**JST: Smart Systems and Devices**
*Volume 31, Issue 2, September 2021, 019-026*

# Designing SPI to I2C Protocol Converter Base on ASIC Technology and Implementing on the FPGA Platform

***Nguyen Hoang Dung***
*Hanoi University of Science and Technology, Hanoi, Viet Nam*
*[*]Email: dung.nguyenhoang@hust.edu.vn*

**Abstract**

*Nowadays embedded systems are using a lot of different communication standards to transfer data such as USB, UART, SPI, I2C, etc. To be able to transfer data with each communication standard, the system needs at least one controller block for that communication standard. This has added to the complexity of the system and the cost of manufacturing hardware. Embedded systems only support SPI communication if desired, which can still be communicated with peripherals with I2C standard. However, the SPI cannot be directly connected to the I2C but must use a standard communication converter. This paper will primarily focus on designing an IP core communication standard converter from SPI to I2C using APB (Advanced Peripheral Bus) communication as one of the AMBA (Advanced Microcontroller Bus Architecture) communication sets. In particular, APB is a bus used to communicate with peripherals that do not require fast processing speeds such as UART, SPI, I2C, etc.*

Keywords: SPI, I2C, FPGA.

## 1. Introduction

Serial Peripheral Interface (SPI) is one of the most widely used interfaces between microcontrollers and peripherals such as sensors, analog-to-digital converters (ADC), and digital-to-analog converters (DAC), shift register or SRAM. Whereas Inter Integrated Circuit (I2C) is a two-wire bus used to enable communication between two or more devices on the same system. Let's assume an embedded system having only the SPI communication standard has to interface with peripherals that only use the I2C communication standard, next, by using design flow of ASIC (Application-Specific Integrated Circuit) and FPGA (Field-Programmable Gate Array) technologies, a SPI to I2C communication converter is created. Previously published researches such as Design of SPI to I2C Bridge for High Speed Data Interfacing in Digital System [1] and FPGA Implementation of Serial Protocol using SPI and I2C [2] have some limitations such as: Cannot transmit data correctly if SPI device and I2C device operate in different frequency domains; can't use bus to communicate with the system (APB, AHB,..); only can works with an initial pre configuration; the maximum transfer rate of the I2C interface is only 3.4 Mbit/s. Therefore, the research team proposed to design an SPI to I2C protocol converter based on ASIC technology and deployed it on the FPGA platform to solve these above problems. The structure of the paper is organized as follows: In part 2, the block diagram design of the SPI to I2C communication converter will be presented; Part 3 and part 4 bring up with testing and synthesis of design and implementation on the FPGA platform; Part 5 is the conclusion of the paper and the research team's next development direction on the design of SPI to I2C protocol converter.

## 2. Block Diagram Design

### *2.1. Overview of the Converter*

A protocol converter is a device that is used to convert the standard communication (e.g. SPI) of one device to a suitable communication (e.g. I2C) with another device to have the ability to transmit data between two communication standards. Fig. 1 shows the block diagram of the system SPI to I2C protocol converter.

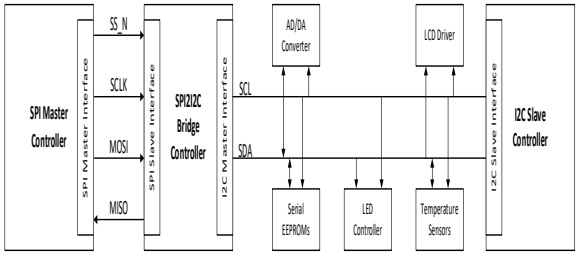


Fig. 1. Connecting an SPI to I2C converter in a system.

### *2.2. Serial Peripheral Interface*

SPI is a synchronous serial communication protocol that is commonly used at short distances, mostly in embedded systems between microprocessors or microcontrollers and peripherals like sensors, ADC, DAC, SRAM, and so on. SPI devices communicate in

ISSN: 2734-9373
https://doi.org/10.51316/jst.152.ssad.2021.31.2.3
Received: December 03, 2020; accepted: January 26, 2021

the duplex-mode with only one single master device and many slave devices. During communication, the master can only pick a slave-based device by dragging the device's Slave Select (SS N) signal pin to a low level[3]. Fig. 2 illustrates SPI communication.

### *2.3. Inter Integration Circuit*

Nowadays, in the communication protocols operating with low frequency, I2C (Inter Integrated Circuit) protocol is very suitable for communication between integrated circuits, slow communication with peripheral devices. Fig. 3 shows a transaction of the I2C interface standard.

For operation mode with 7-bit slave address, the master device will send a start condition, followed immediately by the 7-bit slave address which it wants to communicate with and finally a bit representing a write or read from the slave (bit 0 is written and bit 1 is read). If the slave exists on the bus, the slave will respond with the ACK bit (low-level to confirm) for its address. After that, the master will then be in read mode or write mode based on the previously sent written bit or read a bit [4]. The address and data bytes are sent with the MSB bit first. Starting conditions occur when the SDA pulls from high-level to low-level when the SCL is high and stopping condition is represented by the SDA transition from low-level to high-level when the SCL is high.

### *2.4. Block Diagram*

Some previous architecture was given as follows: The SPI to I2C converter will contain a SPI interface as the slave and a I2C interface as the master. All request signals will be generated by the SPI master controller block. When an external SPI master wants to read or write data to an I2C slave peripheral, it sends a read command or write command and the corresponding address of the I2C slave through the internal SPI slave inside the converter. Thereon, the SPI slave will then send commands, address, and data (in case of writing) to the I2C master, and the internal state machine will perform the necessary data conversion.

These designs will only accommodate data transmission with a clock domain, from there proposing two more FIFO buffers (First in, first out) called TX FIFO (receiving data from the SPI-bus side) and RX FIFO (receiving data from the I2C-bus side) for the purpose of storing pending data. Next, in order to communicate with the CPU and the user can configure the converter, it is proposed to design a register block that is used to configure input values and store input signal information. Output of the circuit. This work is done through an advanced peripheral communication standard APB (Advanced Peripheral Bus) used to communicate with peripherals that do not require fast processing speed such as UART, SPI, I2C, etc. In addition, the converter will have additional interrupt pins to respond to the current state of the circuit. Fig. 4 shows the general block diagram of the SPI to I2C converter.

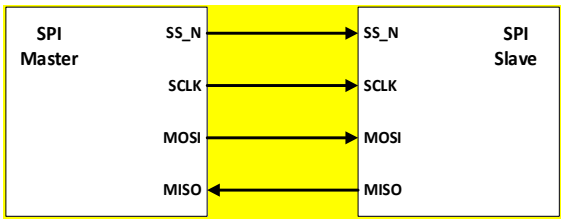


Fig. 2. SPI Interface

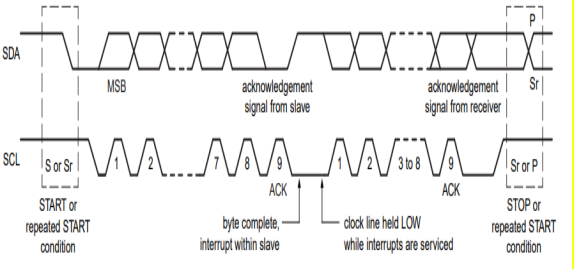


Fig. 3. A transaction of a standard I2C interface.

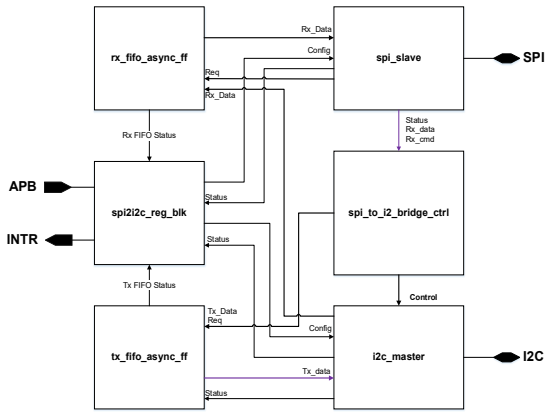


Fig. 4. Block diagram of SPI to I2C converter.

The blocks in Fig. 4 are shown and have the following meanings**: (1) spi2i2c_reg_blk**: Is the block used to configure input values and store information about the active state of the converter; **(2) spi_slave**: Is a block designed to be equivalent as a simple SPI device. This block implements the SPI interface with the external SPI master; **(3) i2c_master**: Is a block designed to be equivalent as a simple SPI device. This block implements I2C interface for data transmission I2C slave devices; **(4) spi_to_i2c_bridge_ctrl**: Is the main processing block of the converter. This block controls the transmission and reception of data from two SPI slave block and I2C master block; **(5) tx_fifo_async_ff**: Is an asynchronous buffer, which means that the write and read times on the watch may vary, and thus is commonly employed as a buffer data storage devices. This block retains data that has been transmitted from an external SPI master and is waiting to be processed before being delivered to the I2C-bus;

**(6) rx_fifo_async_ff**: This block stores data transmitted from external I2C slave devices waiting to be processed to send those data to the SPI-bus.

### *2.5. Operating Principle*

In order for the SPI master to communicate with the I2C slave devices, the research team designed a set of commands that can be customized according to the previous convention between the two sides of the SPI master and the converter, and then send those commands on the SPI-bus. Most SPI devices currently operate data within a transaction (i.e. every time SS_N is active) whose number of bits transmitted must be a multiple of 8 from which the total width of a transaction on the SPI-bus Designed so that the number of bits is divisible by 8, this converter has an instruction width of 1 byte so that the SPI master can transmit properly. The SPI master commands that can be transmitted on the SPI-bus are listed in Table 1. The following is the data frame for the commands in the table:

#### *2.5.1. Data transfer to slave I2C peripheral device (0x01)*

When the SPI master pulls the SS_N signal pin a low level, the data received from the external SPI master (SCLK) clocked MOSI pin is processed by the SPI slave block in the converter and then forwarded to the processing block to add the elements of the I2C interface standard as the start bit, the address bits, the ACK bit (controlled by the I2C slave) and the stop bit. The SPI master will send a command to the converter in the following order of bytes: 1 byte of writting instruction (0x01), 1 byte of data for the number of bytes that the SPI master wants to transmit (maximum 256 bytes), 1 byte of data for the address of the I2C slave and finally the data transmitted to those I2C devices. The data stream sent on the SPI-bus when the write command (0x01) is executed is shown as shown in Fig. 5.

The data after being filtered by the SPI slave block continues to be processed by the processor block (spi_to_i2c_bridge_ctrl) and sent to the I2C master block for transmission to the I2C slave devices. Fig. 6 shows the data stream sent on the I2C-bus when the SPI master executes a write command (0x01).

#### *2.5.2. Receive data from slave I2C peripheral device*

The data received from the I2C slave devices are processed by the internal I2C master block by removing the I2C interface standard bits which are the start bit, the address bits, the ACK bit and the stop bit. The received data is stored in the RX FIFO memory and will not be returned to the external SPI master without executing the buffer read (0x03) instruction described in the next section. To receive data the SPI master will send a command to the I2C slave device with the order of bytes sent as follows: 1 byte of the read command (0x02), 1 byte of data for the number of bytes that the SPI master wants to receive (maximum 256 bytes) and 1 byte of data for the address of the I2C slave device. Fig. 7 shows the data stream sent on the SPI-bus when the SPI master performs a read command (0x02).

Table 1. The commands the SPI master can send on the SPI-bus.

| No | Command | Description |
|---|---|---|
| 1 | 0x01 | Write N bytes to device I2C |
| 2 | 0x02 | Read N bytes from I2C device |
| 3 | 0x03 | Read buffer command. |

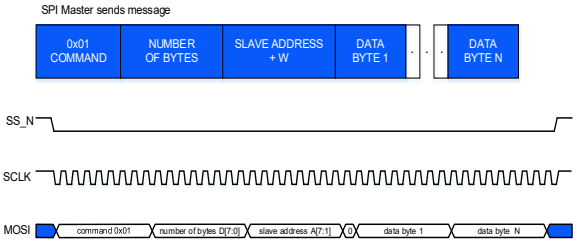


Fig. 5. Data is transferred over the SPI-bus when a write signal is issued, (0x01).

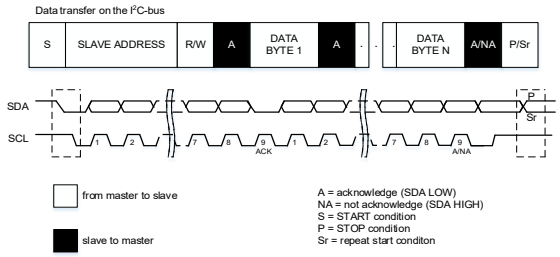


Fig. 6. Data is transferred across the I2C-bus when the SPI master issues a write signal, (0x01).

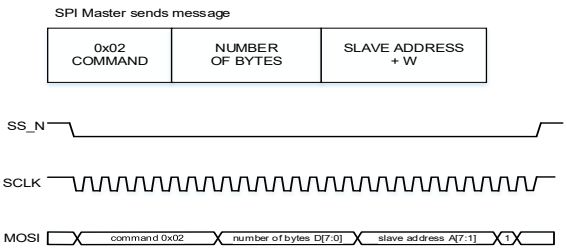


Fig. 7. The data stream sent on the SPI-bus when the SPI master performs a read command (0x02).

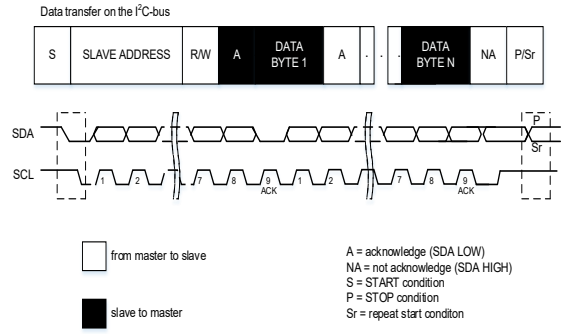


Fig. 8. The data stream sent on the I2C-bus when the SPI master performs a read command (0x02).

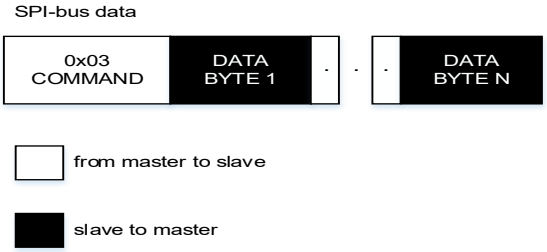


Fig. 9. The data stream sent on the SPI-bus when the SPI master performs a buffer read (0x03).

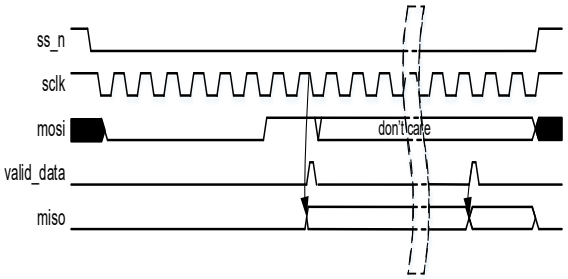


Fig. 10. The data takes effect when the SPI master reads the buffer,

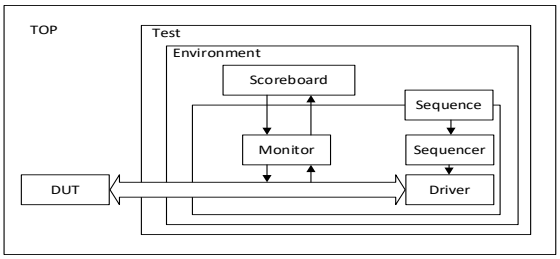


Fig. 11. Basic structure of UVM environment.

The data after being filtered by the SPI slave block is further processed by the processing block (spi_to_i2c_bridge_ctrl) and sent to the I2C master block for transmission to the I2C slave devices and then receiving the returned data. Fig. 8 shows the data stream sent on the I2C-bus when the SPI master performs a read (0x02).

*2.5.3. Read buffer*

After the SPI master executes a buffer read (0x03) command from the RX FIFO buffer, the data being stored in the buffer will be returned to the SPI master via the MISO signal pin. Fig. 9 shows the data stream sent on the SPI-bus when the SPI master performs a buffer read (0x03).

Because the speed of operation of the SPI and I2C communication protocols differs, data received from the I2C device can be delivered at any moment because the SS_N pin is pulled to a low level and the SPI master cannot identify when the data takes effect, Read buffer command (0x03) is created. As a result, the buffer reading command was designed to address this issue. The data in effect when the SPI Master executes the buffer reading instruction is shown in Fig. 10.

## 3. Design Verification and Synthesis.

### *3.1. Design Testing.*

With an RTL (Register-Transfer Level) design according to ASIC (Application-Specific Integrated Circuit) technology, before the product is packaged, it is critical to test the design. Building a verification environment has become an essential phase in the product design process that cannot be overlooked in order to ensure that a product works properly under real-world settings. The UVM (Universal Verification Methodology) library and QuestaSim software for design verification are two of the most successful verification approaches for ASIC-based designs [5].

*3.1.1. Introduction to UVM library*

UVM (Universal Verification Methodology) is a method of simulating hardware designs. The UVM library is a collection of classes and methods built on the System Verilog language. UVM was built with the goal of creating a common simulation method for the IC industry. Simulation testing is a very time-consuming process. In particular, the time spent building and editing the simulation environment is quite large. Without a common methodology, a simulation environment will be difficult to reuse, extend and understand by many different engineers, organizations or companies.

The basic structure of a UVM environment consisting of blocks is shown in Fig. 11. The blocks in Fig. 11 are described as follows: (1) Top is the element that contains the entire test environment, including the DUT; (2) Test is the top layer containing all UVM components; (3) DUT (Design Under Test) is the design to be tested; (4) Environment is a grouping

element of other UVM environmental components such as Agent and Scoreboard; (5) Agent groups the components that test the connection to the communication of the DUT. It can have many different components, but usually has 3 main components: Monitor is the component that monitors information and communication signals with the DUT, Sequencer controls the transmission of transactions from sequences, and Driver receives transactions. From the sequences is to convert them in turn to the corresponding values to drive the signals for communication with the DUT; (6) Scoreboard is the component that examines the desired behavior, activity or data of the DUT and (7) Sequence is the object that generates transactions and provides them to the UVM Sequencer to deliver to the Driver.

*3.1.2. Build testbench structure*

Fig. 12 shows the block structure of UVM for design testing. The UVM class library brings a lot of automation to the SystemVerilog hardware description language such as sequencing and data automation features (packing, copying, comparing), etc. After building the structure testbench as shown in Fig. 11, perform design testing on QuestaSim software. Imagine that when a transaction (0x01) is executed, the data flow sent via the SPI-bus consists of the following 24 bits of commands: Write 0x01 with 8 bits of commands, 8 bits of transferred bytes, and 8 bits of I2C address. The SPI master will then send the actual data on the I2C bus (in this example is 4 bytes of data). On the SPI-bus, the operating frequency is 10 MHz, and I2C is in Fast-mode Plus (1 MHz). With the above operating frequency, the minimum required depth of TX FIFO is 10. Fig. 13 shows the design simulation results on the SPI-bus data input side. After receiving the input data from the SPI with the request to perform data transmission to the I2C slave device, the I2C interface of the converter will have the waveform shown in Fig. 14.

The results after simulation are shown in Fig. 13 and Fig. 14, when directly observed, it can be seen that the results are correct compared to the requirements set forth. But for the convenience of design testing, QuestaSim software with the use of the UVM library will generate a report, helping the designer to determine the correct operation and coverage of the design, thereby can create input plans and goals for parts of the design. The results when running the test cases will be saved by UVM and extracted to a coverage report generated on Questa software with the results as shown in Fig. 15. The important information of the coverage report includes:

- dti_spi_to_i2c_top (87.83%): is the coverage result of code RTL coverage. It measures how much of the RTL code has been executed through all the test cases. This includes implementing design blocks, line numbers, conditions, FSM, toggles, and paths. UVM will automatically extract code coverage from the RTL code.

- dti_spi2i2c_sva (100%): is the result of checking the correctness of the functions and timing required in the design.

- dti_cvg (100%): shows that all items in the test plan for design features have been tested.

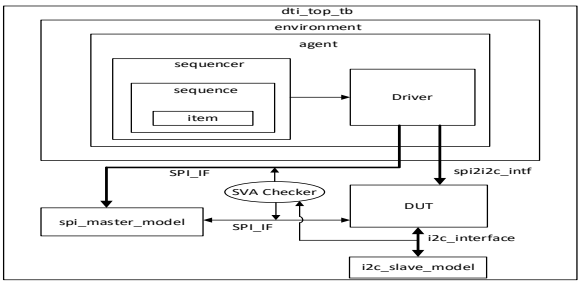


Fig. 12. Testbench structure using UVM to test the design.

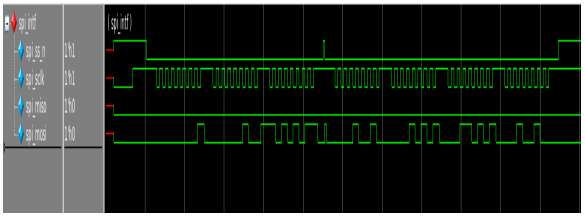

Fig. 13. Simulation results of the converter on the SPI data input.

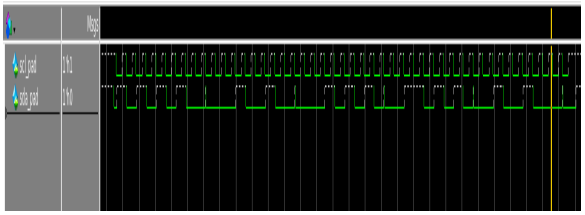

Fig. 14. Simulation results of data transmission on I2C-bus.

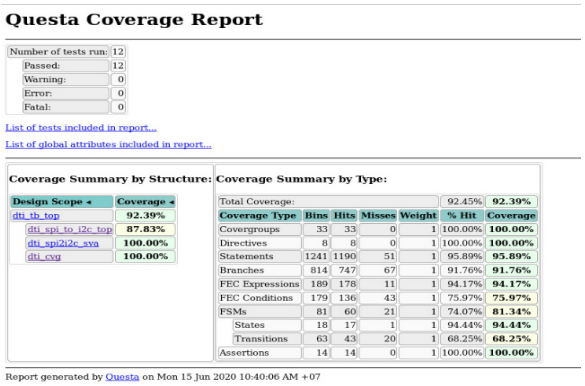


Fig. 15. Coverage report of the design on QuestaSim software.

Each verified design requires coverage metrics to gauge performance and help determine when the design is good enough to stop. In this report the dti_tb_top information reaches 92.39% indicating that design testing can be stopped to make the move to the next steps of the ASIC design flow.

### *3.2. Design Synthesis*

Advanced techniques are used for ASIC chip synthesis, physics synthesis, static timing testing and analysis, optimization, dynamic simulation, formal verification, DFT scan insertion, link to layout, physical synthesis and static timing analysis using the Design Compiler tool of Synopsys [6]. Configure the software with an operating frequency of 150 MHz with optimal options for timing, area and power. To be able to synthesize, in addition to the RTL design, a library of standard cell is required, the library used for synthesis in this design is a standard cell library with 40 nm technology, -40 ºC operating temperature (which is the condition where standard cells work the worst) and the voltage used is 0.99 V. The important results are area (area) and time (timing) in the design synthesis are shown as follows:

*3.2.1 Area*

```
Number of ports:                          108
Number of nets:                          6040
Number of cells:                         5957
Number of combinational cells:           4394
Number of sequential cells:              1563
Number of macros/black boxes:               0
Number of buf/inv:                        999
Number of references:                      67
Combinational area:               5096.587949
Buf/Inv area:                      587.608000
Noncombinational area:            6409.591938
Macro/Black Box area:                0.000000
Net Interconnect area:   undefined  (No wire load specified)
Total cell area:                 11506.179887
```

Area results including ports, nets, cells, combined, non-combined and total area reports are shown in the report. The important information shown in the area report is that the number of combinational cells is 4394, the number of sequential cells is 1563, and the total design area is 11506.179887 ($\mu m^2$).

*3.2.2. Timing*

Slack is a type of timing in the Static Timing Analysis of techniques for testing designs in terms of time. Slack includes setup slack and hold slack, defined as the difference between the actual or achieved time and the desired time for a timing path. Slack timing path helps designers determine their design can work with the required frequency or not. Below is a part of the report with the part cited as the path with the lowest slack line. The important information shown in the report is the slack parameter, if the result of an aggregate is the result of slack (MET) means that the value of slack is a non-negative value and the design can work with the frequency considered, otherwise slack (VIOLATED) means that the design value of slack is a negative value and the design cannot work with the frequency considered. In this report, the slack (MET) is equal to 0.00030 ns (positive), which means the design can work with the original specified frequency of 150 MHz.

```
Startpoint:dti_spi2i2c_reg_blk/rb_regs/iscl_scl_low_cnt_reg[0](rising
edge-triggered flip-flop clocked by pclk)
Endpoint:
dti_spi_to_i2c/dti_i2c_core/i2c_master_ctrl_inst/master_i2c_bit_cnt_inst
/count_reg_reg[2](rising edge-triggered flip-flop clocked by clk)
Path Group: clk
Path Type: max
clock clk (rise edge)                                  45.00000   45.00000
  clock network delay (ideal)                           0.00000   45.00000
  clock uncertainty                                     3.00000   42.00000
dti_spi_to_i2c/dti_i2c_core/i2c_master_ctrl_inst/master_i2c_bit_cnt_inst
/count_reg_reg[2]/CK                                            (dti_ffqa01x1)
0.00000   42.00000 r
  library setup time                                    0.15634   41.84366
  data required time                                              41.84366
  slack(MET)                                                    0.00030
  normalization delay                                           5.00000
  normalized slack                                              0.00006
```

## 4. Design Implementation on FPGA

### *4.1. Deployment Block Diagram on Xilinx FPGA Platform*

The I2C and I2S interfaces are designed to control the on-board audio codec to output audio signals under this article, which is built on the Xilinx FPGA platform. Before being transferred to the codec, audio data is stored in the FPGA's internal RAM block. The SPI2I2C controller is the decoder configured via the I2C interface. Digital audio data can be sent across the DAC interface of the codec using the I2S controller once it is configured and ready. Thereafter, the digital audio data is fed through the WM8731 Audio Codec for conversion to analog signals and output to multimedia devices [7]. The block diagram based on the Xilinx FPGA platform is shown in Fig. 16.

In order for the implementation to be designed on FPGA, the selected FPGA platform should have at least the following parameters: (1) System frequency: 200 MHz; (2) controllers: I2C, I2S; (3) Support APB Interface; (4) Software design support. From there, using the Xilinx Kintex Ultra-Scale FPGA platform is able to meet the listed requirements and is suitable for deploying IPs of great complexity.

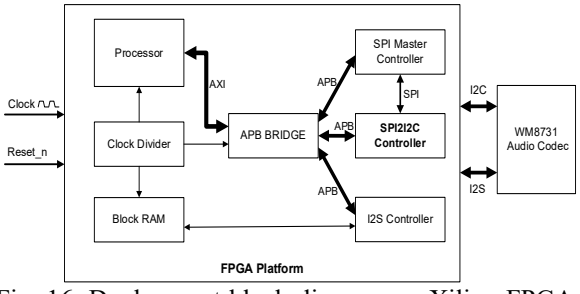


Fig. 16. Deployment block diagram on Xilinx FPGA platform.

### *4.2. Work*

The BRAM block carries the original data in the coe file format. Furthermore, using the I2C interface generated by the SPI2I2C converter to configure the internal registers of the WM8731 Audio Codec, assign the registers inside the IPs to SPI Master, SPI2I2C, and I2S. After configuring the IP register configuration, allow the I2S SCK clock to synchronize data between

I2S and WM8731. The values inside the BRAM are read out and recorded in the FIFO memory, and the step was repeated until the BRAM signals the data to be empty. The operational flow of the design implemented on the Xilinx FPGA can be seen in Fig. 17.

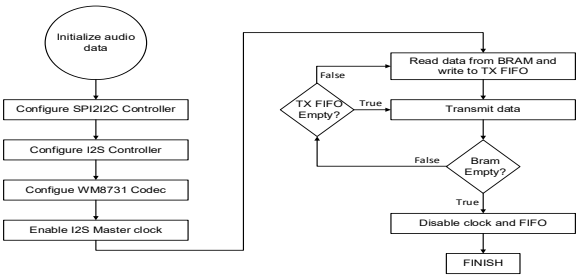


Fig. 17. The operational flow of the design implemented on the Xilinx FPGA.

### *4.3. Synthesized Results on FPGA.*

The following synthesis results were generated by the converter using Xilinx Vivado 2018.1 software with a configured operating frequency of 150 Mhz.

#### *4.3.1. CLB Logic*

Club Logic is a critical component of FPGA technology. Reconfigurable logic gates may be created using logic blocks. The most typical FPGA architecture is logic blocks, which are commonly arranged in an array. I/Os (to communicate with external signals) and routing channels are required by logic blocks (to connect logic blocks). A logic block is made up of many logic cells (called ALM, LE, Slice, ...). One cell consists of a 4-input LUT, a Full Adder, and a D-type flip-flop. Below is the result of the synthesized logic CLB. The results show that the number of CLB LUTs used accounted for only 0.24%, CLB Register accounted for 0.15%, which means that the design takes up very little resources of the FPGA.

| Site Type | Used | Fixed | Available | Util% |
|---|---|---|---|---|
| **CLB LUTs*** | **1264** | **0** | **537600** | **0.24** |
| LUT as Logic | 1264 | 0 | 537600 | 0.24 |
| LUT as Memory | 0 | 0 | 76800 | 0.00 |
| **CLB Registers** | **1570** | **0** | **1075200** | **0.15** |
| Register as Flip Flop | 1570 | 0 | 1075200 | 0.15 |
| Register as Latch | 0 | 0 | 1075200 | 0.00 |
| CARRY8 | 14 | 0 | 67200 | 0.02 |
| F7 Muxes | 100 | 0 | 268800 | 0.04 |
| F8 Muxes | 9 | 0 | 134400 | <0.01 |
| F9 Muxes | 0 | 0 | 67200 | 0.00 |

#### *4.3.2. Primitives*

A resource in an FPGA that is directly recognized by the implementation software and typically corresponds to a logical resource in a logic resource (e.g. I/O pin, buffer, logic gate, or flip-flop). The report below shows the average number of Primitives used compared to a small IP core deployed on an FPGA.

| Ref Name | Used | Functional Category |
|---|---|---|
| FDRE | 800 | Register |
| LUT6 | 744 | CLB |
| FDCE | 741 | Register |
| LUT4 | 212 | CLB |
| LUT2 | 202 | CLB |
| LUT5 | 167 | CLB |
| LUT3 | 142 | CLB |
| MUXF7 | 100 | CLB |
| INBUF | 52 | I/O |
| IBUFCTRL | 52 | Others |
| OBUF | 49 | I/O |
| FDPE | 29 | Register |
| CARRY8 | 14 | CLB |
| LUT1 | 11 | CLB |
| MUXF8 | 9 | CLB |
| OBUFT | 2 | I/O |
| BUFGCE | 2 | Clock |

## 5. Discussion Results.

Table 2 compares the design features of the converter by the design team with previously published papers such as Design of SPI to I2C Bridge for High-Speed Data Interfacing in Digital System of IJARCCE, FPGA Implementation of Serial Protocol using SPI and I2C of IJEEE and industrial products of other companies such as Silicon Labs' CP2120 [8] or NXP's SC18IS600 [9]. The current design of the research team solved the problems posed in the introduction. Table 2 shows that SPI mode can work with all modes of SPI protocol, so it can communicate with all systems which have SPI bus. The design uses the APB peripheral communication bus with a faster register configuration speed than using the SPI bus directly for configuration like Silicon Labs's product.

Today, technology products are increasingly developed, the speed of system communication must also increase to meet the needs. Therefore, the research team has considered improving the speed of SPI and I2C interface standards in the design.

Table 2. Compare and contrast the design aspects of various converters.

| **Features** | **IJARCCE [1]** | **IJERT [2]** | **Silicon Labs [8]** (CP2120) | **NXP [9]** (SC18IS600) | **dti_spi2i2c** (Our Design) |
|---|---|---|---|---|---|
| Mode SPI | Not mention. | Not mention. | Mode 3 | Mode 3 | Mode 0, 1, 2, 3 |
| SPI data transmission direction | Not mention. | Not mention. | LSB or MSB | LSB or MSB | MSB |
| Register configuration | Can't configurable. | Can't configurable. | Using SPI bus | Using SPI bus | Using APB bus |
| SPI speed | Not mention. | Not mention. | 1 Mbit/s | 1.2 Mbit/s | 25 Mbit/s |
| I2C max speed | 3.4 Mbit/s | 3.4 Mbit/s | 400 kbit/s | 400 kbit/s | 5 Mbit/s |
| Buffer | Not | Not | 256 bytes | 96 bytes | 256 bytes |

The speed of the SPI interface is also significantly improved from 1.2 Mbit/s up to 25 Mbit/s by making this design works well with most of the systems which using the SPI interface standard available on the market. In addition, the speed of the I2C interface is also improved to a maximum of 5 Mbit/s.

Because of the speed disparity between SPI and I2C, a buffer is required to hold data received from the two interfaces. The buffer's breadth grows in proportion to the difference between the two values. This design solves the problem of the performance gap between the two communication protocols being too big by using a 256-byte buffer, however, this increases the hardware cost of the design.

## 6. Conclusion

The converter designed by the research team can work with all the features proposed by the founders of these two communication standards that some other industrial companies do not have.

The protocol converter design is proposed to be implemented in both ASIC and FPGA technologies, according to the study. The protocol converter has more features than the previous mentioned designs in terms of design. For example, to compensate for the speed differential between the two communication standards, a set of two FIFO buffers has been incorporated to the architecture. Additionally, the SPI and I2C bus speeds have been enhanced (25 MHz SPI-bus and 5 MHz I2C-bus). Furthermore, the design is capable of operating at a high frequency of up to 150 MHz.

## References

[1]. Sidra Anam and Vinod Kapse, Design of SPI to I2C bridge for high speed data interfacing in digital system, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 5, Issue 4, April 2016.

[2]. Saniya Farheen and Dr Baswaraj Gadgay, FPGA implementation of serial protocol using SPI and I2C, International Journal of Ethics in Engineering & Management Education (IJEEE), Volume 2, Issue 6, June 2015.

[3]. Sanjeeb Mishra and Vijayakrishnan Rousseau, System on chip interfaces for low power design, Elsevier Inc, 2016, pp. 239-243. https://doi.org/10.1016/C2014-0-00336-X

[4]. Sanjeeb Mishra and Vijayakrishnan Rousseau, System on chip interfaces for low power design, Elsevier Inc, 2016, pp. 243-247.

[5]. Uvm Cookbook Complete Verification Academy, 1rd, Menter Graphics, 2017, pp. 1-4.

[6]. S. Gayathri and T. C. Taranath, RTL synthesis of case study using design compiler, in Proc. 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), 2017, pp. 1-7, https://doi.org/10.1109/ICEECCOT.2017.8284603.

[7]. Peter Athanas, Dionisios Pnevmatikatos and Nicolas Sklavos, Embedded systems design with FPGAs, Springer, 2015, pp. 1-4.

[8]. SPI To I2c Bridge And Gpio Port Expander, 1rd, Silicon Labs, 2015.

[9]. SPI to I2C-bus Interface, 8rd, NXP Semiconductors, 2019.