Makara Journal of Technology

Volume 26 | Issue 1 Article 5

4-30-2022

Design and Implementation of I2C Bus Protocol on Master and Slave Data Transfer Based on FPGA

Mohamad Khairi Ishak
School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Nibong Tebal, Pulau Penang 14300, Malaysia, khairiishak@usm.my

Meenal Pradeep Kumar
School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Nibong Tebal, Pulau Penang 14300, Malaysia

Recommended Citation

Ishak, Mohamad Khairi and Kumar, Meenal Pradeep (2022) "Design and Implementation of I2C Bus Protocol on Master and Slave Data Transfer Based on FPGA," *Makara Journal of Technology*: Vol. 26: Iss. 1, Article 5.
DOI: 10.7454/mst.v26i1.1416
Available at: https://scholarhub.ui.ac.id/mjt/vol26/iss1/5

Makara J. Technol. 26/1 (2022), 30–36
doi: 10.7454/mst.v26i1.1416

Design and Implementation of I2C Bus Protocol on Master and Slave Data Transfer Based on FPGA

Mohamad Khairi Ishak* and Meenal Pradeep Kumar

School of Electrical and Electronic Engineering, Universiti Sains Malaysia, Nibong Tebal, Pulau Penang 14300, Malaysia

*E-mail: khairiishak@usm.my

Abstract

This paper presents the design of the inter-integrated circuit (I2C) protocol with different types of features, such as combined messages, addressing modes, different data patterns and start addresses, clock frequencies, and types of modes between the field-programmable gate array (FPGA) and test card. Moreover, all these features can be randomized and run for long hours. The FPGA and the test card respectively act as master and slave. The design architecture comprises master and slave. The master generates a START condition, in which the serial data will transact between high to low levels and the serial clock will remain high. Then, the master also generates the STOP condition. Additionally, a few types of messaging modes, such as PIO read, PIO write, PIO write–read, and PIO read–write, are available. By contrast, the master also transfers and receives data to or from slave devices by using different addressing modes. The implemented addressing modes are 7 and 10 bits. This paper also focuses on randomizing the sent data byte and the start address. Particularly, data sending, reading, and writing operations are conducted and stimulated by capturing the signal using a logic analyzer. The signal is then examined and compared with the actual I2C protocol format. A stress test is performed by randomizing all the features and running for long hours (4 h). The stress test aims to stress the IP and ensure the health of IP.

Abstrak

Rancangan dan Implementasi Protokol Bus I2C pada Pemindahan Data *Master* dan *Slave* berdasarkan FPGA. Artikel ini menyajikan rancangan protokol *inter-integrated circuit* (I2C) dengan berbagai jenis fitur, seperti pesan gabungan, mode pengalamatan, pola data dan alamat awal yang berbeda, frekuensi clock, dan jenis mode antara field-programmable gate array (FPGA) dan kartu tes. Selain itu, semua fitur ini dapat diacak dan dijalankan selama berjam-jam. FPGA dan kartu tes masing-masing bertindak sebagai *master* dan *slave*. Arsitektur desain terdiri dari *master* dan *slave*. *Master* menghasilkan kondisi *START*, di mana data serial akan ditransaksikan antara tingkat tinggi ke rendah dan *serial clock* akan tetap tinggi. Kemudian, *master* juga membangkitkan kondisi *STOP*. Sebagai tambahan, tersedia beberapa jenis mode pesan, seperti *PIO read*, *PIO write*, *PIO write–read*, dan *PIO read–write*. Sebaliknya, *master* juga mentransfer dan menerima data ke atau dari perangkat *slave* dengan menggunakan mode pengalamatan yang berbeda. Mode pengalamatan yang diimplementasikan adalah 7 dan 10 bit. Artikel ini juga berfokus pada pengacakan bita data yang dikirim dan alamat awal. Khususnya, operasi pengiriman, pembacaan, dan penulisan data dilakukan dan dirangsang dengan menangkap sinyal menggunakan penganalisis logika. Sinyal tersebut kemudian diuji dan dibandingkan dengan format protokol I2C yang sebenarnya. Uji ketahanan dilakukan dengan mengacak semua fitur dan berjalan selama berjam-jam (4 jam). Uji ketahanan bertujuan untuk menekankan IP dan memastikan kesehatan IP.

Keywords: field-programmable gate array (FPGA), master, PIO read, PIO write, serial clock line (SCL), serial data line (SDA), Slave

1. Introduction

The Inter-Integrated Circuit (I2C) bus protocol was developed by Philips Electronics [1]. I2C allows communication between integrated circuits, and its communication is obtained from different manufacturers [2]. I2C is a synchronous bus protocol, which enables the fast device to communicate without any data loss. Moreover, I2C is commonly used for signal processing devices [3].

I2C protocol is a two-wire serial interface or even a bidirectional wire interface. These bidirectional wires comprise serial data (SDA) and serial clock (SCL). First, the SDA line is where data pass through when sent from one device to another [3]. Second, the SCL line is

generated by the master device and controlled during data sending and under the read operation. Both wires carry messages between the master and slave, which is connected in the bus [4]. This master or slave is the device that is connected on the bus. A unique address represents each device on the bus, and this device can act as a transmitter or a receiver [5].

I2C IP can run at various speeds, covering three speed categories: standard, fast mode, and high speed. The standard speed operates at 100 kHz. Meanwhile, the fast mode speed operates at 400 kHz, and the high speed operates at 3.4 MHz [6].

The implementation of the I2C bus protocol with different features, such as combined message, different types of mode transfer and speed, addressing mode, different patterns of data transmission, and number of bytes, is investigated in this paper. This protocol is also the best bus for control application, wherein devices may be added or removed from the system.

I2C is crucial for the future technological purpose of chip integration. Therefore, noise and signal integrity issues should be eliminated [1]. Such elimination is important because I2C validation has many drawbacks considering signal integrity and noise [3, 7]. Therefore, validation through a test card, which will replicate the actual device, will reduce the signal integrity and noise issues.

Moreover, cost and complexity issues remain major issues within the customers. The I2C device is one of the best solutions to reduce cost and complexity. Compared with other IPs, such as RS232, RS485, and QEP, I2C comprises only two signals, namely SDA and SCL [5, 8, 9]. By contrast, I2C IP is the most suitable when many devices are connected on the bus because it can reduce the cost and scale down the complexity of the circuit. On the other hand, the chip select line is commonly used in many other IPs, such as SPI IP. This chip select aims to choose one or more than one set of an integrated circuit which is connected on the same bus. Component and complexity will increase when the chip select line is used [4, 10, 11]; cost will also rise. This issue is eliminated by adding an addressing feature that can support 7- and 10-bit addressing features and a combined message that can support read–write and write–read transactions.

The most common problem is when the IP only runs with the implemented selected feature without any stress test. The health of an IP is crucial [12−14]. Therefore, the stress test is important to examine the healthiness of the IP and investigate the performance of data transfer and packet loss. Stress tests can be performed via randomization among all features that are implemented and run for long hours [2, 15−17].
Thus, the contribution of this project is to improve the health of the IP by running randomization with all the features that have been implemented for a long period. Furthermore, noise and signal integrity issues are currently addressed through the use of a test card that can replicate the actual device. Finally, the complexity and production cost will be reduced.

2. Design of I2C Bus Controller

Methodology. The software in this work comprises an Register-Transfer Level (RTL) code, which is used to code the flow of I2C as master and slave. Xilinx software tool is used to flash the FPGA. This research used the following: some hardware, such as Altera FPGA and protocol analyzer, to capture the SDA and SCL signals; test card, which acts as a slave to the I2C controller that acts as master. The flow chart of I2C acts as a master and the test card acts as a slave, as respectively shown in Figs. 1 and 2.

i-programming flow of test card as a master. Figure 4 shows that IC_Slave _Disable is set to 1 by disabling the slave and enabling IC_MASTER_MODE to set I2C as Master. Configuring IC_10bitADDR_MASTER enables 10- or 7-bit addressing modes. The slave address must be configured in the I2C Target Address Register in the IC_TAR bit after configuring the Master and addressing mode. This configured address is the target address for any master transaction.

The next step is to program the I2C Clock SCL High Count and Low Count Register according to the configuration speed, such as standard, fast, or high. This register must be set before the occurrence of any I2C bus operation. This step is crucial to ensure that the I2C bus has suitable IO timing and avoids signal integrity issues. Thereafter, the RX_FIFO and TX_FIFO threshold levels are set. Both levels control the number of entries for transmission and receiving. The valid range is from 0 to 255. When a value of 0 is set, the threshold is set for 1 entry; by contrast, when 255 is set, the threshold will be 256 entries.

After all the above configurations, I2C is enabled by configuring IC_Enable. Writing to the I2C_Data_CMD register is performed after the I2C is enabled. The function of the CMD bit is to control the performance of a read or write operation. This bit does not regulate the direction of I2C (whether master or slave) but only controls the direction when it acts as a master. This bit differentiates the write and read commands when a certain command arrives in the TX FIFO. A read command should be written for every byte that is to be received for the I2C to continue acknowledging reading; otherwise, I2C will stop acknowledging. Data to be transmitted will be configured on the DATA bit of the I2C_DATA_CMD register.

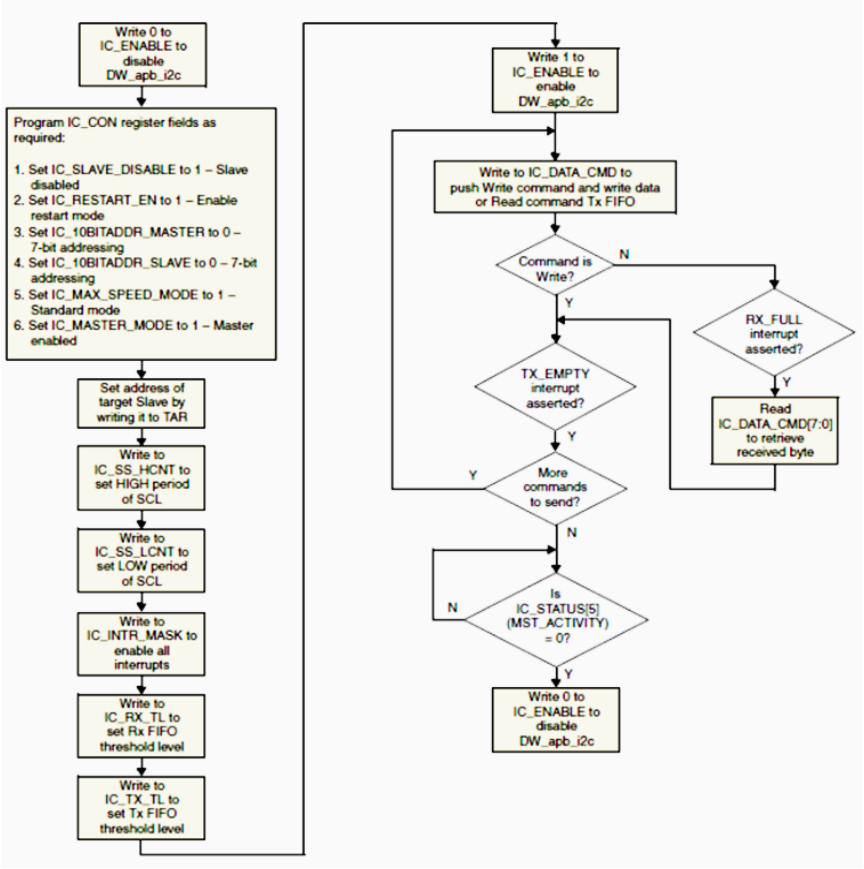

Figure 1. Flow Chart of I2C as Master

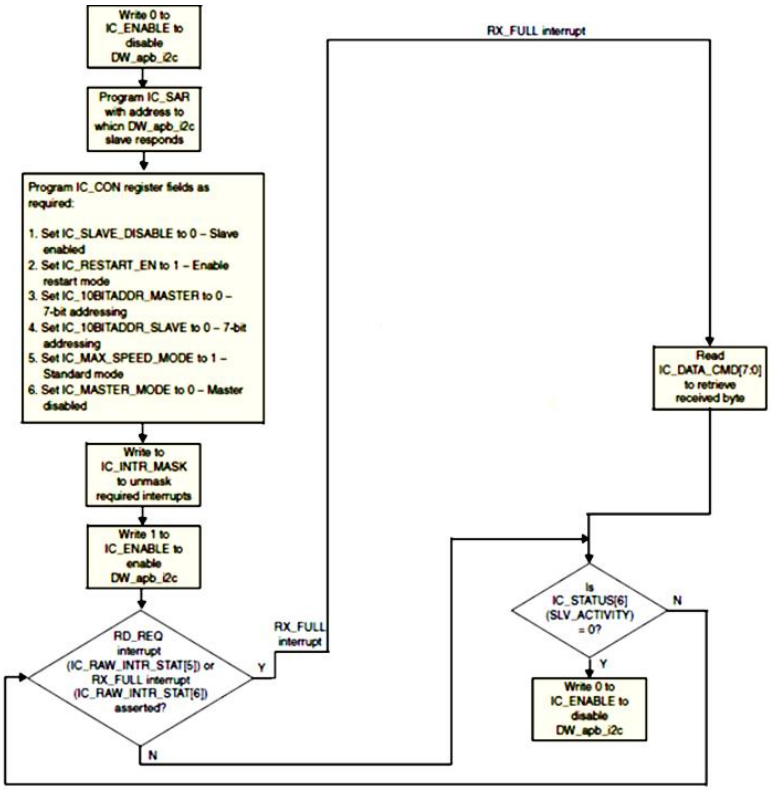

Figure 2. Flow Chart of Test Card as a Slave

ii-programming flow of test card act as slave. The test card is set as the slave as shown in Figure 2. First, the address where the I2C slave will respond is programmed. Then, IC_Slave _Disable is set to 0 by enabling the slave and disabling the master mode. IC_MASTER_MODE is configured as zero to disable the master mode. IC_10bitADDR_MASTER is configured to enable 10- or 7-bit addressing modes. If this bit is programmed to zero, then functioning is realized at 7-bit addressing mode. Next, the IC_MAX_SPEED_MODE is set to configure the speed of the I2C Bus. The IC_MAX_SPEED_MODE comprises two bits. When this mode is programmed to zero, the I2C bus will be functioning at standard speed, one as fast speed, and two as high speed. An effective finetuning of HCNT and LCNT is also necessary to configure the speed. An interrupt mask is written to unmask the required interrupt after configuring the slave and addressing modes. All the programmed configurations at the slave, such as the addressing mode and speed, should be similar to those programmed at the master mode. The I2C is then enabled by writing to IC_Enable to enable the I2C Bus.

The test card will start acting as the slave when the I2C Bus is enabled. The test card, which acts as the slave, will read the interrupt request by interpreting IC_RAW_INTR_STAT bits 5 and 6. If the status shows that the bus is RX Full Interrupt, then the slave will read the IC_DATA_CMD to retrieve the byte. Otherwise, the slave will read the status of the slave activity by interpreting IC_STATUS bit 6. This slave activity determines if the slave is active or inactive. The slave will read the interrupt register back when the slave activity is not equal to zero and will disable the slave by writing zero to IC_Enable if the slave activity is one.

Design implementation. The hardware used is FPGA and test card. FPGA will act as a master, while the test card will act as a slave. SDA and SCL lines are respectively connected from the FPGA and the test card. Figure 3 shows the FPGA and test card hardware connection.

When the mode is programmed as PIO write, the FPGA will act as the master, while the test card will act as a slave. FPGA will then program a bit to indicate a write transaction. When the mode is programmed as PIO read, the FPGA will remain as the master, but the FPGA will be programmed as a read transaction. The Verilog code for the I2C controller, which had been developed and assigned with SDA and SCL, is transformed into a bit file. This bit file is then flashed onto the FPGA using the Xilinx software tool and Xilinx hardware. The Xilinx hardware tool is connected to the FPGA to flash the Verilog code that is being developed. The test card is programmed as a slave using the USB Blaster after the Verilog code has been programmed. The transaction of the I2C protocol will run, and the signal is captured

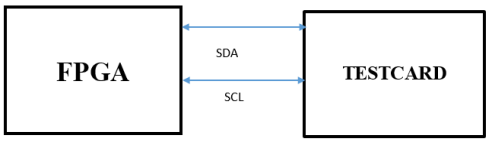

Figure 3. FPGA and Test Card Connection

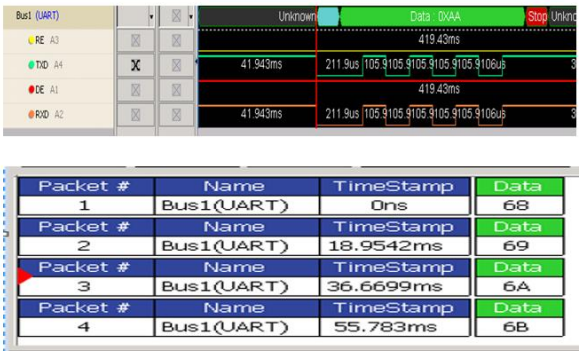

Figure 4. Logic Analyzer Software

using a logic analyzer. The signal is discussed by comparing it with the actual protocol format. Figure 4 shows the logic analyzer software that captures the signal and data packet.

3. Simulation Results

The protocol analyzer is used to capture signals which are compared with the actual protocol format and then discussed. Eleven different colors of protocol analyzer are available to describe each operation, such as start, data, and slave address. Figure 5 describes each color with its condition.

PIO write and PIO read. For the PIO write mode transfer, the I2C controller is the master while the test card is the slave. The I2C controller, which acts as the master, will begin by sending a START transaction. The start pattern, which is known as the slave address, is then generated by the master. The slave sends an Address ACK bit, and a write bit is generated after the master receives this bit. Afterward, the data are sent to the slave. Data ACK bit, which is generated by the slave, will be available after every data. The data NACK bit is sent by the slave after completing the data transfer. The master will generate a STOP condition after receiving the Data NACK bit. Figure 6 shows the PIO write transaction.

For the PIO read transaction, a read bit is generated by the master after receiving the Address ACK bit sent by the slave. The data are then sent to the slave. The ACK bit, which is generated by the slave, will be available after every data. Data NACK bit is sent by the slave after completing the data transfer. The master will generate a STOP condition after receiving the data NACK bit. Figure 7 shows the PIO read transaction.

Combined message. PIO write–read transaction is a combined message mode. The first part of the message is the write, followed by a restart bit and then a read transaction. Meanwhile, PIO read–write transaction is also a combined message mode. The first part is the read transaction followed by the write transaction. Figures 8 and 9 respectively show the PIO write–read and the PIO read–write transaction signals.

Addressing mode. Two types of addressing modes are available: 7- and 10-bit addressing modes. Figures 10 and 11 respectively show the 7- and 10-bit addressing mode signals.

Data byte. Data byte randomizes from 0 to 240 bytes. Figure 12 shows the randomization of the number of bytes of 8, 16, 32, 64, and 240 data bytes.

Start address. The start address, which is also known as the slave address, has been randomized. Figure 13 shows the randomization of the slave address at 0 × 79, 0 × 51, and 0 × 7f.

Figure 16: Different types of start addresses: (a) −0 × 79 slave address, (b) −0 × 51 slave address, (c) −0 × 7f slave address.

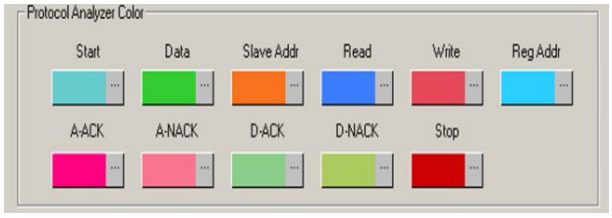

Figure 5. Color Coding for each condition

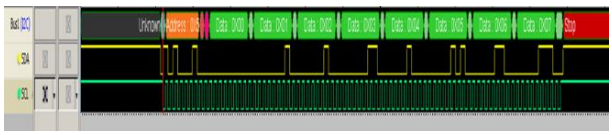

Figure 6. PIO Write Transaction

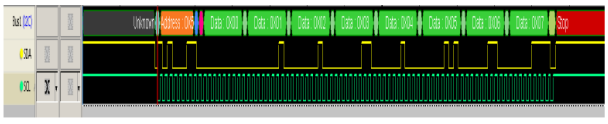

Figure 7. PIO Read Transaction

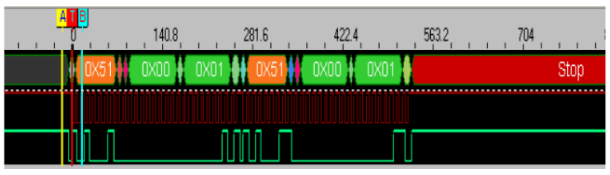

Figure 8. PIO Write–read Transaction

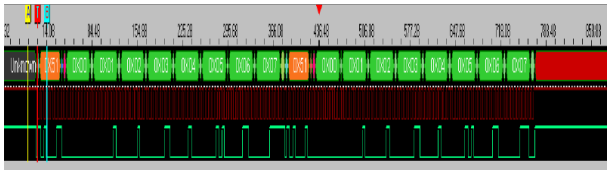

Figure 9. PIO Read–write Transaction

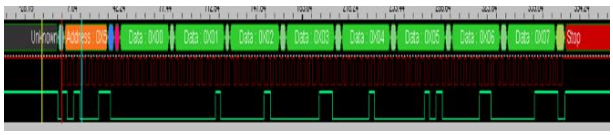

Figure 10. 7-bit Addressing Mode

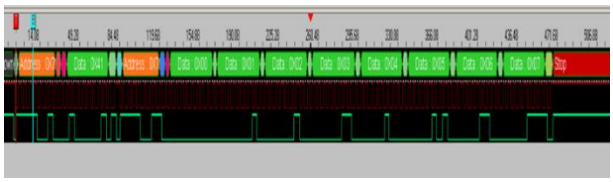

Figure 11. 10-bit Addressing Mode

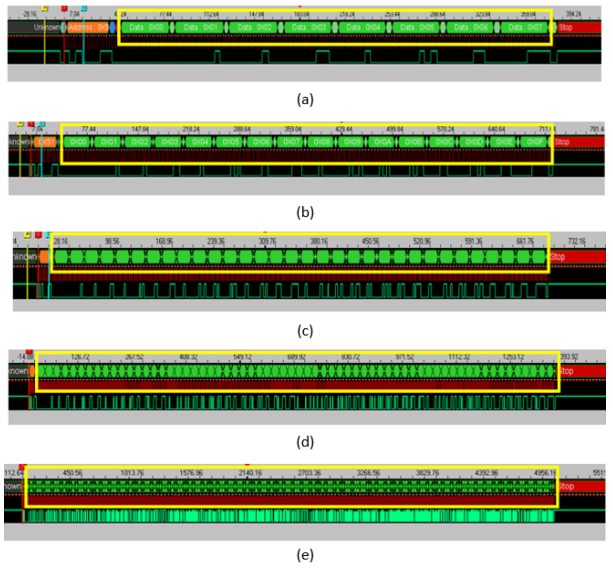

Figure 12. Different Numbers of Bytes: (a) −8 Data Byte, (b) −16 Data Byte, (c) −32 Data Byte, (d) −64 Data Bye, (e) −240 Data Bytes

Pattern. Data patterns are also randomized from increment, gray code, and Fibonacci pattern. The three patterns are randomized during the long-hour regression test in Section G.

Long-hour regression. An automation regression was run for long hours. The minimum run is one. This automation regression will randomize the mode of transfer by choosing the write, read, write–read, or read–write transactions. This regression will also randomize between 7- and 10-bit addressing modes. Moreover,

the data pattern will perform randomization by selecting between gray code, increment, or Fibonacci. Meanwhile, this automation will send data with different types of speed, such as standard, fast, or high. The slave address, which is also known as the start address, as well as the data byte, will also be randomized throughout the regression process. Table 1 shows a summary of the randomization parameters.

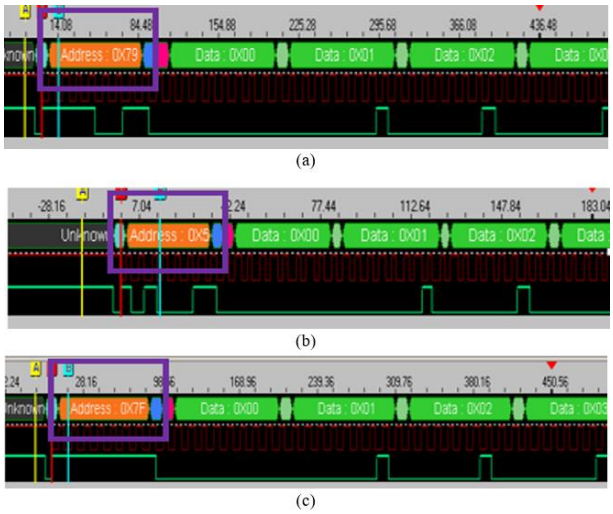

Figure 13. Different Types of Start Addresses: (a) −0 × 79 Slave Address, (b) −0 × 51 Slave Address, (c) −0 × 7f Slave Address

Table 1. Randomization Parameters

Randomization Configuration	Parameter
Mode Transfer	Write Transaction
	Read Transaction
Combined Message	Write–Read Transaction
	Read–Write Transaction
Addressing Mode	7-bit addressing mode
	10-bit addressing mode
Clock Frequency	Standard Speed
	Fast Speed
	High Speed
Start Address	Randomization according to the addressing mode
Number of Bytes	0–240 bytes
Pattern	Increment
	Gray Code
	Fibonacci

Table 2. Long-hour Regression Results

Long-hour Regression	Test Result	No of Cycles
1 h	Pass	11,711
2 h	Pass	23,376
3 h	Pass	28,820
4 h	Pass	44,464

Table 2 shows the summary of long-hour regression results by randomizing all the parameters with the number of cycles. Four hours long has been identified in this work as sufficient time based on the randomized parameters and computer performance. The test result that will be displayed is either Pass or Fail. A Pass means that the I2C bus had completed the I2C transaction. The number of cycles refers to the number of I2C transactions with different parameters. For example, cycle number 1, which is known as the first transaction, indicates read transaction with the following conditions: 7-bit addressing mode, 100 bytes of data, slave address of 0 × 51, standard speed, and increment data pattern. The second transaction also indicates read transaction but under the following conditions: 10-bit addressing mode, 20 data bytes, fast speed, and gray code data pattern.

4. Conclusions

This research proposed a methodology for the implementation of the I2C bus protocol with different types of features, such as combined message, addressing mode, data pattern, number of data bytes, slave address, and speed. This methodology is designed between the Xilinx FPGA and the test card, which can act as the actual device. All the implemented features can be randomized and run for long hours.

The code that acts as the I2C slave and master, as well as different features, such as addressing bits, modes, and various slave addresses, have been developed in Verilog code. This code is then run on FPGA (master) and test card (slave). The protocol analyzer is used to capture signals. The signals of all the respective features implemented are compared with the actual protocol format.

A stress test, known as the automation regression, is conducted for the healthiness of the I2C IP. The minimum run is 1 h. This regression test randomizes all the features and is run for long hours. The stress test is implemented to ensure the absence of packet loss throughout the long-hour regression. The implementation of this stress test can increase the coverage of the implemented features and reduce the bug escape of I2C IP.

References

[1] R.R.C., R.A. Kumar, Int. J. Sci. Eng. Res. 5/3 (2014) 130.
[2] The I2C Bus Specification, Philips Semiconductor, 2010.
[3] A. Anagha, M. Mathurakani, 2016 International Conference on Communication and Signal Processing (ICCSP), India, 2016, p.2124.
[4] R.K. Megalingam, J.M. Varghese, S.A. Anil, 2016 International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, p.1.
[5] I.G. Cristian, 2015 IEEE 21th international Symposium/or Design and Technology in Electronic Packaging (SIITME), Romania, 2015, p.131.
[6] J. Chhikara, R. Sinha, S. Kaila, 2015 IEEE International Conference on Computational Intelligence & Communication Technology, 2015, p.235.
[7] V. Jan, Design of a digital I2C Slave IP Block, 2012.
[8] M.P. Kumar, Int. J. Innov. Res. Comput. Commun. Eng. 2/3 (2013) 4.
[9] H. Kaneriya, S. Jagtap, Int. J. Recent Innov. Trends Comput. Comm. 3/5 (2015) 4.
[10] S. Tripti, Int. J. Res. Appl. Sci. Technol. 2/12 (2014) 4.
[11] T.L.-del Río, G. Juarez-Gracia, L.N. Oliva-Moreno, Res. Comput. Sci. 75 (2014) 31.
[12] N. Gupta, V. Gupta, Int. J. Innov. Res. Electr. Electron. Instrum. Contr. Eng. 1/9 (2013) 4.
[13] B. Debasis, Design an Modeling of I2C Bus Controller, 2011.
[14] I2C Bus, Tracii XL 2.0.
[15] A.S. Tadkal, P. Patil, Int. J. Electr. Electron. Comouter Syst. 2/3 (2014).
[16] Xilinx Inc, Spartan-3E FPGA Family Data Sheet, DS312 Product Specification, 2013.
[17] L. Bacciarelli, G. Lucia, S. Saponara, L. Fanucci, M. Forliti, Res. Microelectron. Electr. (2006) 373.