Innovative and rapid design for I2C (inter-integrated circuit) interface communication of FPGA chips

Cheng Shion Shieh
Department of Electronic Engineering, Far East University, Taiwan

E-mail: csshieh@mail.feu.edu.tw

Abstract. A new design flow is proposed to greatly improve the efficiency and accuracy of the HDL program design. This paper successfully completes the FPGA (Field-Programmable Gate Array) chip’s external communication I2C interface design not traditionally one. FPGA chip is often used in mobile phones, military, aerospace or other industries. Even INTEL and Microsoft and other large companies have begun to acquire FPGA-related companies, showing its importance. But the FPGA language is written in the Hardware Description Language (HDL) which is a mechanical engineering staff obstacles. Therefore, this article makes use of the MATLAB / SIMULINK tool, which is familiar with the engineering staff, to complete the FPGA chip’s I2C interface design. The advantage is that the functional simulation and HDL program can be completed at the same time, do not have to perform the function after the simulation before designing HDL program. The traditional design wastes many times to revise back and forth between function simulation and HDL program design. Finally an experimental illustration is proposed to verify the proposed design of the I2C communication interface for an HDL program on the read and write experiments of Electrically-Erasable Programmable Read-Only Memory (EEPROM).

1. Introduction

With the rapid development of electronic technology, many ICs need to communicate with each other. For this reason, in 1982 Philips Semiconductors of the Netherlands developed a bidirectional I2C (Inter-Integrated Circuit) bus [1] to reduce the number of microcontrollers or CPUs to connect many low-speed peripherals with fewer pins. Its advantages are simplicity and effectiveness. The purpose of the initial development of the I2C bus was to develop audio and video equipment that could be more easily connected to the CPU. Today, it has become the standard for inter-chip low-speed serial communication and is widely used in entertainment, consumer electronics and control electronics. [2].

Recently, FPGA chips have been widely used in industrial control [3-5], mobile phones, artificial intelligence, deep learning [6]. It shows the importance of FPGA chip design that Intel and Microsoft's technology companies have made great efforts to acquire related FPGA companies. FPGA chips often need to communicate with peripheral I2C-interfaced chips. Although there are specialized I2C bus interface chips on the market, most of them have fixed performance, single function, and inconvenient use. Therefore, it is necessary to redesign an HDL program that can implement the I2C bus interface read/write function on the FPGA chip according to the I2C bus protocol. HDL programming is an obstacle for most engineers because it is time-consuming and error-prone. Not easy to complete.

Most of the engineers are more familiar with the graphic control software Matlab/Simulink [7], which can directly simulate the program design right or wrong. As long as the simulation of the
Matlab/Simulink program is verified, this Matlab/Simulink program can be directly converted to the corresponding HDL code. Then use FPGA-specific software Quartus II to compile and download the HDL program. The design flow mentioned in this paper combines design and simulation to quickly achieve the desired control results which not only saves time but also helps to clarify of programming concepts.

This article is divided into: Section II describes the I2C communication bus agreement and design, the third section proposes the I2C’s implementation by Simulink tool and QuartusII one, and the fourth section describes the I2C read design concept and proposes the implementation in Matlab/Simulink platform. Section 5 verifies FPGA’s I2C Read/Write function on real circuit equipment.

2. I2C communication agreement and design [8]
I2C data communication consists of a data line (SDIN) and a clock line (SCLK). A master chip controls multiple slave ones. The master chip can be an FPGA, DSP, or MCU. The slave can be an audio codec chip, a TV Decoder chip, or an EEPROM chip. The schematic diagram is shown in Figure 1. The start condition of the I2C bus is that when both SCLK and SDIN are high, the SDIN first changes from high to low, and then SCLK changes from high to low. The stop condition is when both SCLK and SDIN signals are low, SCLK changes from low to high, and then SDIN changes from low to high, as shown in Figure 2.

![Figure 1. I2C master-slave device block.](image1)

![Figure 2. Start (ST) and End (SP) signals of the I2C bus.](image2)

The purpose of this paper is to develop a new I2C design process for master device FPGA. Verify the proposed process by firstly writing a piece of 8-bit data from the master device to the slave one, and then the master device FPGA will read the data from the slave device EEPROM to check master writing action right or wrong. That is, complete the write and read actions from the master device FPGA to the slave one EEPROM. This paper does not use the traditional program-writing method, but uses a Matlab/Simulink design in the form of graphic control design, and converts these graphic control programs into HDL codes, which are described in the following sections.
3. I2C write action of the master device

The master writes an 8-bit data to a certain register of the slave device. Firstly, find out the device address of the slave device and the address of the slave device register to be placed. The control style is that the master device generates a start signal ST. Then, the master control terminal sends a set of device address 7 bits to be controlled, plus a write (W) of 1 bit (write set to 0). At this point, the master terminal waits for the slave device to respond with an ACK1 signal (the master sends a response from the device to send ACK1=0). Once the master device receives an ACK1 signal from the slave, the master sends a register address to the slave and waits for the its response to an ACK2 signal. When the master device receives ACK2, the master sends 8-bit data to be written, and then the master terminal waits for the slave device to respond with an ACK3 signal again. If the responses of the three ACK1–ACK3 signals from slave device are all zero, it indicates that the signal transmitted from the master to the slave device is normal.

| Master | Slave |
|--------|-------|
| ST     | ACK1  |
| Device address[8:0] | ACK2  |
| W=0 ACK1=1 | ACK3  |
| Register address[7:0] |  |
| Data[7:0] | ACK1=0 |
| 27 bits | ACK2=0 |
| ACK3=1 | SP    |

**Figure 3.** 8-bit I2C write operation bitmap.

Finally, the master sends the end signal SP to the slave one. (The interpretation of the ST and SP signals is shown in Figure 2). The above description is the process of writing data to the registered address of the slave address (Device address [6:0]). The transmitting bit diagram is shown in Figure 3. The I2C of master writing program is referred to conference [1] which block diagram is shown in Figure 4. The function of each sub-block is described as follows:

(i) 「Counter」: Because the master control element in Figure 3 has 27 bits to pass to the slave device, there must be a counter from 0 to 26 to count the current bit transmission. We design the counter by Simulink, as in Figure 5. After the simulation result is correct, the graphical program can be directly converted to the corresponding Verilog language as shown in Figure 6. There is no need to manually rewrite the Verilog language, which significantly saves time in programming the Verilog language.

**Figure 4.** Master I2C write diagram.
Figure 5. Design and Simulation of 0~16 counters on Simulink Platform.

Figure 6. Verilog program of 0~16 counter converted by Simulink program Figure 5.

(ii) 「Shift register with parallel-bit input/serial single-bit output」: The function of this block is left-shift action per clock. That is, the parallel input is a 27-bit character and output is a single bit output. The input 27th bit is output with left rotation in every clock time. The Simulink platform design and simulation are shown in Figure 7.

Figure 7. Design of shifting register with parallel-bit input/series-bit outputs in Simulink platform.

(iii) 「State machine for I2C writing」: The master of the I2C communication writes the characters of the state of each clock of the slave device shown in Figure 4. The states of state-machine are X_IDLE, X_GO, X_START, X_WAIT, X_SHIFT, X_STOP, X_FINAL, X_END which timing
diagrams are shown in Figure 8. The signal values corresponding to each state and the next state are summarized in Table 1. According to Table 1, we design the program on the Simulink platform as shown in Figure 9 which simulation waveform is shown in Figure 10.

Figure 8. The write state machine timing diagram of the master I2C [8].

Figure 9. Design of the state machine of I2C write in the Simulink platform.

Figure 10. Simulation of the state machine of I2C writes for Figure 9.
(iv) 「Loadable Registers」: If reset=1, the output ACK1=0, ACK2=0, ACK3=0. Otherwise if ACK1=0, ACK2=0, ACK3=0 if resetACK=1, otherwise ACK1=SDIN if InACK1=1, ACK2=SDIN if InACK2=1, ACK3=SDIN if InACK3=3. In this way, it can be confirmed whether the slave device has received the signal from the master device. If so, ACK=0 in the block of Figure 4. This block of Simulink simulation is shown in Figure 11.

(v) 「Generate SCLK」: The relationship between the three inputs (CLK, SHEN, SCLK_Temp) and one output SCLK is when SHEN is greater than or equal to 1 SCLK = -CLK, otherwise SCLK = SCLK_Temp. The Simulink program is shown in Figure 12. After the Simulink program simulation of I2C master write diagram Figure 4 confirms that the simulation results are correct, the Simulink program can be directly converted into Verilog language and then integrated into the Quartus II software platform shown as Figure 13 which is a successfully compiled program. This Quartus II, FPGA-specific software, project is a fully Verilog language program for I2C communication.

| Table 1. input/output relationships between states of I2C write state machine. |
|-------------------------------|-------------------------------|-------------------------------|-------------------------------|
| Current state | Counter value | trig er | Read/Write | Output | Next state |
| X_IDLE | X | 0 | 0(W) | 1 | 1 | 1'b1 | 1 | 3'b000 | 0 | 0 | 0 | 0 | 0 | X_IDLE |
| X_IDLE | X | 1 | 0(W) | 1 | 1 | 1'b1 | 1 | 3'b000 | 0 | 0 | 0 | 0 | 0 | X_GO |
| X_GO | X | X | 0(W) | 1 | 1 | 1'b1 | 0 | 3'b000 | 0 | 0 | 0 | 1 | 0 | X_STA RT |
| X_STA RT | X | X | 0(W) | 1 | 1 | 1'b0 | 0 | 3'b000 | 0 | 0 | 0 | 0 | 0 | X_WAI T |
| X_WAI T | X | X | 0(W) | 1 | 1 | 1'b0 | 0 | 3'b000 | 0 | 0 | 0 | 0 | 0 | X_SHIF T |
| X_SHIF T | Counting for not 8 and not 17 and not 26) | | | | | | | | | | | |
| X_IDLE | 8 | X | 0(W) | 0 | ~CLK | 1'b0 | 3'b000 | 1 | 0 | 0 | 0 | 1 | X_SHIF T |
| X_SHIF T | 17 | X | 0(W) | 0 | ~CLK | 1'b0 | 3'b001 | 1 | 1 | 0 | 0 | 1 | X_STOP |
| X_STOP | X | X | 0(W) | 0 | 1'b0 | 1'b0 | 0 | 3'b000 | 0 | 0 | 0 | 0 | 0 | X_FINA L |
| X_FINA L | X | X | 0(W) | 1 | 1'b1 | 1'b1 | 0 | 3'b000 | 0 | 0 | 0 | 0 | 0 | X_END |
| X_END | X | X | 0(W) | 1 | 1'b1 | 1'b1 | 0 | 3'b000 | 0 | 0 | 1 | 0 | 0 | X_IDLE |
4. I2C read action of the master device

Read an 8-bit data via the register address (Register address [7:0]) of the slave device. The control method is to generate a start signal ST from the master device first. Then, the master sends a set of address 7 bits (Device address [6:0]) plus a write (W) 1 bit (write set to 0). At this point, the master waits for slave device to respond with an ACK1 signal (the master sends a response from the device to send ACK1=0). Then the master sends a register address [7:0] to the register and waits for the slave to respond with an ACK2 signal (the master sends 1 to send a response with ACK2=0 from the device).

The master sends 1 bit named “Sr”. This bit represents the start signal (SCLK = 1 indicates that the SDIN should have a moving action from 1 to 0.). Then the master control again sends a set of 7 address bits (Device address[6:0]) to be controlled, plus one read (R) bit (read set to 1) and waits for the slave responds with an ACK3 signal (the slave response ACK3=0 when the master sends
ACK³=1). The master sends 8 bits 11111111 to the slave device, waits for the slave respond to these 8 bits (the 8-bit value to be read), and then the master sends NACK=1 to the slave device.

Finally the master sends an end signal of the SP to complete the reading of the data from the loadable register. Its transmission bitmap is shown in Figure 14. Based on the above discussion, this paper proposes a program for I²C master control reading. The block diagram is shown in Figure 15. The function of each sub-block is described as follows:

Figure 14. 8-bit I²C read operation bitmap.

(i) 「Counter」: Since there are 38 bits to be controlled, a counter from 0 to 37 is needed to design like the module in Figure 5 which is easy to program for tuning the inside parameters to obtain counter function.

(ii) 「Shift register with parallel bits input series bit output」: The input is a 38-bit character (0 to 37 characters). Each clock performs a left-shifting action, allowing only the 37th bit of the highest bit to pass through a single bit, which is the 37th-bit output per clock time. Its design diagram is similar to Figure 7, the number of transmitting bits is 38.

(iii) 「The state machine of I²C read」: Take use of the state machine to design the master of I²C communication that reads the state of the slave device data. Similar to figure 8 shows the timing chart. The status of the reading of the master is divided into 8 states, X_IDLE, X.GO, X.START, X_WAIT, X_SHIFT, X_STOP, X_FINAL, X.END. The corresponding values of the states and the next state are summarized in Table 2. Complete the state machine design of Table 2 in the Simulink platform shown as Figure 16.

Figure 15. The master I²C read diagram.
(iv) 「The 19th bit Q[18] indicates restart」: When count through the counter (from 0 to 37) to 19, we let a clock with a phase shift of 90 degrees be treated as a single bit Q[18] so that is a restart action. The remaining modules in Figure 15 all have the same function as the module written by the master I2C in Figure 4.

The master I2C block Figure 15 is also simulated with Simulink programming. After verifying simulation, they can be converted into Verilog language and integrated into Quartus II software as shown in Figure 17. Projects in the Quartus II software are I2C-read communications programs.

**Figure 16.** Design of the I2C of read state machine on the Simulink platform.

**Figure 17.** The corresponding Verilog block integration of Figure 15.

**Table 2.** input/output relationships between states of I2C read state machine.

| State    | Counter value | Trigger | Read/Write | SCLK_Temp | SCLK | SDI | SD0 | [IdnACK1, | IdnACK2, | IdnACK3] | Bitcount | Rst bitcount | RstACK | LDE | SHE | Next State |
|----------|---------------|---------|------------|-----------|------|-----|-----| {IdnACK1,| {IdnACK2,| {IdnACK3}  |
|          |               |         |            |           |      |     |     | | | } | | | | |
| X_IDLE   | X             | 0       | 1(R)       | 1         | 1    | 1'b1| 1   | 3'b000  | 0       | 0       | 0       | 0       | 0       | X_IDLE  |
5. The actual circuit verification

In this paper, the DE0-Nano verification platform as shown in Figure 18. The Cyclone IV FPGA chip and the 24AA02 EEPROM (Electronically Erasable Read-Only Memory) whose element address is 1010000. The data written to EEPROM temporary address one is (170)10=(10101010)2. The master device transmits data with 27-bits [26:0] which binary expression is (101000010000001110101010101)2. The I2C writing simulation is shown in Figure 19. From ACK=0

![Figure 18. DE0-Nano FPGA verification platform.](image)
in Figure 19, the data sending from the master to the slave device is successful. That is, the master-side FPGA chip has completed the I2C write operation.

The reading program from FPGA to I2C is shown in Figure 17. We want the data previously written into the EEPROM be read out. The simulation is as shown in Figure 20. ACK = 0 indicates that the slave did receive control of the I2C read issued by the master, and also actually read the value 170 just written from the EEPROM from the signal probe Figure 20. From the simulation of Figures 19 and 20, it can be concluded that the architectural design is feasible.

6. Conclusions
This paper proposes an I2C write and read of architectural design for the FPGA master device. We also successfully completes its functional verification. The special point of this article is that it is not written directly in the Verilog language. Instead, all the sub-block functions are verified using the Simulink program. Then turn into the Verilog language that FPGA needs. Finally, use a logic analyzer to confirm sub-block function right or wrong. This design process is based on the Simulink block language. This is a way for engineers to shorten the writing time of the Verilog language. It is also a simulation and verification done at once, and there is a faster and clearer design step for the I2C design.

References
[1] Microchip Technology Inc. 2000, The I2C Bus Specification Version 2.1
   https://www.csd.uoc.gr/~hy428/reading/i2c_spec.pdf
[2] Stephen B and Fischer R L 2000 An I2C Network Protocol for Environmental Monitoring, AN736
   http://ww1.microchip.com/downloads/en/AppNotes/00736a.pdf
[3] Venkata Y. and Bin W. 2016 Predictive Control of Wind Energy Conversion Systems (John Wiley & Sons, Inc., Hoboken, New Jersey)
[4] Bai H, Liu C and Rathore A K 2019 IEEE Transactions on Industrial Electronics 66 pp 6581—6591
[5] Ana Dalia P A, Brisbane O M Esteban T C, Jesus Manuel M P and Luis G F 2019 Communications In Nonlinear Science And Numerical Simulation 72 pp 516 – 527

[6] Chen Z, He N and Huang Y 2019 Genomics Proteomics & Bioinformatics 16 pp 451 – 459

[7] MATLAB/SIMULINK TOOL, https://cn.mathworks.com/products/matlab.html

[8] Logical Circuit Design DE2-115 Practical Collection, Terasic Technology Publishing