Experiment 1 - Clock and Periodic Signal Generation

Amirarsalan Shahbazi [a*], Mohammad Hossein Mazhari [b]
[a] *B.A Student, Department of Computer Engineering, University of Tehran, Tehran, Iran.*
[*]Corresponding author: Email: shahbaziarsalan@ut.ac.ir. Student ID: 810101451
[b] *B.A Student, Department of Computer Engineering, University of Tehran, Tehran, Iran.*
Email: smh.mazhari@ut.ac.ir. Student ID: 810101520

I. Onepulser

A. Briefly explain the structure of the Onepulser.

Onepulser consists of three states that during these three states we manage to produce a wave that is called clock enable and it will become one during a whole clock cycle. With that we can manage the length of the clock enable that we are producing.

When we want to test our program with FPGA, we enter clock cycles by pushing a button. Because this pushing is not fast enough in range of our clock, we need to do something in order to synchronize the progress of our program with our pushing speed. So, with this Onepulser we make the program to progress with each clock push button that we push, not only with clock.

Onepulser structure in Verilog code can be seen in Fig 1.

Onepulser.v

```
1  module Onepulser(clkPB,clock,reset,clkEN);
2      input clkPB,clock,reset;
3      output reg clkEN;
4      reg [1:0] pstate , nstate;
5
6      parameter [1:0] A = 0 , B = 1 , C = 2;
7
8      always @(posedge clock,posedge reset)
9      begin
10         if(reset)
11             pstate <= A;
12         else
13             pstate <= nstate;
14     end
15
16     always @(pstate , clkPB)
17     begin
18         case (pstate)
19             A:nstate = clkPB ? B : A;
20             B:nstate = C;
21             C: nstate = ~clkPB ? A : C;
22             default:nstate = A;
23         endcase
24     end
25
26     always @(pstate)
27     begin
28         clkEN = 1'b0;
29         case (pstate)
30             B : clkEN = 1'b1;
31         endcase
32     end
33
34 endmodule
```

Fig. 1 Verilog code of Onepulser

B. Show the state machine diagram of the Onepulser.

As mentioned above the state machine has three states and is shown in Fig 2.

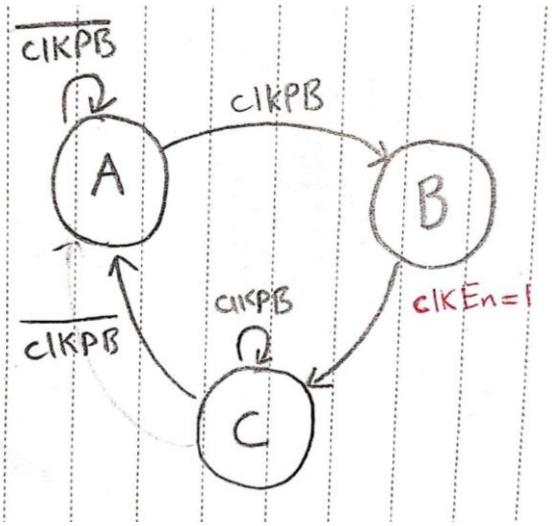

Fig. 2 State machine diagram of Onepulser

C. Show the simulation result of the Onepulser.

First of all, it started on state A (00) then when it sees clockPB it will transit to state B (01) and it will make clkEN rise and hold one for a full clock cycle. After making a clkEN cycle it will proceed to next state with no transition condition and in state C (10) it waits until we do not hold our hand anymore and when we pull our hand it will proceed back to the first state and will restart the sequence.

Result of the simulation is shown in Fig. 3. The vertical line at the end specifies end of the state C and restarting the sequence.

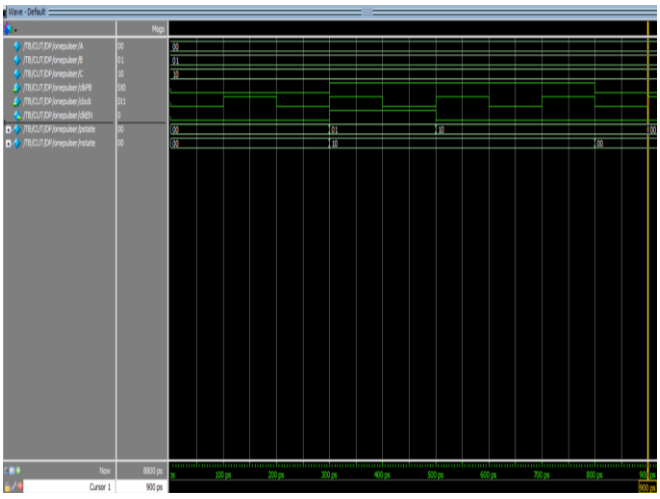

Fig. 3 Simulation result of Onepulser

II. Finite State Machine and the Counters

A. Briefly explain the structure of the controller.

First, in the Init state we wait for the SerIn to become 0 and after that the circuit starts the cycle. Then we stay two cycles in state A to receive the port number. Then we go to state B to receive number of bits we want to read. During state C, the serOutvalid remains one. The signal done will be set in the last state of the controller after receiving the whole data.

B. Show the state machine diagram of the controller.

The controller state machine is shown in Fig. 4.

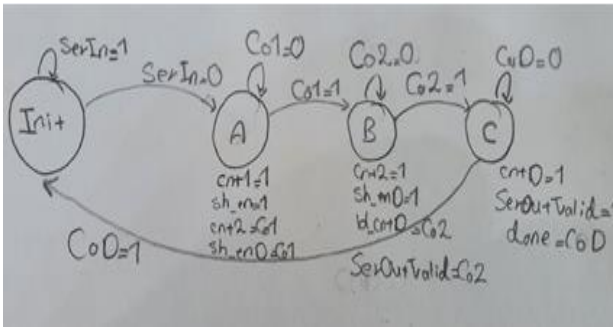

Fig. 4 Controller state machine

C. Report the Verilog code of the controller with Huffman-style coding.

Verilog code of the controller is shown in Fig. 5 and Fig .6.

controller.v

```
1  module controller(SerIn,clkEN,clock,reset,co1,co2,coD,cnt1,cnt2,cntD,ldcntD,sh_en,sh_enD,SerOutValid,done);
2      input SerIn,clkEN,clock,reset,co1,co2,coD;
3      output reg cnt1,cnt2,cntD,ldcntD,sh_en,sh_enD,SerOutValid,done;
4
5      parameter [1:0]  Init = 0 , A = 1, B = 2,C = 3;
6
7      reg [1:0] pstate , nstate;
8
9          always @(posedge clock,posedge reset)
10         begin
11             if(reset)
12                 pstate <= Init;
13             else if(clock && clkEN)
14                 pstate <= nstate;
15         end
16
17         always @(pstate , co1,co2,coD,SerIn)
18         begin
19             case (pstate)
20                 Init: nstate = SerIn? Init : A;
21                 A: nstate = co1 ? B : A;
22                 B: nstate = co2 ? C : B;
23                 C: nstate = coD ? Init : C;
24                 default:nstate = Init;
25             endcase
26         end
27
```

Fig. 5 Controller Verilog part1

```
28         always @(pstate, coD)
29         begin
30             {cnt1,cnt2,cntD,ldcntD,sh_en,sh_enD,SerOutValid,done} = 8'b0;
31             case (pstate)
32                 A:
33                 begin
34                     cnt1 = 1;
35                     sh_en = 1;// ~co1
36                     cnt2 = co1;
37                     sh_enD = co1;
38                 end
39                 B:
40                 begin
41                     cnt2 = 1;
42                     sh_enD = 1;//~co2
43                     ldcntD = co2;
44                     SerOutValid = co2;
45                 end
46                 C:
47                 begin
48                     cntD = 1;
49                     SerOutValid = 1;
50                     done = coD;
51                 end
52             endcase
53         end
54
55 endmodule
```

Fig. 6 Controller Verilog part2

D. Briefly explain each Counter and its task.

We have three counters:

1) A one-bit counter for counting for the port number and to receiving two bits.

2) A two-bit counter to count four bits for receiving number of bits we want to read next(n).

3) A 4-bit down counter to read n bits.

E. Show the simulation result of the down-counter module.

Down counter waveform result is shown in Fig. 7.

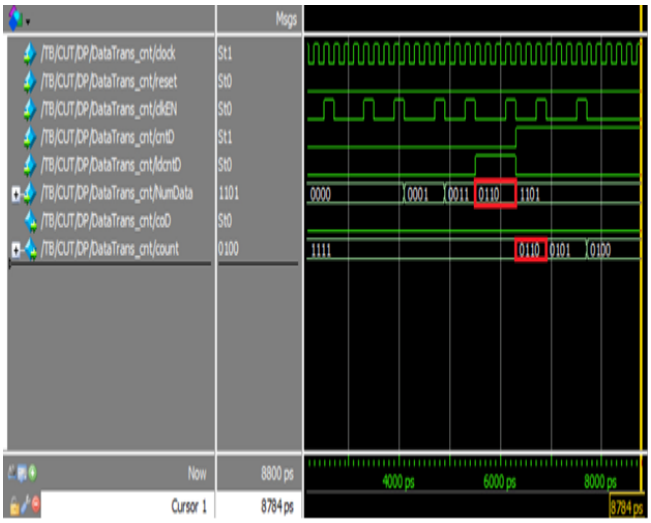

Fig. 7 Down counter result

III. Shift Registers, Demultiplexer, and SSD

A. Briefly explain each Shift register and its task.

We have two shift registers:

1) Port number shift register to save the port number.

*2)*Data Number shift register to save number bits we want to read next.

B. Show the simulation result of the data shift register module.

Simulation of a shift register is shown in Fig. 8.

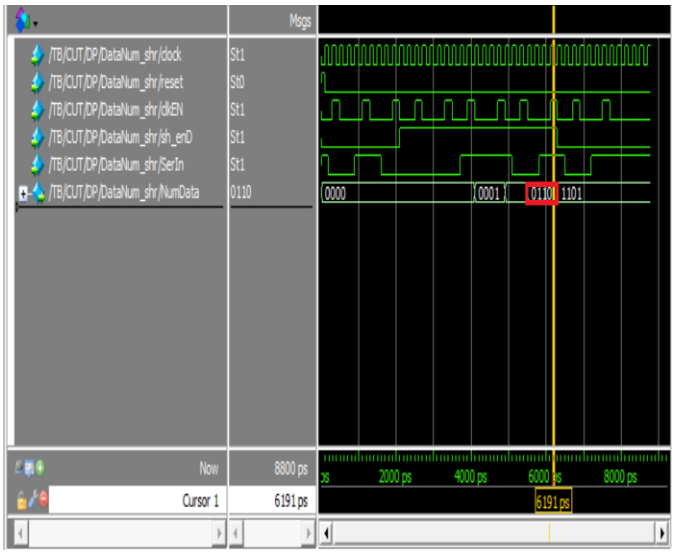

Fig. 8 Shift register waveform

C. Briefly explain the function of the Demultiplexer and show the simulation result of it.

The Demultiplexer receives the port number and the SerIn and shows the SerIn signal (0 or 1 / on or off) on the proper port.

You can see simulation result of it in Fig. 9.

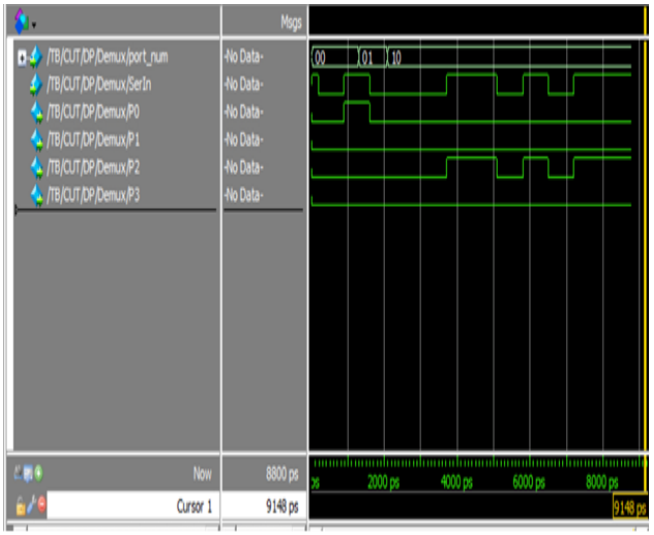

Fig. 9 Demultiplexer waveform

D. Briefly explain the function of the SSD and show the simulation result of it.

This element gets 4 bits and converts it to seven bits according to the given table shown in Fig. 10. These seven bits put a number on SSD in the circuit.

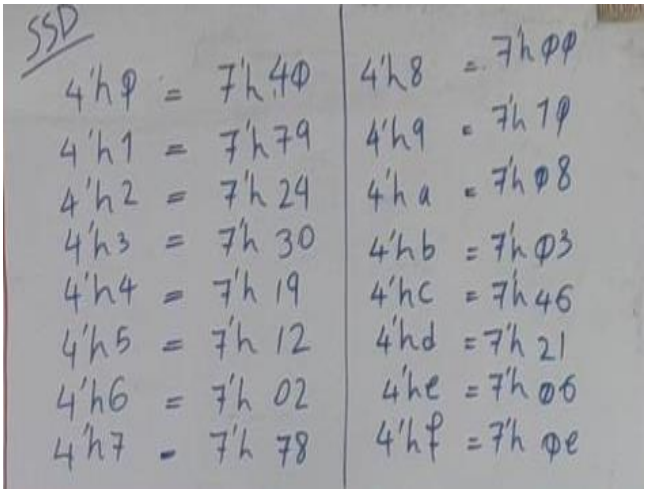

Fig. 10 SSD conversion

Simulation result of SSD is shown in Fig. 11.

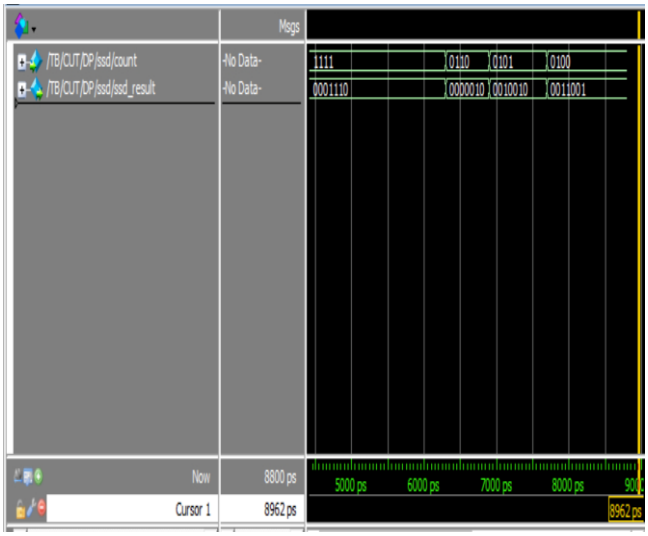

Fig. 11 SSD simulation result

IV. MSSD Simulation

A. Briefly explain the functionality of MSSD (top module) and its necessary components.

MSSD has a controller and a Datapath. Controller is discussed above but Datapath is not.

Datapath includes several components that we discussed them one by one above like registers and counters.

Datapath has 2- and 4-bit shift registers, 1- and 2- and 4-bit counters, a demultiplexer, a SSD and a Onepulser. With wiring between these components and leading them by controller we can build our MSSD.

Functionality of MSSD is that serial bits of data appear on the serIn input of MSSD. Normally, in" no transmission" mode, serIn is 1. Transmission begins when serIn makes a 1 to 0 transition. The 0-value bit marks the beginning of the transmission and has no other information. The two bits that follow are the port number, p, MSB first in time. The next 4 bits are the number of bits, n, MSB comes first in time. Data on serIn are synchronized with the MSSD clock, clk. This demultiplexer extracts the destination port and the number of bits that will go to the destination port. Following this, the next n bits will be transmitted to the destination port. Four output ports, px, will be used to transfer the data of each destination and one output port will be used to show the destination port's data count on the seven-segment.

In addition to these output ports, a done signal, and a serOutvalid signal are the outputs of MSSD. The done signal becomes 1 when the data transmission to the specified port is completed.

B. Show the simulation result of MSSD (top module) using two data sequences.

Fig. 12 shows the result simulation for data sequence (010001101011111).

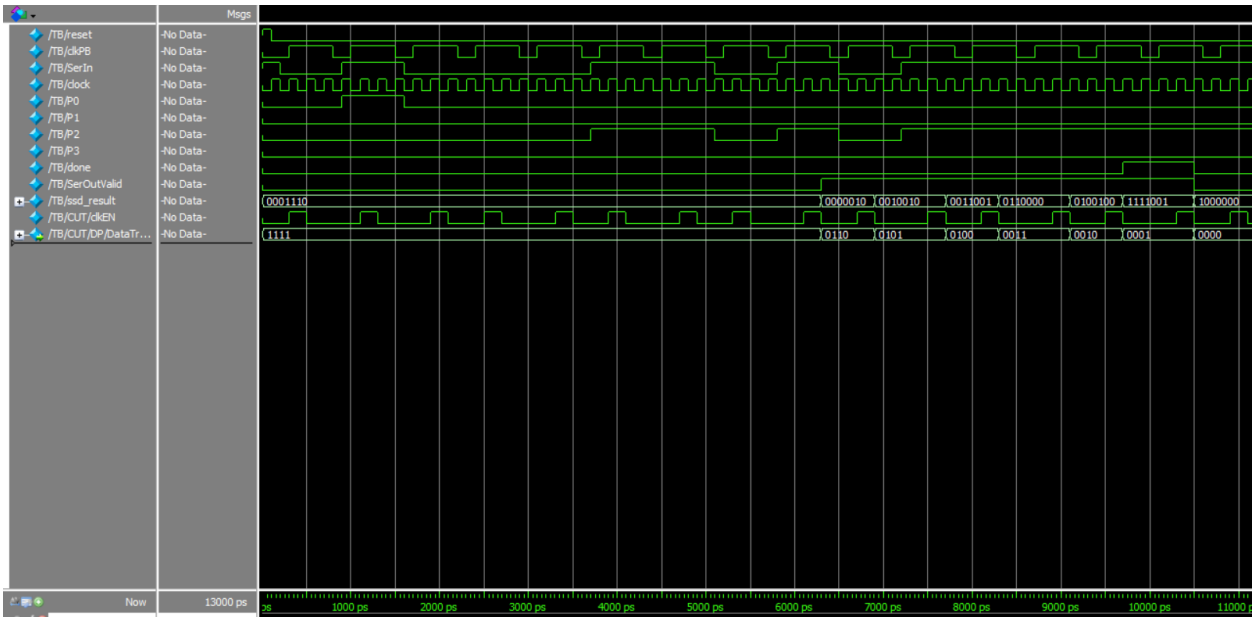

Fig. 12 First testbench waveform

Fig. 13 shows the result simulation for data sequence (10101101000000000000).

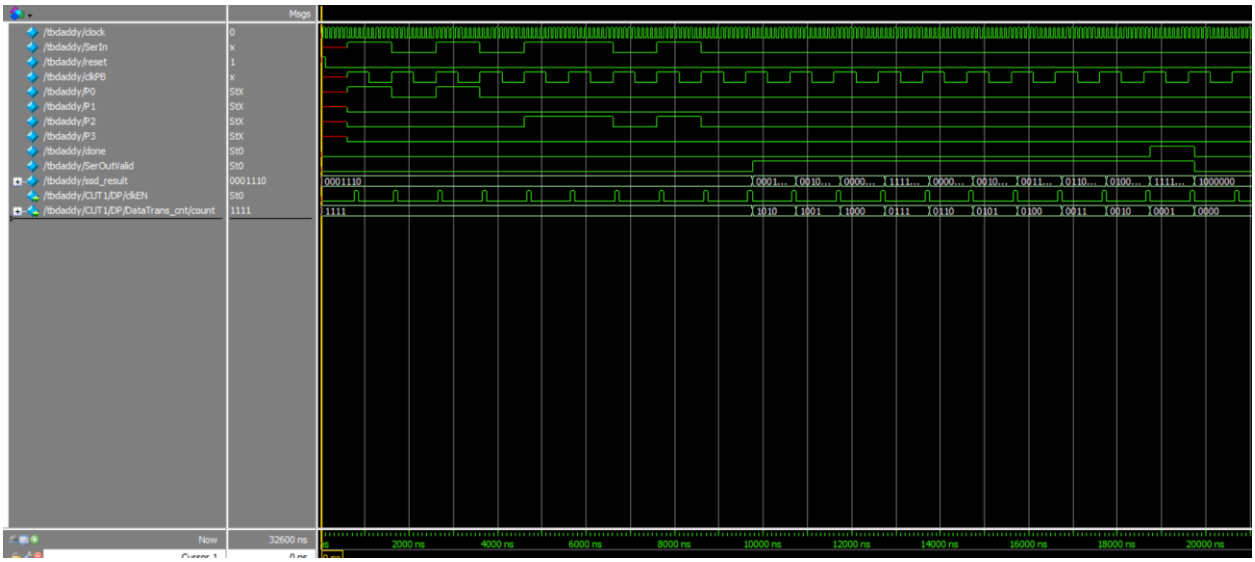

Fig. 13 Second testbench waveform

V. MSSD Implementation

A. Briefly explain your procedures to implement MSSD on the FPGA board.

First of all, we connect FPGA to a computer. Then we synthesize our Verilog code and then we connect pins of the FPGA to desired signals of our Verilog code. The manual of the pins is extracted by a reference which was introduced in class.

In details the procedure is as follows:

1. Make a Quartus project.

2. Add the top-level Verilog codes to your project.

3. Connect the main FPGA clock to the clock input of your circuit.

4. Connect a switch key to the serIn of the serial transmitter circuit, another push-button to the input of the one-pulser circuit.

5. Perform the synthesis of your design.

6. Program the Cyclone II device.

7. For the output display, use four output LEDs for displaying p0 to p3 serial data. Use two other LEDs for the done and serOutvalid signals. Use one seven-segment for displaying the pDcnt. The seven-segment is dedicated to displaying the destination port's data count. 8. When applying a serial input, change the switch key of the serIn according to the value you want on the input and then press the ClkPB push-button once.

B. Report the resource utilization of the top module after synthesizing in Quartus.

Resource utilization of top module MSSD after synthesizing is shown in Fig. 14

Flow Status	Successful - Fri May 03 23:08:34 2024
Quartus Prime Version	20.1.0 Build 711 06/05/2020 SJ Lite Edition
Revision Name	MSSD
Top-level Entity Name	MSSD
Family	Cyclone IV E
Device	EP4CE6E22A7
Timing Models	Final
Total logic elements	39 / 6,272 (< 1 %)
Total combinational functions	34 / 6,272 (< 1 %)
Dedicated logic registers	20 / 6,272 (< 1 %)
Total registers	20
Total pins	17 / 92 (18 %)
Total virtual pins	0
Total memory bits	0 / 276,480 (0 %)
Embedded Multiplier 9-bit elements	0 / 30 (0 %)
Total PLLs	0 / 2 (0 %)

Fig. 14 Synthesize result

C. Show your pin assignment selection in Quartus and specify them on the board in your captured picture.

Pin assignment selection is shown in Fig. 15.

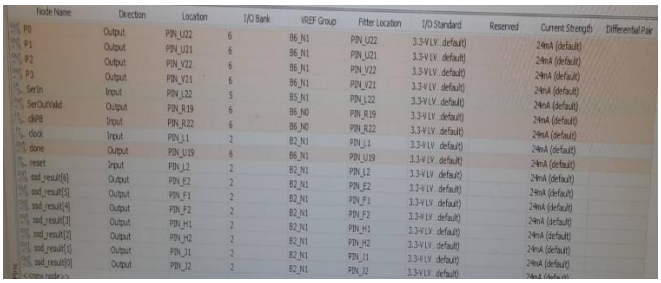

Fig. 15 Pin assignment selection

Top view of the FPGA board is shown in Fig. 16.

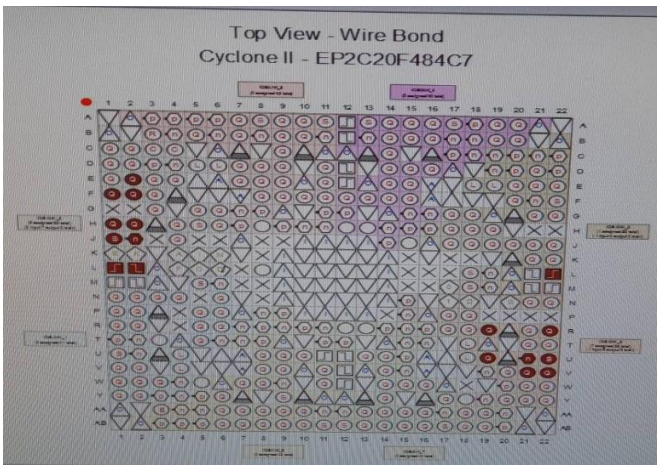

Fig. 16 FPGA board