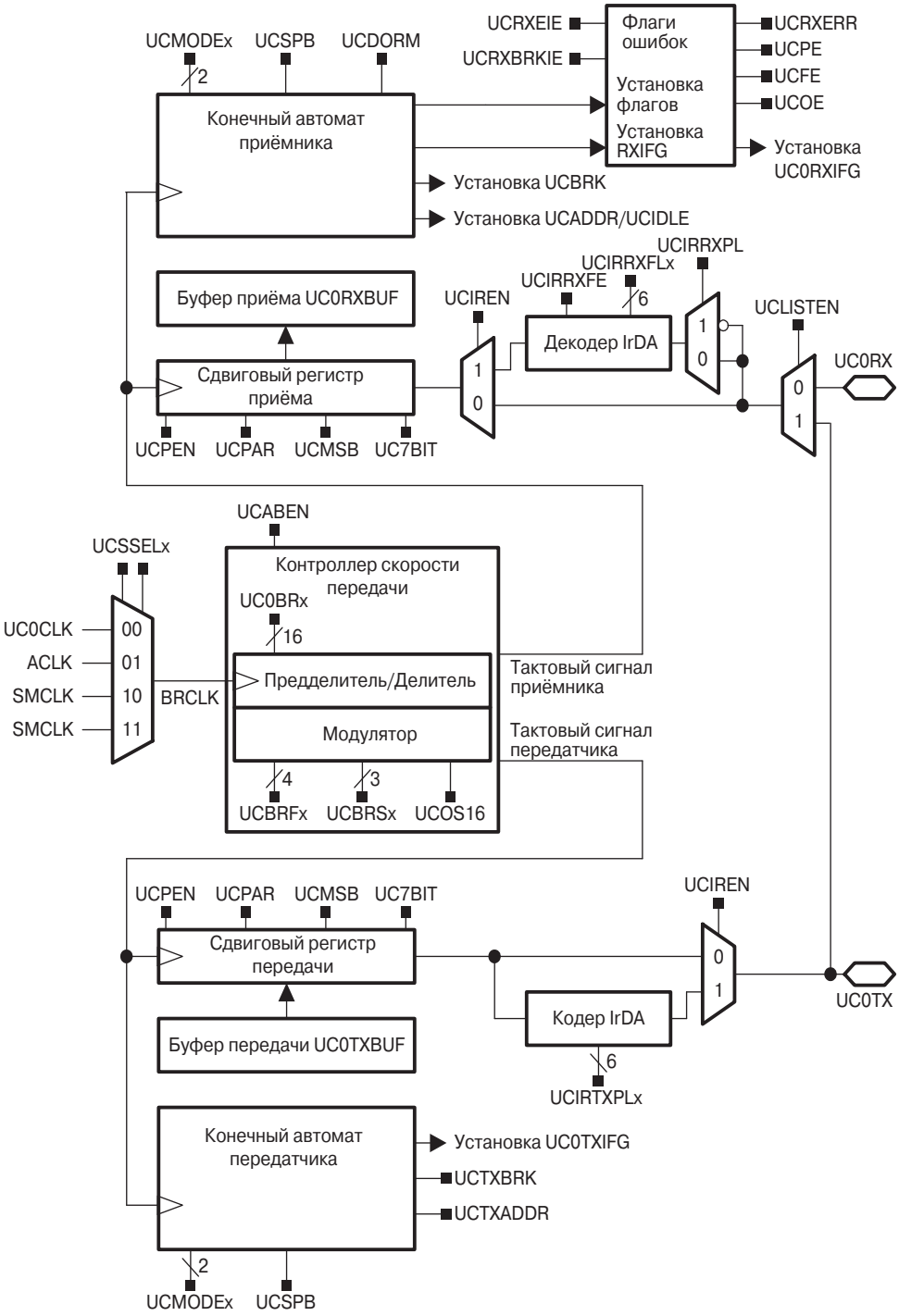


Рис. 15.1. Блок-схема модуля USCI_Ax: режим UART (UCSYNC = 0).

15.3.2. Формат символа

Как показано на **Рис. 15.2,** символ, передаваемый UART, содержит старт-бит, семь или восемь битов данных, бит чётности, бит адреса (в соответствующем режиме), а также один или два стоп-бита. Порядок передачи битов данных определяется битом UCMSB. При использовании UART данные обычно передаются начиная с младшего значащего бита.

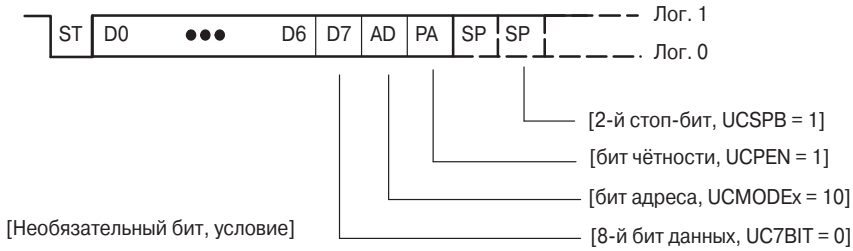


Рис. 15.2. Формат символа.

15.3.3. Форматы асинхронного обмена

При реализации асинхронного обмена между двумя устройствами от протокола обмена не требуется поддержка многопроцессорного режима работы. Однако для случаев, когда обмен осуществляется между тремя и более устройствами, модулем USCI поддерживаются два формата многопроцессорного обмена: idle-line (неактивная линия) и address-bit (адресный бит).

Формат многопроцессорного обмена idle-line

При UCMODEx = 01 выбирается формат многопроцессорного обмена idle-line. При использовании этого формата блоки данных разделяются периодом неактивности на линиях передачи или приёма, как показано на **Рис. 15.3**. Неактивное состояние линии приёма определяется в случае приёма 10 и более единичных битов после одного или двух стоп-битов символа. При обнаружении неактивного состояния линии контроллер скорости передачи выключается до появления переднего фронта старт-бита следующего символа. При обнаружении неактивного состояния линии устанавливается бит UCIDLE.

Первый символ, принятый после периода неактивности, является адресным. Бит UCIDLE служит в качестве адресной метки для каждого блока символов. При использовании формата idle-line этот бит находится в установленном состоянии, когда принятый символ содержит адрес.

В случае формата idle-line управление приёмом данных осуществляется с помощью бита UCDORM. При UCDORM = 1 все неадресные символы принимаются, но в регистр UCAxRXBUF они не загружаются, и прерывания не генерируются. При приёме адресного символа, если бит UCRXEIE = 1, этот символ помещается в регистр UCAxRXBUF, и устанавливается флаг UCAxRXIFG, а также соответствующие флаги ошибок (при их наличии). Если бит UCRXEIE = 0, то при обнаружении ошибки кадра или ошибки чётности во время приёма адресно-

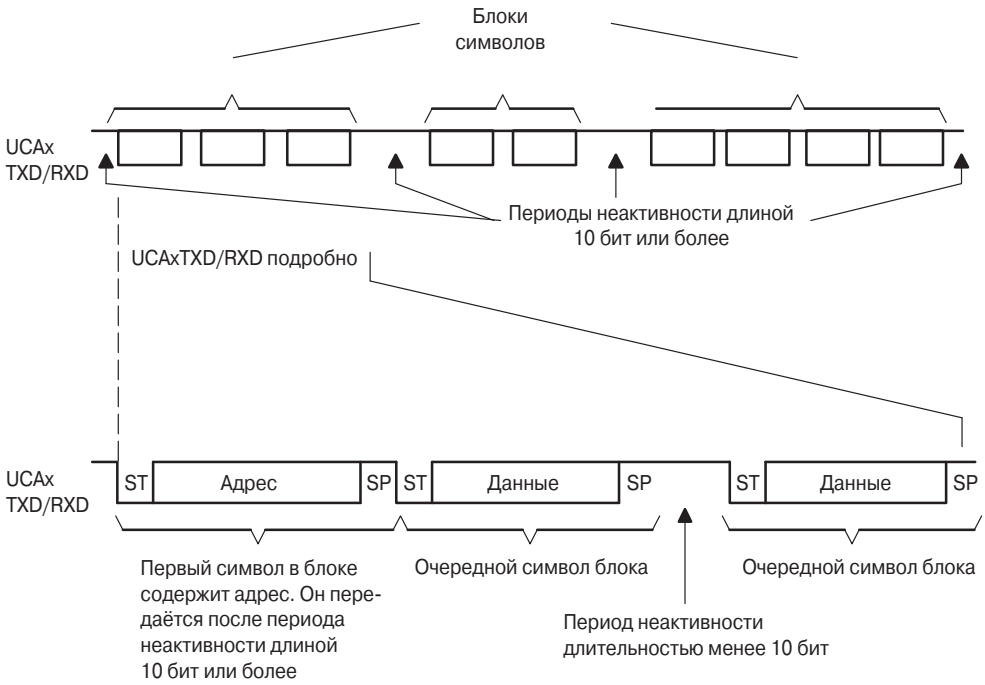


Рис. 15.3. Формат многопроцессорного обмена idle-line.

го символа, этот символ в регистр UCAxRXBUF не помещается, и флаг UCAxRXIFG не устанавливается.

После получения адреса пользовательская программа может проверить его значение и, при необходимости, сбросить бит UCDORM, чтобы продолжить приём данных. Если бит UCDORM останется установленным, то будут приниматься только адресные символы. При сбросе бита UCDORM во время приёма символа флаг прерывания будет установлен после завершения приёма. Состояние бита UCDORM не изменяется модулем USCI автоматически.

При передаче адреса модуль USCI может формировать период неактивности заданной длительности. Для этого используется флаг UCTXADDR, имеющий двойную буферизацию. Установленный флаг UCTXADDR информирует модуль о том, что перед передачей символа, загруженного в регистр UCTAxTXBUF, необходимо сформировать период неактивности длительностью 11 бит. Флаг UCTXADDR автоматически сбрасывается, когда генерируется старт-бит.

Формирование кадра неактивности

Следующая процедура формирует кадр неактивности, указывающий на то, что далее будет передаваться адресный символ, сопровождаемый соответствующими данными.

1. Устанавливается бит UCTXADDR, после чего в регистр UCAxTXBUF загружается адрес. Регистр UCAxTXBUF должен быть готов к приёму новых данных (UCAxTXIFG = 1).

В результате линия переводится в неактивное состояние на время, равное 11 битовым интервалам, после чего передаётся адресный символ. Флаг UCTXADDR сбрасывается автоматически при пересылке адресного символа из регистра UCAxTXBUF в сдвиговый регистр.

2. Требуемые символы данных поочерёдно загружаются в регистр UCAxTXBUF. Регистр UCAxTXBUF должен быть готов к приёму новых данных (UCAxTXIFG = 1).

   Данные, загружаемые в UCAxTXBUF, пересылаются в сдвиговый регистр и передаются по мере готовности сдвигового регистра к получению новых данных.

   Между передачей адреса и первого слова, также как и между передачей слов данных, не допускается появление задержек длительностью, превышающей длительность интервала неактивности. В противном случае переданное слово данных будет ошибочно интерпретировано как адрес.

Формат многопроцессорного обмена address-bit

При UCMODEx = 10 выбирается формат многопроцессорного обмена address-bit. При использовании этого формата все обрабатываемые символы содержат дополнительный бит, служащий признаком адреса, как показано на **Рис. 15.4**. Первый символ передаваемого блока содержит установленный бит адреса, указывающий на то, что данный символ является адресом. При приёме символа с установленным битом адреса и перемещении его в регистр UCAxRXBUF устанавливается бит UCADDR.

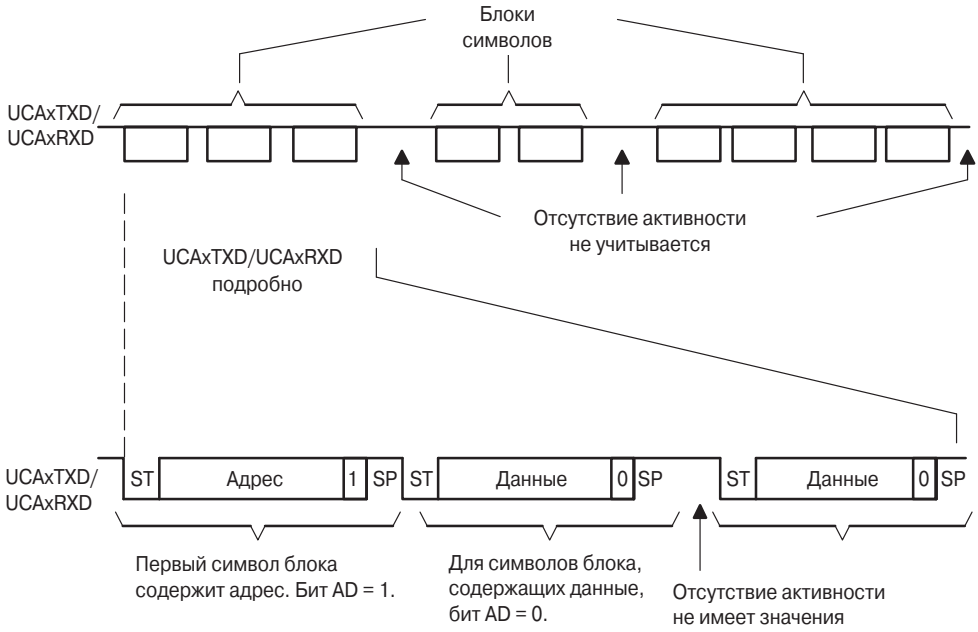


Рис. 15.4. Формат многопроцессорного обмена address-bit.

При использовании формата address-bit управление приёмом данных осуществляется битом UCDORM. При UCDORM = 1 все символы со сброшенным битом адреса принимаются, но в регистр UCAxRXBUF они не загружаются, и прерывания не генерируются. При приёме символа с установленным битом адреса, если бит UCRXEIE = 1, этот символ помещается в регистр UCAxRXBUF, и устанавливается флаг UCAxRXIFG, а также соответствующие флаги ошибок (при их наличии). Если бит UCRXEIE = 0, то при обнаружении ошибки кадра или ошибки чётности во время приёма символа с установленным битом адреса, этот символ в регистр UCAxRXBUF не помещается, и флаг UCAxRXIFG не устанавливается.

После получения адреса пользовательская программа может проверить его значение и, при необходимости, сбросить бит UCDORM, чтобы продолжить приём данных. Если бит UCDORM останется установленным, то будут приниматься только символы с установленным битом адреса. Состояние бита UCDORM не изменяется автоматически модулем USCI.

Когда бит UCDORM = 0, флаг UCAxRXIFG будет устанавливаться при приёме любого символа. При сбросе бита UCDORM во время приёма символа флаг прерывания будет установлен после завершения приёма.

На стороне передатчика значение бита адреса передаваемого символа определяется битом UCTXADDR. Значение бита UCTXADDR заносится в бит адреса символа, перегружаемого из регистра UCTAxTXBUF в сдвиговый регистр передачи. Бит UCTXADDR автоматически сбрасывается, когда генерируется старт-бит.

Обнаружение и формирование состояния обрыва связи

Когда UCMODEx = 00, 01 или 10, приёмник регистрирует обрыв связи, если все биты данных, бит чётности и все стоп-биты принятого символа оказываются сброшенными, независимо от режима контроля чётности, режима адресации и прочих установок. При обнаружении обрыва связи устанавливается бит UCBRK. При установленном бите разрешения прерывания UCBRKIE также будет установлен флаг прерывания приёма UCAxRXIFG. В регистре буфера UCAxRXBUF в этом случае содержится 00h, поскольку все биты данных были равны 0.

Для формирования состояния обрыва связи необходимо установить бит UCTXBRK, после чего загрузить 00h в регистр UCAxTXBUF. Регистр UCAxTXBUF должен быть готов к приёму новых данных (UCAxTXIFG = 1). В результате будет сформировано искомое состояние. Бит UCTXBRK автоматически сбрасывается, когда генерируется старт-бит.

15.3.4. Автоматическое определение скорости передачи

При UCMODEx = 11 выбирается режим работы UART с автоматическим определением скорости передачи. Для определения скорости передачи кадр данных предваряется синхронизирующей последовательностью, состоящей из поля паузы и поля синхронизации. Паузой считается приём 11 и более идущих подряд нулевых битов. Если длительность паузы оказывается больше длительности 22 битовых интервалов, то устанавливается флаг ошибки UCBTOE. После паузы передаётся поле синхронизации, как показано на **Рис. 15.5**.

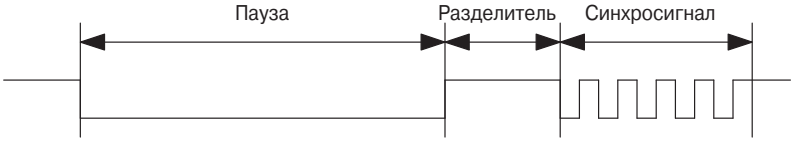


Рис. 15.5. Автоматическое определение скорости обмена.

Для совместимости с протоколом LIN символы должны иметь следующий формат: 8 бит данных, первым передаётся младший бит, контроль чётности отсутствует, один стоп-бит. Бит адреса не используется (отсутствует).

В битах данных поля синхронизации содержится число 055h, как показано на **Рис. 15.6**. Синхронизация основана на измерении длительности интервала между первым и последним спадающими фронтами. Автоматическое определение скорости обмена разрешается установкой бита UCABDEN, при этом для измерения используется контроллер скорости передачи. Если же автоматическое определение скорости не разрешено, то указанная битовая последовательность будет принята без измерения длительности. Результат измерения перегружается в регистры управления контроллера скорости обмена UCAxBR0, UCAxBR1 и UCAxMCTL. Если длина поля синхронизации окажется больше допустимой, то будет установлен флаг тайм-аута синхронизации UCSTOE.

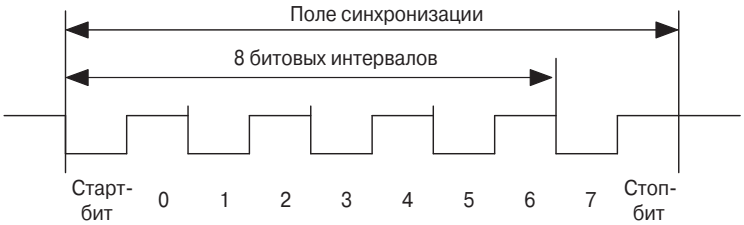


Рис. 15.6. Автоматическое определение скорости обмена – поле синхронизации.

Для управления приёмом данных в этом режиме используется бит UCDORM. При UCDORM = 1 принимаются все символы, но в регистр UCAxRXBUF они не загружаются, и прерывания не генерируются. При обнаружении синхронизирующей последовательности устанавливается флаг UCBRK. Символ, следующий за этой последовательностью, помещается в регистр UCAxRXBUF, и устанавливается флаг прерывания UCAxRXIFG, а также соответствующие флаги ошибок (при их наличии). При установленном бите UCBRKIE флаг UCAxRXIFG устанавливается и при приёме синхронизирующей последовательности. Бит USBRK сбрасывается программно или автоматически при чтении буфера приёма UCAxRXBUF.

После приёма синхронизирующей последовательности пользовательская программа должна сбросить бит UCDORM, чтобы продолжить приём данных. Если бит UCDORM останется установленным, то в дальнейшем будет принят только символ, переданный после следующей синхронизирующей последовательности. Состояние бита UCDORM не изменяется модулем USCI автоматически.

Когда бит UCDORM = 0, флаг UCAxRXIFG будет устанавливаться при приёме каждого символа. При сбросе бита UCDORM во время приёма символа флаг прерывания будет установлен после завершения приёма.

С некоторыми ограничениями режим автоматического определения скорости передачи можно использовать и в дуплексных системах. Во-первых, модуль UCSI не может передавать данные во время приёма синхронизирующей последовательности. Во-вторых, при возникновении ошибки кадра во время приёма байта со значением 00h все данные, переданные в это время, окажутся повреждёнными. Последнюю ситуацию можно обнаружить, проверяя принятые данные и бит UCFE.

Передача синхронизирующей последовательности

Для передачи синхронизирующей последовательности необходимо выполнить следующие действия:

1. Установить бит UCTXBRK при UMOCEx = 11.
2. Загрузить 055h в регистр UCAxTSBUF. Регистр UCAxTXBUF должен быть готов к приёму новых данных (UCAxTXIFG = 1).
   В результате будет сформировано поле паузы длительностью 13 бит, сопровождаемое разделителем и символом синхронизации. Длина разделителя определяется содержимым битов UCDELIMx. Бит UCTXBRK сбрасывается автоматически при пересылке символа синхронизации из регистра UCAxTXBUF в сдвиговый регистр передачи.
3. Поочерёдно загрузить требуемые символы данных в регистр UCAxTXBUF. Регистр UCAxTXBUF должен быть готов к приёму новых данных (UCAxTXIFG = 1).
   Данные, загружаемые в UCAxTXBUF, пересылаются в сдвиговый регистр и передаются по мере готовности сдвигового регистра к получению новых данных.

15.3.5. Кодирование и декодирование сигналов IrDA

При установке бита UCIREN включаются кодер и декодер IrDA, обеспечивающие формирование битовых последовательностей для обмена по протоколу IrDA.

Кодирование сигналов IrDA

Кодер IrDA формирует импульс для каждого нулевого бита в потоке битов, поступающем от UART, как показано на **Рис. 15.7**. Длительность импульса задаётся битами UCIRTXPLx, содержимое которых определяет число полупериодов тактового сигнала, заданного битом UCIRTXCLK.

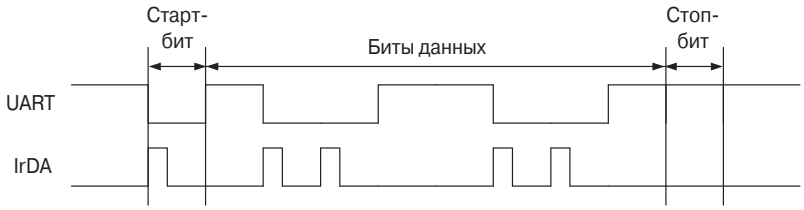


Рис. 15.7. UART и формат данных IrDA.

Для получения длительности импульса, равной 3/16 от длительности битового интервала, как того требует стандарт IrDA, выбирается тактовый сигнал BITCLK16 (UCIRTXCLK = 1), а длина импульса задаётся равной шести полупериодам тактового сигнала при UCIRTXPLx = 6 – 1 = 5.

При UCIRTXCLK = 0 длительность импульса t_{PULSE} определяется частотой сигнала BRCLK и вычисляется из следующего выражения:

$$UCIRTXPLx = t_{PULSE} \cdot 2 \cdot f_{BRCLK} - 1.$$

В случае, когда длительность импульса определяется сигналом BRCLK, содержимое регистра предделителя UCBRx должно быть больше или равно 5.

Декодирование сигналов IrDA

При UCIRRXPL = 0 декодер осуществляет обнаружение положительных импульсов, в противном случае — отрицательных. Установкой бита UCIRRXFE можно в дополнение к аналоговому помехоподавляющему фильтру включить программируемый цифровой фильтр. При установленном бите UCIRRXFE на вход декодера пропускаются только те импульсы, длительность которых больше заданной длины фильтра. Импульсы меньшей длительности игнорируются. Длина фильтра UCIRRXFLx определяется из выражения:

$$UCIRRXFLx = (t_{PULSE} - t_{WAKE}) \cdot 2 \cdot f_{BRCLK} - 4,$$

где: t_{PULSE} — минимальная длительность принимаемых импульсов;
t_{WAKE} — время выхода из режима пониженного энергопотребления. Оно равно нулю, если ЦПУ находится в активном режиме.

15.3.6. Автоматическое обнаружение ошибок

Подавление импульсных помех позволяет предотвратить случайный запуск модуля USCI. Любые импульсы на выводе UCAxRXD, длительность которых меньше t_τ (примерно 150 нс), будут игнорироваться. Точные значения параметра t_τ приводятся в справочной документации на конкретные модели.

При появлении на выводе UCAxRXD сигнала НИЗКОГО уровня длительностью свыше t_τ модуль USCI пытается определить наличие старт-бита, используя мажорирование. Если по результатам мажорирования старт-бит обнаружен не будет, то модуль USCI прекращает приём символа и снова переходит к ожиданию сигнала НИЗКОГО уровня на выводе UCAxRXD. Мажорирование также используется при чтении каждого бита слова.

При приёме символов модуль USCI автоматически обнаруживает ошибки кадра, ошибки чётности, ошибки переполнения, а также обрывы связи. Обнаружение тех или иных ошибок приводит к установке соответствующих битов UCFE, UCPE, UCOE и UCBRK. Одновременно с установкой любого из битов UCFE, UCPE или UCOE также устанавливается бит UCRXERR. Все ошибки приёма описаны в **Табл. 15.1**.

Таблица 15.1. **Ошибки приёма**

Ошибка	Флаг ошибки	Описание
Ошибка кадра	UCFE	Ошибка кадра возникает при обнаружении стоп-бита с нулевым значением. Если используются два стоп-бита, то проверяется значение обоих. При обнаружении ошибки кадра устанавливается бит UCFE
Ошибка чётности	UCPE	Ошибка чётности возникает при несоответствии числа единичных битов в слове и значения бита чётности. Если бит адреса присутствует в символе, то он тоже участвует в определении чётности. При обнаружении ошибки чётности устанавливается бит UCPE
Переполнение приёмника	UCOE	Ошибка переполнения возникает, если принятый символ помещается в регистр UCAxRXBUF до прочтения из него предыдущего. При обнаружении переполнения устанавливается бит UCOE
Обрыв связи	UCBRK	Если не используется автоматическое определение скорости передачи, то состояние обрыва связи регистрируется тогда, когда все биты данных, бит чётности и все стоп-биты принятого символа имеют нулевое значение. При обнаружении обрыва связи устанавливается бит UCBRK. Если установлен бит разрешения прерывания UCBRKIE, то при обнаружении обрыва связи также устанавливается флаг прерывания UCAxRXIFG

Если бит UCRXEIE = 0, то при обнаружении ошибки кадра или ошибки чётности принятый символ не загружается в регистр UCAxRXBUF. При UCRXEIE = 1 все принимаемые символы помещаются в регистр UCAxRXBUF независимо от наличия ошибок.

При установке любого из битов UCFE, UCPE, UCOE, UCBRK и UCRXERR он остаётся в этом состоянии до тех пор, пока не будет сброшен программно или в результате чтения регистра UCAxRXBUF. Бит UCOE должен сбрасываться только путём чтения регистра UCAxRXBUF, в противном случае этот бит не будет работать правильно. Для надёжного обнаружения переполнения рекомендуется следующая последовательность действий. После приёма символа и установки флага UCAxRXIFG сначала выполняется чтение регистра UCAxSTAT для проверки флагов ошибок, включая флаг переполнения UCOE. Затем выполняется чтение регистра UCAxRXBUF. При этом будут сброшены все флаги ошибок, за исключением UCOE, если между операциями чтения регистров UCAxSTAT и UCAxRXBUF в последний было записано новое значение. Для обнаружения такой ситуации необходимо проверять состояние флага UCOE после чтения регистра UCAxRXBUF. Обратите внимание — флаг UCRXERR в данном случае не устанавливается.

15.3.7. Разрешение приёма USCI

После включения модуля USCI сбросом бита UCSWRST приёмник готов к работе и находится в состоянии ожидания. Контроллер скорости передачи находится в состоянии готовности, но не генерирует никаких тактовых сигналов, поскольку задающий тактовый сигнал на него не поступает.

По спадающему фронту старт-бита разрешается работа контроллера скорости передачи, а конечный автомат UART проверяет наличие корректного старт-бита. При отсутствии корректного старт-бита конечный автомат переходит в состояние ожидания, а контроллер скорости передачи снова выключается. В случае обнаружения корректного старт-бита будет принят очередной символ данных.

В режиме многопроцессорного обмена idle-line (UCMODEx = 01) конечный автомат UART после приёма символа контролирует наличие активности на линии. При обнаружении старт-бита осуществляется приём очередного символа. В противном случае после приёма 10 подряд идущих единичных битов устанавливается флаг UCIDLE, конечный автомат UART переходит в состояние ожидания, а контроллер скорости передачи выключается.

Подавление импульсных помех при приёме

Подавление импульсных помех позволяет предотвратить случайный запуск модуля USCI. Любые импульсы на выводе UCAxRXD, длительность которых меньше t_τ (примерно 150 нс), будут игнорироваться модулем, как показано на **Рис. 15.8**. Точные значения параметра t_τ приводятся в справочной документации на конкретные модели.

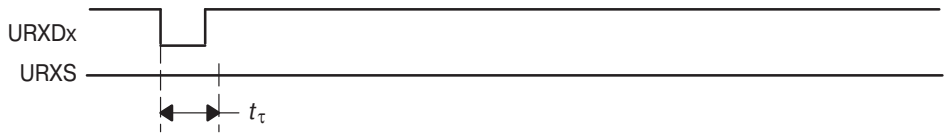


Рис. 15.8. Подавление импульсной помехи, приёмник USCI не запускается.

При появлении на выводе UCAxRXD импульса длительностью больше t_τ, приёмник модуля USCI пытается определить наличие старт-бита, используя мажорирование, как показано на **Рис. 15.9**. Если старт-бит обнаружен не будет, то модуль USCI прекращает приём символа.

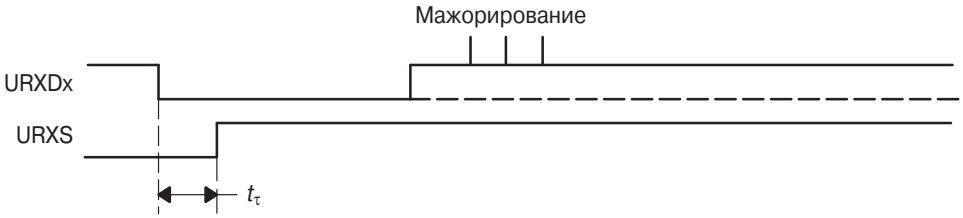


Рис. 15.9. Подавление импульсной помехи, модуль USCI активирован.

15.3.8. Разрешение передачи USCI

После включения модуля USCI сбросом бита UCSWRST передатчик готов к работе и находится в состоянии ожидания. Контроллер скорости передачи находится в состоянии готовности, но не генерирует никаких тактовых сигналов, поскольку задающий тактовый сигнал на него не поступает.

Передача инициируется записью значения в регистр UCAxTXBUF. При этом включается контроллер скорости передачи, а содержимое регистра пересылается в сдвиговый регистр передачи по следующему после опустошения сдвигового регистра такту сигнала BITCLK. После того как регистр UCAxTXBUF будет готов к получению новых данных, устанавливается флаг UCAxTXIFG.

Процесс передачи осуществляется до тех пор, пока имеются данные, загружаемые в регистр UCAxTXBUF до завершения передачи предыдущего значения. Если к моменту завершения передачи очередного значения регистр UCAxTXBUF оказывается пустым, то передатчик возвращается в состояние ожидания, а контроллер скорости передачи выключается.

15.3.9. Контроллер скорости передачи UART

Контроллер скорости передачи модуля USCI обеспечивает стандартные скорости обмена при использовании опорных тактовых сигналов нестандартных частот. Контроллер может работать в двух режимах, определяемых битом UCOS16.

Низкочастотный режим

Низкочастотный режим выбирается при UCOS16 = 0. Этот режим позволяет осуществлять обмен при использовании низкочастотного задающего сигнала (скажем, со скоростью 9600 бод при использовании кварцевого резонатора частотой 32 768 Гц). Использование задающего сигнала низкой частоты уменьшает потребление модуля. Данный режим можно использовать и с сигналами более высокой частоты при задании более высоких коэффициентов деления предделителя, однако при этом для мажорирования будет использоваться временное окно гораздо меньшего размера, что сведёт на нет все преимущества, предоставляемые указанным методом.

В низкочастотном режиме контроллер скорости передачи использует один предделитель и один модулятор для формирования сигнала битовой синхронизации. Такое сочетание позволяет получать дробные коэффициенты деления для обеспечения заданной скорости обмена. Максимальная скорость обмена в этом режиме составляет одну треть от частоты тактового сигнала UART BRCLK.

Временные диаграммы для одного бита приведены на **Рис. 15.10**. Значение каждого принятого бита определяется путём мажорирования. Выборки осуществляются в периоды $N/2 - 1/2$, $N/2$ и $N/2 + 1/2$ сигнала BRCLK, где N — число периодов BRCLK, укладывающихся в один период BITCLK.

Модуляция определяется содержимым битов UCBRSx, как показано в **Табл. 15.2**. Значение 1 в таблице говорит о том, что $m = 1$ и соответствующий период сигнала BITCLK на один период BRCLK больше, чем при $m = 0$. Модуляция циклически повторяется после 8 бит и перезапускается при появлении очередного старт-бита.

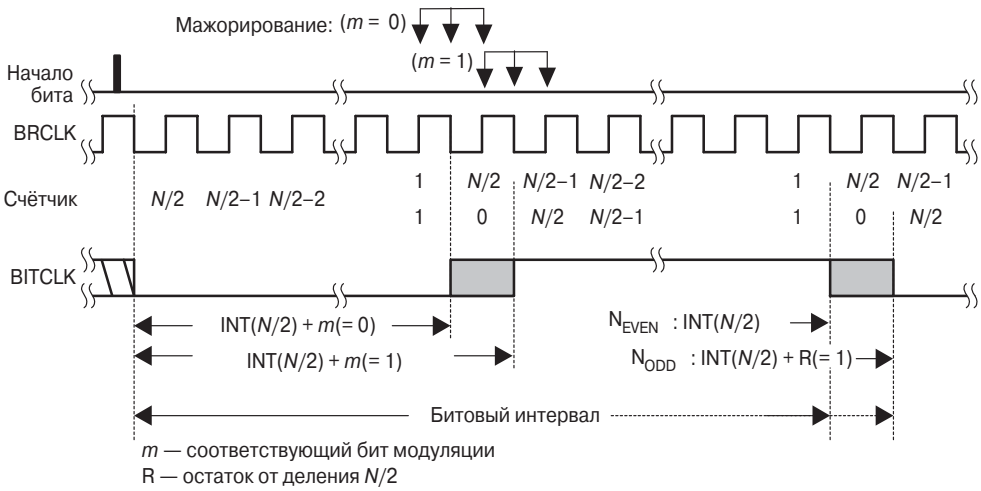


Рис. 15.10. Тактовый сигнал BITCLK при UCOS16 = 0.

Таблица 15.2. **Модуляция BITCLK**

UCBRSx	Бит 0 (старт-бит)	Бит 1	Бит 2	Бит 3	Бит 4	Бит 5	Бит 6	Бит 7
0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0
3	0	1	0	1	0	1	0	0
4	0	1	0	1	0	1	0	1
5	0	1	1	1	0	1	0	1
6	0	1	1	1	0	1	1	1
7	0	1	1	1	1	1	1	1

Высокочастотный режим

Высокочастотный режим выбирается при UCOS16 = 1. Этот режим поддерживает выборку битового потока UART при использовании задающего тактового сигнала большей частоты. В результате выборка сигнала для мажорирования всегда производится с интервалом, равным 1/16 периода сигнала битовой синхронизации. Кроме того, при включённых кодере и декодере IrDA этот режим облегчает формирование и приём импульсов стандарта IrDA длительностью 3/16 битового интервала.

В данном режиме используется один предделитель и один модулятор для генерации сигнала BITCLK16, частота которого в 16 раз больше частоты BITCLK. Дополнительные делитель и модулятор используются для формирования из BITCLK16 сигнала BITCLK. Такое сочетание позволяет получать дробные коэффициенты деления сигналов BITCLK и BITCLK16 для обеспечения заданной скорости обмена. Максимальная скорость обмена в этом режиме составляет 1/16 от частоты тактового сигнала UART BRCLK. Если биты UCBRx = 0 или 1, то пер-

вая секция модулятора/предделителя не используется и сигнал BRCLK идентичен сигналу BITCLK16.

Модуляция сигнала BITCLK16 определяется содержимым битов UCBRFx, как показано в **Табл. 15.3**. Значение 1 в таблице говорит о том, что $m = 1$ и соответствующий период сигнала BITCLK16 на один период BRCLK больше, чем при $m = 0$. Модуляция перезапускается в начале каждого битового интервала.

Модуляция сигнала BITCLK определяется содержимым битов UCBRSx (**Табл. 15.3**), как было описано выше.

Таблица 15.3. **Модуляция BITCLK16**

UCBRFx	Число тактов BITCLK16 после последнего спадающего фронта BITCLK															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
00h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
01h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
02h	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
03h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
04h	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1
05h	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1
06h	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1
07h	0	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1
08h	0	1	1	1	1	0	0	0	0	0	0	0	1	1	1	1
09h	0	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1
0Ah	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1
0Bh	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1
0Ch	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1
0Dh	0	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
0Eh	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
0Fh	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

15.3.10. Установка скорости обмена

Для данного тактового сигнала BRCLK используемая скорость обмена определяет требуемое значение коэффициента деления N:

$$N = \frac{f_{\text{BRCLK}}}{\text{Baud rate}}.$$

Коэффициент деления N часто является дробным числом, поэтому для максимально точного учёта значения N используется, как минимум, одна секция делителя и модулятора.

Если N больше или равно 16, то можно использовать высокочастотный режим, установив бит UCOS16.

Установки при использовании низкочастотного режима

В низкочастотном режиме целая часть коэффициента деления реализуется предделителем:

$$\text{UCBRx} = \text{INT}(N),$$

а дробная часть реализуется модулятором, установки которого определяются выражением:

$$\text{UCBRSx} = \text{round}((N - \text{INT}(N)) \cdot 8).$$

Инкрементирование или декрементирование значения UCBRSx иногда позволяет уменьшить максимальную частоту битовых ошибок для любого конкретного бита. Чтобы узнать, справедливо ли это в данном случае, необходимо определить величину ошибки для каждого бита при всех значениях UCBRSx.

Установки при использовании высокочастотного режима

В высокочастотном режиме установки пределителя определяются формулой:

$$\text{UCBRx} = \text{INT}(N/16),$$

а установки модулятора первой секции определяются выражением:

$$\text{UCBRFx} = \text{round}(((N/16) - \text{INT}(N/16)) \cdot 16).$$

Если необходимо получить бóльшую точность, то можно дополнительно использовать второй модулятор. Чтобы определить установки, обеспечивающие наименьшее значение максимальной частоты битовых ошибок, необходимо определить величину ошибки для всех значений UCBRSx из диапазона 0...7 при начальных установках UCBRFx, а также при увеличенном и уменьшенном на 1 значениях UCBRFx.

15.3.11. Синхронизация при передаче

Синхронизация при передаче любого символа осуществляется отдельно для каждого бита. Использование модуляции позволяет уменьшить накопленную битовую ошибку. Значения битовых ошибок для отдельных битов можно вычислить по приведённым ниже формулам.

Синхронизация в низкочастотном режиме

В низкочастотном режиме время передачи i-го бита $T_{\text{bit,TX}}[i]$ определяется содержимым битов UCBRx и UCBRSx:

$$T_{\text{bit,TX}}[i] = \frac{1}{f_{\text{BRCLK}}}\left(\text{UCBRx} + m_{\text{UCBRSx}}[i]\right),$$

где $m_{\text{UCBRSx}}[i]$ — значение модуляции для i-го бита из **Табл. 15.2**.

Синхронизация в высокочастотном режиме

В высокочастотном режиме время передачи i-го бита $T_{\text{bit,TX}}[i]$ определяется содержимым битов UCBRx, UCBRFx и UCBRSx:

$$T_{\text{bit,TX}}[i] = \frac{1}{f_{\text{BRCLK}}}\left((16 + m_{\text{UCBRSx}}[i]) \cdot \text{UCBRx} + \sum_{j=0}^{15} m_{\text{UCBRFx}}[j]\right),$$

где $\sum_{j=0}^{15} m_{\text{UCBRFx}}[j]$ — сумма единичных значений соответствующей строки **Табл. 15.3**;

$m_{\text{UCBRSx}}[i]$ — значение модуляции для i-го бита из **Табл. 15.2**.

Время завершения передачи очередного бита $t_{\text{bit,TX}}[i]$ равно сумме всех предыдущих и текущего битовых интервалов:

$$t_{\text{bit,TX}}[i] = \sum_{j=0}^{i} T_{\text{bit,TX}}[j].$$

Для определения величины битовой ошибки это время сравнивается с идеальным значением битового интервала $t_{\text{bit,ideal,TX}}[i]$:

$$t_{\text{bit,ideal,TX}}[i] = \frac{1}{\text{Baud rate}}(i+1).$$

В конечном итоге ошибка, приведённая к величине идеального битового интервала (1/Baud rate), получается равной:

$$\text{Error}_{\text{TX}}[i] = (t_{\text{bit,TX}}[i] - t_{\text{bit,ideal,TX}}[i]) \cdot \text{Baud rate} \cdot 100\%.$$

15.3.12. Синхронизация при приёме

Существует два источника ошибок во время приёма. Первый источник — это побитовая ошибка, аналогичная ошибке битовой синхронизации при передаче. Второй источник ошибок — это задержка между появлением фронта старт-бита и распознаванием этого фронта модулем USCI. На **Рис. 15.11** показаны асинхронные ошибки синхронизации между сигналом на выводе UCAxRXD и внутренним тактовым сигналом, формируемым контроллером скорости передачи. Ошибка синхронизации t_{SYNC} находится в пределах от $-1/2$ до $+1/2$ периода BRCLK независимо от выбранного режима работы контроллера скорости передачи.

Идеальным моментом выборки входного сигнала $t_{\text{bit,ideal,RX}}[i]$ является середина битового периода:

$$t_{\text{bit,ideal,RX}}[i] = \frac{1}{\text{Baud rate}}(i+0.5).$$

Реальное же время выборки $t_{\text{bit,RX}}[i]$ равно сумме битовых интервалов для всех предыдущих битов (см. формулы предыдущего подраздела) плюс половина периода BITCLK для текущего бита i, плюс ошибка синхронизации t_{SYNC}.

Таким образом, для низкочастотного режима работы контроллера скорости передачи время выборки бита $t_{\text{bit,RX}}[i]$ определяется выражением:

$$t_{\text{bit,RX}}[i] = t_{\text{SYNC}} + \sum_{j=0}^{i-1} T_{\text{bit,RX}}[j] + \frac{1}{f_{\text{BRCLK}}}\left(\text{INT}\left(\tfrac{1}{2}\text{UCBRx}\right) + m_{\text{UCBRSx}}[i]\right),$$

где $T_{\text{bit,RX}}[i] = \frac{1}{f_{\text{BRCLK}}}\left(\text{UCBRx} + m_{\text{UCBRSx}}[i]\right)$;

$m_{\text{UCBRSx}}[i]$ — значение модуляции для i-го бита из **Табл. 15.2**.

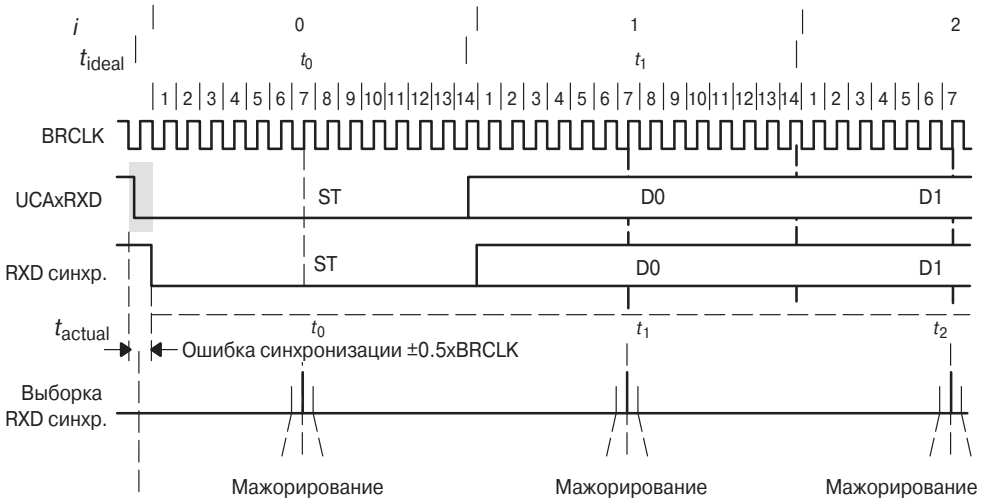


Рис. 15.11. Ошибка приёма.

Для высокочастотного режима работы время выборки бита $t_{\text{bit,RX}}[i]$ определяется выражением:

$$t_{\text{bit,RX}}[i] = t_{\text{SYNC}} + \sum_{j=0}^{i-1} T_{\text{bit,RX}}[j] + \frac{1}{f_{\text{BRCLK}}}\left(\left(8 + m_{\text{UCBRSx}}[i]\right)\cdot \text{UCBRx} + \sum_{j=0}^{7+m_{\text{UCBRSx}}[i]} m_{\text{UCBRFx}}[j]\right),$$

где $$T_{\text{bit,RX}}[i] = \frac{1}{f_{\text{BRCLK}}}\left(\left(16 + m_{\text{UCBRSx}}[i]\right)\cdot \text{UCBRx} + \sum_{j=0}^{15} m_{\text{UCBRFx}}[j]\right);$$

$\sum_{j=0}^{7+m_{\text{UCBRSx}}[i]} m_{\text{UCBRFx}}[j]$ — сумма единичных значений из столбцов 0...7 + $m_{\text{UCBRSx}}[i]$ соответствующей строки **Табл. 15.3**;

$m_{\text{UCBRSx}}[i]$ — значение модуляции для i-го бита из **Табл. 15.2**.

В конечном итоге ошибка, приведённая к величине идеального битового интервала (1/Baud rate), получается равной:

$$\text{Error}_{\text{RX}}[i] = (t_{\text{bit,RX}}[i] - t_{\text{bit,ideal,RX}}[i])\cdot \text{Baud rate}\cdot 100\%.$$

15.3.13. Типовые скорости обмена и величины ошибок

Установки битов UCBRx, UCBRSx и UCBRFx, необходимые для получения стандартных скоростей обмена при использовании тактового сигнала ACLK частотой 32 768 Гц (от кварцевого резонатора) и тактового сигнала SCLK, имеющего типичные значения частоты, приведены в **Табл. 15.4** и **Табл. 15.5**. Убедитесь, что частота выбранного источника BRCLK не превышает максимально допустимое значение частоты тактового сигнала для модуля USCI (см. справочную документацию на конкретную модель).

Таблица 15.4. **Установки и значения ошибок для типовых скоростей обмена, UCOS16 = 0**

Частота BRCLK [Гц]	Скорость обмена [бод]	UCBRx	UCBRSx	UCBRFx	Максимальная ошибка передачи [%]		Максимальная ошибка приёма [%]	
32 768	**1200**	27	2	0	–2.8	1.4	–5.9	2.0
32 768	**2400**	13	6	0	–4.8	6.0	–9.7	8.3
32 768	**4800**	6	7	0	–12.1	5.7	–13.4	19.0
32 768	**9600**	3	3	0	–21.1	15.2	–44.3	21.3
1 048 576	**9600**	109	2	0	–0.2	0.7	–1.0	0.8
1 048 576	**19 200**	54	5	0	–1.1	1.0	–1.5	2.5
1 048 576	**38 400**	27	2	0	–2.8	1.4	–5.9	2.0
1 048 576	**56 000**	18	6	0	–3.9	1.1	–4.6	5.7
1 048 576	**115 200**	9	1	0	–1.1	10.7	–11.5	11.3
1 048 576	**128 000**	8	1	0	–8.9	7.5	–13.8	14.8
1 048 576	**256 000**	4	1	0	–2.3	25.4	–13.4	38.8
1 000 000	**9600**	104	1	0	–0.5	0.6	–0.9	1.2
1 000 000	**19 200**	52	0	0	–1.8	0	–2.6	0.9
1 000 000	**38 400**	26	0	0	–1.8	0	–3.6	1.8
1 000 000	**56 000**	17	7	0	–4.8	0.8	–8.0	3.2
1 000 000	**115 000**	8	6	0	–7.8	6.4	–9.7	16.1
1 000 000	**128 000**	7	7	0	–10.4	6.4	–18.0	11.6
1 000 000	**256 000**	3	7	0	–29.6	0	–43.6	5.2
4 000 000	**9600**	416	6	0	–0.2	0.2	–0.2	0.4
4 000 000	**19 200**	208	3	0	–0.2	0.5	–0.3	0.8
4 000 000	**38 400**	104	1	0	–0.5	0.6	–0.9	1.2
4 000 000	**56 000**	71	4	0	–0.6	1.0	–1.7	1.3
4 000 000	**115 200**	34	6	0	–2.1	0.6	–2.5	3.1
4 000 000	**128 000**	31	2	0	–0.8	1.6	–3.6	2.0
4 000 000	**256 000**	15	5	0	–4.0	3.2	–8.4	5.2
8 000 000	**9600**	833	2	0	–0.1	0	–0.2	0.1
8 000 000	**19 200**	416	6	0	–0.2	0.2	–0.2	0.4
8 000 000	**38 400**	208	3	0	–0.2	0.5	–0.3	0.8
8 000 000	**56 000**	142	7	0	–0.6	0.1	–0.7	0.8
8 000 000	**115 200**	69	4	0	–0.6	0.8	–1.8	1.1
8 000 000	**128 000**	62	4	0	–0.8	0	–1.2	1.2
8 000 000	**256 000**	31	2	0	–0.8	1.6	–3.6	2.0
12 000 000	**9600**	1250	0	0	0	0	–0.05	0.05
12 000 000	**19 200**	625	0	0	0	0	–0.2	0
12 000 000	**38 400**	312	4	0	–0.2	0	–0.2	0.2
12 000 000	**56 000**	214	2	0	–0.3	0.2	–0.4	0.5
12 000 000	**115 200**	104	1	0	–0.5	0.6	–0.9	1.2
12 000 000	**128 000**	93	6	0	–0.8	0	–1.5	0.4
12 000 000	**256 000**	46	7	0	–1.9	0	–2.0	2.0
16 000 000	**9600**	1666	6	0	–0.05	0.05	–0.05	0.1
16 000 000	**19 200**	833	2	0	–0.1	0.05	–0.2	0.1
16 000 000	**38 400**	416	6	0	–0.2	0.2	–0.2	0.4
16 000 000	**56 000**	285	6	0	–0.3	0.1	–0.5	0.2
16 000 000	**115 200**	138	7	0	–0.7	0	–0.8	0.6
16 000 000	**128 000**	125	0	0	0	0	–0.8	0
16 000 000	**256 000**	62	4	0	–0.8	0	–1.2	1.2

Таблица 15.5. **Установки и значения ошибок для типовых скоростей обмена, UCOS16 = 1**

Частота BRCLK [Гц]	Скорость обмена [бод]	UCBRx	UCBRSx	UCBRFx	Максимальная ошибка передачи [%]		Максимальная ошибка приёма [%]	
1 048 576	**9600**	6	0	13	–2.3	0	–2.2	0.8
1 048 576	**19 200**	3	1	6	–4.6	3.2	–5.0	4.7
1 000 000	**9600**	6	0	8	–1.8	0	–2.2	0.4
1 000 000	**19 200**	3	0	4	–1.8	0	–2.6	0.9
1 000 000	**57 600**	1	7	0	–34.4	0	–33.4	0
4 000 000	**9600**	26	0	1	0	0.9	0	1.1
4 000 000	**19 200**	13	0	0	–1.8	0	–1.9	0.2
4 000 000	**38 400**	6	0	8	–1.8	0	–2.2	0.4
4 000 000	**57 600**	4	5	3	–3.5	3.2	–1.8	6.4
4 000 000	**115 200**	2	3	2	–2.1	4.8	–2.5	7.3
4 000 000	**230 400**	1	7	0	–34.4	0	–33.4	0
8 000 000	**9600**	52	0	1	–0.4	0	–0.4	0.1
8 000 000	**19 200**	26	0	1	0	0.9	0	1.1
8 000 000	**38 400**	13	0	0	–1.8	0	–1.9	0.2
8 000 000	**57 600**	8	0	11	0	0.88	0	1.6
8 000 000	**115 200**	4	5	3	–3.5	3.2	–1.8	6.4
8 000 000	**230 400**	2	3	2	–2.1	4.8	–2.5	7.3
8 000 000	**460 800**	1	7	0	–34.4	0	–33.4	0
12 000 000	**9600**	78	0	2	0	0	–0.05	0.05
12 000 000	**19 200**	39	0	1	0	0	0	0.2
12 000 000	**38 400**	19	0	8	–1.8	0	–1.8	0.1
12 000 000	**57 600**	13	0	0	–1.8	0	–1.9	0.2
12 000 000	**115 200**	6	0	8	–1.8	0	–2.2	0.4
12 000 000	**230 400**	3	0	4	–1.8	0	–2.6	0.9
16 000 000	**9600**	104	0	3	0	0.2	0	0.3
16 000 000	**19 200**	52	0	1	–0.4	0	–0.4	0.1
16 000 000	**38 400**	26	0	1	0	0.9	0	1.1
16 000 000	**57 600**	17	0	6	0	0.9	–0.1	1.0
16 000 000	**115 200**	8	0	11	0	0.9	0	1.6
16 000 000	**230 400**	4	5	3	–3.5	3.2	–1.8	6.4
16 000 000	**460 800**	2	3	2	–2.1	4.8	–2.5	7.3

Ошибка приёма — это отношение накопленного времени к идеальному времени выборки (в середине битового интервала). Значения ошибок приведены для случая приёма 8-битных символов с контролем чётности и одним стоп-битом с учётом ошибки синхронизации.

Ошибка передачи — это отношение накопленной ошибки синхронизации к длительности идеального битового интервала. Значения ошибок приведены для случая передачи 8-битных символов с контролем чётности и одним стоп-битом.

15.3.14. Использование модуля USCI в режиме UART совместно с режимами пониженного энергопотребления

Модуль USCI обеспечивает автоматическую активацию тактового сигнала SMCLK, что делает возможным использование модуля в режимах пониженного энергопотребления. Если модуль USCI тактируется сигналом SMCLK, который неактивен по причине нахождения устройства в режиме пониженного энергопотребления, то модуль автоматически активирует данный тактовый сигнал, независимо от значений управляющих битов выбора источника системного тактового сигнала. Указанный тактовый сигнал будет активным до тех пор, пока модуль USCI не вернётся в состояние ожидания. После этого контроль над источником тактового сигнала вернётся к модулю синхронизации. Автоматическая активация тактового сигнала ACLK не предусмотрена.

При активации модулем USCI неактивного источника тактового сигнала этот источник становится доступным для всего устройства, так что его активация может затронуть любой периферийный модуль, сконфигурированный для использования данного тактового сигнала. Например, значение таймера, использующего SMCLK, будет инкрементироваться в течение всего времени, пока модуль USCI будет находиться в активном режиме.

15.3.15. Прерывания модуля USCI

Модуль USCI имеет по одному вектору прерывания для передачи и приёма.

Прерывание USCI при передаче

Флаг прерывания UCAxTXIFG, устанавливаемый передатчиком, показывает готовность регистра UCAxTXBUF к загрузке нового символа. Если установлены биты GIE и UCAxTXIE, то при установке флага UCAxTXIFG генерируется запрос прерывания. Флаг UCAxTXIFG автоматически сбрасывается при записи в регистр UCAxTXBUF.

Флаг UCAxTXIFG устанавливается после сигнала сброса PUC или при UCSWRST = 1. Бит разрешения прерывания UCAxTXIE сбрасывается после сигнала сброса PUC или при UCSWRST = 1.

Прерывание USCI при приёме

Флаг прерывания UCAxRXIFG устанавливается каждый раз при приёме символа и загрузке его в регистр UCAxRXBUF. При установленных битах GIE и UCAxRXIE генерируется запрос прерывания. Биты UCAxRXIFG и UCAxRXIE сбрасываются сигналом системного сброса PUC или при UCSWRST = 1. Флаг UCAxRXIFG автоматически сбрасывается при чтении регистра UCAxRXBUF.

Существует ещё несколько битов, влияющих на возможность генерации прерываний:

- при UCAxRXEIE = 0 символы, принятые с ошибками, не устанавливают флаг UCAxRXIFG;

- при UCDORM = 1 и работе в многопроцессорном режиме символы, не содержащие адреса, не устанавливают флаг UCAxRXIFG. При использовании обычного режима UART ни один принятый символ не вызовет установки флага UCAxRXIFG;
- при UCBRKIE = 1 обрыв связи приведёт к установке бита UCBRK и флага UCAxRXIFG.

Использование прерываний модуля USCI

Модули USCI_Ax и USCI_Bx используют одни и те же векторы прерываний. Флаги прерываний приёма UCAxRXIFG и UCBxRXIFG связаны с одним вектором прерываний, тогда как флаги прерываний передачи UCAxTXIFG и UCBxTXIFG — с другим вектором.

Примеры использования разделяемого вектора прерывания

В следующем примере содержится фрагмент процедуры обработки прерывания, отвечающей за обработку прерываний приёма модуля USCI_A0, работающего в режиме UART или SPI, и модуля USCI_B0, работающего в режиме SPI.

```
USCIA0_RX_USCIB0_RX_ISR
  BIT.B  #UCA0RXIFG, &IFG2   ; Прерывание приёма USCI_A0?
  JNZ    USCIA0_RX_ISR
USCIB0_RX_ISR?
  ; Читаем UCB0RXBUF (сбрасывается UCB0RXIFG)
  ...
  RETI
USCIA0_RX_ISR
  ; Читаем UCA0RXBUF (сбрасывается UCA0RXIFG)
  ...
  RETI
```

В следующем примере содержится фрагмент процедуры обработки прерывания, отвечающей за обработку прерываний передачи модуля USCI_A0, работающего в режиме UART или SPI, и модуля USCI_B0, работающего в режиме SPI.

```
USCIA0_TX_USCIB0_TX_ISR
  BIT.B  #UCA0TXIFG, &IFG2   ; Прерывание передачи USCI_A0?
  JNZ    USCIA0_TX_ISR
USCIB0_TX_ISR
  ; Пишем в UCB0TXBUF (сбрасывается UCB0TXIFG)
  ...
  RETI
USCIA0_TX_ISR
  ; Пишем в UCA0TXBUF (сбрасывается UCA0TXIFG)
  ...
  RETI
```

15.4. Регистры модуля USCI: режим UART

Список регистров модуля USCI, использующихся в режиме UART, приведён в **Табл. 15.6** и **Табл. 15.7**.

Таблица 15.6. **Регистры управления и состояния модуля USCI_A0**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USCI_A0	UCA0CTL0	Чтение/запись	060h	Сбрасывается после PUC
Регистр управления 1 модуля USCI_A0	UCA0CTL1	Чтение/запись	061h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_A0	UCA0BR0	Чтение/запись	062h	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_A0	UCA0BR1	Чтение/запись	063h	Сбрасывается после PUC
Регистр управления модуляцией модуля USCI_A0	UCA0MCTL	Чтение/запись	064h	Сбрасывается после PUC
Регистр состояния модуля USCI_A0	UCA0STAT	Чтение/запись	065h	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_A0	UCA0RXBUF	Чтение	066h	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_A0	UCA0TXBUF	Чтение/запись	067h	Сбрасывается после PUC
Регистр автоопределения скорости передачи модуля USCI_A0	UCA0ABCTL	Чтение/запись	05Dh	Сбрасывается после PUC
Регистр управления передачей IrDA модуля USCI_A0	UCA0IRTCTL	Чтение/запись	05Eh	Сбрасывается после PUC
Регистр управления приёмом IrDA модуля USCI_A0	UCA0IRRCTL	Чтение/запись	05Fh	Сбрасывается после PUC
Регистр разрешения прерываний 2	IE2	Чтение/запись	001h	Сбрасывается после PUC
Регистр флагов прерываний 2	IFG2	Чтение/запись	003h	00Ah после PUC

Таблица 15.7. **Регистры управления и состояния модуля USCI_A1**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USCI_A1	UCA1CTL0	Чтение/запись	0D0h	Сбрасывается после PUC
Регистр управления 1 модуля USCI_A1	UCA1CTL1	Чтение/запись	0D1h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_A1	UCA1BR0	Чтение/запись	0D2h	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_A1	UCA1BR1	Чтение/запись	0D3h	Сбрасывается после PUC
Регистр управления модуляцией модуля USCI_A1	UCA1MCTL	Чтение/запись	0D4h	Сбрасывается после PUC
Регистр состояния модуля USCI_A1	UCA1STAT	Чтение/запись	0D5h	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_A1	UCA1RXBUF	Чтение	0D6h	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_A1	UCA1TXBUF	Чтение/запись	0D7h	Сбрасывается после PUC
Регистр автоопределения скорости передачи модуля USCI_A1	UCA1ABCTL	Чтение/запись	0CDh	Сбрасывается после PUC
Регистр управления передачей IrDA модуля USCI_A1	UCA1IRTCTL	Чтение/запись	0CEh	Сбрасывается после PUC
Регистр управления приёмом IrDA модуля USCI_A1	UCA1IRRCTL	Чтение/запись	0CFh	Сбрасывается после PUC
Регистр разрешения прерываний модулей USCI_A1/B1	UC1IIE	Чтение/запись	006h	Сбрасывается после PUC
Регистр флагов прерываний модулей USCI_A1/B1	UC1UFG	Чтение/запись	007h	00Ah после PUC

Примечание. Изменение битов регистров управления прерываниями

Чтобы исключить изменение управляющих битов, используемых другими модулями, для установки или сброса битов IEx и IFGx рекомендуется вместо команд `MOV.B` и `CLR.B` применять команды `BIS.B` и `BIC.B`.

UCAxCTL0, регистр управления 0 модуля USCI_Ax

7	6	5	4	3	2	1	0
UCPEN	**UCPAR**	**UCMSB**	**UC7BIT**	**UCSPB**	**UCMODEx**		**UCSYNC**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

UCPEN Бит 7 Контроль чётности. Вход в режиме SPI, вход или выход с открытым стоком в режиме I^2C.
- 0 Контроль чётности отключен
- 1 Контроль чётности включен. Бит чётности генерируется (UCAxTXD) и принимается (UCAxRXD). В многопроцессорном режиме address-bit бит адреса участвует в вычислении бита чётности

UCPAR Бит 6 Тип контроля чётности. Бит UCPAR не используется при отключённом контроле чётности.
- 0 Проверка на нечётность
- 1 Проверка на чётность

UCMSB Бит 5 Порядок передачи битов. Этот бит определяет направление сдвига принимаемых и передаваемых данных.
- 0 Младший бит первый
- 1 Старший бит первый

UC7BIT Бит 4 Размер символа. Этот бит определяет длину передаваемых и принимаемых данных.
- 0 8-битные данные
- 1 7-битные данные

UCSPB Биты 3...2 Выбор стоп-битов. Этот бит определяет количество стоп-битов
- 0 Один стоп-бит
- 1 Два стоп-бита

UCMODEx Бит 1 Режим работы модуля USCI. Биты UCMODEx задают один из асинхронных режимов работы при UCSYNC = 0.
- 00 Режим UART
- 01 Многопроцессорный режим idle-line
- 10 Многопроцессорный режим address-bit
- 11 Режим UART с автоматическим определением скорости обмена

UCSYNC Бит 0 Разрешение синхронного режима работы
- 0 Асинхронный режим
- 1 Синхронный режим

UCAxCTL1, регистр управления 1 модуля USCI_Ax

7	6	5	4	3	2	1	0
UCSSELx		**UCRXEIE**	**UCBRKIE**	**UCDORM**	**UCTXADDR**	**UCTXBRK**	**UCSWRST**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–1

UCSSELx Биты 7...6 Выбор источника тактового сигнала модуля USCI. Эти биты определяют источник сигнала BRCLK.
- 00 UCLK
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

UCRXEIE Бит 5 Разрешение прерывания при приёме символа с ошибкой.
- 0 Ошибочный символ игнорируется, флаг UCAxRXIFG не устанавливается
- 1 Принятый ошибочный символ вызывает установку флага UCAxRXIFG

UCBRKIE Бит 4 Разрешение прерывания при обнаружении обрыва связи.
- 0 Флаг UCAxRXIFG не устанавливается при обрыве связи
- 1 Флаг UCAxRXIFG устанавливается при обрыве связи

UCDORM Бит 3 Управление приёмом. Этот бит переводит модуль USCI в «спящий» режим.
- 0 Модуль активен. Любой принятый символ вызывает установку флага UCAxRXIFG
- 1 Модуль не активен. Установка флага UCAxRXIFG производится только при приёме адресного символа. В режиме UART с автоматическим определением скорости обмена установка флага UCAxRXIFG производится только при приёме синхронизирующей последовательности

UCTXADDR Бит 2 Передача адреса. Следующий передаваемый кадр будет помечен как адресный в зависимости от выбранного многопроцессорного режима.
- 0 Следующий передаваемый кадр содержит данные
- 1 Следующий передаваемый кадр содержит адрес

UCTXBRK Бит 1 Формирование состояния обрыва связи. Формирует состояние обрыва связи при следующей записи в регистр передачи. В режиме UART с автоматическим определением скорости обмена в регистр UCAxTXBUF необходимо записать 055h для формирования корректной синхронизирующей последовательности. В противном случае в буфер передачи должно быть записано 00h.
- 0 Следующий передаваемый кадр не формирует состояние разрыва связи
- 1 Следующий передаваемый кадр формирует состояние разрыва связи или синхронизирующую последовательность

UCSWRST Бит 0 Программный сброс модуля USCI.
- 0 Модуль USCI в рабочем состоянии
- 1 Модуль USCI удерживается в состоянии сброса

UCAxBR0, регистр управления 0 скоростью обмена модуля USCI_Ax

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

UCAxBR1, регистр управления 1 скоростью обмена модуля USCI_Ax

7	6	5	4	3	2	1	0
UCBRx							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx Коэффициент деления предделителя контроллера скорости обмена. Содержимое этих регистров образует 16-битное (UCAxBR0 + UCAxBR1 × 256) значение коэффициента деления.

UCAxMCTL, регистр управления модуляцией модуля USCI_Ax

7	6	5	4	3	2	1	0
UCBRFx				**UCBRSx**			**UCOS16**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

UCBRFx Биты 7...4 Выбор первой секции модулятора. Эти биты определяют шаблон модуляции сигнала BITCLK16 при UCOS16 = 1. При UCOS16 = 0 содержимое битов игнорируется. Шаблоны модуляции приведены в **Табл. 15.3**.

UCBRSx Биты 3...1 Выбор второй секции модулятора. Эти биты определяют шаблон модуляции сигнала BITCLK. Шаблоны модуляции приведены в **Табл. 15.2**.

UCOS16 Бит 0 Включение высокочастотного режима.
- 0 Выключен
- 1 Включён

UCAxSTAT, регистр состояния модуля USCI_Ax

7	6	5	4	3	2	1	0
UCLISTEN	**UCFE**	**UCOE**	**UCPE**	**UCBRK**	**UCRXERR**	**UCADDR UCIDLE**	**UCBUSY**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	r–0

UCLISTEN Бит 7 Режим прослушивания. Этот бит включает режим кольцевой проверки.
- 0 Выключен
- 1 Включен. Вывод UCAxTXD внутри модуля подключен к входу приёмника

UCFE Бит 6 Флаг ошибки кадра.
- 0 Нет ошибки
- 1 Принятый символ содержит стоп-бит с нулевым значением

UCOE Бит 5 Флаг ошибки переполнения. Этот бит устанавливается, если принятый символ помещается в регистр UCAxRXBUF до прочтения из него предыдущего. Бит UCOE сбрасывается автоматически при чтении регистра UCAxRXBUF и не должен сбрасываться программно. В противном случае этот бит не будет работать правильно.
- 0 Нет ошибки
- 1 Обнаружена ошибка переполнения

UCPE Бит 4 Ошибка чётности. При UCPEN = 0 бит UCPE читается как 0.
- 0 Нет ошибки
- 1 Символ принят с ошибкой

UCBRK Бит 3 Флаг обрыва связи.
- 0 Нет обрыва связи
- 1 Обнаружен обрыв связи

UCRXERR Бит 2 Флаг ошибки приёма. Этот бит показывает, что символ был принят с ошибкой (ошибками). Бит UCRXERR = 1 при установке одного или более флагов ошибки (UCFE, UCPE, UCOE). Бит UCRXERR сбрасывается автоматически при чтении регистра UCAxRXBUF.
- 0 Ошибок приёма не обнаружено
- 1 Обнаружена ошибка приёма

UCADDR	Бит 1	Признак приёма адреса в многопроцессорном режиме address-bit. 0 Принятый символ содержит данные 1 Принятый символ содержит адрес
UCIDLE		Признак обнаружения неактивного состояния линии в многопроцессорном режиме idle-line. 0 Неактивное состояние линии не обнаружено 1 Обнаружено неактивное состояние линии
UCBUSY	Бит 0	Модуль USCI занят. Этот бит указывает на то, что в данный момент выполняется приём или передача. 0 Модуль USCI неактивен 1 Модуль USCI осуществляет приём или передачу

UCAxRXBUF, регистр буфера приёма модуля USCI_Ax

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx	Биты 7...0	Буфер приёма доступен пользователю и содержит последний принятый символ, скопированный из сдвигового регистра приёма. При чтении регистра UCAxRXBUF сбрасываются флаги ошибок приёма, бит UCADDR/UCIDLE, а также флаг UCAxRXIFG. При работе с 7-битными данными содержимое UCAxRXBUF выравнивается по младшему биту, а старший бит всегда сброшен.

UCAxTXBUF, регистр буфера передачи модуля USCI_Ax

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx	Биты 7...0	Буфер передачи доступен пользователю и содержит значение, которое будет перегружено в сдвиговый регистр передачи и передано на UCAxTXD. При записи в регистр UCAxTXBUF сбрасывается флаг UCAxTXIFG. При работе с 7-битными данными старший бит регистра UCAxTXBUF не используется и всегда сброшен.

UCAxIRTCTL, регистр управления передачей IrDA модуля USCI_Ax

7	6	5	4	3	2	1	0
UCIRTXPLx						**UCIRTXCLK**	**UCIREN**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

UCIRTXPLx	Биты 7...2	Длительность передаваемых импульсов. Длительность импульсов t_{PULSE} = (UCIRTXPLx + 1)/(2·$f_{IRTXCLK}$)
UCIRTXCLK	Бит 1	Выбор тактового сигнала IrDA. 0 BRCLK 1 BITCLK при UCOS16 = 1. В противном случае — BRCLK
UCIREN	Бит 0	Включение кодера/декодера IrDA. 0 Кодер/декодер IrDA выключены 1 Кодер/декодер IrDA включены

UCAxIRRCTL, регистр управления приёмом IrDA модуля USCI_Ax

7	6	5	4	3	2	1	0
UCIRRXFLx						UCIRRXPL	UCIRRXFE
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

UCIRRXFLx Биты 7...2 Длина фильтра. Минимальная длительность воспринимаемых импульсов определяется, как
$t_{MIN} = (UCIRRXFLx + 4)/(2 \cdot f_{IRTXCLK})$

UCIRRXPL Бит 1 Полярность входного сигнала IrDA.
- 0 Приёмопередатчик IrDA формирует положительный импульс при появлении светового импульса
- 1 Приёмопередатчик IrDA формирует отрицательный импульс при появлении светового импульса

UCIRRXFE Бит 0 Включение программируемого фильтра.
- 0 Приёмный фильтр выключен
- 1 Приёмный фильтр включен

UCAxABCTL, регистр автоопределения скорости передачи модуля USCI_Ax

7	6	5	4	3	2	1	0
Reserved		UCDELIMx		UCSTOE	UCBTOE	Reserved	UCABDEN
r–0	r–0	rw–0	rw–0	rw–0	rw–0	r–0	rw–0

Reserved Биты 7...6 Зарезервированы. Читаются как 0.

UCDELIMx Биты 5...4 Длина разделителя полей паузы и синхронизации.
- 00 1 битовый интервал
- 01 2 битовых интервала
- 10 3 битовых интервала
- 11 4 битовых интервала

UCSTOE Бит 3 Ошибка тайм-аута поля синхронизации.
- 0 Нет ошибки
- 1 Длина поля синхронизации больше допустимой

UCBTOE Бит 2 Ошибка тайм-аута поля паузы.
- 0 Нет ошибки
- 1 Длина поля паузы больше 22 битовых интервалов

Reserved Бит 1 Зарезервирован. Читается как 0

UCABDEN Бит 0 Разрешение автоматического определения скорости передачи.
- 0 Автоматическое определение скорости передачи отключено. Длительности полей паузы и синхронизации не измеряются
- 1 Автоматическое определение скорости передачи включено. Измеряются длительности полей паузы и синхронизации. В соответствии с результатами измерения изменяются установки контроллера скорости передачи

IE2, регистр разрешения прерываний 2

7	6	5	4	3	2	1	0
						UCA0TXIE	**UCA0RXIE**
						rw–0	rw–0

	Биты 7...2	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
UCA0TXIE	Бит 1	Разрешение прерывания передачи модуля USCI_A0. 0 Прерывание запрещено 1 Прерывание разрешено
UCA0RXIE	Бит 0	Разрешение прерывания приёма модуля USCI_A0. 0 Прерывание запрещено 1 Прерывание разрешено

IFG2, регистр флагов прерываний 2

7	6	5	4	3	2	1	0
						UCA0TXIFG	**UCA0RXIFG**
						rw–1	rw–0

	Биты 7...2	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
UCA0TXIFG	Бит 1	Флаг прерывания передачи модуля USCI_A0. Бит UCA0TXIFG устанавливается при опустошении регистра UCA0TXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCA0RXIFG	Бит 0	Флаг прерывания приёма модуля USCI_A0. Бит UCA0RXIFG устанавливается при копировании принятого символа в регистр UCA0RXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания

UC1IE, регистр разрешения прерываний модуля USCI_A1

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**			**UCA1TXIE**	**UCA1RXIE**
rw–0	rw–0	rw–0	rw–0			rw–0	rw–0

Unused	Биты 7...4	Не используются.
	Биты 3...2	Эти биты могут использоваться другими модулями USCI. См документацию на конкретный микроконтроллер.
UCA1TXIE	Бит 1	Разрешение прерывания передачи модуля USCI_A1. 0 Прерывание запрещено 1 Прерывание разрешено
UCA1RXIE	Бит 0	Разрешение прерывания приёма модуля USCI_A1. 0 Прерывание запрещено 1 Прерывание разрешено

UC1IFG, регистр флагов прерываний модуля USCI_A1

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**			**UCA1TXIFG**	**UCA1RXIFG**
rw–0	rw–0	rw–0	rw–0			rw–1	rw–0

Unused Биты 7...4 Не используются.

Биты 3...2 Эти биты могут использоваться другими модулями USCI. См документацию на конкретный микроконтроллер.

UCA1TXIFG Бит 1 Флаг прерывания передачи модуля USCI_A1. Бит UCA1TXIFG устанавливается при опустошении регистра UCA1TXBUF.

0 Не было запроса прерывания

1 Есть запрос прерывания

UCA1RXIFG Бит 0 Флаг прерывания приёма модуля USCI_A1. Бит UCA1RXIFG устанавливается при копировании принятого символа в регистр UCA1RXBUF.

0 Не было запроса прерывания

1 Есть запрос прерывания

ГЛАВА 16

УНИВЕРСАЛЬНЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ КОММУНИКАЦИОННЫЙ ИНТЕРФЕЙС: РЕЖИМ SPI

Модуль универсального последовательного коммуникационного интерфейса (USCI) поддерживает несколько режимов обмена по последовательному каналу. В этой главе описывается работа модуля в режиме последовательного периферийного интерфейса SPI.

16.1. Введение

Модули универсального последовательного коммуникационного интерфейса (USCI) поддерживают несколько режимов передачи данных по последовательному каналу, при этом разные модули USCI поддерживают различные режимы. Модули USCI имеют уникальное обозначение. Так, модуль USCI_A отличается от модуля USCI_B и т.д. Если в конкретном устройстве имеется более одного модуля USCI с идентичными функциями, то в обозначение таких модулей добавляется порядковый номер. Например, если в устройстве имеется два модуля USCI_A, то они имеют обозначения USCI_A0 и USCI_A1. Чтобы узнать, какие именно модули USCI реализованы и реализованы ли вообще в конкретном микроконтроллере, обратитесь к справочной документации.

Модули USCI_A поддерживают:

- режим UART;
- формирование импульсов для обмена по протоколу IrDA;
- автоматическое определение скорости обмена для поддержки протокола LIN;
- режим SPI.

Модули USCI_B поддерживают:

- режим I^2C;
- режим SPI.

16.2. Введение в модуль USCI: режим SPI

В синхронном режиме для связи микроконтроллеров семейства MSP430 с внешним устройствам используется три или четыре вывода: UCxSIMO, UCxSOMI, UCxCLK и UCxSTE. Включение режима SPI осуществляется установкой бита UCSYNC, а разновидность интерфейса SPI (3- или 4-проводный) определяется битами UCMODEx.

Режим SPI имеет следующие особенности:

- 7- или 8-битные данные;
- изменяемый порядок передачи и приёма битов;
- поддержка 3- и 4-проводного интерфейса SPI;
- поддержка режимов ведущего и ведомого;
- независимые сдвиговые регистры передачи и приёма;
- раздельные буферные регистры передачи и приёма;
- поддержка непрерывной передачи и приёма;
- возможность выбора полярности и фазы тактового сигнала;
- программируемая частота тактового сигнала в режиме ведущего;
- независимые прерывания передачи и приёма;
- работа в качестве ведомого в режиме LPM4.

Блок-схема модуля USCI, сконфигурированного для работы в режиме UART, приведена на **Рис. 16.1**.

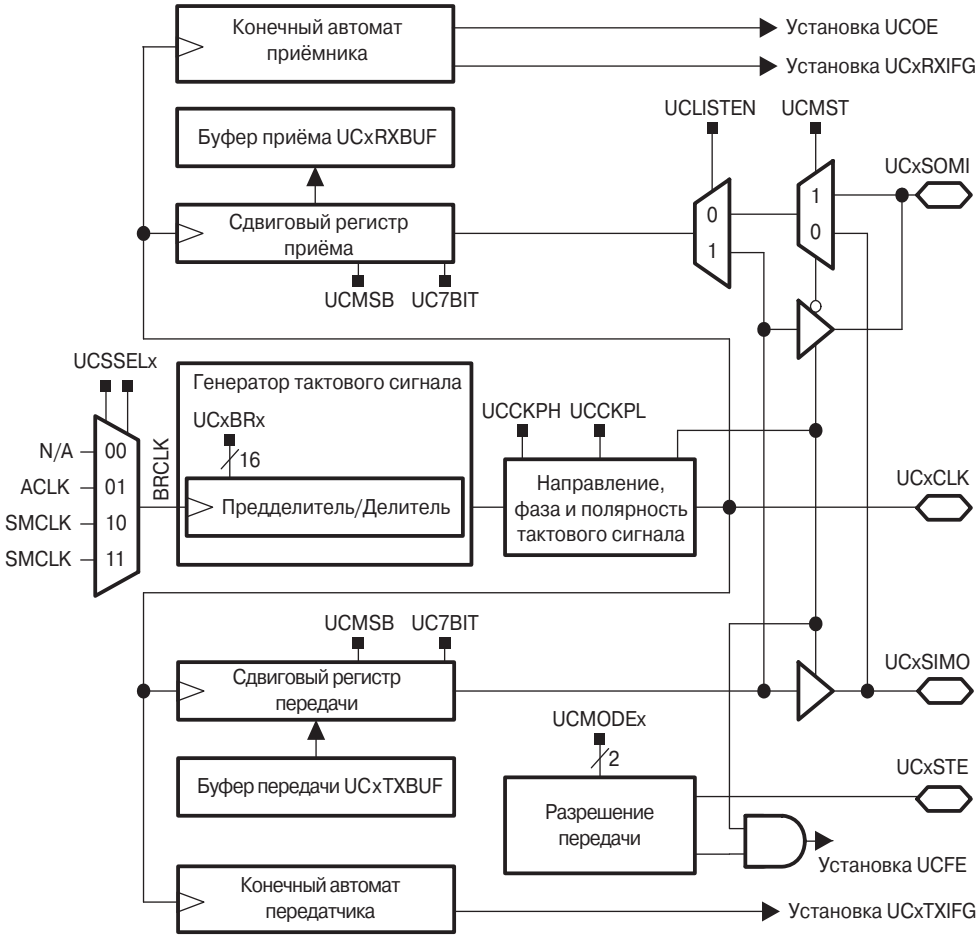


Рис. 16.1. Блок-схема модуля USCI: режим SPI.

16.3. Функционирование модуля USCI: режим SPI

В режиме SPI последовательные данные передаются и принимаются различными устройствами с использованием общего тактового сигнала, генерируемого ведущим устройством. Дополнительный вывод UCxSTE, управляемый ведущим, используется, чтобы разрешать устройству приём и передачу данных.

Для обмена данными по интерфейсу SPI используются три или четыре сигнала:

UCxSIMO Вход ведомого, выход ведущего.
Режим ведущего: UCxSIMO — линия передачи данных
Режим ведомого: UCxSIMO — линия приёма данных

UCxSOMI Выход ведомого, вход ведущего.
Режим ведущего: UCxSOMI — линия приёма данных
Режим ведомого: UCxSOMI — линия передачи данных

UCxCLK Тактовый сигнал SPI.
Режим ведущего: UCxCLK — выход
Режим ведомого: UCxCLK — вход

UCxSTE Разрешение передачи ведомым. Используется в 4-проводном режиме для поддержки нескольких ведущих устройств на одной шине. В 3-проводном режиме не используется. Разные варианты использования UCxSTE приведены в **Табл. 16.1**.

Таблица 16.1. **Использование вывода UCxSTE**

UCMODEx	Активный уровень UCxSTE	UCxSTE	Ведомый	Ведущий
01	ВЫСОКИЙ	0	Не активен	Активен
		1	Активен	Не активен
10	НИЗКИЙ	0	Активен	Не активен
		1	Не активен	Активен

16.3.1. Инициализация и сброс модуля USCI

Модуль USCI сбрасывается по сигналу PUC или в результате установки бита USCSWRST. При появлении сигнала PUC бит USCSWRST автоматически устанавливается, переводя модуль в состояние сброса. При установке бита USCSWRST сбрасываются биты UCxRXIE, UCxTXIE, UCxRXIFG, UCOE, UCFE и устанавливается бит UCxTXIFG. Сброс бита USCSWRST переводит модуль USCI в рабочий режим.

Примечание. Инициализация и реконфигурирование модуля USCI
Инициализацию и реконфигурирование модуля USCI рекомендуется выполнять в следующей последовательности:

1. Установить бит UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Инициализировать все регистры USCI установкой UCSWRST = 1 (включая регистр UCxCTL1).
3. Сконфигурировать порты ввода/вывода.
4. Сбросить в программе бит UCSWRST (`BIC.B #UCSWRST,&UCxCTL1`).
5. Разрешить прерывания (при необходимости) установкой битов UCxRXIE и/или UCxTXIE.

16.3.2. Формат символа

В режиме SPI модуль USCI поддерживает 7- и 8-битные символы, что определяется битом UC7BIT.

В 7-битном режиме содержимое регистра UCxRXBUF выравнивается по младшему биту, а старший бит регистра всегда сброшен. Порядок передачи битов символа определяется битом UCMSB.

Примечание. Формат символа по умолчанию
По умолчанию, данные в режиме SPI передаются начиная с младшего значащего бита. Однако существуют реализации интерфейса SPI, требующие передачи данных начиная со старшего бита.

16.3.3. Режим ведущего

На **Рис. 16.2** показано использование модуля USCI в качестве ведущего в обеих (3- и 4-проводной) конфигурациях. Передача данных инициируется модулем USCI при загрузке значения в регистр буфера передачи UCxTXBUF. Если сдвиговый регистр передачи пуст, то содержимое буфера пересылается в него, запуская побитовую выдачу символа на вывод UCxSIMO, начиная со старшего или младшего бита, в зависимости от значения бита UCMSB. Данные, поступающие на вывод UCxSOMI, вдвигаются в сдвиговый регистр приёма по противоположному фронту тактового сигнала. После приёма заданного числа битов полученное значение пересылается из сдвигового регистра в буфер приёма UCxRXBUF. Одновременно устанавливается флаг прерывания приёма UCxRXIFG, сигнализирующий об окончании процесса приёма/передачи.

Установка флага прерывания передачи UCxTXIFG извещает о том, что данные были перемещены из буфера UCxTXBUF в сдвиговый регистр передачи и что буфер готов к приёму новых данных. Этот флаг никак не связан с завершением операции приёма/передачи.

Для приёма данных в режиме ведущего, необходимо записать что-либо в регистр UCxTXBUF, поскольку операции приёма и передачи выполняются одновременно.

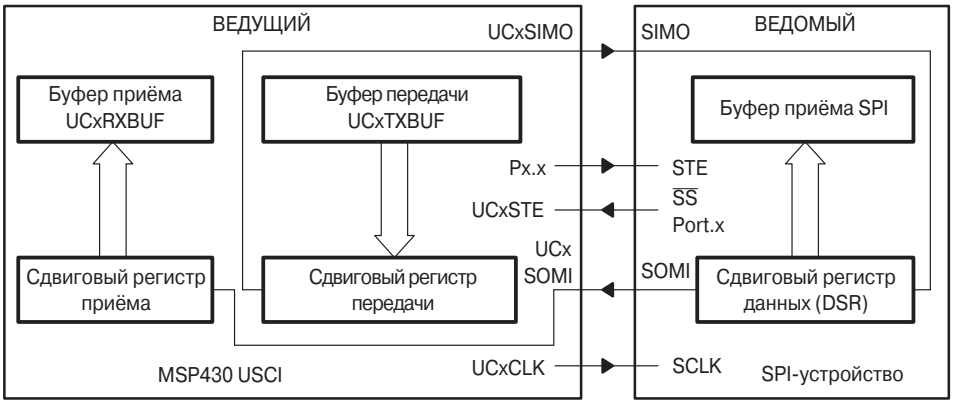


Рис. 16.2. Модуль USCI в режиме ведущего и внешнее ведомое устройство.

Работа ведущего в 4-проводном режиме

В 4-проводном режиме вывод UCxSTE используется для предотвращения конфликтов с другими ведущими устройствами, подключёнными к шине, и управляет работой ведущего в соответствии с **Табл. 16.1**. Когда сигнал на выводе UCxSTE переводит ведущего в неактивное состояние:

- выводы UCxSIMO и UCxSCLK переключаются на вход и не влияют на состояние шины;
- устанавливается флаг ошибки UCFE, сигнализирующий о нарушении целостности процесса обмена, чтобы пользователь мог обработать эту ситуацию;
- внутренний конечный автомат модуля сбрасывается и процесс сдвига прерывается.

В случае загрузки данных в регистр UCxTXBUF в то время, пока ведущий удерживается в неактивном состоянии сигналом UCxSTE, эти данные будут переданы сразу же, как только сигнал UCxSTE переведёт ведущего в активное состояние. Если текущая пересылка была прервана сигналом UCxSTE, то для повторной передачи этих данных после возврата модуля в активное состояние их необходимо снова загрузить в регистр UCxTXBUF. При работе ведущего в 3-проводном режиме сигнал UCxSTE не используется.

16.3.4. Режим ведомого

На **Рис. 16.3** показано использование модуля USCI в качестве ведомого в обеих (3- и 4-проводной) конфигурациях. Вывод UCxCLK является входом для тактового сигнала SPI, который должен генерироваться ведущим устройством. Скорость пересылки данных определяется параметрами этого сигнала и не зависит от настроек внутреннего генератора тактового сигнала. При появлении сигнала UCxCLK данные, загруженные в регистр UCxTXBUF и перемещённые в сдвиговый регистр передачи, побитно выдаются на вывод UCxSOMI. Данные, поступающие на вывод UCxSMO, вдвигаются в сдвиговый регистр приёма по противоположному фронту тактового сигнала. После приёма заданного числа битов полученное значение пересылается из сдвигового регистра в буфер приёма UCxRXBUF. Одновременно устанавливается флаг прерывания UCxRXIFG, сигнализирующий о появлении новых данных. Если ранее принятые данные не были считаны из регистра UCxRXBUF перед загрузкой в него новых данных из сдвигового регистра, то устанавливается флаг переполнения UCOE.

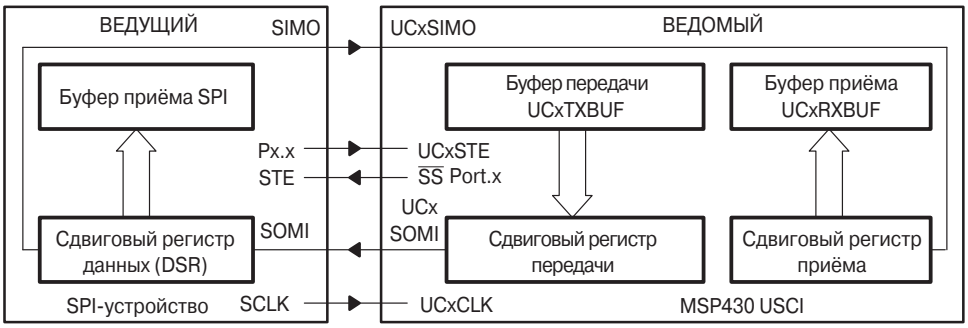


Рис. 16.3. Модуль USCI в режиме ведомого и внешнее ведущее устройство.

Работа ведомого в 4-проводном режиме

В 4-проводном режиме вывод UCxSTE используется, чтобы разрешать или запрещать ведомому выполнение операций приёма/передачи данных. Управляется этот вывод ведущим устройством. Когда уровень сигнала UCxSTE соответствует активному состоянию ведомого, последний работает в обычном режиме. Когда же сигнал UCxSTE переводит ведомого в неактивное состояние:

- приём данных, поступающих на вход UCxSIMO, прекращается;
- вывод UCxSOMI переключается на вход;
- операции сдвига прекращаются до тех пор, пока на линии UCxSTE не появится сигнал, соответствующий активному состоянию ведомого устройства.

При работе ведомого в 3-проводном режиме сигнал UCxSTE не используется.

16.3.5. Разрешение обмена по интерфейсу SPI

После включения модуля USCI сбросом бита UCSWRST приёмник и передатчик модуля готовы к работе. В режиме ведущего генератор тактового сигнала находится в состоянии готовности, но сигнал не генерирует, поскольку задающий тактовый сигнал на него не поступает. В режиме ведомого генератор тактового сигнала отключён, а сигнал синхронизации поступает от ведущего устройства.

Признаком выполнения операции приёма или передачи является установленный бит UCBUSY.

При появлении сигнала системного сброса PUC или при установке бита UCSWRST модуль USCI сразу же выключается, и текущая пересылка данных прерывается.

Разрешение передачи

В режиме ведущего запись в регистр UCxTXBUF запускает генератор тактового сигнала, в результате чего начинается передача данных.

В режиме ведомого передача начинается при появлении тактового сигнала от ведущего и, в 4-проводной конфигурации, при наличии на выводе UCxSTE сигнала, соответствующего активному состоянию ведомого.

Разрешение приёма

Приём данных по интерфейсу SPI осуществляется при включении передатчика. Операции приёма и передачи выполняются одновременно.

16.3.6. Управление тактовым сигналом

Тактовый сигнал на шине SPI формируется ведущим устройством. При UCMST = 1 сигнал от генератора тактового сигнала модуля USCI выводится на вывод UCxCLK. Исходный сигнал, из которого формируется тактовый сигнал SPI, задаётся битами UCSSELx. При UCMST = 0 тактовый сигнал для модуля USCI поступает на вывод UCxCLK от ведущего устройства. Генератор тактового сигнала модуля при этом не используется и состояние битов UCSSELx безразлич-

но. Приёмник и передатчик SPI работают параллельно и тактируются одним и тем же сигналом.

В регистрах управления тактовым генератором UCxxBR1 и UCxxBR0 содержится 16-битное значение UCBRx, представляющее собой коэффициент деления тактового сигнала модуля USCI (BRCLK). Таким образом, максимальная частота тактового сигнала, формируемого в режиме ведущего, равна частоте сигнала BCLK. Модуляция в режиме SPI не применяется, и регистр UCAxMCTL при использовании модуля USCI_A должен быть сброшен. Частота тактового сигнала UCAxCLK/UCBxCLK определяется выражением:

$$f_{\text{BitClock}} = \frac{f_{\text{BRCLK}}}{\text{UCBRx}}.$$

Полярность и фаза тактового сигнала

Полярность и фаза тактового сигнала UCxCLK задаются независимо друг от друга битами UCCKPL и UCCKPH регистра управления модуля USCI. Временные диаграммы для различных значений этих битов приведены на **Рис. 16.4**.

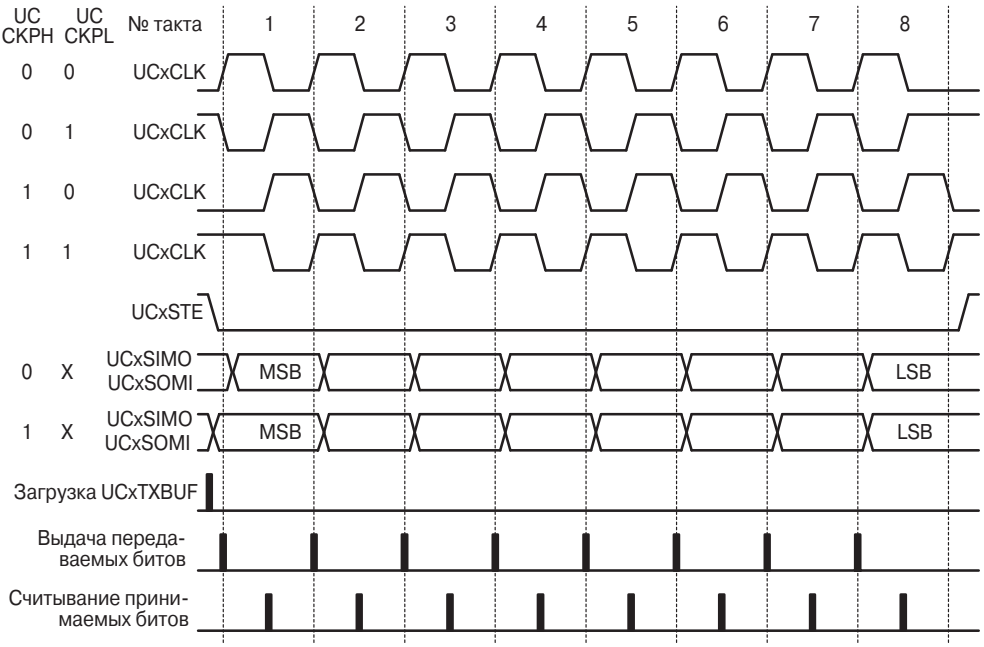


Рис. 16.4. Временные диаграммы модуля USCI в режиме SPI при UCMSB = 1.

16.3.7. Использование режима SPI совместно с режимами пониженного энергопотребления

Модуль USCI обеспечивает автоматическую активацию тактового сигнала SMCLK, что делает возможным использование модуля в режимах пониженного

энергопотребления. Если модуль USCI тактируется сигналом SMCLK, который неактивен по причине нахождения устройства в режиме пониженного энергопотребления, то модуль автоматически активирует этот тактовый сигнал при необходимости, независимо от значений управляющих битов выбора источника системного тактового сигнала. Данный тактовый сигнал будет активным до тех пор, пока модуль USCI не вернётся в состояние ожидания. После этого контроль над источником тактового сигнала вернётся к модулю синхронизации. Автоматическая активация тактового сигнала ACLK не предусмотрена.

При активации модулем USCI неактивного источника тактового сигнала, этот источник становится доступным для всего устройства, так что его активация может затронуть любой периферийный модуль, сконфигурированный для использования данного тактового сигнала. Например, значение таймера, использующего SMCLK, будет инкрементироваться в течение всего времени, пока модуль USCI будет находиться в активном режиме.

В режиме ведомого SPI наличие внутреннего тактового сигнала не требуется, поскольку сигнал синхронизации формируется внешним ведущим устройством. Это даёт возможность использовать модуль USCI в качестве ведомого SPI-устройства даже при использовании режима пониженного энергопотребления LPM4, в котором выключены все источники тактовых сигналов.

16.3.8. Прерывания в режиме SPI

Модуль USCI имеет по одному вектору прерывания для передачи и приёма.

Прерывание SPI при передаче

Флаг прерывания UCxTXIFG, устанавливаемый передатчиком, показывает готовность регистра UCxTXBUF к загрузке нового символа. Если установлены биты GIE и UCxTXIE, то при установке флага UCxTXIFG генерируется запрос прерывания. Флаг UCxTXIFG автоматически сбрасывается при записи в регистр UCxTXBUF. Флаг UCxTXIFG устанавливается после сигнала сброса PUC или при UCSWRST = 1. Бит разрешения прерывания UCxTXIE сбрасывается после сигнала сброса PUC или при UCSWRST = 1.

Примечание. Запись в регистр UCxTXBUF в режиме SPI
Запись в регистр UCxTXBUF при UCxTXIFG = 0 может вызвать передачу некорректных данных.

Прерывание SPI при приёме

Флаг прерывания UCxRXIFG устанавливается каждый раз при приёме символа и загрузке его в регистр UCxRXBUF. При установленных битах GIE и UCxRXIE генерируется запрос прерывания. Биты UCxRXIFG и UCxRXIE сбрасываются сигналом системного сброса PUC или при UCSWRST = 1. Флаг UCxRXIFG автоматически сбрасывается при чтении регистра UCxRXBUF.

Использование прерываний модуля USCI

Модули USCI_Ax и USCI_Bx используют одни и те же векторы прерываний. Флаги прерываний приёма UCAxRXIFG и UCBxRXIFG связаны с одним вектором прерываний, тогда как флаги прерываний передачи UCAxTXIFG и UCBxTXIFG связаны с другим вектором.

Примеры использования разделяемого вектора прерывания

В следующем примере содержится фрагмент процедуры обработки прерывания, отвечающей за обработку прерываний приёма модуля USCI_A0, работающего в режиме UART или SPI, и модуля USCI_B0, работающего в режиме SPI.

```
USCIA0_RX_USCIB0_RX_ISR
  BIT.B  #UCA0RXIFG, &IFG2   ; Прерывание приёма USCI_A0?
  JNZ    USCIA0_RX_ISR
USCIB0_RX_ISR?
  ; Читаем UCB0RXBUF (сбрасывается UCB0RXIFG)
  ...
  RETI
USCIA0_RX_ISR
  ; Читаем UCA0RXBUF (сбрасывается UCA0RXIFG)
  ...
  RETI
```

В следующем примере содержится фрагмент процедуры обработки прерывания, отвечающей за обработку прерываний передачи модуля USCI_A0, работающего в режиме UART или SPI, и модуля USCI_B0, работающего в режиме SPI.

```
USCIA0_TX_USCIB0_TX_ISR
  BIT.B  #UCA0TXIFG, &IFG2   ; Прерывание передачи USCI_A0?
  JNZ    USCIA0_TX_ISR
USCIB0_TX_ISR
  ; Пишем в UCB0TXBUF (сбрасывается UCB0TXIFG)
  ...
  RETI
USCIA0_TX_ISR
  ; Пишем в UCA0TXBUF (сбрасывается UCA0TXIFG)
  ...
  RETI
```

16.4. Регистры модуля USCI: режим SPI

Список регистров модулей USCI_A0 и USCI_B0, использующихся в режиме SPI, приведён в **Табл. 16.2**. Регистры модулей USCI_A1 и USCI_B1, использующиеся в режиме SPI, перечислены в **Табл. 16.3**.

Таблица 16.2. **Регистры управления и состояния модулей USCI_A0 и USCI_B0**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USCI_A0	UCA0CTL0	Чтение/запись	060h	Сбрасывается после PUC
Регистр управления 1 модуля USCI_A0	UCA0CTL1	Чтение/запись	061h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_A0	UCA0BR0	Чтение/запись	062h	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_A0	UCA0BR1	Чтение/запись	063h	Сбрасывается после PUC
Регистр управления модуляцией модуля USCI_A0	UCA0MCTL	Чтение/запись	064h	Сбрасывается после PUC
Регистр состояния модуля USCI_A0	UCA0STAT	Чтение/запись	065h	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_A0	UCA0RXBUF	Чтение	066h	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_A0	UCA0TXBUF	Чтение/запись	067h	Сбрасывается после PUC
Регистр управления 0 модуля USCI_B0	UCB0CTL0	Чтение/запись	068h	001h после PUC
Регистр управления 1 модуля USCI_B0	UCB0CTL1	Чтение/запись	069h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_B0	UCB0BR0	Чтение/запись	06Ah	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_B0	UCB0BR1	Чтение/запись	06Bh	Сбрасывается после PUC
Регистр управления модуляцией модуля USCI_B0	UCB0MCTL	Чтение/запись	06Ch	Сбрасывается после PUC
Регистр состояния модуля USCI_B0	UCB0STAT	Чтение/запись	06Dh	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_B0	UCB0RXBUF	Чтение	06Eh	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_B0	UCB0TXBUF	Чтение/запись	06Fh	Сбрасывается после PUC
Регистр разрешения прерываний 2	IE2	Чтение/запись	001h	Сбрасывается после PUC
Регистр флагов прерываний 2	IFG2	Чтение/запись	003h	00Ah после PUC

Примечание. Изменение битов регистров управления прерываниями

Чтобы исключить изменение управляющих битов, используемых другими модулями, для установки или сброса битов IEx и IFGx рекомендуется вместо команд `MOV.B` и `CLR.B` применять команды `BIS.B` и `BIC.B`.

Таблица 16.3. **Регистры управления и состояния модулей USCI_A1и USCI_B1**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USCI_A1	UCA1CTL0	Чтение/запись	0D0h	Сбрасывается после PUC
Регистр управления 1 модуля USCI_A1	UCA1CTL1	Чтение/запись	0D1h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_A1	UCA1BR0	Чтение/запись	0D2h	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_A1	UCA1BR1	Чтение/запись	0D3h	Сбрасывается после PUC
Регистр управления модуляцией модуля USCI_A1	UCA1MCTL	Чтение/запись	0D4h	Сбрасывается после PUC
Регистр состояния модуля USCI_A1	UCA1STAT	Чтение/запись	0D5h	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_A1	UCA1RXBUF	Чтение	0D6h	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_A1	UCA1TXBUF	Чтение/запись	0D7h	Сбрасывается после PUC
Регистр управления 0 модуля USCI_B1	UCB1CTL0	Чтение/запись	0D8h	001h после PUC
Регистр управления 1 модуля USCI_B1	UCB1CTL1	Чтение/запись	0D9h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_B1	UCB1BR0	Чтение/запись	0DAh	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_B1	UCB1BR1	Чтение/запись	0DBh	Сбрасывается после PUC
Регистр управления модуляцией модуля USCI_B1	UCB1MCTL	Чтение/запись	0DCh	Сбрасывается после PUC
Регистр состояния модуля USCI_B1	UCB1STAT	Чтение/запись	0DDh	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_B1	UCB1RXBUF	Чтение	0DEh	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_B1	UCB1TXBUF	Чтение/запись	0DFh	Сбрасывается после PUC
Регистр разрешения прерываний модулей USCI_A1/B1	UC1IE	Чтение/запись	006h	Сбрасывается после PUC
Регистр флагов прерываний модулей USCI_A1/B1	UC1IFG	Чтение/запись	007h	00Ah после PUC

UCAxCTL0, регистр управления 0 модуля USCI_Ax
UCBxCTL0, регистр управления 0 модуля USCI_Bx

7	6	5	4	3	2	1	0
UCCKPH	**UCCKPL**	**UCMSB**	**UC7BIT**	**UCMST**	**UCMODEx**		**UCSYNC=1**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	

UCCKPH Бит 7 Фаза тактового сигнала.
- 0 Данные выставляются по первому фронту сигнала UCLK, а считываются по следующему фронту
- 1 Данные считываются по первому фронту сигнала UCLK, а выставляются по следующему фронту

UCCKPL Бит 6 Полярность тактового сигнала.
- 0 Неактивное состояние — НИЗКИЙ уровень
- 1 Неактивное состояние — ВЫСОКИЙ уровень

UCMSB Бит 5 Порядок передачи битов. Этот бит определяет направление сдвига принимаемых и передаваемых данных.
- 0 Младший бит первый
- 1 Старший бит первый

UC7BIT Бит 4 Размер символа. Этот бит определяет длину передаваемых и принимаемых данных.
- 0 8-битные данные
- 1 7-битные данные

UCMST Биты 3...2 Выбор режима ведущего.
- 0 Режим ведомого
- 1 Режим ведущего

UCMODEx Бит 1 Режим работы модуля USCI. Биты UCMODEx задают синхронный режим работы при UCSYNC = 1.
- 00 3-проводный SPI
- 01 4-проводный SPI, активный уровень UCxSTE — ВЫСОКИЙ: работа ведомого разрешается при UCxSTE = 1
- 10 4-проводный SPI, активный уровень UCxSTE — НИЗКИЙ: работа ведомого разрешается при UCxSTE = 0
- 11 Режим I^2C

UCSYNC Бит 0 Разрешение синхронного режима работы.
- 0 Асинхронный режим
- 1 Синхронный режим

UCAxCTL1, регистр управления 1 модуля USCI_Ax
UCBxCTL1, регистр управления 1 модуля USCI_Bx

7	6	5	4	3	2	1	0
UCSSELx		**Unused**					**UCSWRST**
rw–0	rw–0	rw–0*, r0**	rw–0	rw–0	rw–0	rw–0	rw–1

* UCAxCTL1 (USCI_Ax)
** UCBxCTL1 (USCI_Bx)

UCSSELx Биты 7...6 Выбор источника тактового сигнала модуля USCI. Эти биты определяют источник сигнала BRCLK в режиме ведущего. В режиме ведомого всегда используется сигнал UCxCLK.
- 00 —
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

Unused Биты 5...1 Не используются

UCSWRST Бит 0 Программный сброс модуля USCI.
- 0 Модуль USCI в рабочем состоянии
- 1 Модуль USCI удерживается в состоянии сброса

UCAxBR0, регистр управления 0 скоростью обмена модуля USCI_Ax
UCBxBR0, регистр управления 0 скоростью обмена модуля USCI_Bx

7	6	5	4	3	2	1	0
UCBRx – младший байт							
rw	rw	rw	rw	rw	rw	rw	rw

UCAxBR1, регистр управления 1 скоростью обмена модуля USCI_Ax
UCBxBR1, регистр управления 1 скоростью обмена модуля USCI_Bx

7	6	5	4	3	2	1	0
UCBRx – старший байт							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx Коэффициент деления предделителя контроллера скорости обмена. Содержимое этих регистров образует 16-битное (UCxxBR0 + UCxxBR1 × 256) значение коэффициента деления.

UCAxSTAT, регистр состояния модуля USCI_Ax
UCBxSTAT, регистр состояния модуля USCI_Bx

7	6	5	4	3	2	1	0
UCLISTEN	**UCFE**	**UCOE**	**Unused**	**Unused**	**Unused**	**Unused**	**UCBUSY**
rw–0	rw–0	rw–0	rw–0*, r0**	rw–0	rw–0	rw–0	r–0

* UCAxSTAT (USCI_Ax)
** UCBxSTAT (USCI_Bx)

UCLISTEN Бит 7 Режим прослушивания. Этот бит включает режим кольцевой проверки.
- 0 Выключен
- 1 Включён. Выход передатчика внутри модуля подключен к входу приёмника

UCFE Бит 6 Флаг ошибки кадра. Этот бит служит признаком конфликта на шине в режиме ведущего при использовании 4-проводного интерфейса. В режиме ведущего 3-проводного интерфейса, а также в режиме ведомого бит UCFE не используется.
- 0 Нет ошибки
- 1 Зафиксирован конфликт на шине

UCOE Бит 5 Флаг ошибки переполнения. Этот бит устанавливается, если принятый символ помещается в регистр UCxRXBUF до прочтения из него предыдущего. Бит UCOE сбрасывается автоматически при чтении регистра UCxRXBUF и не должен сбрасываться программно. В противном случае этот бит не будет работать правильно.
- 0 Нет ошибки
- 1 Обнаружена ошибка переполнения

Unused Биты 4...1 Не используются.

UCBUSY Бит 0 Модуль USCI занят. Этот бит указывает на то, что в данный момент выполняется приём или передача.
- 0 Модуль USCI неактивен
- 1 Модуль USCI осуществляет приём или передачу

UCAxRXBUF, регистр буфера приёма модуля USCI_Ax
UCBxRXBUF, регистр буфера приёма модуля USCI_Bx

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx	Биты 7...0	Буфер приёма доступен пользователю и содержит последний принятый символ, скопированный из сдвигового регистра приёма. При чтении регистра UCxRXBUF сбрасываются флаги ошибок приёма и флаг UCxRXIFG. При работе с 7-битными данными содержимое UCxRXBUF выравнивается по младшему биту, а старший бит всегда сброшен.

UCAxTXBUF, регистр буфера передачи модуля USCI_Ax
UCBxTXBUF, регистр буфера передачи модуля USCI_Bx

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx	Биты 7...0	Буфер передачи доступен пользователю и содержит значение, которое будет перегружено в сдвиговый регистр передачи и передано по шине. При записи в регистр UCxTXBUF сбрасывается флаг UCxTXIFG. При работе с 7-битными данными старший бит регистра UCxTXBUF не используется и всегда сброшен.

IE2, регистр разрешения прерываний 2

7	6	5	4	3	2	1	0
				UCB0TXIE	UCB0RXIE	UCA0TXIE	UCA0RXIE
				rw–0	rw–0	rw–0	rw–0

	Биты 7...4	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
UCB0TXIE	Бит 3	Разрешение прерывания передачи модуля USCI_B0. 0 Прерывание запрещено 1 Прерывание разрешено
UCB0RXIE	Бит 2	Разрешение прерывания приёма модуля USCI_B0. 0 Прерывание запрещено 1 Прерывание разрешено
UCA0TXIE	Бит 1	Разрешение прерывания передачи модуля USCI_A0. 0 Прерывание запрещено 1 Прерывание разрешено
UCA0RXIE	Бит 0	Разрешение прерывания приёма модуля USCI_A0. 0 Прерывание запрещено 1 Прерывание разрешено

IFG2, регистр флагов прерываний 2

7	6	5	4	3	2	1	0
				UCB0TXIFG	**UCB0RXIFG**	**UCA0TXIFG**	**UCA0RXIFG**
				rw–1	rw–0	rw–1	rw–0

	Биты 7...4	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
UCB0TXIFG	Бит 3	Флаг прерывания передачи модуля USCI_B0. Бит UCB0TXIFG устанавливается при опустошении регистра UCB0TXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCB0RXIFG	Бит 2	Флаг прерывания приёма модуля USCI_B0. Бит UCB0RXIFG устанавливается при копировании принятого символа в регистр UCB0RXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCA0TXIFG	Бит 1	Флаг прерывания передачи модуля USCI_A0. Бит UCA0TXIFG устанавливается при опустошении регистра UCA0TXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCA0RXIFG	Бит 0	Флаг прерывания приёма модуля USCI_A0. Бит UCA0RXIFG устанавливается при копировании принятого символа в регистр UCA0RXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания

UC1IE, регистр разрешения прерываний модулей USCI_A1/USCI_B1

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**UCB1TXIE**	**UCB1RXIE**	**UCA1TXIE**	**UCA1RXIE**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Unused	Биты 7...4	Не используются.
UCB1TXIE	Бит 3	Разрешение прерывания передачи модуля USCI_B1. 0 Прерывание запрещено 1 Прерывание разрешено
UCB1RXIE	Бит 2	Разрешение прерывания приёма модуля USCI_B1. 0 Прерывание запрещено 1 Прерывание разрешено
UCA1TXIE	Бит 1	Разрешение прерывания передачи модуля USCI_A1. 0 Прерывание запрещено 1 Прерывание разрешено
UCA1RXIE	Бит 0	Разрешение прерывания приёма модуля USCI_A1. 0 Прерывание запрещено 1 Прерывание разрешено

UC1IFG, регистр флагов прерываний модулей USCI_A1/USCI_B1

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**UCB1TXIFG**	**UCB1RXIFG**	**UCA1TXIFG**	**UCA1RXIFG**
rw–0	rw–0	rw–0	rw–0	rw–1	rw–0	rw–1	rw–0

Unused Биты 7...4 Не используются.

UCB1TXIFG Бит 3 Флаг прерывания передачи модуля USCI_B1. Бит UCB1TXIFG устанавливается при опустошении регистра UCB1TXBUF.
0 Не было запроса прерывания
1 Есть запрос прерывания

UCB1RXIFG Бит 2 Флаг прерывания приёма модуля USCI_B1. Бит UCB1RXIFG устанавливается при копировании принятого символа в регистр UCB1RXBUF.
0 Не было запроса прерывания
1 Есть запрос прерывания

UCA1TXIFG Бит 1 Флаг прерывания передачи модуля USCI_A1. Бит UCA1TXIFG устанавливается при опустошении регистра UCA1TXBUF.
0 Не было запроса прерывания
1 Есть запрос прерывания

UCA1RXIFG Бит 0 Флаг прерывания приёма модуля USCI_A1. Бит UCA1RXIFG устанавливается при копировании принятого символа в регистр UCA1RXBUF.
0 Не было запроса прерывания
1 Есть запрос прерывания

ГЛАВА 17

УНИВЕРСАЛЬНЫЙ ПОСЛЕДОВАТЕЛЬНЫЙ КОММУНИКАЦИОННЫЙ ИНТЕРФЕЙС: РЕЖИМ I^2C

Модуль универсального последовательного коммуникационного интерфейса (USCI) поддерживает несколько режимов обмена по последовательному каналу. В этой главе описывается работа модуля в режиме I^2C.

17.1. Введение

Модули универсального последовательного коммуникационного интерфейса (USCI) поддерживают несколько режимов передачи данных по последовательному каналу, при этом разные модули USCI поддерживают различные режимы. Модули USCI имеют уникальное обозначение. Так, модуль USCI_A отличается от модуля USCI_B и т.д. Если в конкретном устройстве имеется более одного модуля USCI с идентичными функциями, то в обозначение таких модулей добавляется порядковый номер. Например, если в устройстве имеется два модуля USCI_A, то они имеют обозначения USCI_A0 и USCI_A1. Чтобы узнать, какие именно модули USCI реализованы и реализованы ли вообще в конкретном микроконтроллере, обратитесь к справочной документации.

Модули USCI_A поддерживают:

- режим UART;
- формирование импульсов для обмена по протоколу IrDA;
- автоматическое определение скорости обмена для поддержки протокола LIN;
- режим SPI.

Модули USCI_B поддерживают:

- режим I^2C;
- режим SPI.

17.2. Введение в модуль USCI: режим I^2C

В режиме I^2C модуль USCI обеспечивает взаимодействие микроконтроллеров семейства MSP430 и различных I^2C-совместимых устройств, подключённых к 2-проводной последовательной шине I^2C. Внешние устройства, подключённые к шине, передают и/или принимают в последовательном виде данные к/от модуля USCI с помощью 2-проводного интерфейса I^2C.

Режим I²C имеет следующие особенности:

- Соответствует спецификации I²C версии 2.1 компании Philips Semiconductor:
  - 7- и 10-битный режимы адресации;
  - поддержка общих вызовов;
  - поддержка состояний СТАРТ/ПОВСТАРТ/СТОП;
  - режимы ведущий-передатчик/ведущий-приёмник с поддержкой нескольких ведущих на шине;
  - режимы ведомый-приёмник/ведомый-передатчик;
  - поддержка стандартного (до 100 Кбит/с) и скоростного (до 400 Кбит/с) режимов.
- Программируемая частота UCxCLK в режиме ведущего.
- Низкое потребление.
- Обнаружение ведомым-приёмником состояния СТАРТ для автоматического выхода из режимов LPMx.
- Работа в качестве ведомого в режиме LPM4.

Блок-схема модуля USCI, сконфигурированного для работы в режиме I²C, приведена на **Рис. 17.1**.

17.3. Функционирование модуля USCI: режим I²C

В режиме I²C поддерживается взаимодействие с любыми ведомыми или ведущими I²C-совместимыми устройствами. Пример использования шины I²C показан на **Рис. 17.2**. Каждое устройство на шине имеет уникальный адрес и может выступать в качестве либо передатчика, либо приёмника. Кроме того, все устройства, подключённые к шине I²C, можно разделить на ведущие и ведомые устройства. Ведущее устройство инициирует процесс пересылки данных и формирует тактовый сигнал SCL. Любое устройство, адресованное ведущим, рассматривается как ведомое.

Обмен по шине I²C осуществляется с использованием двух линий: линии данных (SDA) и линии синхронизации (SCL). Обе линии являются двунаправленными и должны быть подключены к источнику постоянного напряжения через подтягивающие резисторы.

Примечание. Уровни сигналов SDA и SCL
Уровни сигналов на выводах SDA и SCL микроконтроллеров MSP430 не должны превышать напряжение питания V_{CC} микроконтроллера.

17.3.1. Инициализация и сброс модуля USCI

Модуль USCI сбрасывается по сигналу PUC или в результате установки бита USCSWRST. При появлении сигнала PUC бит USCSWRST автоматически устанавливается, переводя модуль в состояние сброса. Для переключения модуля в режим I²C необходимо установить оба бита UCMODEx = 11. После инициализации модуля он готов к передаче или приёму данных. Сброс бита USCSWRST переводит модуль USCI в рабочий режим.

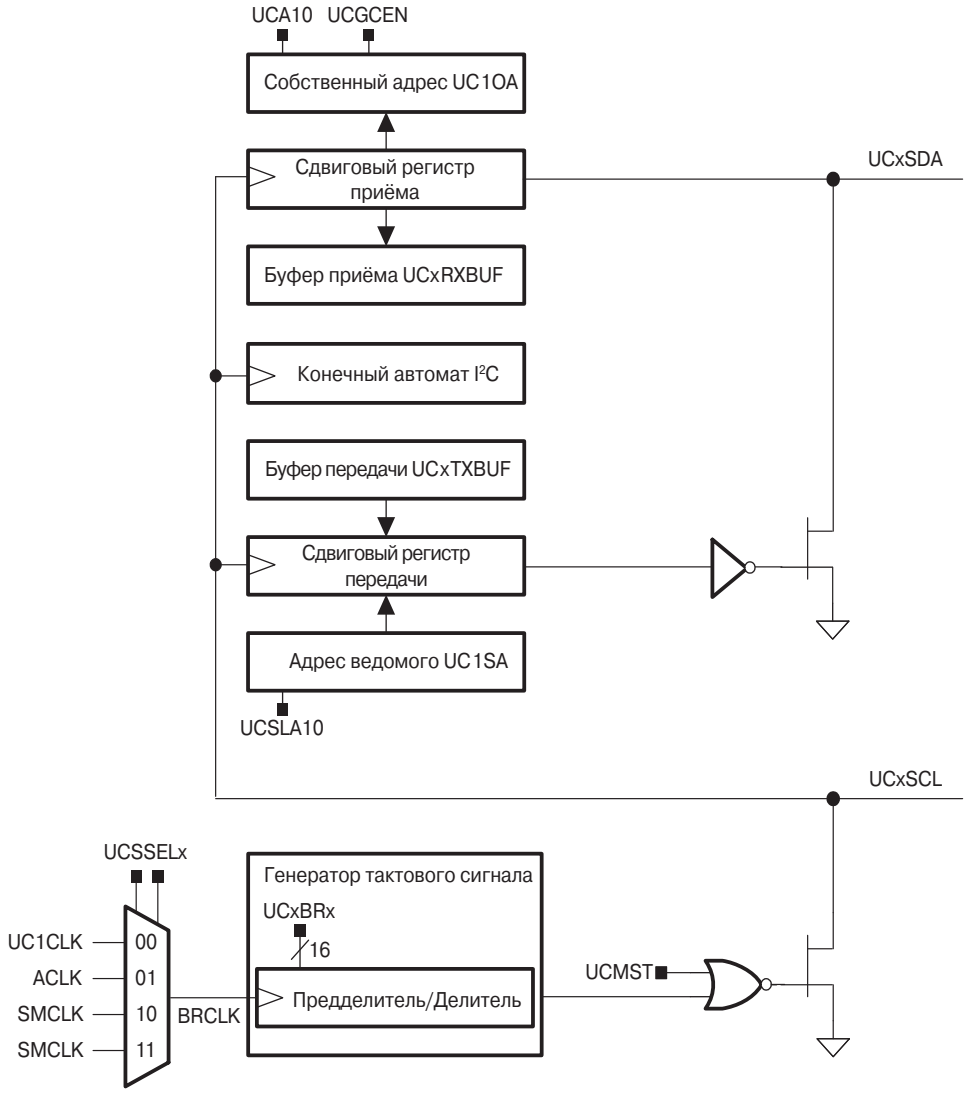


Рис. 17.1. Блок-схема модуля USCI: режим I²C.

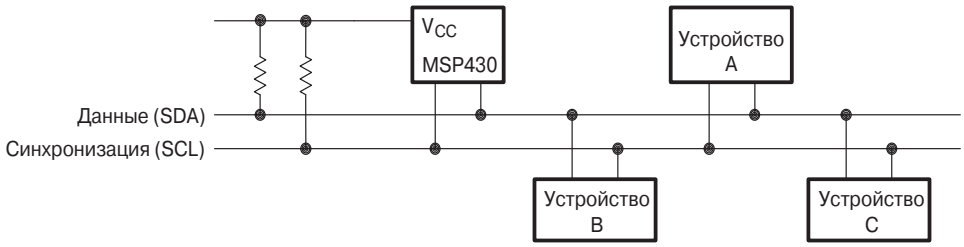


Рис. 17.2. Соединение устройств с помощью шины I²C.

Конфигурирование и реконфигурирование модуля USCI необходимо осуществлять при установленном бите UCSWRST, чтобы предотвратить непредсказуемое поведение модуля. При установке бита UCSWRST в режиме I²C происходит следующее:

- обмен по шине I²C прекращается;
- выводы SCL и SDA переводятся в высокоимпедансное состояние;
- биты 6...0 регистра UCBxI2CSTAT сбрасываются;
- биты UCBxTXIE и UCBxRXIE сбрасываются;
- биты UCBxTXIFG и UCBxRXIFG сбрасываются;
- остальные биты и регистры не изменяются.

Примечание. Инициализация и реконфигурирование модуля USCI

Инициализацию и реконфигурирование модуля USCI рекомендуется выполнять в следующей последовательности:

1. Установить бит UCSWRST (`BIS.B #UCSWRST,&UCxCTL1`).
2. Инициализировать все регистры USCI установкой UCSWRST = 1 (включая регистр UCxCTL1).
3. Сконфигурировать порты ввода/вывода.
4. Сбросить в программе бит UCSWRST (`BIC.B #UCSWRST,&UCxCTL1`).
5. Разрешить прерывания (при необходимости) установкой битов UCxRXIE и/или UCxTXIE.

17.3.2. Передача данных по шине I²C

Для каждого передаваемого бита данных ведущий генерирует один тактовый импульс. В режиме I²C обмен осуществляется побайтно. Данные передаются начиная со старшего значащего бита, как показано на **Рис. 17.3**.

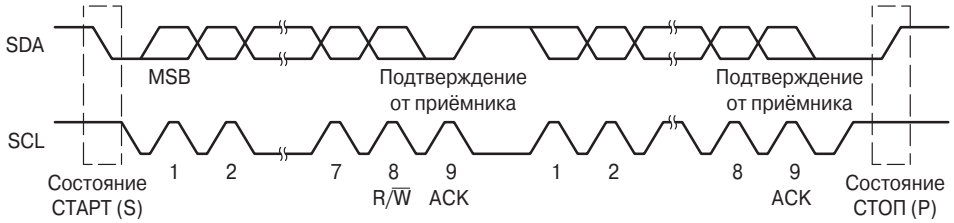


Рис. 17.3. Передача данных по шине I²C.

Первый байт, передаваемый после состояния СТАРТ, содержит 7-битный адрес ведомого и бит направления передачи R/$\overline{W}$. При R/$\overline{W}$ = 0 ведущий передаёт данные ведомому. При R/$\overline{W}$ = 1 ведущий принимает данные от ведомого. После каждого байта приёмник во время 9-го импульса SCL передаёт бит квитирования ACK.

Состояния СТАРТ и СТОП формируются ведущим устройством, как показано на **Рис. 17.3**. Состоянию СТАРТ соответствует изменение уровня на линии SDA с ВЫСОКОГО на НИЗКИЙ при ВЫСОКОМ уровне на линии SCL. Состоянию СТОП соответствует изменение уровня на линии SDA с НИЗКОГО на ВЫСОКИЙ при ВЫСОКОМ уровне на линии SCL. Флаг занятости UCBBUSY устанавливается после формирования состояния СТАРТ и сбрасывается после формирования состояния СТОП.

Сигнал на линии SDA должен быть неизменным в течение всего времени, пока на линии SCL присутствует ВЫСОКИЙ уровень, как показано на **Рис. 17.4**. Уровень сигнала на линии SDA может изменяться только при НИЗКОМ уровне на линии SCL, в противном случае произойдет формирование состояния СТАРТ или СТОП.

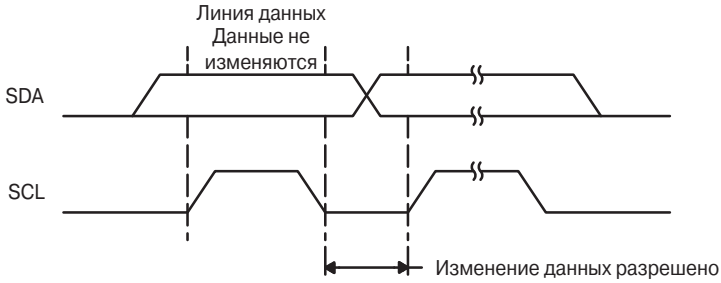


Рис. 17.4. Передача бита по шине I²C.

17.3.3. Режимы адресации I²C

Модуль I²C поддерживает 7- и 10-битный режимы адресации.

7-битная адресация

При использовании 7-битной адресации первый байт пакета содержит 7-битный адрес и бит R/$\overline{W}$, как показано на **Рис. 17.5**. После получения каждого байта приёмник отсылает бит ACK.

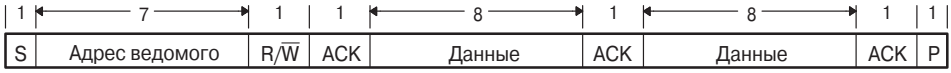


Рис. 17.5. 7-битный режим адресации модуля I²C.

10-битная адресация

При использовании 10-битной адресации (**Рис. 17.6**) первый байт пакета содержит значение 11110b плюс два старших бита 10-битного адреса, а также бит R/$\overline{W}$. Второй байт пакета содержит оставшиеся 8 бит 10-битного адреса. Затем по шине пересылаются 8-битные данные. После получения каждого байта приёмник отсылает бит ACK.

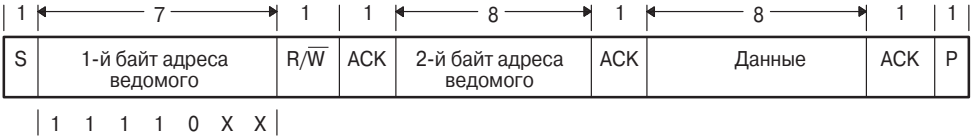


Рис. 17.6. 10-битный режим адресации модуля I²C.

Состояние ПОВСТАРТ

Направление потока данных, передаваемых по линии SDA, может быть изменено ведущим устройством без предварительного останова процесса пересылки. Для этого ведущий повторно формирует состояние СТАРТ, которое в данном случае имеет название ПОВСТАРТ. После ПОВСТАРТ снова передаётся адрес ведомого устройства, но уже с другим значением бита R/$\overline{W}$, определяющего направление пересылки данных. Использование состояния ПОВСТАРТ показано на **Рис. 17.7**.

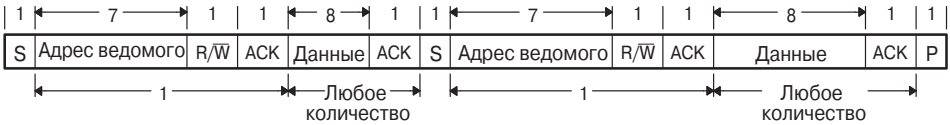


Рис. 17.7. Адресация с использованием состояния ПОВСТАРТ.

17.3.4. Режимы работы модуля I²C

В режиме I²C модуль USCI может работать как ведущий-передатчик, ведущий-приёмник, ведомый-передатчик или ведомый-приёмник. Эти режимы работы рассматриваются в следующих подразделах. Для иллюстрации режимов используются циклограммы.

На **Рис. 17.8** показана расшифровка условных обозначений, используемых в циклограммах. Данные, передаваемые ведущим, изображаются серыми прямоугольниками, а данные, передаваемые ведомым, — белыми. Данные, передаваемые модулем USCI, независимо от его режима работы изображаются прямоугольниками большей высоты.

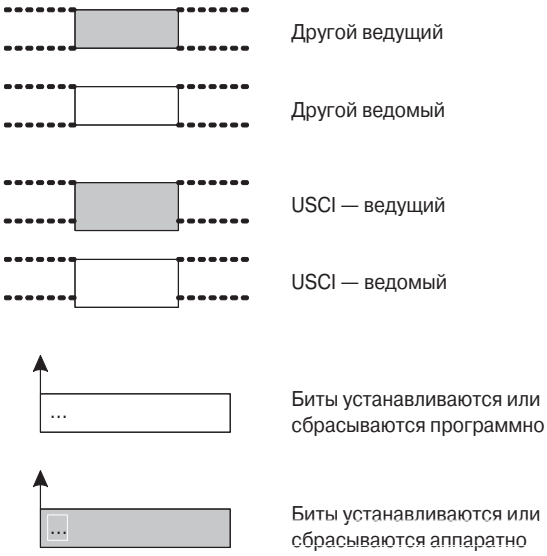


Рис. 17.8. Условные обозначения циклограмм I²C.

Действия, выполняемые модулем USCI, указываются в серых прямоугольниках со стрелкой. Стрелка показывает конкретное место в потоке данных, в котором производятся эти действия. Действия, которые должна выполнять пользовательская программа, изображаются белыми прямоугольниками со стрелкой. Стрелка показывает место в потоке данных, в котором эти действия должны быть выполнены.

Режим ведомого

Для переключения модуля USCI в режим ведомого I^2C необходимо выбрать режим I^2C, установив биты UCMODEx = 11 и UCSYNC = 1, и сбросить бит UCMST.

Сначала модуль USCI должен быть сконфигурирован как приёмник (бит UCTR = 0) для приёма адреса I^2C. Далее режим работы (приёмник/передатчик) будет выбран автоматически в зависимости от значения бита $R/\overline{W}$, полученного вместе с адресом.

Собственный адрес модуля USCI хранится в регистре UCBxI2COA. При UCA10 = 0 используется 7-битная адресация, а при UCA10 = 1 — 10-битная. Состояние бита UCGCEN определяет реакцию модуля на общие вызовы.

После того как модуль USCI обнаружит на шине состояние СТАРТ, он принимает адрес, переданный ведущим, и сравнивает его со своим адресом, хранящимся в регистре UCBxI2COA. При равенстве полученного адреса и собственного адреса модуля USCI будет установлен флаг прерывания UCSTTIFG.

Режим «ведомый-передатчик» модуля I^2C

Ведомое устройство переключается в режим передачи в том случае, если адрес, переданный ведущим, совпадает с собственным адресом модуля и бит $R/\overline{W}$ установлен. Ведомый-передатчик побитно выставляет данные на линию SDA синхронно с тактовыми импульсами, генерируемыми ведущим устройством. Ведомое устройство не генерирует тактовый сигнал, однако может удерживать НИЗКИЙ уровень на линии SCL, если после передачи очередного байта требуется вмешательство ЦПУ.

Если ведущий запрашивает данные от ведомого устройства, то модуль USCI автоматически конфигурируется как передатчик, при этом устанавливаются биты UCTR и UCBxTXIFG. Модуль удерживает НИЗКИЙ уровень на линии SCL до тех пор, пока первый передаваемый байт данных не будет загружен в буфер передачи UCBxTXBUF. После этого формируется подтверждение приёма адреса, сбрасывается флаг UCBxTXIFG и начинается передача данных. Одновременно с перегрузкой содержимого буфера в сдвиговый регистр передачи снова устанавливается флаг UCBxTXIFG. После того как ведущий подтвердит получение данных, передаётся следующий байт, находящийся в регистре буфера UCBxTXBUF. Если буфер пуст, то процесс обмена приостанавливается (удержанием НИЗКОГО уровня на линии SCL после получения бита квитирования) до тех пор, пока в UCBxTXBUF не будут загружены новые данные. При посылке ведущим неподтверждения (NACK) с последующим формированием состояния СТОП устанавливается флаг UCSTPIFG. Если после бита NACK формируется состояние ПОВ-

СТАРТ, то конечный автомат I²C модуля USCI возвращается в состояние ожидания приёма адреса.

Работа модуля в режиме «ведомый-передатчик» проиллюстрирована на **Рис. 17.9**.

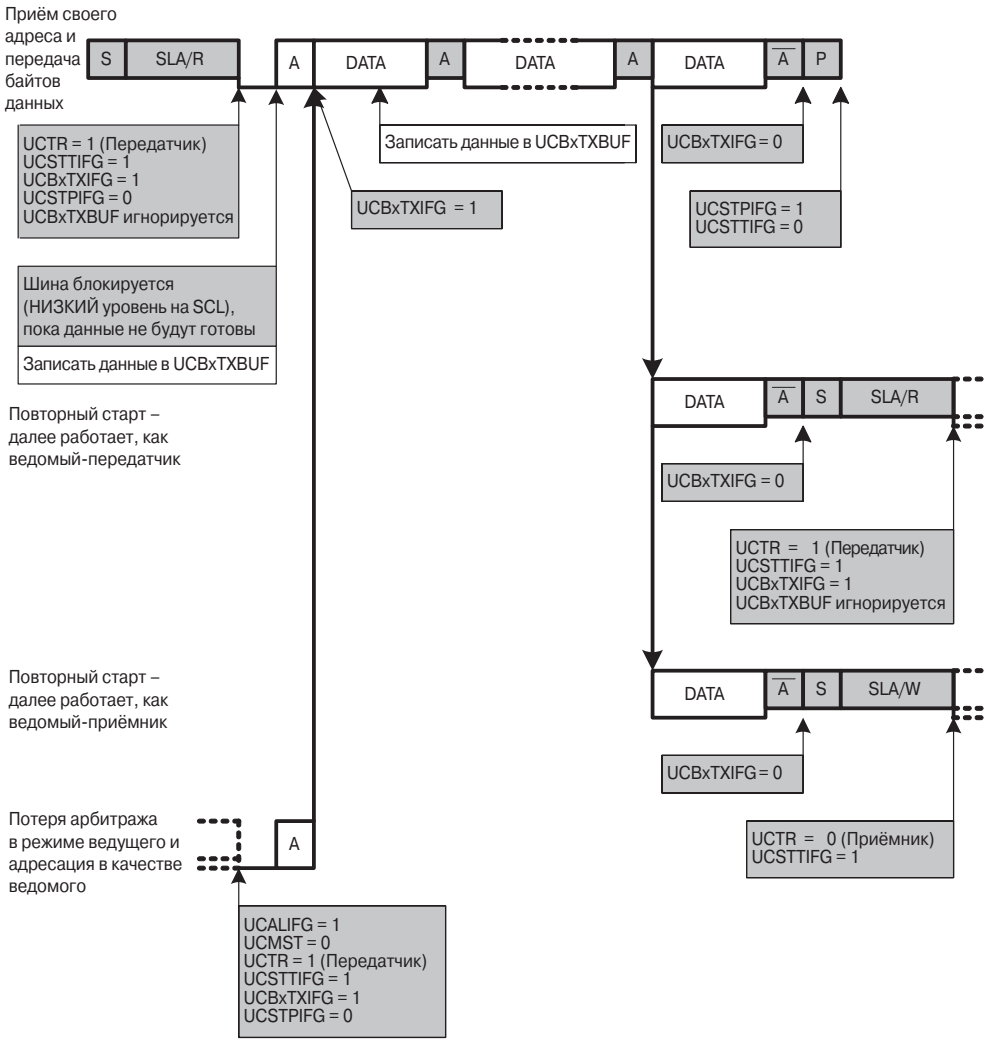


Рис. 17.9. Режим «ведомый-передатчик» модуля I²C.

Режим «ведомый-приёмник» модуля I²C

Модуль переключается в режим «ведомый-приёмник» в том случае, если адрес, переданный ведущим, совпадает с собственным адресом модуля, а бит R/$\overline{W}$ сброшен. В этом режиме биты данных, передаваемые по линии SDA, считываются модулем синхронно с тактовыми импульсами, генерируемыми ведущим устройством. Ведомое устройство не генерирует тактовый сигнал, однако оно может

удерживать НИЗКИЙ уровень на линии SCL, если после приёма очередного байта требуется вмешательство ЦПУ.

Если ведомый должен принимать данные от ведущего, то модуль USCI автоматически конфигурируется как приёмник, при этом сбрасывается бит UCTR. После приёма первого байта данных устанавливается флаг прерывания приёма UCBxRXIFG. Модуль автоматически подтверждает получение данных и переходит к ожиданию следующего байта данных.

Если к моменту завершения приёма байта ранее принятое значение не было считано из буфера приёма UCBxRXBUF, то обмен на шине приостанавливается удержанием НИЗКОГО уровня на линии SCL. Сразу же после чтения регистра UCBxRXBUF в него перегружаются новые данные, отсылается подтверждение ведущему устройству и разрешается приём следующих данных.

Установленный бит UCTXNACK вызовет передачу ведущему устройству сигнала NACK в такте квитирования. Сигнал NACK отсылается даже в том случае, если буфер UCBxRXBUF не готов к приёму последних данных. Если бит UCTXNACK устанавливается в то время, пока модуль удерживает на линии SCL НИЗКИЙ уровень, то шина освобождается, сразу же передаётся сигнал NACK, а в буфер UCBxRXBUF загружаются последние принятые данные. Поскольку предыдущие данные не были считаны, они теряются. Чтобы избежать потери данных, необходимо перед установкой бита UCTXNACK выполнить чтение регистра UCBxRXBUF.

При формировании ведущим состояния СТОП устанавливается флаг UCSTPIFG.

Если ведущий формирует состояние ПОВСТАРТ, то конечный автомат I^2C модуля USCI возвращается в состояние ожидания приёма адреса.

Работа модуля в режиме «ведомый-приёмник» проиллюстрирована на **Рис. 17.10**.

Ведомый I^2C и режим 10-битной адресации

Режим 10-битной адресации выбирается при UCA10 = 1; работа ведомого с использованием 10-битной адресации проиллюстрирована на **Рис. 17.11**. При 10-битной адресации ведомое устройство после получения полного адреса находится в режиме «ведомый-приёмник». Модуль USCI сигнализирует об этом установкой флага UCSTTIFG с одновременным сбросом бита UCTR. Для перевода модуля в режим «ведомый-передатчик» ведущее устройство формирует состояние ПОВСТАРТ и передаёт первый байт адреса с установленным битом $R/\overline{W}$. При получении этого байта устанавливается флаг UCSTTIFG (если ранее он уже был сброшен программой), и модуль USCI переключается в режим передатчика с одновременной установкой бита UCTR = 1.

Режим ведущего

Для переключения модуля USCI в режим ведущего I^2C необходимо выбрать режим I^2C, установив биты UCMODEx = 11 и UCSYNC = 1, а также установить бит UCMST. При наличии в системе нескольких ведущих устройств должен быть установлен бит UCMM, а в регистр UCBxI2COA — загружен собственный адрес модуля. При UCA10 = 0 используется 7-битная адресация, а при UCA10 = 1 —

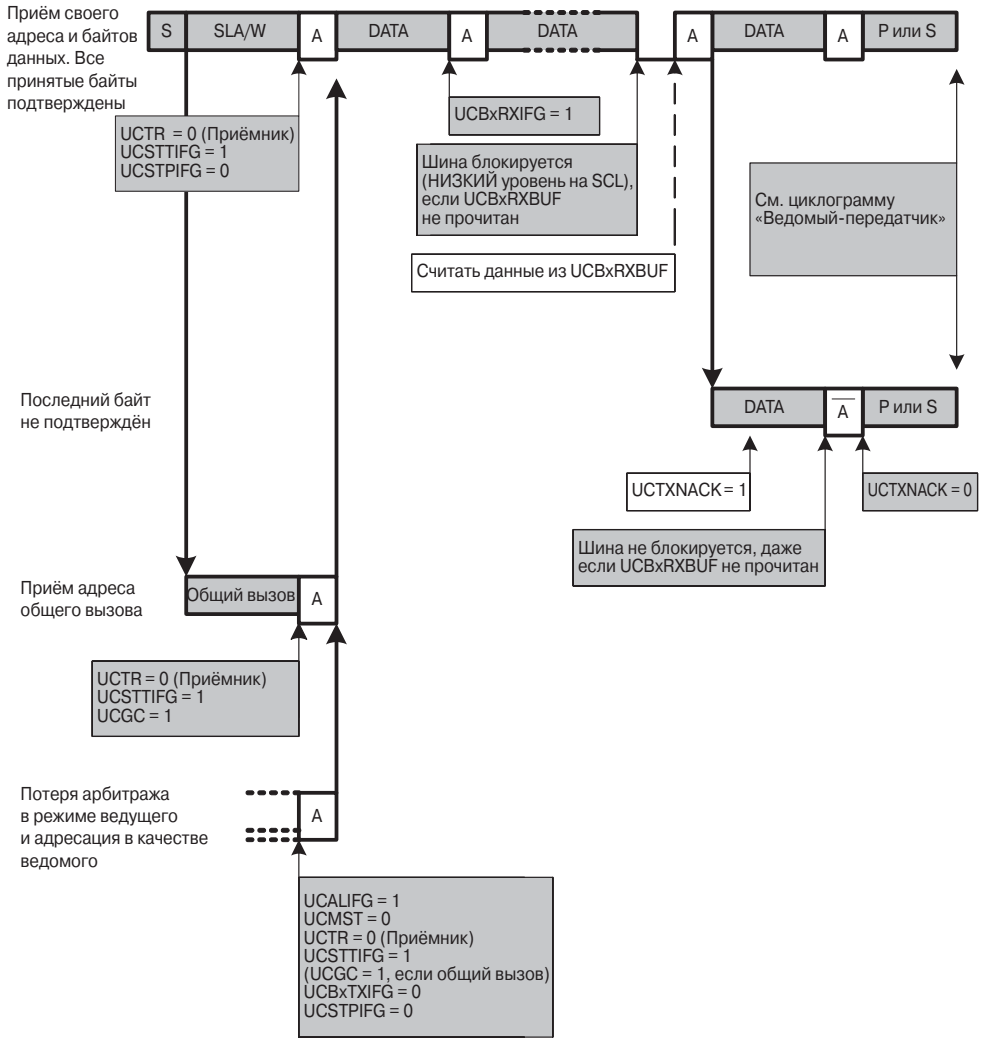


Рис. 17.10. Режим «ведомый-приёмник» модуля I²C.

10-битная адресация. Состояние бита UCGCEN определяет реакцию модуля на общие вызовы.

Режим «ведущий-передатчик» модуля I²C

После инициализации модуля переключение последнего в режим «ведущий-передатчик» производится записью в регистр UCBxI2CSA адреса искомого ведомого устройства, выбором разрядности адреса ведомого при помощи бита UCSLA10, установкой бита UCTR для включения режима передатчика и установкой бита UCTXSTT для формирования состояния СТАРТ.

Модуль USCI проверяет доступность шины, формирует состояние СТАРТ и передаёт адрес ведомого устройства. После того как сформировано состояние

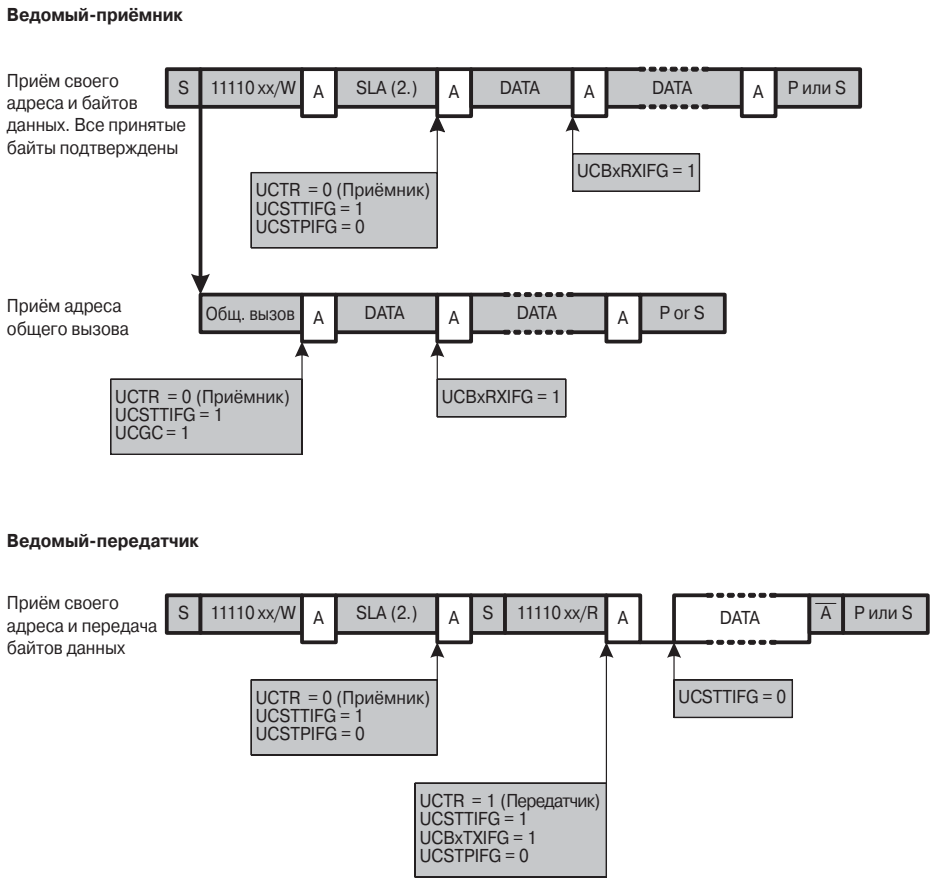


Рис. 17.11. Ведомый I²C и режим 10-битной адресации.

СТАРТ и в регистр UCBxTXBUF может быть загружен первый байт передаваемых данных, устанавливается бит UCBxTXIFG. Сразу же после подтверждения ведомым устройством отправленного адреса бит UCTXSTT сбрасывается.

Если во время передачи адреса не было потери арбитража, то модуль начинает передавать данные, загруженные в регистр UCBxTXBUF. Одновременно с перегрузкой содержимого буфера в сдвиговый регистр снова устанавливается флаг UCBxTXIFG. Если данные не были загружены в регистр UCBxTXBUF до начала фазы квитирования, то процесс обмена по шине приостанавливается удержанием НИЗКОГО уровня на линии SCL до записи данных в регистр UCBxTXBUF. Этот процесс (передачи данных или удержания шины) продолжается до тех пор, пока не будет установлен бит UCTXSTP или UCTXSTT.

Установка бита UCTXSTP вызывает формирование состояния СТОП после получения следующего подтверждения от ведомого устройства. Если бит UCTXSTP устанавливается во время передачи адреса ведомого или в то время, пока модуль ожидает записи данных в буфер UCBxTXBUF, состояние СТОП будет сформировано даже в том случае, если данные ведомому устройству не пере-

давались. Если необходимо передать единственный байт данных, то бит UCSTP должен быть установлен либо во время пересылки этого байта, либо в любое время после начала передачи без загрузки новых данных в UCBxTXBUF. В противном случае будет передан только адрес. При перегрузке данных из буфера в сдвиговый регистр устанавливается флаг UCBxTXIFG, сообщающий о том, что началась передача данных и бит UCTXSTP может быть установлен.

Установка бита UCTXSTT вызывает формирование состояния ПОВСТАРТ. При этом установкой или сбросом бита UCTR может быть изменён режим работы модуля, а в регистр UCBxI2CSA, при необходимости, может быть загружен новый адрес ведомого устройства.

Если ведомое устройство не подтверждает переданные ему данные, то устанавливается флаг отсутствия подтверждения UCNACKIFG. При установке этого флага ведущий должен сформировать состояние СТОП либо ПОВСТАРТ. Если в регистр UCBxTXBUF уже были загружены данные, то они теряются. Если эти данные должны быть переданы после ПОВСТАРТ, их необходимо снова записать в UCBxTXBUF. Значение бита UCTXSTT также игнорируется. Для формирования состояния ПОВСТАРТ бит UCTXSTT необходимо установить повторно.

Работа модуля в режиме «ведущий-передатчик» проиллюстрирована на **Рис. 17.12**.

Режим «ведущий-приёмник» модуля I^2C

После инициализации модуля переключение последнего в режим «ведущий-приёмник» производится записью в регистр UCBxI2CSA адреса искомого ведомого устройства, выбором разрядности адреса ведомого при помощи бита UCSLA10, сбросом бита UCTR для включения режима передатчика и установкой бита UCTXSTT для формирования состояния СТАРТ.

Модуль USCI проверяет доступность шины, формирует состояние СТАРТ и передаёт адрес ведомого устройства. Сразу же после подтверждения ведомым устройством отправленного адреса бит UCTXSTT сбрасывается.

После получения подтверждения принимается первый байт данных, отправленный ведомым, отсылается подтверждение и устанавливается флаг UCBxRXIFG. Приём данных от ведомого устройства осуществляется до тех пор, пока не будет установлен бит UCTXSTP или UCTXSTT. Если чтение регистра UCBxRXBUF не производится, то ведущий при приёме последнего бита данных удерживает на линии SCL НИЗКИЙ уровень до тех пор, пока этот регистр не будет прочитан.

Если ведомое устройство не подтверждает переданный ему адрес, то устанавливается флаг отсутствия подтверждения UCNACKIFG. При установке этого флага ведущий должен сформировать состояние СТОП либо ПОВСТАРТ.

Установка бита UCXSTP вызывает формирование состояния СТОП. При установке этого бита отсылается сигнал NACK, после чего формируется состояние СТОП. Эти действия выполняются после получения следующего байта данных от ведомого или, если модуль ожидает чтения регистра UCxRXBUF, сразу же после установки бита.

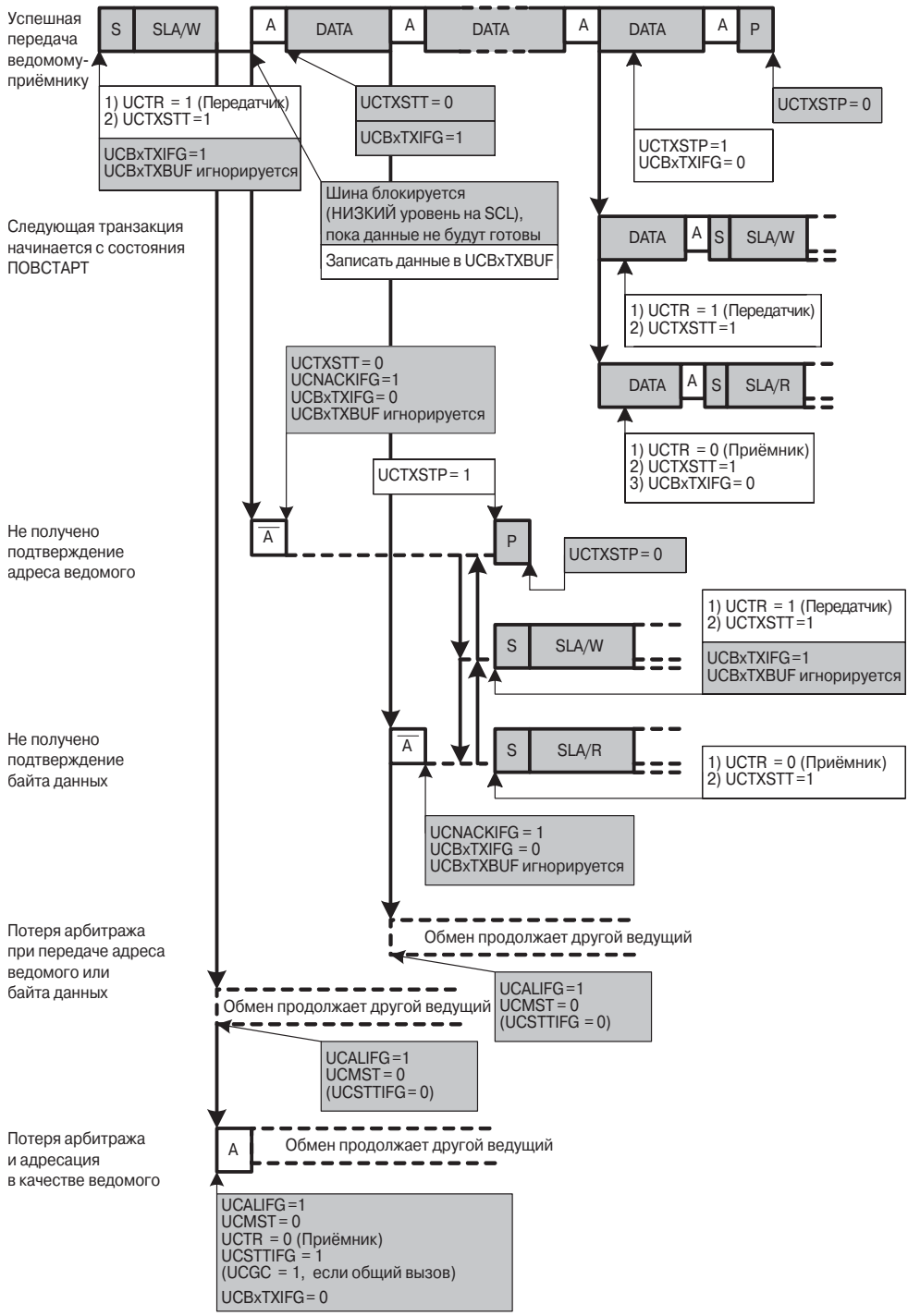


Рис. 17.12. Режим «ведущий-передатчик» модуля I²C.

Если ведущему необходимо принять единственный байт данных, то бит UCTXSTP должен быть установлен во время приёма этого байта. Данный момент можно определить по состоянию бита UCTXSTT:

```
         BIS.B  #UCTXSTT,&UCBOCTL1 ; Формируем состояние СТАРТ
POLL_STT BIT.B  #UCTXSTT,&UCBOCTL1 ; Читаем бит UCTXSTT
         JC     POLL_STT           ; При сбросе бита
         BIS.B  #UCTXSTP,&UCB0CTL1 ; формируем состояние СТОП
```

Установка бита UCTXSTT вызывает формирование состояния ПОВСТАРТ. При этом установкой или сбросом бита UCTR может быть изменён режим работы модуля, а в регистр UCBxI2CSA, при необходимости, может быть загружен новый адрес ведомого устройства.

Работа модуля в режиме «ведущий-приёмник» проиллюстрирована на **Рис. 17.13**.

Ведущий I²C и режим 10-битной адресации

Режим 10-битной адресации выбирается при UCA10 = 1; работа ведущего с использованием 10-битной адресации проиллюстрирована на **Рис. 17.14**.

Арбитраж

Если два или более ведущих устройств, подключённых к шине, одновременно начинают передачу, то запускается механизм арбитража. На **Рис. 17.15** показан процесс арбитража между двумя устройствами. При арбитраже используются значения битов, выставляемые на линию SDA конкурирующими передатчиками. Первый ведущий-передатчик, выставивший на линию сигнал ВЫСОКОГО уровня, блокируется другим ведущим-передатчиком, выставившим НИЗКИЙ уровень. Таким образом, в процессе арбитража приоритет отдаётся устройству, передающему битовые данные с наименьшим значением. Ведущий, потерявший контроль над шиной, переключается в режим «ведомый-приёмник» и устанавливает флаг потери арбитража UCALIFG. Если первые байты, передаваемые двумя и более устройствами, одинаковы, то процесс арбитража продолжается при передаче последующих байтов.

Если во время арбитража на шине будет сформировано состояние СТАРТ или СТОП, то все ведущие-передатчики, принимающие участие в этом процессе, сразу же должны сформировать состояние СТОП или ПОВСТАРТ. Арбитраж не должен выполняться между:

- состоянием ПОВСТАРТ и передачей бита данных;
- состоянием СТОП и передачей бита данных;
- состояниями ПОВСТАРТ и СТОП.

17.3.5. Генерация и синхронизация тактового сигнала I²C

Сигнал синхронизации на линии SCL генерируется ведущим шины I²C. При работе модуля USCI в режиме ведущего, тактовый сигнал BITCLK формируется генератором модуля из его тактового сигнала. Источник тактового сигнала модуля USCI задаётся битами UCSSELx. В режиме ведомого генератор не используется и состояние битов UCSSELx безразлично.

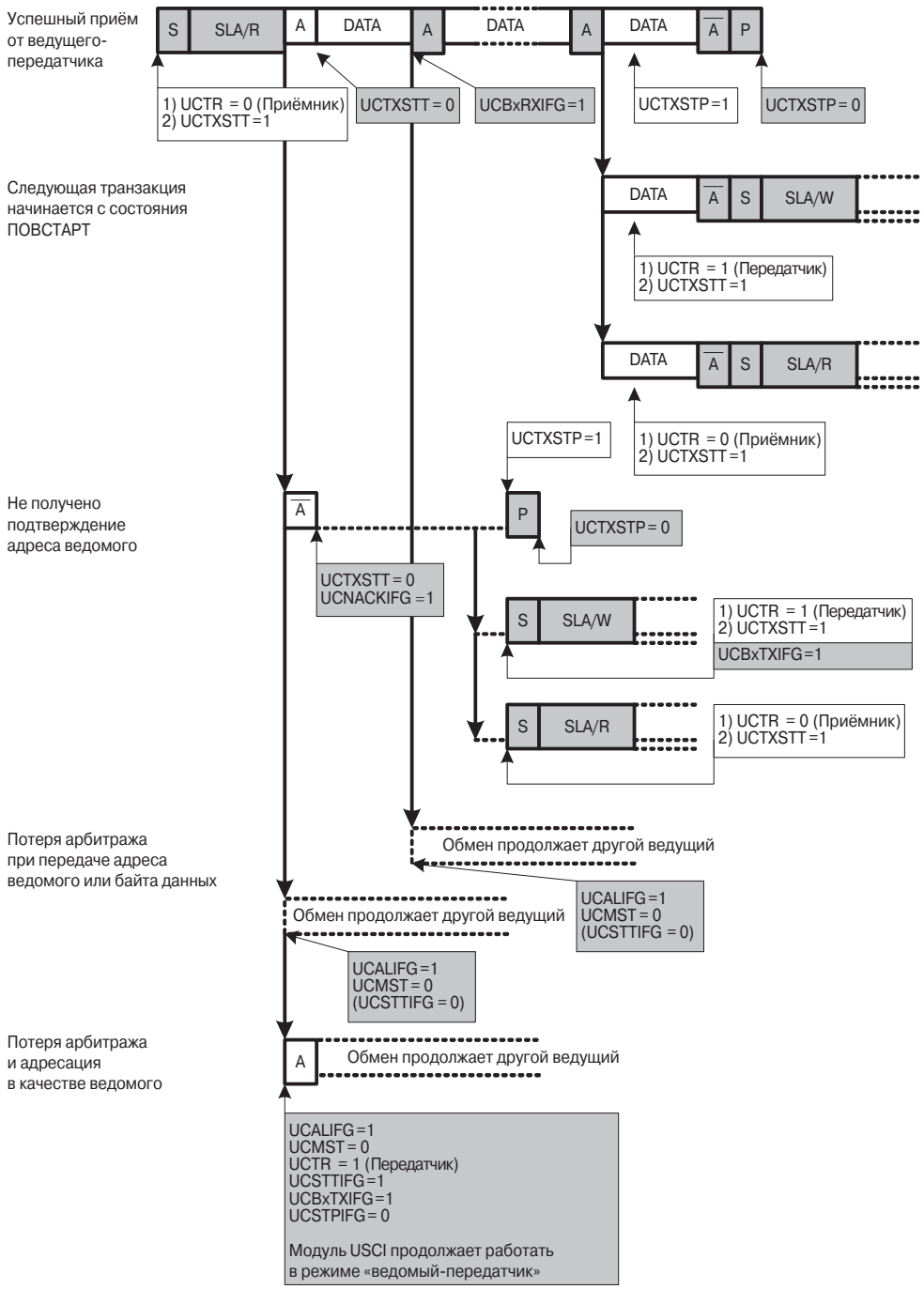


Рис. 17.13. Режим «ведущий-приёмник» модуля I²C.

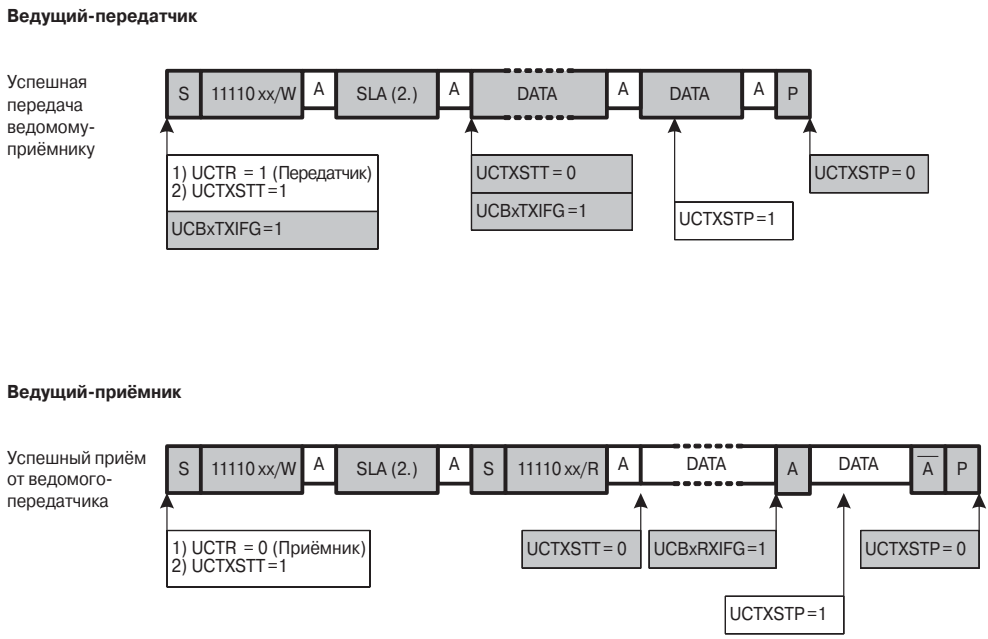


Рис. 17.14. Ведущий I²C и режим 10-битной адресации.

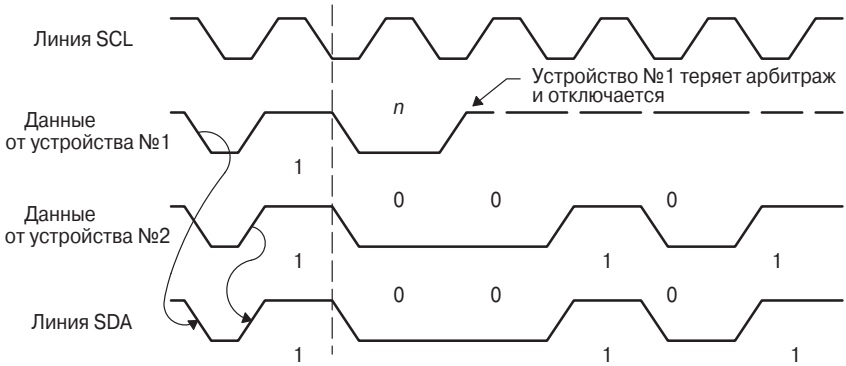


Рис. 17.15. Процесс арбитража между двумя ведущими.

В регистрах UCBxBR1 и UCBxBR0 содержится 16-битное значение UCBRx, представляющее собой коэффициент деления тактового сигнала модуля USCI (BRCLK). При работе в системе с одним ведущим на шине частота сигнала BITCLK не должна превышать значения $f_{BRCLK}/4$. При наличии на шине нескольких ведущих максимальная частота сигнала составляет $f_{BRCLK}/8$. Частота тактового сигнала BITCLK определяется выражением:

$$f_{BITCLK} = \frac{f_{BRCLK}}{UCBRx}.$$

Минимальная длительность импульсов и пауз тактового сигнала на линии SCL составляет:

$$t_{\mathrm{LOW,MIN}} = t_{\mathrm{HIGH,MIN}} = \frac{\mathrm{UCBRx}/2}{f_{\mathrm{BRCLK}}}$$ при чётном значении UCBRx и

$$t_{\mathrm{LOW,MIN}} = t_{\mathrm{HIGH,MIN}} = \frac{(\mathrm{UCBRx}-1)/2}{f_{\mathrm{BRCLK}}}$$ при нечётном значении UCBRx.

Частота тактового сигнала модуля USCI и коэффициент деления предделителя должны быть выбраны таким образом, чтобы параметры сигнала на шине SCL удовлетворяли требованиям спецификации I²C.

Во время арбитража тактовые сигналы от различных ведущих должны быть синхронизированы. Устройство, первым формирующее НИЗКИЙ уровень на линии SCL, вынуждает остальных ведущих начать формирование паузы тактового сигнала. Максимальная длительность сигнала НИЗКОГО уровня на линии SCL определяется устройством, генерирующим сигнал с наибольшей паузой между импульсами. Остальные устройства должны ждать освобождения линии SCL, чтобы начать формирование импульсов тактовых сигналов. Это позволяет медленным устройствам замедлить более быстрого ведущего. Процесс синхронизации тактовых сигналов показан на **Рис. 17.16**.

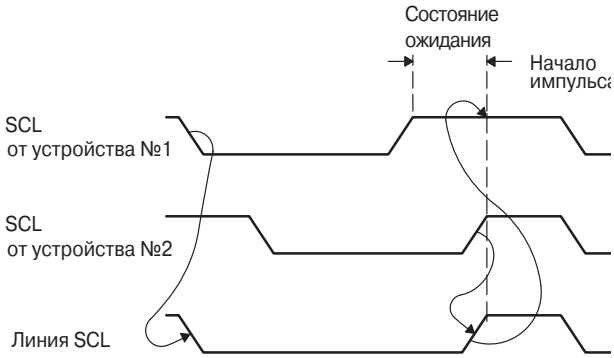


Рис. 17.16. Синхронизация двух тактовых сигналов I²C во время арбитража.

Задержка импульсов синхронизации

Модуль USCI поддерживает задержку импульсов синхронизации — использование этой возможности было описано в разделах, посвящённых описанию режимов работы модуля.

Модуль USCI, освободивший линию SCL, может использовать бит UCSLLOW, чтобы проверить наличие других устройств, удерживающих на линии SCL НИЗКИЙ уровень. Такая ситуация может возникнуть, если:

- модуль USCI работает в режиме ведущего, и подключённое к шине ведомое устройство выставило на SCL НИЗКИЙ уровень;

- модуль USCI работает в режиме ведущего, и другое ведущее устройство выставило на SCL НИЗКИЙ уровень во время арбитража.

Кроме того, бит USCLLOW устанавливается при удержании модулем НИЗКОГО уровня на линии SCL по причине ожидания записи данных в регистр UCBxTXBUF (в режиме передатчика) или чтения данных из регистра UCBxRXBUF (в режиме приёмника).

Возможна кратковременная установка бита USCLLOW по нарастающему фронту сигнала SCL из-за работы схемы, сравнивающей состояние линии с сигналом SCL, генерируемым модулем.

17.3.6. Использование модуля USCI в режиме I²C совместно с режимами пониженного энергопотребления

Модуль USCI обеспечивает автоматическую активацию тактового сигнала SMCLK, что делает возможным использование модуля в режимах пониженного энергопотребления. Если модуль USCI тактируется сигналом SMCLK, который неактивен по причине нахождения устройства в режиме пониженного энергопотребления, то модуль автоматически активирует данный тактовый сигнал при необходимости, независимо от значений управляющих битов выбора источника системного тактового сигнала. Указанный тактовый сигнал будет активным до тех пор, пока модуль USCI не вернётся в состояние ожидания. После этого контроль над источником тактового сигнала вернётся к модулю синхронизации. Автоматическая активация тактового сигнала ACLK не предусмотрена.

При активации модулем USCI неактивного источника тактового сигнала этот источник становится доступным для всего устройства, так что его активация может затронуть любой периферийный модуль, сконфигурированный для использования данного тактового сигнала. Например, значение таймера, использующего SMCLK, будет инкрементироваться в течение всего времени, пока модуль USCI будет находиться в активном режиме.

В режиме ведомого I²C наличие внутреннего тактового сигнала не требуется, поскольку сигнал синхронизации формируется внешним ведущим устройством. Это даёт возможность использовать модуль USCI в качестве ведомого I²C-устройства даже при использовании режима пониженного энергопотребления LPM4, в котором выключены все источники тактовых сигналов. Вывод ЦПУ из любого режима пониженного энергопотребления возможен по прерываниям приёма или передачи.

17.3.7. Прерывания в режиме I²C

Модуль USCI имеет два вектора прерывания. Один из векторов в режиме I²C связан с флагами прерываний передачи и приёма. Другой вектор связан с четырьмя флагами прерываний, отображающими изменение состояния. Каждому флагу соответствует индивидуальный бит разрешения прерывания. При разрешённом прерывании и установленном бите GIE установка флага генерирует запрос прерывания. В устройствах, имеющих контроллер DMA, пересылка данных управляется флагами UCBxTXIFG и UCBxRXIFG.

Прерывание I²C при передаче

Флаг прерывания UCBxTXIFG, устанавливаемый передатчиком, показывает готовность регистра UCBxTXBUF к загрузке нового значения. Если установлены биты GIE и UCBxTXIE, то при установке флага UCBxTXIFG генерируется запрос прерывания. Флаг UCBxTXIFG автоматически сбрасывается при записи в регистр UCBxTXBUF или при приёме сигнала NACK. Флаг UCBxTXIFG устанавливается при UCSWRST = 1 и выборе режима I²C. Бит разрешения прерывания UCBxTXIE сбрасывается после сигнала сброса PUC или при UCSWRST = 1.

Прерывание I²C при приёме

Флаг прерывания UCBxRXIFG устанавливается каждый раз при приёме символа и загрузке его в регистр UCBxRXBUF. При установленных битах GIE и UCBxRXIE генерируется запрос прерывания. Биты UCBxRXIFG и UCBxRXIE сбрасываются сигналом системного сброса PUC или при UCSWRST = 1. Флаг UCBxRXIFG автоматически сбрасывается при чтении регистра UCBxRXBUF.

Прерывания при изменении состояния

Флаги прерываний, устанавливаемые при изменении состояния модуля и/или шины, перечислены в **Табл. 17.1**.

Таблица 17.1. **Флаги прерываний по изменению состояния**

Флаг прерывания	Условие генерации прерывания
UCALIFG	Потеря арбитража. Арбитраж может быть утерян, когда два или более ведущих на шине одновременно начинают передачу или когда модуль USCI, работающий в режиме ведущего, оказывается адресован в качестве ведомого другим ведущим. При потере арбитража устанавливается флаг UCALIFG. При установке бита UCALIFG сбрасывается бит UCMST и контроллер I²C переключается в режим ведомого
UCNACKIFG	Нет подтверждения. Этот флаг устанавливается, когда модуль не получает ожидаемого подтверждения. Флаг UCNACKIFG автоматически сбрасывается при обнаружении состояния СТАРТ
UCSTTIFG	Обнаружено состояние СТАРТ. Этот флаг устанавливается, когда модуль I²C, работающий в режиме ведомого, обнаруживает состояние СТАРТ с последующим приёмом своего адреса. Флаг UCSTTIFG используется только в режиме ведомого и автоматически сбрасывается при обнаружении состояния СТОП
UCSTPIFG	Обнаружено состояние СТОП. Этот флаг устанавливается, когда модуль I²C, работающий в режиме ведомого, обнаруживает состояние СТОП. Флаг UCSTPIFG используется только в режиме ведомого и автоматически сбрасывается при обнаружении состояния СТАРТ

Использование прерываний модуля USCI

Модули USCI_Ax и USCI_Bx используют одни и те же векторы прерываний. В режиме I²C флаги прерываний по изменению состояния UCSTTIFG, UCSTPIFG, UCIFG, UCALIFG модуля USCI_Bx и флаг приёма UCAxRXIFG модуля USCI_Ax связаны с одним вектором прерываний. Флаги прерываний передачи UCBxTXIFG и приёма UCBxRXIFG модуля USCI_Bx и флаг прерывания передачи UCAxTXIFG модуля USCI_Ax связаны с другим вектором.

Примеры использования разделяемого вектора прерывания

В следующем примере содержится фрагмент процедуры обработки прерывания, отвечающей за обработку прерываний приёма модуля USCI_A0, работающего в режиме UART или SPI, и прерываний по изменению состояния модуля USCI_B0, работающего в режиме I²C.

```
USCIA0_RX_USCIB0_I2C_STATE_ISR
  BIT.B  #UCA0RXIFG, &IFG2   ; Прерывание приёма USCI_A0?
  JNZ    USCIA0_RX_ISR
USCIB0_I2C_STATE_ISR
  ; Уточняем изменение состояния I2C ...
  ; Уточняем изменение состояния I2C ...
  ...
  RETI
USCIA0_RX_ISR
  ; Читаем UCA0RXBUF ... - сбрасывается UCA0RXIFG
  ...
  RETI
```

В следующем примере содержится фрагмент процедуры обработки прерывания, отвечающей за обработку прерываний передачи модуля USCI_A0, работающего в режиме UART или SPI, и прерываний передачи модуля USCI_B0, работающего в режиме I²C.

```
USCIA0_TX_USCIB0_I2C_DATA_ISR
  BIT.B  #UCA0TXIFG, &IFG2   ; Прерывание передачи USCI_A0?
  JNZ    USCIA0_TX_ISR
USCIB0_I2C_DATA_ISR
  BIT.B  #UCB0RXIFG, &IFG2
  JNZ    USCIB0_I2C_RX
USCIB0_I2C_TX
  ; Пишем в UCB0TXBUF... - сбрасывается UCB0TXIFG
  ...
  RETI
USCIB0_I2C_RX
  ; Читаем UCB0RXBUF... - сбрасывается UCB0RXIFG
  ...
  RETI
USCIA0_TX_ISR
  ; Пишем в UCA0TXBUF ... - сбрасывается UCA0TXIFG
  ...
  RETI
```

17.4. Регистры модуля USCI: режим I²C

Список регистров модулей USCI_B0 и USCI_B1, использующихся в режиме SPI, приведён в **Табл. 17.2** и в **Табл. 17.3**.

Таблица 17.2. **Регистры управления и состояния модуля USCI_B0**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USCI_B0	UCB0CTL0	Чтение/запись	068h	001h после PUC
Регистр управления 1 модуля USCI_B0	UCB0CTL1	Чтение/запись	069h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_B0	UCB0BR0	Чтение/запись	06Ah	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_B0	UCB0BR1	Чтение/запись	06Bh	Сбрасывается после PUC
Регистр разрешения прерываний I^2C модуля USCI_B0	UCB0I2CIE	Чтение/запись	06Ch	Сбрасывается после PUC
Регистр состояния модуля USCI_B0	UCB0STAT	Чтение/запись	06Dh	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_B0	UCB0RXBUF	Чтение	06Eh	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_B0	UCB0TXBUF	Чтение/запись	06Fh	Сбрасывается после PUC
Регистр собственного адреса I^2C модуля USCI_B0	UCB0I2COA	Чтение	0118h	Сбрасывается после PUC
Регистр адреса I^2C ведомого модуля USCI_B0	UCB0I2CSA	Чтение/запись	011Ah	Сбрасывается после PUC
Регистр разрешения прерываний 2	IE2	Чтение/запись	001h	Сбрасывается после PUC
Регистр флагов прерываний 2	IFG2	Чтение/запись	003h	00Ah после PUC

Таблица 17.3. **Регистры управления и состояния модуля USCI_B1**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля USCI_B1	UCB1CTL0	Чтение/запись	0D8h	001h после PUC
Регистр управления 1 модуля USCI_B1	UCB1CTL1	Чтение/запись	0D9h	001h после PUC
Регистр управления 0 скоростью обмена модуля USCI_B1	UCB1BR0	Чтение/запись	0DAh	Сбрасывается после PUC
Регистр управления 1 скоростью обмена модуля USCI_B1	UCB1BR1	Чтение/запись	0DBh	Сбрасывается после PUC
Регистр разрешения прерываний I^2C модуля USCI_B1	UCB1I2CIE	Чтение/запись	0DCh	Сбрасывается после PUC
Регистр состояния модуля USCI_B1	UCB1STAT	Чтение/запись	0DDh	Сбрасывается после PUC
Регистр буфера приёма модуля USCI_B1	UCB1RXBUF	Чтение	0DEh	Сбрасывается после PUC
Регистр буфера передачи модуля USCI_B1	UCB1TXBUF	Чтение/запись	0DFh	Сбрасывается после PUC
Регистр собственного адреса I^2C модуля USCI_B1	UCB1I2COA	Чтение	017Ch	Сбрасывается после PUC
Регистр адреса I^2C ведомого модуля USCI_B1	UCB1I2CSA	Чтение/запись	017Eh	Сбрасывается после PUC
Регистр разрешения прерываний модулей USCI_A1/B1	UC1IE	Чтение/запись	006h	Сбрасывается после PUC
Регистр флагов прерываний модулей USCI_A1/B1	UC1IFG	Чтение/запись	007h	00Ah после PUC

Примечание. Изменение битов регистров управления прерываниями
Чтобы исключить изменение управляющих битов, используемых другими модулями, для установки или сброса битов IEx и IFGx рекомендуется вместо команд `MOV.B` и `CLR.B` применять команды `BIS.B` и `BIC.B`.

UCBxCTL0, регистр управления 0 модуля USCI_Bx

7	6	5	4	3	2	1	0
UCA10	**UCSLA10**	**UCMM**	**Unused**	**UCMST**	**UCMODEx**		**UCSYNC = 1**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	r–1

UCA10 Бит 7 Разрядность собственного адреса.
- 0 Собственный адрес 7-битный
- 1 Собственный адрес 10-битный

UCSLA10 Бит 6 Разрядность передаваемого адреса ведомого.
- 0 Адрес ведомого 7-битный
- 1 Адрес ведомого 10-битный

UCMM Бит 5 Режим с несколькими ведущими.
- 0 Один ведущий на шине. Блок сравнения адреса выключен.
- 1 Несколько ведущих на шине

Unused Бит 4 Не используется.

UCMST Биты 3...2 Выбор режима ведущего. В случае потери ведущим арбитража в системе с несколькими ведущими (UCMM = 1) бит UCMST автоматически сбрасывается и модуль переключается в режим ведомого.
- 0 Режим ведомого
- 1 Режим ведущего

UCMODEx Бит 1 Режим работы модуля USCI. Биты UCMODEx задают синхронный режим работы при UCSYNC = 1.
- 00 3-проводный SPI
- 01 4-проводный SPI (разрешение ведущего/ведомого при STE = 1)
- 10 4-проводный SPI (разрешение ведущего/ведомого при STE = 0)
- 11 Режим I^2C

UCSYNC Бит 0 Разрешение синхронного режима работы.
- 0 Асинхронный режим
- 1 Синхронный режим

UCBxCTL1, регистр управления 1 модуля USCI_Bx

7	6	5	4	3	2	1	0
UCSSELx		**Unused**	**UCTR**	**UCTXNACK**	**UCTXSTP**	**UCTXSTT**	**UCSWRST**
rw–0	rw–0	r0	rw–0	rw–0	rw–0	rw–0	rw–1

UCSSELx Биты 7...6 Выбор источника тактового сигнала модуля USCI. Эти биты определяют источник сигнала BRCLK.
- 00 UCLKI
- 01 ACLK
- 10 SMCLK
- 11 SMCLK

Unused Бит 5 Не используется.

UCTR Бит 4 Передатчик или приёмник.
0 Приёмник
1 Передатчик

UCTXNACK Бит 3 Передать NACK. Бит UCTXNACK автоматически сбрасывается после передачи NACK.
0 Отослать ACK (подтверждение)
1 Отослать NACK (нет подтверждения)

UCTXSTP Бит 2 Сформировать состояние СТОП в режиме ведущего. Игнорируется в режиме ведомого. В режиме «ведущий-приёмник» состоянию СТОП предшествует сигнал NACK. Бит UCTXSTP автоматически сбрасывается после формирования состояния СТОП.
0 Не формировать СТОП
1 Сформировать состояние СТОП

UCTXSTT Бит 1 Сформировать состояние СТАРТ в режиме ведущего. Игнорируется в режиме ведомого. В режиме «ведущий-приёмник» состоянию ПОВСТАРТ предшествует сигнал NACK. Бит UCTXSTT автоматически сбрасывается после формирования состояния СТАРТ и передачи адреса.
0 Не формировать СТАРТ
1 Сформировать состояние СТАРТ

UCSWRST Бит 0 Программный сброс модуля USCI.
0 Модуль USCI в рабочем состоянии
1 Модуль USCI удерживается в состоянии сброса

UCBxBR0, регистр управления 0 скоростью обмена модуля USCI_Bx

7	6	5	4	3	2	1	0
UCBRx – младший байт							
rw	rw	rw	rw	rw	rw	rw	rw

UCBxBR1, регистр управления 1 скоростью обмена модуля USCI_Bx

7	6	5	4	3	2	1	0
UCBRx – старший байт							
rw	rw	rw	rw	rw	rw	rw	rw

UCBRx Коэффициент деления предделителя контроллера скорости обмена. Содержимое этих регистров образует 16-битное (UCBxBR0 + UCBxBR1 × 256) значение коэффициента деления.

UCBxSTAT, регистр состояния модуля USCI_Bx

7	6	5	4	3	2	1	0
Unused	**UCSCLLOW**	**UCGC**	**UCBBUSY**	**UCNACKIFG**	**UCSTPIFG**	**UCSTTIFG**	**UCALIFG**
rw–0	r–0	rw–0	r–0	rw–0	rw–0	rw–0	rw–0

Unused Бит 7 Не используется.

UCSCLLOW Бит 6 На линии SCL НИЗКИЙ уровень.
0 На SCL не удерживается НИЗКИЙ уровень
1 На линии SCL удерживается НИЗКИЙ уровень

UCGC	Бит 5	Принят адрес общего вызова. Бит UCGS автоматически сбрасывается при обнаружении состояния СТАРТ. 0 Не было общего вызова 1 Был принят адрес общего вызова
UCBBUSY	Бит 4	Шина занята. 0 Шина неактивна 1 Шина занята
UCNACKIFG	Бит 3	Флаг прерывания по приёму NACK. Флаг UCNACKIFG автоматически сбрасывается при обнаружении состояния СТАРТ. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCSTPIFG	Бит 2	Флаг прерывания по состоянию СТОП. Флаг UCSTPIFG автоматически сбрасывается при обнаружении состояния СТАРТ. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCSTTIFG	Бит 1	Флаг прерывания по состоянию СТАРТ. Флаг UCSTTIFG автоматически сбрасывается при обнаружении состояния СТОП. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCALIFG	Бит 0	Флаг прерывания при потере арбитража. 0 Не было запроса прерывания 1 Есть запрос прерывания

UCBxRXBUF, регистр буфера приёма модуля USCI_Bx

7	6	5	4	3	2	1	0
UCRXBUFx							
r	r	r	r	r	r	r	r

UCRXBUFx	Биты 7...0	Буфер приёма доступен пользователю и содержит последний принятый символ, скопированный из сдвигового регистра приёма. При чтении регистра UCBxRXBUF сбрасывается флаг прерывания приёма UCBxRXIFG.

UCBxTXBUF, регистр буфера передачи модуля USCI_Bx

7	6	5	4	3	2	1	0
UCTXBUFx							
rw	rw	rw	rw	rw	rw	rw	rw

UCTXBUFx	Биты 7...0	Буфер передачи доступен пользователю и содержит значение, которое будет перегружено в сдвиговый регистр передачи и передано по шине. При записи в регистр UCBxTXBUF сбрасывается флаг UCBxTXIFG.

UCBxI2COA, регистр собственного адреса I²C модуля USCI_Bx

15	14	13	12	11	10	9	8
UCGCEN	**0**	**0**	**0**	**0**	**0**	**I2COAx**	
rw–0	r0	r0	r0	r0	r0	rw–0	rw–0

7	6	5	4	3	2	1	0
I2COAx							
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

UCGCEN Бит 15 Распознавание адреса общего вызова.
0 Не отвечать на общие вызовы
1 Отвечать на общие вызовы

I2COAx Биты 9...0 Собственный адрес I²C. В битах I2COAx содержится собственный адрес контроллера I²C модуля USCI_Bx. Адрес выравнивается по правой границе. В 7-битном режиме адресации 6-й бит – старший, биты с 9-го по 7-й игнорируются. В режиме 10-битной адресации бит 9 – старший.

UCBxI2CSA, регистр адреса I²C ведомого модуля USCI_Bx

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**I2CSAx**	
r0	r0	r0	r0	r0	r0	rw–0	rw–0

7	6	5	4	3	2	1	0
I2CSAx							
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

I2COAx Биты 9...0 Адрес ведомого I²C. В битах I2CSAx содержится адрес внешнего I²C-устройства, адресуемого модулем USCI_Bx. Этот регистр используется только в режиме ведущего. Адрес выравнивается по правой границе. В 7-битном режиме адресации 6-й бит – старший, биты с 9-го по 7-й игнорируются. В режиме 10-битной адресации бит 9 – старший.

UCBxI2CIE, регистр разрешения прерываний I²C модуля USCI_Bx

7	6	5	4	3	2	1	0
Reserved				**UCNACKIE**	**UCSTPIE**	**UCSTTIE**	**UCALIE**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

Reserved Биты 7...4 Зарезервированы.

UCNACKIE Бит 3 Разрешение прерывания по NACK.
0 Прерывание запрещено
1 Прерывание разрешено

UCSTPIE Бит 2 Разрешение прерывания по состоянию СТОП.
0 Прерывание запрещено
1 Прерывание разрешено

UCSTTIE	Бит 1	Разрешение прерывания по состоянию СТАРТ. 0 Прерывание запрещено 1 Прерывание разрешено
UCALIE	Бит 0	Разрешение прерывания при потере арбитража. 0 Прерывание запрещено 1 Прерывание разрешено

IE2, регистр разрешения прерываний 2

7	6	5	4	3	2	1	0
				UCB0TXIE	**UCB0RXIE**		
				rw–0	rw–0		

	Биты 7...4	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
UCB0TXIE	Бит 3	Разрешение прерывания передачи модуля USCI_B0. 0 Прерывание запрещено 1 Прерывание разрешено
UCB0RXIE	Бит 2	Разрешение прерывания приёма модуля USCI_B0. 0 Прерывание запрещено 1 Прерывание разрешено
	Биты 1...0	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

IFG2, регистр флагов прерываний 2

7	6	5	4	3	2	1	0
				UCB0TXIFG	**UCB0RXIFG**		
				rw–1	rw–0		

	Биты 7...4	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.
UCB0TXIFG	Бит 3	Флаг прерывания передачи модуля USCI_B0. Бит UCB0TXIFG устанавливается при опустошении регистра UCB0TXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания
UCB0RXIFG	Бит 2	Флаг прерывания приёма модуля USCI_B0. Бит UCB0RXIFG устанавливается при копировании принятого символа в регистр UCB0RXBUF. 0 Не было запроса прерывания 1 Есть запрос прерывания
	Биты 1...0	Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

UC1IE, регистр разрешения прерываний модуля USCI_B1

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**UCB1TXIE**	**UCB1RXIE**		
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0		

Unused Биты 7...4 Не используются.

UCB1TXIE Бит 3 Разрешение прерывания передачи модуля USCI_B1.
0 Прерывание запрещено
1 Прерывание разрешено

UCB1RXIE Бит 2 Разрешение прерывания приёма модуля USCI_B1.
0 Прерывание запрещено
1 Прерывание разрешено

Биты 1...0 Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

UC1IFG, регистр флагов прерываний модуля USCI_B1

7	6	5	4	3	2	1	0
Unused	**Unused**	**Unused**	**Unused**	**UCB1TXIFG**	**UCB1RXIFG**		
rw–0	rw–0	rw–0	rw–0	rw–1	rw–0		

Unused Биты 7...4 Не используются.

UCB1TXIFG Бит 3 Флаг прерывания передачи модуля USCI_B1. Бит UCB1TXIFG устанавливается при опустошении регистра UCB1TXBUF.
0 Не было запроса прерывания
1 Есть запрос прерывания

UCB1RXIFG Бит 2 Флаг прерывания приёма модуля USCI_B1. Бит UCB1RXIFG устанавливается при копировании принятого символа в регистр UCB1RXBUF.
0 Не было запроса прерывания
1 Есть запрос прерывания

Биты 1...0 Эти биты могут использоваться другими модулями. См документацию на конкретный микроконтроллер.

ГЛАВА 18

МОДУЛЬ ОПЕРАЦИОННОГО УСИЛИТЕЛЯ ОА

Модуль ОА представляет собой операционный усилитель общего назначения. В этой главе описывается работа модуля ОА. В микроконтроллерах MSP430x22x4 реализовано два модуля ОА.

18.1. Введение

Модуль операционного усилителя ОА предназначен для обработки внешнего аналогового сигнала для последующего аналого-цифрового преобразования.

Модуль ОА имеет следующие особенности:

- однополярное питание, малый потребляемый ток;
- выходной сигнал с размахом, равным напряжению питания (выход rail-to-rail);
- программируемое соотношение между временем установления и потреблением;
- программное конфигурирование;
- программно-конфигурируемый резистивный делитель в цепи обратной связи для реализации усилителей с программируемым коэффициентом усиления (PGA).

Примечание. Несколько модулей ОА

Некоторые модели могут иметь в своём составе более одного модуля ОА. В этом случае все модули имеют одинаковые характеристики.

В данной главе названия регистров даются в виде OAxCTL0, где x — порядковый номер модуля. Если номер указан в названии регистра, то описание относится к конкретному модулю. В тех случаях, когда модули идентичны, используются обобщённые названия регистров, например OAxCTL0.

Блок-схема модуля ОА приведена на **Рис. 18.1**.

18.2. Функционирование модуля ОА

Конфигурирование модуля ОА осуществляется пользовательской программой. Настройка модуля и его функционирование рассматриваются в следующих подразделах.

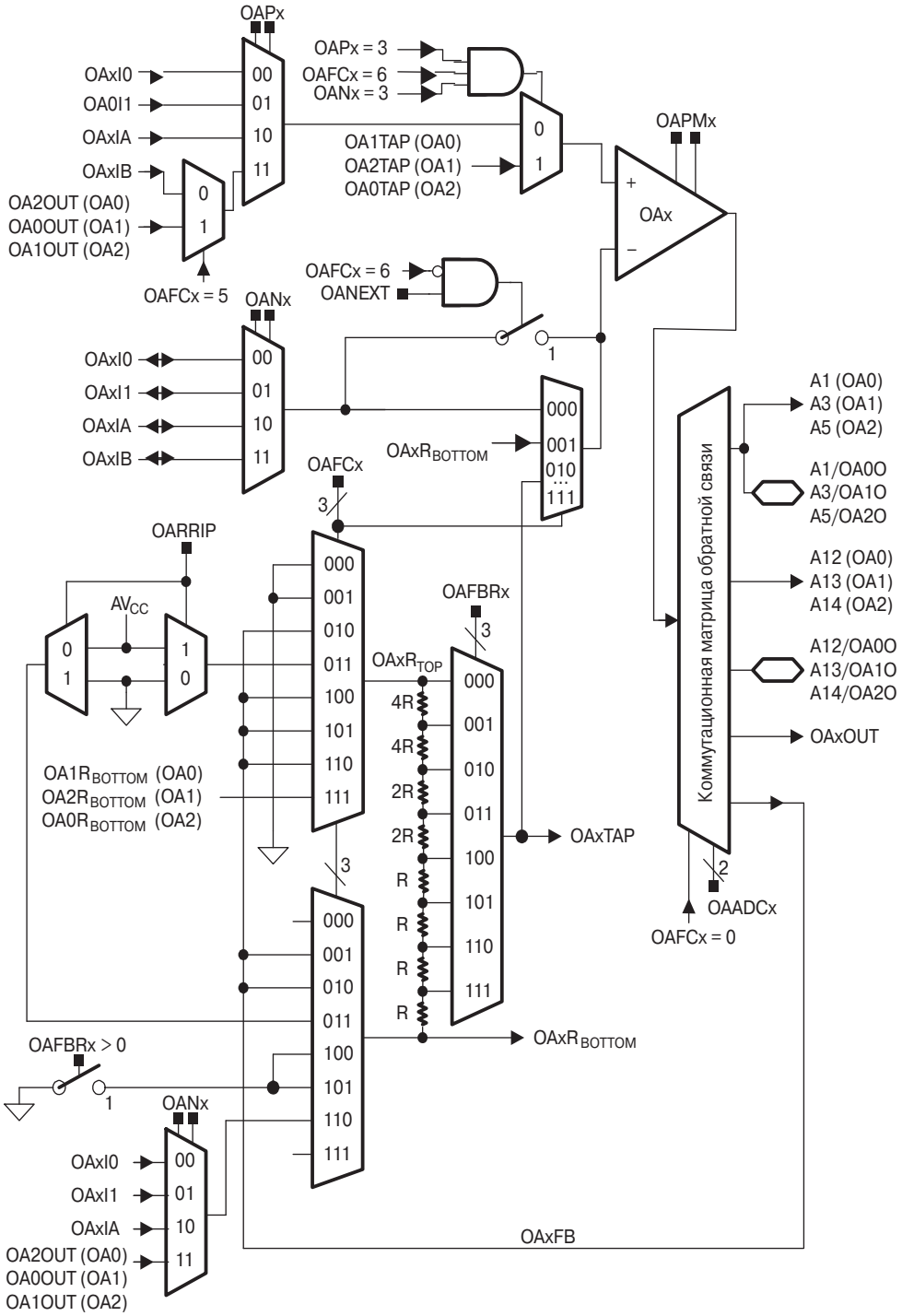


Рис. 18.1. Блок-схема модуля ОА.

18.2.1. Операционный усилитель

Модуль ОА представляет собой конфигурируемый операционный усилитель (ОУ) с малым потреблением и с выходом типа «rail-to-rail». Он может быть сконфигурирован как инвертирующий или неинвертирующий усилитель, а также соединён с другими модулями ОА для реализации различных вариантов дифференциальных усилителей. Скорость нарастания выходного напряжения модуля задаётся программно битами OAMPx, что позволяет выбирать оптимальное соотношение между временем установления и потребляемым током. При OAMPx = 0 операционный усилитель выключен и его выход находится в высокоимпедансном состоянии. При OAMPx > 0 операционный усилитель включён. Значения параметров операционного усилителя приводятся в справочной документации на конкретные модели.

18.2.2. Входы модуля ОА

Модуль имеет конфигурируемые входы. Сигналы, поступающие на инвертирующий и неинвертирующий входы операционного усилителя модуля, задаются независимо друг от друга битами OANx и OAPx соответственно. Эти сигналы могут быть как внешними, так и внутренними. Сигналы OAxI0 и OAxI1 являются внешними сигналами, индивидуальными для каждого модуля. Сигнал OA0I1 — это внешний сигнал для неинвертирующего входа ОУ, который является общим для всех модулей ОА. Назначение сигналов OAxIA и OAxIB зависит от модели микроконтроллера. Соответствие этих сигналов выводам микроконтроллера приводится в справочной документации на конкретную модель.

Если инвертирующий вход модуля не используется, то, установив бит OANEXT, можно подключить инвертирующий вход ОУ непосредственно к выводу микроконтроллера.

18.2.3. Выход модуля ОА и организация обратной связи

Модуль ОА имеет конфигурируемый выход. Конфигурация выхода определяется битами OAADCx и OAFCx. Выход модуля может быть подключён внутри кристалла к входам A12 (OA0), A13 (OA1) или A14 (OA2) модуля 12-битного АЦП ADC12, при этом возможен вариант одновременного подключения выхода ОУ к соответствующему выводу микроконтроллера. Также выходные сигналы модуля могут поступать на входы АЦП A1 (OA0), A3 (OA1) или A5 (OA2) и одновременно на соответствующие выводы микроконтроллера. Посредством битов OAFCx выход ОУ также может быть подключён к внутреннему резистивному делителю. Коэффициент деления этого делителя определяется битами OAFBRx, что позволяет реализовать усилитель с программируемым коэффициентом усиления.

Различные конфигурации выхода модуля ОА перечислены в **Табл. 18.1**. При OAFCx = 0 модуль работает как обычный ОУ, а обратная связь реализуется внешними элементами. При OAFCx > 0 и OAADCx = 00 или 11 выход модуля ОА подключается к внутренним цепям микроконтроллера. При OAFCx > 0 и OAADCx = 01 или 10 выход модуля подключается как к внутренним цепям, так и к выводам микроконтроллера.

Таблица 18.1. **Конфигурация выхода модуля ОА**

OAFCx	OAADCx	Выход ОА и реализация обратной связи
= 0	x0	OAxOUT подключается к выводам микроконтроллера и входу A1, A3 или A5 АЦП
= 0	x1	OAxOUT подключается к выводам микроконтроллера и входу A12, A13 или A14 АЦП
> 0	00	OAxOUT подключается к внутренним цепям микроконтроллера
> 0	01	OAxOUT подключается к выводам микроконтроллера и входу A12, A13 или A14 АЦП
> 0	10	OAxOUT подключается к выводам микроконтроллера и входу A1, A3 или A5 АЦП
> 0	11	OAxOUT подключается к входу A12, A13 или A14 АЦП. Вывод микроконтроллера, соответствующий этому входу, отключается от модуля АЦП

18.2.4. Конфигурация модуля ОА

Модуль ОА может использоваться для реализации различных вариантов дифференциальных усилителей. Конкретный режим работы модуля определяется битами OAFCx согласно **Табл. 18.2**.

Таблица 18.2. **Конфигурация выхода модуля ОА**

OAFCx	Режим модуля ОА
000	Операционный усилитель общего назначения
001	Буфер с единичным усилением для реализации дифференциального усилителя из трёх ОУ
010	Буфер с единичным усилением
011	Компаратор
100	Неинвертирующий PGA
101	Многокаскадный неинвертирующий PGA
110	Инвертирующий PGA
111	Дифференциальный усилитель

Операционный усилитель общего назначения

В этом режиме внутренний делитель обратной связи отключён от ОУ модуля, а конфигурация выводов операционного усилителя определяется битами регистра управления OAxCTL0. Сигналы, поступающие на входы ОУ, задаются битами OAPx и OANx. Выход ОУ подключается к одному из входов модуля ADC12, определяемому установками регистра OAxCTL0.

Буфер с единичным усилением для дифференциального усилителя

В этом режиме выход ОУ соединён с его инвертирующим входом, обеспечивая коэффициент усиления, равный единице. Сигнал, подаваемый на неинвертирующий вход, определяется битами OAPx. Подача внешних сигналов на инвертирующий вход невозможна, поэтому содержимое битов OANx игнорируется. Выход ОУ подключается к одному из выводов внутренней резистивной цепочки. Этот режим используется только при реализации дифференциального усилителя на трёх ОУ.

Буфер с единичным усилением

В этом режиме выход ОУ соединён с его инвертирующим входом, обеспечивая коэффициент усиления, равный единице. Сигнал, подаваемый на неинвертирующий вход, определяется битами OAPx. Подача внешних сигналов на инвертирующий вход невозможна, поэтому содержимое битов OANx игнорируется. Выход ОУ подключается к одному из входов модуля ADC12, определяемому установками регистра OAxCTL0.

Компаратор

В этом режиме выход ОУ отключён от резистивной цепочки. При OARRIP = 0 один конец этой цепочки (R_{TOP}) подключён к AV_{SS}, а другой (R_{BOTTOM}) — к AV_{CC}. При OARRIP = 1 цепочка включается в обратном направлении: R_{TOP} к AV_{CC}, а R_{BOTTOM} к AV_{SS}. Выход делителя OAxTAP подключается к инвертирующему входу ОУ, образуя компаратор с программируемым порогом, величина которого задаётся битами OAFBRx. Сигнал, подаваемый на неинвертирующий вход, определяется битами OAPx. Можно ввести в схему гистерезис, добавив внешний резистор положительной обратной связи. Подача внешних сигналов на инвертирующий вход невозможна, поэтому содержимое битов OANx игнорируется. Выход ОУ подключается к одному из входов модуля ADC12, определяемому установками регистра OAxCTL0.

Неинвертирующий PGA

В этом режиме выход ОУ подключён к R_{TOP}, а R_{BOTTOM} — к AV_{SS}. Выход делителя OAxTAP подключается к инвертирующему входу ОУ, образуя неинвертирующий усилитель с коэффициентом усиления $[1 + k_{OAxTAP}]$. Коэффициент k_{OAxTAP} определяется содержимым битов OAFBRx. При OAFBRx = 0 коэффициент усиления равен единице. Сигнал, подаваемый на неинвертирующий вход, определяется битами OAPx. Подача внешних сигналов на инвертирующий вход невозможна, поэтому содержимое битов OANx игнорируется. Выход ОУ подключается к одному из входов модуля ADC12, определяемому установками регистра OAxCTL0.

Многокаскадный неинвертирующий PGA

В этом режиме допускается последовательное соединение двух или трёх модулей ОА, работающих как неинвертирующие усилители. В данном режиме при OAPx = 11 неинвертирующий вход усилителя OAx подключается к выходу OA2OUT (OA0), OA0OUT (OA1) или OA1OUT (OA2). Выходы ОУ модулей подключаются к входам модуля ADC12, определяемым установками регистров OAxCTL0.

Инвертирующий PGA

В этом режиме выход ОУ подключён к R_{TOP}, а R_{BOTTOM} — к выходу аналогового мультиплексора, который управляется битами OANx и мультиплексирует сигналы OAxI0, OAxI1, OAxIA, а также сигнал с выхода одного из оставшихся моду-

лей. Выход делителя OAxTAP подключается к инвертирующему входу ОУ, образуя неинвертирующий усилитель с коэффициентом усиления $-k_{OAxTAP}$. Коэффициент k_{OAxTAP} определяется содержимым битов OAFBRx. При OAFBRx = 0 коэффициент усиления равен единице. Сигнал, подаваемый на неинвертирующий вход, определяется битами OAPx. Выход ОУ подключается к одному из входов модуля ADC12, определяемому установками регистра OAxCTL0.

Примечание. Одновременное использование вывода в качестве инвертирующего входа компаратора и входа АЦП

Если вывод, подключённый к мультиплексору инвертирующего входа, одновременно используется в качестве входа АЦП, то возможно появление ошибки преобразования величиной до 5 мВ, вызванной падением напряжения на внутренних цепях.

Дифференциальный усилитель

В этом режиме модули OA микроконтроллера соединяются друг с другом, образуя дифференциальный усилитель на двух или трёх ОУ. На **Рис. 18.2** приведена схема усилителя на двух ОУ, реализованного на модулях OA0 и OA1. В данном режиме выход ОУ первого модуля подключается к линии RTOP этого же модуля через цепи второго модуля, работающего в режиме инвертирующего PGA. Второй конец резистивной цепочки R_{BOTTOM} остаётся неподключённым, образуя буфер с единичным усилением. Этот буфер соединяется с одним или двумя оставшимися модулями, образуя дифференциальный усилитель. Выход усилителя подключается к одному из входов модуля ADC12, определяемому установками регистра OAxCTL0.

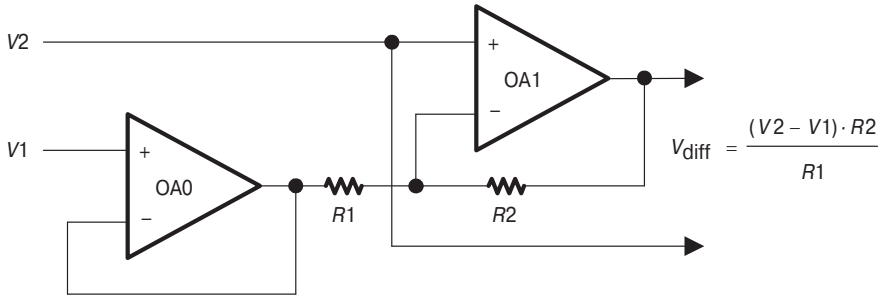


Рис. 18.2. Дифференциальный усилитель на двух ОУ.

На **Рис. 18.2** показана схема дифференциального усилителя на двух ОУ, реализованного на модулях OA0 и OA1. Значения регистров управления обоих модулей приведены в **Табл. 18.3**. Коэффициент усиления усилителя задаётся битами OAFBRx модуля OA1 согласно **Табл. 18.4**. Внутренние соединения модулей показаны на **Рис. 18.3**.

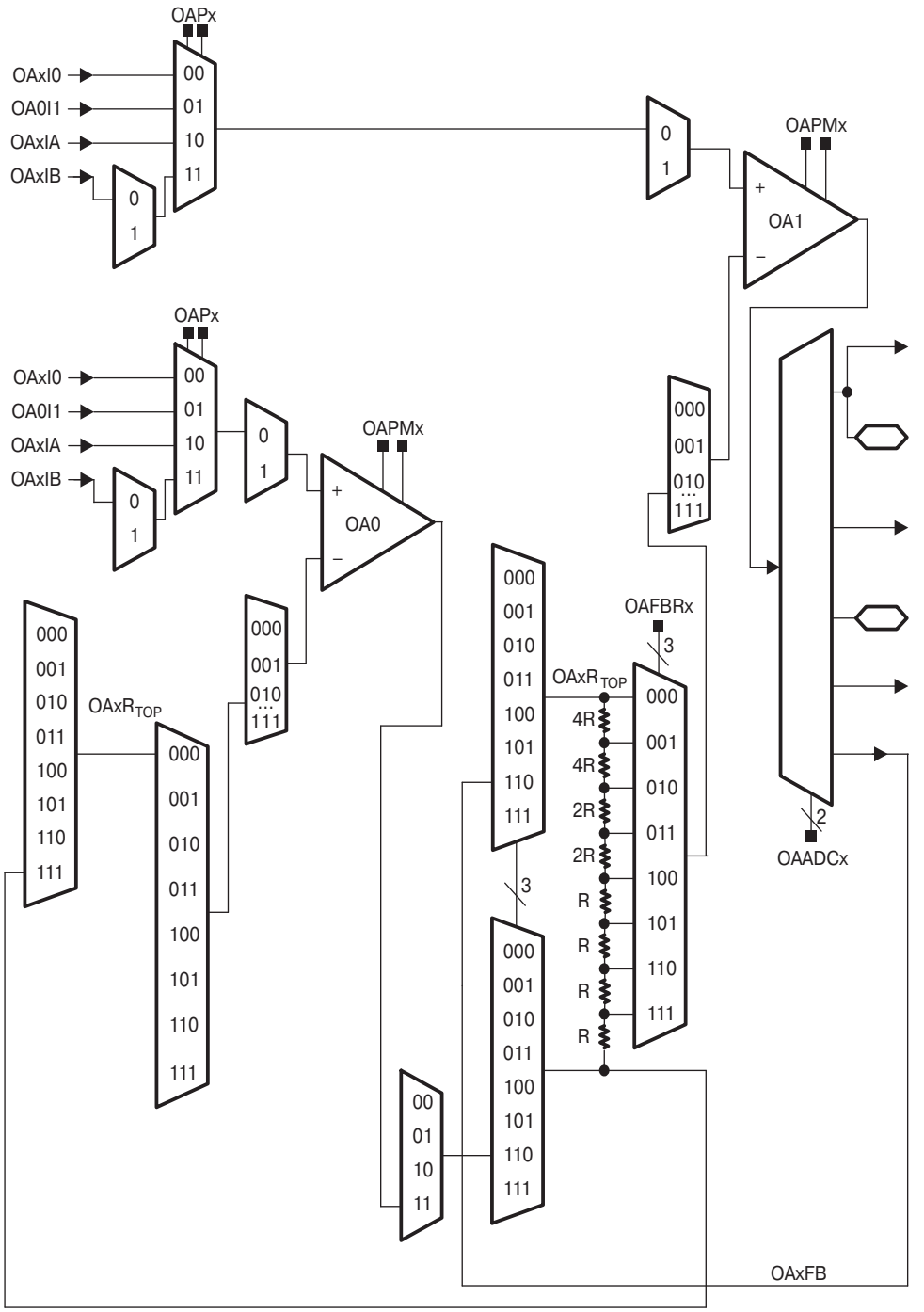


Рис. 18.3. Внутренние соединения модулей ОА при реализации дифференциального усилителя на двух ОУ.

Таблица 18.3. **Значения регистров управления для дифференциального усилителя на двух ОУ**

Регистр	Значение (двоичное)
OA0CTL0	xx xx xx 0 0
OA0CTL1	000 111 0 x
OA1CTL0	11 xx xx x x
OA1CTL1	xxx 110 0 x

Таблица 18.4. **Задание коэффициента усиления дифференциального усилителя на двух ОУ**

OAFBRx модуля OA1	Коэффициент усиления
000	0
001	1/3
010	1
011	1 2/3
100	3
101	4 1/3
110	7
111	15

На **Рис. 18.4** показана схема дифференциального усилителя на трёх ОУ, реализованного на модулях OA0, OA1 и OA2 (три модуля имеются не во всех микроконтроллерах, обратитесь к справочной документации на конкретную модель).

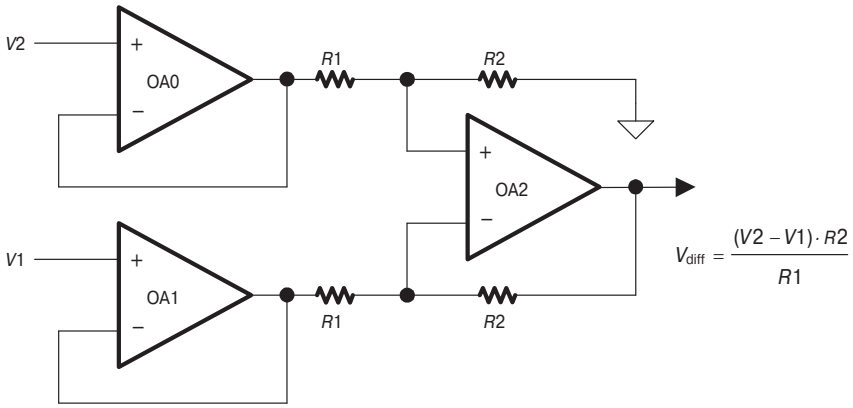


Рис. 18.4. Дифференциальный усилитель на трёх ОУ.

Значения регистров управления всех модулей приведены в **Табл. 18.5**. Коэффициент усиления усилителя задаётся битами OAFBRx модулей OA0 и OA2 согласно **Табл. 18.6**. Оба модуля должны иметь одинаковое усиление. Внутренние соединения модулей OAx показаны на **Рис. 18.5**.

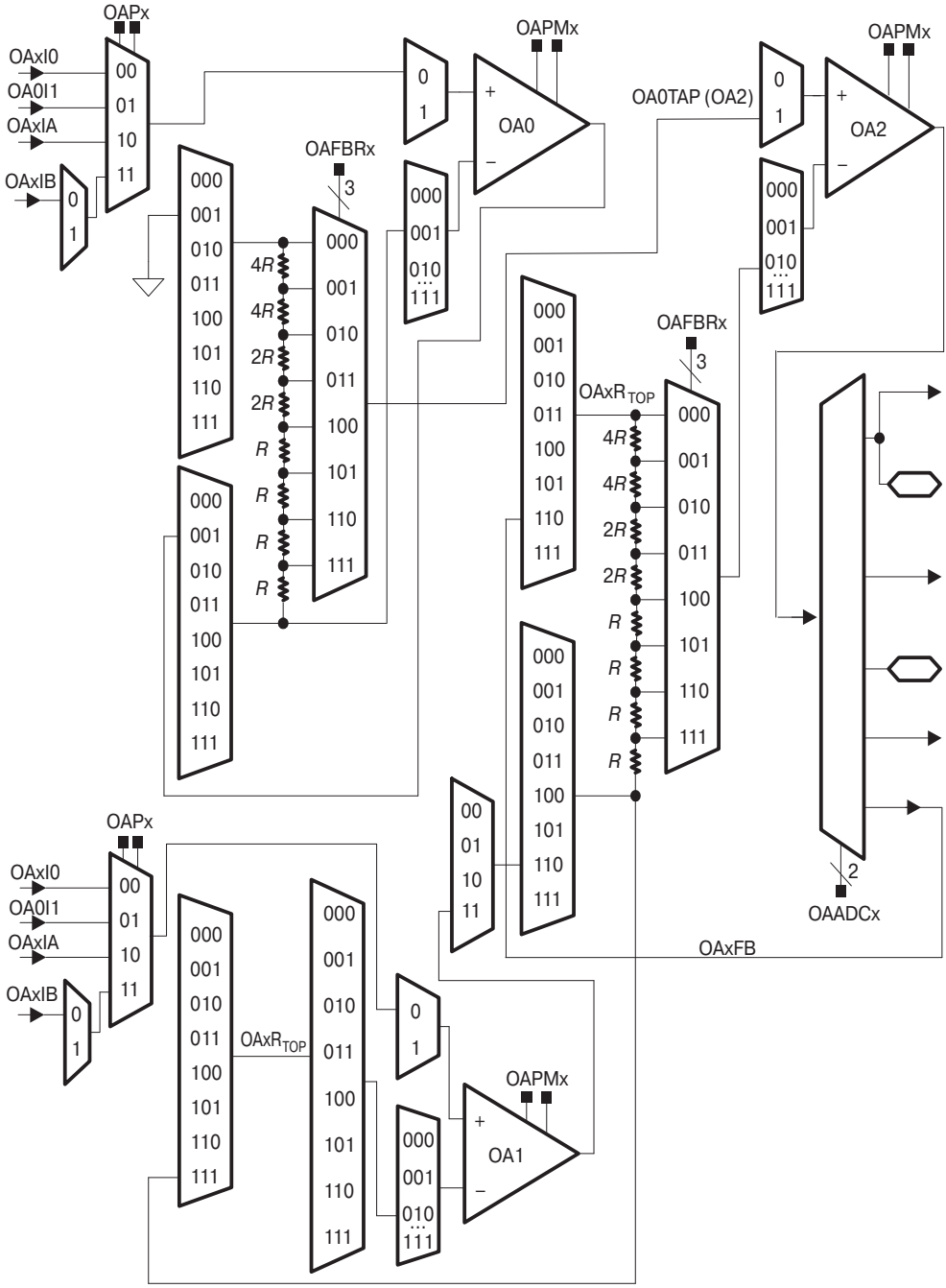


Рис. 18.5. Внутренние соединения модулей ОА при реализации дифференциального усилителя на трёх ОУ.

Таблица 18.5. **Значения регистров управления для дифференциального усилителя на трёх ОУ**

Регистр	Значение (двоичное)
OA0CTL0	xx xx xx 0 0
OA0CTL1	000 001 0 x
OA1CTL0	xx xx xx 0 0
OA1CTL1	000 111 0 x
OA2CTL0	11 11 xx x x
OA2CTL1	xxx 110 0 x

Таблица 18.6. **Задание коэффициента усиления дифференциального усилителя на трёх ОУ**

OAFBRx модулей OA1/OA2	Коэффициент усиления
000	0
001	1/3
010	1
011	1 2/3
100	3
101	4 1/3
110	7
111	15

18.3. Регистры модулей OA

Список регистров модулей OA приведён в **Табл. 18.7**.

Таблица 18.7. **Регистры модулей OA**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля OA0	OA0CTL0	Чтение/запись	0C0h	Сбрасывается после POR
Регистр управления 1 модуля OA0	OA0CTL1	Чтение/запись	0C1h	Сбрасывается после POR
Регистр управления 0 модуля OA1	OA1CTL0	Чтение/запись	0C2h	Сбрасывается после POR
Регистр управления 1 модуля OA1	OA1CTL1	Чтение/запись	0C3h	Сбрасывается после POR
Регистр управления 0 модуля OA2	OA2CTL0	Чтение/запись	0C4h	Сбрасывается после POR
Регистр управления 1 модуля OA2	OA2CTL1	Чтение/запись	0C5h	Сбрасывается после POR

OAxCTL0, регистр управления 0 модуля OAx

7	6	5	4	3	2	1	0
OANx		**OAPx**		**OAPMx**		**OAADCx**	
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

OANx Биты 7...6 Конфигурация инвертирующего входа. Эти биты определяют сигнал, поступающий на инвертирующий вход ОУ модуля.

00 OAxI0
01 OAxI1
10 OAxIA (используемый сигнал см. в документации на конкретную модель)
11 OAxIB (используемый сигнал см. в документации на конкретную модель)

OAPx Биты 5...4 Конфигурация неинвертирующего входа. Эти биты определяют сигнал, поступающий на неинвертирующий вход ОУ модуля.

00 OAxI0
01 OA0I1
10 OAxIA (используемый сигнал см. в документации на конкретную модель)
11 OAxIB (используемый сигнал см. в документации на конкретную модель)

OAPMx Биты 3...2 Выбор скорости нарастания. Эти биты определяют соотношение между скоростью нарастания и током потребления ОУ модуля.

00 Модуль выключен, выход в высокоимпедансном состоянии
01 Малая скорость нарастания
10 Средняя скорость нарастания
11 Большая скорость нарастания

OAADCx Биты 1...0 Конфигурация выхода. Эти биты совместно с битами OAFCx определяют подключение выхода модуля при OAMP > 0.

При OAMP = 0:

00 OAxOUT подключён к внешнему выводу и входу A1, A3 или A5 АЦП
01 OAxOUT подключён к внешнему выводу и входу A12, A13 или A14 АЦП
10 OAxOUT подключён к внешнему выводу и входу A1, A3 или A5 АЦП
11 OAxOUT подключён к внешнему выводу и входу A12, A13 или A14 АЦП

При OAMP > 0:

00 OAxOUT используется только для внутренних соединений
01 OAxOUT подключён к внешнему выводу и входу A12, A13 или A14 АЦП
10 OAxOUT подключён к внешнему выводу и входу A1, A3 или A5 АЦП
11 OAxOUT подключён к входу A12, A13 или A14 АЦП. Вывод микроконтроллера, соответствующий этому входу, отключён от АЦП.

UAxCTL1, регистр управления 0 модуля OAx

7	6	5	4	3	2	1	0
OAFBRx			**OAFCx**			**OANEXT**	**OARRIP**
rw–0	rw–0	r–0	rw–0	rw–0	rw–0	rw–0	rw–0

OAFBRx Биты 7...5 Конфигурация резистивной цепочки обратной связи.
- 000 Отвод 0 — $0R/16R$
- 001 Отвод 1 — $4R/12R$
- 010 Отвод 2 — $8R/8R$
- 011 Отвод 3 — $10R/6R$
- 100 Отвод 4 — $12R/4R$
- 101 Отвод 5 — $13R/36R$
- 110 Отвод 6 — $14R/26R$
- 111 Отвод 7 — $15R/16R$

OAFCx Биты 4...2 Выбор режима работы модуля.
- 000 ОУ общего назначения
- 001 Буфер с единичным усилением для дифференциального усилителя на трёх ОУ
- 010 Буфер с единичным усилением
- 011 Компаратор
- 100 Неинвертирующий PGA
- 101 Многокаскадный неинвертирующий PGA
- 110 Инвертирующий PGA
- 111 Дифференциальный усилитель

OANEXT Бит 1 Доступность инвертирующего входа ОУ. При установленном бите OANEXT инвертирующий вход ОУ подключается к выводу микроконтроллера, если используется внутренняя резистивная цепочка.
- 0 Инвертирующий вход ОУ недоступен для внешних устройств
- 1 Инвертирующий вход ОУ доступен для внешних устройств

OARRIP Бит 0 Обратное подключение резистивной цепочки в режиме компаратора.
- 0 $R_{ТОР}$ подключён к AV_{SS}, а $R_{ВОТТОМ}$ — к AV_{CC} при OAFCx = 3
- 1 $R_{ТОР}$ подключён к AV_{CC}, а $R_{ВОТТОМ}$ — к AV_{SS} при OAFCx = 3

ГЛАВА 19

МОДУЛЬ АНАЛОГОВОГО КОМПАРАТОРА COMPARATOR_A+

Модуль Comparator_A+ представляет собой аналоговый компаратор напряжения. В этой главе описывается работа модуля Comparator_A+, реализованного в микроконтроллерах семейства MSP430x2xx.

19.1. Введение

Модуль компаратора Comparator_A+ обеспечивает выполнение точных аналого-цифровых преобразований методом прямого интегрирования, контроль напряжения питания и мониторинг внешних аналоговых сигналов.

Модуль Comparator_A+ имеет следующие особенности:

- мультиплексоры на обоих (инвертирующем и неинвертирующем) входах модуля;
- программно подключаемый *RC*-фильтр на выходе компаратора;
- возможность подключения выхода компаратора к входу захвата Таймера A;
- программное управление входным буфером порта;
- поддержка прерываний;
- конфигурируемый генератор опорного напряжения;
- возможность выключения компаратора и генератора опорного напряжения;
- входной мультиплексор компаратора.

Блок-схема модуля компаратора приведена на **Рис. 19.1**.

19.2. Функционирование модуля Comparator_A+

Конфигурирование модуля Comparator_A+ осуществляется пользовательской программой. Настройка модуля и его функционирование рассматриваются в следующих подразделах.

19.2.1. Компаратор

Компаратор сравнивает аналоговые напряжения на неинвертирующем (+) и инвертирующем (–) входах. Если напряжение на неинвертирующем входе больше, чем на инвертирующем, то на выходе компаратора CAOUT появляется сигнал ВЫСОКОГО уровня. Компаратор может быть включен или выключен посредством уп-

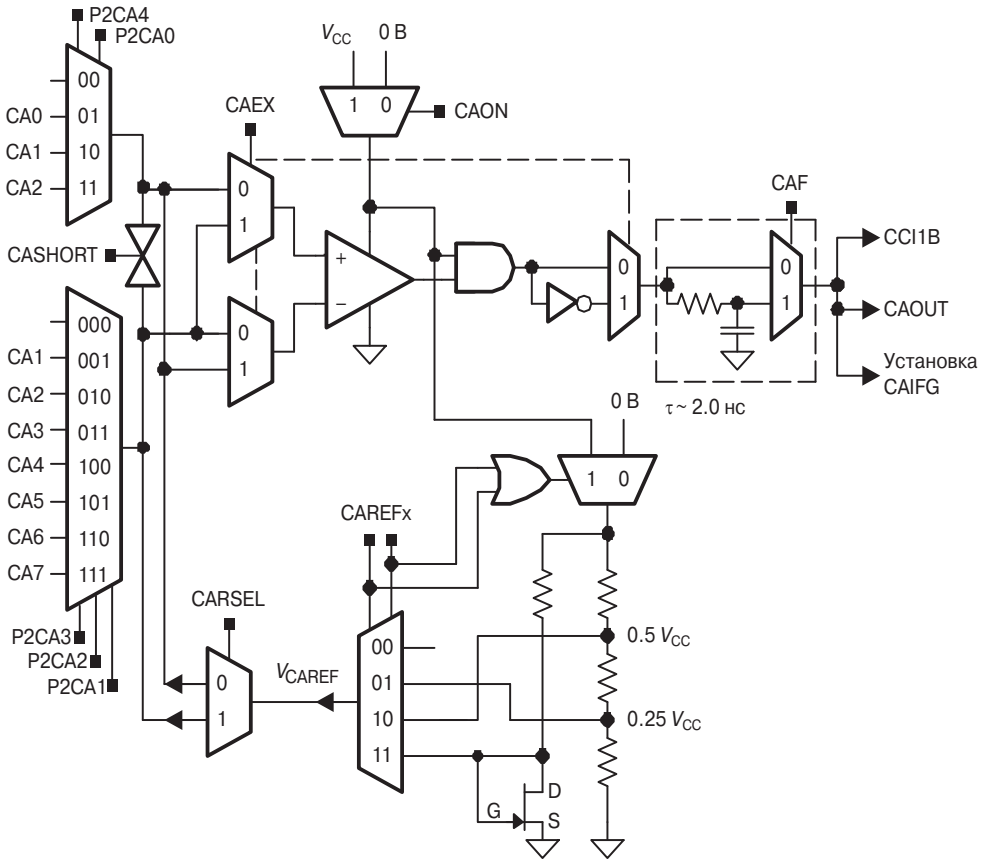


Рис. 19.1. Блок-схема модуля Comparator_A+.

равляющего бита CAON. Если компаратор не используется, то для уменьшения общего потребляемого тока его необходимо отключать. При выключенном компараторе на выходе CAOUT постоянно присутствует НИЗКИЙ уровень.

19.2.2. Входные аналоговые ключи

Входные аналоговые ключи используются для подключения или отключения входов модуля к/от соответствующих выводов микроконтроллера. Конфигурация входов компаратора определяется битами P2CAx. Оба входа компаратора конфигурируются независимо друг от друга. Биты P2CAx позволяют:

- подавать внешние сигналы на неинвертирующий и инвертирующий входы компаратора;
- выводить напряжение от внутреннего генератора опорного напряжения на соответствующий вывод микроконтроллера.

Все входные ключи выполнены по Т-образной схеме, чтобы уменьшить искажения при прохождении сигнала.

Примечание. Подключение входов компаратора
При включённом компараторе его входы должны быть подключены к источникам сигналов, питанию или земле. В противном случае плавающие потенциалы на входах могут стать причиной неожиданных прерываний и вызвать увеличение потребляемого тока.

Бит CAEX управляет входным мультиплексором собственно компаратора и определяет, какой из входных сигналов модуля будет подан на инвертирующий вход компаратора, а какой — на неинвертирующий. Если подключение входов компаратора меняется на противоположное, то выходной сигнал компаратора инвертируется. Это позволяет пользователю учесть или скомпенсировать входное напряжение смещения компаратора.

19.2.3. Ключ замыкания входов

Бит CASHORT позволяет замыкать между собой входы компаратора. Эта возможность может использоваться для реализации простого устройства выборки и хранения для компаратора, как показано на **Рис. 19.2**.

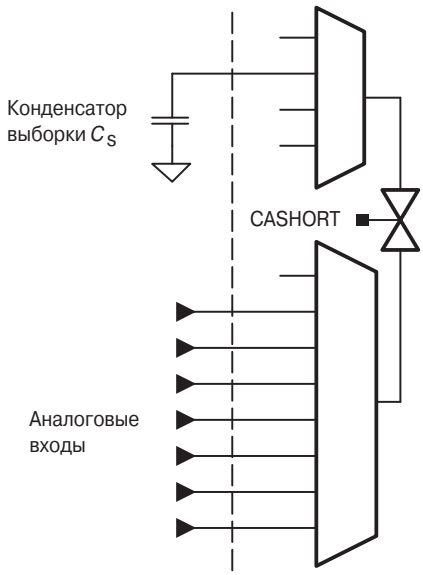


Рис. 19.2. Устройство выборки и хранения для компаратора Comparator_A+ .

Требуемое время выборки прямо пропорционально ёмкости конденсатора C_S, сопротивлению входных ключей, включённых последовательно с ключом замыкания входов (R_I) и сопротивлению источника сигнала R_S. Типовое значение сопротивления R_I лежит в диапазоне от 2 до 10 кОм. Ёмкость конденсатора C_S должна быть более 100 пФ. Постоянная времени τ заряда конденсатора может быть определена из выражения:

$$\tau = (R_i + R_S)\cdot C_S.$$

В зависимости от требуемой точности преобразования время выборки должно составлять от 3τ до 10τ. При времени выборки, равном 3τ, конденсатор заряжается до уровня, составляющего примерно 95% от напряжения входного сигнала, при 5τ — более чем 99%, а при 10τ напряжения на конденсаторе достаточно для достижения 12-битной точности.

19.2.4. Выходной фильтр

Если необходимо, то к выходу компаратора может быть подключён внутренний фильтр. При установленном бите CAF выходной сигнал компаратора проходит через встроенный *RC*-фильтр.

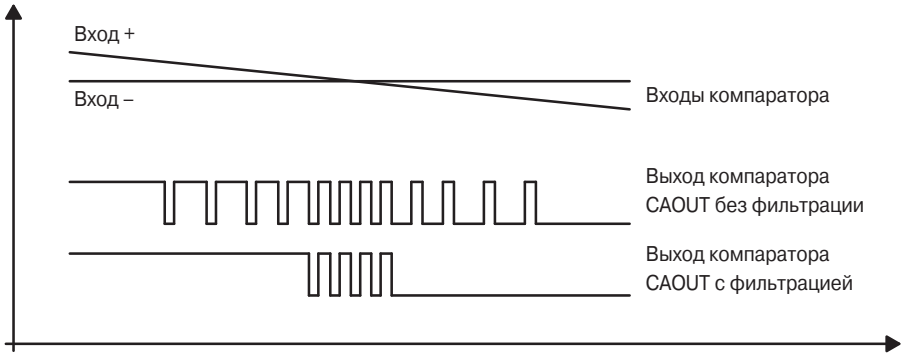


Рис. 19.3. Влияние *RC*-фильтра на выходной сигнал компаратора.

Если разница напряжений на входах компаратора достаточно мала, то на его выходе всегда начинают генерироваться колебания. Эта генерация (**Рис. 19.3**) вызывается внутренними и внешними паразитными связями, а также перекрёстными помехами между сигнальными линиями, линиями питания и другими элементами системы. Колебания на выходе компаратора снижают точность и разрешающую способность результата сравнения. Использование фильтра может уменьшить ошибки, вызванные генерацией на выходе компаратора.

19.2.5. Генератор опорного напряжения

Генератор опорного напряжения используется для формирования сигнала V_{CAREF}, которое может быть подано на любой из входов компаратора. Выходное напряжение генератора задаётся битами CAREFx. Бит CARSEL определяет вывод компаратора, на который поступает опорное напряжение от генератора. Если на оба входа компаратора подаются внешние сигналы, то внутренний генератор опорного напряжения следует выключать для уменьшения тока, потребляемого модулем. Генератор может формировать напряжение, равное четверти или половине напряжения питания V_{CC}, или же фиксированное напряжение, равное пороговому напряжению транзистора (~0.55 В).

19.2.6. Компаратор и регистр отключения порта CAPD

Входы и выход модуля компаратора мультиплексированы с определёнными линиями порта ввода/вывода, которые представляют собой цифровые схемы на КМОП-элементах. Аналоговый сигнал, поступающий на вход КМОП-элемента, может вызвать сквозной ток, протекающий от линии питания V_{CC} к земле (GND). Этот паразитный ток появляется в том случае, если входное напряжение близко к пороговому напряжению элемента. Отключение буфера вывода порта исключает появление паразитного тока и, таким образом, уменьшает общее потребление микроконтроллера.

При установленных битах CAPDx отключаются входные и выходные буферы соответствующего вывода порта P2, как показано на **Рис. 19.4**. Если потребление устройства является критичным, то необходимо отключать любые выводы порта, на которые поступают аналоговые сигналы.

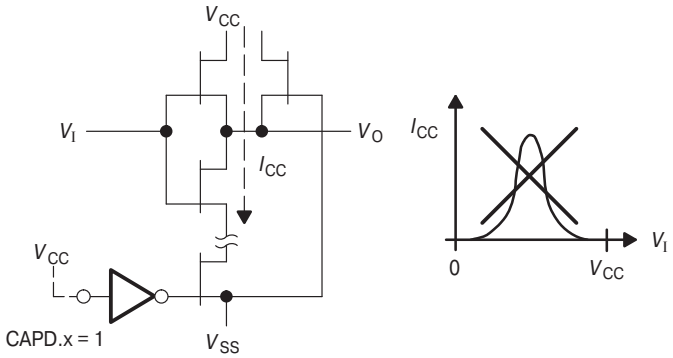


Рис. 19.4. Передаточная характеристика и рассеивание мощности в инверторе/буфере КМОП.

При подключении мультиплексора компаратора посредством битов P2CAx к какому-либо выводу микроконтроллера входной и выходной буферы этого вывода автоматически выключаются, независимо от состояния соответствующего бита CAPDx.

19.2.7. Прерывания компаратора

Модулю Comparator_A+ выделен один вектор прерывания, с которым связан единственный флаг (**Рис. 19.5**). Флаг прерывания CAIFG устанавливается по нарастающему или спадающему фронту сигнала компаратора (определяется битом CAIES). Если установлены оба бита CAIE и GIE, то установка флага CAIFG генерирует запрос прерывания. Флаг CAIFG автоматически сбрасывается при обработке прерывания или может быть сброшен программно.

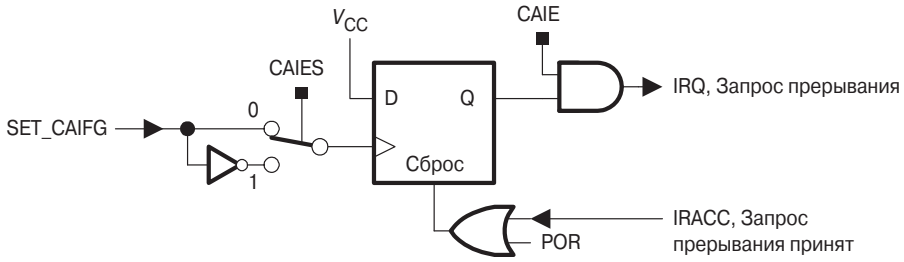


Рис. 19.5. Система прерывания модуля Comparator_A+.

19.2.8. Использование компаратора для измерения сопротивления

Модуль компаратора может применяться для точного измерения сопротивления резистивных элементов с использованием метода аналого-цифрового преобразования однократного интегрирования. Например, для преобразования значения температуры в цифровой код можно использовать термистор, сравнивая время разряда конденсатора через этот термистор и через эталонный резистор, как показано на **Рис. 19.6**. В данном случае сопротивление резистора R_{ref} сравнивается с сопротивлением термистора R_{meas}.

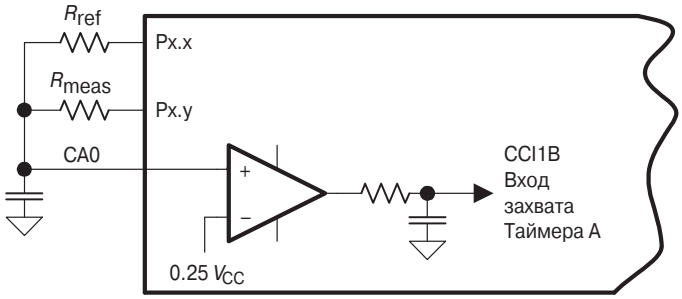


Рис. 19.6. Система измерения температуры.

Для вычисления значения температуры, измеренной при помощи R_{meas}, используются следующие ресурсы микроконтроллера:

- Две цифровых линии ввода/вывода для заряда и разряда конденсатора.
- Для заряда конденсатора на вывод выставляется ВЫСОКИЙ уровень (V_{CC}), а для разряда — НИЗКИЙ.
- Когда линия ввода/вывода не используется, она переключается в высокоимпедансное состояние установкой соответствующего бита CAPDx.
- Один выход используется для заряда/разряда конденсатора через резистор R_{ref}.
- Один выход используется для разряда конденсатора через термистор R_{meas}.
- Неинвертирующий вход компаратора подключается к положительной обкладке конденсатора.

- На инвертирующий вход компаратора подаётся опорное напряжение, например $0.25V_{CC}$.
- Для уменьшения коммутационных помех необходимо использовать выходной *RC*-фильтр.
- Выход CAOUT используется для управления входом CCI1B Таймера A, с помощью которого измеряется время разряда конденсатора.

Используя данный способ, можно измерять сопротивление нескольких резистивных элементов. Дополнительные элементы подключаются к выводу CA0 с помощью свободных линий ввода/вывода, которые переводятся в высокоимпедансное состояние в те моменты, когда они не участвуют в измерении.

Измерение сопротивления термистора основано на принципе логометрического преобразования. Отношение времён разряда конденсатора через каждый из резистивных элементов определяется согласно **Рис. 19.7**.

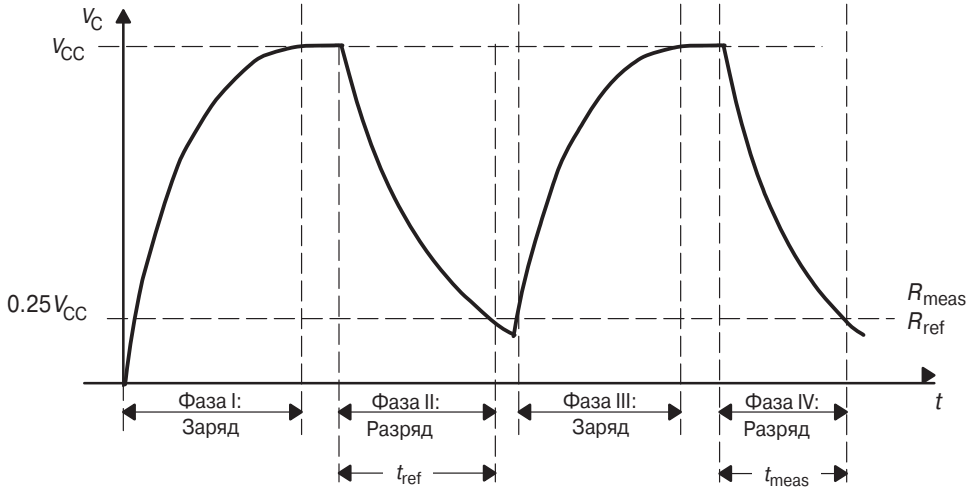


Рис. 19.7. Временная диаграмма системы измерения температуры.

Напряжение питания V_{CC} и ёмкость конденсатора должны оставаться постоянными во время измерения, однако их конкретные значения не критичны, поскольку они сокращаются:

$$\frac{N_{meas}}{N_{ref}} = \frac{-R_{meas} \times C \times \ln\frac{V_{ref}}{V_{CC}}}{-R_{ref} \times C \times \ln\frac{V_{ref}}{V_{CC}}};$$

$$\frac{N_{meas}}{N_{ref}} = \frac{R_{meas}}{R_{ref}};$$

$$R_{meas} = R_{ref} \times \frac{N_{meas}}{N_{ref}}.$$

19.3. Регистры модуля Comparator_A+

Список регистров модуля Comparator_A+ приведён в **Табл. 19.1**.

Таблица 19.1. **Регистры модуля Comparator_A+**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 1 модуля Comparator_A+	CACTL1	Чтение/запись	059h	Сбрасывается после POR
Регистр управления 2 модуля Comparator_A+	CACTL2	Чтение/запись	05Ah	Сбрасывается после POR
Отключение порта модуля Comparator_A+	CAPD	Чтение/запись	05Bh	Сбрасывается после POR

CACTL1, регистр управления 1 модуля Comparator_A+

7	6	5	4	3	2	1	0
CAEX	**CARSEL**	**CAREFx**		**CAON**	**CAIES**	**CAIE**	**CAIFG**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

CAEX Бит 7 Перестановка входов компаратора. При установленном бите CAEX входы компаратора меняются местами, а выходной сигнал компаратора инвертируется.

CARSEL Бит 6 Выбор входа для опорного напряжения. Этот бит определяет, на какой из входов компаратора будет подано опорное напряжение V_{CAREF}.

При CAEX = 0:
0 V_{CAREF} подаётся на неинвертирующий (+) вход
1 V_{CAREF} подаётся на инвертирующий (–) вход

При CAEX = 0:
0 V_{CAREF} подаётся на инвертирующий (–) вход
1 V_{CAREF} подаётся на неинвертирующий (+) вход

CAREFx Биты 5...4 Конфигурация генератора опорного напряжения. Эти биты определяют величину опорного напряжения V_{CAREF}.
00 Внутренний генератор опорного напряжения выключен. Можно использовать внешний источник опорного напряжения.
01 $0.25 \cdot V_{CC}$
10 $0.50 \cdot V_{CC}$
11 Равно падению напряжения на диоде

CAON Бит 3 Включение компаратора. Этот бит включает компаратор. Когда компаратор выключен, его потребляемый ток равен нулю. Генератор опорного напряжения включается и выключается независимо от компаратора.
0 Компаратор выключен
1 Компаратор включен

CAIES Бит 2 Выбор фронта для генерации прерывания.
0 Нарастающий фронт
1 Спадающий фронт

CAIE Бит 1 Разрешение прерывания компаратора.
0 Прерывание запрещено
1 Прерывание разрешено

CAIFG Бит 0 Флаг прерывания компаратора.
0 Не было запроса прерывания
1 Есть запрос прерывания

CACTL2, регистр управления 2 модуля Comparator_A+

7	6	5	4	3	2	1	0
CASHORT	**P2CA4**	**P2CA3**	**P2CA2**	**P2CA1**	**P2CA0**	**CAF**	**CAOUT**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–(0)

CASHORT Бит 7 Замыкание входов. Этот бит используется для замыкания между собой входов модуля компаратора.
0 Входы не замкнуты
1 Входы замкнуты

P2CA4 Бит 6 Выбор входа. Этот бит совместно с битом P2CA0 определяет подключение неинвертирующего (CAEX = 0) или инвертирующего (CAEX = 1) входов компаратора.

P2CA3 **P2CA2** **P2CA1** Биты 5...3 Выбор входа. Эти биты определяют подключение инвертирующего (CAEX = 0) или неинвертирующего (CAEX = 1) входов компартора.
000 Не подключён
001 CA1
010 CA2
011 CA3
100 CA4
101 CA5
110 CA6
111 CA7

P2CA0 Бит 2 Выбор входа. Этот бит совместно с битом P2CA5 определяет подключение неинвертирующего (CAEX = 0) или инвертирующего (CAEX = 1) входов компаратора.
00 Не подключён
01 CA0
10 CA1
11 CA2

CAF Бит 1 Выходной фильтр компаратора.
0 Выходной сигнал компаратора не фильтруется
1 Выходной сигнал компаратора фильтруется

CAOUT Бит 0 Выход компаратора. Данный бит отображает значение на выходе компаратора. Запись в этот бит не имеет смысла и игнорируется.

CAPD, регистр отключения порта модуля Comparator_A+

7	6	5	4	3	2	1	0
CAPD7	**CAPD6**	**CAPD5**	**CAPD4**	**CAPD3**	**CAPD2**	**CAPD1**	**CAPD0**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

CAPDx Биты 7...0 Отключение порта компаратора. Эти биты используются для индивидуального отключения входных буферов для выводов порта микроконтроллера, связанного с модулем Comparator_A+. Например, если вход CA0 мультиплексирован с выводом P2.3, то биты CAPDx могут использоваться для включения или отключения буферов каждого из выводов P2.x порта. Бит CAP0 отключает P2.0, CAPD1 отключает P2.1 и т.д.
0 Входной буфер включён
1 Входной буфер отключён

ГЛАВА 20

МОДУЛЬ 10-БИТНОГО АЦП ADC10

Модуль ADC10 представляет собой быстродействующий 10-битный аналого-цифровой преобразователь. В этой главе описывается работа модуля ADC10, реализованного в микроконтроллерах семейства MSP430x2xx.

20.1. Введение

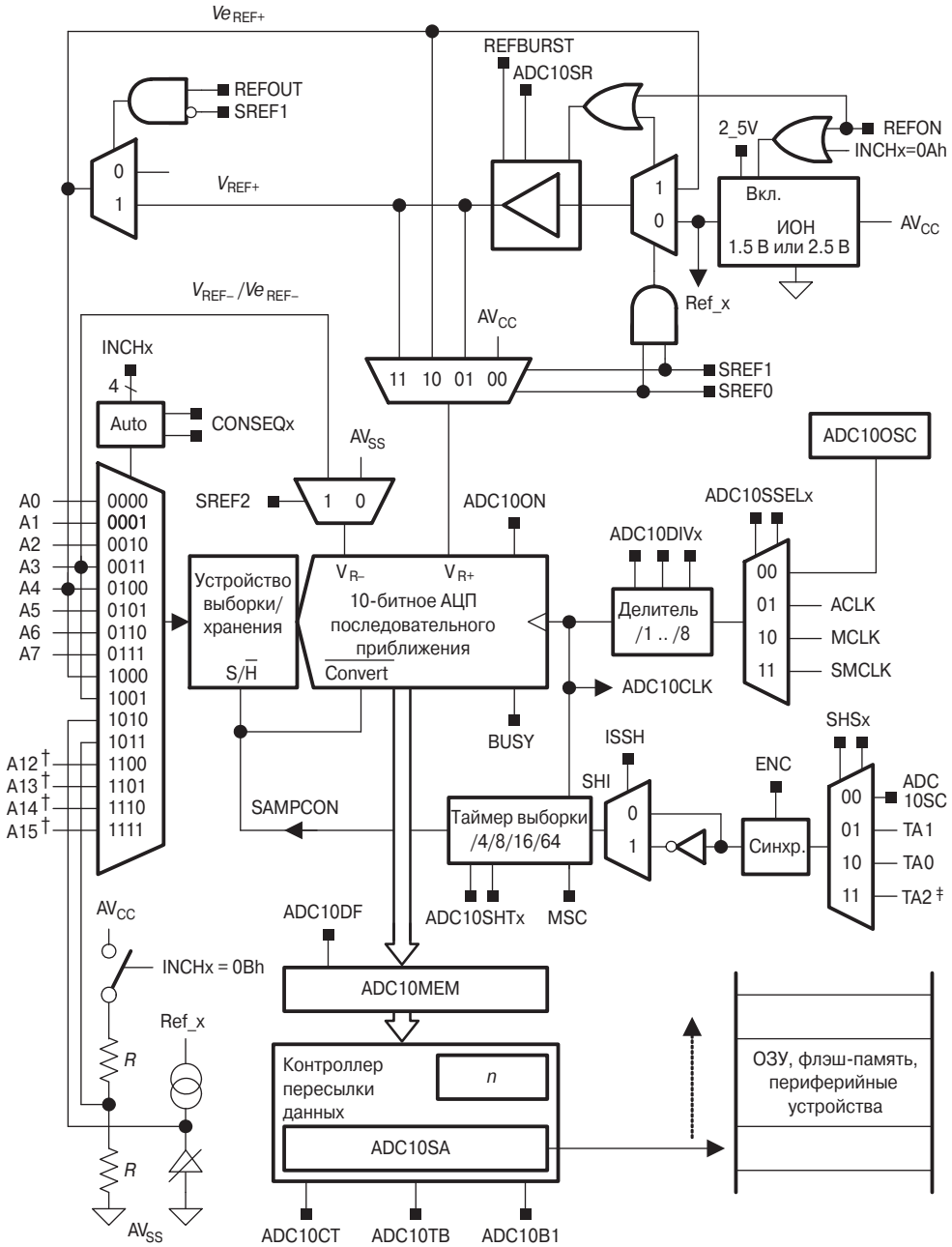
Модуль ADC10 обеспечивает быстрое выполнение 10-битных аналого-цифровых преобразований. В составе модуля имеется 10-битное ядро с регистром последовательного приближения, блок управления выборкой, генератор опорного напряжения и контроллер пересылки данных (DTC).

Контроллер DTC позволяет модулю сохранять результаты преобразований в пределах всего адресного пространства без использования ЦПУ. Модуль может конфигурироваться пользовательской программой в соответствии с требованиями различных приложений.

Модуль ADC10 имеет следующие особенности:

- максимальная скорость преобразования — более 200 тыс. выборок/с;
- 10-битный преобразователь с монотонной характеристикой без пропуска кодов;
- устройство выборки/хранения с программируемым временем выборки;
- запуск преобразования производится программно или по сигналу от Таймера B;
- программно конфигурируемый внутренний генератор опорного напряжения (1.5 В или 2.5 В);
- внешний или внутренний источник опорного напряжения (выбирается программно) ;
- восемь каналов преобразования внешних сигналов (12 в моделях MSP430x22x);
- каналы преобразования для внутреннего датчика температуры, напряжения V_{CC} и внешних опорных напряжений;
- конфигурируемый источник тактового сигнала;
- четыре режима преобразования: одноканальный, многократный одноканальный, последовательный и многократный последовательный;
- ядро АЦП и ИОН могут выключаться независимо друг от друга;
- контроллер пересылки данных для автоматического сохранения результатов преобразований.

Блок-схема модуля ADC10 приведена на **Рис. 20.1**.



† Только в моделях MSP430x22xx. В остальных моделях каналы A12-A15 соединены с каналом A11.
‡ TA1 в моделях MSP430x20x2.

Рис. 20.1. Блок-схема модуля ADC10.

20.2. Функционирование модуля ADC10

Конфигурирование модуля ADC10 осуществляется пользовательской программой. Настройка модуля и его функционирование рассматриваются в следующих подразделах.

20.2.1. Ядро 10-битного АЦП

Ядро АЦП преобразует аналоговый сигнал в 10-битный цифровой код и сохраняет результат в регистре ADC10MEM. Верхний и нижний пределы преобразования определяются двумя программируемыми уровнями напряжения (V_{R+} и V_{R-}). Результат преобразования равен максимальному значению (03FFh), если уровень входного сигнала больше или равен V_{R+}, и нулю, если уровень входного сигнала меньше или равен V_{R-}. Входной канал и уровни опорных напряжений (V_{R+} и V_{R-}) задаются регистрами управления модуля. Результат преобразования может быть представлен в обычном виде или в дополнительном коде. При использовании обычного двоичного кода результат аналого-цифрового преобразования определяется выражением:

$$N_{ADC} = 1023 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}}.$$

Конфигурирование ядра АЦП осуществляется двумя регистрами управления ADC10CTL0 и ADC10CTL1. Включается ядро АЦП установкой бита ADC10ON. За некоторыми исключениями биты управления модуля ADC10 можно изменять только при ENC = 0. Перед выполнением любого преобразования бит ENC должен быть установлен в 1.

Выбор тактового сигнала преобразования

Сигнал ADC10CLK используется как в качестве тактового сигнала преобразования, так и для формирования периодов выборки. Источник сигнала ADC10CLK задаётся битами ADC10SSELx. Выбранный сигнал поступает на АЦП через предделитель, коэффициент деления которого (1, 2, 4 или 8) определяется битами ADC10DIVx. Для формирования сигнала ADC10CLK могут использоваться системные тактовые сигналы SMCLK, MCLK, ACLK, а также сигнал внутреннего генератора ADC10OSC.

Частота сигнала ADC10OSC, формируемого внутри модуля, не превышает 5 МГц и зависит от конкретного устройства, напряжения питания и температуры. Точные параметры сигнала ADC10CLK приводятся в документации на конкретные модели.

Пользователь должен гарантировать, что тактовый сигнал, используемый для формирования ADC10CLK, останется активным до завершения процесса преобразования. Если тактовый сигнал будет остановлен во время преобразования, то операция завершена не будет, а полученный результат будет неверным.

20.2.2. Входы модуля ADC10 и мультиплексор

Входной аналоговый мультиплексор позволяет выбирать для преобразования один из 8 внешних и 4 внутренних аналоговых сигналов. Мультиплексор (**Рис. 20.2**) выполнен по схеме break-before-make (разрыв перед включением) для уменьшения перекрёстных помех, возникающих при переключении каналов. Кроме того, ключи мультиплексора выполнены по Т-образной схеме, позволяющей свести к минимуму взаимное влияние каналов. Неиспользуемые в данный момент времени каналы отключаются от АЦП, а средняя точка подключается к аналоговой земле V_{SS}. В результате паразитная ёмкость заземляется, что помогает устранить перекрёстные помехи.

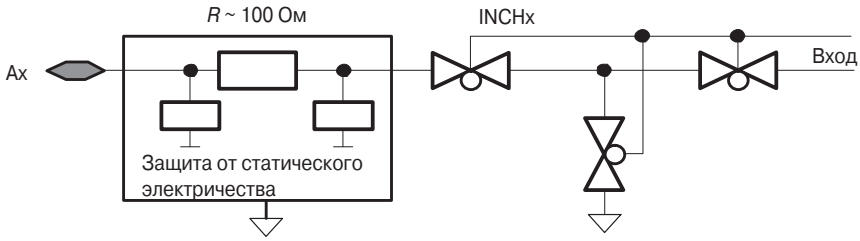


Рис. 20.2. Аналоговый мультиплексор.

Модуль ADC10 использует метод перераспределения заряда. Переключение входов внутри модуля может вызвать переходные процессы во входном сигнале. Эти переходные процессы затухают достаточно быстро, чтобы не стать причиной некорректного преобразования.

Выбор аналогового порта

Внешние входы модуля ADC10 Ax, Ve_{REF+} и V_{REF-} мультиплексированы с линиями портов ввода/вывода, которые представляют собой цифровые схемы на КМОП-элементах (см. справочную документацию на конкретную модель). Аналоговый сигнал, поступающий на вход КМОП-элемента, может вызвать сквозной ток, протекающий от линии питания V_{CC} к земле (GND). Этот паразитный ток появляется в том случае, если входное напряжение близко к пороговому напряжению элемента. Отключение буфера вывода порта исключает появление паразитного тока и, таким образом, уменьшает общее потребление микроконтроллера. Биты ADC10AEx позволяют отключать входные и выходные буферы соответствующих выводов порта.

```
; P2.3 в микроконтроллере MSP430x22xx используется как аналоговый вход
  BIS.B #08h,&ADC10AE0  ; P2.3 - аналоговый вход для ADC10
```

20.2.3. Генератор опорного напряжения

Модуль ADC10 содержит встроенный генератор опорного напряжения, способный формировать напряжение двух номиналов. Генератор опорного напряжения включается установкой бита REFON = 1. При REF2_5V = 1 внутреннее опорное напряжение равно 2.5 В, а при REF2_5V = 0 внутреннее опорное напря-

жение составляет 1.5 В. Это напряжение может использоваться внутри кристалла, а также может быть выведено на вывод V_{REF+} при REFOUT = 1.

Уровни V_+ и V_- могут задаваться внешними источниками, подключаемыми к выводам A4 и A3 соответственно. При использовании внешних источников опорного напряжения или же при использовании в качестве опорного напряжения питания V_{CC}, внутренний генератор опорного напряжения можно отключить для уменьшения потребления устройства.

Внешнее положительное опорное напряжение Ve_{REF+} может буферироваться — для этого следует установить в 1 биты SREF0 и SREF1. Наличие буфера даёт возможность использовать источник с большим внутренним сопротивлением, но ценой увеличения тока потребления микроконтроллера. Если бит REFBURST = 1, то потребляемый ток увеличивается только во время выборки и преобразования.

Модуль ADC10, в отличие от модуля ADC12, не требует подключения внешних фильтрующих конденсаторов к входам источников опорного напряжения.

Экономичный внутренний ИОН

Встроенный генератор опорного напряжения модуля ADC10 разработан для использования в приложениях с низким энергопотреблением. В составе генератора имеется источник напряжения (ИОН) и отдельный буфер. Значение тока, потребляемого каждым из указанных узлов, приводится в справочной документации на конкретные модели. При REFON = 1 оба узла включены, а при REFON = 0 — выключены. Полное время установления опорного напряжения после установки бита REFON составляет не более 30 мкс.

Если REFON = 1, но преобразование не выполняется, то буфер автоматически выключается, а включается (тоже автоматически), когда в нём возникает потребность. В выключенном состоянии ток потребления буфера равен нулю. Источник напряжения в этом случае остаётся активным.

При REFOUT = 1 работой буфера внутреннего источника опорного напряжения управляет бит REFBURST. При REFBURST = 0 буфер будет включён постоянно, обеспечивая наличие опорного напряжения на соответствующем выводе микроконтроллера. Если REFBURST = 1, то буфер автоматически выключается между преобразованиями и автоматически включается на время преобразования.

Также имеется возможность выбора между быстродействием АЦП и током потребления буфера встроенного ИОН. При скорости преобразования менее 50 тыс. выборок/с установка бита ADC10SR = 1 уменьшает ток потребления буфера примерно на 50%.

20.2.4. Автоматическое отключение

Модуль ADC10 разработан для использования в приложениях с низким энергопотреблением. Поэтому ядро автоматически отключается в те периоды времени, когда модуль не выполняет преобразований, и автоматически включается при необходимости. Аналогичным образом автоматически включается и отключается внутренний генератор модуля ADC10SC. В выключенном состоянии ток потребления, как ядра АЦП, так и внутреннего генератора, равен нулю.

20.2.5. Синхронизация выборки и преобразования

Запуск аналого-цифрового преобразования производится по нарастающему фронту сигнала выборки SHI. Источником сигнала SHI, который задаётся битами SHSx, может быть:

- бит ADC10SC;
- модуль вывода 1 Таймера A;
- модуль вывода 0 Таймера A;
- модуль вывода 2 Таймера A.

Полярность источника сигнала SHI может быть инвертирована посредством бита ISSH. Биты SHTx определяют период выборки t_{sample}, который может быть равен 4, 8, 16 или 64 тактам ADC10CLK. Таймер выборки переводит сигнал SAMPCON в состояние ВЫСОКОГО уровня на время, равное заданному времени выборки, после синхронизации с сигналом ADC10CLK. Полное время выборки составляет $t_{sample} + t_{sync}$. Запуск аналого-цифрового преобразования, для выполнения которого требуется 13 тактов ADC10CLK, производится по спадающему фронту сигнала SAMPCON (**Рис. 20.3**).

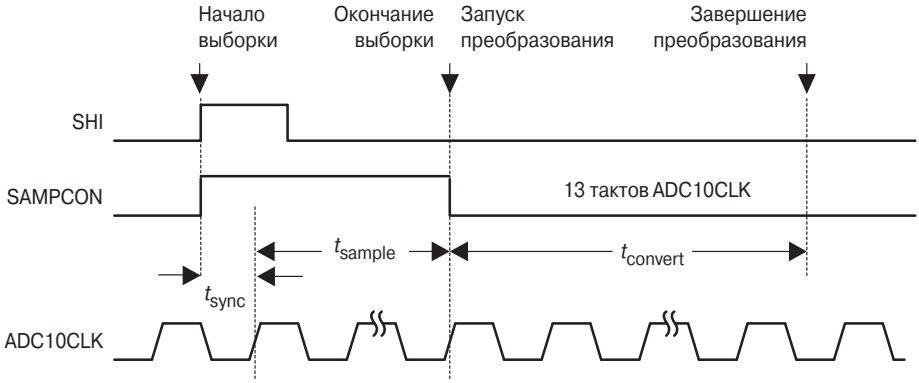


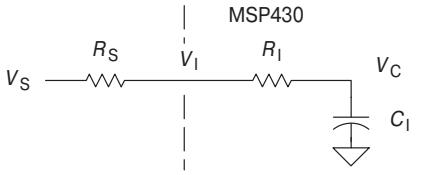
Рис. 20.3. Временные диаграммы процесса выборки.

Определение времени выборки

При SAMPCON = 0 все аналоговые входы Ax находятся в высокоимпедансном состоянии. При SAMPCON = 1 выбранный вход Ax на время выборки t_{sample} можно представить в виде *RC*-фильтра нижних частот, как показано на **Рис. 20.4**. Источник сигнала оказывается нагруженным на входное сопротивление мультиплексора R_I (не более 2 кОм), включённое последовательно с конденсатором C_I (не более 27 пФ). Для выполнения преобразования с 10-битной точностью напряжение V_C на конденсаторе C_I должно отличаться от напряжения источника V_S не более чем на $^1/_2$ LSB.

Время выборки t_{sample} зависит от сопротивления источника R_S и сопротивления R_I. Минимальное время выборки для выполнения 10-битного преобразования можно определить по формуле:

$$t_{sample} > (R_S + R_I)\cdot\ln(2^{11})\cdot C_I.$$



V_I — Входное напряжение на выводе Ax
V_S — Напряжение внешнего источника
R_S — Сопротивление внешнего источника
R_I — Входное сопротивление внутреннего мультиплексора во включённом состоянии
C_I — Входная ёмкость
V_C — Напряжение заряда конденсатора

Рис. 20.4. Эквивалентная схема аналогового входа.

Подставляя в это выражение значения R_I и C_I, приведённые выше, получаем:

$$t_{sample} > (R_S + 2\ \text{кОм})\cdot 7.625\cdot 27\ \text{пФ}.$$

Например, при R_S = 10 кОм время выборки должно быть более 2.47 мкс.

При использовании буфера опорного напряжения в пакетном режиме, время выборки должно быть больше как вычисленного значения, так и времени установления буфера $t_{REFBURST}$:

$$t_{sample} > \begin{cases} (R_S + R_I)\times \ln(2^{11})\times C_I \\ t_{REFBURST} \end{cases}$$

Например, если V_{Ref} = 1.5 В и R_S = 10 кОм, то время t_{sample} должно быть больше 2.47 мкс при ADC10SR = 0 или же больше 2.5 мкс при ADC10SR = 1. Точные значения приводятся в справочной документации на конкретные модели.

При использовании внешнего опорного напряжения время установления буфера определяется по формуле:

$$t_{REFBURST} = SR\cdot V_{ref} - 0.5\ \text{мкс},$$

где SR — скорость нарастания сигнала буфера (~1 мкс/В при ADC10SR = 0 и ~2 мкс/В при ADC10SR = 1);

V_{Ref} — значение внешнего опорного напряжения.

20.2.6. Режимы преобразования

Модуль ADC10 поддерживает четыре режима работы, задаваемые битами CONSEQx. Эти режимы перечислены в **Табл. 20.1**.

Таблица 20.1. **Режимы преобразования модуля ADC10**

CONSEQx	Режим	Операция
00	Однократный одноканальный	Однократно выполняется преобразование одного канала
01	Однократный последовательный	Однократно выполняется серия преобразований нескольких последовательно расположенных каналов (по одному преобразованию на канал)
10	Циклический одноканальный	Периодически выполняется преобразование одного канала
11	Циклический последовательный	Периодически выполняется серия преобразований нескольких последовательно расположенных каналов (по одному преобразованию на канал)

Однократный одноканальный режим преобразования

В этом режиме выполняется однократное преобразование канала, заданного битами INCHx. Результат преобразования заносится в регистр ADC10MEM. Диаграмма состояний модуля ADC10 для однократного одноканального режима преобразования приведена на **Рис. 20.5**. Если запуск преобразования производится установкой бита ADC10SC, то для запуска последующих преобразований достаточно установить этот бит повторно. Если же запуск преобразования осуществляется по сигналу от другого источника, то между последовательными преобразованиями необходимо сбрасывать и повторно устанавливать бит ENC.

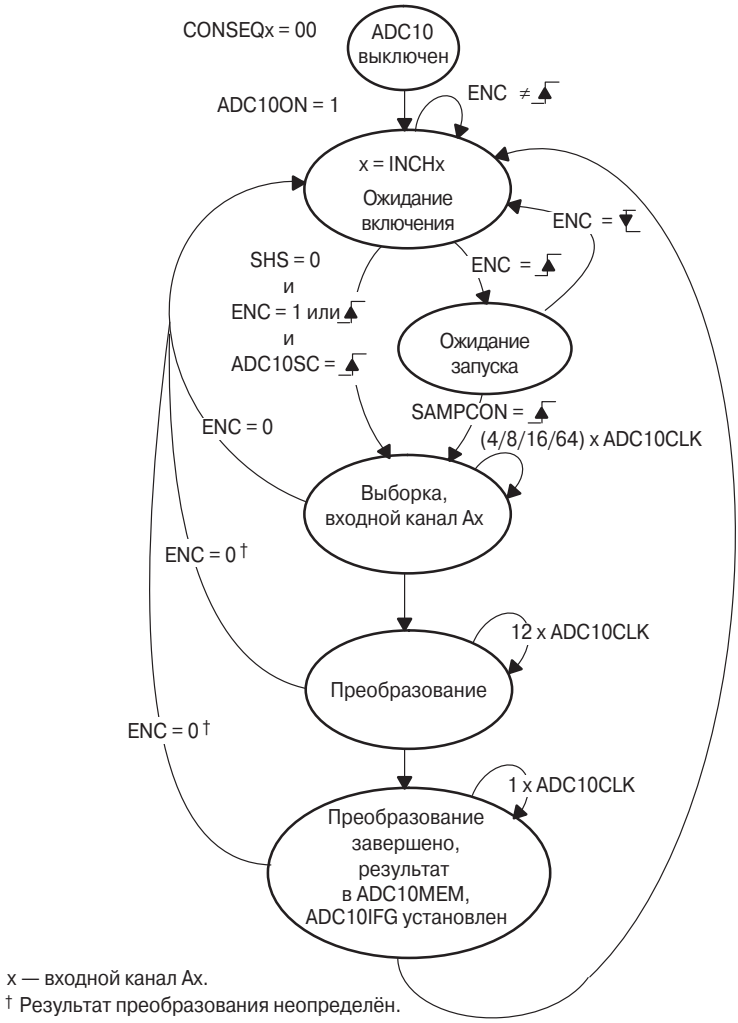


Рис. 20.5. Однократный одноканальный режим преобразования.

Однократный последовательный режим преобразования

В этом режиме выполняется цикл преобразований нескольких последовательно расположенных каналов, по одному преобразованию на канал. Первый канал последовательности задаётся битами INCHx, а последним является канал A0. Результат каждого преобразования заносится в регистр ADC10MEM. Диаграмма состояний модуля ADC10 для однократного последовательного режима преобразования приведена на **Рис. 20.6**. Если запуск цикла преобразований производится установкой бита ADC10SC, то для запуска последующих циклов достаточно установить этот бит повторно. Если же запуск цикла преобразований осуществляется по сигналу от другого источника, то между последовательными циклами необходимо сбрасывать и повторно устанавливать бит ENC.

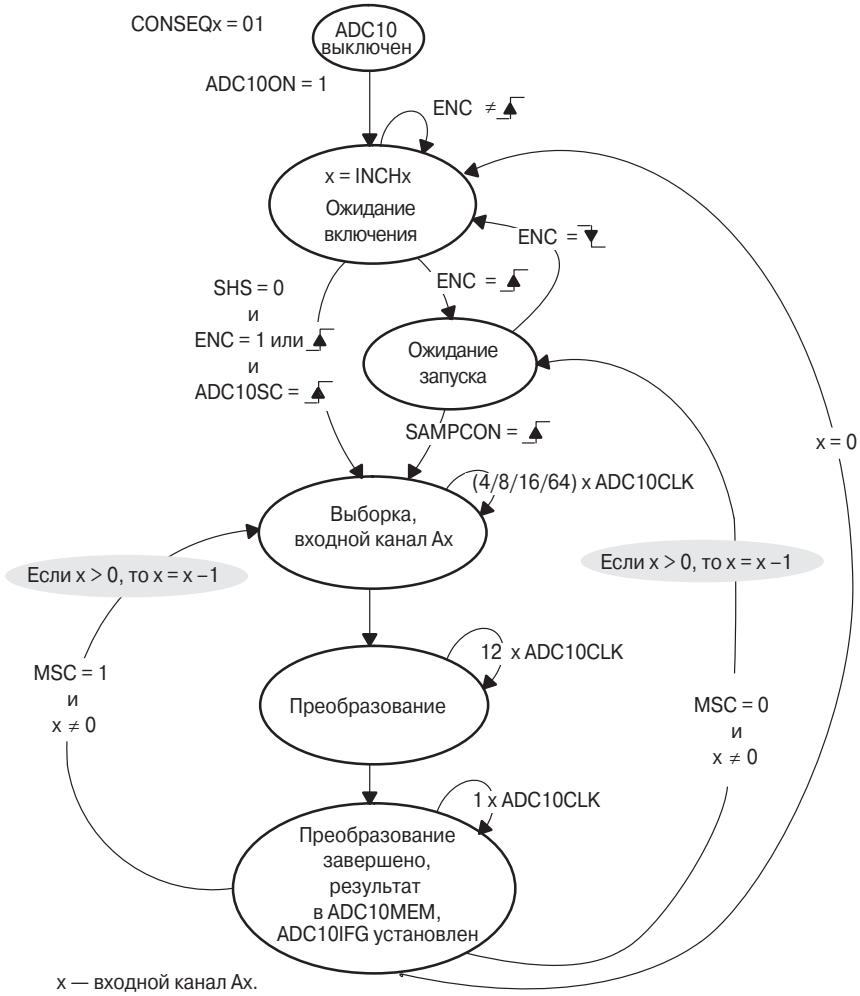


Рис. 20.6. Однократный последовательный режим преобразования.

Циклический одноканальный режим преобразования

В этом режиме периодически выполняется преобразование канала, заданного битами INCHx. Результат преобразования заносится в регистр ADC10MEM. Диаграмма состояний модуля ADC10 для циклического одноканального режима преобразования приведена на **Рис. 20.7**.

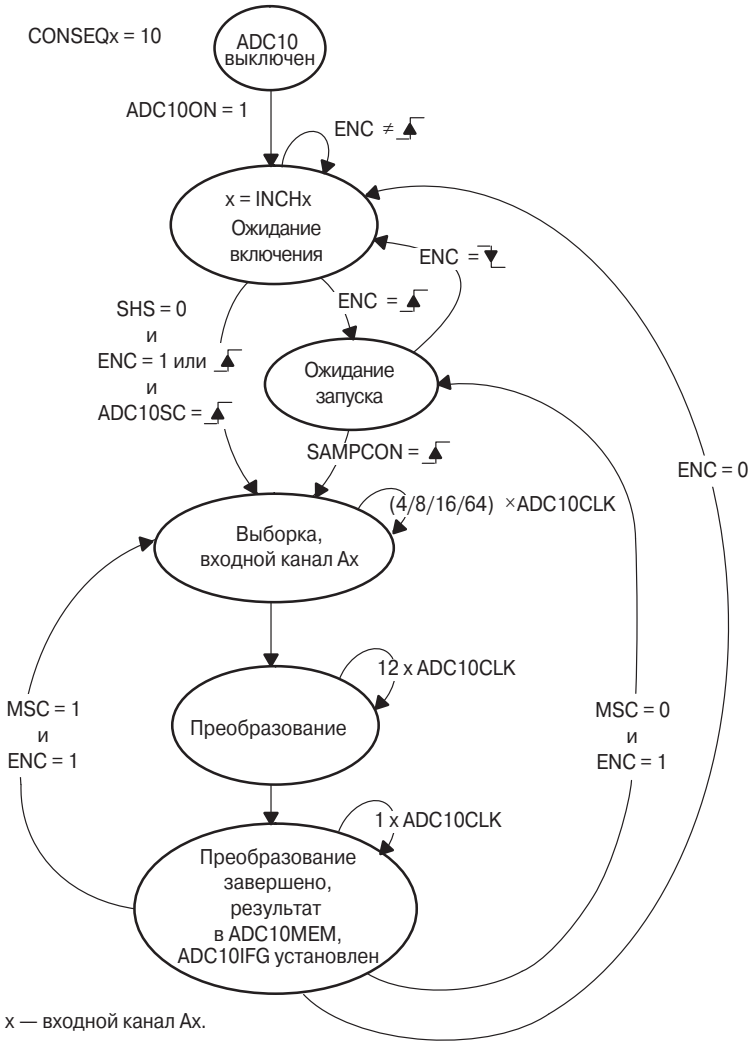


Рис. 20.7. Циклический одноканальный режим преобразования.

Циклический последовательный режим преобразования

В этом режиме периодически выполняется цикл преобразований нескольких последовательно расположенных каналов, по одному преобразованию на канал. Первый канал последовательности задаётся битами INCHx, а последним является канал A0. Результат каждого преобразования заносится в регистр ADC10MEM. Цикл завершается преобразованием канала A0, а повторный запуск цикла производится по следующему сигналу запуска. Диаграмма состояний модуля ADC10 для циклического последовательного режима преобразования приведена на **Рис. 20.8**.

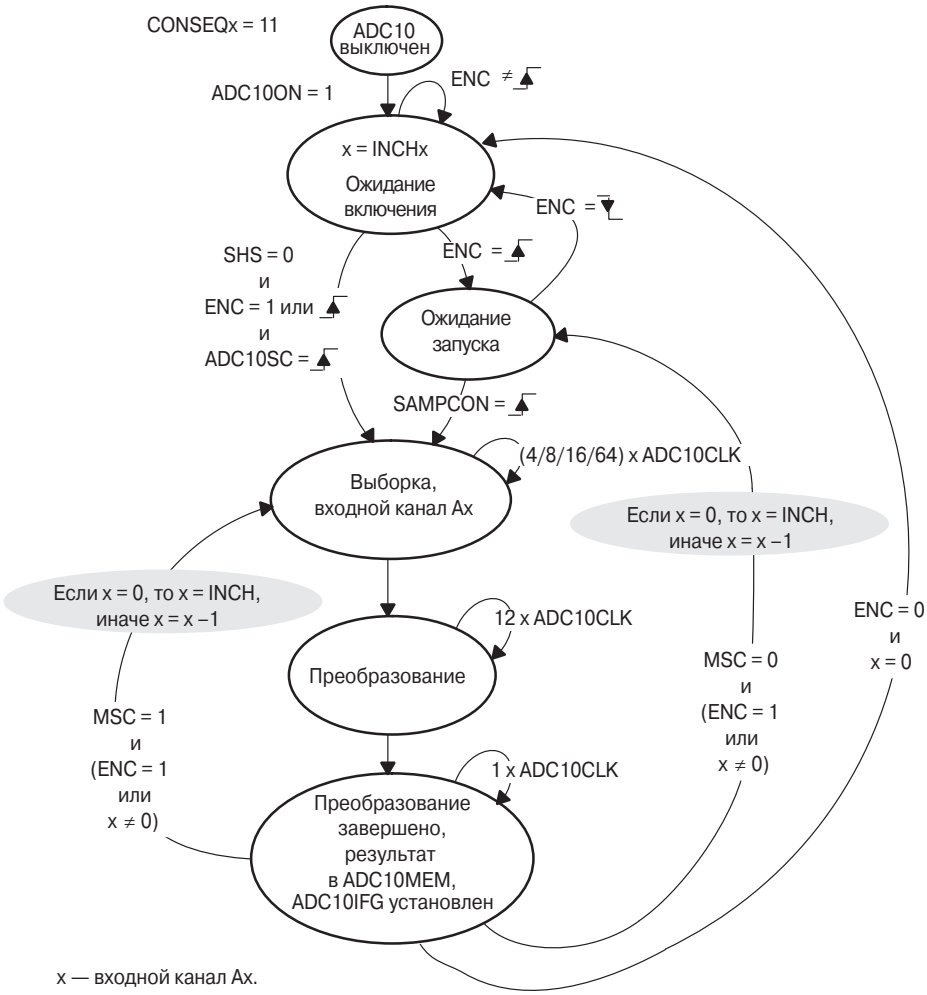


Рис. 20.8. Циклический последовательный режим преобразования.

Использование бита MSC

Чтобы АЦП могло автоматически выполнять последовательные преобразования с максимально возможной скоростью, в модуле предусмотрена функция многократной выборки/преобразования. При MSC = 1 и CONSEQx > 0 первый нарастающий фронт сигнала SHI запускает первое преобразование. Последующие преобразования запускаются автоматически сразу же после завершения предыдущего. Прочие нарастающие фронты сигнала SHI будут игнорироваться до тех пор, пока не будет завершена серия преобразований (однократный последовательный режим) или пока не будет сброшен и повторно установлен бит ENC (циклический одноканальный или циклический последовательный режимы). Функция бита ENC при использовании бита MSC не меняется.

Останов преобразований

Останов модуля ADC10 производится по-разному, в зависимости от его режима работы. Для прекращения активного преобразования (серии преобразований) рекомендуется следующее:

- Сброс бита EMC в однократном одноканальном режиме немедленно останавливает процесс преобразования, при этом результат преобразования будет неопределённым. Для получения корректного результата перед сбросом бита EMC необходимо дождаться сброса бита ADC10BUSY.
- Сброс бита EMC в циклическом одноканальном режиме останавливает АЦП после завершения текущего преобразования.
- Сброс бита ENC в однократном или циклическом последовательном режимах останавливает процесс преобразования после завершения последовательности преобразований.
- В любом режиме процесс преобразования может быть моментально остановлен загрузкой нулевого значения в биты CONSEQx и сбросом бита ENC. Результат преобразования в этом случае будет некорректным.

20.2.7. Контроллер передачи данных модуля ADC10

В составе модуля ADC10 имеется контроллер передачи данных (DTC), предназначенный для автоматической пересылки результатов преобразования из регистра ADC10MEM в другие ячейки внутренней памяти. Контроллер DTC включается при записи в регистр ADC10DTC1 ненулевого значения.

При включённом контроллере пересылка данных производится каждый раз, когда модуль завершает преобразование и загружает результат в регистр ADC10MEM. Со стороны программы не требуется никаких действий по управлению модулем до тех пор, пока не будет передано заданное число отсчётов. Каждая пересылка выполняется за один такт MCLK. Чтобы при пересылке данных не возник конфликт на шине, ЦПУ останавливается, если оно было активно, на один такт сигнала MCLK, требуемый для пересылки значения.

Пересылка не должна инициироваться в то время, когда АЦП занято. То есть конфигурирование контроллера необходимо производить в те моменты, когда преобразование или серия преобразований не выполняется:

```
; Проверка активности ADC10
          BIC.W #ENC,&ADC10CTL0  ;
busy_test BIT.W #BUSY,&ADC10CTL1 ;
          JNZ   busy_test        ;
          MOV.W #xxx,&ADC10SA    ; Безопасно
          MOV.B #xx,&ADC10DTC1   ;
; Продолжаем конфигурирование
```

Режим пересылки одного блока

Режим пересылки одного блока включается при бите ADC10TB = 0. Значение *n*, загружаемое в регистр ADC10TC1, определяет общее число пересылок в блоке. Начальный адрес блока, который заносится в 16-битный регистр ADC10SA, может находиться в любом месте адресного пространства MSP430. Конец блока располагается по адресу ADC10SA+2*n*–2. Режим пересылки одного блока показан на **Рис. 20.9**.

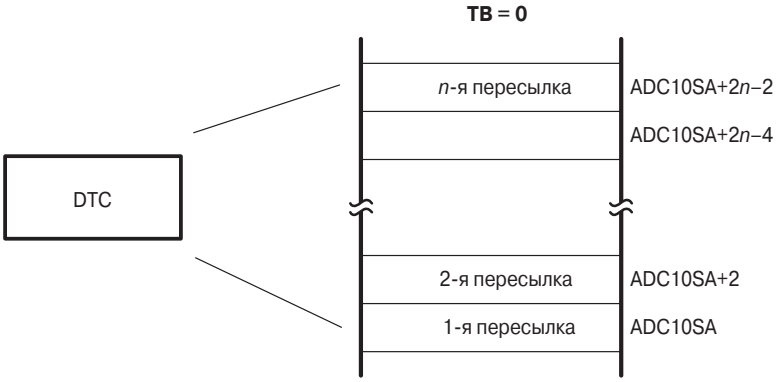


Рис. 20.9. Пересылка одного блока.

Первоначально внутренний указатель адреса равен ADC10SA, а внутренний счётчик пересылок равен *n*. Внутренние указатель и счётчик программно недоступны. Контроллер DTC пересылает 16-битное содержимое регистра ADC10MEM по адресу, находящемуся в ADC10SA. После каждой пересылки внутренний указатель адреса увеличивается на два, а счётчик пересылок — декрементируется.

Контроллер продолжает пересылать данные при каждой загрузке в ADC10MEM до тех пор, пока внутренний счётчик не станет равным нулю. После этого контроллер возобновит пересылку данных только после повторной записи в регистр ADC10SA. При использовании DTC в режиме пересылки одного блока, флаг ADC10IFG устанавливается только после пересылки всего блока. Диаграмма состояний для этого режима приведена на **Рис. 20.10**.

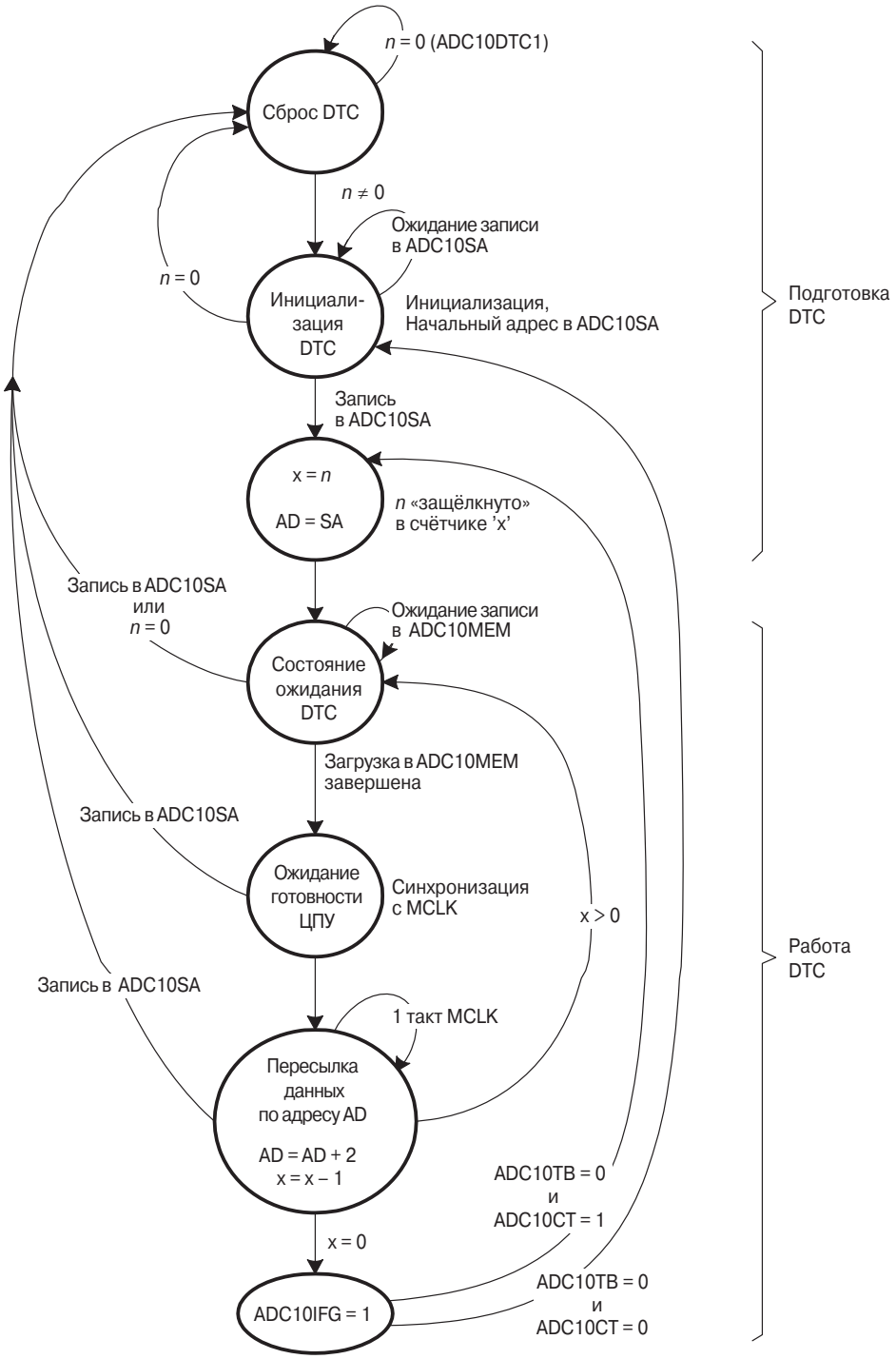


Рис. 20.10. Диаграмма состояний DTC в режиме пересылки одного блока.

Режим пересылки двух блоков

Режим пересылки двух блоков включается при бите ADC10TB = 1. Значение *n*, загружаемое в регистр ADC10TC1, определяет число пересылок для одного блока. Начальный адрес первого блока, который заносится в 16-битный регистр ADC10SA, может находиться в любом месте адресного пространства MSP430. Конец первого блока располагается по адресу ADC10SA+2*n*–2. Второй блок располагается по адресам с ADC10SA+2*n* по ADC10SA+4*n*–1. Режим пересылки двух блоков показан на **Рис. 20.11**.

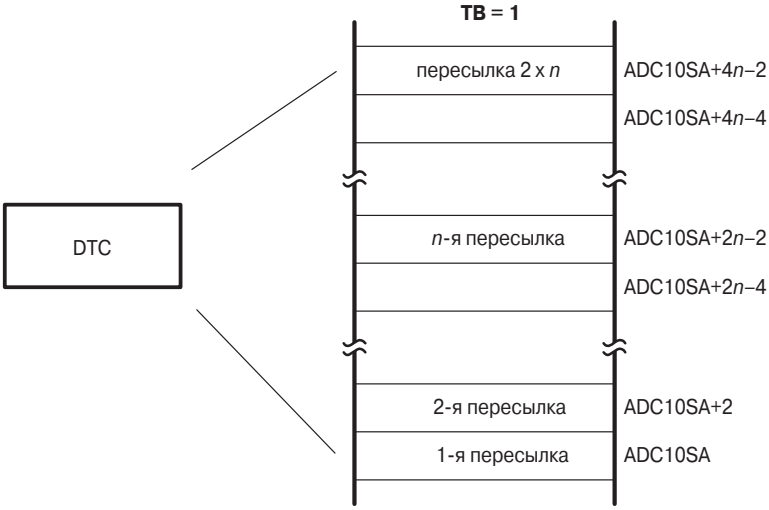


Рис. 20.11. Пересылка двух блоков.

Первоначально внутренний указатель адреса равен ADC10SA, а внутренний счётчик пересылок равен *n*. Внутренние указатель и счётчик программно недоступны. Контроллер DTC пересылает 16-битное содержимое регистра ADC10MEM по адресу, находящемуся в ADC10SA. После каждой пересылки внутренний указатель адреса увеличивается на два, а счётчик пересылок — декрементируется.

Контроллер продолжает пересылать данные при каждой загрузке в ADC10MEM до тех пор, пока внутренний счётчик не станет равным нулю. При этом устанавливаются флаг ADC10IFG и бит ADC10B1, свидетельствуя о заполнении 1-го блока. Пользователь может определить завершение пересылки первого блока, проверив состояние бита ADC10B1.

Контроллер DTC продолжает работать со вторым блоком. В счётчик пересылок автоматически перезагружается число *n*. При загрузке следующего результата в регистр ADC10MEM контроллер начинает пересылку результатов преобразования во второй блок. Этот блок заполняется после выполнения *n* пересылок, при этом устанавливается флаг ADC10IFG, а бит ADC10B1 сбрасывается. Пользователь может определить завершение пересылки блока, проверив состояние бита ADC10B1. Диаграмма состояний для этого режима приведена на **Рис. 20.12**.

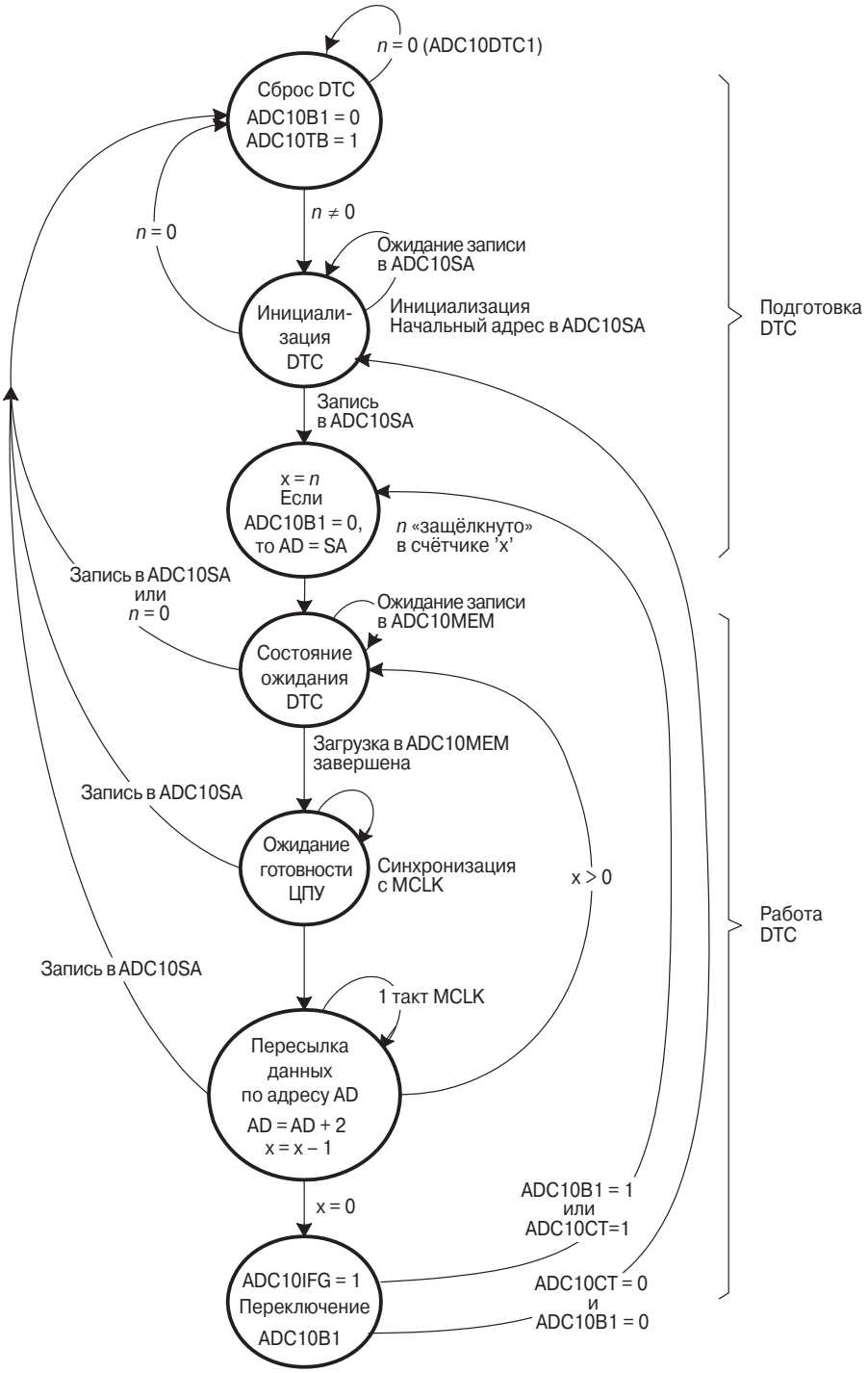


Рис. 20.12. Диаграмма состояний DTC в режиме пересылки двух блоков.

Непрерывная передача данных

Режим непрерывной передачи данных включается установкой бита ADC10CT. В этом случае контроллер DCT не останавливается после завершения пересылки 1-го (режим пересылки одного блока) или 2-го (режим пересылки двух блоков) блока. Во внутренний указатель адреса и внутренний счётчик пересылок автоматически загружаются значения ADC10SA и *n* соответственно, и контроллер вновь начинает пересылать данные в 1-й блок. При сбросе бита ADC10CT контроллер DCT прекращает пересылку данных после заполнения 1-го (режим пересылки одного блока) или 2-го (режим пересылки двух блоков) блока.

Длительность цикла пересылки

Для пересылки одного значения из регистра ADC10MEM контроллеру DTC требуется один или два такта MCLK для синхронизации, один такт для выполнения собственно пересылки (при остановленном ЦПУ) и один такт для ожидания. Поскольку контроллер DTC тактируется сигналом MCLK, длительность цикла пересылки зависит от режима работы MSP430 и настроек модуля синхронизации.

Если источник сигнала MCLK активен, а ЦПУ выключено, то контроллер DTC использует этот источник для выполнения всех пересылок, при этом ЦПУ не включается. Если источник MCLK неактивен, то контроллер DTC временно (только на время пересылки) активирует сигнал MCLK, используя в качестве источника сигнал DCOCLK. Центральный процессор остаётся отключённым, а сигнал MCLK повторно отключается после завершения пересылки контроллером DTC. Максимальные длительности цикла пересылки для различных режимов работы приведены в **Табл. 20.2**.

Таблица 20.2. **Максимальная длительность цикла пересылки DTC**

Режим работы ЦПУ	Источник тактового сигнала	Максимальная длительность цикла пересылки
Активный режим	MCLK = DCOCLK	3 такта MCLK
Активный режим	MCLK = LFXT1CLK	3 такта MCLK
Режим пониженного энергопотребления LPM0/1	MCLK = DCOCLK	4 такта MCLK
Режим пониженного энергопотребления LPM3/4	MCLK = DCOCLK	4 такта MCLK + 2 мкс*
Режим пониженного энергопотребления LPM0/1	MCLK = LFXT1CLK	4 такта MCLK
Режим пониженного энергопотребления LPM3	MCLK = LFXT1CLK	4 такта MCLK
Режим пониженного энергопотребления LPM4	MCLK = LFXT1CLK	4 такта MCLK + 2 мкс*

* Дополнительные 2 мкс требуются для запуска генератора DCO. Более подробная информация приводится в справочной документации на конкретные модели.

20.2.8. Использование встроенного датчика температуры

Для использования встроенного датчика температуры пользователь выбирает соответствующий аналоговый вход INCHx = 1010. Остальные настройки АЦП, такие как выбор опорного напряжения, выбор представления результата и т.п., выполняются как при работе с внешним сигналом.

Типичная передаточная функция датчика температуры приведена на **Рис. 20.13**. При использовании датчика температуры время выборки должно быть не менее 30 мкс. Этот датчик имеет большую погрешность смещения и для получения абсолютных значений температуры требует калибровки. Более подробная информация приводится в справочной документации на конкретные модели.

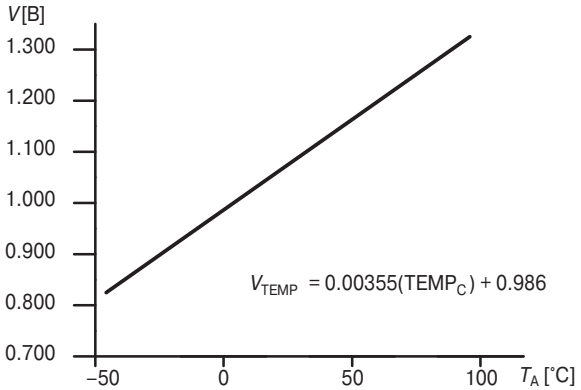


Рис. 20.13. Типичная передаточная функция встроенного датчика температуры.

При выборе датчика температуры встроенный генератор опорного напряжения автоматически задействуется в качестве источника напряжения для датчика. Однако выход V_{REF+} при этом не включается и настройки источника опорного напряжения для преобразования не изменяются. Выбор опорного напряжения для преобразования сигнала от датчика температуры производится так же, как и для любого другого канала преобразования.

20.2.9. Заземление и борьба с помехами при использовании модуля ADC10

При использовании модуля ADC10, как и в случае любого другого АЦП с высокой разрешающей способностью, необходимо следовать определённым рекомендациям по разводке печатной платы и организации заземления с тем, чтобы избежать появления паразитных земляных контуров, нежелательных эффектов и шумов.

Паразитные земляные контуры появляются в том случае, если обратный ток от АЦП протекает по проводникам, являющимися общими с другими аналоговыми или цифровыми элементами схемы. Если не принять специальных мер, то этот ток может вызвать появление небольшого напряжения смещения, влияющего на величину опорного напряжения или уровень входного сигнала аналого-цифрового преобразователя. Подключение модуля АЦП, позволяющее этого избежать, показано на **Рис. 20.14** и **Рис. 20.15**.

Помимо паразитных земляных токов, на результат преобразования также могут оказывать влияние пульсации и выбросы на шинах питания, возникающие при работе цифровых узлов или импульсных источников питания. Отсутствие

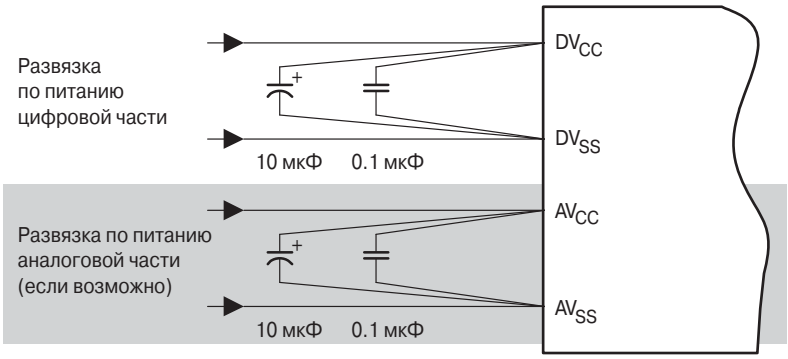


Рис. 20.14. Заземление и борьба с помехами при использовании модуля ADC10 (внутреннее V_{REF}).

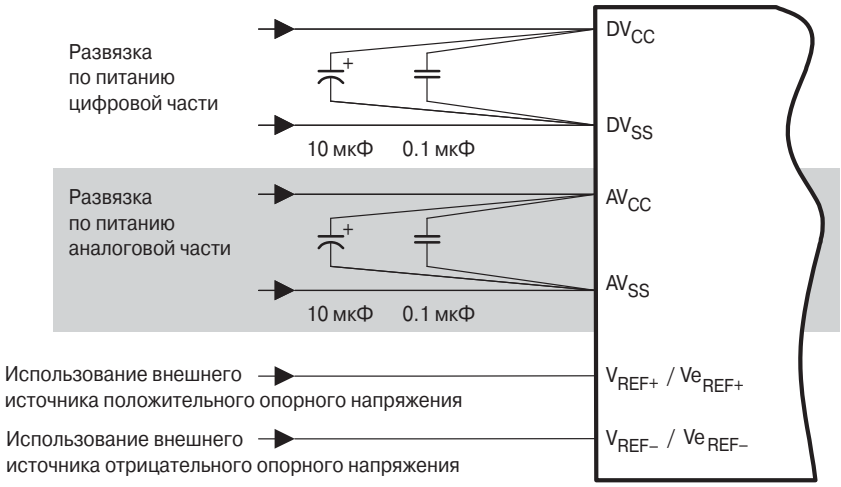


Рис. 20.15. Заземление и борьба с помехами при использовании модуля ADC10 (внешнее V_{REF}).

шумов является одним из основных факторов, позволяющих достичь высокой точности преобразований.

20.2.10. Прерывания модуля ADC10

Модулю ADC10 выделен один вектор прерывания, с которым связан единственный флаг (**Рис. 20.16**). Если контроллер DTC не используется (ADC10DTC1 = 0), то флаг прерывания ADC10IFG устанавливается при загрузке результата преобразования в регистр ADC10MEM. При использовании контроллера DTC (ADC10DTC1 > 0) флаг ADC10IFG устанавливается, когда завершается пересылка блока данных и внутренний счётчик пересылок становится равным нулю. Если установлены биты ADC10IE и GIE, то при установке флага ADC10IFG генерируется запрос прерывания. Флаг ADC10IFG автоматически сбрасывается при обработке прерывания или может быть сброшен программно.

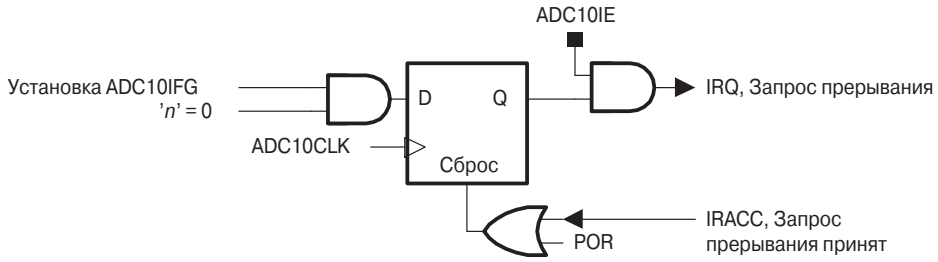


Рис. 20.16. Система прерывания модуля ADC10.

20.3. Регистры модуля ADC10

Список регистров модуля ADC10 приведён в **Табл. 20.3**.

Таблица 20.3. **Регистры модуля ADC10**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр 0 разрешения входа модуля ADC10	ADC10AE0	Чтение/запись	04Ah	Сбрасывается после POR
Регистр 1 разрешения входа модуля ADC10	ADC10AE1	Чтение/запись	04Bh	Сбрасывается после POR
Регистр управления 0 модуля ADC10	ADC10CTL0	Чтение/запись	01B0h	Сбрасывается после POR
Регистр управления 1 модуля ADC10	ADC10CTL1	Чтение/запись	01B2h	Сбрасывается после POR
Регистр данных модуля ADC10	ADC10MEM	Чтение	01B4h	Не изменяется
Регистр управления 0 пересылкой данных модуля ADC10	ADC10DTC0	Чтение/запись	048h	Сбрасывается после POR
Регистр управления 1 пересылкой данных модуля ADC10	ADC10DTC1	Чтение/запись	049h	Сбрасывается после POR
Регистр начального адреса пересылки данных модуля ADC10	ADC10SA	Чтение/запись	01BCh	0200h после POR

ADC10CTL0, регистр управления 0 модуля ADC10

15	14	13	12	11	10	9	8
SREFx			**ADC10SHTx**		**ADC10SR**	**REFOUT**	**REFBURST**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
MSC	**REF2_5V**	**REFON**	**ADC10ON**	**ADC10IE**	**ADC10IFG**	**ENC**	**ADC10SC**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Может быть изменён только при ENC = 0.

SREFx Биты 15...13 Выбор источника опорного напряжения.

000 $V_{R+} = V_{CC}$, $V_{R-} = V_{SS}$
001 $V_{R+} = V_{REF+}$, $V_{R-} = V_{SS}$
010 $V_{R+} = Ve_{REF+}$, $V_{R-} = V_{SS}$
011 V_{R+} = буферизованное Ve_{REF+}, $V_{R-} = V_{SS}$
100 $V_{R+} = V_{CC}$, $V_{R-} = V_{REF-}/Ve_{REF-}$
101 $V_{R+} = V_{REF+}$, $V_{R-} = V_{REF-}/Ve_{REF-}$
110 $V_{R+} = Ve_{REF+}$, $V_{R-} = V_{REF-}/Ve_{REF-}$
111 V_{R+} = буферизованное Ve_{REF+}, $V_{R-} = V_{REF-}/Ve_{REF-}$

ADC10SHTx	Биты 12...11	Время выборки. 00 4 такта ADC10CLK 01 8 тактов ADC10CLK 10 16 тактов ADC10CLK 11 64 такта ADC10CLK
ADC10SR	Бит 10	Частота выборок. Этот бит определяет максимальную частоту выборок, поддерживаемую буфером встроенного ИОН. Установка бита ADC10SR уменьшает ток потребления буфера. 0 Буфер поддерживает скорости до ~200 тыс. выборок/с 1 Буфер поддерживает скорости до ~50 тыс. выборок/с
REFOUT	Бит 9	Выход источника опорного напряжения. 0 Выход источника опорного напряжения отключен 1 Выход источника опорного напряжения включён
REFBURST	Бит 8	Управление буфером встроенного ИОН. 0 Буфер ИОН включён постоянно 1 Буфер ИОН включается только на время выборки/преобразования
MSC	Бит 7	Многократные выборки/преобразования. Используется только в последовательном или циклических режимах. 0 Каждая выборка/преобразование инициируется нарастающим фронтом сигнала SHI 1 Таймер выборки запускается по первому нарастающему фронту сигнала SHI, а остальные выборки/преобразования запускаются автоматически после завершения предшествующего преобразования
REF2_5V	Бит 6	Величина внутреннего опорного напряжения. Также должен быть установлен бит REFON. 0 1.5 В 1 2.5 В
REFON	Бит 5	Включение генератора опорного напряжения. 0 Генератор опорного напряжения выключен 1 Генератор опорного напряжения включён
ADC10ON	Бит 4	Включение модуля ADC10. 0 Модуль ADC10 выключен 1 Модуль ADC10 включён
ADC10IE	Бит 3	Разрешение прерывания модуля ADC10. 0 Прерывание запрещено 1 Прерывание разрешено
ADC10IFG	Бит 2	Флаг прерывания модуля ADC10. Данный флаг устанавливается при загрузке в регистр ADC10MEM результата преобразования. Этот флаг сбрасывается автоматически при обработке прерывания или может быть сброшен программно. При использовании контроллера DTC флаг ADC10IFG устанавливается после завершения пересылки блока данных. 0 Не было запроса прерывания 1 Есть запрос прерывания
ENC	Бит 1	Разрешение преобразования. 0 Работа ADC10 запрещена 1 Работа ADC10 разрешена
ADC10SC	Бит 0	Запуск выборки/преобразования. Этот бит используется для программного запуска процесса выборки и преобразования. Биты ADC10SC и ENC могут быть установлены одной командой. Бит ADC10SC сбрасывается автоматически. 0 Не начинать выборку/преобразование 1 Начать процесс выборки/преобразования

ADC10CTL1, регистр управления 1 модуля ADC10

15	14	13	12	11	10	9	8
INCHx				SHSx		ADC10DF	ISSH
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
ADC10DIVx			ADC10SSELx		CONSEQx		ADC10 BUSY
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–0

Может быть изменён только при ENC = 0.

INCHx Биты 15...12 Выбор входного канала. Эти биты определяют конкретный канал для однократных преобразований или канал с максимальным номером для последовательных преобразований.

0000 A0
0001 A1
0010 A2
0011 A3
0100 A4
0101 A5
0110 A6
0111 A7
1000 Ve_{REF+}
1001 V_{REF-}/Ve_{REF-}
1010 Датчик температуры
1011 $(V_{CC} - V_{SS})/2$
1100 $(V_{CC} - V_{SS})/2$, A12 в моделях MSP430x22xx
1101 $(V_{CC} - V_{SS})/2$, A13 в моделях MSP430x22xx
1110 $(V_{CC} - V_{SS})/2$, A14 в моделях MSP430x22xx
1111 $(V_{CC} - V_{SS})/2$, A15 в моделях MSP430x22xx

SHSx Биты 11...10 Источник сигнала запуска выборки/преобразования.

00 Бит ADC10SC
01 Модуль вывода 1 Таймера A
10 Модуль вывода 0 Таймера A
11 Модуль вывода 2 Таймера A (модуль вывода 1 в моделях MSP430x20x2)

ADC10DF Бит 9 Формат результата преобразования.

0 Обычный двоичный код
1 Дополнительный код

ISSH Бит 8 Инвертирование сигнала запуска выборки/преобразования.

0 Сигнал запуска выборки/преобразования не инвертируется
1 Сигнал запуска выборки/преобразования инвертируется

ADC10DIVx Биты 7...5 Коэффициент деления тактового сигнала модуля ADC10.

000 1
001 2
010 3
011 4
100 5
101 6
110 7
111 8

ADC10SSELx	Биты 4...3	Выбор источника тактового сигнала модуля ADC10. 00 ADC10SC 01 ACLK 10 MCLK 11 SMCLK
CONSEQx	Биты 2...1	Выбор режима преобразования. 00 Однократный одноканальный 01 Однократный последовательный 10 Циклический одноканальный 11 Циклический последовательный
ADC10BUSY	Бит 0	Модуль ADC10 занят. Этот бит показывает активность операций выборки или преобразования. 0 Нет активности 1 Производится выборка, преобразование или выполняется последовательность преобразований

ADC10AE0, регистр 0 разрешения аналоговых входов модуля ADC10

7	6	5	4	3	2	1	0
ADC10AE0x							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC10AE0x	Биты 7...0	Разрешение аналогового входа модуля ADC10. Эти биты разрешают работу соответствующих выводов порта в качестве аналоговых входов. Бит 0 соответствует входу A0, бит 1 — входу A1 и т.д. 0 Аналоговый вход выключен 1 Аналоговый вход включён

ADC10AE1, регистр 1 разрешения аналоговых входов модуля ADC10

7	6	5	4	3	2	1	0
ADC10AE1x				**Reserved**	**Reserved**	**Reserved**	**Reserved**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

ADC10AE1x	Биты 7...4	Разрешение аналогового входа модуля ADC10. Эти биты разрешают работу соответствующих выводов порта в качестве аналоговых входов. Бит 4 соответствует входу A12, бит 5 — входу A13, бит 6 — входу A14, а бит 7 — входу A15. 0 Аналоговый вход выключен 1 Аналоговый вход включен

ADC10MEM, регистр данных модуля ADC10, обычный код

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**Результат преобразования**	
r0	r0	r0	r0	r0	r0	r	r

7	6	5	4	3	2	1	0
Результат преобразования							
r	r	r	r	r	r	r	r

Результат преобразования	Биты 15...0	10-битный результат преобразования выровнен по правому краю, представлен в обычном двоичном коде. Бит 9 — MSB. Биты 15...10 всегда равны 0.

ADC10MEM, регистр данных модуля ADC10, дополнительный код

15	14	13	12	11	10	9	8
Результат преобразования							
r	r	r	r	r	r	r	r
7	6	5	4	3	2	1	0
Результат преобразования		**0**	**0**	**0**	**0**	**0**	**0**
r	r	r0	r0	r0	r0	r0	r0

Результат преобразования	Биты 15...0	10-битный результат преобразования выровнен по левому краю, представлен в дополнительном коде. Бит 15 — MSB. Биты 5...0 всегда равны 0.

ADC10DTC0, регистр управления 0 пересылкой данных модуля ADC10

7	6	5	4	3	2	1	0
Reserved				**ADC10TB**	**ADC10CT**	**ADC10B1**	**ADC10 FETCH**
r0	r0	r0	r0	rw–(0)	rw–(0)	r–(0)	rw–(0)

Reserved	Биты 7...4	Зарезервированы. Всегда читаются как 0.
ADC10TB	Бит 3	Режим пересылки. 0 Режим пересылки одного блока 1 Режим пересылки двух блоков
ADC10CT	Бит 2	Непрерывная передача данных. 0 Пересылка данных прекращается после заполнения одного (режим пересылки одного блока) или двух (режим пересылки двух блоков) блоков. 1 Данные пересылаются непрерывно. Контроллер DTC останавливается только при сбросе бита ADC10CT или при записи в регистр ADC10SA.
ADC10B1	Бит 1	Признак 1-го блока. В режиме пересылки двух блоков этот бит показывает, в какой из блоков заносятся результаты преобразований. Значение бита ADC10B1 становится корректным только после первой установки флага ADC10IFG во время работы контроллера DTC. Также должен быть установлен бит ADC10TB. 0 Блок 2 заполнен 1 Блок 1 заполнен
ADC10 FETCH	Бит 0	При нормальной работе этот бит должен быть сброшен.

ADC10DTC1, регистр управления 1 пересылкой данных модуля ADC10

7	6	5	4	3	2	1	0
DTC Transfers							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

DTC Transfers	Биты 7...0	Количество пересылок. Эти биты определяют число отсчётов в каждом блоке. 0 Контроллер DTC выключен 01h–0FFh Количество пересылок в блоке

ADC10SA, регистр начального адреса пересылки данных модуля ADC10

15	14	13	12	11	10	9	8
ADC10SAx							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
ADC10SAx							**0**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r0

ADC10SAx	Биты 15...1	Начальный адрес блока. Эти биты определяют начальный адрес для контроллера DTC. Запись в регистр ADC10SA требуется для запуска процесса пересылки данных.
Unused	Бит 0	Не используется, только для чтения. Всегда читается как 0.

ГЛАВА 21

МОДУЛЬ 12-БИТНОГО АЦП ADC12

Модуль ADC12 представляет собой быстродействующий 12-битный аналого-цифровой преобразователь. В этой главе описывается работа модуля ADC12, реализованного в микроконтроллерах семейства MSP430x2xx.

21.1. Введение

Модуль ADC12 обеспечивает быстрое выполнение 12-битных аналого-цифровых преобразований. В составе модуля имеется 12-битное ядро с регистром последовательного приближения, блок управления выборкой, генератор опорного напряжения и буфер (conversion memory) размером 16 слов. Этот буфер позволяет модулю формировать до 16 независимых выборок с последующим преобразованием и сохранением результата без использования ЦПУ.

Модуль ADC12 имеет следующие особенности:

- максимальная скорость преобразования более 200 тыс. выборок/с;
- 12-битный преобразователь с монотонной характеристикой без пропуска кодов;
- устройство выборки/хранения с программируемым временем выборки, определяемым таймерами или программно;
- запуск преобразования производится программно, по сигналу от Таймера A или по сигналу от Таймера B;
- программно конфигурируемый внутренний генератор опорного напряжения (1.5 В или 2.5 В);
- внешний или внутренний источник опорного напряжения (выбирается программно) ;
- восемь каналов преобразования внешних сигналов;
- каналы преобразования для внутреннего датчика температуры, напряжения AV_{CC} и внешних опорных напряжений;
- независимые источники положительного и отрицательного опорного напряжений, задаваемые отдельно для каждого канала;
- конфигурируемый источник тактового сигнала;

- четыре режима преобразования: одноканальный, многократный одноканальный, последовательный и многократный последовательный;
- ядро АЦП и источник опорного напряжения могут выключаться независимо друг от друга;
- регистр вектора прерывания для быстрого декодирования 18 прерываний АЦП;
- 16 регистров для хранения результатов преобразований.

Блок-схема модуля ADC12 приведена на **Рис. 21.1**.

21.2. Функционирование модуля ADC12

Конфигурирование модуля ADC12 осуществляется пользовательской программой. Настройка модуля и его функционирование рассматриваются в следующих подразделах.

21.2.1. Ядро 12-битного АЦП

Ядро АЦП преобразует аналоговый сигнал в 12-битный цифровой код и сохраняет результат в буфере. Верхний и нижний пределы преобразования определяются двумя программируемыми уровнями напряжения (V_{R+} и V_{R-}). Результат преобразования равен максимальному значению (0FFFh), если уровень входного сигнала больше или равен V_{R+}, и нулю, если уровень входного сигнала меньше или равен V_{R-}. Входной канал и уровни опорных напряжений (V_{R+} и V_{R-}) задаются в блоке управления преобразованием, входящем в состав буфера данных. Результат аналого-цифрового преобразования определяется выражением:

$$N_{\mathrm{ADC}} = 4095 \times \frac{V_{\mathrm{in}} - V_{\mathrm{R-}}}{V_{\mathrm{R+}} - V_{\mathrm{R-}}}.$$

Конфигурирование ядра АЦП осуществляется двумя регистрами управления ADC12CTL0 и ADC12CTL1. Включается ядро АЦП установкой бита ADC12ON. Если модуль ADC12 не используется, то его можно отключить для уменьшения потребления. За некоторыми исключениями биты управления модуля ADC12 можно изменять только при ENC = 0. Перед выполнением любого преобразования бит ENC должен быть установлен в 1.

Выбор тактового сигнала преобразования

Сигнал ADC12CLK используется как в качестве тактового сигнала преобразования, так и для формирования периодов выборки при использовании импульсного режима выборки. Источник сигнала ADC12CLK задаётся битами ADC12SSELx. Выбранный сигнал поступает на АЦП через предделитель, коэффициент деления которого (1, 2, 4 или 8) определяется битами ADC12DIVx. Для формирования сигнала ADC12CLK могут использоваться системные тактовые сигналы SMCLK, MCLK, ACLK, а также сигнал внутреннего генератора ADC12OSC.

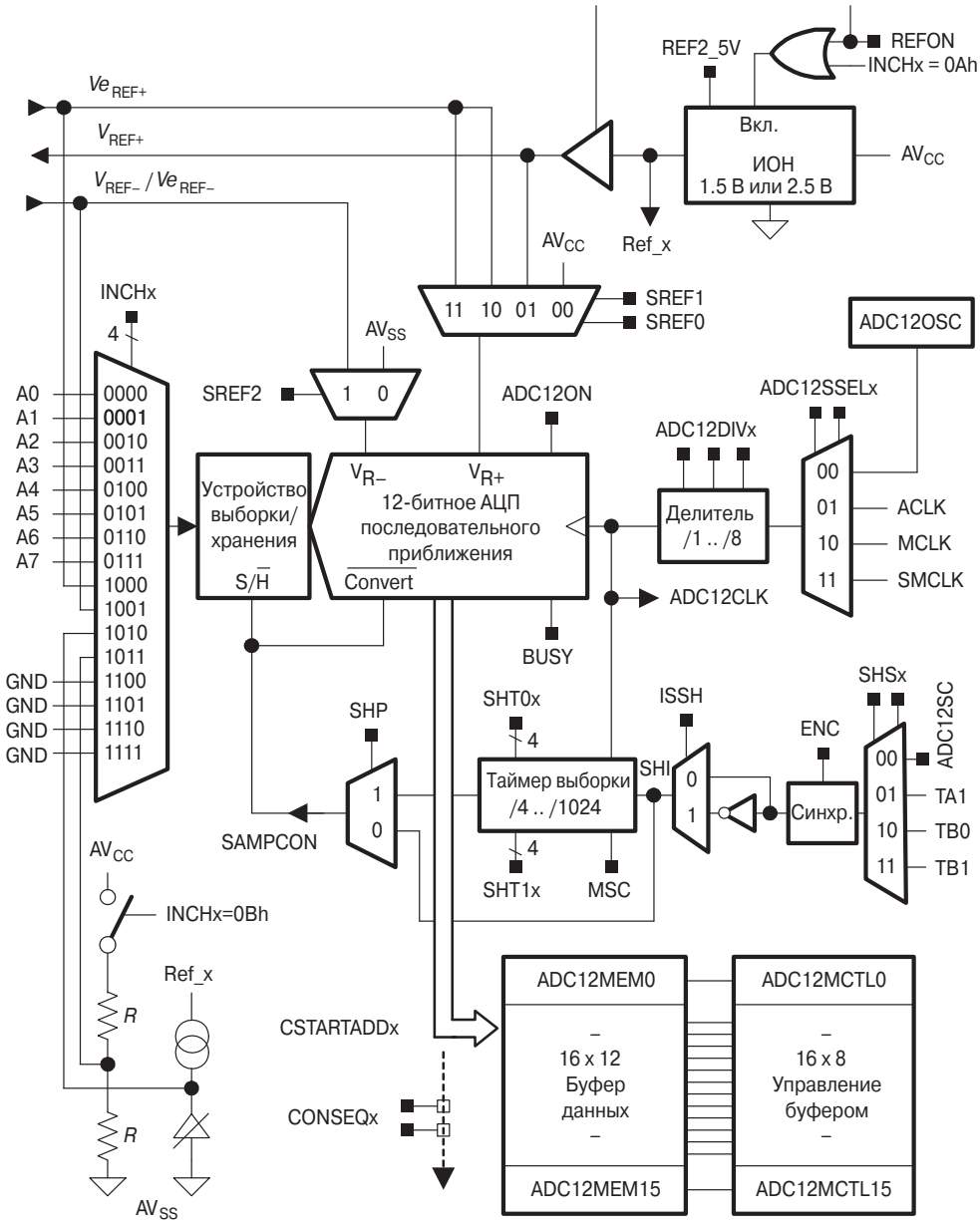


Рис. 21.1. Блок-схема модуля ADC12.

Частота сигнала ADC12OSC, формируемого внутри модуля, не превышает 5 МГц и зависит от конкретного устройства, напряжения питания и температуры. Точные параметры сигнала ADC12CLK приводятся в документации на конкретные модели.

Пользователь должен гарантировать, что тактовый сигнал, используемый для формирования ADC12CLK, останется активным до завершения процесса преобразования. Если тактовый сигнал будет остановлен во время преобразования, то операция завершена не будет, а полученный результат будет неверным.

21.2.2. Входы модуля ADC12 и мультиплексор

Входной аналоговый мультиплексор позволяет выбирать для преобразования один из 8 внешних и 4 внутренних аналоговых сигналов. Мультиплексор (**Рис. 21.2**) выполнен по схеме break-before-make (разрыв перед включением) для уменьшения перекрёстных помех, возникающих при переключении каналов. Кроме того, ключи мультиплексора выполнены по Т-образной схеме, позволяющей свести к минимуму взаимное влияние каналов. Неиспользуемые в данный момент времени каналы отключаются от АЦП, а средняя точка подключается к аналоговой земле AV_{SS}. В результате паразитная ёмкость заземляется, что помогает устранить перекрёстные помехи.

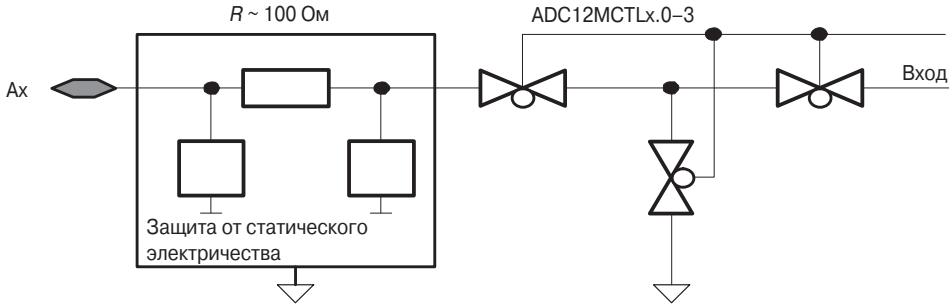


Рис. 21.2. Аналоговый мультиплексор.

Модуль ADC12 использует метод перераспределения заряда. Переключение входов внутри модуля может вызвать переходные процессы во входном сигнале. Эти переходные процессы затухают достаточно быстро, чтобы исключить возможность некорректного преобразования.

Выбор аналогового порта

Внешние входы модуля ADC12 мультиплексированы с линиями порта P6, которые представляют собой цифровые схемы на КМОП-элементах. Аналоговый сигнал, поступающий на вход КМОП-элемента, может вызвать сквозной ток, протекающий от линии питания V_{CC} к земле (GND). Этот паразитный ток появляется в том случае, если входное напряжение близко к пороговому напряжению элемента. Отключение буфера вывода порта исключает появление паразитного тока и, таким образом, уменьшает общее потребление микроконтроллера. Биты P6SELx позволяют отключать входные и выходные буферы соответствующих выводов порта.

```
; P6.0 и P6.1 используются как аналоговые входы
  BIS.B #03h,&P6SEL  ; P6.0 и P6.1 - аналоговые входы для ADC12
```

21.2.3. Генератор опорного напряжения

Модуль ADC12 содержит встроенный генератор опорного напряжения, способный формировать напряжение двух номиналов — 1.5 В и 2.5 В. Это напряжение может использоваться внутри кристалла, а также может быть выведено на вывод V_{REF+}.

Генератор опорного напряжения включается установкой бита REFON = 1. При REF2_5V = 1 внутреннее опорное напряжение составляет 2.5 В, а при REF2_5V = 0 внутреннее опорное напряжение равно 1.5 В. Если встроенный генератор опорного напряжения не используется, то его можно выключить, чтобы уменьшить ток потребления микроконтроллера.

Для стабильной работы встроенного генератора опорного напряжения между выводами V_{REF+} и AV_{SS} необходимо подключить внешний конденсатор. Рекомендуется использовать два параллельно соединённых конденсатора с ёмкостями 0.1 мкФ и 10 мкФ. После включения генератора опорного напряжения для полного заряда этих конденсаторов требуется не более 17 мс. Если внутреннее опорное напряжение не используется при выполнении преобразования, то внешние конденсаторы не требуются.

Примечание. Развязка опорного напряжения
При определении двух младших битов результата во время преобразования от любого источника опорного напряжения, используемого модулем ADC12, потребляется ток около 200 мкА. В связи с эти рекомендуется к выходу любого используемого источника опорного напряжения подключать параллельно соединённые конденсаторы емкостью 10 мкФ и 0.1 мкФ, как показано далее на **Рис. 21.11**.

Уровни V_+ и V_- могут задаваться внешними источниками, подключаемыми к выводам Ve_{REF+} и V_{REF-}/Ve_{REF-} соответственно.

21.2.4. Синхронизация выборки и преобразования

Запуск аналого-цифрового преобразования производится по нарастающему фронту сигнала выборки SHI. Источником сигнала SHI, который задаётся битами SHSx, может быть:

- бит ADC12SC;
- модуль вывода 1 Таймера A;
- модуль вывода 0 Таймера B;
- модуль вывода 1 Таймера B.

Полярность источника сигнала SHI может быть инвертирована посредством бита ISSH. Период выборки и момент запуска преобразования определяются внутренним сигналом SAMPCON. Выборка производится при ВЫСОКОМ уровне сигнала SAMPCON. Запуск аналого-цифрового преобразования, которое выполняется за 13 тактов ADC12CLK, производится при смене уровня сигнала SAMPCON с ВЫСОКОГО на НИЗКИЙ. Бит SHP определяет один из двух режимов выборки — расширенный режим или импульсный режим.

Расширенный режим выборки

Расширенный режим выборки выбирается при SHP = 0. Сигнал SAMPCON управляется непосредственно сигналом выборки SHI, который и определяет время выборки t_{sample}. Выборка осуществляется при ВЫСОКОМ уровне сигнала SAMPCON. Запуск аналого-цифрового преобразования производится при смене уровня сигнала SAMPCON с ВЫСОКОГО на НИЗКИЙ после синхронизации с тактовым сигналом ADC12CLK (**Рис. 21.3**).

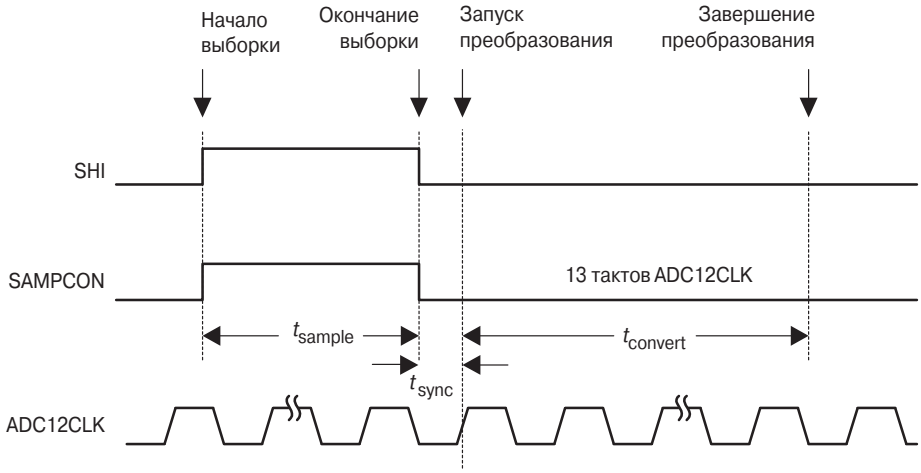


Рис. 21.3. Расширенный режим выборки.

Импульсный режим выборки

Импульсный режим выборки выбирается при SHP = 1. В этом случае сигнал SHI используется для запуска таймера выборки. Период этого таймера, который определяет время выборки t_{sample}, задаётся битами SHT0x и SHT1x регистра ADC12CTL0. Таймер выборки переводит сигнал SAMPCON в состояние ВЫСОКОГО уровня на время, равное заданному времени выборки t_{sample}, после синхронизации с тактовым сигналом ADC12CLK (**Рис. 21.4**).

Биты SHTx задают время выборки с дискретностью, равной 4 тактам ADC12CLK. Биты SHT0x задают время выборки для регистров ADC12MCTL0...7, а биты SHT1x — для регистров ADC12MCTL8...15.

Определение времени выборки

При SAMPCON = 0 все аналоговые входы Ax находятся в высокоимпедансном состоянии. При SAMPCON = 1 выбранный вход Ax на время выборки t_{sample} можно представить в виде *RC*-фильтра нижних частот, как показано на **Рис. 21.5**. Источник сигнала оказывается нагруженным на входное сопротивление мультиплексора R_I (не более 2 кОм), включённое последовательно с конденсатором C_I (не более 40 пФ). Для выполнения преобразования с 12-битной точностью на-

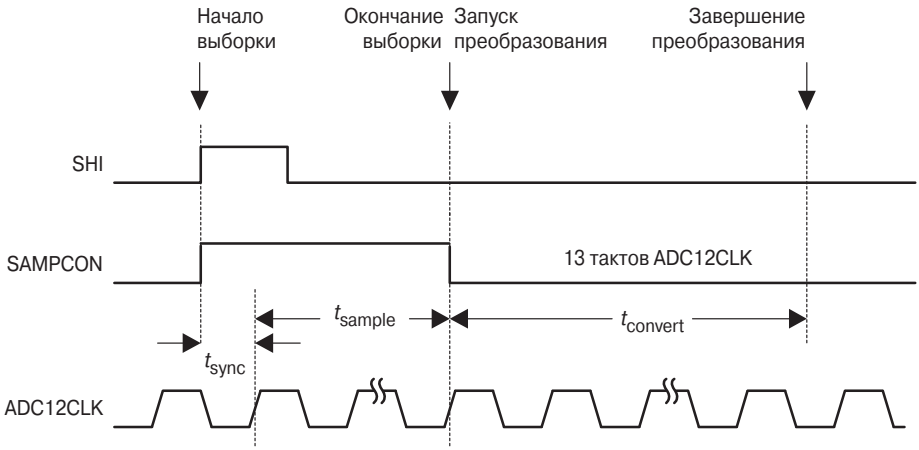


Рис. 21.4. Импульсный режим выборки.

пряжение V_C на конденсаторе C_I должно отличаться от напряжения источника V_S не более чем на $^1/_2$ LSB.

Время выборки t_{sample} зависит от сопротивления источника R_S и сопротивления R_I. Минимальное время выборки для выполнения 12-битного преобразования можно определить по формуле:

$$t_{sample} > (R_S + R_I)\cdot\ln(2^{13})\cdot C_I + 800 \text{ нс}.$$

Подставляя в это выражение значения R_I и C_I, приведённые выше, получаем:

$$t_{sample} > (R_S + 2 \text{ кОм})\cdot 9.011\cdot 40 \text{ пФ} + 800 \text{ нс}.$$

Например, при R_S = 10 кОм время выборки должно быть более 5.13 мкс.

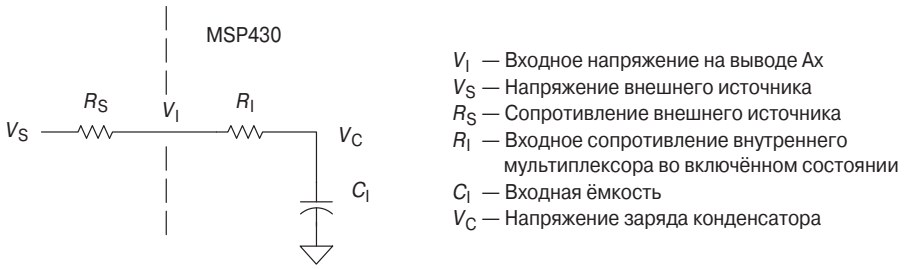


Рис. 21.5. Эквивалентная схема аналогового входа.

21.2.5. Сохранение результатов преобразования

В модуле ADC12 предусмотрено 16 регистров ADC12MEMx для хранения результатов преобразований. Конфигурация каждого из регистров ADC12MEMx задаётся соответствующим регистром управления ADC12MCTLx. Биты SREFx этого регистра задают опорные напряжения, а биты INCHx определяют входной канал. Бит EOS определяет конец последовательности при использовании последовательных режимов преобразований. Если бит EOS регистра ADC12MCTL15 не установлен, то после сохранения результата очередного преобразования последовательности в регистре ADC12MEM15 последующие результаты будут сохраняться, начиная с регистра ADC12MEM0.

Биты CSTARTADDx определяют 1-й регистр ADC12MCTLx, используемый для преобразования, независимо от режима работы. При работе в однократном одноканальном или циклическом одноканальном режимах биты CSTARTADDx задают единственный используемый регистр ADC12MCTLx.

Если модуль работает в однократном последовательном либо циклическом последовательном режимах, то биты CSTARTADDx определяют первый из регистров ADC12MCTLx последовательности. После завершения очередного преобразования указатель, недоступный из программы, автоматически инкрементируется, указывая на следующий регистр ADC12MCTLx. Перебор регистров продолжается до тех пор, пока не встретится регистр с установленным битом EOS — это будет последний регистр последовательности. При загрузке результата преобразования в очередной регистр ADC12MEMx устанавливается соответствующий флаг в регистре ADC12IFGx.

21.2.6. Режимы преобразования

Модуль ADC12 поддерживает четыре режима работы, задаваемые битами CONSEQx. Эти режимы приведены в **Табл. 21.1**.

Таблица 21.1. **Режимы преобразования модуля ADC12**

CONSEQx	Режим	Операция
00	Однократный одноканальный	Однократно выполняется преобразование одного канала
01	Однократный последовательный	Однократно выполняется серия преобразований нескольких каналов (по одному преобразованию на канал)
10	Циклический одноканальный	Периодически выполняется преобразование одного канала
11	Циклический последовательный	Периодически выполняется серия преобразований нескольких каналов (по одному преобразованию на канал)

Однократный одноканальный режим преобразования

В этом режиме выполняется однократное преобразование одного канала. Результат преобразования заносится в регистр ADC12MEMx, определяемый битами CSTARTADDx. Диаграмма состояний модуля ADC12 для однократного одноканального режима преобразования приведена на **Рис. 21.6**.

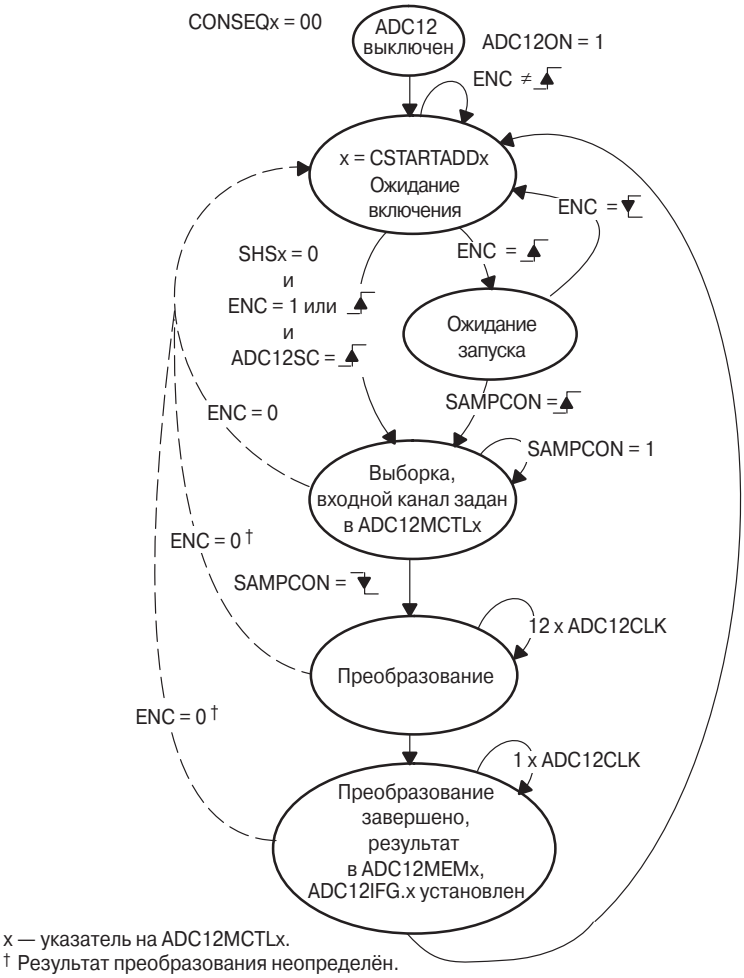


Рис. 21.6. Однократный одноканальный режим преобразования.

Если запуск преобразования производится установкой бита ADC12SC, то для запуска последующих преобразований достаточно установить этот бит повторно. Если же запуск преобразования осуществляется по сигналу от другого источника, то между последовательными преобразованиями необходимо сбрасывать и повторно устанавливать бит ENC.

Однократный последовательный режим преобразования

В этом режиме выполняется цикл преобразований нескольких каналов, по одному преобразованию на канал. Результаты преобразований заносятся в буфер, начиная с регистра ADC12MEMx, заданного битами CSTARTADDx. Последовательность завершается после измерения в канале с установленным битом EOS. Диаграмма состояний модуля ADC12 для однократного последовательного режима преобразования приведена на **Рис. 21.7**.

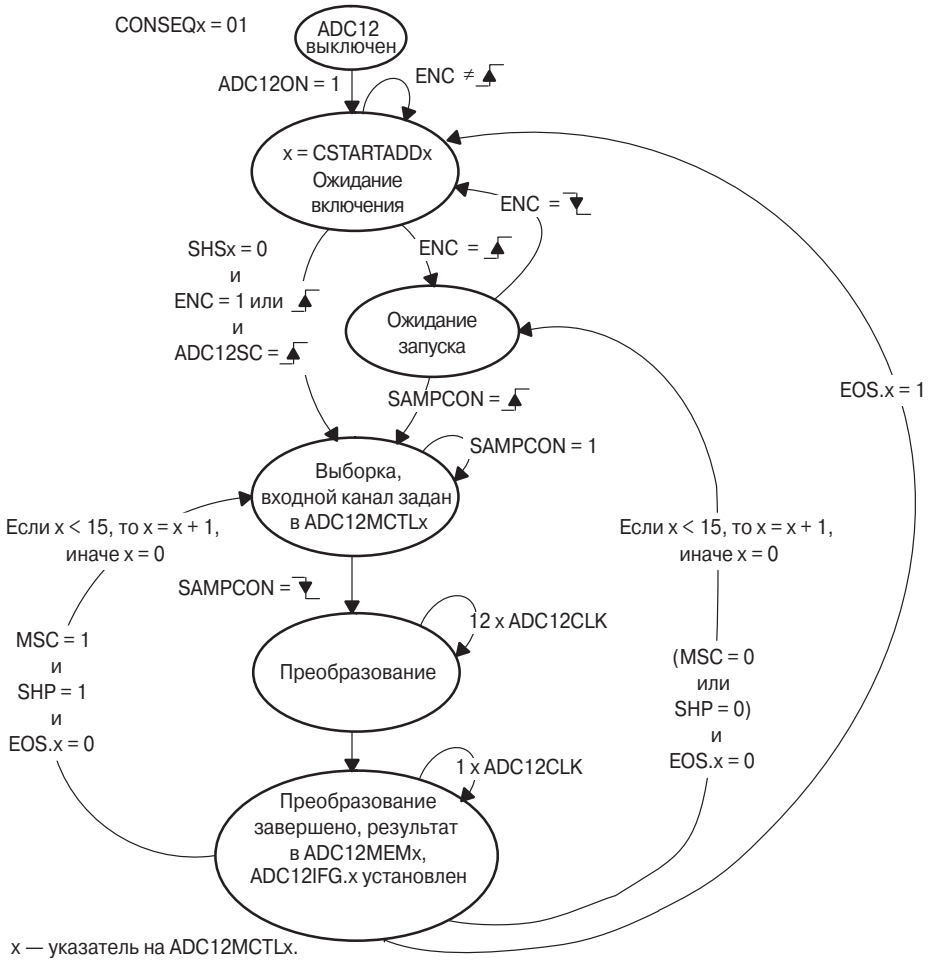


Рис. 21.7. Однократный последовательный режим преобразования.

Если запуск цикла преобразований производится установкой бита ADC12SC, то для запуска последующих циклов достаточно установить этот бит повторно. Если же запуск цикла преобразований осуществляется по сигналу от другого источника, то между последовательными циклами необходимо сбрасывать и повторно устанавливать бит ENC.

Циклический одноканальный режим преобразования

В этом режиме периодически выполняется преобразование одного канала. Результат преобразования заносится в регистр ADC12MEMx, определяемый битами CSTARTADDx. Поскольку в данном режиме результаты всех преобразований сохраняются в один и тот же регистр ADC12MEMx, необходимо выполнять чтение этого регистра после каждого преобразования — в противном случае текущий результат будет перезаписан после следующего преобразования. Диаграмма состояний модуля ADC12 для циклического одноканального режима преобразования приведена на **Рис. 21.8**.

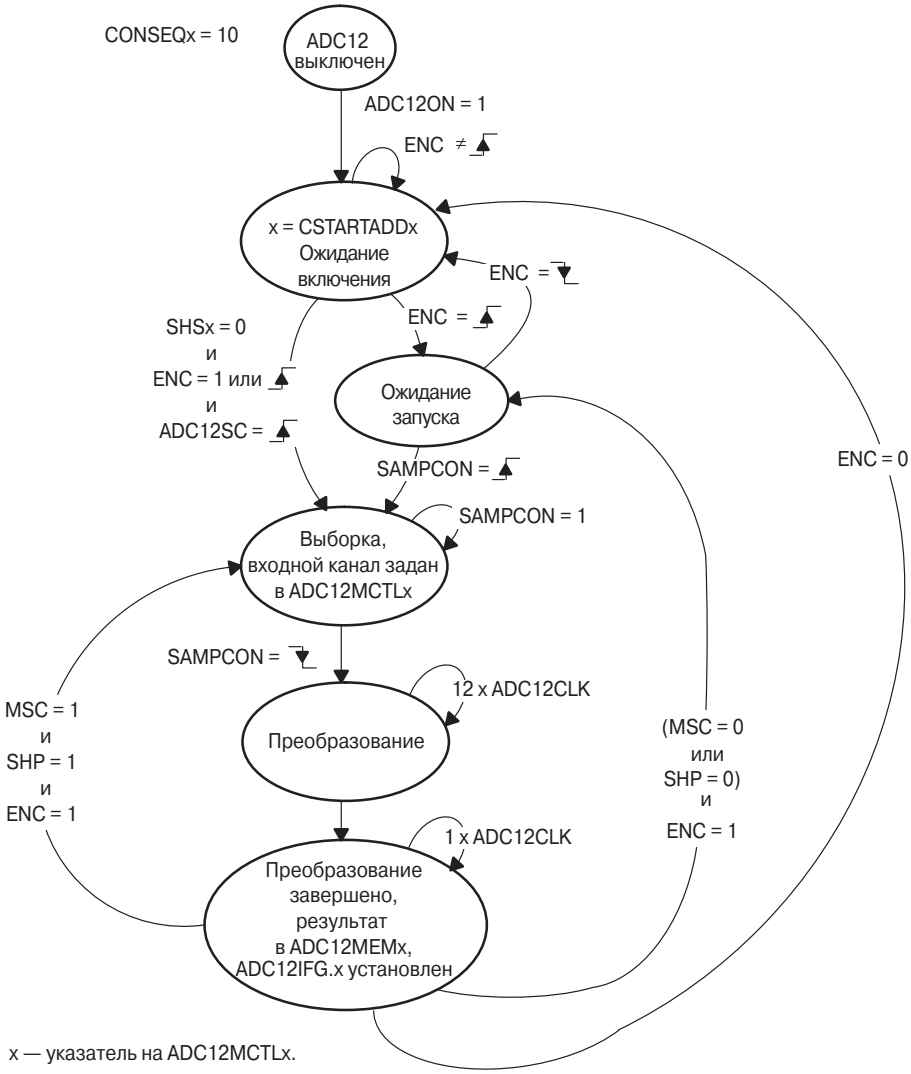


Рис. 21.8. Циклический одноканальный режим преобразования.

Циклический последовательный режим преобразования

В этом режиме периодически выполняется цикл преобразований нескольких каналов, по одному преобразованию на канал. Результаты преобразований заносятся в буфер, начиная с регистра ADC12MEMx, заданного битами CSTARTADDx. Последовательность преобразований заканчивается после измерения в канале с установленным битом EOS в регистре ADC12MCTLx, а повторный запуск цикла производится по следующему сигналу запуска. Диаграмма состояний модуля ADC12 для циклического последовательного режима преобразования приведена на **Рис. 21.9**.

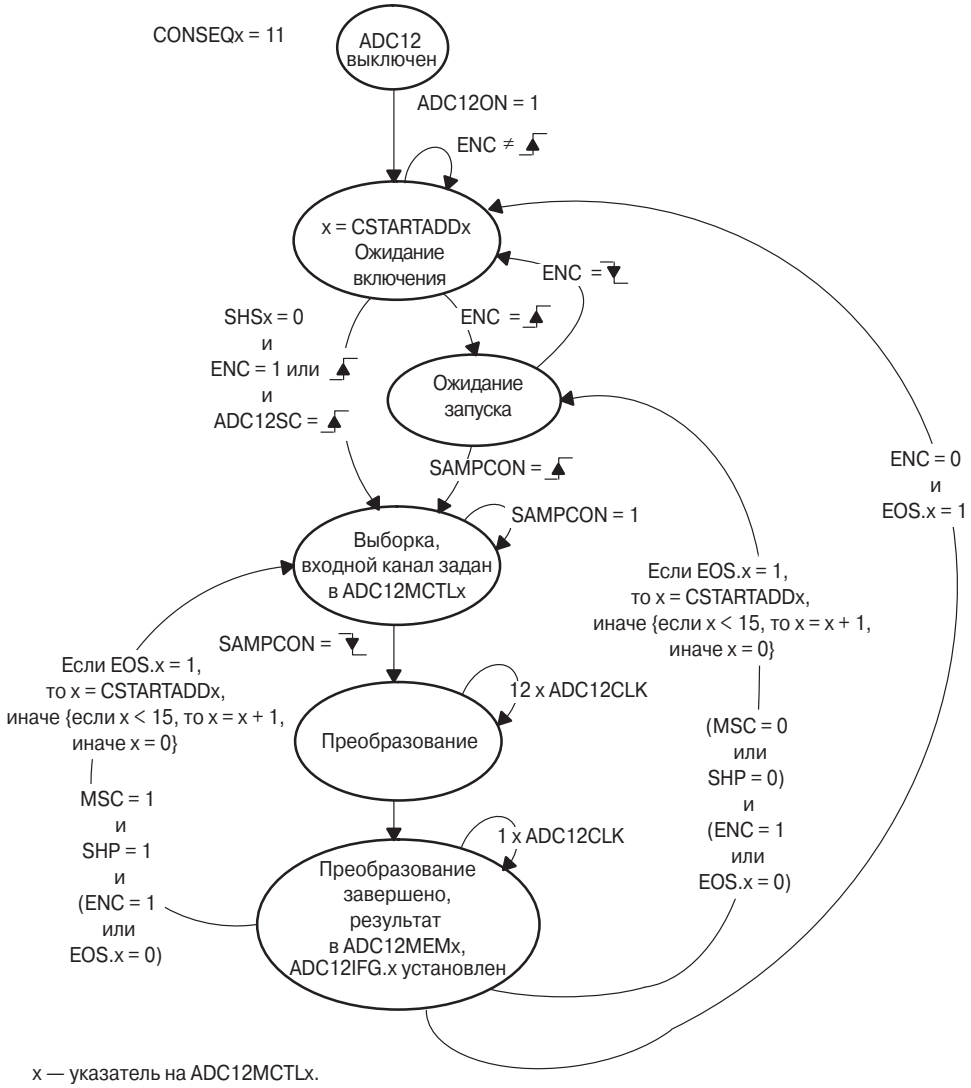


Рис. 21.9. Циклический последовательный режим преобразования.

Использование бита MSC

Чтобы АЦП мог автоматически выполнять последовательные преобразования с максимально возможной скоростью, в модуле предусмотрена функция многократной выборки/преобразования. При MSC = 1, CONSEQx > 0 и использовании таймера выборки первый нарастающий фронт сигнала SHI запускает первое преобразование. Последующие преобразования запускаются автоматически сразу же после завершения предыдущего. Прочие нарастающие фронты сигнала SHI будут игнорироваться до тех пор, пока не будет завершена серия преобразований (однократный последовательный режим) или пока не будет сброшен и повторно установлен бит ENC (циклический одноканальный или циклический последовательный режимы). Функция бита ENC при использовании бита MSC не меняется.

Останов преобразований

Останов модуля ADC12 производится по-разному, в зависимости от его режима работы. Для прекращения активного преобразования (серии преобразований) рекомендуется следующее:

- Сброс бита EMC в однократном одноканальном режиме немедленно останавливает процесс преобразования, при этом результат преобразования будет неопределённым. Для получения корректного результата перед сбросом бита EMC необходимо дождаться сброса бита ADC12BUSY.
- Сброс бита EMC в циклическом одноканальном режиме останавливает АЦП после завершения текущего преобразования.
- Сброс бита ENC в однократном или циклическом последовательном режимах останавливает процесс преобразования после завершения последовательности преобразований.
- В любом режиме процесс преобразования может быть моментально остановлен загрузкой нулевого значения в биты CONSEQx и сбросом бита ENC. Результат преобразования в этом случае будет некорректным.

Примечание. Отсутствие установленного бита EOS при преобразовании последовательности каналов

Если используется один из последовательных режимов преобразований и ни в одном из регистров ADC12MCTLx не установлен бит EOS, то сброс бита ENC не приведёт к завершению текущей последовательности. Для останова последовательности необходимо сначала выбрать одноканальный режим и только затем сбросить бит ENC.

21.2.7. Использование встроенного датчика температуры

Для использования встроенного датчика температуры пользователь выбирает соответствующий аналоговый вход INCHx = 1010. Остальные настройки АЦП, такие как выбор опорного напряжения, выбор регистра буфера и т.п., выполняются как при работе с внешним сигналом.

Типичная передаточная функция датчика температуры приведена на **Рис. 21.10**. При использовании датчика температуры время выборки должно

быть не менее 30 мкс. Этот датчик имеет большую погрешность смещения и для получения абсолютных значений температуры в большинстве случаев требует калибровки. Более подробная информация приводится в справочной документации на конкретные модели.

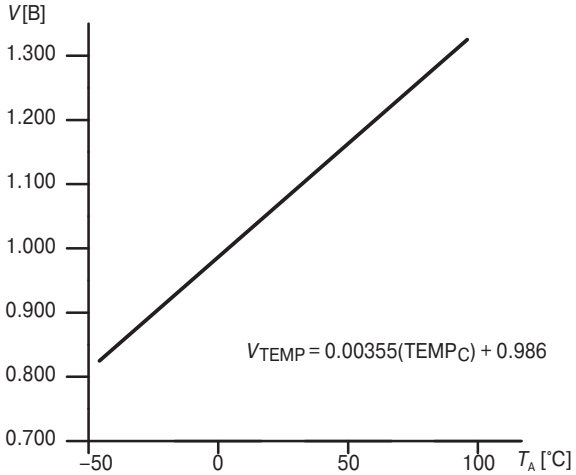


Рис. 21.10. Типичная передаточная функция встроенного датчика температуры.

При выборе датчика температуры встроенный генератор опорного напряжения автоматически задействуется в качестве источника напряжения для датчика. Однако выход V_{REF+} при этом не включается и настройки источника опорного напряжения для преобразования не изменяются. Выбор опорного напряжения для преобразования сигнала от датчика температуры производится так же, как и для любого другого канала преобразования.

21.2.8. Заземление и борьба с помехами при использовании модуля ADC12

При использовании модуля ADC12, как и в случае любого другого АЦП с высокой разрешающей способностью, необходимо следовать определённым рекомендациям по разводке печатной платы и организации заземления с тем, чтобы избежать появления паразитных земляных контуров, нежелательных эффектов и шумов.

Паразитные земляные контуры появляются в том случае, если обратный ток от АЦП протекает по проводникам, являющимися общими с другими аналоговыми или цифровыми элементами схемы. Если не принять специальных мер, то этот ток может вызвать появление небольшого напряжения смещения, влияющего на величину опорного напряжения или уровень входного сигнала аналого-цифрового преобразователя. Подключение модуля АЦП, позволяющее этого избежать, показано на **Рис. 21.11**.

Помимо паразитных земляных токов, на результат преобразования также могут оказывать влияние пульсации и выбросы на шинах питания, возникающие

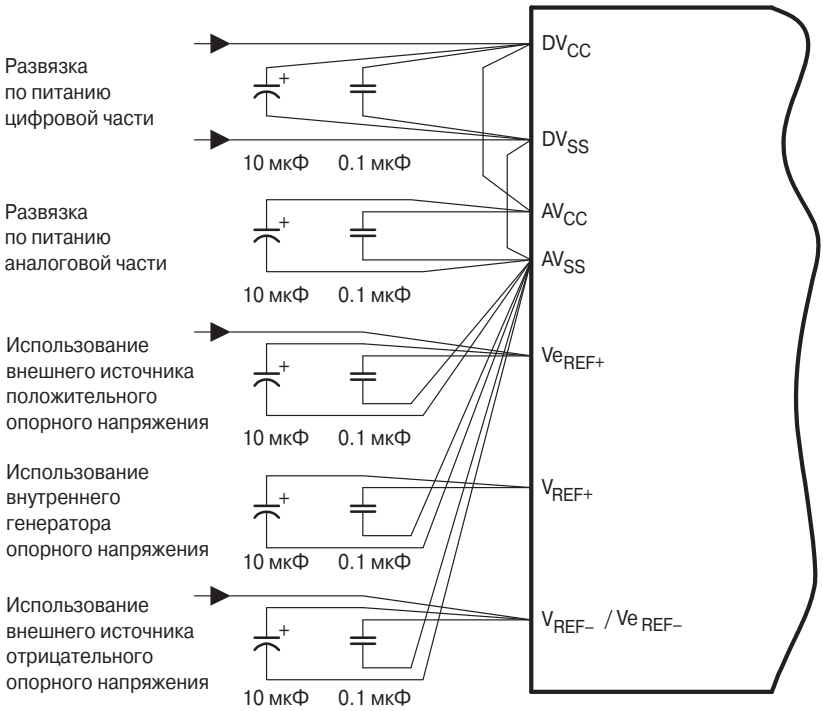


Рис. 21.11. Заземление и борьба с помехами при использовании модуля ADC12.

при работе цифровых узлов или импульсных источников питания. Один из методов разводки печатных плат, позволяющих обеспечить высокую точность преобразований, состоит в использовании раздельных аналоговой и цифровой земляных шин, соединённых в одной точке.

21.2.9. Прерывания модуля ADC12

Модуль ADC12 имеет 18 источников прерываний:

- ADC12IFG0 – ADC12IFG15;
- ADC12OV, переполнение ADC12MEMx;
- ADC12TOV, превышение времени преобразования.

Биты ADC12IFGx устанавливаются при загрузке результата преобразования в соответствующий регистр данных ADC12MEMx. Если соответствующий бит ADC12IEx и бит общего разрешения прерываний GIE установлены, то при установке флага генерируется запрос прерывания. Флаг ADC12OV устанавливается в том случае, если результат преобразования заносится в какой-либо из регистров ADC12MEMx до считывания из него предыдущего результата. Флаг ADC12TOV устанавливается, если очередная операция выборки/преобразования запускается до завершения текущего преобразования. Пересылка с использованием прямого доступа к памяти запускается после завершения преобразования (в одноканальных режимах) или после завершения последовательности в последовательных режимах.

Регистр вектора прерывания ADC12IV

Все источники прерываний модуля ADC12 имеют различные приоритеты и связаны с единственным вектором прерываний. Для определения того, какой из разрешённых источников вызвал генерацию прерывания, используется регистр вектора прерывания ADC12IV.

Разрешённое прерывание с наивысшим приоритетом формирует в регистре ADC12IV число (см. описание регистра). Это число можно оценить или же просто прибавить к счётчику команд для автоматического перехода к соответствующей секции программы. Запрещённые прерывания модуля ADC12 не влияют на содержимое регистра ADC12IV.

При любом обращении к регистру ADC12IV как для чтения, так и для записи флаги ADC12OV и ADC12TOV автоматически сбрасываются, если ожидалась обработка соответствующего прерывания. Флаги ADC12IFGx не сбрасываются при обращении к регистру ADC12IV. Эти флаги сбрасываются автоматически при обращении к соответствующим регистрам ADC12MEMx или же могут быть сброшены программно.

Если имеется ещё один установленный флаг, то сразу же после обработки текущего прерывания будет сгенерировано новое прерывание. Например, если на момент обращения к регистру ADC12IV в процедуре обработки прерывания были установлены флаги ADC12OV и ADC12IFG3, то флаг ADC12OV будет сброшен автоматически. После выполнения команды `RETI` процедуры обработки прерывания, флаг ADC12IFG3 вызовет генерацию нового прерывания.

Пример процедуры обработки прерываний модуля ADC12

В следующем примере показано рекомендуемое использование регистра ADC12IV и приведена информация о накладных расходах на обработку прерывания. Содержимое регистра ADC12IV прибавляется к счётчику команд для автоматического перехода к соответствующей процедуре.

Значения, расположенные по правому краю, показывают число тактов ЦПУ, требуемое для выполнения каждой команды. Накладные расходы на обработку различных источников прерывания включают в себя задержку обслуживания прерывания и время, требуемое для возврата из прерывания, но не время, необходимое для выполнения собственно задачи. Задержки обработки прерывания имеют следующие значения:

- ADC12IFG0...ADC12IFG14, ADC12TOV и ADC12OV — 16 тактов
- ADC12IFG15 — 14 тактов

В обработчике прерывания ADC12IFG15 используется решение, позволяющее, не выходя из процедуры обработки прерывания, проверить, не было ли сгенерировано нового запроса прерывания во время обработки флага ADC15IFG. При наличии нового прерывания, ожидающего обработки, это решение позволяет сэкономить 9 тактов ЦПУ.

```
; Обработчик прерываний модуля ADC12                                Тактов
INT_ADC12              ; Точка входа в обработчик прерывания          6
  ADD   &ADC12IV,PC    ; Прибавляем смещение в таблице переходов      3
  RETI                 ; Вектор 0: Нет прерывания                     5
```

```
  JMP   ADOV          ; Вектор 2: Переполнение АЦП                    2
  JMP   ADTOV         ; Вектор 4: Превышение времени преобразования   2
  JMP   ADM0          ; Вектор 6: ADC12IFG0                           2
     ...              ; Векторы 8-32                                  2
  JMP   ADM14         ; Вектор 34: ADC12IFG14                         2
;
; Обработчик ADC12IFG15 начинается здесь. Команды JMP не требуется
;
ADM15    MOV &ADC12MEM15,xxx ; Копируем результат, флаг сбрасывается
         ...                 ; Что-нибудь ещё?
         JMP INT_ADC12       ; Проверим наличие других прерываний
;
; Обработчики ADC12IFG14-ADC12IFG1 расположены здесь
;
ADM0     MOV &ADC12MEM0,xxx  ; Копируем результат, флаг сбрасывается
         ...                 ; Что-нибудь ещё?
         RETI                ; Возвращаемся                          5
;
ADTOV    ...                 ; Обработаем превышение времени преобразования
         RETI                ; Возвращаемся                          5
;
ADOV     ...                 ; Обработаем переполнение ADCMEMx
         RETI                ; Возвращаемся                          5
```

21.3. Регистры модуля ADC12

Список регистров модуля ADC12 приведён в **Табл. 21.2**.

Таблица 21.2. **Регистры модуля ADC12**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления 0 модуля ADC12	ADC12CTL0	Чтение/запись	01A0h	Сбрасывается после POR
Регистр управления 1 модуля ADC12	ADC12CTL1	Чтение/запись	01A2h	Сбрасывается после POR
Регистр флагов прерываний модуля ADC12	ADC12IFG	Чтение/запись	01A4h	Сбрасывается после POR
Регистр разрешения прерываний модуля ADC12	ADC12IE	Чтение/запись	01A6h	Сбрасывается после POR
Регистр вектора прерываний модуля ADC12	ADC12IV	Чтение	01A8h	Сбрасывается после POR
Регистр данных 0 буфера модуля ADC12	ADC12MEM0	Чтение/запись	0140h	Не изменяется
Регистр данных 1 буфера модуля ADC12	ADC12MEM1	Чтение/запись	0142h	Не изменяется
Регистр данных 2 буфера модуля ADC12	ADC12MEM2	Чтение/запись	0144h	Не изменяется
Регистр данных 3 буфера модуля ADC12	ADC12MEM3	Чтение/запись	0146h	Не изменяется
Регистр данных 4 буфера модуля ADC12	ADC12MEM4	Чтение/запись	0148h	Не изменяется
Регистр данных 5 буфера модуля ADC12	ADC12MEM5	Чтение/запись	014Ah	Не изменяется

Таблица 21.2. **Регистры модуля ADC12** *(продолжение)*

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр данных 6 буфера модуля ADC12	ADC12MEM6	Чтение/запись	014Ch	Не изменяется
Регистр данных 7 буфера модуля ADC12	ADC12MEM7	Чтение/запись	014Eh	Не изменяется
Регистр данных 8 буфера модуля ADC12	ADC12MEM8	Чтение/запись	0150h	Не изменяется
Регистр данных 9 буфера модуля ADC12	ADC12MEM9	Чтение/запись	0152h	Не изменяется
Регистр данных 10 буфера модуля ADC12	ADC12MEM10	Чтение/запись	0154h	Не изменяется
Регистр данных 11 буфера модуля ADC12	ADC12MEM11	Чтение/запись	0156h	Не изменяется
Регистр данных 12 буфера модуля ADC12	ADC12MEM12	Чтение/запись	0158h	Не изменяется
Регистр данных 13 буфера модуля ADC12	ADC12MEM13	Чтение/запись	015Ah	Не изменяется
Регистр данных 14 буфера модуля ADC12	ADC12MEM14	Чтение/запись	015Ch	Не изменяется
Регистр данных 15 буфера модуля ADC12	ADC12MEM15	Чтение/запись	015Eh	Не изменяется
Регистр управления 0 буфера модуля ADC12	ADC12MCTL0	Чтение/запись	080h	Сбрасывается после POR
Регистр управления 1 буфера модуля ADC12	ADC12MCTL1	Чтение/запись	081h	Сбрасывается после POR
Регистр управления 2 буфера модуля ADC12	ADC12MCTL2	Чтение/запись	082h	Сбрасывается после POR
Регистр управления 3 буфера модуля ADC12	ADC12MCTL3	Чтение/запись	083h	Сбрасывается после POR
Регистр управления 4 буфера модуля ADC12	ADC12MCTL4	Чтение/запись	084h	Сбрасывается после POR
Регистр управления 5 буфера модуля ADC12	ADC12MCTL5	Чтение/запись	085h	Сбрасывается после POR
Регистр управления 6 буфера модуля ADC12	ADC12MCTL6	Чтение/запись	086h	Сбрасывается после POR
Регистр управления 7 буфера модуля ADC12	ADC12MCTL7	Чтение/запись	087h	Сбрасывается после POR
Регистр управления 8 буфера модуля ADC12	ADC12MCTL8	Чтение/запись	088h	Сбрасывается после POR
Регистр управления 9 буфера модуля ADC12	ADC12MCTL9	Чтение/запись	089h	Сбрасывается после POR
Регистр управления 10 буфера модуля ADC12	ADC12MCTL10	Чтение/запись	08Ah	Сбрасывается после POR
Регистр управления 11 буфера модуля ADC12	ADC12MCTL11	Чтение/запись	08Bh	Сбрасывается после POR
Регистр управления 12 буфера модуля ADC12	ADC12MCTL12	Чтение/запись	08Ch	Сбрасывается после POR
Регистр управления 13 буфера модуля ADC12	ADC12MCTL13	Чтение/запись	08Dh	Сбрасывается после POR
Регистр управления 14 буфера модуля ADC12	ADC12MCTL14	Чтение/запись	08Eh	Сбрасывается после POR
Регистр управления 15 буфера модуля ADC12	ADC12MCTL15	Чтение/запись	08Fh	Сбрасывается после POR

ADC12CTL0, регистр управления 0 модуля ADC12

15	14	13	12	11	10	9	8
SHT1x				SHT0x			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
MSC	REF2_5V	REFON	ADC12ON	ADC12OVIE	ADC12 TOVIE	ENC	ADC12SC
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Может быть изменён только при ENC = 0.

SHT1x Биты 15...12 Время выборки. Эти биты определяют время выборки для регистров с ADC12MEM8 по ADC12MEM15.

SHT0x Биты 11...8 Время выборки. Эти биты определяют время выборки для регистров с ADC12MEM0 по ADC12MEM7.

Биты SHTx	Количество тактов ADC12CLK
0000	4
0001	8
0010	16
0011	32
0100	64
0101	96
0110	128
0111	192
1000	256
1001	384
1010	512
1011	768
1100	1024
1101	1024
1110	1024
1111	1024

MSC Бит 7 Многократные выборки/преобразования. Используется только в последовательном или циклических режимах.

0 Каждая выборка/преобразование инициируется нарастающим фронтом сигнала SHI

1 Таймер выборки запускается по первому нарастающему фронту сигнала SHI, а остальные выборки/преобразования запускаются автоматически после завершения предшествующего преобразования

REF2_5V Бит 6 Величина внутреннего опорного напряжения. Также должен быть установлен бит REFON.

0 1.5 В

1 2.5 В

REFON Бит 5 Включение генератора опорного напряжения.
0 Генератор опорного напряжения выключен
1 Генератор опорного напряжения включён

ADC12ON Бит 4 Включение модуля ADC12.
0 Модуль ADC12 выключен
1 Модуль ADC12 включён

ADC12OVIE Бит 3 Разрешение прерывания по переполнению. Для разрешения прерывания также должен быть установлен бит GIE.
0 Прерывание запрещено
1 Прерывание разрешено

ADC12 TOVIE Бит 2 Разрешение прерывания по превышению времени преобразования. Для разрешения прерывания также должен быть установлен бит GIE.
0 Прерывание запрещено
1 Прерывание разрешено

ENC Бит 1 Разрешение преобразования.
0 Работа ADC12 запрещена
1 Работа ADC12 разрешена

ADC12SC Бит 0 Запуск выборки/преобразования. Этот бит используется для программного запуска процесса выборки и преобразования. Биты ADC12SC и ENC могут быть установлены одной командой. Бит ADC12SC сбрасывается автоматически.
0 Не начинать выборку/преобразование
1 Начать процесс выборки/преобразования

ADC12CTL1, регистр управления 1 модуля ADC12

15	14	13	12	11	10	9	8
CSTARTADDx				**SHSx**		**SHP**	**ISSH**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
ADC12DIVx			**ADC12SSELx**		**CONSEQx**		**ADC12 BUSY**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	r–(0)

Может быть изменён только при ENC = 0.

CSTARTADDx Биты 15...12 Начальный адрес преобразования. Эти биты определяют номер ячейки буфера для однократного преобразования или номер 1-й ячейки для последовательности преобразований. В битах CSTARTADDx могут храниться значения от 0 до 0Fh, соответствующие регистрам с ADC12MEM0 по ADC12MEM15.

SHSx Биты 11...10 Источник сигнала запуска выборки/преобразования.
00 Бит ADC12SC
01 Модуль вывода 1 Таймера A
10 Модуль вывода 0 Таймера B
11 Модуль вывода 1 Таймера B

SHP	Бит 9	Выбор импульсного режима работы устройства выборки/хранения. Этот бит определяет источник сигнала выборки SAMPCON — выход таймера выборки или непосредственно сигнал выборки. 0 Сигнал SAMPCON формируется из входного сигнала выборки 1 Сигнал SAMPCON формируется таймером выборки
ISSH	Бит 8	Инвертирование сигнала запуска выборки/преобразования. 0 Сигнал запуска выборки/преобразования не инвертируется 1 Сигнал запуска выборки/преобразования инвертируется
ADC12DIVx	Биты 7...5	Коэффициент деления тактового сигнала модуля ADC12. 000 1 001 2 010 3 011 4 100 5 101 6 110 7 111 8
ADC12SSELx	Биты 4...3	Выбор источника тактового сигнала модуля ADC12. 00 ADC12SC 01 ACLK 10 MCLK 11 SMCLK
CONSEQx	Биты 2...1	Выбор режима преобразования. 00 Однократный одноканальный 01 Однократный последовательный 10 Циклический одноканальный 11 Циклический последовательный
ADC12BUSY	Бит 0	Модуль ADC12 занят. Этот бит показывает активность операций выборки или преобразования. 0 Нет активности 1 Производится выборка, преобразование или выполняется последовательность преобразований

ADC12MEMx, регистр данных буфера модуля ADC12

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**Результат преобразования**			
r0	r0	r0	r0	rw	rw	rw	rw

7	6	5	4	3	2	1	0
Результат преобразования							
rw	rw	rw	rw	rw	rw	rw	rw

Результат преобразования	Биты 15...0	12-битный результат преобразования выровнен по правому краю. Бит 11 — MSB. Биты 15...12 всегда равны 0. Запись в регистры памяти преобразований вызовет порчу результата.

ADC12MCTLx, регистр управления памяти преобразований модуля ADC12

7	6	5	4	3	2	1	0
EOS	**SREFx**			**INCHx**			
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Может быть изменён только при ENC = 0.

EOS Бит 7 Конец последовательности. Этот бит помечает последнее преобразование в последовательности.
0 Не конец последовательности
1 Конец последовательности

SREFx Биты 6...4 Выбор источника опорного напряжения.
000 $V_{R+} = AV_{CC}$, $V_{R-} = AV_{SS}$
001 $V_{R+} = V_{REF+}$, $V_{R-} = AV_{SS}$
010 $V_{R+} = Ve_{REF+}$, $V_{R-} = AV_{SS}$
011 $V_{R+} = Ve_{REF+}$, $V_{R-} = AV_{SS}$
100 $V_{R+} = AV_{CC}$, $V_{R-} = V_{REF-}/Ve_{REF-}$
101 $V_{R+} = V_{REF+}$, $V_{R-} = V_{REF-}/Ve_{REF-}$
110 $V_{R+} = Ve_{REF+}$, $V_{R-} = V_{REF-}/Ve_{REF-}$
111 $V_{R+} = Ve_{REF+}$, $V_{R-} = V_{REF-}/Ve_{REF-}$

INCHx Биты 3...0 Выбор входного канала. Эти биты определяют конкретный канал для однократных преобразований или канал с максимальным номером для последовательных преобразований.
0000 A0
0001 A1
0010 A2
0011 A3
0100 A4
0101 A5
0110 A6
0111 A7
1000 Ve_{REF+}
1001 V_{REF-}/Ve_{REF-}
1010 Датчик температуры
1011 $(AV_{CC} - AV_{SS})/2$
1100 GND
1101 GND
1110 GND
1111 GND

ADC12IE, регистр разрешения прерываний модуля ADC12

15	14	13	12	11	10	9	8
ADC12IE15	**ADC12IE14**	**ADC12IE13**	**ADC12IE12**	**ADC12IE11**	**ADC12IE10**	**ADC12IE9**	**ADC12IE8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

7	6	5	4	3	2	1	0
ADC12IE7	**ADC12IE6**	**ADC12IE5**	**ADC12IE4**	**ADC12IE3**	**ADC12IE2**	**ADC12IE1**	**ADC12IE0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IEx Биты 15...0 Разрешение прерывания. Эти биты разрешают или запрещают генерацию запроса прерывания при установке соответствующего флага ADC12IFGx.
0 Прерывание запрещено
1 Прерывание разрешено

ADC12IFG, регистр флагов прерываний модуля ADC12

15	14	13	12	11	10	9	8
ADC12 IFG15	**ADC12 IFG14**	**ADC12 IFG13**	**ADC12 IFG12**	**ADC12 IFG11**	**ADC12 IFG10**	**ADC12 IFG9**	**ADC12 IFG8**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)
7	6	5	4	3	2	1	0
ADC12 IFG7	**ADC12 IFG6**	**ADC12 IFG5**	**ADC12 IFG4**	**ADC12 IFG3**	**ADC12 IFG2**	**ADC12 IFG1**	**ADC12 IFG0**
rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)	rw-(0)

ADC12IFGx Биты 15...0 Флаг прерывания регистра ADC12MEMx. Эти биты устанавливаются при загрузке результата преобразования в соответствующий регистр ADC12MEMx. Биты ADC12IFGx сбрасываются автоматически при любом обращении к соответствующему регистру ADC12MEMx или программно.

0 Не было запроса прерывания
1 Есть запрос прерывания

ADC12IFG, регистр флагов прерываний модуля ADC12

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0
7	6	5	4	3	2	1	0
0	**0**	**ADC12IVx**					**0**
r0	r0	r-(0)	r-(0)	r-(0)	r-(0)	r-(0)	r0

ADC12IVx Биты 15...0 Значение вектора прерывания модуля ADC12

Содержимое ADC12IV	Источник прерывания	Флаг прерывания	Приоритет прерывания
00h	Нет прерывания	—	
02h	Переполнение ADC12MEMx	—	Высший
04h	Превышение времени преобразования	—	
06h	Флаг прерывания ADC12MEM0	ADC12IFG0	
08h	Флаг прерывания ADC12MEM1	ADC12IFG1	
0Ah	Флаг прерывания ADC12MEM2	ADC12IFG2	
0Ch	Флаг прерывания ADC12MEM3	ADC12IFG3	
0Eh	Флаг прерывания ADC12MEM4	ADC12IFG4	
010h	Флаг прерывания ADC12MEM5	ADC12IFG5	
012h	Флаг прерывания ADC12MEM6	ADC12IFG6	
014h	Флаг прерывания ADC12MEM7	ADC12IFG7	
016h	Флаг прерывания ADC12MEM8	ADC12IFG8	
018h	Флаг прерывания ADC12MEM9	ADC12IFG9	
01Ah	Флаг прерывания ADC12MEM10	ADC12IFG10	
01Ch	Флаг прерывания ADC12MEM11	ADC12IFG11	
01Eh	Флаг прерывания ADC12MEM12	ADC12IFG12	
020h	Флаг прерывания ADC12MEM13	ADC12IFG13	
022h	Флаг прерывания ADC12MEM14	ADC12IFG14	
024h	Флаг прерывания ADC12MEM15	ADC12IFG15	Низший

ГЛАВА 22

СТРУКТУРА TLV

Структуры TLV (Tag-Length-Value — тег-длина-значение) используются в микроконтроллерах семейства MSP430x2xx для представления информации, зависящей от устройства, такой как калибровочные значения. Эти структуры располагаются в сегменте A информационной секции флэш-памяти. Аппаратно-зависимые реализации данных структур описываются в справочной документации на конкретные модели.

22.1. Введение

Структуры TLV описывают специфическую для данного устройства информацию, хранящуюся в сегменте A. В качестве примера в **Табл. 22.1** показано содержимое сегмента A некоего условного устройства.

В двух первых байтах сегмента A (0x10C0 и 0x10C1) содержится контрольная сумма остальной части сегмента (адреса с 0x10C2 по 0x10FF).

Первый тег располагается по адресу 0x10C2 — в данном примере это тег TAG_EMPTY. В следующем байте (0x10C3) содержится длина последующего поля данных. Длина структуры TAG_EMPTY равна 0x16, поэтому следующий тег, TAG_ADC12_1, располагается по адресу 0x10DA. Аналогично, в следующем байте содержится длина поля данных структуры TAG_ADC12_1.

Структуры TLV занимают всё адресное пространство сегмента A от 0x10C2 до 0x10FF. Пользовательская программа, проверяющая наличие тегов начиная с первого адреса сегмента (0x10C2), может извлечь всю требуемую информацию даже в том случае, если эта информация расположена по другим (зависящим от устройства) абсолютным адресам.

22.2. Поддерживаемые теги

Каждое устройство поддерживает определённые теги, перечисленные в **Табл. 22.2**. Более подробная информация приводится в справочной документации на конкретные модели.

Таблица 22.1. **Пример содержимого сегмента A**

Адрес слова	Старший байт	Младший байт	Адрес тега и смещение
0x10FE	CALBC1_1MHZ	CALDCO1_1MHZ	0x10F6 + 0x0008
0x10FC	CALBC1_8MHZ	CALDCO1_8MHZ	0x10F6 + 0x0006
0x10FA	CALBC1_12MHZ	CALDCO1_12MHZ	0x10F6 + 0x0004
0x10F8	CALBC1_16MHZ	CALDCO1_16MHZ	0x10F6 + 0x0002
0x10F6	0x08 (ДЛИНА)	TAG_DCO_30	0x10F6
0x10F4	0xFF	0xFF	
0x10F2	0xFF	0xFF	
0x10F0	0xFF	0xFF	
0x10EE	0xFF	0xFF	
0x10EC	0x08 (ДЛИНА)	TAG_EMPTY	0x10EC
0x10EA	CAL_ADC_25T85		0x10DA + 0x0010
0x10E8	CAL_ADC_25T30		0x10DA + 0x000E
0x10E6	CAL_ADC_25VREF_FACTOR		0x10DA + 0x000C
0x10E4	CAL_ADC_15T85		0x10DA + 0x000A
0x10E2	CAL_ADC_15T30		0x10DA + 0x0008
0x10E0	CAL_ADC_15VREF_FACTOR		0x10DA + 0x0006
0x10DE	CAL_ADC_OFFSET		0x10DA + 0x0004
0x10DC	CAL_ADC_GAIN_FACTOR		0x10DA + 0x0002
0x10DA	0x10 (ДЛИНА)	TAG_ADC12_1	0x10DA
0x10D8	0xFF	0xFF	
0x10D6	0xFF	0xFF	
0x10D4	0xFF	0xFF	
0x10D2	0xFF	0xFF	
0x10D0	0xFF	0xFF	
0x10CE	0xFF	0xFF	
0x10CC	0xFF	0xFF	
0x10CA	0xFF	0xFF	
0x10C8	0xFF	0xFF	
0x10C6	0xFF	0xFF	
0x10C4	0xFF	0xFF	
0x10C2	0x16 (ДЛИНА)	TAG_EMPTY	0x10C2
0x10C0	Результат побитовой операции «Исключающее ИЛИ», представленный в дополнительном коде		0x10C0

Таблица 22.2. **Поддерживаемые теги (зависит от устройства)**

Тег	Описание	Значение
TAG_EMPTY	Неиспользуемая область памяти	0xFE
TAG_DCO_30	Калибровочные значения DCO для комнатной температуры и DV_{CC} = 3 В	0x01
TAG_ADC12_1	Калибровочные значения для модуля ADC12	0x08

22.2.1. Структура TLV калибровочных значений DCO

Для калибровки генератора DCO используются регистры BCSCTL1 и DCOCTL модуля синхронизации BCS+. Значения, хранящиеся сегменте A информационной секции флэш-памяти (**Табл. 22.3**), загружаются в указанные регистры модуля BSC+.

Таблица 22.3. **Данные для калибровки DCO (зависит от устройства)**

Метка	Описание	Смещение
CALBC1_1MHZ	Содержимое регистра BCSCTL1 для 1 МГц, $T_A = 25°C$	0x07
CALDCO_1MHZ	Содержимое регистра DCOCTL для 1 МГц, $T_A = 25°C$	0x06
CALBC1_8MHZ	Содержимое регистра BCSCTL1 для 8 МГц, $T_A = 25°C$	0x05
CALDCO_8MHZ	Содержимое регистра DCOCTL для 8 МГц, $T_A = 25°C$	0x04
CALBC1_12MHZ	Содержимое регистра BCSCTL1 для 12 МГц, $T_A = 25°C$	0x03
CALDCO_12MHZ	Содержимое регистра DCOCTL для 12 МГц, $T_A = 25°C$	0x02
CALBC1_16MHZ	Содержимое регистра BCSCTL1 для 16 МГц, $T_A = 25°C$	0x01
CALDCO_16MHZ	Содержимое регистра DCOCTL для 16 МГц, $T_A = 25°C$	0x00

Использование абсолютного режима адресации

Данные для калибровки DCO содержатся во всех моделях семейства 2xx и хранятся по одним и тем же абсолютным адресам. Поэтому для обращения к содержимому сегмента A в конкретном устройстве можно использовать абсолютную адресацию, как показано в следующем примере.

```
; Калибровка DCO на частоту 1 МГц
   CLR.B &DCOCTL                  ; Выбираем наименьшие значения
                                  ; для битов DCOx и MODx
   MOV.B &CALBC1_1MHZ,&BCSCTL1    ; Задаём RSELx
   MOV.B &CALDCO_1MHZ,&DCOCTL     ; Задаём DCOx и MODx
```

Использование структуры TLV

Структура TLV позволяет обращаться к содержимому флэш-памяти, используя адрес тега TAG_DCO_30. В следующем фрагменте показан пример адресации калибровочных значений DCO с использованием тега TAG_DCO_30.

```
; Калибровка DCO на частоту 8 МГц
; Полагаем, что в регистре R10 содержится адрес тега TAG_DCO_30
  CLR.B &DCOCTL            ; Выбираем наименьшие значения
                           ; для битов DCOx и MODx
  MOV.B 7(R10),&BCSCTL1    ; Задаём RSELx
  MOV.B 6(R10),&DCOCTL     ; Задаём DCOx и MODx
```

22.2.2. Структура TLV калибровочных значений модуля ADC12

Данные для калибровки модуля ADC12 (**Табл. 22.4**) представляют собой набор из восьми двухбайтных значений.

Таблица 22.4. **Данные для калибровки модуля ADC12 (зависит от устройства)**

Метка	Описание	Смещение
CAL_ADC_25T85	VREF2_5 = 1, T_A = +85 ±2°C, 12-битный результат преобразования	0x0E
CAL_ADC_25T30	VREF2_5 = 1, T_A = +30 ±2°C, 12-битный результат преобразования	0x0C
CAL_ADC_25VREF_FACTOR	VREF2_5 = 1, T_A = +30 ±2°C	0x0A
CAL_ADC_15T85	VREF2_5 = 0, T_A = +85 ±2°C, 12-битный результат преобразования	0x08
CAL_ADC_15T30	VREF2_5 = 0, T_A = +30 ±2°C, 12-битный результат преобразования	0x06
CAL_ADC_15VREF_FACTOR	VREF2_5 = 0, T_A = +30 ±2°C	0x04
CAL_ADC_OFFSET	Ve_{REF} = 2.5 В, T_A = +85 ±2°C, $f_{ADC12CLK}$ = 5 МГц	0x02
CAL_ADC_GAIN_FACTOR	Ve_{REF} = 2.5 В, T_A = +85 ±2°C, $f_{ADC12CLK}$ = 5 МГц	0x00

Данные для калибровки датчика температуры

Датчик температуры калибруется с использованием встроенного источника опорного напряжения. Результаты преобразования для температур +30°C и +85°C при разных значениях бита VREF2_5 заносятся в соответствующие ячейки сегмента A (см. **Табл. 22.4**).

Данные для калибровки встроенного источника опорного напряжения

Опорные напряжения (VREF2_5 = 0 и 1) измеряются при комнатной температуре. Измеренное значение перед записью в сегмент A флэш-памяти нормализуется относительно уровня 1.5/2.5 В:

$$\text{CAL_ADC_15VREF_FACTOR} = \frac{Ve_{\text{REF}}}{1.5\ \text{В}} \times 2^{15}.$$

Корректировка результата преобразования производится его умножением на коэффициент CAL_ADC_15VREF_FACTOR (или CAL_ADC_25VREF_FACTOR) с последующим делением на 2^{15}:

$$\text{ADC}_{\text{коррект}} = \text{ADC}_{\text{исх}} \times \text{CAL_ADC_15VREF_FACTOR} \times \frac{1}{2^{15}}.$$

Пример использования калибровки источника опорного напряжения

В следующем примере при выполнении преобразования используется внутреннее опорное напряжение 1.5 В.

1. Результат преобразования: 0x0100.
2. Калибровочный коэффициент опорного напряжения: 0x7BBB.

Для корректирования результата преобразования модуля ADC12 можно воспользоваться аппаратным умножителем:

1. Умножить результат преобразования на 2 (эта операция упростит последующее деление).

2. Умножить результат на величину CAL_ADC_15VREF_FACTOR.
3. Разделить результат на 2^{16} (взять старшее слово 32-битного результата умножения RESHI).

В нашем примере:

1. 0x0100 × 0x0002 = 0x0200
2. 0x0200 × 0x7BBB = 0x00F7_7600
3. 0x00F7_7600 ÷ 0x0001_0000 = 0x0000_00F7 (= 247)

Ниже приведён фрагмент программы, выполняющий описанные действия с использованием аппаратного умножителя.

```
; Результат АЦП находится в регистре ADC12MEM0
; Предполагается, что R9 содержит адрес TAG_ADC12_1.
; Скорректированное значение заносится в ячейку ADC_COR
  MOV.W &ADC12MEM0,R10    ; Копируем результат в R10
  RLA.W R10               ; R10 x 2
  MOV.W R10,&MPY          ; 1-й операнд, беззнаковое умножение
  MOV.W CAL_ADC_15VREF_FACTOR(R9),&OP2
                          ; 2-й операнд — калибровочное значение
  MOV.W &RESHI,&ADC_COR   ; Результат: старшие 16 бит регистра MPY
```

Данные для калибровки смещения и коэффициента усиления

Значение смещения модуля ADC12 хранится в сегменте A в виде 2-байтного значения, представленного в дополнительном коде. Ошибка смещения корректируется сложением результата преобразования со значением CAL_ADC_OFFSET:

$$\mathrm{ADC}_{\text{коррект}} = \mathrm{ADC}_{\text{исх}} + \mathrm{CAL_ADC_OFFSET}.$$

Коэффициент усиления модуля ADC12, расположенный по смещению 0x00, вычисляется следующим образом:

$$\mathrm{CAL_ADC_GAIN_FACTOR} = \frac{1}{\mathrm{GAIN}} \times 2^{15}.$$

Корректирование результата преобразования по усилению производится его умножением на значение CAL_ADC_GAIN_FACTOR и последующим делением на 2^{15}:

$$\mathrm{ADC}_{\text{коррект}} = \mathrm{ADC}_{\text{исх}} \times \mathrm{CAL_ADC_GAIN_FACTOR} \times \frac{1}{2^{15}}.$$

Если результат преобразования корректируется по обоим параметрам, то первой осуществляется корректировка усиления:

$$\mathrm{ADC}_{\text{коррект, усил}} = \mathrm{ADC}_{\text{исх}} \times \mathrm{CAL_ADC_GAIN_FACTOR} \times \frac{1}{2^{15}};$$

$$\mathrm{ADC}_{\text{итог}} = \mathrm{ADC}_{\text{коррект, усил}} + \mathrm{CAL_ADC_OFFSET}.$$

Пример использования калибровки смещения и усиления

В следующем примере при выполнении преобразования используется внешний источник опорного напряжения.

- Результат преобразования: 0x0800 (= 2048).
- Калибровочный коэффициент по усилению: 0x7BBB.
- Смещение: 0xFFFE (–2 в дополнительном коде).

Для корректирования результата преобразования модуля ADC12 можно воспользоваться аппаратным умножителем:

1. Умножить результат преобразования на 2 (эта операция упростит последующее деление).
2. Умножить результат на величину CAL_ADC_GAIN_FACTOR.
3. Разделить результат на 2^{16} (взять старшее слово 32-битного результата умножения RESHI).
4. Прибавить к результату значение CAL_ADC_OFFSET.

В нашем примере:

1. 0x0800 × 0x0002 = 0x1000
2. 0x1000 × 0x8010 = 0x0801_0000
3. 0x0801_0000 ÷ 0x0001_0000 = 0x0000_0801 (= 2049)
4. 0x801 + 0xFFFE = 0x07FF (= 2047)

Ниже приведён фрагмент программы, выполняющий описанные действия с использованием аппаратного умножителя.

```
; Результат АЦП находится в регистре ADC12MEM0
; Предполагается, что R9 содержит адрес TAG_ADC12_1.
; Скорректированное значение заносится в ячейку ADC_COR
     MOV.W &ADC12MEM0,R10  ; Копируем результат в R10
     RLA.W R10             ; R10 x 2
     MOV.W R10,&MPY        ; 1-й операнд, беззнаковое умножение
     MOV.W CAL_ADC_GAIN_FACTOR(R9),&OP2
                           ; 2-й операнд — калибровочное значение
     MOV.W &RESHI,&ADC_COR ; Используем старшие 16 бит регистра MPY
     ADD.W CAL_ADC_OFFSET(R9),&ADC_COR
                           ; Корректируем смещение
```

22.3. Проверка целостности содержимого сегмента A

По адресам 0x10C0 и 0x10C1 расположена 2-байтная контрольная сумма содержимого сегмента A с адреса 0x10C2 по 0x10FF. Контрольная сумма представляет собой результат побитовой операции «Исключающее ИЛИ» между остальными словами сегмента, представленный в дополнительном коде.

Ниже приведён фрагмент кода, в котором вычисляется контрольная сумма сегмента A.

```
; Проверка целостности содержимого сегмента A флэш-памяти
; путём вычисления обратного кода слов, расположенных по
; адресам 0x10C2 - 0x10FE
```

```
; Предполагается, что начальный адрес сегмента A находится
; в R10. Регистр R11 инициализирован нулевым значением.
; Метка TLV_CHKSUM соответствует адресу 0x10C0
      ADD.W #2,R10          ; Пропускаем поле контрольной суммы
LP0   XOR.W @R10+,R11       ; Прибавляем слово к контрольной сумме
      CMP.W #0x10FF,R10     ; Последнее слово?
      JN    LP0             ; Нет, продолжаем
      ADD.W &TLV_CHKSUM,R11 ; Прибавляем контрольную сумму
      JNZ   CSNOK           ; Контрольная сумма не верна
      ...                   ; Используем данные сегмента A
CSNOK ...                   ; Не используем данные сегмента A
```

22.4. Анализ содержимого сегмента A

Ниже приведён пример кода, выполняющего анализ содержимого сегмента A.

```
; Предполагается, что начальный адрес сегмента A находится в R10
LP1   ADD.W #2,R10          ; Пропускаем два байта
      CMP.W #0x10FF,R10     ; Дошли до конца сегмента?
      JGE   DONE            ; Да, переходим к метке DONE
      CMP.B #TAG_EMPTY,0(R10)
                            ; TAG_EMPTY?
      JNZ   T1              ; Нет, продолжаем
      JMP   LP2             ; Да, обрабатываем тег TAG_EMPTY
T1    CMP.B #TAG_ADC12_1,0(R10)
                            ; TAG_ADC12_1?
      JNZ   T2              ; Нет, продолжаем
      ...                   ; Да, найден TAG_ADC12_1
      JMP   LP2             ; Обрабатываем тег TAG_ADC12_1
T2    CMP.B #DCO_30,0(R10)  ; TAG_DCO_30?
      JNZ   T3              ; Нет, продолжаем
      CLR.B &DCOCTL         ; Устанавливаем наименьшее значение DCOx
      MOV.B 7(R10),&BCSCTL1 ; Да, используем данные для 8 МГц
      MOV.B 6(R10),&DCOCTL  ; и задаём DCOx и MODx
      JMP   LP2             ; Обрабатываем тег TAG_DCO_30
T3    ...                   ; Проверяем на следующий тег
      ...                   ;
      JMP   LP2             ; Обрабатываем следующий тег
LP2   MOV.B 1(R10),R11      ; Запоминаем длину в R11
      ADD.W R11,R10         ; Прибавляем длину к R10
      JMP   LP1             ; Продолжаем анализ
DONE                        ;
```

ГЛАВА 23

МОДУЛЬ 12-БИТНОГО ЦАП DAC12

Модуль DAC12 представляет собой 12-битный цифро-аналоговый преобразователь с потенциальным выходом. В этой главе описывается работа модуля ADC12, реализованного в микроконтроллерах семейства MSP430x2xx.

23.1. Введение

Модуль DAC12 представляет собой 12-битный ЦАП с потенциальным выходом. Модуль DAC12 может работать в 8- или 12-битном режимах и использоваться совместно с контроллером DMA. При наличии в устройстве нескольких модулей, они могут быть сгруппированы для синхронного выполнения операции обновления.

Модуль DAC12 имеет следующие особенности:

- монотонная характеристика в пределах 12-битного диапазона;
- потенциальный выход с разрешением 8 или 12 бит;
- программируемое соотношение между временем установления и потреблением;
- внешний или внутренний источник опорного напряжения ;
- использует данные в натуральном двоичном коде или в дополнительном коде;
- возможность самокалибровки для коррекции напряжения смещения;
- возможность синхронного обновления выходов нескольких модулей DAC12.

Примечание. Несколько модулей DAC12

Некоторые модели могут иметь в своём составе более одного модуля DAC12. В этом случае все модули имеют одинаковые характеристики.

В данной главе названия регистров даются в виде DAC12_xDAT или DAC12_xCTL, где x — порядковый номер модуля. Если в названии регистра номер указан, то описание относится к конкретному модулю. В тех случаях, когда модули идентичны, используются обобщённые названия регистров, например DAC12_xCTL.

Блок-схема модуля DAC12 приведена на **Рис. 23.1**.

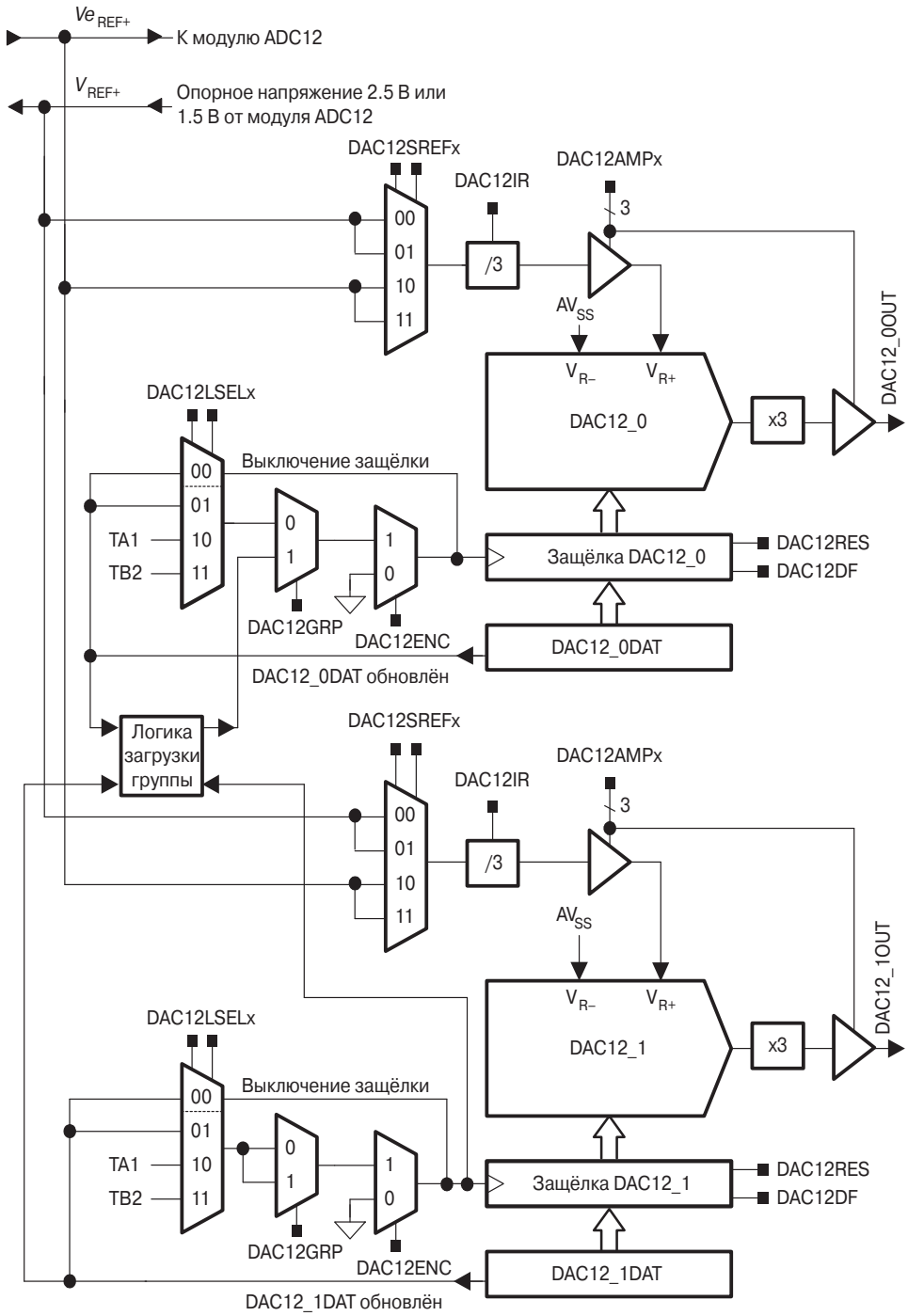


Рис. 23.1. Блок-схема модуля DAC12.

23.2. Функционирование модуля ADC12

Конфигурирование модуля DAC12 осуществляется пользовательской программой. Настройка модуля и его функционирование рассматриваются в следующих подразделах.

23.2.1. Ядро 12-битного ЦАП

С помощью бита DAC12RES модуль DAC12 может быть сконфигурирован для работы в 8- или 12-битном режиме. Максимальный уровень выходного сигнала (однократное или трёхкратное значение выбранного опорного напряжения) определяется битом DAC12IR. Это позволяет пользователю управлять динамическим диапазоном ЦАП. Бит DAC12DF используется для выбора формата входных данных ЦАП — натуральный двоичный код или дополнительный код. Формулы для определения выходного напряжения ЦАП при использовании натурального кода приведены в **Табл. 23.1**.

Таблица 23.1. **Выходной диапазон ЦАП ($V_{ref} = Ve_{REF+}$ или V_{REF+})**

Разрешение	DAC12RES	DAC12IR	Выходное напряжение
12 бит	0	0	$V_{out} = V_{ref} \times 3 \times \frac{DAC12_xDAT}{4096}$
12 бит	0	1	$V_{out} = V_{ref} \times \frac{DAC12_xDAT}{4096}$
8 бит	1	0	$V_{out} = V_{ref} \times 3 \times \frac{DAC12_xDAT}{256}$
8 бит	1	1	$V_{out} = V_{ref} \times \frac{DAC12_xDAT}{256}$

В 8-битном режиме наибольшее используемое значение регистра DAC12_xDAT составляет 0FFh, а в 12-битном режиме — 0FFFh. Значения, превышающие указанные величины, могут быть записаны в регистр, однако все неиспользуемые старшие биты будут проигнорированы.

Выбор порта модуля DAC12

Выходы модулей DAC12 мультиплексированы с выводами порта P6, аналоговыми входами модуля ADC12, а также с выводами Ve_{REF+}. При DAC12AMPx > 0 соответствующий вывод микроконтроллера автоматически подключается к модулю DAC12, независимо от состояния связанных с ним битов PxSELx и PxDIRx. Бит DAC12OPS используется для выбора конкретного вывода, к которому будет подключён выход ЦАП. Так, при DAC12OPS = 0 выход модуля DAC12_0 подключается к выводу порта P6.6, а выход модуля DAC12_1 — к выводу порта P6.7. При DAC12OPS = 1 выход модуля DAC12_0 подключается к выводу Ve_{REF+}, а выход модуля DAC12_1 — к выводу порта P6.5. Для получения дополнительной информации обратитесь к принципиальной схеме вывода порта, которая приводится в справочной документации на конкретный микроконтроллер.

23.2.2. Опорное напряжение модуля DAC12

Модуль DAC12 может использовать либо внешнее опорное напряжение, либо напряжение величиной 1.5 В/2.5 В от внутреннего генератора опорного напряжения модуля ADC12. Выбор источника опорного напряжения осуществляется битами DAC12SREFx. При DAC12SREFx = {0, 1} в качестве опорного используется напряжение V_{REF+}, а при DAC12SREFx = {2, 3} — напряжение Ve_{REF+}.

При использовании внутреннего опорного напряжения необходимо с помощью соответствующих управляющих битов включить и сконфигурировать генератор опорного напряжения модуля ADC12.

Буферы опорного напряжения и выходного напряжения модуля DAC12

Буфер внешнего опорного напряжения и выходной буфер модуля DAC12 могут быть сконфигурированы для выбора оптимального соотношения между временем установления буфера и потребляемым током. С помощью битов DAC12AMPx можно выбрать одну из восьми возможных комбинаций. Чем меньше быстродействие буфера, тем меньше потребляемый им ток. Точные значения указанных параметров приводятся в справочной документации на конкретный микроконтроллер.

23.2.3. Обновление состояния выхода модуля ADC12

Регистр DAC12_xDAT может подключаться к ядру ЦАП напрямую или через регистр-защёлку. Событие, вызывающее обновление выходного сигнала модуля DAC12, определяется битами DAC12LSELx.

При DAC12LSELx = 0 защёлка данных отключается, и регистр DAC12_xDAT подключается напрямую к ядру ЦАП. В этом случае обновление выхода модуля DAC12 происходит сразу же после записи нового значения в регистр DAC12_xDAT, независимо от состояния бита DAC12ENC.

При DAC12LSELx = 1 новое значение модуля DAC12 сохраняется в защёлке и перегружается в ядро ЦАП после записи в регистр DAC12_xDAT очередного значения. При DAC12LSELx = 2 или 3 данные защёлкиваются по нарастающему фронту выходного сигнала блока CCR1 Таймера A или блока CCR2 Таймера B соответственно. Для защёлкивания новых данных при DAC12LSELx > 0 должен быть установлен бит DAC12ENC.

23.2.4. Формат содержимого DAC12_xDAT

Модуль DAC12 поддерживает два формата представления данных: натуральный двоичный код и обратный код. При использовании данных, представленных в натуральном коде, максимальному выходному сигналу в 12-битном режиме соответствует значение 0FFFh (0FFh в 8-битном режиме), как показано на **Рис. 23.2**.

При использовании данных, представленных в дополнительном коде, диапазон сдвигается таким образом, чтобы при значении 0800h регистра DAC12_xDAT (0080h в 8-битном режиме) выходное напряжение ЦАП было равно нулю, при 0000h — половине от максимального значения, а при 07FFh (007Fh в 8-битном режиме) — максимальному значению, как показано на **Рис. 23.3**.

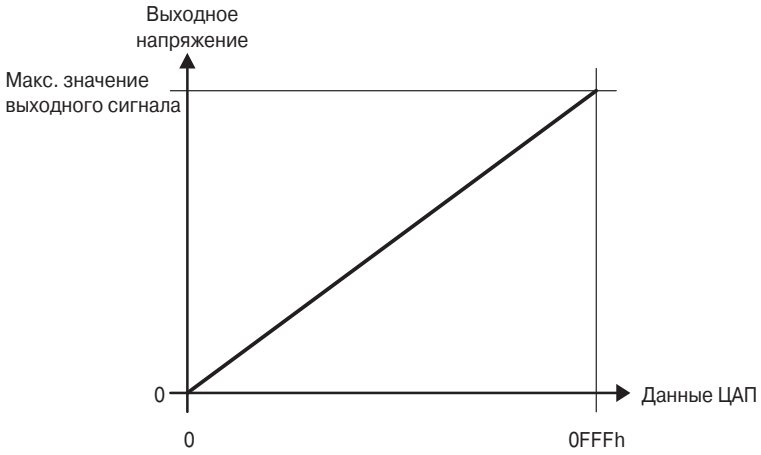


Рис. 23.2. Зависимость выходного напряжения модуля DAC12 от входных данных, представленных в натуральном двоичном коде (12-битный режим).

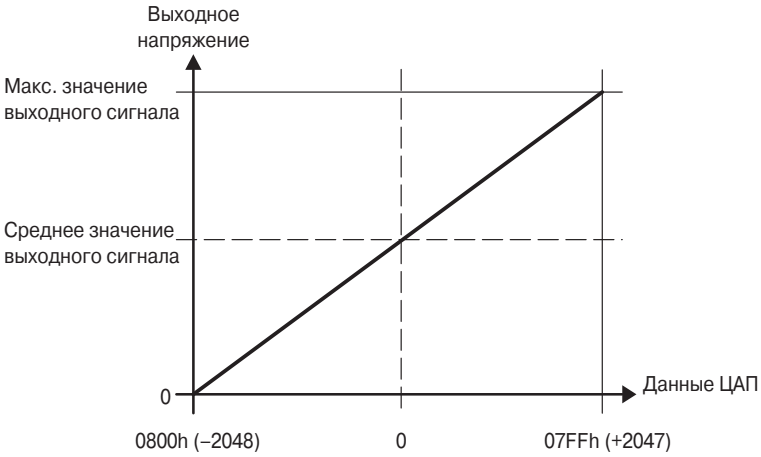


Рис. 23.3. Зависимость выходного напряжения модуля DAC12 от входных данных, представленных в дополнительном коде (12-битный режим).

23.2.5. Калибровка смещения выходного усилителя модуля DAC12

Напряжение смещения выходного усилителя модуля DAC12 может быть как положительным, так и отрицательным. При отрицательном смещении усилитель пытается сформировать отрицательное напряжение, но не может этого сделать. Выходное напряжение остаётся равным нулю до тех пор, пока входной сигнал модуля не вызовет появления положительного выходного напряжения, превыша-

ющего по абсолютной величине напряжение смещения. Получающаяся передаточная функция приведена на **Рис. 23.4**.

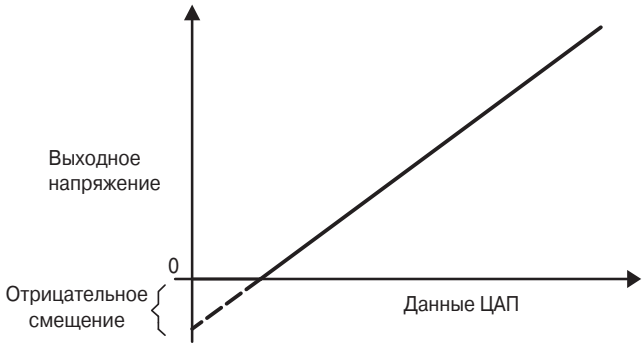


Рис. 23.4. Отрицательное смещение.

При положительном смещении входной сигнал, соответствующий нулевому выходному напряжению, вызывает появление напряжения, отличного от нуля. Соответственно, напряжение на выходе модуля DAC12 достигает максимума раньше, чем на вход модуля будет подано значение, соответствующее максимальному напряжению. Эта ситуация показана на **Рис. 23.5**.

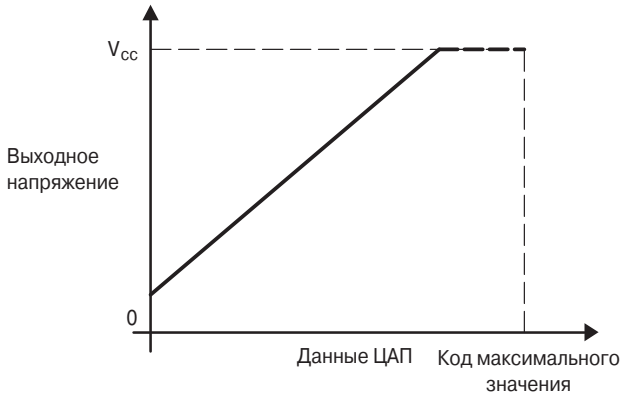


Рис. 23.5. Положительное смещение.

В модуле DAC12 предусмотрена возможность калибровки напряжения смещения выходного усилителя. Процесс калибровки запускается установкой бита DAC12CALON. Калибровка должна быть завершена до начала использования модуля DAC12. После завершения калибровки бит DAC12CALON автоматически сбрасывается. Биты DAC12AMPx должны быть сконфигурированы перед калибровкой. Для достижения наилучших результатов необходимо на время калибровки свести к минимуму активность ЦПУ и портов ввода/вывода.

23.2.6. Группирование нескольких модулей DAC12

Допускается группирование нескольких модулей DAC12 посредством бита DAC12GRP для синхронизации момента обновления выходного сигнала каждого из модулей. При этом гарантируется, что обновление состояния всех модулей группы произойдёт одновременно, вне зависимости от наличия обычных либо немаскируемых прерываний.

Группирование модулей DAC12_0 и DAC12_1 производится установкой бита DAC12GRP модуля DAC12_0. Значение бита DAC12GRP модуля DAC12_1 безразлично. После группирования модулей DAC12_0 и DAC12_1:

- биты DAC12LSELx модуля DAC12_1 определяют событие, инициирующее обновление, для обоих модулей;
- содержимое битов DAC12LSELx обоих модулей должно быть отличным от нуля;
- биты DAC12ENC обоих модулей должны быть установлены в 1.

При сгруппированных модулях DAC12_0 и DAC12_1 перед обновлением выходов необходимо осуществлять запись в оба регистра DAC12_xDAT, даже если входные данные одного или обоих модулей не изменяются. На **Рис. 23.6** приведены временные диаграммы процесса обновления выходов сгруппированных модулей DAC12_0 и DAC12_1.

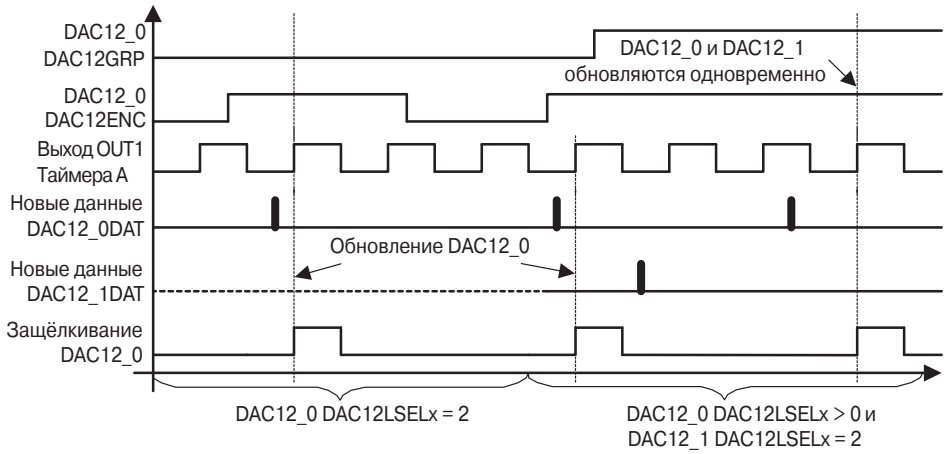


Рис. 23.6. Пример обновления состояния сгруппированных модулей DAC12, запуск по сигналу Таймера A.

Если при установленном бите DAC12GRP модуля DAC12_0 и отличном от нуля содержимом битов DAC12LSELx обоих модулей бит DAC12ENC хотя бы в одном из модулей будет сброшен, то состояние обоих модулей не изменится.

Примечание. Время установления модуля DAC12
Контроллер DMA способен пересылать данные в модуль DAC12 быстрее, чем может изменяться состояние выхода модуля. Поэтому при использовании контроллера DMA пользователь должен гарантировать, что интервал загрузки очередных данных будет не меньше времени установления модуля DAC12. Точное значение этого параметра приводится в справочной документации на конкретную модель.

23.2.7. Прерывания модуля DAC12

В некоторых моделях модуль DAC12 использует тот же вектор прерывания, что и контроллер DMA (см. справочную документацию на микроконтроллер). В этом случае в процедуре обработки прерывания необходимо проверять оба флага DMAIFG и DAC12IFG для определения источника прерывания.

Флаг DAC12IFG устанавливается при DAC12LSELx > 0, когда входное значение модуля DAC12 перегружается из регистра DAC12_xDAT в защёлку данных. При DAC12LSELx = 0 флаг ADC12IFG не устанавливается.

Установленный флаг DAC12IFG означает, что модуль DAC12 готов к получению новых данных. Если установлены оба бита DAC12IE и GIE, то установка флага DAC12IFG генерирует запрос прерывания. Флаг DAC12IFG не сбрасывается автоматически — его необходимо сбрасывать программно.

23.3. Регистры модуля DAC12

Список регистров модуля DAC12 приведён в **Табл. 23.2**.

Таблица 23.2. **Регистры модуля DAC12**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления модуля DAC12_0	DAC12_0CTL	Чтение/запись	01C0h	Сбрасывается после POR
Регистр данных модуля DAC12_0	DAC12_0DAT	Чтение/запись	01C8h	Сбрасывается после POR
Регистр управления модуля DAC12_1	DAC12_1CTL	Чтение/запись	01C2h	Сбрасывается после POR
Регистр данных модуля DAC12_1	DAC12_1DAT	Чтение/запись	01CAh	Сбрасывается после POR

DAC12_xCTL, регистр управления модуля DAC12_x

15	14	13	12	11	10	9	8
DAC12OPS	**DAC12SREFx**		**DAC12RES**	**DAC12LSELx**		**DAC12 CALON**	**DAC12IR**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
DAC12AMPx			**DAC12DF**	**DAC12IE**	**DAC12IFG**	**DAC12ENC**	**DAC12 GRP**
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Может быть изменён только при DAC12ENC = 0.

DAC12OPS Бит 15 Выбор выхода модуля DAC12.
- 0 Выход DAC12_0 — P6.6, выход DAC12_1 — P6.7
- 1 Выход DAC12_0 — Ve_{REF+}, выход DAC12_1 — P6.5

DAC12 SREFx Биты 14...13 Выбор опорного напряжения модуля DAC12.
- 00 V_{REF+}
- 01 V_{REF+}
- 10 Ve_{REF+}
- 11 Ve_{REF+}

DAC12 RES Бит 12 Выбор разрядности модуля DAC12.
- 0 12-битный ЦАП
- 1 8-битный ЦАП

DAC12 LSELx Биты 11...10 Управление загрузкой данных в защёлку DAC12. Эти биты определяют сигнал, по которому производится загрузка данных в защёлку модуля. Если DAC12LSELx > 0, то для обновления состояния ЦАП должен быть установлен бит DAC12ENC.
- 00 Данные загружаются в защёлку при записи в регистр DAC12_xDAT (состояние бита DAC12ENC игнорируется)
- 01 Данные загружаются в защёлку при записи в регистр DAC12_xDAT или, при использовании группирования, после записи в регистры DAC12_xDAT всех модулей группы
- 10 Нарастающий фронт сигнала OUT1 Таймера A (TA1)
- 11 Нарастающий фронт сигнала OUT2 Таймера B (TB2)

DAC12 CALON Бит 9 Включение режима калибровки модуля DAC12. Установка этого бита инициирует процесс калибровки смещения модуля DAC12. Бит DAC12CALON сбрасывается автоматически после завершения калибровки.
- 0 Калибровка не выполняется
- 1 Начать калибровку/выполняется калибровка

DAC12IR Бит 8 Входной диапазон модуля DAC12. Этот бит устанавливает зависимость между величиной опорного напряжения и максимальным выходным сигналом.
- 0 Максимальный выходной сигнал ЦАП равен трёхкратному значению опорного напряжения
- 1 Максимальный выходной сигнал ЦАП равен однократному значению опорного напряжения

DAC12 AMPx Биты 7...5 Установки буферов модуля DAC12. Эти биты определяют соотношение между временем установления и потребляемым током входных и выходных буферов модуля DAC.

DAC12AMPx	Входной буфер	Выходной буфер
000	Выключен	DAC12 выключен, выход в высокоимпедансном состоянии
001	Выключен	DAC12 выключен, на выходе 0 В
010	Низкое быстродействие/ малый ток	Низкое быстродействие/ малый ток
011	Низкое быстродействие/ малый ток	Среднее быстродействие/ средний ток
100	Низкое быстродействие/ малый ток	Высокая скорость/ большой ток
101	Среднее быстродействие/ средний ток	Среднее быстродействие/ средний ток
110	Среднее быстродействие/ средний ток	Высокое быстродействие/ большой ток
111	Высокое быстродействие/ большой ток	Высокое быстродействие/ большой ток

DAC12DF Бит 4 Формат данных модуля DAC12.
0 Натуральный двоичный код
1 Дополнительный код

DAC12IE Бит 3 Разрешение прерывания модуля DAC12.
0 Прерывание запрещено
1 Прерывание разрешено

DAC12IFG Бит 2 Флаг прерывания модуля DAC12.
0 Не было запроса прерывания
1 Есть запрос прерывания

DAC12 ENC Бит 1 Разрешение преобразования. Этот бит разрешает работу модуля DAC12 при DAC12LSESLx > 0. При DAC12LSESLx = 0 бит DAC12ENC игнорируется.
0 Работа DAC12 запрещена
1 Работа DAC12 разрешена

DAC12 GRP Бит 0 Группирование модулей DAC12. Этот бит отвечает за объединение модуля DAC12_x со следующим по порядку модулем. В модуле DAC12_1 не используется.
0 Модули DAC12 не сгруппированы
1 Модули DAC12 сгруппированы

DAC12_xDAT, регистр данных модуля DAC12_x

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**Данные DAC12**			
r(0)	r(0)	r(0)	r(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

7	6	5	4	3	2	1	0
Данные DAC12							
rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)	rw–(0)

Unused Биты 15...12 Не используются. Эти биты всегда равны 0 и не используются ядром ЦАП.

Данные DAC12 Биты 11...0 Входные данные модуля DAC12

Формат данных DAC12	Данные DAC12
12-битный натуральный код	Данные выравниваются по правому краю. Бит 11 – MSB
12-битный дополнительный код	Данные выравниваются по правому краю. Бит 11 – MSB (знак)
8-битный натуральный код	Данные выравниваются по правому краю. Бит 7 – MSB. Состояние битов 11...8 безразлично, они не используются ядром ЦАП
8-битный дополнительный код	Данные выравниваются по правому краю. Бит 7 – MSB (знак). Состояние битов 11...8 безразлично, они не используются ядром ЦАП

ГЛАВА 24

МОДУЛЬ 16-БИТНОГО АЦП SD16_A

Модуль SD16_A представляет собой 16-битный сигма-дельта аналого-цифровой преобразователь, имеющий буфер с высоким входным сопротивлением. В этой главе описывается работа модуля SD16_A, реализованного в микроконтроллерах MSP430x20x3.

24.1. Введение

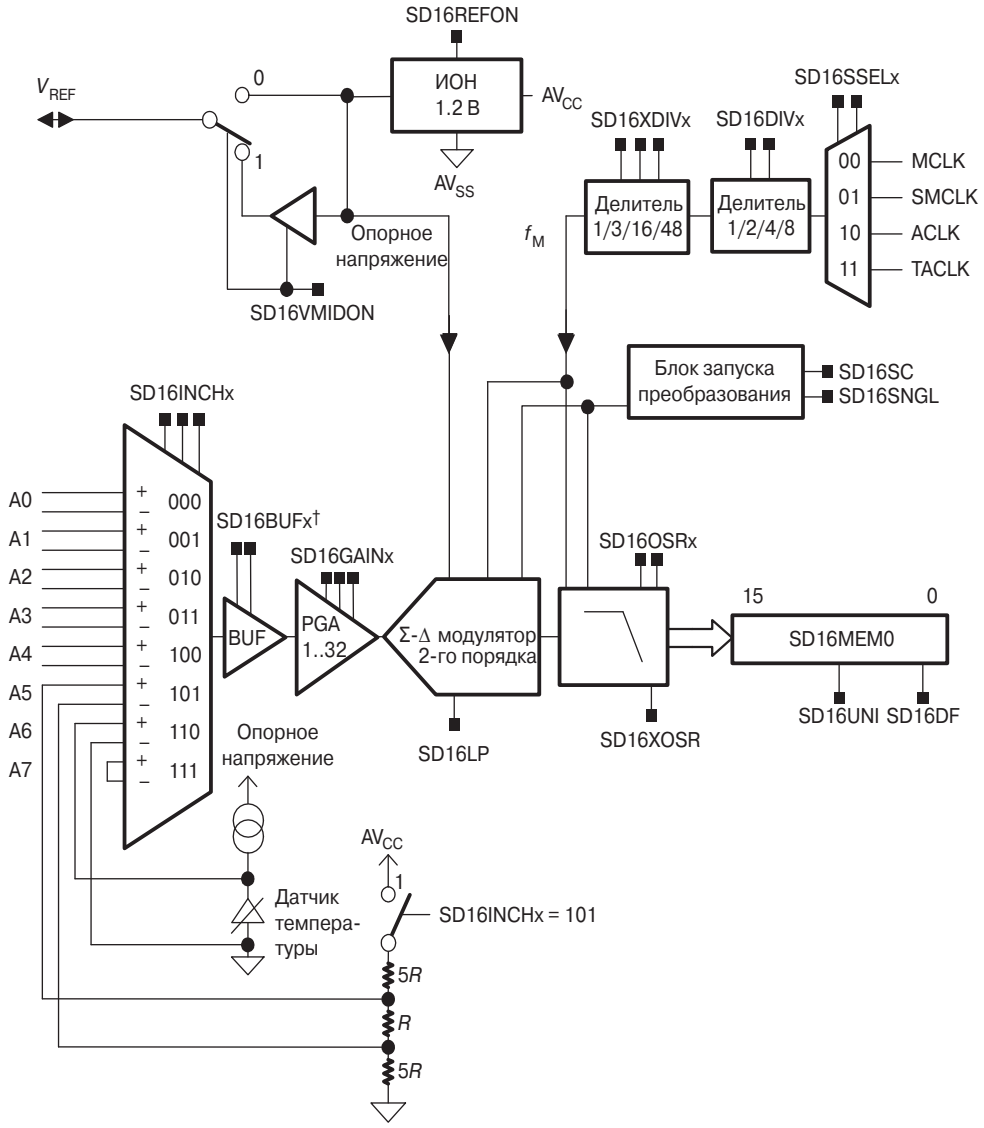
В состав модуля SD16_A входит сигма-дельта АЦП, имеющий буфер с высоким входным сопротивлением, и внутренний источник опорного напряжения. Модуль имеет до восьми мультиплексированных полностью дифференциальных аналоговых каналов, включая каналы для измерения сигнала от встроенного датчика температуры и напряжения питания. АЦП реализовано на базе сигма-дельта модулятора второго порядка и децимирующего цифрового фильтра. Для децимации используется гребенчатый фильтр с программируемым коэффициентом передискретизации (до 1024). Дополнительная фильтрация может осуществляться программно.

Входной буфер с высоким входным сопротивлением не реализован в моделях MSP430x20x3.

Модуль SD16_A имеет следующие особенности:

- 16-битная сигма-дельта архитектура;
- до восьми мультиплексированных дифференциальных аналоговых входов на канал (число входов зависит от модели, см. справочную документацию на конкретный микроконтроллер);
- программно активируемый встроенный генератор опорного напряжения (1.2 В);
- внешний или внутренний источник опорного напряжения (выбирается программно) ;
- встроенный датчик температуры;
- входная частота модулятора до 1.1 МГц;
- входной буфер с высоким входным сопротивлением (реализован не во всех моделях, обращайтесь к документации на конкретное устройство);
- программно активируемый режим преобразования с пониженным потреблением.

Блок-схема модуля SD16_A приведена на **Рис. 24.1**.



† Не реализован в моделях MSP430x20x3.

Рис. 24.1. Блок-схема модуля SD16_A.

24.2. Функционирование модуля SD16_A

Конфигурирование модуля SD16_A осуществляется пользовательской программой. Настройка модуля и его функционирование рассматриваются в следующих подразделах.

24.2.1. Ядро АЦП

Аналого-цифровое преобразование осуществляется однобитным сигма-дельта модулятором 2-го порядка. Однобитный компаратор, входящий в состав модулятора, выполняет квантование входного сигнала с частотой модуляции f_M. Получающийся в результате битовый поток усредняется цифровым фильтром для формирования результата преобразования.

24.2.2. Диапазон входного аналогового сигнала и усилитель с программируемым коэффициентом усиления (PGA)

Максимальный размах входного напряжения для каждой пары аналоговых входов зависит от установленного коэффициента усиления PGA. Входной сигнал должен находиться в диапазоне $\pm V_{FSR}$, при этом величина V_{FSR} определяется выражением:

$$V_{FSR} = \frac{V_{REF}/2}{GAIN_{PGA}}.$$

При опорном напряжении 1.2 В и единичном коэффициенте усиления максимальный размах входного сигнала равен:

$$\pm V_{FSR} = \frac{1.2\,В/2}{1} = \pm 0.6\,В.$$

Максимальные значения входных сигналов приводятся в справочной документации на конкретные модели.

24.2.3. Генератор опорного напряжения

Модуль SD16_A имеет встроенный источник опорного напряжения 1.2 В, который включается установкой бита SD16REFON. Для снижения уровня шумов при использовании этого источника рекомендуется между выводами V_{REF} и AV_{SS} подключать внешний конденсатор ёмкостью 0.1 мкФ. Напряжение внутреннего источника может использоваться вне микроконтроллера при SD16VMIDON = 1. Максимальная нагрузочная способность буфера встроенного источника опорного напряжения составляет 1 мА. При использовании внутреннего опорного напряжения вне микроконтроллера между выводами V_{REF} и AV_{SS} необходимо подключить конденсатор ёмкостью 0.47 мкФ. Точные значения параметров встроенного источника опорного напряжения приводятся в документации на конкретные модели.

При сброшенных битах SD16REFON и SD16VMIDON модуль может использовать внешнее опорное напряжение, подаваемое на вход V_{REF}.

24.2.4. Автоматическое отключение

Модуль SD16_A разработан для использования в приложениях с низким энергопотреблением. Поэтому он автоматически отключается в те периоды времени, когда преобразование не выполняется, и автоматически включается при запуске преобразования. Генератор опорного напряжения автоматически не отключается, однако его можно отключить вручную, сбросив бит SD16REFON. При отключенных ядре АЦП и генераторе опорного напряжения их ток потребления равен нулю.

24.2.5. Выбор входного канала

Модуль SD16_A может выполнять преобразование по восьми дифференциальным входам, подключённым через мультиплексор к PGA. Для преобразования внешних сигналов используется до пяти аналоговых входов (A0...A4). К входу A5 подключён внутренний резистивный делитель, предназначенный для измерения напряжения питания. К входу A6 подключен внутренний датчик температуры. Вход A7 предназначен для калибровки смещения входного каскада модуля SD16_A — прямой и инверсный выводы этого входа соединены между собой.

Конфигурирование аналогового входа

Для конфигурирования аналогового входа используются регистры SD16INCTL0 и SD16AE. С помощью битов SD16INCHx выбирается один из восьми дифференциальных входов аналогового мультиплексора. Коэффициент усиления PGA устанавливается битами SD16GAINx — всего доступно шесть значений коэффициента усиления. Биты SD16AEx используются для разрешения/запрещения использования выводов микроконтроллера в качестве аналоговых входов. Установка любого бита SD16AEx отключает входные и выходные буферы вывода цифрового порта, с которым мультиплексирован соответствующий аналоговый вход. Принципиальные схемы выводов портов приводятся в справочной документации на конкретные модели.

При изменении битов SD16INCHx и SD16GAINx во время преобразования, новые установки вступят в действие при очередном шаге децимации цифрового фильтра. Результаты трёх последующих преобразований, выполненных после изменения этих битов, могут быть некорректными из-за конечного времени установления цифрового фильтра. Эта проблема может быть разрешена автоматически с помощью битов SD16INTDLYx. При SD16INTDLYx = 00h запрос прерывания будет сгенерирован только после завершения 4-го по счёту преобразования.

В тех моделях, в которых имеется буфер с высоким входным сопротивлением, управление этим буфером осуществляется посредством битов SD16BUFx. Требуемое быстродействие буфера определяется частотой модулятора модуля SD16_A и задаётся в соответствии с **Табл. 24.1**.

Таблица 24.1. **Управление буфером с высоким входным сопротивлением**

SD16BUFx	Буфер	Частота модуляции f_M модуля SD16_A
00	Буфер отключен	—
01	Низкое быстродействие/малый ток	f_M < 200 кГц
10	Среднее быстродействие/средний ток	200 кГц < f_M < 700 кГц
11	Высокое быстродействие/большой ток	700 кГц < f_M < 1.1 МГц

Чтобы предотвратить наложение спектров, модуль SD16_A рекомендуется использовать с внешним антиэлайзинговым *RC*-фильтром. При частоте модулятора 1 МГц и коэффициенте передискретизации OSR = 256 частота среза такого фильтра должна быть не более 10 кГц. В приложениях, работающих с более узкополосными сигналами, можно применять фильтр с гораздо меньшей частотой среза.

24.2.6. Параметры аналогового входа

В модуле SD16_A используется входной каскад с переключаемыми конденсаторами, являющийся нагрузкой для источника сигнала, как показано на **Рис. 24.2**.

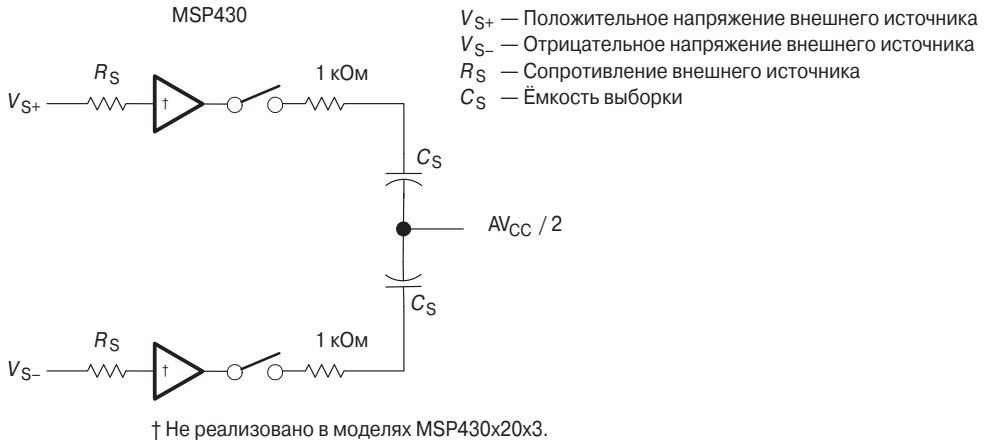


Рис. 24.2. Эквивалентная схема аналогового входа.

При использовании буферов сопротивление R_S не оказывает влияния на частоту дискретизации f_S. Если же буферы не применяются или просто отсутствуют в данной модели, то максимальную частоту дискретизации f_S можно определить, исходя из минимального значения времени установления $t_{settling}$ схемы дискретизации, согласно выражению:

$$t_{settling} \geq (R_S + 1\text{кОм}) \times C_S \times \ln\left(\frac{\text{GAIN} \times 2^{17} \times V_{Ax}}{V_{REF}}\right),$$

где $f_S = \dfrac{1}{2 \times t_{settling}}$ и $V_{Ax} = \max\left(\left|\dfrac{AV_{CC}}{2} - V_{S+}\right|, \left|\dfrac{AV_{CC}}{2} - V_{S-}\right|\right)$,

при этом значения V_{S+} и V_{S-} измеряются относительно AV_{SS}.

Ёмкость C_S зависит от заданного коэффициента усиления PGA (**Табл. 24.2**).

Таблица 24.2. **Ёмкость выборки**

Коэффициент усиления PGA	Ёмкость выборки C_S [пФ]
1	1.25
2, 4	2.5
8	5
16, 32	10

24.2.7. Цифровой фильтр

Цифровой фильтр, обрабатывающий битовый поток от модулятора, имеет АЧХ вида sinc^3. Передаточная функция этого фильтра в z-области имеет вид

$$H(z)=\left(\frac{1}{\text{OSR}}\times\frac{1-z^{-\text{OSR}}}{1-z^{-1}}\right)^3.$$

В частотной области передаточная функция фильтра определяется выражением

$$H(f)=\left[\frac{\text{sinc}\left(\text{OSR}\,\pi\frac{f}{f_M}\right)}{\text{sinc}\left(\pi\frac{f}{f_M}\right)}\right]^3=\left(\frac{1}{\text{OSR}}\times\frac{\sin\left(\text{OSR}\times\pi\frac{f}{f_M}\right)}{\sin\left(\pi\frac{f}{f_M}\right)}\right)^3,$$

где коэффициент передискретизации OSR представляет собой отношение частоты модулятора f_M к частоте дискретизации f_S.

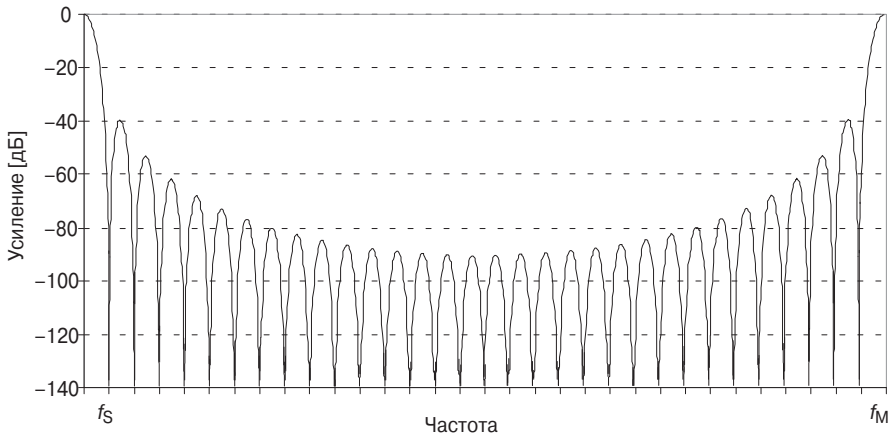


Рис. 24.3. Частотная характеристика гребенчатого фильтра при OSR = 32.

На **Рис. 24.3** приведена частотная характеристика фильтра при OSR = 32. Первый провал АЧХ фильтра (первый полюс), располагается на частоте $f_S = f_M/\text{OSR}$. Частоту полюса можно скорректировать, изменяя частоту модулятора f_M с помощью битов SD16SSELx и SD16DIVx и коэффициент передискретизации OSR с помощью битов SD16OSRx и SD16XOSR.

Цифровой фильтр осуществляет прореживание (децимацию) битового потока для всех разрешённых каналов АЦП и сохраняет результаты преобразования в регистре SD16MEM0 с периодичностью, соответствующей частоте выборки f_S.

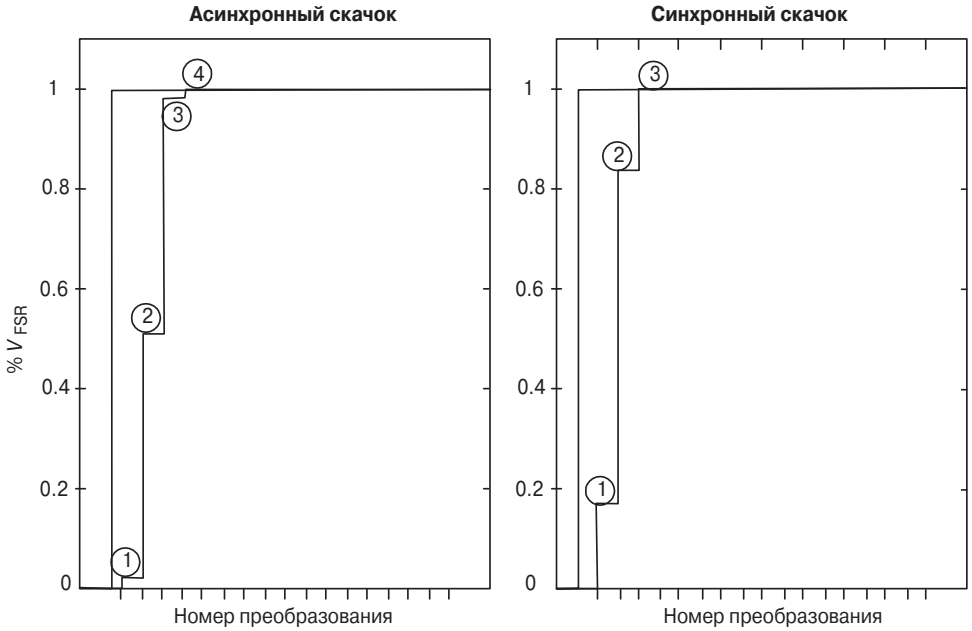


Рис. 24.4. Переходная характеристика цифрового фильтра и моменты преобразований.

На **Рис. 24.4** изображена переходная характеристика цифрового фильтра и указаны моменты преобразований. При скачкообразном изменении входного сигнала результат преобразования станет корректным только через некоторое время (время установления) после запуска преобразования. Биты SD16INTDLYx позволяют сформировать паузу, обеспечивающую получение корректного результата при изменении входного сигнала АЦП от минимального до максимального значения. Если скачок сигнала происходит синхронно с процессом прореживания, то корректный результат будет получен при выполнении третьего преобразования. При асинхронном изменении сигнала для получения корректного результата требуется одно дополнительное преобразование.

Выход цифрового фильтра

Разрядность значений на выходе цифрового фильтра зависит от коэффициента передискретизации и колеблется от 15 до 30 бит. На **Рис. 24.5** показаны выходные значения цифрового фильтра и их размещение в регистре SD16MEM0 для различных значений OSR, LSBACC и SD16UNI. Например, при OSR = 1024, LSBACC = 0 и SD16UNI = 1 в регистре SD16MEM0 содержатся биты 28...13 результата фильтрации. При OSR = 32 один (SD16UNI = 0) или два (SD16UNI = 1) младших бита результата всегда равны нулю.

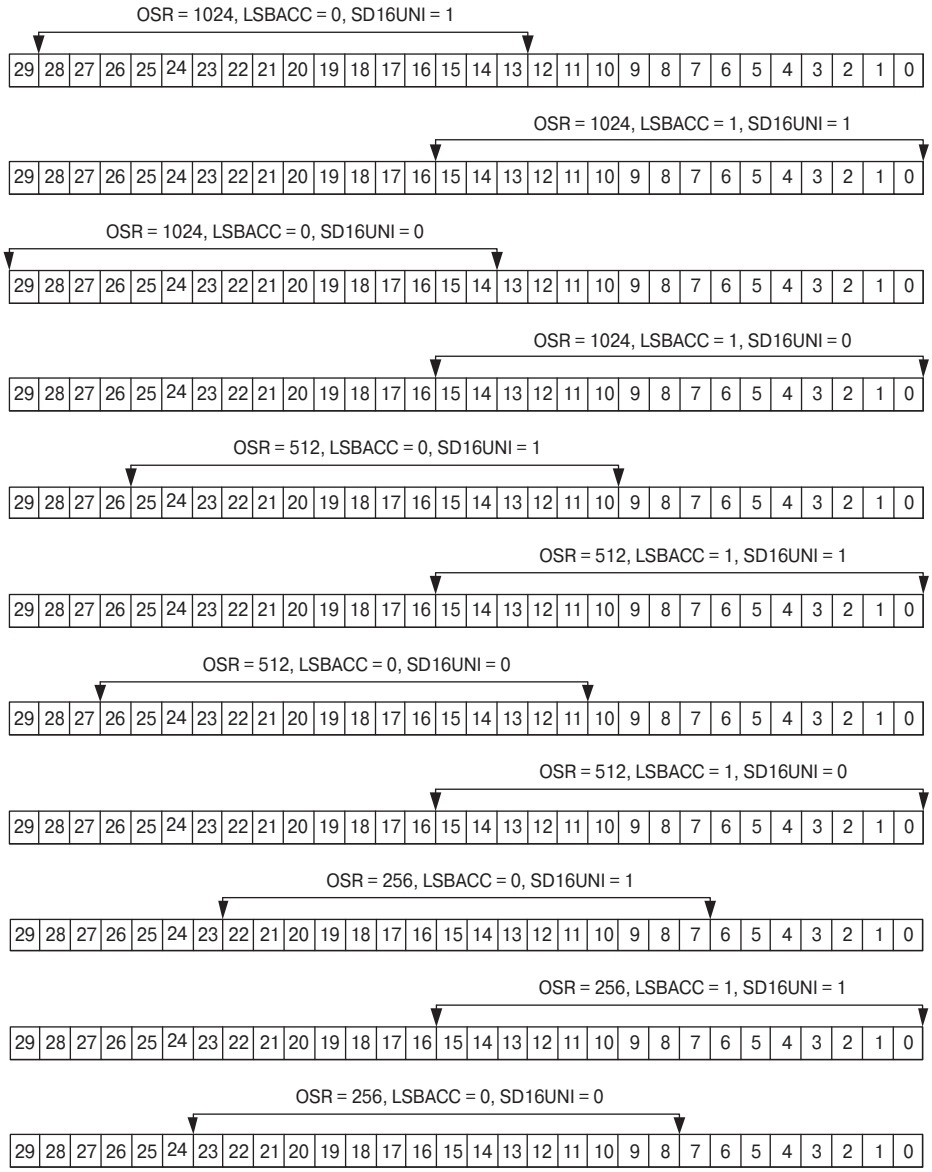


Рис. 24.5. Используемые биты результата цифровой фильтрации (*начало*).

Управляющие биты SD16LSBACC и SD16LSBTOG используются для доступа к младшим битам выходных значений фильтра. При SD16LSBACC = 1 из регистра SD16MEM0 можно считать младшие 16 бит результата преобразования, используя команду, оперирующую двухбайтными операндами. К регистру SD16MEM0 можно обращаться и с использованием однобайтных команд — в этом случае будет считано только 8 младших битов результата.

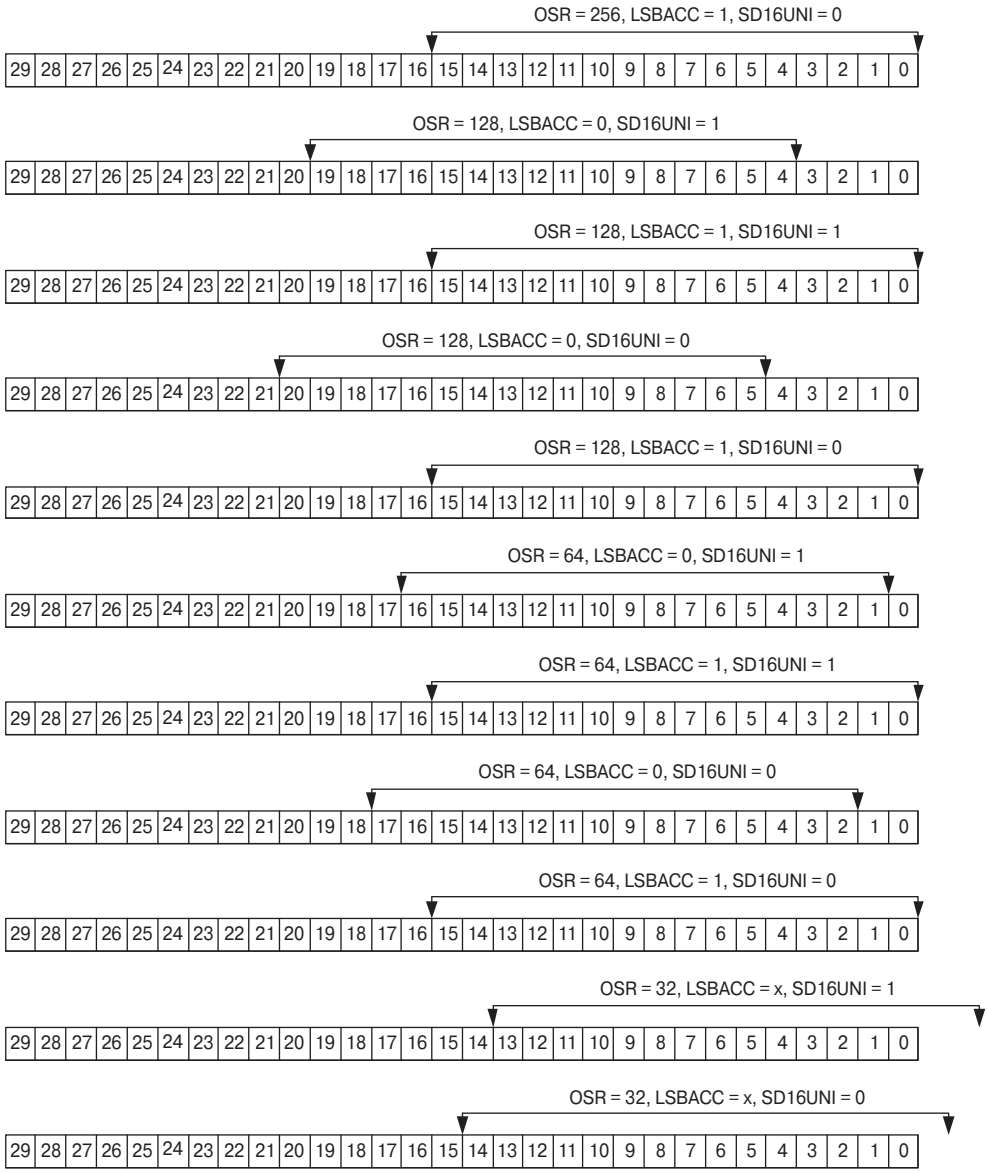


Рис. 24.5. Используемые биты результата цифровой фильтрации (*окончание*).

При SD16LSBTOG = 1 состояние бита SD16LSBACC автоматически изменяется при каждой операции чтения регистра SD16MEM0. Это позволяет получать полный результат преобразования за две операции чтения регистра SD16MEM0. Установка или сброс бита SD16LSBTOG не изменяет состояния бита SD16LSBACC до следующего обращения к регистру SD16MEM0.

24.2.8. Регистр данных SD16MEM0

С модулем SD16_A связан регистр SD16MEM0. Результаты преобразований загружаются в регистр SD16MEM0 после каждого шага децимации, осуществляемой цифровым фильтром. При записи нового значения в регистр SD16MEM0 устанавливается бит SD16IFG. Этот бит сбрасывается автоматически при чтении регистра SD16MEM0, а также может быть сброшен программно.

Формат выходных данных

Результат фильтрации может быть представлен в дополнительном коде, смещённом двоичном коде и прямом (однополярном) двоичном коде согласно **Табл. 24.3**. Формат данных определяется битами SD16DF и SD16UNI.

Таблица 24.3. **Формат данных**

SD16UNI	SD16DF	Формат	Напряжение на входе	SD16MEM0*	Выход цифрового фильтра (OSR = 256)
0	0	Двухполярный смещённый двоичный код	$+V_{FSR}$	FFFF	FFFFFF
			0	8000	800000
			$-V_{FSR}$	0000	000000
0	1	Двухполярный дополнительный код	$+V_{FSR}$	7FFF	7FFFFF
			0	0000	000000
			$-V_{FSR}$	8000	800000
1	0	Однополярный прямой код	$+V_{FSR}$	FFFF	FFFFFF
			0	0000	800000
			$-V_{FSR}$	0000	00000

* Независимо от установок битов SD16OSRx и SD16XOSRx; бит SD16LSBACC = 0.

Примечание. Операция измерения смещения и формат данных

При измерении смещения как по внешнему входу, так и с использованием внутреннего дифференциального входа A7 соответствующий канал должен работать в двухполярном режиме с SD16UNI = 0.

Зависимость между входным напряжением при его изменении от минимального $-V_{FSR}$ до максимального $+V_{FSR}$ значения и результатом преобразования показана на **Рис. 24.6**.

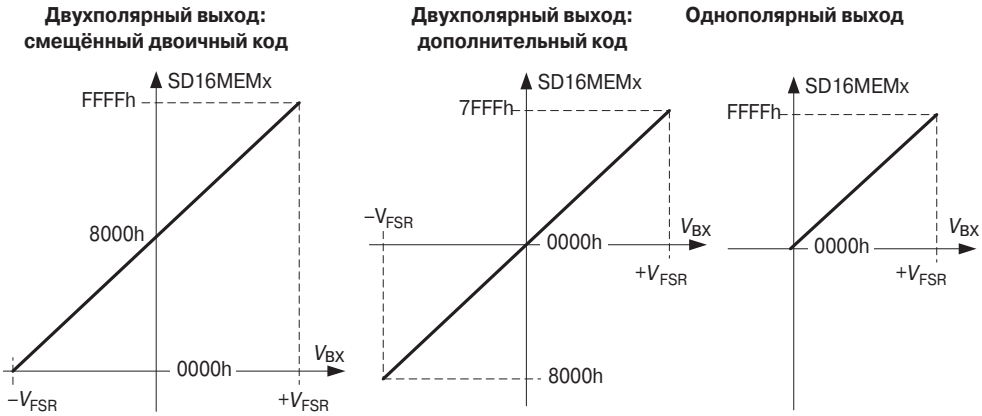


Рис. 24.6. Зависимость результата преобразования от входного напряжения.

24.2.9. Режимы преобразования

Модуль SD16_A поддерживает два режима работы, определяемые битом SD16SNGL. Эти режимы указаны в **Табл. 24.4**.

Таблица 24.4. **Режимы преобразования модуля ADC10**

SD16SNGL	Режим	Операция
0	Однократного преобразования	Выполняется однократное преобразование заданного канала
1	Непрерывного преобразования	Выполняется периодическое преобразование заданного канала

Режим однократного преобразования

Режим однократного преобразования включается при SD16SNGL = 1. В этом режиме установка бита SD16SC запускает преобразование выбранного канала. Бит SD16SC сбрасывается автоматически после завершения преобразования.

В случае сброса бита SD16SC до завершения преобразования оно немедленно останавливается, а канал и цифровой фильтр выключаются. При сбросе бита SD16SC также может измениться содержимое регистра SD16MEM0. Чтобы предотвратить считывание некорректного результата преобразования рекомендуется перед сбросом бита SD16SC выполнять чтение регистра SD16MEM0.

Режим непрерывного преобразования

Режим непрерывного преобразования включается при SD16SNGL = 0. В этом режиме при установке бита SD16SC запускается процесс преобразования выбранного канала, который продолжается до тех пор, пока этот бит не будет сброшен программно.

При сбросе бита SD16SC процесс преобразования немедленно останавливается, а канал и цифровой фильтр выключаются. При сбросе бита SD16SC также может измениться содержимое регистра SD16MEM0. Чтобы предотвратить считывание некорректного результата преобразования, рекомендуется перед сбросом бита SD16SC выполнять чтение регистра SD16MEM0.

Процесс преобразования показан на **Рис. 24.7**.

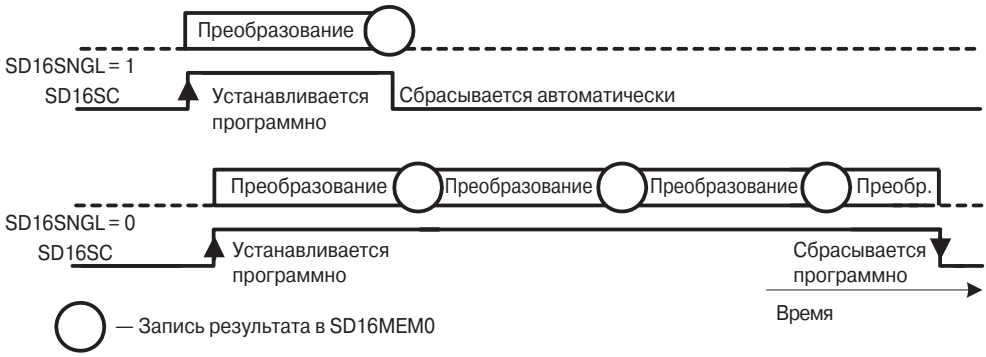


Рис. 24.7. Преобразование одного канала.

24.2.10. Использование встроенного датчика температуры

Для использования встроенного датчика температуры пользователь выбирает соответствующий аналоговый вход SD16INCHx = 110 и устанавливает бит SD16REFON = 1. Остальные настройки АЦП, включая установки битов SD16INTDLYx и SD16GAINx, задаются так же, как и при задействовании внешних аналоговых входов. Поскольку для применения встроенного датчика температуры должен быть включён внутренний источник опорного напряжения, при преобразовании сигнала от датчика нельзя использовать внешнее опорное напряжение. Более того, в этом случае внутренний и внешний источники опорного напряжения оказываются соединёнными друг с другом. Чтобы свести к минимуму влияние внешнего источника на результат преобразования, можно установить бит SD16VMIDON.

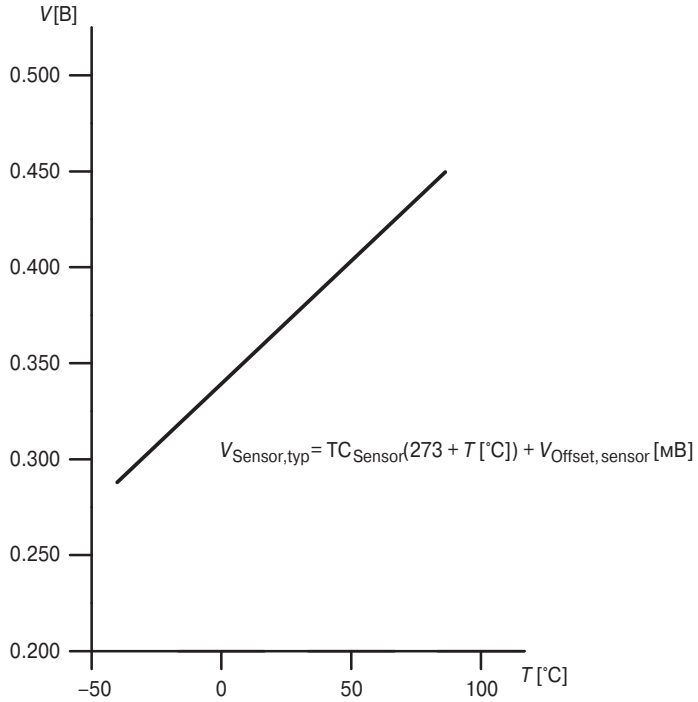


Рис. 24.8. Типичная передаточная функция встроенного датчика температуры.

Типичная передаточная функция датчика температуры приведена на **Рис. 24.8**. При переключении входа АЦП на канал датчика температуры необходимо с помощью битов SD16INDLYx сформировать задержку, достаточную для установления цифрового фильтра и гарантирующую получение корректного результата. Датчик температуры может иметь значительную погрешность смещения, поэтому в большинстве случаев требует калибровки. Более подробная информация приводится в справочной документации на конкретные модели.

24.2.11. Обработка прерываний

Модуль SD16_A имеет два источника прерываний:

- SD16IFG;
- SD16OVIFG.

Флаг SD16IFG устанавливается при загрузке результата преобразования в регистр SD16MEM0. Если бит SD16IEx и бит общего разрешения прерываний GIE установлены, то при установке флага SD16IFG генерируется запрос прерывания. Флаг SD16OVIFG устанавливается в том случае, если результат преобразования заносится в регистр SD16MEM0 до считывания из него предыдущего результата.

Регистр вектора прерывания SD16IV

Все источники прерываний модуля SD16_A имеют различные приоритеты и связаны с единственным вектором прерываний. Чтобы определить, какой из разрешённых источников вызвал генерацию прерывания, используется регистр вектора прерывания SD16IV. Разрешённое прерывание с наивысшим приоритетом формирует в регистре SD16IV число (см. описание регистра). Это число можно оценить или же просто прибавить к счётчику команд для автоматического перехода к соответствующей секции программы. Запрещённые прерывания модуля SD16_A не влияют на содержимое регистра SD16IV.

При любом обращении к регистру ADC12IV как для чтения, так и для записи состояние флагов SD16IFG и SD16OVIFG не изменяется. Флаг SD16IFG сбрасывается автоматически при чтении регистра SD16MEM0 или же может быть сброшен программно. Флаг SD16OVIFG сбрасывается только программно.

При наличии второго установленного флага, сразу же после обработки текущего прерывания генерируется новое прерывание. Например, если на момент обращения в процедуре обработки прерывания к регистру SD16IV были установлены оба флага SD16IFG и SD16OVIFG, то первым будет обработано прерывание SD16OVIFG (этот флаг должен быть сброшен программно). После выполнения команды `RETI` процедуры обработки прерывания, флаг SD16IFG вызовет генерацию нового прерывания.

Задержка генерации прерывания

Биты SD16INTDLYx управляют моментом генерации запроса прерывания. Эти биты позволяют сформировать задержку между запуском преобразования и генерацией запроса прерывания длительностью до 4 циклов преобразования для установления цифрового фильтра перед генерацией запроса. Данная задержка формируется каждый раз при установке бита SD16SC или изменении битов SD16GAINx или SD16INCHx. Биты SD16INTDLYx запрещают генерацию прерывания по переполнению на заданное число циклов. Во время формирования задержки запросы прерываний не генерируются.

24.3. Регистры модуля SD16_A

Список регистров модуля SD16_A приведён в **Табл. 24.5**.

Таблица 24.5. **Регистры модуля SD16_A**

Регистр	Обозначение	Тип регистра	Адрес	Исходное состояние
Регистр управления модуля SD16_A	SD16CTL	Чтение/запись	0100h	Сбрасывается после PUC
Регистр вектора прерываний модуля SD16_A	SD16IV	Чтение/запись	0110h	Сбрасывается после PUC
Регистр управления канала 0 модуля SD16_A	SD16CCTL0	Чтение/запись	0102h	Сбрасывается после PUC
Регистр данных модуля SD16_A	SD16MEM0	Чтение/запись	0112h	Сбрасывается после PUC
Регистр управления входом модуля SD16_A	SD16INCTL0	Чтение/запись	0B0h	Сбрасывается после PUC
Регистр разрешения аналоговых входов модуля SD16_A	SD16AE	Чтение/запись	0B7h	Сбрасывается после PUC

SD16CTL, регистр управления модуля SD16_A

15	14	13	12	11	10	9	8
Reserved				**SD16XDIVx**			**SD16LP**
r0	r0	r0	r0	rw–0	rw–0	rw–0	rw–0

7	6	5	4	3	2	1	0
SD16DIVx		**SD16SSELx**		**SD16 VMIDON**	**SD16 REFON**	**SD16OVIE**	**Reserved**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	r0

Reserved Биты 15...12 Зарезервированы. Читаются как 0.

SD16XDIVx Биты 11...9 Коэффициент деления дополнительного делителя тактового сигнала модуля SD16_A.
- 000 /1
- 001 /3
- 010 /16
- 011 /48
- 1xx Зарезервировано

SD16LP Бит 8 Режим пониженного энергопотребления. Этот бит включает режим работы модуля SD16_A с пониженной скоростью преобразования и уменьшенным потреблением.
- 0 Режим пониженного энергопотребления выключен
- 1 Режим пониженного энергопотребления включён. Максимальная тактовая частота модуля SD16_A снижена.

SD16DIVx Биты 7...6 Коэффициент деления основного делителя тактового сигнала модуля SD16_A.
- 00 /1
- 01 /2
- 10 /4
- 11 /8

SD16SSELx	Биты 5...4	Выбор источника тактового сигнала модуля SD16_A. 00 MCLK 01 SMCLK 10 ACLK 11 Внешний сигнал TACLK
SD16 VMIDON	Бит 3	Включение буфера V_{MID}. 0 Буфер выключен 1 Буфер включен
SD16 REFON	Бит 2	Включение генератора опорного напряжения. 0 Генератор опорного напряжения выключен 1 Генератор опорного напряжения включён
SD16OVIE	Бит 1	Разрешение прерывания по переполнению. Для разрешения прерывания также должен быть установлен бит GIE. 0 Прерывание запрещено 1 Прерывание разрешено
Reserved	Бит 0	Зарезервирован. Читается как 0.

SD16CCTL0, регистр управления канала 0 модуля SD16_A

15	14	13	12	11	10	9	8
Reserved	**SD16BUFx***		**SD16UNI**	**SD16XOSR**	**SD16SINGL**	**SD16OSRx**	
r0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

7	6	5	4	3	2	1	0
SD16 LSBTOG	**SD16 LSBACC**	**SD16 OVIFG**	**SD16DF**	**SD16IE**	**SD16IFG**	**SD16SC**	**Reserved**
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	r–0

Reserved	Бит 15	Зарезервирован. Читается как 0.
SD16BUFx	Биты 14...13	Режим работы буфера с высоким входным сопротивлением. 00 Буфер отключён 01 Низкое быстродействие/малый ток 10 Среднее быстродействие/средний ток 11 Высокое быстродействие/большой ток
SD16UNI	Бит 12	Выбор однополярного режима. 0 Двухполярный режим 1 Однополярный режим
SD16XOSR	Бит 11	Расширенный диапазон коэффициента передискретизации. Этот бит совместно с битами SD16OSRx определяет коэффициент передискретизации. См. описание битов SD16OSRx.
SD16SINGL	Бит 10	Выбор режима однократного преобразования. 0 Режим непрерывного преобразования 1 Режим однократного преобразования

SD16OSRx	Биты 9...8	Коэффициент передискретизации. При SD16XOSR = 0: 00 256 01 128 10 64 11 32 При SD16XOSR = 1: 00 512 01 1024 10 Зарезервировано 11 Зарезервировано
SD16 LSBTOG	Бит 7	Переключение бита доступа к младшим битам результата. При установленном бите SD16LSBTOG состояние бита SB16LSBACC изменяется после каждой операции чтения регистра SD16MEM0. 0 Состояние SB16LSBACC не изменяется при чтении регистра SD16MEM0 1 Состояние SB16LSBACC изменяется при каждом чтении регистра SD16MEM010
SD16 LSBACC	Бит 6	Доступ к младшим битам результата преобразования. Бит SD16LSBACC позволяет обращаться к старшим или младшим 16 бит результата преобразования АЦП. 0 SD16MEMx содержат старшие 16 бит результата преобразования 1 SD16MEMx содержат младшие 16 бит результата преобразования
SD16 OVIFG	Бит 5	Флаг прерывания по переполнению SD16MEM0. 0 Не было запроса прерывания 1 Есть запрос прерывания
SD16DF	Бит 4	Представление результата преобразования. 0 Смещённый двоичный код 1 Дополнительный код
SD16IE	Бит 3	Разрешение прерывания модуля SD16_A. 0 Прерывание запрещено 1 Прерывание разрешено
SD16IFG	Бит 2	Флаг прерывания модуля SD16_A. Этот бит устанавливается при появлении нового результата преобразования. Флаг SD16IFG сбрасывается автоматически при чтении регистра SD16MEM0 или может быть сброшен программно. 0 Не было запроса прерывания 1 Есть запрос прерывания
SD16SC	Бит 1	Запуск преобразования. 0 Не начинать преобразование 1 Начать преобразование
Reserved	Бит 0	Зарезервирован. Читается как 0.

SD16INCTL0, регистр управления входом модуля SD16_A

7	6	5	4	3	2	1	0
SD16INTDLYx		SD16GAINx			SD16INCHx		
rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0	rw–0

SD16 INTDLYx Биты 7...6 Задержка генерации прерывания после запуска преобразования. Эти биты определяют задержку генерации первого прерывания после запуска преобразования.
00 Прерывание генерируется после 4-го преобразования
01 Прерывание генерируется после 3-го преобразования
10 Прерывание генерируется после 2-го преобразования
11 Прерывание генерируется после 1-го преобразования

SD16GAINx Биты 5...3 Коэффициент усиления входного усилителя.
000 ×1
001 ×2
010 ×4
011 ×8
100 ×16
101 ×32
110 Зарезервировано
111 Зарезервировано

SD16INCHx Биты 2...0 Выбор входного канала.
000 A0
001 A1
010 A2
011 A3
100 A4
101 A5 — $(AV_{CC} - AV_{SS})/11$
110 A6 — Датчик температуры
111 A7 — Короткозамкнутый вход для измерения смещения PGA

SD16MEM0, регистр данных модуля SD16_A

15	14	13	12	11	10	9	8
Результат преобразования							
r	r	r	r	r	r	r	r

7	6	5	4	3	2	1	0
Результат преобразования							
r	r	r	r	r	r	r	r

Результат преобразования Биты 15...0 Результат преобразования. Регистр SD16MEM0 содержит старшие или младшие 16 бит выходного значения цифрового фильтра в зависимости от значения бита SD16LSBACC.

SD16AE, регистр разрешения аналоговых входов модуля SD16_A

7	6	5	4	3	2	1	0
SD16AE7	**SD16AE6**	**SD16AE5**	**SD16AE4**	**SD16AE3**	**SD16AE2**	**SD16AE1**	**SD16AE0**
rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0	rw-0

SD16AEx Биты 7...0 Разрешение аналогового входа модуля SD16_A.
0 Внешний вход отключён. Инверсные входы внутри кристалла подключаются к V_{SS}
1 Внешний вход включён

SD16IV, регистр вектора прерываний модуля SD16_A

15	14	13	12	11	10	9	8
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**
r0	r0	r0	r0	r0	r0	r0	r0

7	6	5	4	3	2	1	0
0	**0**	**0**	**SD16IVx**				**0**
r0	r0	r–(0)	r–0	r–0	r–0	r–0	r0

SD16IVx Биты 15...0 Значение вектора прерывания модуля SD16_A

Содержимое SD16IV	Источник прерывания	Флаг прерывания	Приоритет прерывания
00h	Нет прерывания	—	
02h	Переполнение SD16MEMx	SD16OVIFG (регистр SD16CCTLx)	Высший
04h	Прерывание SD16_A	SD16IFG (регистр SD16CCTL0)	
06h	Зарезервировано	—	
08h	Зарезервировано	—	
0Ah	Зарезервировано	—	
0Ch	Зарезервировано	—	
0Eh	Зарезервировано	—	
010h	Зарезервировано	—	Низший

ГЛАВА 25

ВСТРОЕННЫЙ МОДУЛЬ ЭМУЛЯЦИИ EEM

В этой главе описывается работа встроенного модуля эмуляции (EEM), реализованного во всех микроконтроллерах семейства MSP430.

25.1. Введение

Во всех флэш-микроконтроллерах семейства MSP430 имеется встроенный модуль эмуляции EEM. Доступ к данному модулю и управление им осуществляется по интерфейсу JTAG. Варианты исполнений модуля EEM меняются от устройства к устройству и описаны в разделе 25.3 «Конфигурации модуля EEM», а также в справочной документации на конкретные модели.

В общей сложности модуль EEM имеет следующие возможности:

- исполнение кода без вмешательства в основную программу с управлением точками останова в реальном времени;
- пошаговое выполнение программы (шаг, шаг с заходом в блок и шаг с пропуском блока);
- полная поддержка режимов пониженного энергопотребления;
- поддержка всех допустимых частот для всех источников тактового сигнала;
- до восьми (зависит от модели) аппаратных триггеров/точек останова на шину адреса (MAB) или шину данных (MDB);
- до двух (зависит от модели) аппаратных триггеров/точек останова на запись в регистр ЦПУ;
- триггеры на шины MAB, MDB и на обращение к регистрам ЦПУ могут объединяться для формирования до восьми (зависит от устройства) комбинированных триггеров/точек останова;
- задание последовательности срабатывания триггеров (зависит от устройства);
- сохранение состояний внутренней шины и сигналов управления во встроенном буфере трассировки (зависит от устройства);
- управление тактовыми сигналами таймеров, коммуникационных и других периферийных модулей на уровне всего устройства или отдельно для каждого модуля при останове процесса эмуляции.

На **Рис. 25.1** приведена упрощённая блок-схема наиболее развитой на момент написания руководства реализации модуля EEM моделей 2xx.

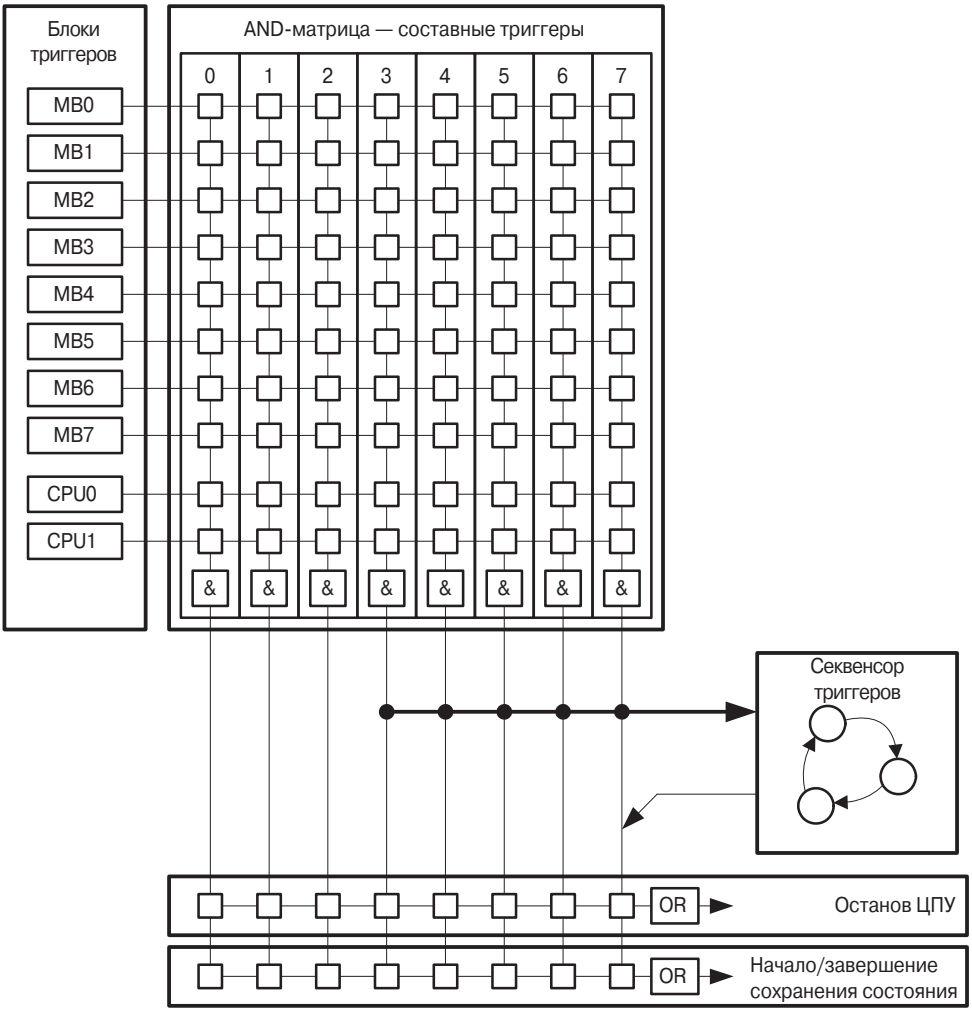


Рис. 25.1. Наиболее развитая реализация встроенного модуля эмуляции EEM.

Более подробно об использовании модуля EEM совместно с отладчиком интегрированной среды разработки IAR Embedded Workbench™ можно узнать из документа «Advanced Debugging Using the Enhanced Emulation Module» (SLAA263), имеющегося на сайте www.msp430.com. Отладчик среды Code Composer Essential (CCE) и подавляющее большинство других отладчиков, поддерживающих микроконтроллеры семейства MSP430, обладают такими же или близкими возможностями. Более подробную информацию можно получить из руководства пользователя для используемого отладчика.

25.2. Функциональные узлы модуля EEM

25.2.1. Триггеры

Управление событиями в модуле EEM осуществляется посредством триггеров, которые представляют собой внутренние сигналы, извещающие о наступлении того или иного события. Эти триггеры могут использоваться как простые точки останова. Кроме того, можно объединять несколько триггеров, чтобы сделать возможным обнаружение сложных событий и выполнение различных ответных действий без останова ЦПУ.

Триггеры могут использоваться для управления следующими функциональными блоками модуля EEM:

- точки останова (ЦПУ останавливается);
- буфер трассировки;
- секвенсор.

Существует две разновидности триггеров: триггер памяти и триггер записи в регистр ЦПУ.

Любой из блоков триггера памяти может использоваться независимо от других блоков для сравнения содержимого MAB или MDB с заданным значением. В зависимости от конкретного исполнения модуля EEM возможны операции сравнения =, ≠, ≥ и ≤. Кроме того, используя маску, можно сравнивать только заданные биты. В зависимости от исполнения модуля маска может быть побитовой или побайтовой. Помимо выбора шины и операции сравнения, можно задавать условие активации триггера. Такими условиями являются операция чтения, операция записи, обращение контроллера DMA и выборка команды.

Любой из блоков триггера записи в регистр ЦПУ может использоваться независимо от других блоков для сравнения записываемого значения с заданным. Контролируемый регистр задаётся отдельно для каждого триггера. Возможны операции сравнения =, ≠, ≥ и ≤. Кроме того, используя маску, можно сравнивать только заданные биты.

Триггеры обоих типов могут объединяться для образования более сложных триггеров. Такой составной триггер, например, может сигнализировать о записи конкретного значения по заданному адресу.

25.2.2. Секвенсор триггеров

Секвенсор триггеров позволяет задавать определённую последовательность срабатывания триггеров, вызывающую формирование события останова или сохранения состояния. Секвенсор обладает следующими возможностями:

- четыре состояния;
- два перехода из одного состояния к любому другому состоянию;
- триггер сброса, переводящий секвенсор в состояние №0.

Запуск секвенсора всегда начинается с состояния №0, а выполнение заданного действия производится при его переходе в состояние №3. Если состояния №1 и №2 не требуются, то они могут быть пропущены.

25.2.3. Внутренний буфер трассировки

Функция сохранения состояния используется для сохранения во внутреннем буфере информации о шинах MAB, MDB и управляющих сигналах ЦПУ (т.е. чтение, запись или выборка команды) без вмешательства в выполнение программы. Внутренний буфер может содержать до восьми элементов. Гибкая конфигурация позволяет пользователю очень эффективно записывать интересующую его информацию.

25.2.4. Управление тактовыми сигналами

Модуль EEM обеспечивает гибкое управление системой синхронизации микроконтроллера (возможности такого управления также зависят от конкретной модели). Это может оказаться полезным в тех случаях, если периферийные модули должны продолжать работу после останова ЦПУ (например, чтобы модуль UART смог завершить передачу символа или чтобы таймер продолжал генерировать ШИМ-сигнал).

Управление системой синхронизации является довольно гибким и поддерживает как модули, требующие наличия тактового сигнала, так и модули, которые должны останавливаться одновременно с ЦПУ при попадании на точку останова.

25.3. Конфигурации модуля EEM

Возможные варианты исполнений модуля EEM в микроконтроллерах семейства MSP430x2xx указаны в **Табл. 25.1**. Наличие той или иной конфигурации зависит от конкретной модели — для уточнения следует изучить справочную документацию на интересующий микроконтроллер.

Таблица 25.1. **Конфигурации модуля EEM семейства MSP430x2xx**

Возможности	XS	S	M	L
Триггеры памяти	2 (только = и ≠)	3	5	8
Маскирование триггера памяти	1) Младший байт 2) Старший байт	1) Младший байт 2) Старший байт	1) Младший байт 2) Старший байт	Все 16 или 20 бит
Триггеры записи в регистр ЦПУ	0	1	1	2
Составные триггеры	2	4	6	8
Секвенсор	Нет	Нет	Нет	Есть
Сохранение состояния	Нет	Нет	Нет	Есть

Модули EEM всех моделей семейства 2xx имеют, как минимум, следующие возможности:

- По меньшей мере, два триггера MAB/MDB, поддерживающие:
  – распознавание обращений к ЦПУ, DMA, и операций чтения и записи;
  – операции сравнения =, ≠, ≥, ≤ (в конфигурации XS только =, ≠).
- По крайней мере, два регистра составных триггеров.
- Аппаратные точки останова, осуществляющие останов ЦПУ.
- Управление системой синхронизации с индивидуальным управлением тактированием отдельных периферийных модулей (в некоторых конфигурациях XS управление тактовыми сигналами модулей осуществляется аппаратно).

Книги Издательского дома «Додэка-XXI» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: **123242, Москва, а/я 20**
или по электронномуадресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки:

тел. **(495) 258-91-94, 258-91-95**;
e-mail: **books@alians-kniga.ru**.

Семейство микроконтроллеров MSP430x2xx.
Архитектура, программирование, разработка приложений

Подписано в печать 10.04.2010. Формат 70x100/16. Бумага офсетная.
Гарнитура «NewtonC». Печать офсетная.
Объем 34,0 п. л. Усл. п. л. 44,1.
Тираж 1 000 экз. Код MSP430

Издательский дом «Додэка-XXI»
ОКП 95 3000
105318 Москва, а/я 70
Тел./факс: (495) 366-04-56, 366-11-55
E-mail: red@dodeca.ru

Отпечатано с готовых диапозитивов в ОАО «Щербинская типография»
117623 Москва, ул. Типографская, д.10

Новинки издательства «Додэка-XXI»

Бонни Бэйкер

Что нужно знать цифровому разработчику об аналоговой электронике

Книга может служить практическим руководством разработчика, охватывая наиболее важные сферы проектирования аналоговой электроники: аналого-цифровое и цифро-аналоговое преобразование, применение операционных усилителей и фильтров, а также интеграцию аналоговых и цифровых систем. Материал книги составлен таким образом, чтобы помочь инженерам, занимающимся, в основном, разработкой цифровой электроники, освоить проектирование аналоговых схем.

Книга помогает по-новому, используя аналоговый подход, взглянуть на проектирование устройств, что позволяет быстро решать возникающие проблемы. Особое внимание уделяется таким основополагающим, но почему-то редко рассматриваемым в подобных изданиях темам, как влияние шумов на качество работы схемы, применение инструментальных средств и оборудования при настройке и тестировании аналоговых устройств. Книгу отличают многочисленные описания практических примеров проектирования схем, а также советы по выбору подходящих инструментальных программно-аппаратных средств, без которых не обойтись в процессе разработки.

Книга предназначена для инженеров-практиков, разрабатывающих цифровые схемы, а также будет полезна студентам и преподавателям, сталкивающимся с проблемами проектирования аналоговых схем для цифровых устройств. Свободный стиль повествования, используемый автором, одинаково хорошо воспринимается как студентами, так и профессиональными инженерами.

Финкенцеллер К.

RFID-технологии. Справочное пособие

Данный справочник представляет собой исчерпывающий обзор систем RFID (систем радиочастотной идентификации), который, главным образом, ориентирован на практические вопросы их применения. Системы RFID находят применение в самых разнообразных областях, например, в системах контроля допуска на предприятия или в гостиничные номера, в качестве электронных иммобилайзеров или же как средства предотвращения краж в супермаркетах. Основой подобных систем являются электронные носители данных, не обладающие собственным источником питания (транспондеры). Информация с такого носителя считывается бесконтактным способом.

В книге описываются физические принципы работы систем радиочастотной идентификации, содержится информация по действующим в этой области стандартам и основным областям практического применения RFID-систем. Представлены также материалы, касающиеся физических принципов функционирования СВЧ и микроволновых систем, которые приобретают все большее значение в связи с открытием соответствующих частотных диапазонов.

Для иллюстрации достаточно сложных понятий используются многочисленные рисунки. Приводятся примеры, которые поясняют вопросы, связанные с практическим применением систем радиочастотной идентификации. В приложении содержится контактная информация, а также обзор стандартов и рекомендаций, приводятся ссылки на литературу и на источники информации в сети Интернет.

Предназначена для разработчиков систем радиочастотной идентификации, инженеров, студентов, а также будет полезна менеджерам, занимающимся вопросами применения устройств RFID.