Implementation of an FPGA - Raspberry Pi SPI Connection

Haissam Hajjar
Department of Applied Business Computer,
Faculty of Technology, Lebanese University
Saîda, Lebanon
haissamh@ul.edu.lb

Hussein Mourad
Department of Applied Business Computer,
Faculty of Technology, Lebanese University
Saîda, Lebanon
mourad_hussein@hotmail.com

***Abstract*— The use of Field Programmable Gate Arrays (FPGAs) requires low level programming. This makes it difficult to have a friendly user interface. The presented work explains FPGA techniques in detail. There are few works demonstrating an application integrating FPGA and ergonomic user-interface techniques. This article describes the connection of an FPGA to a Raspberry PI using a Serial Peripheral Interface (SPI) link. A Python SPI driver is developed on the Raspberry side. A Very High-Speed Integrated Circuit Hardware Description Language (VHDL) driver is developed on the FPGA side. A Web client-server application is developed to demonstrate the usage of SPI link and its integration with a standard Web application to control the FPGA inputs and outputs.**

Keywords*—SPI VHDL driver; VHDL; Raspberry PI; Altera Cyclone II; Python VHDL communication; Python-PHP socket communication.

I. INTRODUCTION

FPGAs are typically used in electronic circuits. Usually, they are programed in VHDL or Verilog [1]. This is well suited to stay at the hardware level but remains very poor and complex when developing a user-friendly human-machine interface.

The VHDL implementation of SPI protocol is developed in some previous works [2][3]. However, these works focus their efforts on the electronic aspect by neglecting the application aspect.

The objective of this paper is to connect an FPGA to a Raspberry PI so that one side can use the FPGA for the electronic part, while the Raspberry PI can be used to develop a friendly user interface using common well-known techniques. The utilization of a Raspberry PI is taken to demonstrate a low-cost solution for this implementation.

As the Raspberry runs under Linux operating system and the FPGA is programmed at an electronic level, we elected to use the SPI standard that does not need to use a common clock (see Figure 1). The communication is synchronized by a clock signal delivered by the SPI Master, independently of the internal clock frequency of each side.

On the programming language level, we choose to use Python for the Linux side (Raspberry) and VHDL for the FPGA side. So, the SPI driver can be integrated with the commonly used frameworks on the Linux side.

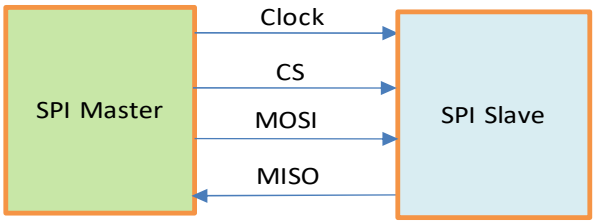

Figure 1. SPI Single Master – Single Slave signals - Chip Select (CS), Master Out Slave In (MOSI), Master In Slave Out (MISO)

This paper covers the following topics: Section I has provided an introduction. In Section II, we give a functional description of the implemented system. In Section III, we describe the hardware implementation and the materials used. In Section IV, we develop the SPI implementation on the Master level and the Slave level. In Section V, we present testing results of the SPI. In Section VI, we describe a high-level user interface developed to illustrate the SPI utilization. Finally, a conclusion is included in Section VII.

II. FUNCTIONAL DESCRIPTION

Figure 2 illustrates a functional representation of the whole system:

A. Raspberry PI

A Raspberry PI 3 [4] functions as Master of the SPI link. The Raspberry PI is equipped with a General Purpose Input/Output (GPIO). The SPI was implemented using 4 lines of this GPIO. A Python implementation of SPI Master is utilized. An Apache Web server is implemented inside the Raspberry to allow implementation of ergonomic and easy to use interface for testing and demonstration purposes. A SPI Master driver is developed using Python language and libraries. As the system must work efficiently with regards to real time response time, two independent processes were implemented within the Raspberry PI system:

- To handle the user requests: an Apache Web server is implemented using PHP scripting for the Web server side.
- To handle the SPI link during the communication with the FPGA. This driver is written using Python.

 ISBN: 978-1-61208-748-1

- The communication between these two processes is executed using TCP/IP socket communication.

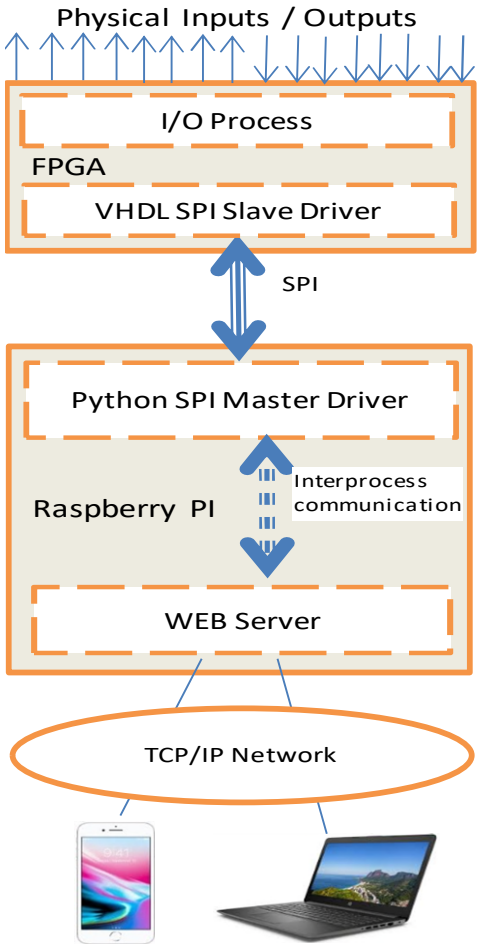

Figure 2. Functional representation

B. FPGA

An 'Altera DE2' Development and Education Board' [5] is used to implement the FPGA part. This board is built on a Cyclone II EP2C35F672C6 FPGA working up to 50 MHz clock frequency. This board has a 2 lines/16-character LCD display, a set of 18 toggle switches for digital inputs, a set of 4 pushbuttons, a set of 18 red led for digital outputs and 2 forty lines extension headers for external connection. The SPI is implemented using one of these extension headers.

To illustrate the successful operation of the SPI link, we devise three functional usages:

- Send order, starting from the user interface, to drive the 18 FPGA board digital outputs
- Receive the status of the 18 lines of digital input to display on the user screen.
- Send 32 bytes, entered on the user screen, in order to be displayed on the FPGA LCD 2 lines display.

Compared to OSI communication layer, we can consider the SPI drivers as the physical layer and these functional usages as a link layer. So, this can be extended to implement other functional types of messages exchanged between the FPGA and the Raspberry.

***TCP/IP network*:** The Raspberry PI has an 802.11 Wi-Fi 2.4 GHz interface. This interface is used to allow the connection of a Web-based client using a standard browser. A Web-based user interface is developed to allow the usage of the previously mentioned illustration function for the use of the SPI connection.

III. PHYSICAL IMPLEMENTATION

Figure 3 represents the physical implementation:

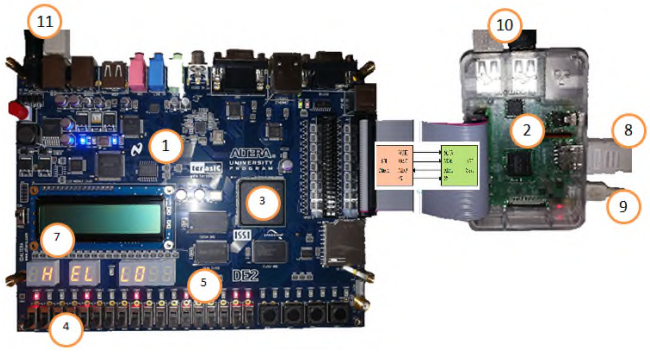

Figure 3. Physical implementation

1. DE2 Development and Education Board: we use this board for the FPGA implementation part. For detailed documentation, refer to the Intel official documentation [5]
2. Raspberry Pi 3 board [3]: this board is equipped with 1 GB Ram, processor Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz.
3. Cyclone® II 2C35 FPGA in a 672-pin package, working at 50 MHz
4. 18 switches used as digital inputs
5. 18 LEDs used as digital outputs
6. 40 pins flat cable used as connector between the Raspberry GPIO and the extension header of the DE2 board. This cable is used to implements the SPI connection between the Raspberry PI and the Altera DE2 FPGA evaluation board.
7. 2x16 LCD and eight 7 segment digital display
8. HDMI connector for Raspberry PI
9. Power supply for Raspberry PI
10. Mouse and keyboard USB connectors
11. Power Supply and Programmer connection

 ISBN: 978-1-61208-748-1

IV. SPI IMPLEMENTATION

The implemented system is represented in Figure 4. A single Master with single Slave scenario is shown in the following paragraphs.

A. Physical interface

Figure 4 presents the SPI signals. The Raspberry PI is the master and the FPGA is the slave. A configuration of one Master/one Slave is implemented:

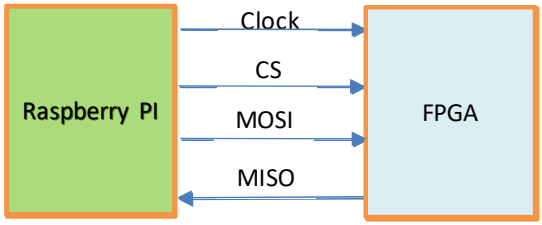

Figure 4. SPI implementation

- Clock: the clock is generated by the Master. This signal drives the communication in both directions.
- CS (Chip Select) high when the FPGA is not selected: No communication; low when selected.
- MOSI: Master Out Slave In: data transferred from the Master to the Slave.
- MISO: Master In Slave Out: data transmitted from the Slave to the Master.

B. Implementation principle

The communication is driven by the Master. The first byte determines the type of communication. To illustrate the usage of the drivers, three types of messages were implemented:

- Send Memory: The Master sends to the Slave 32 bytes of data for displaying on the LCD.
- Send Outputs: The Master sends to the Slave the order to set its digital outputs ON or OFF
- Receive Inputs: The Slave sends to the Master the status of its digital inputs.

C. Master Driver

This driver is based on the RPI.GPIO Python library [9]. After initialization, two functions are available for an upper level usage:

Sendbyte: send a byte from Master to Slave.

Receivebyte: receive a byte from Slave to Master.

To send a bit, MOSI is set, and then a Clock is generated (SCLK from Low to High).

To receive a bit, a Clock rising is generated, and then the MISO line level is read.

SPI Initialization (Figure 5): the GPIO of the Raspberry includes 2 SPI lines. We had trouble driving these lines with the standard Raspberry library. We opted to drive the SPI signals directly through our program. This allowed us to control the CS and Clock lines easily and to reach the maximum possible communication speed with a Python driver.

```
import RPi.GPIO as GPIO
# Line definition
  MOSI = 5
  MISO = 10
  SCLK = 15
  CE0 = 7
#
# SPI line initialization
def initspi():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)
    GPIO.setup(MOSI, GPIO.OUT)
    GPIO.setup(MISO, GPIO.IN)
    GPIO.setup(SCLK, GPIO.OUT)
    GPIO.setup(CE0, GPIO.OUT)
```

Figure 5. Master driver - SPI initialization

Send byte (Figure 6): The transmission of a byte starts with the change of the signal CS (CS low). This will initiate the reception process on the VHDL side. The MOSI level is set according to the bits to be sent and a clock signal is generated. After transmission of the 8 bits, this CS signal returns to the high level.

```
def sendbyte(cc):
    c=ord(cc)
    # select slave
    GPIO.output(CE0, GPIO.LOW)
    bitsx = [0,0,0,0,0,0,0,0]
    # determine bits 0/1
    for x in range(8):
        bitsx[7-x] = int(c % 2)
        c = int((c - bitsx[7-x])/2)
    # set Mosi signal level
    for x in range(8):
        if (bitsx[x]>0):
            GPIO.output(MOSI,
GPIO.HIGH)
        else:
            GPIO.output(MOSI, GPIO.LOW)
    # clock
        GPIO.output(SCLK, GPIO.LOW)
        GPIO.output(SCLK, GPIO.HIGH)
    # end of byte transmission
    GPIO.output(CE0, GPIO.HIGH)
    GPIO.output(SCLK, GPIO.LOW)
    GPIO.output(SCLK, GPIO.HIGH)
```

Figure 6. Master driver – Send byte

 ISBN: 978-1-61208-748-1

Receive byte (Figure 7): The reception of a byte starts with the change of the signal CS (CS low). This will initiate the transmission process on the VHDL side. The MISO level is read according to the bits received each clock signal generated. After reception of the 8 bits, this CS signal returns to the high level.

```
def receivebyte():
    GPIO.output(SCLK, GPIO.LOW)
    # select slave
    GPIO.output(CE0, GPIO.LOW)
    out = 0b0
    # read 8 bits on MISO
    for x in range(8):
        GPIO.output(SCLK, GPIO.LOW)
        GPIO.output(SCLK, GPIO.HIGH)
        out = out*2
        if GPIO.input(MISO):
            out = out + 1
    GPIO.output(CE0, GPIO.HIGH)
    return out
```

Figure 7. Master driver - Receivebyte

D. Slave Driver

Figure 8 presents the process handling the communication on the FPGA.

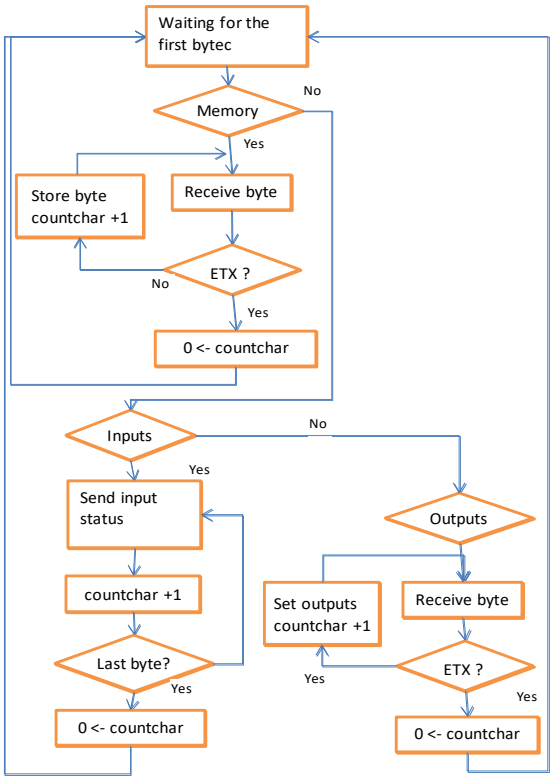

Figure 8. Master driver - Receivebyte

The process is normally in a waiting state. It is activated by the CS signal. When the CS is down, the FPGA reads the bits set on MOSI signal on rising edge of Clock signal.

```
process (SCK,CS,reset)
begin
   if (CS = '1' and octet=MasterToSlaveMemory) then
      countbit <= 0; countchar <= 0;
      direction <= RECEIVE_MEMORY;
   elsif (CS = '1' and octet=MasterToSlaveOutput) then
      countbit <= 0;       countchar <= 0;
      direction <= RECEIVE_OUTPUTS;
   elsif (CS = '1' and octet=SlaveToMaster) then
      countbit <= 0; countchar <= 0;
      direction <= SEND_INPUTS;
  elsif …..
```

Figure 9. Slave driver - Message type detection

When the first byte is received, it is tested. Three cases are considered:

- 'Send Memory': the FPGA continues the reception of the following bytes. The received bytes are stored in an internal memory indexed by a reception counter. This will end when an EOT is received.

```
-- Receive memory
if (rising_edge(SCK) and CS='0'
    and direction=RECEIVE_MEMORY) then
      octet <= octet(size-2 downto 0) & mosi;
      countbit <= countbit+1;
      if (countbit=7) then
        memory(countchar) <= octet(size-2 downto 0) &
mosi;
        if memory(countchar) = EOT then
           countchar <= 0;
        else
          countchar <= countchar+1;
        end if;
        countbit <= 0;
   end if;
end if;
```

Figure 10. Slave driver - Receive memory

- 'Send outputs': the FPGA continues the reception of data. The outputs are set/unset according to the received data.

```
-- Receive Outputs
if (rising_edge(SCK) and CS='0' and
    direction=RECEIVE_OUTPUTS) then
-- Receive 3 bytes [18 bits only valid] for digital outputs
      outputs(countchar)(7-countbit) <= mosi;
      countbit <= countbit+1;
      if (countbit=7) then
        if memory(countchar) = EOT then
           countchar <= 0;
        else
          countchar <= countchar+1;
        end if;
        countbit <= 0;
      end if; end if;
```

Figure 11. Slave driver - Receive outputs

- ‘Receive inputs’: the FPGA sends the status of its digital inputs [3 bytes for 18 inputs] using the MISO line. An EOT is sent to inform the Master that the end of sending is reached.

```
if (rising_edge(SCK) and CS='0'
        and direction=RECEIVE_OUTPUTS) then
 -- Receive 3 bytes [18 bits only valid] for digital outputs
   outputs(countchar)(7-countbit) <= mosi;
   countbit <= countbit+1;
   if (countbit=7) then
        if memory(countchar) = EOT then
            countchar <= 0;
        else
           countchar <= countchar+1;
        end if;
     countchar <= countchar+1;
      countbit <= 0;
   end if;
end if;
```

Figure 12. Slave driver - Request to send inputs

V. TESTING AND RESULTS

We present three tests executed to validate communication using this implementation of SPI.

A. Setting outputs

Send order from the Master (Raspberry PI) to the Slave (Altera FPGA DE2 board) to set/unset its digital outputs: ‘Oxxx’: Message of 4 bytes. The first byte represents the type of message; the following bytes represent the required outputs status.

Figure 13 shows the signals observed on the SPI lines. The message sent from the Master: the first byte represents the ASCII representation of the character O (01101111) used as identifier for this message. As described in Section IV, the following 3 bytes represent the value to be set on the digital outputs. The following 3 bytes represent the requested status of FPGA 18 lines output.

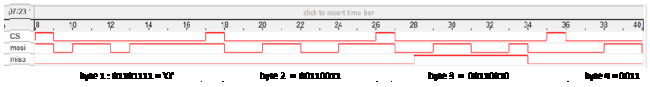

Figure 13. Signals on the SPI lines for sending outputs

Each byte starts when the CS comes down and is sent when the CS goes up again. Figure 14 shows the LEDs corresponding to the signal shown in Figure 13. The LED is on when ‘1’ is received and is off when ‘0’ is received.

Figure 14: Led status on the FPGA board

B. Read inputs

Send order from the Master (Raspberry PI) to the Slave (Altera FPGA DE2 board) ‘r’: This message asks the FPGA to send back to the Raspberry the status of its digital inputs. The Raspberry (Master) must continue to generate the clock. The next bytes are sent by the FPGA (Slave) to the master over the MISO line. As we have 18 inputs, three bytes are used for this function.

Figure 15 shows the SPI signals: the clock is always given by the Master. The Master sends the first ‘r’ byte (01110100) over the MOSI signal. Then, the Slave sends back three bytes.

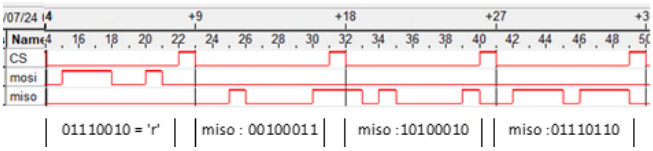

Figure 15: Signals on the SPI lines for read inputs outputs

Figure 16 shows the input switches generating the signals shown in Figure 15.

Figure 16. Input switch corresponding to Schema 6 signals

C. Performance

The performance of this link depends on the SPI Master. For the Raspberry III utilized, the speed of 100kb/s was reached.

VI. APPLICATION TESTING

An application is developed to show a concreate utilization of this work. Figure 17 shows a functional representation of this realization.

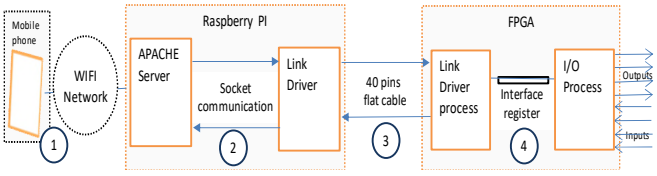

Figure 17. Signals on the SPI lines for sending outputs

The implementation inside the Raspberry PI is performed using two processes: an Apache server and the SPI link driver. This is done for real time constraints. The communication between these two processes is executed using client/server socket communication. The Apache side is developed in PHP and the SPI link driver side is developed in Python. Figure 18 presents the principle of this communication.

 ISBN: 978-1-61208-748-1

Figure 19 shows the user-interface on a smartphone screen using a standard Web browser.

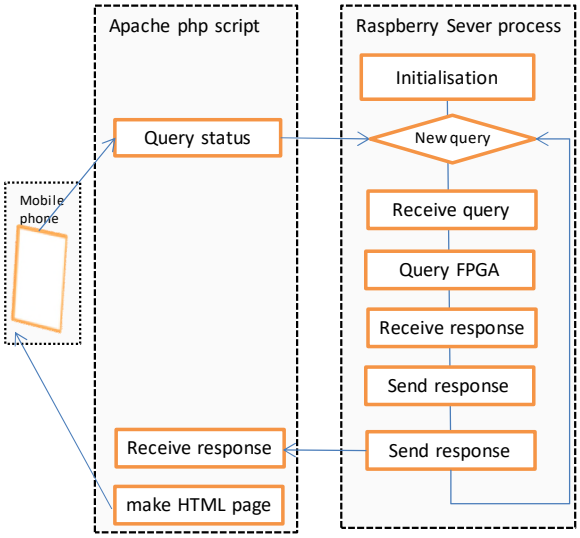

Figure 18. Client/server communication in Raspberry PI

The number of inputs and outputs are reduced to 8 to have an ergonomic user-interface on smartphones. The status of the digital inputs of the FPGA is reported on the user screen. The digital outputs of the FPGA are set according to the radio-button.

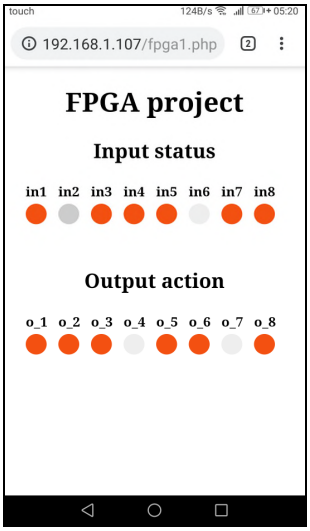

Figure 19. User interface print screen

VII. CONCLUSION

In this work, FPGA-Raspberry Pi communication is developed using the SPI protocol. A high level application is developed using this link. This demonstrates a solution that works by using a low level technique (VHDL) on the FPGA side and using a high level technique on the user interface side.

We have limited the application to digital inputs / outputs. The work can be extended to other functions of the FPGA. This opens the possibility of modifying the behavior of an FPGA dynamically. The job can also be completed in the sense of increasing the transmission speed, which is somehow proportional to the Master's clock frequency.

REFERENCES

[1] https://circuitdigest.com/tutorial/what-is-fpga-introduction-and-programming-tools (9/2019)

[2] N.Q.B.M. Noor and A. Saparon, "FPGA implementation of high speed serial peripheral interface for motion controller," in Proc. 2012 IEEE Symposium on Industrial Electronics and Applications (ISIEA), pp.78-83, Sept. 2012.

[3] Raspberi Pi official site: https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/ (1/2019)

[4] Rapberry Pi 3 board - Official documentation https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ (1/2019)

[5] Altera DE2-115 Development and Education Board - https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html

[6] Python documentation, https://www.python.org/ (1/2019)

[7] https://www.php.net/manual/fr/ (1/2019)

[8] Quartus II Handbook: http://www.altera.com/literature/hb/qts/quartusii_handbook.pdf (4/2019)

[9] GPIO Raspberry installation and usage : https://www.raspberrypi-spy.co.uk/2012/05/install-rpi-gpio-python-library/ (6/2019)

[10] SPI Tutorial – COREIS https://www.corelis.com/education/tutorials/spi-tutorial/ (5/2019)

 ISBN: 978-1-61208-748-1