Test Automation of Inter-Integrated Circuit (I2C) Validation using Hardware Software Approach

by

Akash Panchal

B.Eng. Gujarat Technological University, 2014

A Report Submitted in Partial Fulfillment of the Requirements for

the Degree of

MASTER OF ENGINEERING

in the Department of Electrical and Computer Engineering

Supervisory Committee

Test Automation of Inter-Integrated Circuit (I2C) Validation using Hardware Software Approach

by

Akash Panchal

B.Eng., Gujarat Technological University, 2014

Supervisory Committee

Dr. Harry H.L. Kwok (Department of Electrical and Computer Engineering)

Supervisor

Dr. Ashoka K.S. Bhat (Department of Electrical and Computer Engineering)

Departmental Member

Abstract

The Inter-Integrated Circuit(I2C) is a serial communication protocol used to enable communication between two or more devices. Modern-day semiconductor devices require thorough validation prior to its release, which captures the development and execution of the test plan. With the ever-increasing complexity of semiconductor devices, validation efforts can be more time consuming and costly. Manual execution of hardware test cases which requires a lot of effort raises questions on its reliability. To save time and efforts along with ensuring reliability, test automation can be an effective solution that fulfills the necessity of hardware validation in a short span of time. The primary objective of the project is to develop a test automation by using Python scripting to confirm the operation of Inter-Integrated Circuit (I2C) protocol between the I2C controller of Spartan®-6 FPGA and M24C08 EEPROM. The need of validating I2C controller of a control device manually is eliminated by the proposed test automation. Such test automation can be proven extremely useful in the case of several PCBs requiring I2C protocol validation in a short span of time ensuring high reliability. Spartan®-6 evaluation board SP605 is used to execute the test cases as a hardware platform. Proposed test automation aims to reduce validation efforts required for a semiconductor device by offering advantages such as reusability and high reliability.

Table of Contents

List of Figures

List of Tables

List of Acronyms

FPGA	Field Programmable Gate Array
SoC	System on Chip
SPI	Serial Peripheral Interface
I2C	Inter-Integrated Circuit
UART	Universal Asynchronous Receiver Transmitter
PCB	Printed Circuit Board
IC	Integrated Circuits
EEPROM	Electrically Erasable Programmable Read Only Memory
ADC	Analog to Digital Converter

Acknowledgments

I would like to express my sincere gratitude to my supervisor Dr. Harry Kwok for his constant guidance, support, patience and encouragement throughout my degree. His expertise and knowledge helped me to expand the horizons of my knowledge. This project would not have been possible without his guidance. I would also like to thank all my friends for their constant support throughout my graduate studies at the University of Victoria.

Finally, I would like to express my thanks and love to my parents, brother, and wife for their selfless love, guidance, and support.

Dedication

To my parents, my father Mr. Dilipkumar Panchal and mother, Alka Panchal for allowing me to achieve what I desire for with their constant support and motivation.

To my brother, Darshak & To my sister, Prushthi for their never-ending support.

To my wife, Zarana for her encouragement and support.

Chapter 1: Introduction

This chapter describes the context and goal of the project along with the proposed methodology for the work. It discusses the proposed work at length. At last, it presents an outline of the project report.

1.1 Context

Conventionally, any IC design validation consumes approximately 70% time of the total design cycle. It includes the design and development of hardware and software to execute test cases. However, the record shows that even after spending this much of time and efforts, designs may not be fully correct [1]. Reports also suggest that one out of two chip designs require nearly two tape-outs before its market release [2]. Two tape-out indicates a re-spin of a chip due to logic error. Re-spin delays the product to reach the market with an increase in engineering cost. Thus, validation of the chip design becomes extremely essential as the complexity of the chip design grows.

Electronic circuit boards popularly known as Printed Circuit Boards (PCBs) are designed and developed to support the hardware development life cycle for validation of the design of the SoC (System On Chip) or FPGA (Field Programmable Gate Array) . The complexity of these boards varies from one application to other. Modern-day PCB design incorporates numerous mixed-signal components. Meaning, analog and digital devices can be embedded on a single discrete circuit board. While resistor, capacitor, inductor, transistor form analog part, digital ICs, FPGA and other programmable components build the digital circuitry.

Components such as a microcontroller, FPGA, SoC on the PCBs require the prior configuration and programming to bring their features up before validation begins. Many of the control devices described above incorporate interfaces to support protocols such as I2C, SPI, and UART. Such protocols enable its compatible devices to communicate with control devices. The functionality of such protocol interfaces existing inside a control device must be validated prior to its release.

Functional validation of these interfaces of a control device can be achieved by determining their electrical correctness on the PCBs using equipment such as oscilloscope and function generator.

Such a conventional approach can be proven to be a good option if the design of a PCB is less complex and small in numbers. The complexity of a PCB refers to the number of control devices and its supported protocol interfaces. Hence, those PCBs can be validated without any automation. However, the approach may not bring any success when a large number of PCBs are to be tested by executing a series of test cases manually. Such an approach can also be error-prone. Especially, when there is a demand for high-quality design with a given constraint of time, it is important to plan the validation of the design strategically. It leads towards either finding an alternative approach or improvement in the conventional validation cycle.

Design of a software platform to automate the test cases is a possible robust solution to target high reliability in short time. Test cases related to the confirmation of communication between devices using specific protocol interfaces can be automated to save time and ensuring high reliability. Such an approach of automating the test cases is advantageous as it offers a feature of reusability. While saving time and resources by means of automation promises to be a good approach, the idea of reusability can also be extended for the validation of the next generation design. Thus, test automation can be utilized to validate the protocol specific functionality of any of its compatible devices.

Automation scripts can be developed to test many of the features offered by the design. Once these scripts are developed in a given time, they can be executed on any number of PCBs employing the design to be validated. These scripts can also be designed more intelligently using scripting languages to direct us towards possible debug points in case of a failure.

1.2 Hardware Software Validation

To validate the overall functionality offered by newly designed Integrated Circuits (IC), PCBs are designed in such a way that they can achieve the goal of validation. Set of specifications of IC design to be validated determines the architecture of these evaluation boards. Design of such

evaluation boards follows hardware development life cycle, which begins from the review of the specifications and followed by a concept to the design phase. Successful design sign-off of prototype design propels the assembly stage. Once the components are assembled, PCBs are fabricated and later manufactured in required quantities. However, issues caused due to the malfunction of any of the device on the board or fabrication process may not guarantee the full range of operation for all the components. Thus, manufactured PCBs entail the necessity of functional testing in the validation phase.

It is also noteworthy to mention that the set of PCBs fabricated during the assembly stage may not replicate the same level of performance due to the discrete nature of the components. PCBs comprise of devices that require configuration and programming beforehand such as FPGA and Microcontroller. Hence, validation scripts to configure devices to operate on different protocols plays a crucial role before validation begins. Such development of application-level scripts to propel the communication between various ICs on the designed validation boards (PCB) target towards software design phase for validation. The software is designed by many application-level scripts for debug purpose.

Among the recent trends in the field of design validation, 'Hardware Software Validation' is an approach resulting shorter design cycle time and time-to-market. Both the design cycle described above for hardware and software can be carried out separately in a parallel manner. At a stage when both components of the system achieve stability, software is mapped into the hardware. Validation of the design of IC is performed using an approach known as Hardware Software validation. The approach reflects a fact that it uses application level software to interact with the developed hardware to validate the design of an IC thoroughly.

In this approach, developing a test plan is a standard practice to ensure the proper validation of the functionality of the design. Test plan comprises of coverage matrix and test scenarios which essentially serves the purpose of thorough validation of PCBs. Coverage matrix depicts the scope of the validation for given design. It describes the primary and secondary features need to be tested at various priority levels. Test cases portray a combination of different inputs to the system

and their expected outputs. It is a reference point to compare the actual output with the expected output. It aids to identify a critical bug in the functionality.

1.3 Objectives

The primary objective of the project is to develop the test automation that confirms an I2C transaction between a control device (Spartan®-6 FPGA) and any I2C compatible device (M24C08 EEPROM). Python scripts are developed to enable Inter-Integrated Circuit (I2C) communication between these two I2C compatible devices using hardware software validation approach. Spartan®-6 evaluation board SP605 is used to execute the test cases developed in Python. To be specific, the project also targets at the design and implementation of I2C protocol using Python for data transfer purpose. In proposed work, a base script capturing the overall functionality of I2C protocol is developed. In addition to the base script, device specific script for EEPROM is also developed.

Various functions have been developed to implement read and write operations of I2C in the base script. Read and write operations in the base script utilizes these functions to achieve the overall functionality of I2C. These two functions are so generic that they can extend the support to any of the I2C compatible device's read and write operation.

In the device specific script, read and write functions are also developed. However, these read and write functions call the same functions of I2C. On top of it, test script attempts to validate the I2C communication between the control device and I2C compatible device using base and device specific scripts, respectively. Test scripts confirm the I2C transaction between these devices in a random as well as directed manner.

1.4 I2C Validation

As the aspect of speed has acquired the market of the semiconductor in last few years, the field of discrete circuit design has demonstrated the compatibility by introducing high-speed interfaces for communication among ICs on the circuit boards. Examples are Serial Peripheral

Interface (SPI), Universal Asynchronous Receiver Transmitter (UART), Universal Synchronous Asynchronous Receiver Transmitter (USART), Inter-Integrated Circuits (I2C) and Peripheral Component Interface (PCI).

Inter-Integrated Circuits(I2C) is a serial communication protocol that enables communication among two or more devices. Basically, I2C is a two-wire interface standard developed for high-speed communication purpose between devices. It enables a control device and its compatible devices to communicate over the two wires using some pre-defined conditions. It deploys two devices namely master and slave. Any of the control devices such as Microcontroller and FPGA can act as master. I2C compatible devices such as Electrically Erasable Programmable ROM (EEPROM), Analog to Digital Converter (ADC) are the examples of slave devices.

I2C bus protocol provides a low-cost, but powerful, chip-to-chip communication link, so that it has expanded its communications role to include a wide range of applications such as memories, input and output devices, sensors of many types, real-time clocks, displays, data entry devices, and much more [3].

Based on the application for which the PCB is designed, it may involve one or more control device that can enable the communication among peripherals on the board. Control devices such as FPGA, SoC, Microprocessor or Microcontroller incorporate interfaces supporting I2C, SPI, UART, USART.

Inter-Integrated Circuit(I2C) is one of the popular communication protocol used for data transfer. Modern control devices have an I2C-bus interface inside it to communicate with devices compatible with the protocol. To form the other end of the communication channel, devices in the form of ICs such as EEPROM, ADC, sensors etc. also have an in-built I2C-bus interface. Channel is formed between a control device such as FPGA and I2C compatible devices described above using two bus lines I2C offers. Enabling the communication between such devices using I2C protocol consumes a good amount of time during the validation. Application level access to confirm the functionality of these devices is one of the crucial aspects of validation of I2C-bus interface for the design to be validated. Hence, software is developed using Python to automate the validation of I2C controller of Spartan®-6FPGA design.

Automation work includes a generic script to implement an I2C protocol for the required control device, device specific script to support the I2C interface of a specific device and a test script that validates various scenarios to confirm the communication between devices. PCB design comprises of the control device and I2C compatible devices used in the project can run the test script to validate the functionality of the I2C interface of Spartan®-6 FPGA. To verify the functionality of I2C in the control device in the system, automated scripts are developed to confirm the functional correctness of I2C transactions among control device and I2C compatible devices.

1.5 Report Outline

Throughout the report, term device refers to the semiconductor devices available in the form of Integrated Circuits (IC). The Structure of the report is as follows:

- Chapter 2 provides information about the Inter-Integrated Circuit (I2C) protocol. It highlights the format and specification related to the communication between devices. It also explains the design challenges associated with I2C-bus based systems. At last, it includes literature on an I2C transaction involving read and write operations.

- Chapter 3 summarizes the proposed system architecture. It describes the development of the various scripts developed to achieve the primary objective of test automation.

- Chapter 4 describes details regarding evaluation board SP605 as a hardware platform used in the project to validate I2C master controller of Spartan®-6 FPGA. It also goes through the definitions of the device specific read and write operations for EEPROM.

- Chapter 5 discusses the execution of the various test cases and the corresponding results indicating success and failure of the test cases. It gives an approximate estimate of the time and effort saved by the proposed test automation.

- Chapter 6 concludes the report and discusses some future work.

Chapter 2: Inter-Integrated Circuit (I2C)

This chapter presents a brief introduction to the Inter-Integrated Circuit (I2C) protocol used for data transfer among electronic devices. It describes the electrical characteristics, operation and supported modes of operation. It also provides detailed information about an I2C transaction involving read and write operations at the end of the chapter. Both read and write operations for I2C-bus are explained through an analogy of listening and talking. Term 'electronic devices' refer to the Integrated Circuits (IC) found on printed circuit boards. Meaning of the term remains constant throughout the report.

2.1 Introduction to the Inter-Integrated Circuit (I2C) protocol

Inter-Integrated Circuit (I2C) is a protocol that enables communication between two or more devices on circuit boards for efficient inter-IC control. It is basically a full-duplex two-wire bus developed by Philips Semiconductors. Present day PCBs employ numerous components embedded in the system. Control devices such as Microcontroller, FPGA along with devices as such as ADC, EEPROM, sensors are the integral parts of the circuit boards.

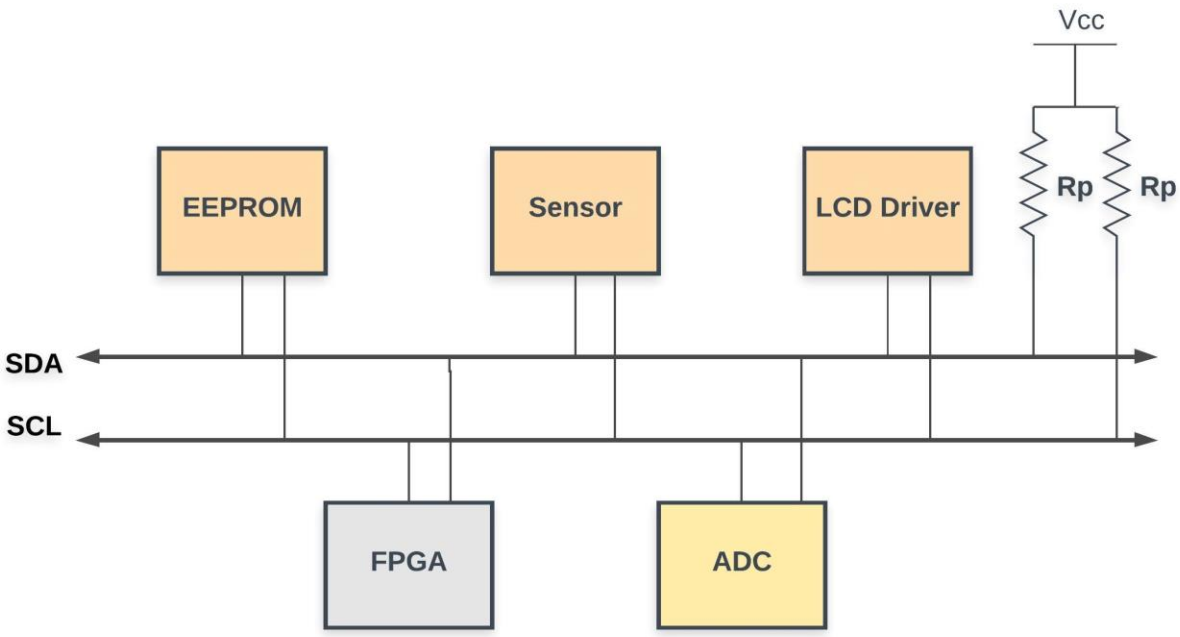

Figure 1 Example of I2C bus [4]

As shown in Figure 1, devices compatible with I2C protocol incorporate two-wire bus that enables them to communicate with each other [5]. The parameter of speed in circuits has also risen in the last few years. Devices found on the circuit boards operate on various clock speeds. Thus, it becomes essential that devices operating at slow speed must not slow down high-speed devices during the period of communication. I2C fulfills this objective of low cost and high reliability by introducing a specification [5]. It specifies the format and procedure affiliated with the communication among devices. It oversees the data transfer between I2C-bus compatible devices.

I2C-bus supports two types of addressing for data transfer mentioned below [5].

1. 7-bit addressing
2. 10-bit addressing

The literature described in this chapter as well as throughout the entire report is for 7-bit addressing only. Hence, the address of any I2C-bus compatible device is assumed to be 7-bits in the report. As 10-bit addressing is not widely used and test cases developed in this project are also designed for 7-bit addressing, its description in this report is not included.

2.2 Supported Features

Key feature differentiating I2C from any other serial bus is a fact that it requires only two buses known as a serial data line (SDA) and a serial clock line (SCL) for the operation. Therefore, I2C is also known as a two-wire interface. This two bus-lines aid to achieve serial full-duplex data transfer at a rate varying from 100kbps to 400kbps depending upon the application [5]. Table-1 lists the speed of operation supported by I2C-bus in different modes [5].

Mode of Operation	Rate of Data Transfer
Standard	100 kbps
Fast	400 kbps
Fast plus	3.4 Mbps
Ultra-fast	5 Mbps

Table 1 Speed of Operation supported by I2C [5]

As stated in Table-1, I2C also supports up to 3.4Mbps and 5Mbps in high-speed modes called fast-mode plus and ultra-fast modes [5]. Each I2C compatible device using the I2C bus is easily identified by a unique address. Apart from that, I2C also employs collision detection to prevent data corruption if two or more control devices attempt to initiate data transfer simultaneously. I2C-bus reduces the necessity of interfacing among different ICs on the circuit boards. All the I2C-bus compatible devices have an I2C interface integrated inside the chip itself. Thus, I2C reduces the requirement of interfacing in circuit boards.

2.3 I2C Terminology

Before moving forward to describe the operation of I2C-bus, it is essential to be familiar with the terminology used for I2C. Table-2 contains the definitions of various terms used frequently in the data transfer between I2C compatible devices.

Term	Definition
Master	The device which • Initiates a transfer,

	• Generates the clock signals, • Terminates a transfer
Slave	The device which is addressed by a master
Multi-master	A situation when more than one master attempts to control the bus at the same time without corrupting the message
Transmitter	The device which sends the data to the bus
Receiver	The device which receives the data from the bus
High	Electrical output equivalent to Logic '1'
Low	Electrical output equivalent to Logic '0'

Table 2 Definition of terms used in I2C communication [5]

2.4 Analogy of Talking and Listening

Prior to exploring the complex format of data transfer of I2C involving read and write operations, understanding the process of talking and listening between two persons, A and B can give enough idea of the I2C operation. Talking and Listening form the basis of the communication process. Also, understanding the analogy of talking and listening can be proven extremely useful to gain a thorough understanding of the write and read operation of I2C. The flow of information in both cases is explained in figure-2 and figure-3.

Case-1) A wants to talk to B.

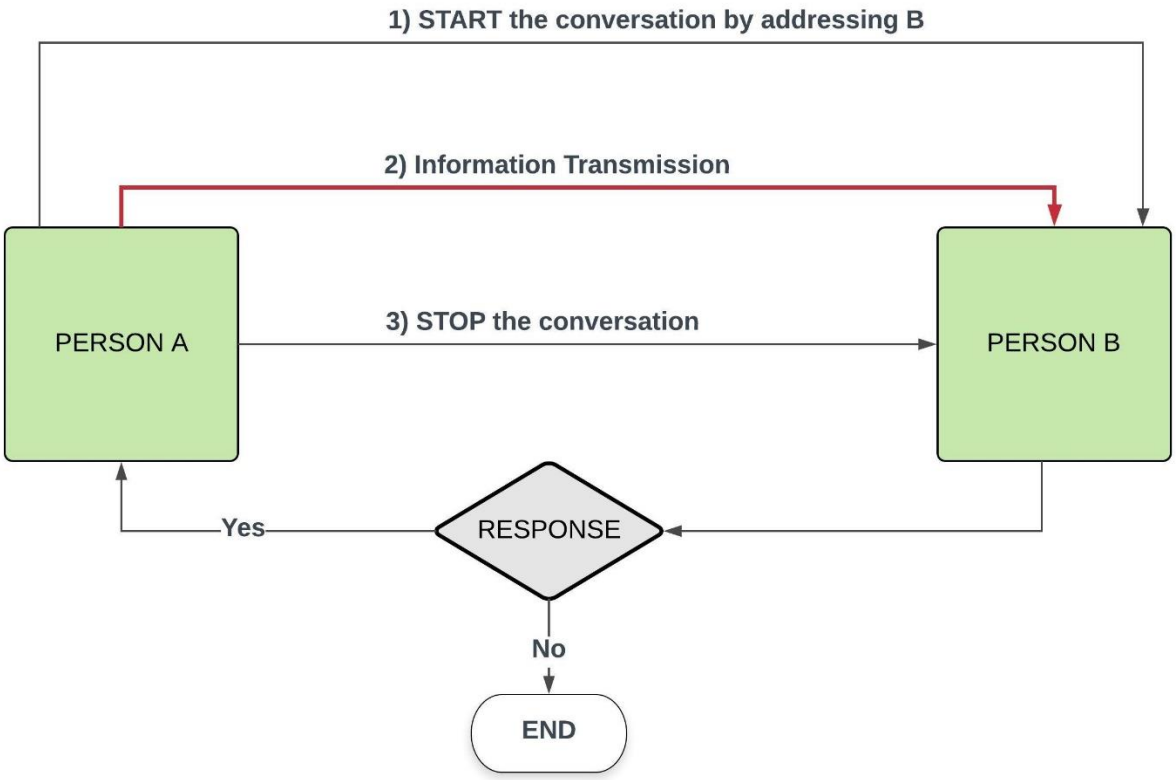

Figure 2 Process of A talking to B

The communication process between A and B will follow the sequence described below.

- A will initiate the conversation by addressing B.
- A will wait for a response from B.
- If B responds positively, A can talk to B.
- Person A will stop the conversation.
- At any given time, no response from B will mark the end of the conversation as it indicates B not showing any interest in talking to A.

A can continue talking to B as long as B responds positively. B may wish to acknowledge the end of the conversation. This points to the end of talking. In the event of talking, A transmits the information to B after beginning the communication. While A is the primary object, B is only responsible for sending a response or no response.

Case-2) Person A wants to listen to Person B.

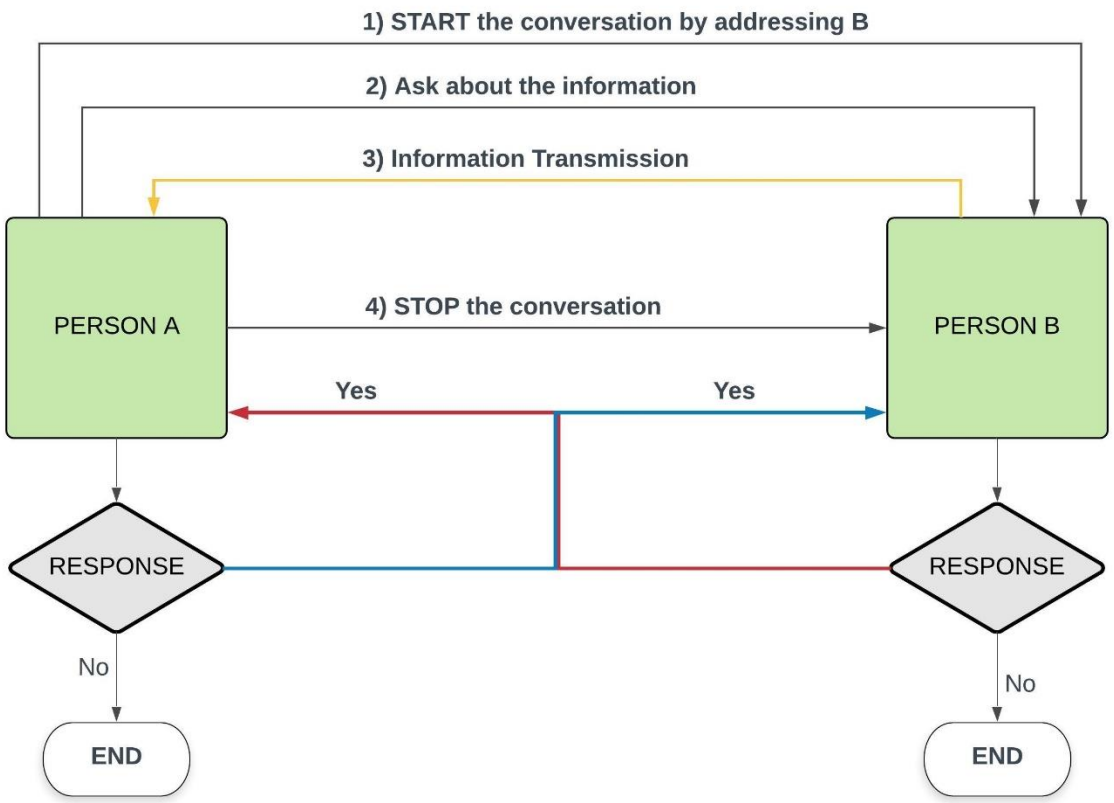

Figure 3 Process of A listening to B

The communication process between person A and person B will follow the sequence described below.

- A will initiate the conversation by addressing B.
- A will wait for a response from B.
- B sends the information to A.
- A listen to the information sent by B.
- A will stop the conversation.
- At any given time, no response from B will mark the end of the conversation.

In this case, person A wants to listen to person B. Thus, information will be transmitted in a reverse manner. The flow of information will be from B to A. Even in this event, person A will initiate the conversation as A wants to get the information from B.

2.5 Operation

SDA and SCL carry the information between the devices connected to the bus. Each device connected to these two buses can be identified by a unique address. Any of the device connected to the I2C-bus can act as either a transmitter or receiver. For example, EEPROM can transmit and receive the data, while the LCD driver may only receive the data. Any of the device connected to the I2C-bus can be considered as either master or slave during data transfer. An I2C transaction comprises of read and write operations. Depending upon the device which initiates the communication and generates the clock signals to allow the transfer, a device can be considered as master. A device which is addressed by the master for data transfer acts as a slave.

Consider two scenarios where two devices FPGA and EEPROM shown in figure-1 want to communicate. Communication between these two devices refers to the process of data transfer. It involves sending the information to the device or receiving the information from the device. Assuming FPGA and EEPROM act as master and slave devices respectively.

2.5.1 Writing Data

Sending the information to the device is known as writing the data. Thus, write operation on the slave of the I2C-bus can be analyzed using an analogy of A talking to B. Replacing A by FPGA and B by EEPROM gives an insight to the write operation on the I2C-bus which is similar to the case of talking described in the previous section. Thus, data transfer in case of a write operation will take place in the sequence described below.

Case:1) FPGA wants to send information to EEPROM: (Similar to A wants to talk to B)

– FPGA (master-transmitter) addresses EEPROM (slave-receiver)

– FPGA (master-transmitter) sends data to EEPROM (slave-receiver)

– FPGA terminates the transfer.

As FPGA is sending the information to the EEPROM, it can be considered as a transmitter. Similarly, EEPROM as the receiver is also true for receiving the information. Therefore, FPGA acts as master-transmitter and EEPROM as slave-receiver.

2.5.2 Reading Data

Receiving the information from the device is known as reading the data. Hence, reading from the slave on the I2C-bus is same as A listening to B explained in the previous section. Replacing A by FPGA and B by EEPROM, FPGA receiving the information from EEPROM defines the read operation on I2C-bus. Thus, data transfer in case of the read operation will take place in the sequence described below.

Case:2) FPGA wants to receive information from EEPROM: (Similar to A wants to listen to B)

– FPGA (master) addresses EEPROM (slave)

– FPGA (master-receiver) receives data from EEPROM (slave-transmitter)

– FPGA terminates the transfer.

In read operation, FPGA is receiving the information from EEPROM. Thus, EEPROM which is a slave device acts as a transmitter. FPGA. the master device is considered as the receiver. Therefore, the information flow is opposite to that of the write operation. Irrespective of the direction for the data transfer, the master device is always accountable for initiating as well as terminating the transfer and generating the clock signals.

Examples described above show an application of a single master and a slave device using the I2C-bus for the data transfer. However, the I2C-bus also supports the data transfer for multiple master devices and a single slave device. This mode of operation defines the I2C-bus as a multi-master bus.

2.5.3 SDA and SCL signals

SDA and SCL are bidirectional lines connected to a positive supply voltage via pull-up resistors. During the data transfer, these buses can be either busy or idle. When idle, both buses will have a logic '1' signal. The levels of the logic '0' and logic '1' are functions of the value of the supply voltage and the process technology. Input reference levels are set as 30 % and 70 % of V_{CC}. V_{IL} is 0.3 V_{CC} and V_{IH} is 0.7 V_{CC}. Figure-4 shows a transfer of a bit on the I2C-bus.

"The data on the SDA line must be stable during the logic '1' (High) period of the clock. The high or low state of the data line can only change when the clock signal on the SCL line is low. One clock pulse is generated for each data bit transferred [5] ."

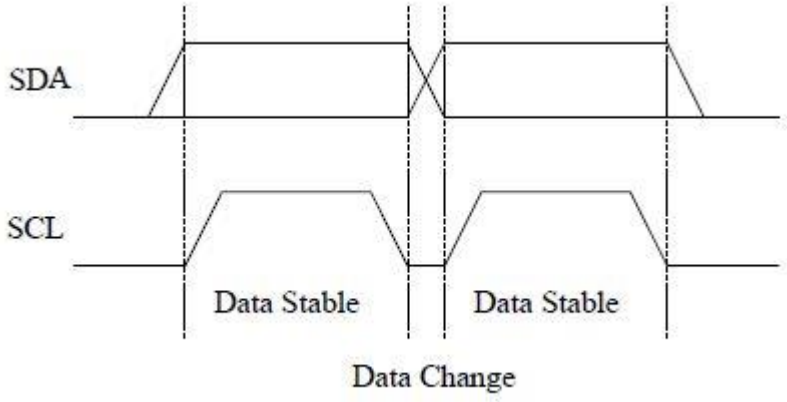

Figure 4 Bit transfer on the I2C-bus [6]

2.5.4 Start and Stop conditions

To begin the data transfer, start condition must be sent to the slave device. Stop condition from the master device is also mandatory to terminate the transfer. Figure 5 shows an example of start and stop conditions.

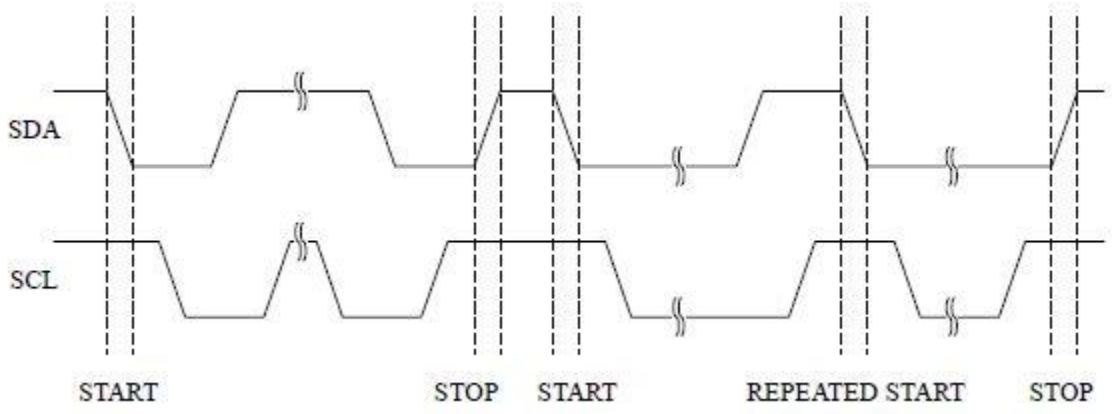

Figure 5 START and STOP conditions [6]

If both the data and clock lines are at logic '1' after a stop condition, the bus is said to be in the idle state. "A high-to-low transition on the SDA line while the SCL is high defines a start condition. A low-to-high transition on the SDA line, while the SCL is high, defines a stop condition [4]." As shown in Figure 5, the master device can also send the repeated start condition instead of stop condition to indicate that it wants to initiate another byte of data transfer.

2.5.5 Byte Format

Every byte put on the SDA line must be eight bits long. I2C-bus do not impose any restriction on the transmission of the number of bytes. However, transmission of 8-bits must be followed by an Acknowledge bit from the receiver.

While master sends a byte to the slave, the slave device may decide not to respond provided that it is busy. The slave device may be busy in the real-time event such as servicing an interrupt. In such scenarios of a slave being busy, it can hold the SCL line to low. As a result, the master will enter a wait state [5]. Slave releasing the clock line indicates that the slave is ready for another byte of data. Thus, data transfer can continue [5]. Data is transferred with the Most Significant Bit (MSB) first as shown in Figure 6.

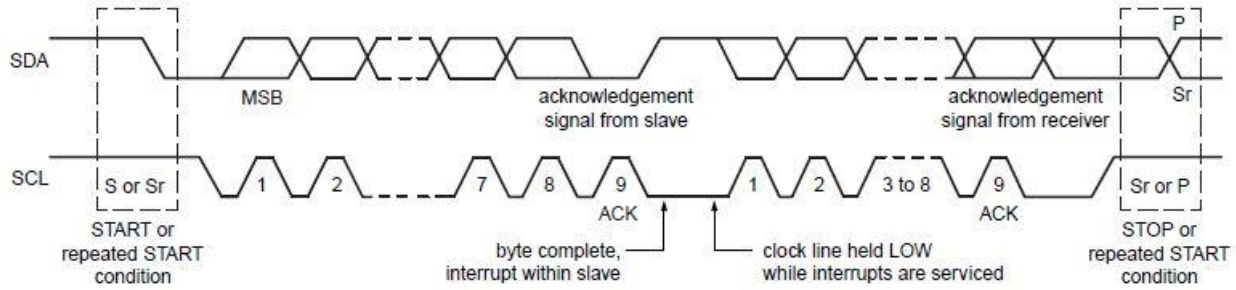

Figure 6 Data transfer on the I2C-bus [5]

2.5.6 Acknowledgment and Not Acknowledgment

In the data transfer between master and slave devices, successful transmission of a byte is followed by an acknowledgment bit from the receiver. Acknowledgment bit can be abbreviated as ACK. This acknowledgment bit signals the transmitter that the receiver has successfully received a byte. Upon receiving the ACK, another byte of data can also be transmitted. However, the transmitter must release the data line before the receiver can send an ACK. If the transmitter fails to do so, the receiver can send a not acknowledgment bit known as NACK. The master generates the acknowledge ninth clock pulse.

“The Acknowledge signal is defined as follows: the transmitter releases the data line during the acknowledge clock pulse, so the receiver can pull the data line low and it remains stable low during the high period of this clock pulse [5].”

Data line during the ninth clock pulse, which is an ACK bit must stay low. If it remains high, it is interpreted as NACK. As a result, the master can decide to abort the transfer by generating a stop condition. [5] Master can also generate a repeated start condition to new transfer. Figure 7 shows an example of a NACK waveform.

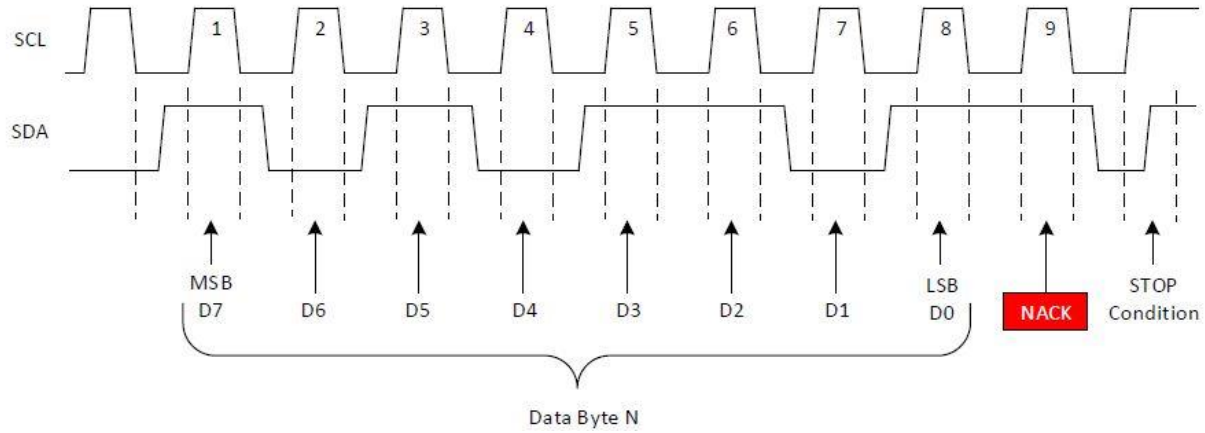

Figure 7 Example of NACK waveform [4]

Few scenarios that may lead to the generation of NACK by the slave device are listed as below.

1. The receiver is busy such as servicing the interrupt
2. The receiver is not able to interpret the data or address
3. The receiver has already crossed its maximum limit of receiving the data
4. The absence of the receiver on the bus

2.6 Data Transfer

Transfer of a single byte of data in the I2C-bus is shown in Figure 8.

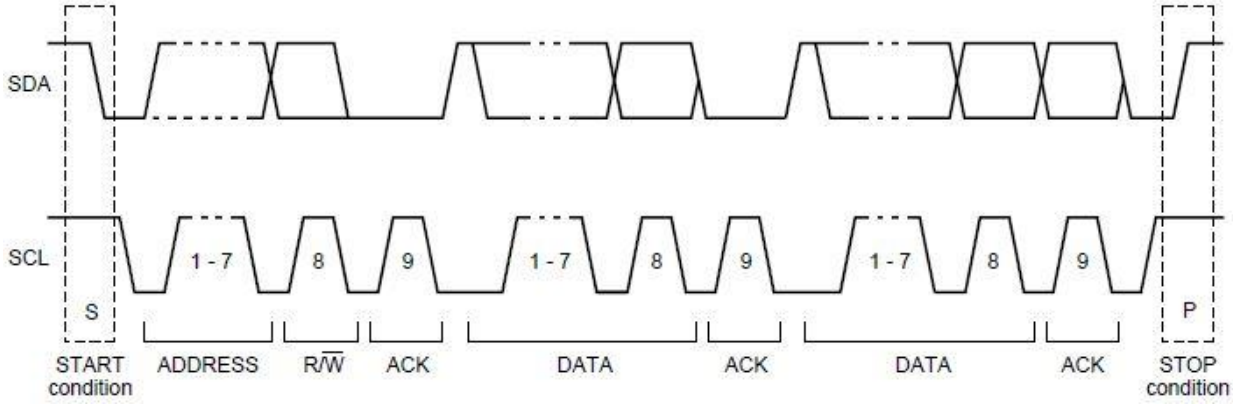

Figure 8 Single byte of data transfer [5]

Transfer of a single byte of data using the I2C-bus involves address packet and data packet. Thus, the data transfer on the I2C-bus can be explained by simplifying the data and address packets

separately. Address packet is responsible for transmitting the address of the slave device master wants to communicate with. On another end, the data packet is intended for either sending the data to a slave or receiving the data from the master. Below sub-sections describe the address and data packet formats.

2.6.1 Address Packet Format

Once the master sends the start condition, it transmits the address of the slave device for data transfer. The format of the slave address is shown in Figure 9.

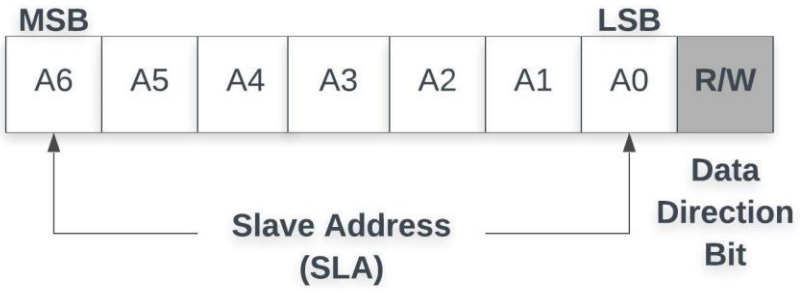

Figure 9 Format of address byte for sending the slave address [5]

Most Significant Bit (MSB) of the slave address is transmitted first. Slave address is 8-bit long address with 7-bits indicating the slave device address and a data direction bit (R/W) to indicate read or write operation. If R/W is set to 0, it means the master wants to write to the slave device. In case of R/W is equal to 1, master desire to read the data from the slave device. If it is not specified, write operation is performed by default.

When the slave device recognizes that it is being addressed, it will send an acknowledgment (ACK) of successful receiving the address packet by pulling the data line to low. Thus, ACK bit is a clock pulse followed by the slave address. In an event when the slave device is busy, the slave can't acknowledge the master's request for communication. Thus, data line will remain high during that clock period and this is known as non-acknowledgment (NACK). After receiving ACK bit, the master continues to transfer the data packet. At the end of the transmission of a byte of data, the master sends a stop condition to terminate the data transfer.

Master may choose to transmit a repeated start condition(Sr) condition instead of stop condition. Master can send the address of the slave other than address '0000 000' which is reserved for a general call.

2.6.2 Data Packet Format

Each byte of data transmitted on the I2C-bus is also followed by an acknowledge (ACK) bit in a similar fashion of an address byte. In the data transfer, the master generates the clock and transmit the start and stop conditions. The receiver must pull the data line low for sending an ACK. If the receiver keeps the data line high, it is signaled as NACK. The receiver can send NACK if it has received the last byte of data or it can't receive any more byte of data.

2.6.3 Combination of Address and Data Packets

Combining the address and data packets directs towards an I2C transaction meaning read and write operation between master and slave device. Depending on the direction of data transfer, either master or slave can be transmitter or receiver during the communication. Both operations are described separately in the next sections with the roles of master and slave devices either as a transmitter or a receiver. Figure 10 presents the data transmission combining address and data packets.

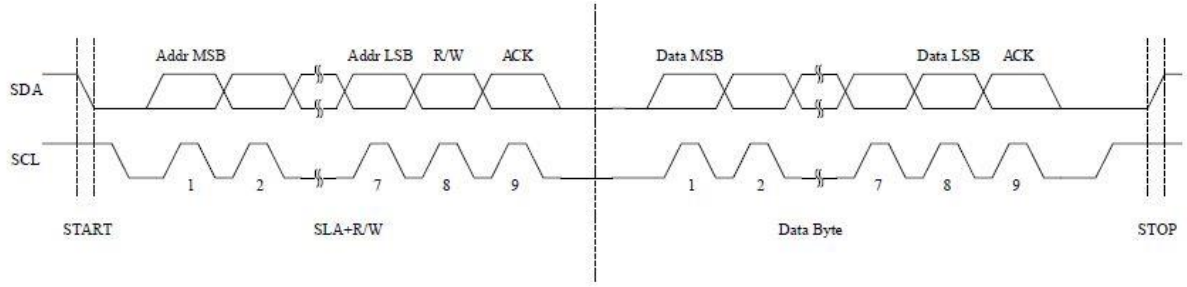

Figure 10 Data Transmission [4]

2.7 Format for Write Operation

To write the data using the I2C-bus, master and slave device communicate each other in the manner described below. Master writing one byte of data to the slave is illustrated in Figure 11. Master can write one byte of data and more than one byte of data to the slave. Throughout the operation of a master writing the data to the slave, the master will be a transmitter and slave will act as a receiver.

1. Master-transmitter sends a start condition(S) on the bus with the slave's address with the R/W bit set to 0 signifying a write operation.
2. Slave-receiver sends an acknowledgment bit (ACK).
3. Master-transmitter then sends the register address of the slave it wishes to write to.
4. Slave-receiver will acknowledge the master again indicating it is ready to receive the data.
5. Master-transmitter transmits the data byte to the slave.
6. Master-transmitter terminates the transmission by sending a stop (P) condition.

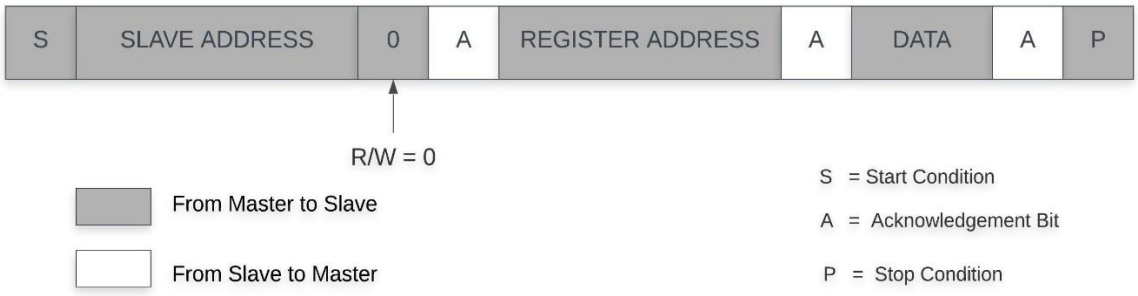

Figure 11 Writing one byte of data to the slave [5]

Master can also write more than one byte of data to the slave. In case of writing multiple bytes of data, the master will send a repeated start condition (Sr) instead of stop condition. Rest of the sequence is executed from the sending the slave address to write another byte of data. At last, the master sends a stop condition when it wants to terminate the data transfer.

2.8 Format for Read Operation

Reading from a slave is very similar to writing. Figure 12 shows an example of master reading a single byte from a slave register. To read from a slave, the master must first instruct the slave which register it wishes to read from.

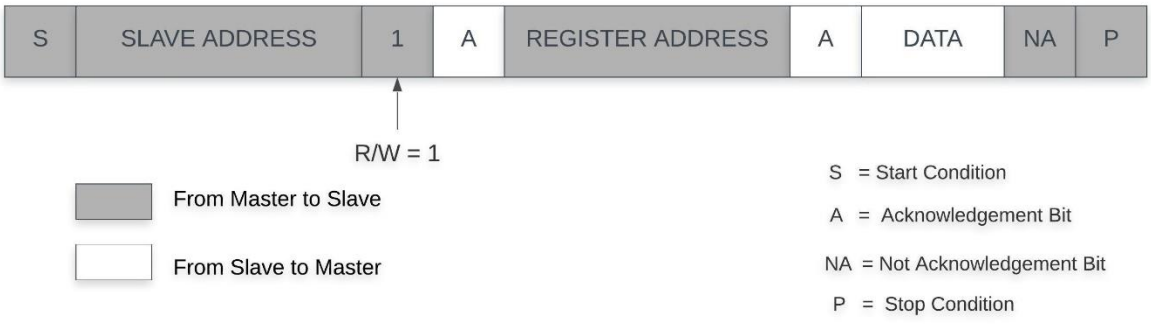

Figure 12 Reading one byte of data from the slave [5]

Now, Master-transmitter sends a start condition(S) on the bus with the slave's address with the R/W bit set to 1 signifying a read operation. Reading a byte of data from the slave follows below described sequence for data transfer.

1. Master-transmitter sends a start condition(S) on the bus with the slave's address with the R/W bit set to 1 signifying a read operation.
2. Slave-receiver sends an acknowledgment bit (ACK).
3. Master-transmitter sends a register address of the slave it wishes to read from.
4. Slave-receiver will acknowledge the master-transmitter again.
5. At this point of time, master-transmitter and slave-receiver will reverse their roles.
6. Slave-transmitter transmits the data byte to the master-receiver.
7. Master-receiver sends non-acknowledgment bit(NACK) after it received the number of bytes it is expecting
8. Master-receiver terminates the transmission by sending a stop (P) condition.

After receiving the second acknowledgment, the master will release the data bus and continue generating the clock signals. In this operation, master-receiver sends NACK, signaling to the slave to halt communications and release the bus [4].

Chapter 3: Design of Test Automation for I2C Validation

This chapter highlights the architecture of the proposed test automation for I2C validation of Spartan®-6 I2C controller. It describes the approach used for software development to automate the validation. It also provides an overview regarding each of the development stage involved in the proposed work. It also highlights the architecture of the I2C protocol implementation.

3.1 System Architecture

Hardware Software validation methodology deals with the system comprising of hardware and software platforms. In the proposed work, Evaluation board SP605 targeting Spartan®-6 FPGA acts as the hardware platform. As illustrated in Figure 13, proposed automated validation system is composed of software platform which interacts with the hardware platform to validate the I2C functionality.

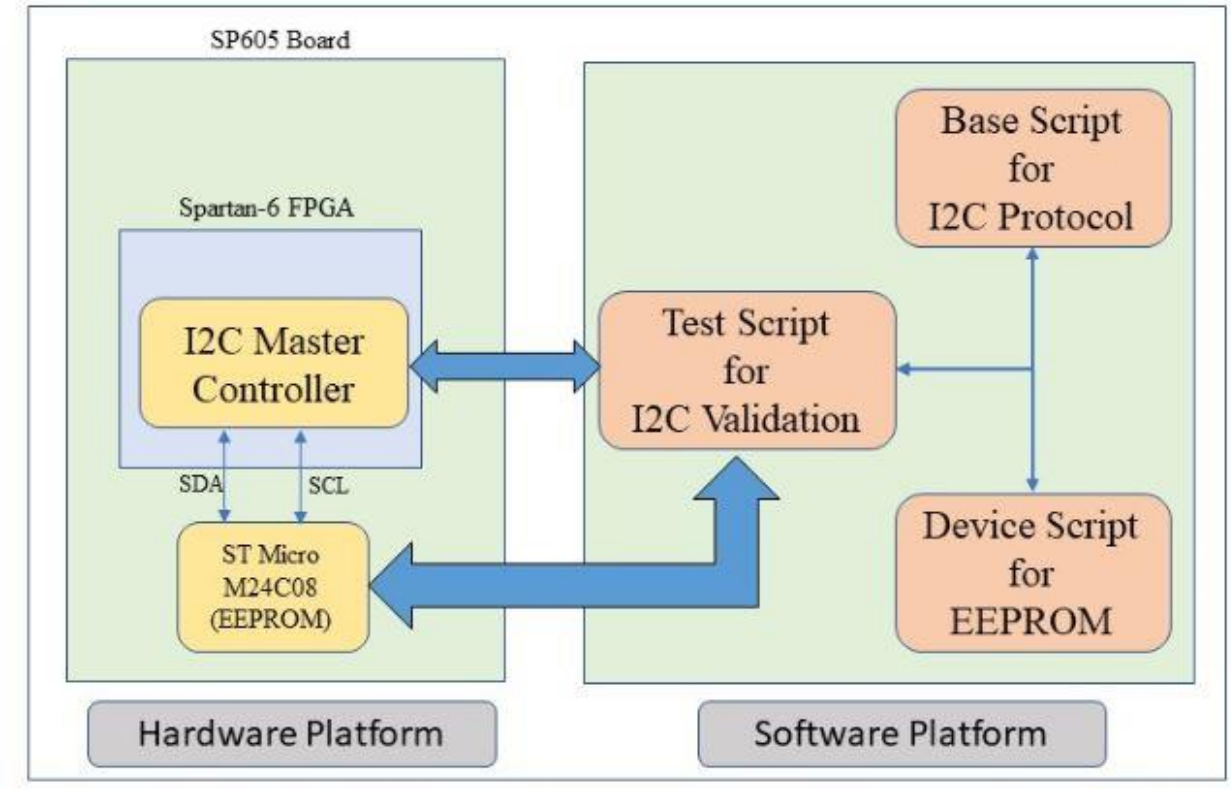

Figure 13 System Architecture of proposed test automation

Software platform to assess the correctness of I2C master controller of the targeted control device is designed and developed in Python. More information on the hardware platform is described in Chapter-4. To assess the correctness of I2C in the project, I2C master controller of the target FPGA, clock and EEPROM on the from hardware platform is considered for building the software platform.

3.2 Design of Software Platform

Proposed work can be explained by breaking it into three-stage software development. These stages are shown in Figure 14.

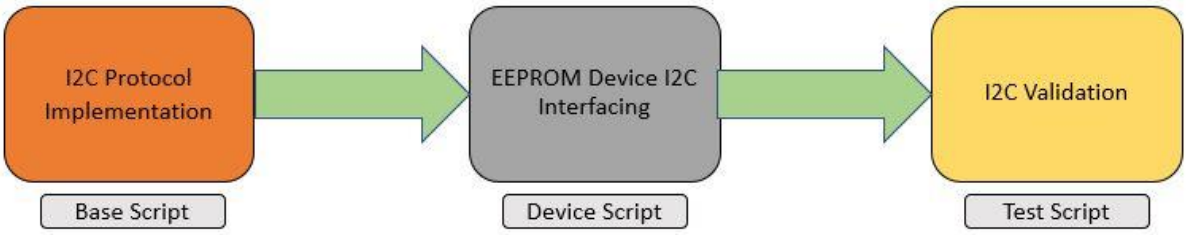

Figure 14 Three-stage Software Development for I2C validation

These three stages which aim to automate the I2C validation between target FPGA and compatible devices are listed below.

1. The base script for implementation of the I2C protocol for I2C controller of Spartan®-6
2. Device specific script for EEPROM
3. Test automation script

This hierarchical design approach drives development of software component. Design of each of the component is described in the next sections. The overall functionality of the I2C protocol involving read and write operations have been captured in the base script. These functions developed in this script are generic meaning that they are capable of extension to any of the I2C

compatible devices. Thus, read and write operations of EEPROM are designed in device script using read and write functions of the base script.

On top of these scripts, test automation script is designed that can be executed any number of times to validate the communication between I2C interfaces of the target device and EEPROM. Automation script validates the I2C communication using random tests and directed tests. Random manner refers to test the data transfer between few registers of the master and slave randomly. Thus, data transfer on any of the register from the list of the registers can be validated. Purpose of random testing is to ensure the reliability of the communication among the devices. In a directed manner, data transfer on the registers is tested in a pre-defined sequence developed inside the script. As both types of tests focus on reading and writing the data, read-only registers in EEPROM are excluded.

3.3 Development of Base Script for I2C Protocol

I2C protocol specification developed by NXP Semiconductors describes the format of communication between two devices. The design of each control device employs the I2C master controller inside it which can be interfaced with slave device such as EEPROM. The I2C master bus controller comprises various registers and memory locations which set various control devices manufactured by different vendors.

I2C functionality has been implemented for the I2C core of the target control device in the base script. The I2C controller of Spartan®-6 FPGA follows the specification and format of the I2C protocol defined by NXP Semiconductors, for implementing read and write operations [7] .

3.3.1 Architecture of the Base Script

Devices want to communicate through I2C require to follow pre-determined conditions. These conditions can be divided into four stages listed below:

1. Start and Stop condition stage
2. Address stage

3. Data stage
4. Interrupt Clear stage

According to the read and write operations described in the Chapter-2, both require start and stop bits at the beginning and end of the operation respectively. Functions to capture start and stop bits have been developed as a part of the Start-Stop stage. Address byte and data byte are required in the case of the write operation. Read requires only address byte indicating the register address of the slave from which master can read the data. Thus, address byte is also a common factor in the I2C transaction. Thus, address and data stage incorporate functions for sending and receiving the address as well data in a reliable manner. In the base script, each of these stages has been characterized by functions. A stage is made up or two or more functions.

By observing the way of operation of I2C compatible devices, read and write operations must go through these stages in a specified manner. The last stage of interrupt clear aids to clear any interrupt slave may be servicing while receiving a request for communication from the master. It ensures the master about the free state of the slave every time master sends a request to a slave. Before the start of communication, master requires to send the slave address denoting which slave device it wants to communicate with in case of Single Master- Multi Slave mode. Slave address functionality is implemented in the address module as well. I2C functionality is implemented by the design of functions for each stage.

Any of the I2C transaction involves either reading or writing the data. Base script targets to develop these primary functions, read() and write() that initiates the reading from or writing to its compatible devices. The functionality of the I2C protocol is implemented in a Python script using object-oriented programming. Bunch of small functions has been developed that can be called inside read() or write() to implement read and write functionality. The base script defines a class named, i2c_interface to encapsulate these functions along with the read and write functions. The architecture of the base script is shown in Figure 15. Read or write function inside a script calls the functions of all the 4 stages to achieve the required functionality. The functions of each stage are called in a manner specified by the I2C protocol.

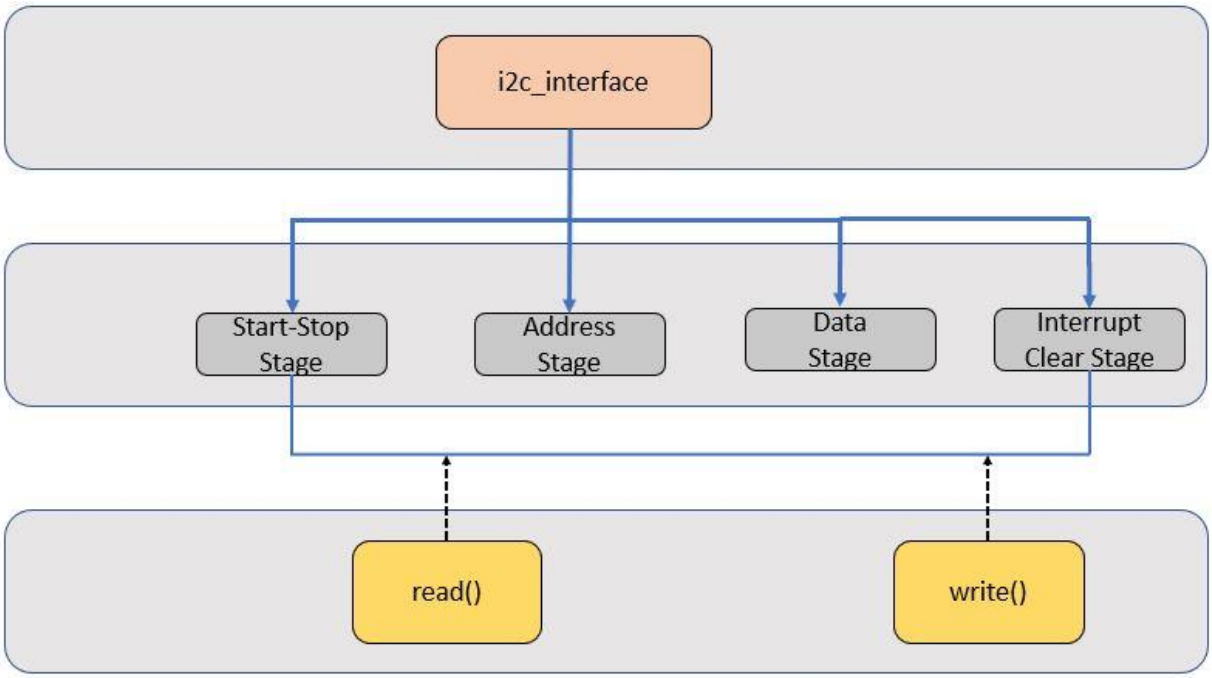

Figure 15 Architecture for I2C protocol implementation

3.3.2 Implementation of I2C Protocol

As the proposed work intends to design a test automation for I2C validation, the base script capturing the I2C protocol functionality forms the basis. The I2C transaction between two of its compatible devices involves either read or write operation. However, as described in I2C operation in chapter-2, some pre-defined conditions must be followed prior to any I2C transaction begins. Therefore, the base script essentially aims to develop two types of functions:

- Functions for pre-defined conditions such as start and stop conditions
- Functions for read and write operations

Read and write operations can be considered as the primary functions and functions setting-up the pre-defined conditions can be the secondary functions.

An alternative approach could have been to develop only secondary functions in the base script and leave the development of primary functions in the device script only. However, this approach lacks efficiency as it attempts to develop read and write functions for each of the compatible devices. Development of device-specific read and write functions ultimately utilizes

logic from the secondary functions from the base script itself. It may lead to redundancy by executing the same chunk of logic over the time. Redundancy scales up if there are more I2C compatible devices on the hardware platform. Thus, it is a good approach to develop generic primary functions in the base script itself. These functions can be called directly to implement any of the specific I2C compatible devices read and write operations.

This work spans over the development of primary functions in the base script itself. By doing that, any of the compatible devices can implement their specific read and write operations just by calling these primary functions from the base script. The interaction between the base script and device script functions has been accomplished by following the design using an object-oriented methodology. Thus, the base script has its base class and design of the device script employs a class which is inherited from the base class. Class in the base script is a parent class and class in the device script is a child class inherited from the parent class. Each of the class can be instantiated by an object. Thus, EEPROM can access any of the functions of the base script by the object.

3.3.3 Design of write()

Flow-chart of write () developed in the base script is shown in Figure 16.

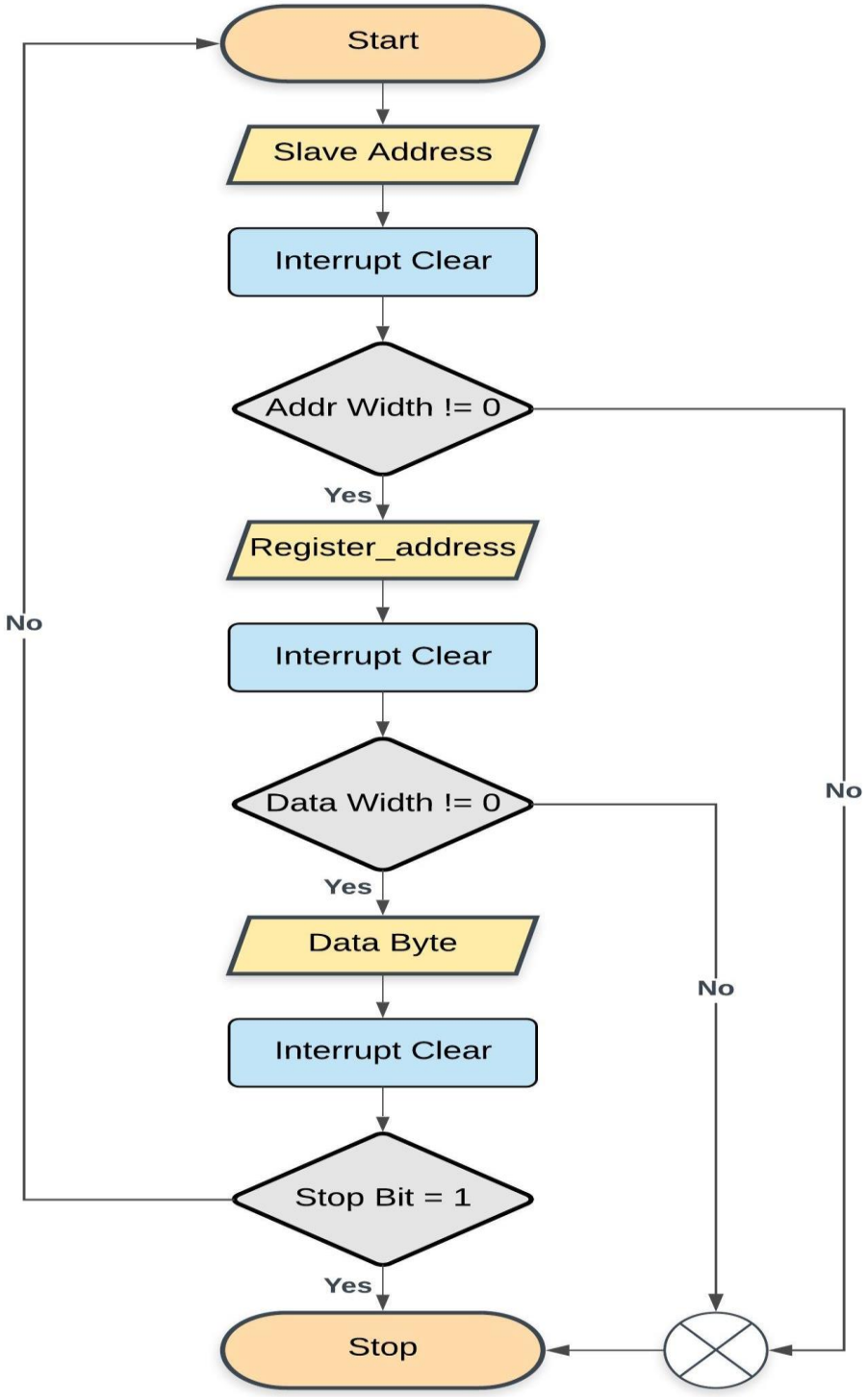

Figure 16 Flow-Chart for I2C write()

Write operation refers to master writing the e data to the slave device. Thus, master and slave devices act as transmitter and receiver respectively. The function takes the arguments of the

slave address, the address of the slave register, the width of the slave register address, data, number of data bytes as an input. It returns nothing. If the slave address passed to the function is present of the I2C bus, this function will successfully write the data on the address of the slave register. It also handles the error in case of failures by asserting an error.

- In case of the width of either slave register address or data width set to zero, the function will terminate the communication by sending the stop bit.
- Zero-width of the address indicates that the master hasn't sent the address of the slave register. Similarly, zero width of data refers to no byte of data sent by the master to the slave.

It also provides a feature of writing multiple bytes of data. If the user passes a list of the data as an input to the function, write() will write the data bytes to the slave register. Such a feature is beneficial in case of the slave register is wide enough to accept multiple bytes of data.

3.3.4 Design of read()

Read operation refers to master reading the data from the slave device. Flow-chart of read() developed in the base script is shown in Figure 17. The function takes the arguments of the slave address, the address of the slave register, the width of the slave register address as an input. It returns the data from the addressed slave register.

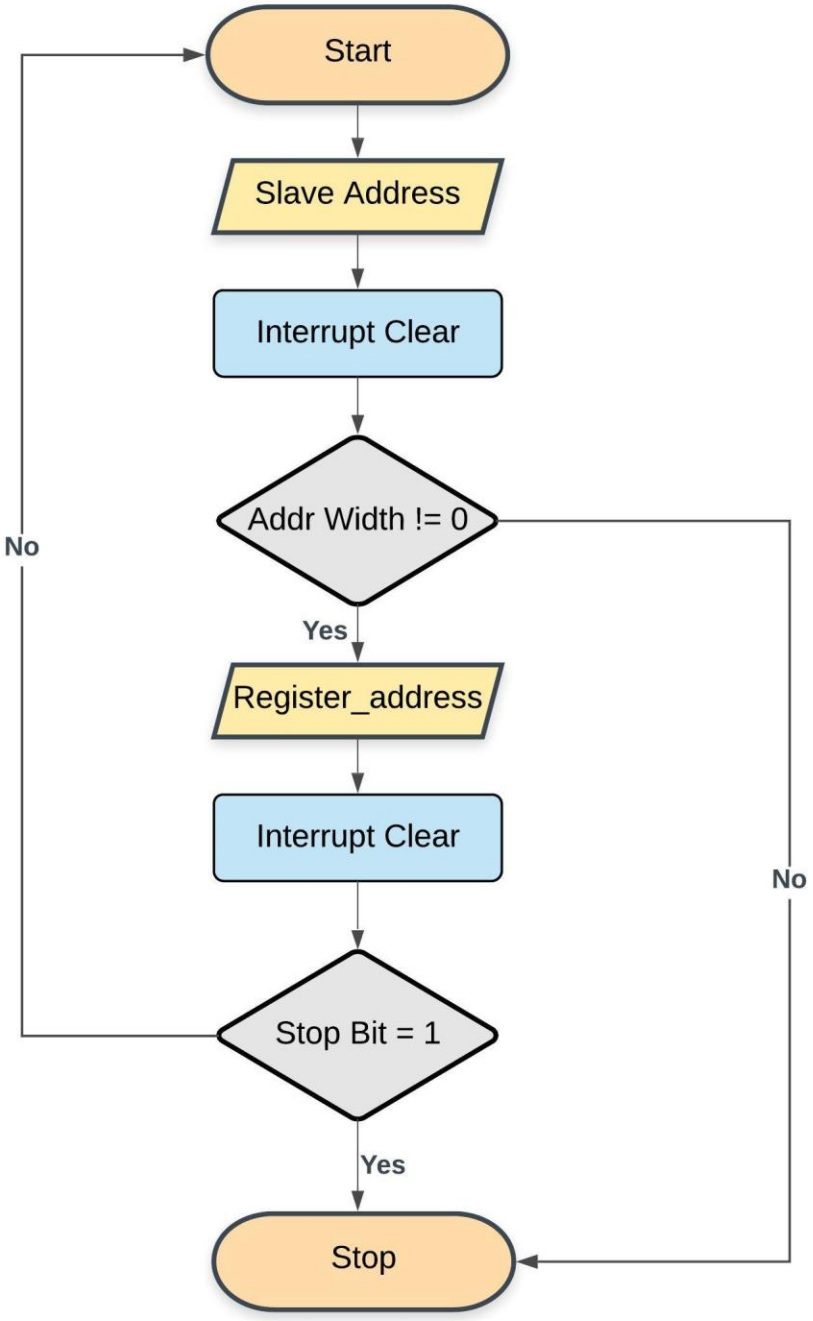

Figure 17 Flow-Chart for I2C read()

3.4 Development of Device Script for EEPROM

Design of the evaluation board SP605 includes the 8K- serial I2C bus Electrically Erasable Programmable ROM (EEPROM), M24C08 manufactured by ST Microelectronics. Datasheet of the EEPROM describes its I2C features at length [8]. While the description of the operation of the device is not in the scope of the project, few details regarding its I2C read and write operations are described here.

EEPROM supports two types of write operation listed in Table-3.

Type of Write	Description
Byte Write	Allows the master to write one byte of data to EEPROM
Page Write	Allows the master to write up to 16 bytes of data in a single write cycle to EEPROM

Table 3 Types of Write operation supported by EEPROM [8]

Development of write() function for EEPROM can be a trivial task keeping the requirement of accommodating both types of the write operation in a single function. However, the generic write() offers a solution to the problem. While write() is designed to write one byte of data to the slave device by default. Thus, byte write for EEPROM is implemented by calling the write().

As described in the previous section of the write operation in the base script, the user can pass a number of data bytes as an input to write(). Thus, the user can set that variable value up to 16 to support the page write operation of EEPROM. Therefore, page write operation has also been implemented using write(). Error is also asserted if the user attempts to write more than 16 bytes at a time.

Development of write() function for EEPROM can be a trivial task keeping the requirement of accommodating both types of the write operation in a single function. However, the generic write() offers a solution to the problem. While write() is designed to write one byte of data to the slave device by default. Thus, byte write for EEPROM is implemented by calling the write().

As described in the previous section of the write operation in the base script, the user can pass a number of data bytes as an input to write(). Thus, the user can set that variable value up to 16 to support the page write operation of EEPROM. Therefore, page write operation has also been implemented using write(). The error is also asserted if the user attempts to write more than 16 bytes at a time.

Write a function for EEPROM is basically developed from the generic structure of the write(). Thus, the flow of data transfer in EEPROM for the write operation is the same as the one illustrated in Figure 16. For read operation using I2C in EEPROM, address counter holds the current address of the addressed register. As EEPROM supports three types of read operations, the address counter plays a crucial role while the master reads the data from the slave. These three types of operations are listed in Table-4 along with the short description.

Type of Read Operation	**Description**
Current Address Read	• Allows the master to read one byte of data from the addressed register of EEPROM
Random Address Read	• Allows the master to read data from EEPROM • Dummy write is performed to load the address into the address counter
Sequential Address Read	• Can be used after either Current Address Read or a Random Address Read

Table 4 Types of Read operation supported by EEPROM [8]

Current address read is similar to the read operation described in the base script. After reading one byte of data, the address counter is incremented by one. Random address read requires dummy address write is first performed without sending a Stop condition to load the address into this address counter. During the dummy write, stop condition is not transmitted [8].

In sequential address read, the master acknowledges the data byte and sends additional clock so that the device continues to output the next byte in sequence [8]. To terminate the stream of bytes, the master generates a Stop condition.

Read function for EEPROM is also derived from the base script's read(). Thus, the flow of data transfer in EEPROM for a read operation is the same as the one illustrated in read().

3.5 Development of Test Script

The test script is the validation script that tests the communication between FPGA and EEPROM using the I2C bus. Test script is made up of the base script and device script. It basically tests various communication scenarios between master and slave device on the hardware platform using these two scripts. It employs a test function to achieve the goal of test automation. Flow-chart of the test script is shown in Figure 18. Test function employs random and directed testing on master and slave device using the I2C bus.

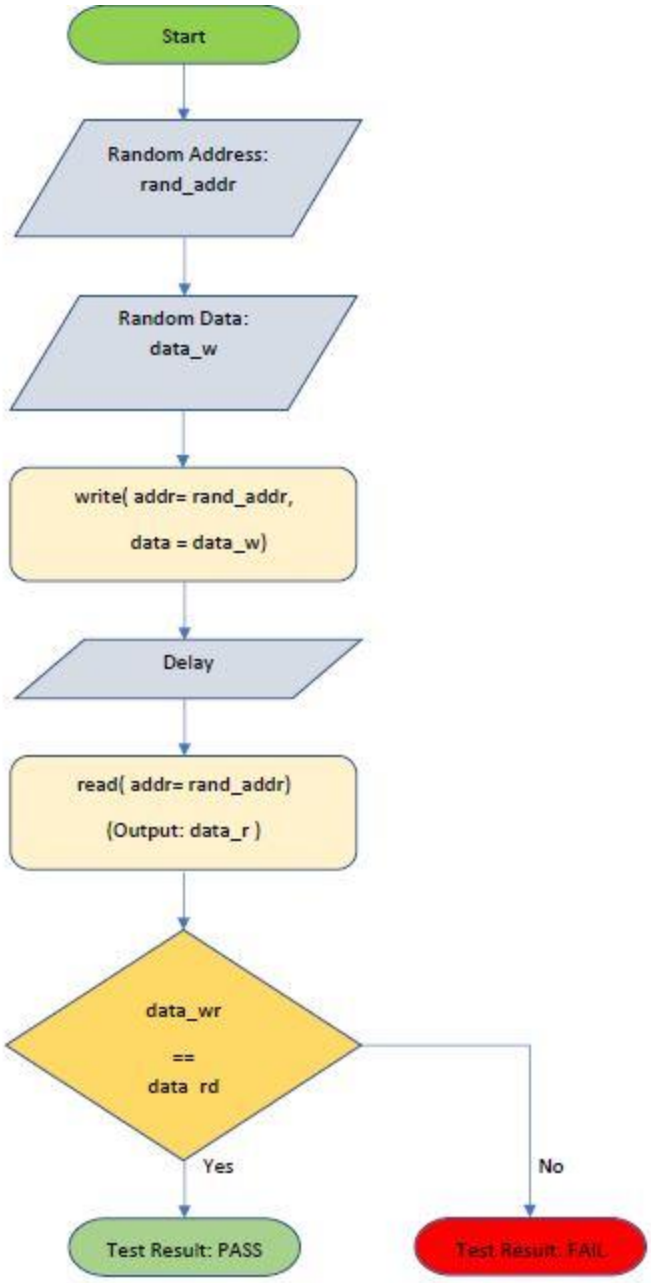

Figure 18 Flow-chart for I2C validation script

The utility of the test script can be explained by breaking it into two types of tests:

1. Random Tests
2. Directed Tests

Irrespective of the type of test, test script provides the same utility shown in the flow-chart.

3.5.1 Random Tests

The random test aims to test the I2C transaction between the FPGA and EEPROM in a random manner. It involves the utilization of random library of Python, a scripting language [9]. Random library tends to generate pseudo-random numbers from a list of integer values. It offers many functions such as randint() and choice() which are useful to get a random data and address from the list of data and address.

Before using any of its functions, it must be provided with a seed value to initialize the generation of random numbers. A seed is a number used to initialize a pseudo-random number generator. It is a starting value used by a random number generator to create random numbers. Although random library generates a pseudo-random number, generated random number is deterministic in nature. Meaning that the generated random number is a result of the seed value passed to it.

Python offers a function called seed() to set the seed value. It is generally used with the random library as random.seed() to set a random seed value. Calling random.seed() determines the random number generated from the given list. If no seed value is specified by passing no argument to random.seed(), then it uses the current time as the seed value. Thus, calling random.seed() to set the seed value is essential prior to random testing.

Each seed value will correspond to a sequence of generated values for a given random number generator. Thus, Providing the same seed value, random.seed() will generate the same number sequence every time. To change the value to be generated using random.seed(), seed value passed to it must be changed. Calling random.seed() a number of times with different seed values in a script is not a viable option. Furthermore, an I2C transaction involving read and write operations between FPGA and EEPROM is tested a few times randomly. Hence, random.seed() with no seed value is used in which current time is used as the seed value in the tests. As a result, the test goes for the value of the current time which will be different for each test case execution. Seed value will vary from test to test depending upon its time of test execution.

List of register addresses is chosen for test cases. It is important to note that EEPROM comprises many read-only registers. These registers allow reading the data only. The master device cannot write the data. As a result, the list of these registers used for the test cases does not include any of these read-only registers. Similarly, the range of data is also chosen that can be written on the EEPROM registers.

These tests comprise of write and read operations. At last, data written on the register is compared against the data received from reading the same register. The result of the test can be either Pass or Fail depending on the result of comparison between data written to register and data read from the same register.

Any of the EEPROM register address and data from the respective lists are chosen randomly. Data is first written on the randomly selected register address. This part forms the write operation. Then after, data from the same register address will be read by the master, which is a read operation. In the end, data read from the EEPROM register is compared against the data written earlier on the same register. If the data read from the EEPROM register is same as the data written on that register, the test result would be Pass. Any mismatch or discrepancy in the written data and read data would classify the test as Fail.

Between write and read operation, delay with a length of a second is inserted to allow the I2C bus to sufficient time before the next operation. As it is a clock sensitive bus requiring synchronization, it is crucial to provide some idle state between two operations. By introducing delay, the I2C bus can adjust itself to come to execute the next operation. Delay is introduced using Python's another library known as time. This library offers sleep() which can be invoked by calling time.sleep(). Passing an argument of 3 will introduce a delay of 3 seconds.

The similar test is iterated a number of times with different register address and data to determine the stability of the data transfer between FPGA's I2C master controller and EEPROM. Executing the same test through iteration also helps to examine the reliability of the functional correctness of I2C.

3.5.2 Directed Tests

While the random test is Random tests aim to validate the I2C transaction in a random manner, directed tests are designed keeping few registers of the EEPROM in focus. These are the registers which are not included in the address list for a random test. These tests are directed in a way that I2C transaction is attempted to be validated on certain registers only. Also, the I2C master controller provides a clock for synchronization. According to the clock speed, EEPROM is expected to complete I2C transaction only in I2C supported speeds such as 100kbps and 400kbps. Validation of such feature is also covered in directed tests. The flow of operation of directed tests is similar as shown in Figure 18.

Chapter 4: Hardware Platform

This chapter describes the hardware platform used for the I2C validation. As I2C master controller of Spartan®-6 FPGA forms the core of the work, the discussion in this chapter is limited to the I2C interfaces of the Spartan®-6 FPGA and EEPROM, one of the I2C compatible devices on the evaluation board. A general overview of the evaluation board SP605 is given in the first section of the chapter. Later sections focus more on the I2C interface of the Spartan®-6 FPGA and M24C08 EEPROM. These sections discuss the hardware architecture of the I2C interface between these two devices.

4.1 SP605 Evaluation Board

The hardware software co-validation approach primarily deals with hardware and software components of the system. To proceed forward in achieving the primary goal of the project, an evaluation board comprising the target device to be assessed is a must. The proposed test automation work to be executed is developed on a hardware platform involving the I2C controller of Spartan®-6 FPGA and its compatible devices. As the proposed work aims to achieve the test automation for determining the functional correctness of I2C master controller of Spartan-6 FPGA, evaluation board SP605 targeting Spartan®-6 FPGA acts as the hardware platform. Figure 19 shows the SP605 evaluation board.

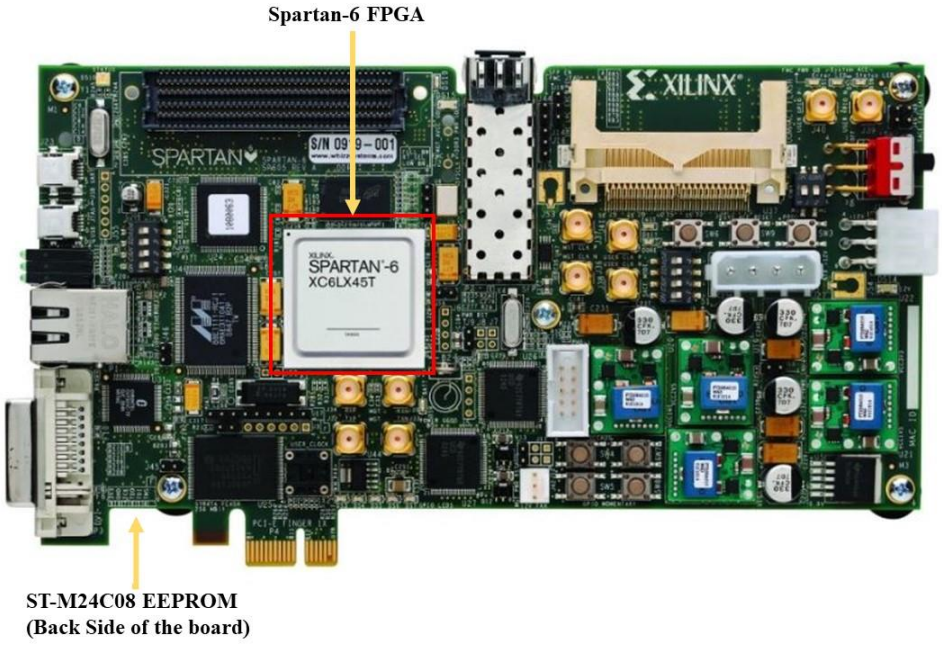

Figure 19 SP605 Evaluation Board [10]

Two components of interest, Spartan®-6 FPGA and ST-M24C08 EEPROM are highlighted in the figure. EEPROM is laid out on the back end of the board. Few of the components on the evaluation board include DDR3 component memory, USB JTAG connectors, series of LEDs and switches [10]. The SP605 board allows enables hardware and software developers to validate designs targeting the Spartan®-6 XC6SLX45T-3FGG484 FPGA [10]. The SP605 provides board features common to many embedded processing systems. As the project interest lies around I2C, the description related to the remaining components on the evaluation boards is skipped.

4.2 I2C Master Controller of Spartan-6

I2C controller designed inside Spartan-6 FPGA provides a low speed, two-wire, serial bus interface to a large number of popular devices. It supports all features, except high-speed modes specified in the I2C specifications. It also offers features as similar as described in chapter-2 of the I2C specification. List of features provided by Spartan-6 FPGA's I2C controller is listed below [7].

- Master or slave operation
- Multi-master operation

- Software selectable acknowledge bit
- Arbitration lost interrupt with automatic mode switching from master to slave
- START and STOP signal generation/detection
- Repeated START signal generation
- Acknowledge bit generation/detection
- Bus busy detection
- Fast mode 400 KHz operation or standard mode 100 KHz
- 7 bits or 10 bits addressing
- General call enable or disable
- General purpose output, 1 bit to 8 bits wide

Table-5 lists the registers which are used in the development of the proposed work in the I2C controller for achieving the I2C transaction between two or more devices.

Register Name	Read/Write
Control Register	R/W
Status Register	R
Slave Address Register	R/W
Soft Reset Register	W
Interrupt Status Register	R/W
Interrupt Enable Register	R/W

Table 5 List of registers available in the I2C master controller [10]

4.3 Interfacing

Evaluation board SP605 provides three I2C interfaces inside the Spartan-6 FPGA [10]. These interfaces are listed below.

1. MAIN I2C bus (Bank-1)

2. DVI I2C bus (Bank-0)
3. SFP I2C bus (Bank-2)

The topology of these I2C interfaces is shown in Figure 20.

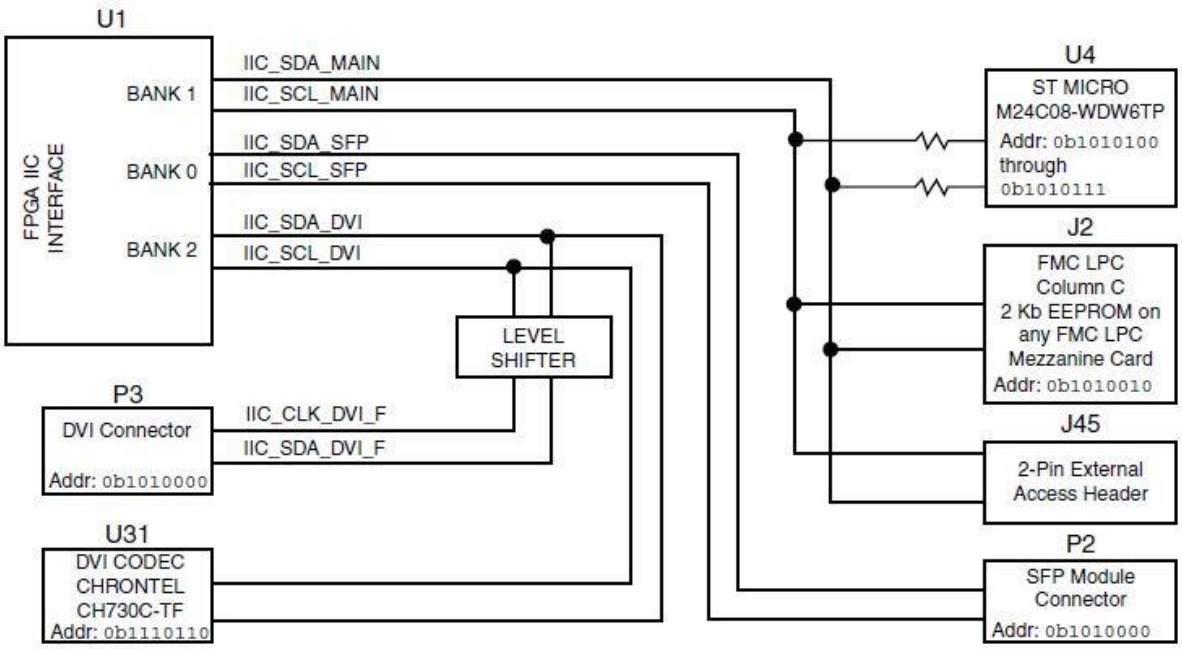

Figure 20 SP605 I2C Bus topology [10]

As shown, bank-0 interfaces with EEPROM. As this project mainly targets at validation of I2C communication between a control device and EEPROM as a slave, bank-0 interface forms the core of the work. Remaining banks and other two supported I2C interfaces are not in the scope of the project. One of the I2C interface FPGA used on the hardware platform comprises three I2C interfaces. The SP605 hosts 8-K ST Microelectronics M24C08-WDW6TP I2C parameter storage memory device. The IIC address of EEPROM is 0b1010100. The schematics of the EEPROM is shown in Figure 21.

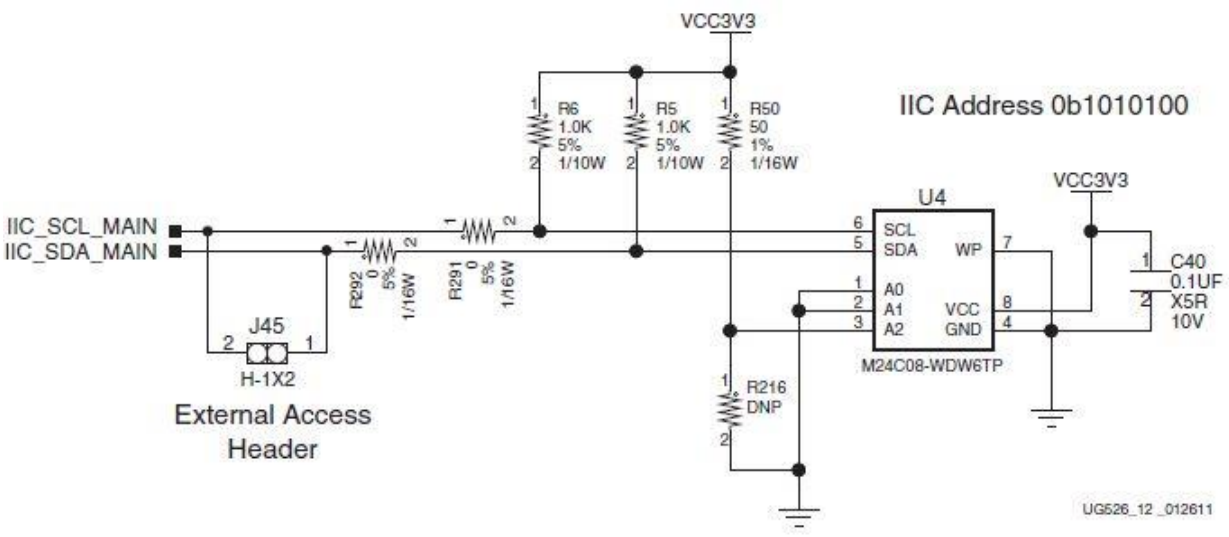

Figure 21 I2C compatible EEPROM available on SP605 [10]

Interfacing connections for EEPROM are listed in Table-6.

Schematic Netname	EEPROM I2C	
	Pin Number	Pin Name
Tied to GND	1	A0
Tied to GND	2	A1
Pulled up (0Ω) to VCC3V3 3	3	A2
IIC_SDA_MAIN	5	SDA
IIC_SCL_MAIN	6	SCL
Tied to GND	7	WP

Table 6 Interfacing connections for M24C08 EEPROM [10]

Chapter 5: Test Results

This chapter summarizes the results provided by the designed software platform for I2C validation. Results are discussed at length in this chapter. Configuration details are highlighted early in the chapter before executing any test cases to give enough insight about test cases. Test cases depicting various scenarios related to I2C communication are executed on the SP605 evaluation board using the developed scripts. In the end, saving in efforts using the proposed work against the conventional approach of validation is also described. All the test attempts to validate the I2C transaction in both standard as a well fast mode. These results help to determine the reliability of I2C communication between Spartan®-6 FPGA and EEPROM on SP605 board. In case of failure, it aids to take corrective approaches for debug purpose.

Throughout the proposed work, Spartan®-6 FPGA acts as a master device. Thus, clock-related aspects are validated within the base script itself. In case of a failure related to clock synchronization, the slave device may send not-acknowledgement (NACK) signal indicating the failure of communication between two devices. Thus, the base script and device script interact with each other during read and write operations. Test cases with regards to the data transfer at a unit level are described first. System tests incorporating the validation script is discussed in the later section. Throughout the development of the scripts, logging, Python's inbuilt library is used to include the application log with various messages. These messages are categorized into either information, error, and warning.

5.1 Configuration Information

As discussed in Chapter-3, three various scripts namely the base script, the device script, and validation script are designed as a part of the validation platform. All these scripts are executed successively on the hardware platform. Executing these scripts in various scenarios ensures the design compliance with the requirement of I2C communication.

Various test cases are designed to test the functionality developed inside these scripts thoroughly. These test cases also aim to validate I2C master controller of Spartan-6 FPGA design

correctness. Few of the test cases developed as a part of testing the designed software platform are presented in this section along with the results.

As EEPROM is considered as a slave device throughout the project, it essential to identify the I2C address of EEPROM from its datasheet. Slave address for the EEPROM used in the project is 0b101000X. X can be set to 0 for a write operation and 1 for a read operation. Thus, slave address in hex and binary for the EEPROM for both operations are listed in Table-7.

Operation	Slave Address in Binary	Slave Address in Hex
Write	0b10101000	0xA8
Read	0b10101001	0xA9

Table 7 Slave address of M24C08 EEPROM for read and write operations

All the test cases are developed to test read and write operation between master and slave devices. While write() doesn't return anything and read() returns the value read from the register, test cases are designed to be executed in debug and non-debug mode.

In debug mode, the entire flow of execution in any of the function is displayed on the console output. This is achieved by adding debug statements in the base and device scripts. Such statements are categorized by the logging library of Python. It is always a good idea to check the flow even if the function is not returning anything.

In both read () and write(), a variable named verbose is used to let the user determine the mode of execution for test cases. Verbose can be set to 1 to while calling the function to execute test case in debug mode. Debug mode helps to understand the entire flow of execution of the function by printing debug, error and warning statements. Debug mode can be proved to be useful in case of a failure of a test case to probe more information. By default, the value passed to verbose is 0. Thus, all the test cases developed to test the scripts within the proposed work are executed in normal mode.

5.2 Test Cases

Few of the test cases developed to test the scripts are listed in Table-8 with their expected outcomes. In all test cases, the Spartan®-6 I2C master controller acts as master and EEPROM is considered as the slave device for I2C communication.

Number	Test Case Description	Expected Result
1	Master attempts to perform byte write operation by writing one byte of data to register of the slave and then reads the same register	One byte of data read from the slave register must be the same as the data written.
2	Master attempts to communicate with a non-existing I2C device on the board by sending a wrong slave address	Master should receive NACK as no slave is on the receiving end to acknowledge the communication.
3	Master tries to initiate the data transfer with the slave being busy in servicing an interrupt	Master should wait for the slave to become free to proceed with data transfer.
4	Master attempts to write data on few of the address of a single page of the slave device	The slave should allow master writing the data on the addressed registers together using page write operation

Table 8 Test Cases developed for validation of I2C communication between master and slave devices

Test Case -1)

Executing the first test case in debug mode provides a console output shown in Figure 22.

```
>>> i2c.write(slave_addr = 0xa8, reg_addr = 0x11, data = 0xaa, addr_width = 1, verbose = 1)
08:45:54 --------------------I2C Write Operation-------------------
08:45:56 I2C Slave Address is set to (0xa8)
08:46:03 (0xaa) has been sent to the slave
08:46:07 Data has been successfully written to the slave
>>> time.sleep(1)
>>> i2c.read(slave_addr = 0xa8, reg_addr = 0x11,addr_width = 1, verbose = 1)
08:46:55 --------------------I2C Read Operation-------------------
08:46:58 I2C Slave Address is set to (0xa8)
08:47:04 (0xaa) has been read from the slave register address (0x11)
08:47:08 Data read from the register of EEPROM is :(0xaa)
```

Figure 22 Result of Test Case- 1 in debug mode

In the write (), the I2C address of the EEPROM and one of its register address is specified along with the data to be written. Argument of address width is set to 1 as the test case is directed to test byte write operation of EEPROM using the I2C bus.

In this test case, the I2C master controller of FPGA first writes the data 0xaa to the register of EEPROM owing an address of 0x11. As the verbose is set to 1, the information messages are displayed on the console to let the user know about the flow of execution.

If it would be set to 0, console output would be as shown in Figure 23 as write () returns nothing in normal mode.

```
>>> i2c.write(slave_addr = 0xa8, reg_addr = 0x11, data = 0xaa, addr_width = 1, verbose = 0)
>>> i2c.read(slave_addr = 0xa8, reg_addr = 0x11,addr_width = 1, verbose = 0)
08:51:24 Data read from the register of EEPROM is :(0xaa)
```

Figure 23 Result of Test Case- 1 in normal mode

Test Case -2)

The result of executing the second test case in debug mode is shown in Figure 24.

```
>>> i2c.write(slave_addr = 0x88, reg_addr = 0x11, data = 0xaa, addr_width = 1, verbose = 1)
08:55:12 -------------------I2C Write Operation-------------------
08:55:14 I2C Slave Address is set to (0x88)
08:55:22 @E: NACK received
08:55:25 @E: Failed to write slave address
08:55:26 @E: I2C communication failure
```

Figure 24 Result of Test Case- 2 in debug mode

Here, the master attempts to address the wrong slave by sending a slave address 0x88. No device on the I2C bus owns this address. Thus, the master receives a NACK signal instead of ACK signal. As it is an error, it is displayed in red with prefix @E indicating an error message. Thus, the proposed work is designed in a way to handle such errors.

Test Case -3)

The result of executing the third test case would be same as shown in Figure 22. In the proposed work, separate logic for interrupt handling is developed as a part of the base script in the case of a slave being busy in servicing the interrupt. Thus, the developed logic will ensure the free state of a slave before data transfer takes place. Hence, interrupt serviced by the slave device will be

handled by the base script which allows I2C transaction takes place once it clears the interrupt. The only difference lies in the execution of these two test cases is the amount of time each takes. As interrupt clear logic requires some time before ensuring the free state to master, the execution of the third test case requires more time for execution than the first one.

Test Case -4)

This test case targets to confirm the page write operation of EEPROM capable of writing 16 bytes of data within a single page. Thus, it is similar to byte write operation with total of 16 numbers of iteration across a page. As mentioned in the datasheet of M24C08 EEPROM, page write is limited to writing 16 bytes of data only. However, it allows writing data within a single page only irrespective of the number of bytes of data actually written to it.

In case of EEPROM address reaching a page boundary, the address is initialized back to the starting address of the page and data is written from there only. Hence, it results in overwriting of the data. This scenario is taken care of in the development of the device script. It prevents the page write operation from attempting to cross a page boundary.

To execute the test case, address range from 0x00 to 0xF forms a page is used for page write operation. Figure 25 shows the console output when 5 bytes of data is written on the addresses available in a single page using page write operations.

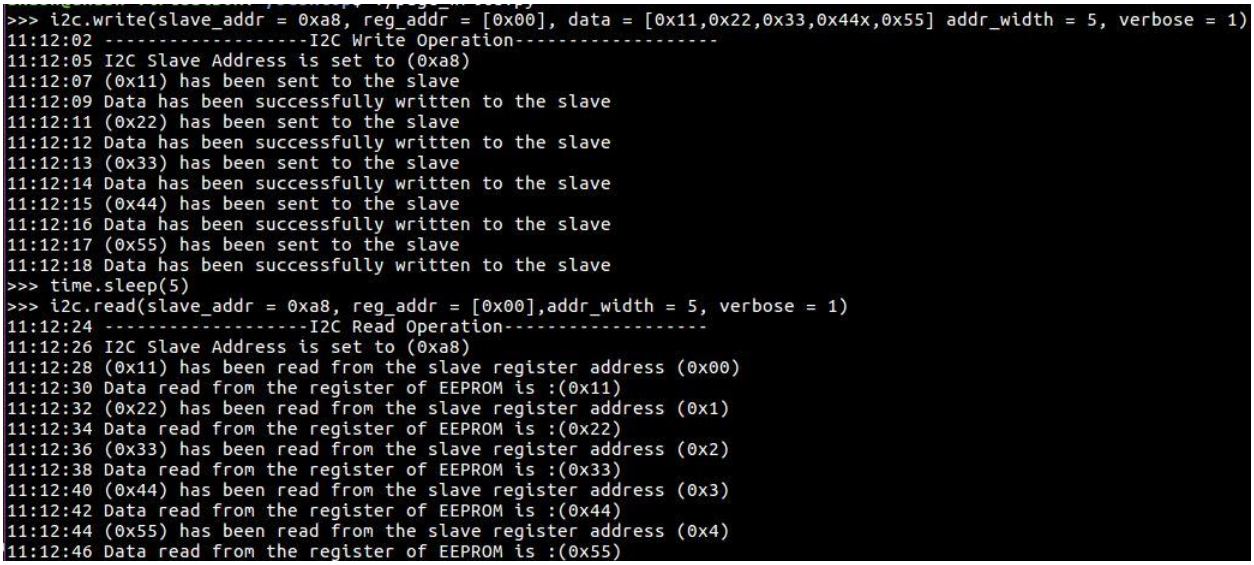

Figure 25 Result of Test Case- 4 in debug mode

In this case, the starting address of page and list of data written to the page is specified by the user. Argument specifying the address width helps to achieve the goal of page write operation. While executing this test case, setting the value of 16 to address width instructs the write() to go for a page write operation. Specifying the register address as a list than an integer also confirms the same. Between write and read operations, delay of 5 seconds is inserted so that EEPROM can store the data.

5.3 System Testing

Validation script is designed to achieve the goal of system testing. It uses the interaction between the base script and device script to validate the functionality of I2C between master and slave devices. It employs the mechanism of automation capable of executing random and directed tests to ensure the reliability of data transfer among devices.

While test cases discussed in the above section allows sufficient information about a single I2C transaction between master and slave devices, it is essential to test the reliability of data transfer. The reliability aspect can be validated by attempting multiple I2C transactions randomly as well as in a directed manner. If devices can go through multiple data transfer with introducing a delay

in between, performance and reliability of I2C master controller inside Spartan-6 FPGA can be confirmed.

In random testing, the I2C transaction is carried out between the I2C master controller and EEPROM randomly. Four to five registers are selected randomly, and random data is written on them. It is followed by reading these registers to confirm the read data is the same as data written. Both types of tests are implemented inside a validation script itself. System testing targets to confirm the byte write and successive read operations. Figure 26 illustrates the output of the validation script. It tests the I2C transaction between master and slave devices four times randomly and twice in a directed manner. In directed testing, a certain register is addressed for testing purpose. If the data read from the register is same as the data written on it, the test is considered as passed.

```
09:12:21------------------Random Tests for I2C transaction------------------
09:12:25 Iteration:1
09:12:27 Randomly selected register  : (0x05)
09:12:29 Randomly selected Data      : (0x45)
09:12:31 Data written on register    : (0x45)
09:12:33 Data Read from the register : (0x45)
09:12:35 Test Result :     PASS
09:12:38 Iteration:2
09:12:40 Randomly selected register  : (0x18)
09:12:42 Randomly selected Data      : (0x20)
09:12:44 Data written on register    : (0x20)
09:12:46 Data Read from the register : (0x20)
09:12:49  Test Result :     PASS
09:12:51 Iteration:3
09:12:53 Randomly selected register  : (0x3b)
09:12:55 Randomly selected Data      : (0x01)
09:12:57 Data written on register    : (0x01)
09:12:59 Data Read from the register : (0x01)
09:13:01 Test Result :     PASS
09:13:03 Iteration:4
09:13:05 Randomly selected register  : (0x24)
09:13:07 Randomly selected Data      : (0x50)
09:13:09 Data written on register    : (0x50)
09:13:11 Data Read from the register : (0x50)
09:13:14 Test Result :     PASS
09:13:16 ------------------Directed Tests for I2C transaction------------------
09:13:18 Iteration:1
09:13:20 Selected register           : (0x52)
09:13:22 Randomly selected Data      : (0x11)
09:13:24 Data written on register    : (0x11)
09:13:26 Data Read from the register : (0x11)
09:13:28 Test Result :     PASS
```

Figure 26 Console output of the validation script showing random and directed testing

5.4 Advantages of Test Automation

Validation refers to the design meeting the set of the specification. I2C validation can also be carried out by a conventional approach involving the clock measurement on I2C bus connecting master and slave devices. It may consume more time depending on the complexity of the design to be validated. Efforts may also vary from one circuit board to other as two same devices may not perform in similar fashion due to their discrete nature. Furthermore, manual validation like the described is prone to error. Determining the integrity of clock signals also imposes its own challenges. Such an approach works fine as long as the number of design to be validated is small and less complex.

Considering a case study of validating the Design of I2C master controller for total hundred number of evaluation boards. We also assume that a single I2C compatible device is on the board. Thus, data transfer must be validated between these two devices using the I2C bus.

While the conventional approach may consume time, the proposed work offers a solution that aids to save time in the validation cycle along with high reliability and robustness. Major efforts are involved in the initial stage only during which such software platform can be developed to automate the validation. Considering a conventional approach of manual validation may take up approximately 25-30 minutes to validate the I2C functionality for each evaluation board.

On other end, proposed test automation approach in this project hardly takes a couple of minutes to give information on the stability of I2C master controller of a control device on the evaluation board. Table-9 compares both approaches in terms of their offerings and limitations.

Parameter	**Conventional Approach**	**Test Automation Approach**
Validation efforts per evaluation board	30 minutes	2 minutes
Reliability	Low	High
Robustness	Less	More

Reusability	Almost zero	High
Software Development Efforts	Zero	High

Table 9 Comparison between Conventional and Proposed Test Automation approach

While efforts are high in the development of the software platform for test automation, but it offers an advantage of reusability on any of the evaluation board involving the similar set of devices. The same validation platform can be deployed on other evaluation boards involving Spartan-6 FPGA and EEPROM as well to quickly validate the I2C master controller design. This is certainly something that conventional approach lacks at. Furthermore, test automation provides a robust design validation. In case of any failure, it offers enough information to the user to debug the failure such as receiving the NACK can be related to either wrong slave address or busy state of the slave device. The conventional approach of manual validation cannot offer such advantages as it is a manual process. Even the error in case of a failure directs the user towards certain scenarios that can be helpful in the debug.

Table-10 lists the estimation for efforts involved in executing the I2C validation using the conventional approach against the proposed test automation approach for total hundred number of evaluation boards.

Approach	**Time Taken for Validation (100 boards)**
Conventional	50 hours
Test Automation	3.33 hours

Table 10 Saving in efforts for I2C validation using the proposed automation approach

Saving in efforts by adopting the test automation approach for validating 100 evaluation boards can be calculated as below. Test automation approach results in saving of 93.34% of efforts that required using the conventional approach. It is equally important to highlight a fact that software development in the proposed approach requires a significant amount of time before validation takes place. However, the advantages offered by the test automation shown in Table-3 outweigh the conventional approach.

Chapter 6: Conclusion and Future Scope

Validation of an IC design requires a good amount of efforts. Any bug escaped during the validation cycle of IC design may cost in a re-spin resulting in a delay to market for the product. Thus, the design must be validated correctly in a given span of time. As most of the design has the in-built interfaces to support communication protocols, it is important to validate the functional correctness of such interfaces. Test automation approach described in the proposed work is an effective solution saving time and efforts in the validation cycle for the interfaces.

Proposed work aims to validate the I2C controller of Spartan®-6 FPGA by means of test automation. Features offered by the described test automation such as reusability and reliability propel the application of such automation framework. In this project, I2C communication between single master and single slave devices has been validated by the developed software component. In particular, communication between the I2C controller of Spartan®-6 FPGA and I2C interface of M24C08 EEPROM has been validated by the development of three scripts.

1. The base script of I2C protocol implementation for the I2C controller of Spartan®-6 FPGA
2. The device script of M24C08 EEPROM for the I2C transaction
3. The test script to confirm I2C communication between master and slave in a random and directed manner.

Future Scope

In the future, automation can be enhanced by developing more device specific scripts for multiple I2C compatible devices on the evaluation board. Also, multiple master devices can also be employed to validate the multi-master mode supported by the I2C protocol. Single master multi-slave operation can also be validated by enhancing the proposed automation to validate the I2C interface of a master device.

References

[1] Ferrari, "System-on-a-chip verification~methodology and techniques," *IEEE Circuits and Devices Magazine,* vol. 18, no. 6, p. 39, 2002.

[2] G. Chen, "Research of Functional Verification Based on System," Ph.D dissertation, South China University of Technology, Guangzhou, 2010.

[3] A. Sahu, R. S. Mishra and P. G. Gour, "Design and Interfacing of High speed model of FPGA using I2C protocol," *International Journal of Computer Technology and Applications,* 2011.

[4] J. Valdez and J. Becker, "Understanding the I2C Bus," Texas Instruments, 2015.

[5] NXP Semiconductors, "UM10204: I2C-bus specification and user manual," 2014.

[6] Atmel, "TWI - 2-wire Serial Interface," in *ATmega328/P*, 2016, p. 260.

[7] Xilinx Inc., "XPS IIC Bus Interface", Product Specification, June 2011.

[8] ST Microelectronics," 8-Kbit serial I2C bus EEPROM," M24C08 Datasheet,Oct. 2017.

[9] "random — Generate pseudo-random numbers," Python Software Foundation, [Online]. Available: https://docs.python.org/2/library/random.html.

[10] Xillinx Inc.,"SP605 Hardware User Guide", 2012.