Implementation of Parallel to Serial Interfacing using Synchronous First-In First-Out Buffer

Puya Fard
Computer Engineering
California State University, Fresno
Fresno,CA

Venkateshwar Reddy Enukonda
Computer Engineering
California State University, Fresno
Fresno,CA

Joseph Salas
Computer Engineering
California State University, Fresno
Fresno,CA

Abstract—Data transmission is the main purpose of communication networks. Systems transferring huge data require high speed data transmission. Parallel to Serial Interface is a subsystem involved in such high speed operating systems. Open Systems Interconnection (OSI) model is a standard that describes communication function of a communication networking system. There are 7 layers in the model, and each layer has to perform a unique functionality. OSI model published by International Organization of Standardization (ISO) in 1984 to bring a common standard in the field of computing and communication systems. The hardware layers are responsible for data transmission and reception at transmitter and receiver respectively. Using a serial connection, we can minimize the number of connection wires, minimizing also the skew problem on the connection itself. So, Parallel to Serial Interface and Serial to Parallel Interface are used in tandem to achieve this in communication networks. For this reason, the transmitter is expected to transmit the data serially independent of the internal data modes. So, the parallel to serial interfacing is important at the transmitter end, so is the serial to parallel interfacing at receiver end. The three layers in the hardware layers are collectively doing this operation. The parallel port to serial port with Synchronous First-In First-Out (FIFO) buffer can do the tasks of Network, Data-Link and Physical layer in the hardware part of the communication network. The Asynchronous transmission is difficult to achieve and requires two different clock frequencies to read and write data. Hence, the current discussion is limited to using Synchronous FIFO buffer for transmitting data.

Keywords— Asynchronous, Buffer, Data, FIFO, Synchronous, Transmission.

I. INTRODUCTION

Data has become an important part of the current century. Be it biomedical applications, social media networking, BigData utilized in forecasting or suggestions in the field of business development, large training data for Artificial Intelligence and Machine Learning techniques, etc. It is obvious that handling that big of a data is very important for data engineers and data scientists. Equally, hardware engineers also need to realize the models for transmitting high speed networking systems for transmission of such huge data. Parallel to Serial Interface is a subsystem involved in such high speed operating systems. The PSI is realized by using three sub modules, namely, parallel port, serial port, and FIFO buffer operated in synchronization with common clock. The communication protocol is directed by the OSI model. Hence, our discussion is divided as follows. The current one, Section-I is about the introduction and need for high speed networking systems. Section-II deals with the Open Systems Interconnection model. In Section-III, we have a brief discussion on the implementation of PSI with synchronous FIFO. Section-IV is about the synthesis portion. We conclude our discussion with conclusion in Section-V. Different ideas was presented during the researching phase among our group, our team leader, Venkateshwar came up with this idea to transmit data using Parallel to Serial Interfacing with Synchronous FIFO Buffer.

II. OPEN SYSTEMS INTERCONNECTION MODEL

Before we came up with this Design idea, we met up as a group and discussed what we want to implement and work as a semester project. All of us came up with different ideas, however, Venkateshwar introduced Parallel to Series implementation with synchronous FIFO buffer idea and we all agreed on it. All of us individually started researching about how it works, and what it does so we can understand how it works so we can implement it in verilog. After a week of research, we had a solid understanding of what needs to be done and we did it. Data transmission is vital for communication networks. The parallel to serial interface is a subsystem of network devices that are involved in high speed data transfer. In the Open Systems Interconnection (OSI) model,this device is in the fragmentation layer, in which networak packets which have headers, data, and trailers all of various amounts of bits, have to be transferred to other devices over a network connection. Multiple bits of information can be sent across a single data stream through a parallel to serial interface.

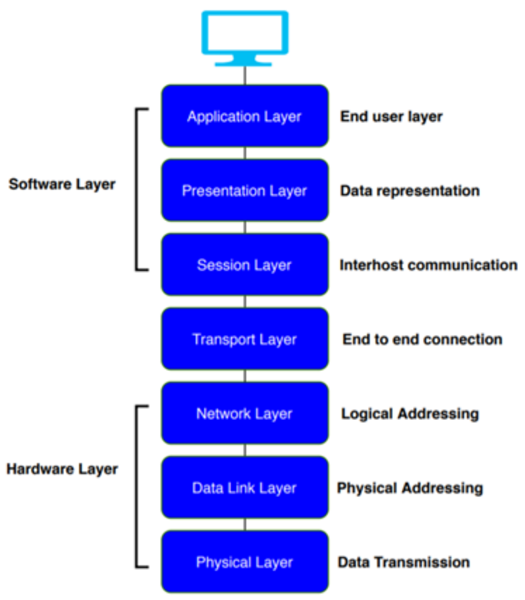


Fig. 1. Seven Layers of OSI Model

We are interested in only the three hardware layers of the OSI model, namely, Network Layer, Datalink Layer, and Physical Layer. The data transmission along with the attached header and trailer parts is shown in Fig. 2 [1]. The header is supposed to contain the addresses of sender and receiver. The trailer part indicates the end of the frame/packet.

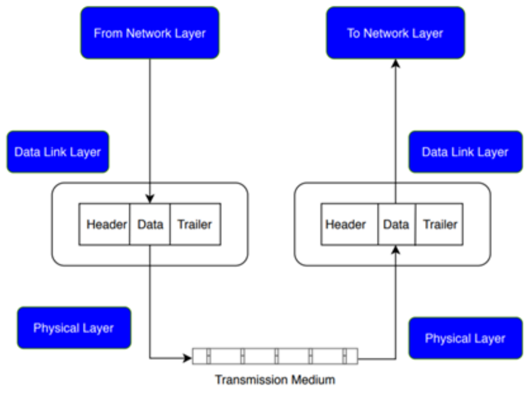
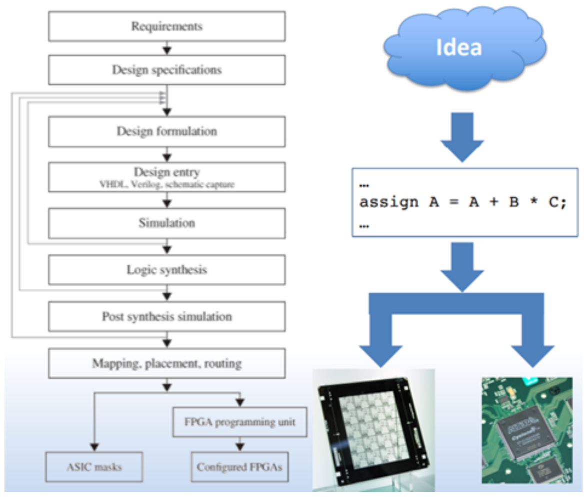


Fig. 2. Hardware Layers of OSI Model

The physical layer is the bottom most layer in the OSI model and it is responsible for the transmission and reception of data between the networking devices and the transmission medium. The transmission rate is controlled by the physical layer. The data transmission between two directly connected points is done by the data link layer. The physical layer converts the received data into bits and sends it to the data link layer. The logical communication between different network devices is provided by the network layer Addressing, an important part of the process is also performed by this layer. It adds the destination and source address with the header to the data arrived at this layer

III. PSI WITH FIFO BUFFER

The design of any modern digital system involves a step-by-step procedure called Full ASIC flow shown in Fig. 3 [2]. Initially, with the requirements of the project, the design specifications are determined. The design is formulated using state diagrams shown in Fig. 4 (a) and Fig. 4 (b) and block diagram shown in Fig. 5.



```
...
assign A = A + B * C;
...
```

Fig. 3. Full ASIC Flow

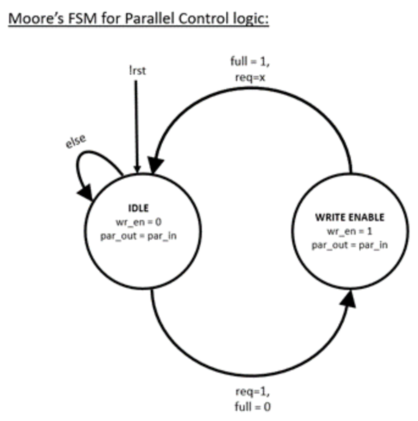


Fig. 4 (a). FSM for Parallel Control Logic

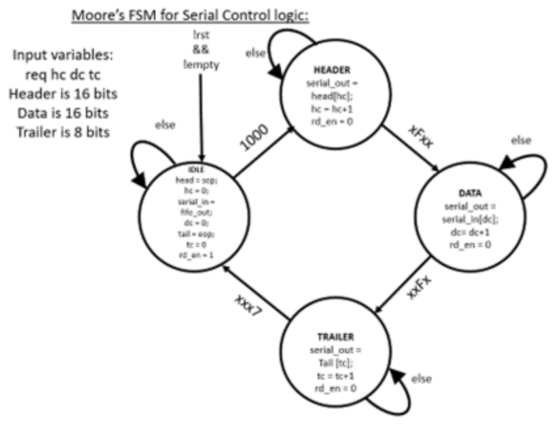


Fig. 4 (b). FSM for Serial Control Logic

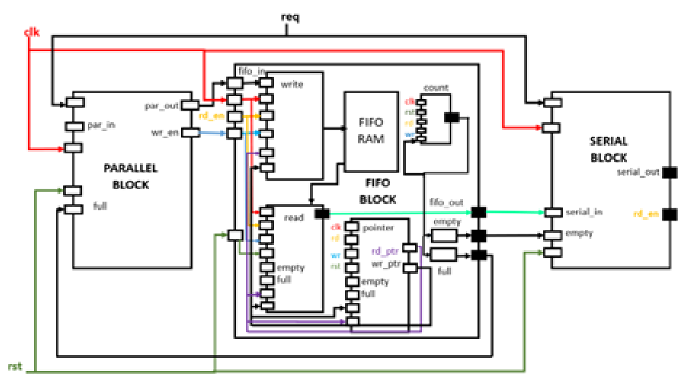


Fig. 5. Schematic of PSI using Synchronous FIFO Buffer

Hardware Description Language (HDL) coding such as VHDL, Verilog etc. is used in Top-Down approach of digital design. This coding is commonly known as Register Transfer Level (RTL) coding in digital design and involves two parts as shown in Fig. 6 [2]. We used Verilog as HDL and simulated in ModelSim of Intel FPGA starter edition provided by the Fresno State Virtual Labs.

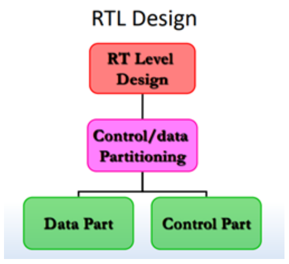


Fig. 6. Parts involved in RTL Design

The three modules are coded separately and combined by the top module. The parallel block takes a parallel input and outputs the parallel output and the write enable to enable the FIFO write port if not full. Clock, reset and request are common to both parallel and serial blocks. Empty and Serial input are inputs to serial block. The serial output and the read enable to enable the FIFO read port if not empty. The FIFO block contains FIFO input which is output of parallel port. Clock and reset are common that of parallel and serial blocks. Read and Write ports are enabled by read enable and write enable signals of serial and parallel ports respectively. The outputs of the FIFO are FIFO output, counter, empty and full flags. The FIFO block [3] contains pointer block, read block, write block, counter block, full, empty registers and a FIFO RAM.

A First-In First-Out (FIFO) is a queue that will output the oldest data in the structure first. As opposed to a stack, which is a Last-In First-Out (LIFO), and outputs the last data in first. This is a very common memory structure in digital systems, often used to buffer values between two different modules. The pointer block provides read and write pointers according to read and write operations into queue. The read block uses the read pointer to read from the queue memory. The write block uses the write pointer to write

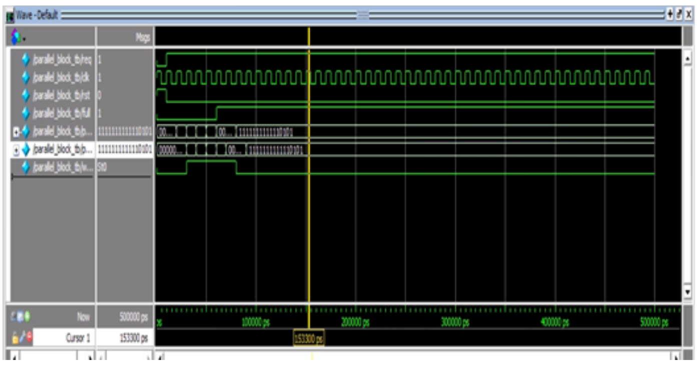


Fig. 7 (a). Waveform of Parallel port module

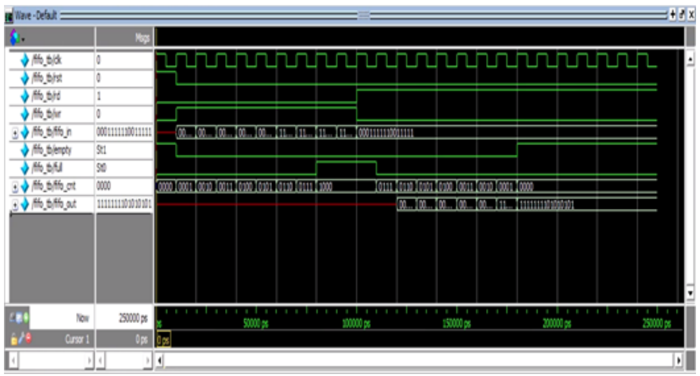


Fig. 7 (b). Waveform of FIFO block module

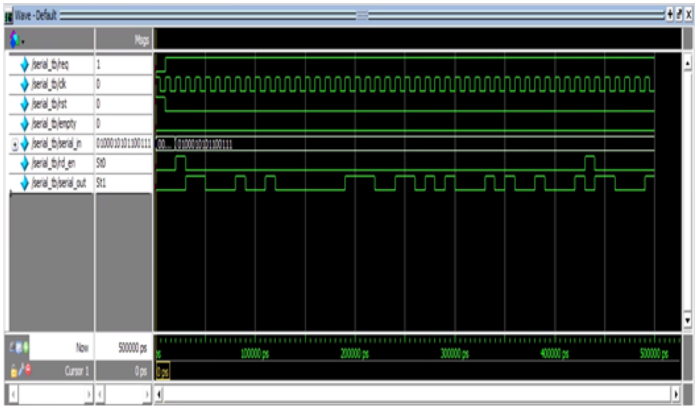
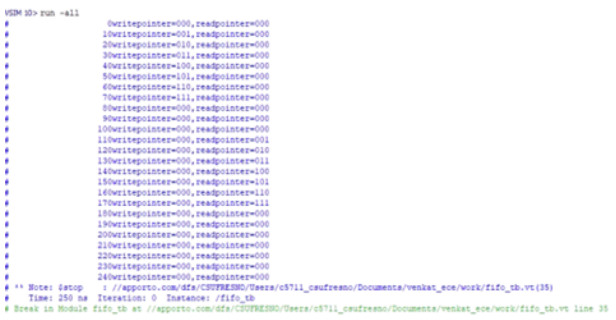


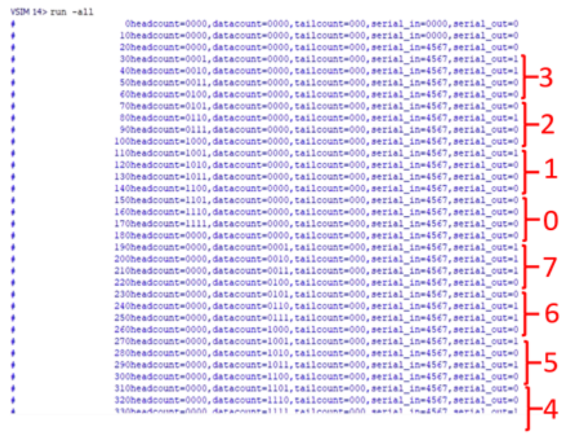
Fig. 7 (c). Waveform of Serial port module

into the queue memory. The counter block keeps track of the number of data in the queue and issues empty and full flags. FIFO RAM is a queue of 16-bit with depth of 8 in our case. The simulation result of the modules is shown in Fig. 7 (a), (b), (c). and the monitored pointers of FIFO block and serial output of serial port module is shown in Fig. 8 (a), (b) respectively.



```
VSIM 10> run -all
#                    0writepointer=000,readpointer=000
#                   10writepointer=001,readpointer=000
#                   20writepointer=010,readpointer=000
#                   30writepointer=011,readpointer=000
#                   40writepointer=100,readpointer=000
#                   50writepointer=101,readpointer=000
#                   60writepointer=110,readpointer=000
#                   70writepointer=111,readpointer=000
#                   80writepointer=000,readpointer=000
#                   90writepointer=000,readpointer=000
#                  100writepointer=000,readpointer=000
#                  110writepointer=000,readpointer=001
#                  120writepointer=000,readpointer=010
#                  130writepointer=000,readpointer=011
#                  140writepointer=000,readpointer=100
#                  150writepointer=000,readpointer=101
#                  160writepointer=000,readpointer=110
#                  170writepointer=000,readpointer=111
#                  180writepointer=000,readpointer=000
#                  190writepointer=000,readpointer=000
#                  200writepointer=000,readpointer=000
#                  210writepointer=000,readpointer=000
#                  220writepointer=000,readpointer=000
#                  230writepointer=000,readpointer=000
#                  240writepointer=000,readpointer=000
# ** Note: $stop    : //apporto.com/dfs/CSUFRESNO/Users/c5711_csufresno/Documents/venkat_ece/work/fifo_tb.vt(35)
#    Time: 250 ns  Iteration: 0  Instance: /fifo_tb
# Break in Module fifo_tb at //apporto.com/dfs/CSUFRESNO/Users/c5711_csufresno/Documents/venkat_ece/work/fifo_tb.vt line 35
```

Fig. 8 (a). Monitored pointers of FIFO block



```
VSIM 14> run -all
#                    0headcount=0000,datacount=0000,tailcount=000,serial_in=0000,serial_out=0
#                   10headcount=0000,datacount=0000,tailcount=000,serial_in=0000,serial_out=0
#                   20headcount=0000,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                   30headcount=0001,datacount=0000,tailcount=000,serial_in=4567,serial_out=1
#                   40headcount=0010,datacount=0000,tailcount=000,serial_in=4567,serial_out=1
#                   50headcount=0011,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                   60headcount=0100,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                   70headcount=0101,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                   80headcount=0110,datacount=0000,tailcount=000,serial_in=4567,serial_out=1
#                   90headcount=0111,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  100headcount=1000,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  110headcount=1001,datacount=0000,tailcount=000,serial_in=4567,serial_out=1
#                  120headcount=1010,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  130headcount=1011,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  140headcount=1100,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  150headcount=1101,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  160headcount=1110,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  170headcount=1111,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  180headcount=0000,datacount=0000,tailcount=000,serial_in=4567,serial_out=0
#                  190headcount=0000,datacount=0001,tailcount=000,serial_in=4567,serial_out=1
#                  200headcount=0000,datacount=0010,tailcount=000,serial_in=4567,serial_out=1
#                  210headcount=0000,datacount=0011,tailcount=000,serial_in=4567,serial_out=1
#                  220headcount=0000,datacount=0100,tailcount=000,serial_in=4567,serial_out=0
#                  230headcount=0000,datacount=0101,tailcount=000,serial_in=4567,serial_out=0
#                  240headcount=0000,datacount=0110,tailcount=000,serial_in=4567,serial_out=1
#                  250headcount=0000,datacount=0111,tailcount=000,serial_in=4567,serial_out=1
#                  260headcount=0000,datacount=1000,tailcount=000,serial_in=4567,serial_out=0
#                  270headcount=0000,datacount=1001,tailcount=000,serial_in=4567,serial_out=1
#                  280headcount=0000,datacount=1010,tailcount=000,serial_in=4567,serial_out=0
#                  290headcount=0000,datacount=1011,tailcount=000,serial_in=4567,serial_out=1
#                  300headcount=0000,datacount=1100,tailcount=000,serial_in=4567,serial_out=0
#                  310headcount=0000,datacount=1101,tailcount=000,serial_in=4567,serial_out=0
#                  320headcount=0000,datacount=1110,tailcount=000,serial_in=4567,serial_out=0
#                  330headcount=0000,datacount=1111,tailcount=000,serial_in=4567,serial_out=1
```

Fig. 8 (b). Monitored serial output of Serial Port module

The pointers and serial output are updated as expected and is therefore working as designed. Hence, we can move onto next stage of ASIC flow, Synthesis.

IV. SYNTHESIS

Synthesis is the extraction of gate-level netlist and other reports related to area, power, timing etc. from the VHDL/Verilog code. Design Compiler (DC) by Synopsys is one of the popular synthesis tools in the industry. We used the same tool provided by Fresno State Virtual Labs. It has a standard cell library, NangateOpenCellLibrary in our case. It has a compiler in addition to standard cell which is linked through .synopsys_dc.setup file. We get output gate netlist and a few reports to analyze. The schematic in Fig. 9. [4] shows the procedure of synthesis process.

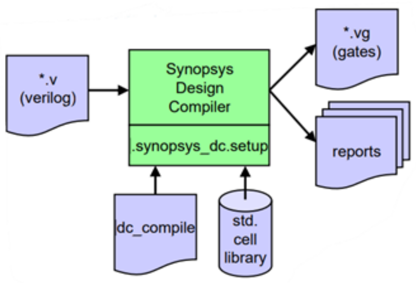
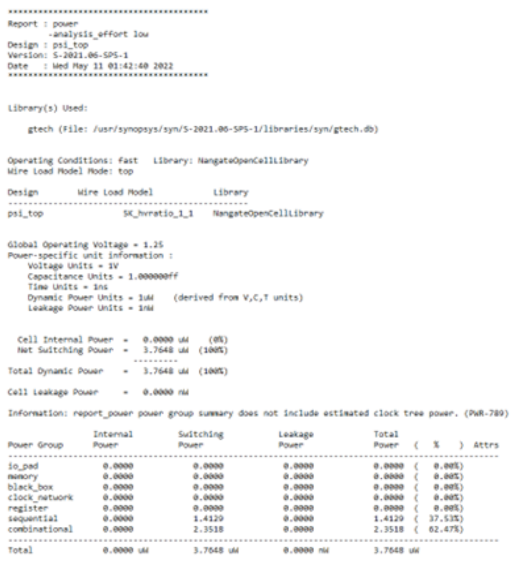


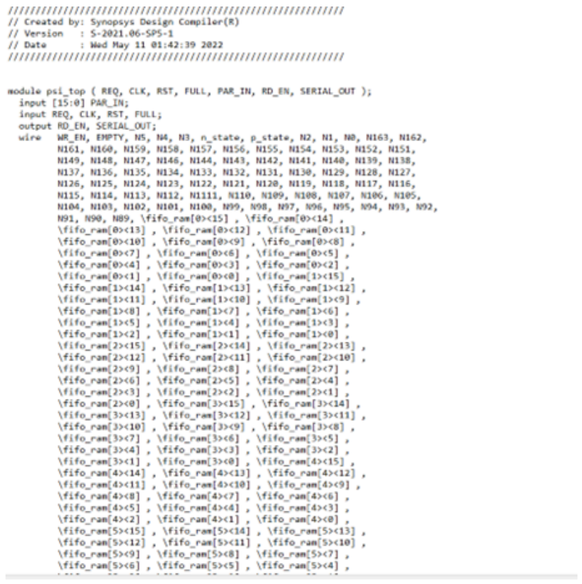
Fig. 9. Synthesis by Synopsys DC

One of the reports and gate netlist synthesized by DC compiler is shown in Fig. 10 (a), (b).



```
****************************************
Report : power
        -analysis_effort low
Design : psi_top
Version: S-2021.06-SPS-1
Date   : Wed May 11 01:42:40 2022
****************************************


Library(s) Used:

    gtech (File: /usr/synopsys/syn/S-2021.06-SPS-1/libraries/syn/gtech.db)


Operating Conditions: fast   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
psi_top                5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.25
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =   0.0000 uW    (0%)
  Net Switching Power  =   3.7648 uW  (100%)
                         ---------
Total Dynamic Power    =   3.7648 uW  (100%)

Cell Leakage Power     =   0.0000 nW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000             0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000             0.0000  (   0.00%)
register           0.0000           0.0000            0.0000             0.0000  (   0.00%)
sequential         0.0000           1.4129            0.0000             1.4129  (  37.53%)
combinational      0.0000           2.3518            0.0000             2.3518  (  62.47%)
--------------------------------------------------------------------------------------------------
Total              0.0000 uW        3.7648 uW         0.0000 nW          3.7648 uW
```

Fig. 10 (a). Power synthesis report



```
/////////////////////////////////////////////////////////////
// Created by: Synopsys Design Compiler(R)
// Version   : S-2021.06-SPS-1
// Date      : Wed May 11 01:42:39 2022
/////////////////////////////////////////////////////////////


module psi_top ( REQ, CLK, RST, FULL, PAR_IN, RD_EN, SERIAL_OUT );
  input [15:0] PAR_IN;
  input REQ, CLK, RST, FULL;
  output RD_EN, SERIAL_OUT;
  wire   WR_EN, EMPTY, N5, N4, N3, n_state, p_state, N2, N1, N0, N163, N162,
         N161, N160, N159, N158, N157, N156, N155, N154, N153, N152, N151,
         N149, N148, N147, N146, N144, N143, N142, N141, N140, N139, N138,
         N137, N136, N135, N134, N133, N132, N131, N130, N129, N128, N127,
         N126, N125, N124, N123, N122, N121, N120, N119, N118, N117, N116,
         N115, N114, N113, N112, N1111, N110, N109, N108, N107, N106, N105,
         N104, N103, N102, N101, N100, N99, N98, N97, N96, N95, N94, N93, N92,
         N91, N90, N89, \fifo_ram[0><15] , \fifo_ram[0><14] ,
         \fifo_ram[0><13] , \fifo_ram[0><12] , \fifo_ram[0><11] ,
         \fifo_ram[0><10] , \fifo_ram[0><9] , \fifo_ram[0><8] ,
         \fifo_ram[0><7] , \fifo_ram[0><6] , \fifo_ram[0><5] ,
         \fifo_ram[0><4] , \fifo_ram[0><3] , \fifo_ram[0><2] ,
         \fifo_ram[0><1] , \fifo_ram[0><0] , \fifo_ram[1><15] ,
         \fifo_ram[1><14] , \fifo_ram[1><13] , \fifo_ram[1><12] ,
         \fifo_ram[1><11] , \fifo_ram[1><10] , \fifo_ram[1><9] ,
         \fifo_ram[1><8] , \fifo_ram[1><7] , \fifo_ram[1><6] ,
         \fifo_ram[1><5] , \fifo_ram[1><4] , \fifo_ram[1><3] ,
         \fifo_ram[1><2] , \fifo_ram[1><1] , \fifo_ram[1><0] ,
         \fifo_ram[2><15] , \fifo_ram[2><14] , \fifo_ram[2><13] ,
         \fifo_ram[2><12] , \fifo_ram[2><11] , \fifo_ram[2><10] ,
         \fifo_ram[2><9] , \fifo_ram[2><8] , \fifo_ram[2><7] ,
         \fifo_ram[2><6] , \fifo_ram[2><5] , \fifo_ram[2><4] ,
         \fifo_ram[2><3] , \fifo_ram[2><2] , \fifo_ram[2><1] ,
         \fifo_ram[2><0] , \fifo_ram[3><15] , \fifo_ram[3><14] ,
         \fifo_ram[3><13] , \fifo_ram[3><12] , \fifo_ram[3><11] ,
         \fifo_ram[3><10] , \fifo_ram[3><9] , \fifo_ram[3><8] ,
         \fifo_ram[3><7] , \fifo_ram[3><6] , \fifo_ram[3><5] ,
         \fifo_ram[3><4] , \fifo_ram[3><3] , \fifo_ram[3><2] ,
         \fifo_ram[3><1] , \fifo_ram[3><0] , \fifo_ram[4><15] ,
         \fifo_ram[4><14] , \fifo_ram[4><13] , \fifo_ram[4><12] ,
         \fifo_ram[4><11] , \fifo_ram[4><10] , \fifo_ram[4><9] ,
         \fifo_ram[4><8] , \fifo_ram[4><7] , \fifo_ram[4><6] ,
         \fifo_ram[4><5] , \fifo_ram[4><4] , \fifo_ram[4><3] ,
         \fifo_ram[4><2] , \fifo_ram[4><1] , \fifo_ram[4><0] ,
         \fifo_ram[5><15] , \fifo_ram[5><14] , \fifo_ram[5><13] ,
         \fifo_ram[5><12] , \fifo_ram[5><11] , \fifo_ram[5><10] ,
         \fifo_ram[5><9] , \fifo_ram[5><8] , \fifo_ram[5><7] ,
         \fifo_ram[5><6] , \fifo_ram[5><5] , \fifo_ram[5><4] ,
```

Fig. 10 (b). Initial part of synthesized gate netlist

V. CONCLUSION

The Parallel to Serial Interfacing using a synchronous FIFO buffer was designed, simulated and synthesized using ModelSim and Synopsys DC Compiler. Initially, all the three modules were designed and simulated. They were tested with individual testbenches to verify their functionality. Later, a top module involving all three layers was simulated and tested with a unique testbench. The outputs were as per the functionalities expected. When the functionality of the designed circuit was achieved, it was then synthesized in DC Compiler. The involvement of complex memories and controls made the synthesis a bit ambiguous and thus there is a need to understand the synthesis cell libraries and this is yet to be done. This constitutes the future scope of this project and also the full ASIC flow post-synthesis is topic to be explored which is to be done as a continuation to this project in any of the Physical Design courses. Thus, any aspiring hardware engineer has to involve with all the stages of ASIC design flow as there are job and research opportunities at every stage.

VI. REFERENCES

[1] Open Systems Interconnection (OSI) model: https://www.baeldung.com/cs/osi-model.

[2] Notes of ECE-176 by Dr. A. Stillmaker at Fresno State.

[3] Navabi Z., Verilog Digital System Design (2nd edition), (2006), McGraw-Hill, pp.no. 167-171.

[4] Handout-Synthesis of EEC-281 by Prof. Dr. B. Baas, UC Davis.

[5] ModelSim® Command Reference Manual, Mentor Graphics®.

[6] Design Compiler® User Guide, Synopsys®.