

Blink them to death using
Embedded Swift



Hostname: Eric Bariaux

Occupation: Software Engineer

Really: Cook at heart

Up time: 53y 77d 6h 34m

Embedded Swift

Embedded systems limitations



Processor

Single core 32-bit

Single core 64-bit

Reduce

Frequency

64 MHz

520 MHz

8x

Runtime overhead

RAM

256 KB

512 MB

2000x

Memory footprint

Flash

1 MB

8 GB

8000x

Executable code size

Embedded Swift

- Official initiative from Apple
- Subset of the language, not dialect
- Compilation mode enforces constraints to achieve goals of reducing:
 - Runtime overhead
 - Memory footprint
 - Executable code size

How?

- Remove everything that is dynamic
 - Dynamic reflection facilities (such as mirrors, as? downcasts, and printing arbitrary values)
 - Existential types (any)
 - Generics instantiation
 - Obj-C interop
 - Dynamic code loading (plug-ins)
- Minimal runtime library / no need for metadata
- Reduced Swift Standard Library (e.g. no Codable)
- Aggressive dead code stripping

In practice

- `swiftc -target <target triple> -enable-experimental-feature Embedded -wmo file.swift -c -o output.o`
- Target triple e.g. for NRF: `armv7em-none-none-eabi`
- From Embedded Swift user manual
 - Embedded Swift is a **compilation model** that's analogous to a traditional C compiler in the sense that the compiler **produces an object file (.o)** that can be simply **linked with your existing code**, and it's not going to require you to port any libraries or runtimes.
- Linking is done as usual using the embedded platform toolchain

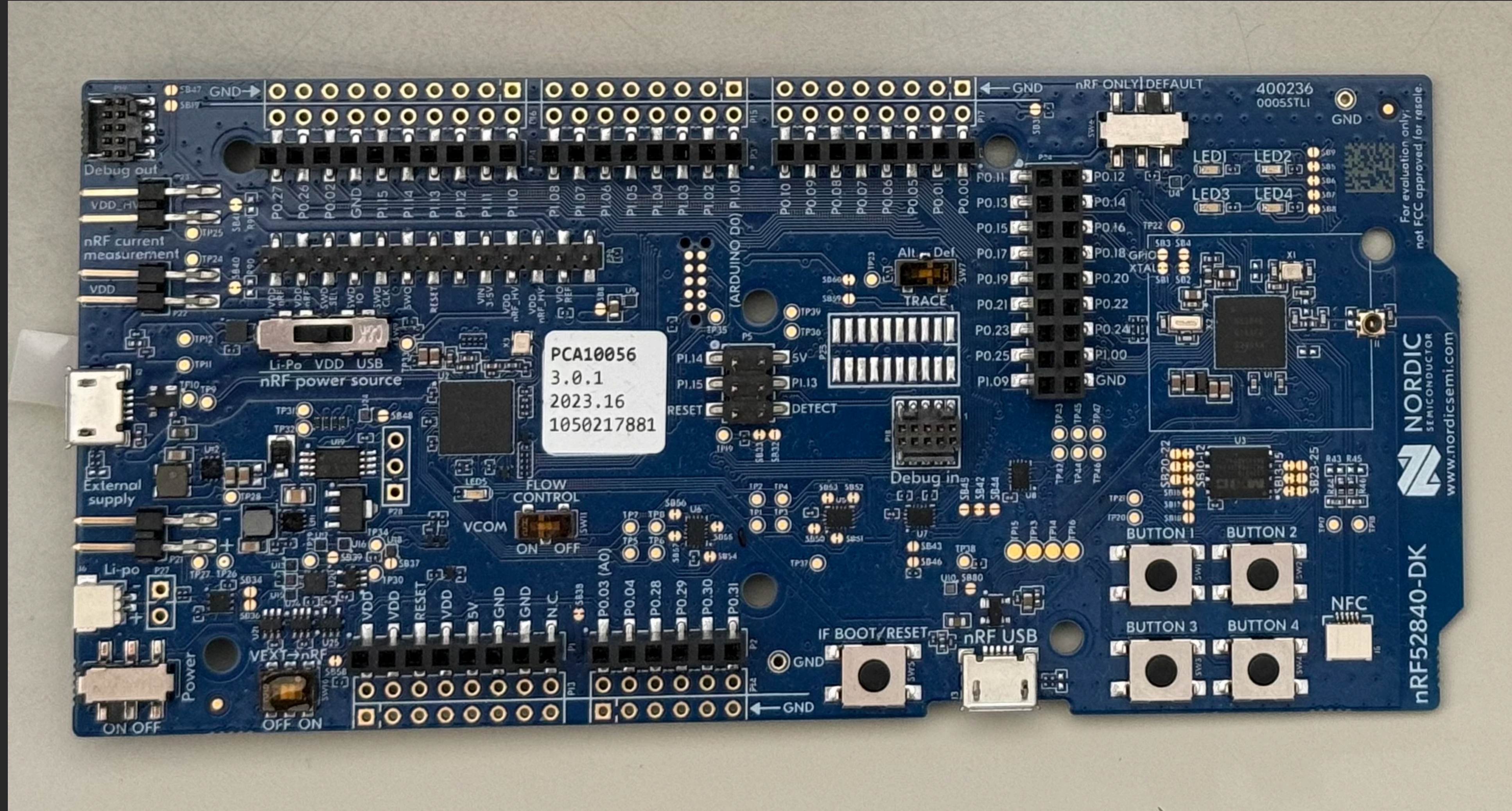
Embedded dev steps

From Embedded Swift user manual

A typical setup and build + run cycle for an embedded development board involves:

- (1) Getting an SDK with the C compilers, headers and libraries for the target
- (2) Building the C source code, and Swift source code into object files.
- (3) Linking all the libraries, C object files, and Swift object files.
- (4) Post-processing the linked firmware into a flashable format (UF2, BIN, HEX, or bespoke formats)
- (5) Uploading the flashable binary to the board over a USB cable using some vendor-provided JTAG/SWD tool, by copying it to a fake USB Mass Storage volume presented by the board or a custom platform bootloader.
- (6) Restarting the board, observing physical effects of the firmware (LEDs light up) or UART output over USB, or presence on network, etc.

TARGET : NRF52840 DK



Step 2

Test the Embedded Swift examples

<https://github.com/apple/swift-embedded-examples/tree/main/nrfx-blink-sdk>

- Before trying to use Swift with the Zephyr SDK, make sure your environment works and can build the provided C/C++ sample projects, in particular:
 - Try building and running the "simple/blink" example from Zephyr written in C.

Building

- Make sure you have a recent nightly Swift toolchain that has Embedded Swift support.
- Build the program in the Zephyr virtualenv, specify the nightly toolchain to be used via the `TOOLCHAINS` environment variable and the target board type via the `-DBOARD=...` CMake setting:

```
$ cd nrfx-blink-sdk
$ source ~/zephyrproject/.venv/bin/activate
(.venv) export TOOLCHAINS='<toolchain-name>'
(.venv) cmake -B build -G Ninja -DBOARD=nrf52840dk_nrf52840 -DUSE_CCACHE=0 .
(.venv) cmake --build build
```



Running

- Connect the nRF52840-DK board over a USB cable to your Mac using the J-Link connector on the board.
- Use `nrfjprog` to upload the firmware and to run it:

```
(.venv) nrfjprog --recover --program build/zephyr/zephyr.hex --verify
(.venv) nrfjprog --run
```



- The green LED should now be blinking in a pattern.

https://www.swift.org/download/#snapshots

swift.org

Snapshots

Trunk Development (main)

Development snapshots are prebuilt binaries that are automatically created from mainline development branches. These snapshots are not official releases. They have gone through automated unit testing, but they have not gone through the full testing that is performed for official releases.

Download	Date	Architecture	Docker Tag
Xcode	August 21, 2024	Universal Debugging Symbols	Unavailable
Ubuntu 18.04	April 13, 2024	x86_64 Signature (x86_64)	nightly-bionic
Ubuntu 20.04	August 21, 2024	x86_64 Signature (x86_64) aarch64 Signature (aarch64)	nightly-focal
Ubuntu 22.04	August 21, 2024	x86_64 Signature (x86_64) aarch64 Signature	nightly-jammy

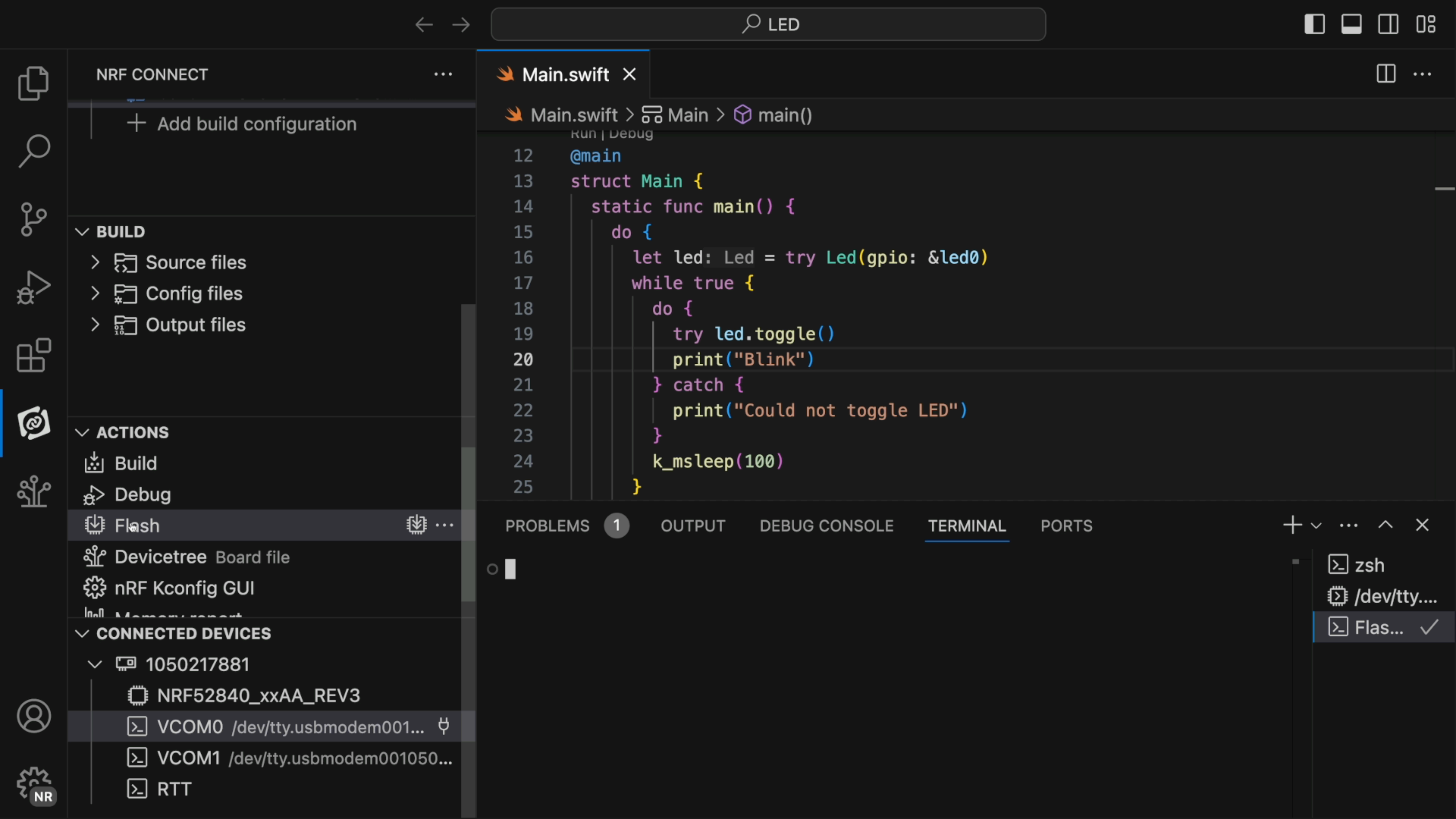


```
[ebariaux@eadu nrfx-blink-sdk % export TOOLCHAINS='org.swift.59202408071a'
[ebariaux@eadu nrfx-blink-sdk % cmake -B build -G Ninja -DBOARD=nrf52840dk_nrf52840 -DUSE_CCACHE=0 .
Loading Zephyr default modules (Freestanding).
-- Application: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk
-- CMake version: 3.30.2
-- Using NCS Toolchain 2.7.20240620.1065210518403 for building. (/opt/nordic/ncs/toolchains/f8037e9b83/cmake)
-- Found Python3: /opt/nordic/ncs/toolchains/f8037e9b83/bin/python3 (found suitable version "3.9.6", minimum required is "3.8") found components: Interpreter
-- Cache files will be written to: /Users/ebariaux/Library/Caches/zephyr
-- Zephyr version: 3.5.99 (/opt/nordic/ncs/v2.6.1/zephyr)
-- Found west (found suitable version "1.2.0", minimum required is "0.14.0")
-- Board: nrf52840dk_nrf52840
-- Found host-tools: zephyr 0.16.5 (/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk)
-- Found toolchain: zephyr 0.16.5 (/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk)
-- Found Dtc: /opt/nordic/ncs/toolchains/f8037e9b83/bin/dtc (found suitable version "1.6.1", minimum required is "1.4.6")
-- Found BOARD.dts: /opt/nordic/ncs/v2.6.1/zephyr/boards/arm/nrf52840dk_nrf52840/nrf52840dk_nrf52840.dts
-- Generated zephyr.dts: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/zephyr.dts
-- Generated devicetree_generated.h: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/include/generated/devicetree_generated.h
-- Including generated dts.cmake file: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/dts.cmake
Parsing /opt/nordic/ncs/v2.6.1/zephyr/Kconfig
Loaded configuration '/opt/nordic/ncs/v2.6.1/zephyr/boards/arm/nrf52840dk_nrf52840/nrf52840dk_nrf52840_defconfig'
Merged configuration '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/prj.conf'
Configuration saved to '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/.config'
Kconfig header saved to '/Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/zephyr/include/generated/autoconf.h'
-- Found GnuLd: /opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/../../lib/gcc/arm-zephyr-eabi/12.2.0/../../arm-zephyr-eabi/bin/ld.bfd (found version "2.38")
-- The C compiler identification is GNU 12.2.0
-- The CXX compiler identification is GNU 12.2.0
-- The ASM compiler identification is GNU
-- Found assembler: /opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/arm-zephyr-eabi-gcc
-- Using ccache: /opt/nordic/ncs/toolchains/f8037e9b83/bin/ccache
-- The Swift compiler identification is Apple 6.0
-- Configuring done (29.7s)
-- Generating done (0.1s)
-- Build files have been written to: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build
[ebariaux@eadu nrfx-blink-sdk % cmake --build build
[1/142] Preparing syscall dependency handling

[10/142] Generating include/generated/version.h
-- Zephyr version: 3.5.99 (/opt/nordic/ncs/v2.6.1/zephyr), build: 3758bcbfa5cd
[136/142] Linking Swift static library app/libapp.a
warning: /Applications/Xcode-16.0.0-Beta.6.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/ranlib: archive library: /Users/ebariaux/Documents/Nelcea/Presentations/SampleCode/swift-embedded-examples/nrfx-blink-sdk/build/app/libapp.a the table of contents is empty (no object file members in the library define global symbols)
[137/142] Linking C executable zephyr/zephyr_pre0.elf
/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/../../lib/gcc/arm-zephyr-eabi/12.2.0/../../arm-zephyr-eabi/bin/ld.bfd: warning: orphan section `swift_modhash' from `app/libapp.a(Main.swift.obj)' being placed in section `swift_modhash'
[142/142] Linking C executable zephyr/zephyr.elf
/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/../../lib/gcc/arm-zephyr-eabi/12.2.0/../../arm-zephyr-eabi/bin/ld.bfd: warning: orphan section `swift_modhash' from `app/libapp.a(Main.swift.obj)' being placed in section `swift_modhash'
Memory region      Used Size  Region Size  %age Used
      FLASH:      23116 B      1 MB      2.20%
      RAM:        7552 B      256 KB      2.88%
      IDT_LIST:      0 GB      32 KB      0.00%
[ebariaux@eadu nrfx-blink-sdk % nrfjprog --recover --program build/zephyr/zephyr.hex --verify
[ ##### ] 1.181s | Erase file - Done erasing
[ ##### ] 0.155s | Program file - Done programming
[ ##### ] 0.154s | Verify file - Done verifying
[ebariaux@eadu nrfx-blink-sdk % nrfjprog --run
Run.
[ebariaux@eadu nrfx-blink-sdk % █
```


Step 3

Set-up a proper dev environment



NRF CONNECT

+ Add build configuration

BUILD

- > Source files
- > Config files
- > Output files

ACTIONS

- Build
- Debug
- Flash

- Devicetree Board file
- nRF Kconfig GUI
- Memory report

CONNECTED DEVICES

- 1050217881
 - NRF52840_xxAA_REV3
 - VCOM0 /dev/tty.usbmodem001...
 - VCOM1 /dev/tty.usbmodem001050...
 - RTT

Main.swift

Main.swift > Main > main()

```
12 @main
13 struct Main {
14     static func main() {
15         do {
16             let led: Led = try Led(gpio: &led0)
17             while true {
18                 do {
19                     try led.toggle()
20                     print("Blink")
21                 } catch {
22                     print("Could not toggle LED")
23                 }
24                 k_msleep(100)
25             }
16 }
```

PROBLEMS 1

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

- zsh
- /dev/tty...
- Flas... ✓

Step 4

Start from the Embedded Swift examples and
create your own project from there

Main.swift

```
@main
struct Main {
    static func main() {
        // Note: & in Swift is not the "address of" operator, but on a global
        // variable declared in C
        // it will give the correct address of the global.
        gpio_pin_configure_dt(&led0, GPIO_OUTPUT | GPIO_OUTPUT_INIT_HIGH |
GPIO_OUTPUT_INIT_LOGICAL)
        while true {
            gpio_pin_toggle_dt(&led0)
            k_msleep(100)
        }
    }
}
```

BridgingHeader.h

```
#include <autoconf.h>
```

```
#include <zephyr/kernel.h>
```

```
#include <zephyr/drivers/gpio.h>
```

```
#define LED0_NODE DT_ALIAS(led0)
```

```
static struct gpio_dt_spec led0 = GPIO_DT_SPEC_GET(LED0_NODE, gpios);
```


Main.swift

```
@main
struct Main {
    static func main() {
        // Note: & in Swift is not the "address of" operator, but on a global
        // variable declared in C
        // it will give the correct address of the global.
        gpio_pin_configure_dt(&led0, GPIO_OUTPUT | GPIO_OUTPUT_INIT_HIGH |
GPIO_OUTPUT_INIT_LOGICAL)
        while true {
            gpio_pin_toggle_dt(&led0)
            k_msleep(100)
        }
    }
}
```

Main.swift

```
struct Led {
    let gpio: UnsafePointer<gpio_dt_spec>

    init(gpio: UnsafePointer<gpio_dt_spec>) {
        self.gpio = gpio
        // Note: & in Swift is not the "address of" operator, but on a global
variable declared in C
        // it will give the correct address of the global.
        gpio_pin_configure_dt(gpio, GPIO_OUTPUT | GPIO_OUTPUT_INIT_HIGH |
GPIO_OUTPUT_INIT_LOGICAL)
    }

    func toggle() {
        gpio_pin_toggle_dt(gpio)
    }
}
```

Main.swift

```
@main
struct Main {
    static func main() {
        let led = Led(gpio: &led0)
        while true {
            led.toggle()
            k_msleep(100)
        }
    }
}
```

Main.swift

```
enum LedError: Error {  
    case notReady  
}
```

```
struct Led {  
    let gpio: UnsafePointer<gpio_dt_spec>  
  
    init(gpio: UnsafePointer<gpio_dt_spec>) throws {  
        if (!gpio_is_ready_dt(gpio)) {  
            throw LedError.notReady  
        }  
        ...  
    }  
}
```

Build error

.../EmbeddedSwift-nRF52-Examples/LED/Main.swift:30:22: error: cannot use a value of protocol type 'any Error' in embedded Swift

```
28 |     init(gpio: UnsafePointer<gpio_dt_spec>) throws {
29 |         if (!gpio_is_ready_dt(gpio)) {
30 |             throw LedError.notReady
    |                                     - error: cannot use a value of protocol type 'any
Error' in embedded Swift
31 |         }
32 |
```

Main.swift

```
struct Led {  
    let gpio: UnsafePointer<gpio_dt_spec>  
  
    init(gpio: UnsafePointer<gpio_dt_spec>) throws(LedError) {  
        if (!gpio_is_ready_dt(gpio)) {  
            throw .notReady  
        }  
    }  
    ...  
}
```

Main.swift

```
@main
struct Main {
    static func main() {
        do {
            let led = try Led(gpio: &led0)
            while true {
                led.toggle()
                k_msleep(100)
            }
        } catch {
            print("Could not initialize LED")
        }
    }
}
```

Build error

```
[3/7] Linking C executable zephyr/zephyr_pre0.elf
FAILED: zephyr/zephyr_pre0.elf zephyr/zephyr_pre0.map .../EmbeddedSwift-nRF52-
Examples/LED/build/zephyr/zephyr_pre0.map
: && ccache /opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-
eabi/bin/arm-zephyr-eabi-gcc ..... -E true
/opt/nordic/ncs/toolchains/f8037e9b83/opt/zephyr-sdk/arm-zephyr-eabi/bin/../
lib/gcc/arm-zephyr-eabi/12.2.0/../../../../../arm-zephyr-eabi/bin/ld.bfd:
warning: orphan section `.swift_modhash' from `app/libapp.a(Main.swift.obj)'
being placed in section `.swift_modhash'
collect2: error: ld returned 1 exit status
ninja: build stopped: subcommand failed.
```


All BranchesShow Remote BranchesAncestor OrderCompact ViewJump to:

Description

Graph

Commit

Author

Date

Uncommitted changes

main

origin/main

origin/HEAD

Update the Pico2 example for more boards (#57)

Rename LED to Led (#53)

Improve README catalog (#52)

Add an RP2350 example project (#51)

nrfx-blink-sdk: Fix incompatible PIC setting and libc usage (#43)

stm32-uart-echo: Add a .sourcekit-lsp/config.json file to pass triple (and other flags) to SourceKit compilations (#47)

Adopt CMake 3.29's native Swift support in ESP32 examples (#41)

Stop using Xcode's libclang_rt.soft_static.a (#40)

Update README.md (#36)

Adds swift-picosystem-example to README (#35)

Switch library type from STATIC to OBJECT to fix the Zephyr sample

Update nrfx-blink-sdk example to use native CMake Swift support, require CMake 3.29

Update links to images/screenshots

Sorted by path

Search

nrfx-blink-sdk/CMakeLists.txt

nrfx-blink-sdk/prj.conf

nrfx-blink-sdk/Stubs.c

nrfx-blink-sdk: Fix incompatible PIC setting and libc usage (#43)

Commit: 9fa675115d412132f944fea9c9f392ead0b50f21 [9fa6751]

Parents: [d3a5697c77](#)

Author: Kuba (Brecka) Mracek <kuba@kubamracek.com>

Date: 16 August 2024 at 19:01:34 CEST

Committer: GitHub <noreply@github.com>

Labels: HEAD main

nrfx-blink-sdk/CMakeLists.txt

Hunk 1 : Lines 25-33

Reverse lines

2525

2626

2727

2828

2929

3030

2831

2932

3033

Use compacted C enums matching GCC

"\$<\$<COMPILE_LANGUAGE:Swift>:SHELL:-Xcc -fshort-enums"

+ # Disable PIC

+ "\$<\$<COMPILE_LANGUAGE:Swift>:SHELL:-Xcc -fno-pic"

+

Assortment of defines for Zephyr

"\$<\$<COMPILE_LANGUAGE:Swift>:SHELL:-Xcc -DKERNEL -Xcc -DNRF52840_XXAA -Xc

Hunk 2 : Lines 45-54

Reverse hunk

4245

4346

4447

4848

4949

5050

5151

4552

4653

4754

target_compile_options(app_swift PRIVATE

-parse-as-library

+ -Osize

+

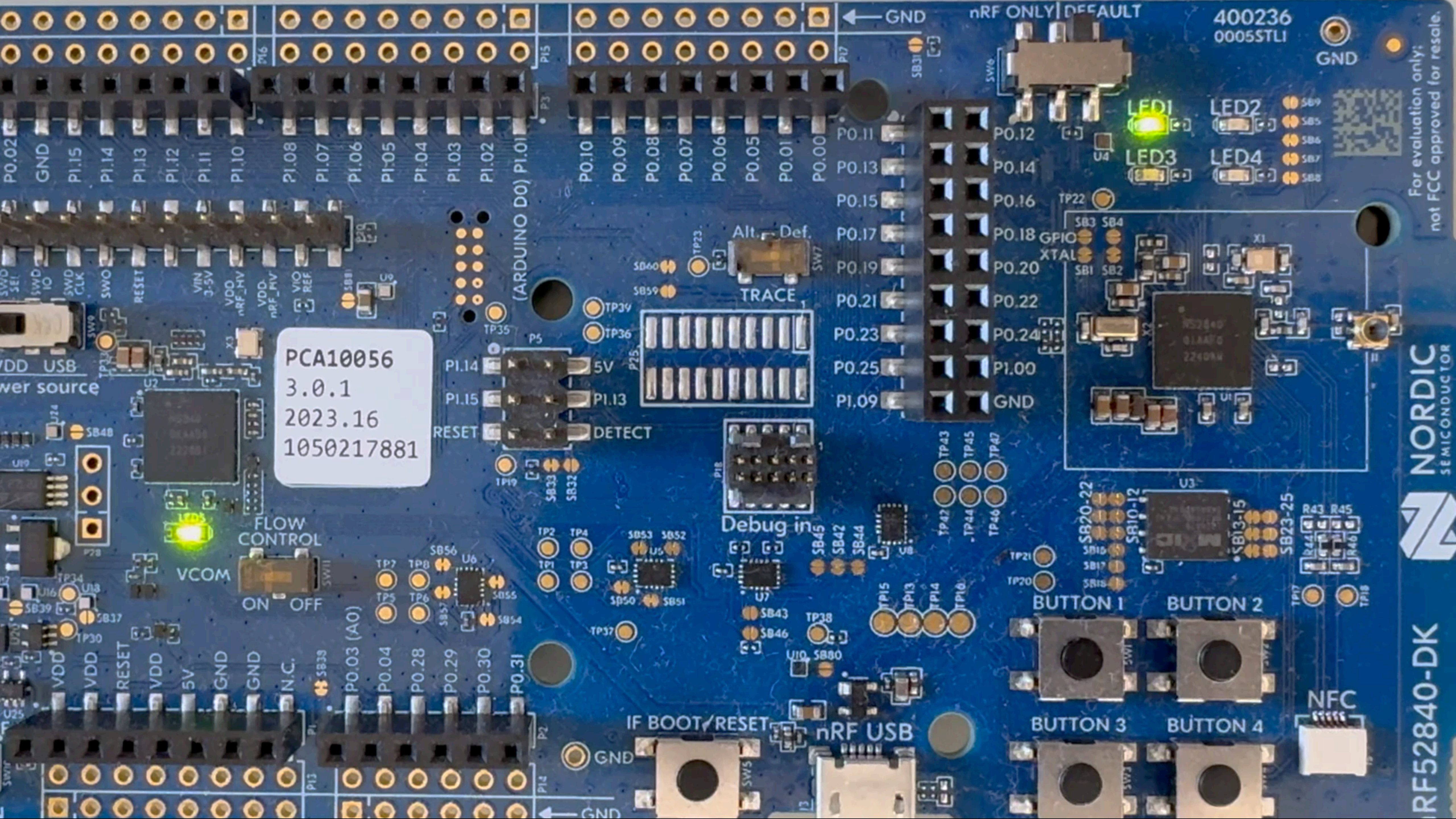
+ -Xfrontend -disable-stack-protector

+

FIXME: add dependency on BridgingHeader.h

-import-bridging-header \${CMAKE_CURRENT_LIST_DIR}/BridgingHeader.h

)



PCA10056
3.0.1
2023.16
1050217881

For evaluation only;
not FCC approved for resale.

NORDIC
SEMICONDUCTOR

nRF52840-DK

Step 4

Just keep going at it...

Eventually you will get there

```
@main
struct Main {
    static func main() {
        let led = Led(gpio: &led0)

        let _ = Button<Led>(gpio: &button, context: led) { _, callback, _ in
            let led = Button<Led>.getContext(callback)
            print("Button pressed")
            led.toggle()
        }

        let logic = Logic()
        logic.run()
    }
}
```

Summary

- Start with the toolchain for your chosen platform
- Have their example working, deployed to the board
- Download the Swift nightly toolchain (still in active development)
- Start from Embedded Swift examples
- Grow your code from there
- At this stage, you need some understanding of embedded dev, your target SDK, C...
- Use forums, community, GitHub...
- Still in development, things will improve
(check `swift/docs/EmbeddedSwift/EmbeddedSwiftStatus.md`)

Thank you!

