Implementation of the communication protocols SPI and I2C using a FPGA by the HDL-Verilog language

Tatiana Leal-del Río[1], Gustavo Juarez-Gracia[1], L. Noé Oliva-Moreno[2]

[1] CICATA, Legaria, México
[2] ESCOM, México
mileydy.1125@gmail.coma, agjuarez@ipn.mxb,loliva@ipn.mx

Abstract. Currently, the most used serial communication protocols to exchange information between different electronic embedded devices are the SPI and I2C. This paper describes the development and implementation of these protocols using a FPGA card. For the implementation of each protocol, it was taken into account different modes of operation, such as master/slave mode sending or pending data mode. For the implementation of the I2C protocol was necessary to perform a tri-state buffer, which makes a bidirectional data line for a successful communication between devices, allowing to take advantage of these sources provided by the FPGA. Verilog is a hardware description language better known as HDL and it was used in the work to implement and simulate these communication protocols with the software version 14.7 of Xilinx ISE Design Suite.

1 Intorduction

Nowaday the integration of different embedded electronic modules include at least some of these functions: intelligent control, general purpose circuits, analog and digital I/O data ports, volatile memories (RAM), non-volatile memories (EEPROM, FLASH), real time clocks, ADC, among others. The integration is possible because of the development of different kind of wired and wireless communications.

The integrated circuit peripherals allow for the interaction among electronic devices for exchanging data, either the integrated circuit performs the default connection tasks or has to be implemented by software.

The wired communication protocols SPI e I2C are important for this work, so this paper summarizes their main features.

I2C (Inter-Interface Circuit). The I2C bus uses a bit in the device address to indicate read or write operations. The Master transmits the Slave's Address and a Read or Write bit to indicate the direction of the transfer. The I2C bus can be either a single-master or multi-master. Each electronic embedded device has a unique 7-bit or 10-bit address and it is limited to 8 bit's transfers. The I2C supports three basic modes of operation providing different levels of performance and device's address mapping: standard mode (up to 100 Kbits/sec, 7 bit addressing); fast mode (up to 400 Kbits/sec, addressing between 7 to 10 bits); high-speed mode (up to 3.4 Mbits/sec, addressing between 7 to 10 bits) [1] [2].

SPI (Serial Protocol Interface). The SPI bus is a 4-wire full-duplex interface synchronous serial data link [3]. Indeed, it is a (3+N)-wire interface where N is the number of devices connected to a single master device on the bus. Only one master can be active on the bus. Unlike I2C, SPI supports a transfer size of integer multiples of 8 bits. Technically the SPI bus shift register's length limits the size of the data transfers. The SPI bus can support a variety of transfer speeds but the bus is limited by the system´s clock. The SPI interface is generally is able data rates of several Mbits/sec.

This paper describes the procedure used to implement the synchronous serial communication protocols SPI and I2C by means of the hardware programming language Verilog HDL (Hardware Description Language).

The outline of this paper is divided in four sections. Section 2 discusses the research course; section 3 illustrates the methods used in the development of this work. Section 4 reports the obtained results and conclusions.

2 Justification

There are many software applications developed in the implementation of the communication protocols SPI and I2C [4] [5] [6] [7]. In general, these researches are focused on the comparisons and implementations of different architectures in order to meet characteristics required by current technologies.

In 2006, Oudjida et al., developed a code to implement to medium/low speed a transmitter slave I2C in a VLSI-architecture that allowed meeting some requirements that were not implemented in other systems such as drive noise filtering, a data unit, a unit equipment side interface, a control unit, among others [4].

More recently, in 2009 Oudjida et al., present an implementation of the SPI and I2C protocols in different FPGA devices, to help designers choose the right architecture for their system. To do this, they designed the code in Verilog (according to each protocol) for the slave SPI and I2C to the different FPGA devices, comparing their functionality in response times and clock settings, concluding that logic can predict certain behaviors for master devices from the results of the slaves [5].

Then in 2011 Lazaro et al., presented a new design in Verilog I2C protocol, focusing on the security of the electronic communications devices, integrating AES-GCM authentication and encryption algorithms [6]; To do this, they adapted the features of the I2C protocol with authentication techniques and encryption of data, comparing the final design with the original protocol, and they concluded that their work reduces the overhead of data flow and it is easily implemented in FPGA.

Zhou et al., developed a verification environment of complex electronic systems from the master SPI interface, and integrate Verilog with object-oriented programming (OOP). To achieve this, they started with the functional description of the requirements for the master SPI and environment design, and they implemented the APB controller in OOP classes [7]. The SPI Master interface was developed and implemented in FPGA Verilog.

Finally, it is important to note that in the above-mentioned works, the designers have used the method of hardware description (Verilog HDL), which helps to imple-

ment and to model the concurrent behavior of the electronic embedded devices, especially when it is a new architecture design [8].

3 Methods

For the realization of this article, it was taken into account certain key features of the standard communication protocols SPI and I2C [1] [2] [3] for implementing them in a FPGA, by using Verilog programming language. The methodology developed for each protocol is presented below.

3.1 SPI protocol

The following features were taken into account for implementing in a FPGA:

- The clock signal (CLK_3) that works at the speed of 3.6Mhz, generated by the master module (which may vary according to the criterion of designer).
- The data signal (SDA), which can be read and / or written by the master or the slave.
- The control signal (CS) enables in the low state and disables in the high state the slave communication.
- MOSI calls the master by sending data and the slave by receiving data.
- MISO calls the master by receiving data and the slave by sending data.
- The MOSI / MISO communication ends with the positive transition of CS.
- The master clock signal sets polarity (CPOL) and phase (CPHA) to "1".
- The master transmits data with positive edge, and receives with negative edge of CLK_3.
- The slave receives with the negative edge, and transmits with positive edge of CLK_3.

3.2 I2C protocol

The following features were taken into account for implementation in FPGA:

- The clock signal (SCL) is set at the frequency of 396Khz. (the frequency can be modified according to the standard: 100Khz/400Khz and 3.4Mhz).
- The data signal (SDA) is a bidirectional line, which can be read and written, by the master and the slave.
- The communication between a master and a slave begins with a START condition followed by the slave address to be reached, one read/write bit, a bit of recognition that can be ACK (if the communication was successful) or NACK (if the communication was unsuccessful or the end of the message is set by the master), the 8 bits of data to send or to receive, the ACK or NACK bit, and finishes with a STOP condition or a condition Repeated START.

- If there is no communication between a master and a slave, the data and clock signals remain in the high impedance state.
- The protocol neither implements the designed multi-master function, nor the extension function clock, nor the sending and receiving of more than one datum.
- Each master and slave manages two operating modes: receive and send data.
- The slave address is 7-bits length.
- If the master receives a non-recognized signal by the slave, a STOP condition is generated.
- When the master or the slave does not acknowledge the received data, both the clock and the data signals change to the high impedance state and thus releasing the bus.

Below shows part of the code developed for the I2C master and slave. The master code is showing a read operation of 8-bit data, the generation of the SDA and SCL (bidirectional) signals using a tri-state buffer.
The slave code shows an example of the writing operation ACK bit in the SCL line using a tri-state buffer. To synchronize data from the slave, reading has handled SDA data on the rising edge of SCL, and for writing the data on SDA, has handled the falling edge of SCL.

```
//**********************************//
//MASTER I2C CODE VERILOG
//**********************************//

`timescale 1ns / 1ps

module maestro(SDA,SCL,clk_50,Dir_esclavo,
Data_out,RW,ACK,NACK,contador_32,clk_3,scl_out_1,scl_out);

inout SDA;         // bidirectional SDA line
inout SCL;         // bidirectional SCL line
parameter sda_in = 0;
parameter sda_in_1 = 0;
input clk_50;      //frequency 50Mhz FPGA
reg [0:6]Dir_modulo = 7'b0011011;
reg [0:7]Data_in = 8'b01011010;
input RW;          //read/write line
reg [0:7]prueba = 8'b11011110;
output reg [0:7]Data_out = 0;
output reg ACK = 0;
output reg NACK = 0;
reg sda_out = 0;
reg sda_z = 0;
output reg scl_out_1 = 0;
output reg scl_out = 0;
reg scl_z = 0;
reg control_scl = 0;
reg control_sda = 0;
output reg clk_3 = 0;
//frequency divider counter FPGA
reg [7:0]contador_div = 0;
//read/write data counter
```

```
output reg [4:0]contador_32 = 0;
reg [0:7]Dato_Esperado = 8'b00000000;
reg [0:6]direccion_modulo = 0;   //control flags
reg ACK_MTx= 0;
reg ACK_MRx= 0;
reg NACK_MTx= 0;
reg NACK_MRx= 0;
reg bd_ACK = 0;
reg bd_NACK = 0;
reg bd_ACK_MRx = 0;
reg bd_NACK_MRx = 0;

                  //*********THIRD STATE LINE********//

assign SCL = control_scl ? 1'bz : scl_out;
assign SDA = control_sda ? 1'bz : sda_out;
//************************************//

always@(posedge clk_50) begin
//operating frequency 396KHz
contador_div = contador_div +1;
if(contador_div == 67) clk_3 =~ clk_3;
if(contador_div == 68) contador_div = 0;
end

always@(negedge clk_3) begin
contador_32 = contador_32 + 1;
end

always @(contador_32) begin
//START
if(contador_32<2 ) begin
control_sda = 1;
bd_ACK = 0; bd_NACK = 0;
bd_ACK_MRx = 0;
bd_NACK_MRx = 0;
direccion_modulo = Dir_modulo;
ACK = 0; NACK = 0;
ACK_MRx = 0;
ACK_MTx = 0;
end
if(contador_32 == 2) begin
control_sda = 0;
sda_out = 1'b0;
end
if(contador_32>2 && contador_32<10) begin
sda_out = Dir_esclavo[contador_32-3];
end
if(contador_32 == 10)
sda_out = RW;
if(ACK_MRx == 1 && contador_32 == 12 && RW == 0)
 begin
       sda_out = 1'b0; control_sda = 0;
 end
 if(ACK_MRx == 1 && contador_32 == 13 && RW == 0)
  bd_ACK_MRx = 1;
if(ACK_MRx == 1 && contador_32 > 13 && RW == 0)
   control_sda = 1;
```

```
if(contador_32 > 11 && contador_32 < 20 && ACK_MRx == 0 && RW == 0)
begin
Data_out[contador_32-12] = prueba[contador_32-12];
control_sda = 0;
sda_out = prueba[contador_32-12];
end
if(contador_32 == 20 && RW == 0 && ACK_MRx == 0 && Data_out ==
Dato_Esperado) begin
control_sda = 0;
NACK = 1;
NACK_MRx = 1;
sda_out = 0;
end
if(contador_32 == 20 && RW == 0 && ACK_MRx == 0 && Data_out !=
Dato_Esperado) begin
control_sda = 0;
NACK = 0;
NACK_MRx = 0;
sda_out = 0;
end
if(ACK_MRx == 0 && NACK_MRx == 1 && contador_32 == 21 && RW == 0)begin
sda_out = 1'b0;
control_sda = 0;
bd_NACK_MRx = 0;
end
if(ACK_MRx == 0 && NACK_MRx == 0 && contador_32 == 21 && RW == 0) begin
sda_out = 0'b0;
bd_NACK_MRx = 1;
end
if(contador_32 > 2 && contador_32 < 22 && (RW == 1 && bd_ACK == 0 &&
bd_NACK == 0))begin
control_scl = 0; end
else begin
control_scl = 1;
scl_out_1 = 0;
end
if(contador_32 >= 22)
control_sda = 1;
end

always @(clk_3) begin
       scl_out = (scl_out_1 & clk_3);
end;

endmodule

//**********************************//
//SLAVE I2C CODE VERILOG
//**********************************//

`timescale 1ns / 1ps

module esclavo (SDA,SCL,Direccion_Eslave,Direccion_Master,b_D);

inout SDA;   // bidirectional SDA line
input SCL;   // bidirectional SCL line
input[0:6]Direccion_Eslave;
reg [0:6]Direccion_Eslave_1 = 0;
reg control_sda = 1;
reg control_sda_p = 1;
```

```
reg control_sda_n = 0;
output reg [0:6]Direccion_Master = 0;
output reg b_D = 0;
reg [0:7]Data_in_Eslave = 0;
reg [0:6]prueba = 7'b0101011;
//synchronization counters
reg [4:0]contador_32=0;
reg [4:0]contador=0;
reg RW = 0;     //read/write register
//control flags
reg ACK = 0;
reg NACK = 0;
reg b_ACK_Rx = 0;
reg b_NACK = 0;
reg b_ACK_Rx_1 = 0;
reg b_NACK_1 = 0;

//*********THIRD STATE LINE********//
assign SDA = control_sda ? 1'bz : 1'b0;
//**********************************//

always @(posedge SCL) begin  //read data SDA
contador_32 = contador_32 + 1;
if(contador_32 == 1) begin
b_D = 0;
b_NACK = 0;
b_ACK_Rx = 0;
control_sda_p = 1;
end
  if(contador_32 >= 1 && contador_32 < 8 ) begin
Direccion_Master[contador_32-1] = SDA;     end
 if(contador_32 == 8 ) begin
RW = SDA;
 if(contador_32 == 8 && Direccion_Master == Direccion_Eslave_1 ) begin
  b_D = 0;
  b_ACK_Rx = ACK;
 end
 if(contador_32 == 8 && Direccion_Master != Direccion_Eslave_1 ) begin
  b_D = 1;
  b_ACK_Rx =~ ACK;
 end
 if((contador_32 == 10 ) || (contador_32 == 19 && b_ACK_Rx == 0 &&
b_NACK == 1))begin
  contador_32 = 0;
  control_sda_p = 1;
 end
 if(contador_32 == 0 && SCL == 0) Direccion_Eslave_1 <=
Direccion_Eslave;
end

always @(*) begin
 control_sda = ~(control_sda_n^control_sda_p);
end

always @(negedge SCL) begin //write data SDA
contador = contador + 1;
 if(contador == 9 && SCL == 1 )
    control_sda_n = 0;
 if(contador == 10) contador = 0;
 if(contador != 9) control_sda_n = 1;
```

```
end

endmodule
```

The communication protocols master/slave SPI and I2C were implemented in two Spartan 3-E FPGA's of 500 and 1200 system gates of the Xilinx [9]. This work was to implemented with the software version 14.7 of Xilinx ISE Design Suite.

4 Results

The following figures were obtained from the Verilog code and the physical implementation into a FPGA.

Fig. 1 shows the CLK_3, SDA and CS signals generated by the master using the SPI module. The data writing process occurs when the CS signal from slave is in the low state.

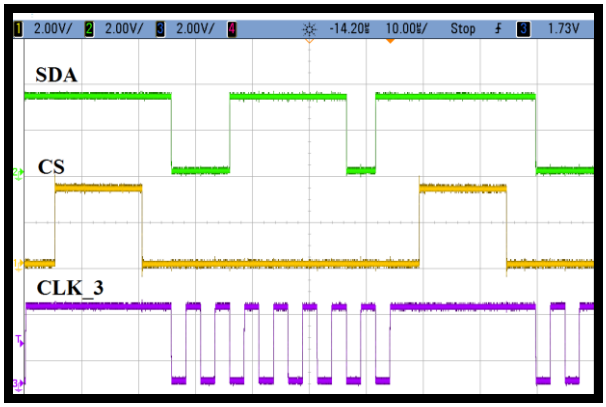

Fig. 1. SPI master write

In Fig. 2, the SDA and SCL signals from the I2C protocol are shown. Here we observe the non-recognition of the address (ACK) from the slave (counter 9). The completion of the communication happens when a STOP condition is performed by the master (counter 10).

The case in which the master writes data to the slave is shown in Fig. 3. The communication ends when the master receives the NACK bit written by the slave SDA, which generates the STOP condition.

In Fig. 4 we observe a similar case as that shown in Fig. 3 but with the difference that the slave does not acknowledge the data sent by the master. The communication ends with the setting the SDA and SCL signals to the high impedance state by the master after the bit NACK.

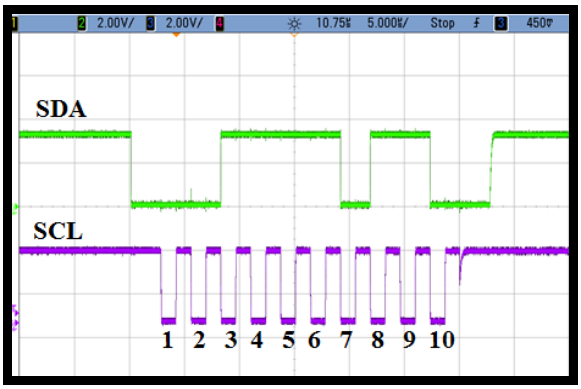

Fig. 2. The I2C signal from slave does not respond to the address sent by the master

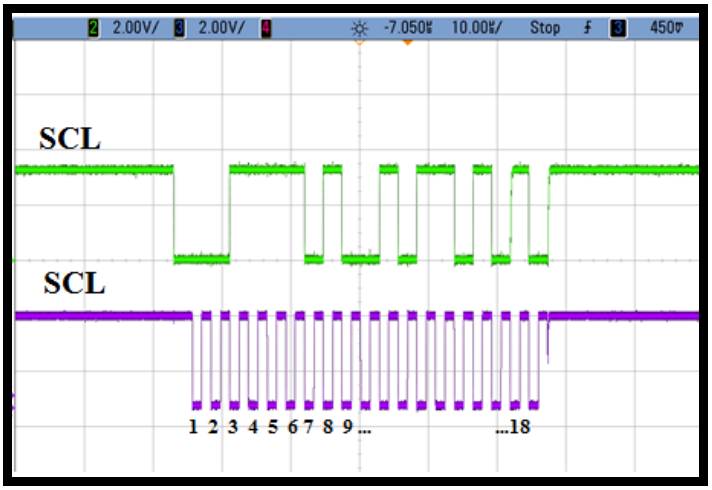

Fig. 3. Correct I2C communication between a master and a slave

5 Conclusions

In the development and implementation of the SPI and I2C protocols into a FPGA, the following conclusions were obtained:

- Verilog is a high level programming language that runs concurrently, to the difference with other programming languages that work sequentially as those used by microcontrollers, so it performs a faster and more efficient communication when implementing the SPI and I2C communication protocols.
- The developed code for the implementation of the I2C protocol is more robust and complex compared to the developed for the SPI protocol, due to the

number of events to take under consideration in communicating devices such as the start and the stop conditions, bit recognition, and read/write data on bidirectional lines.

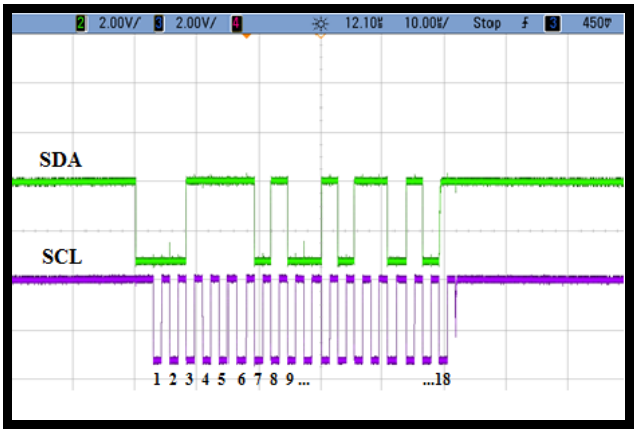

Fig. 4. The slave does not recognize the data sent by the master

- It is needed to set at least two operating conditions for input and output data in, each device acting as master or slave for a correct reading and / or writing process between them, because it is not possible to ensure an ideal behavior in the rising and falling edges of the signals, which are necessary for the execution of the code.
- A tri-stated buffer is implemented in the I2C slave for reading and writing data to the bidirectional SDA line, besides the high impedance statehood protects the FPGA peripherals against the phase shift operation signals.
- In the I2C slave, it is necessary to consider the lag introduced in the SDA and SCL signals at the time of writing the data, because the reading process occurs in the rising edge and the writing process occurs in the falling edge of the clock signal, so that these match the structure of the data line SDA.

References

1. J. Irazabel y S. Blozis, Philips Semiconductors, "I2C-Manual," Application Note, ref, *AN10216-0, March,* vol. 24, 2003. F. S. Motorola y S. B. Guide, V03. 06, February 2003, *Freescale Semiconductor Inc.*
2. I2C-bus specification and user manual, Rev, vol. 3, p. 19, National Semiconductors, 2007.
3. Freescale Motorola Semiconductors Guide, V03. 06, February 2003, Freescale Semiconductor Inc.
4. A. K. Oudjida, A. Liacha, D. Benamrouche, M. Goudjil, R. Tiar y A. 5. Ouchabane, Universal low/medium speed I^2C-slave transceiver: a detailed VLSI implementation, International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006, 2006.

5. A. K. Oudjida, M. L. Berrandjia, R. Tiar, A. Liacha y K. Tahraoui, FPGA implementation of I2C & SPI protocols: A comparative study, Electronics, Circuits, and Systems, 2009. ICECS 2009. 16th IEEE International Conference on, 2009.
6. J. Lazaro, A. Astarloa, A. Zuloaga, U. Bidarte y J. Jimenez, I2CSec: A secure serial Chip-to-Chip communication protocol, Journal of Systems Architecture, vol. 57, n. 2, pp. 206-213, 2011.
7. Z. Zhou, Z. Xie, X. Wang y T. Wang, Development of verification envioronment for SPI master interface using SystemVerilog, 2012 IEEE 11th International Conference on Signal Processing (ICSP), 2012.
8. R. Dubey, Introduction to embedded system design using field programmable gate arrays, Springer, 2009.
9. Xilinx Inc., Spartan-3E FPGA Family Data Sheet, DS312 Product specification, 2013, http://www.xilinx.com/support/documentation/data_sheets/ds312.pdf