DESIGN PROJECT

Serial to Parallel and Parallel to Serial Converter

Student Name: Tian Yuhang

LU ID: 34371567

Supervisor Name: Dr Michael D. Aspinall

Date: 2018/11/26

Lancaster University

Content

I. Overall Introduction

1.1 Background to the Project

The electronic devices consist of interlinking circuits (processors or other ICs) to generate a symbiotic system. In order to exchange the data among these circuits, a universal protocol is needed. As a sequence, a great deal of **communication protocols**, which can be divided into two genres: **parallel and serial**, have been invented to achieve this **information swap** [1] (Blom, 2009).

As the figures shown below, what the main difference between parallel and serial communication is whether transmitted bit(s) is **multiple** or **single**. As for the method of parallel interface, it is a method of conveying multiple binary digits (bits) simultaneously, **as a whole**, but in contrast, serial communication converts only a single bit at a time, **sequentially** [2] (Naskar, 2013).

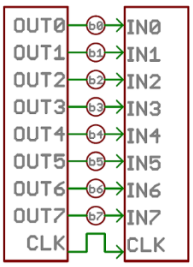

Figure 1 Parallel Communication

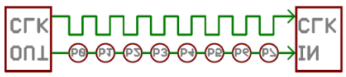

Figure 2 Serial Communication

The advantage using parallel communication is that this method provides **faster speed communication**, which shows higher profitability and efficiency. It is often used among the circuits in a computer system. Comparing to that, the advantages using serial communication are that it requires fewer interconnecting cables contributing to the save on wire area, and reduces the crosstalk effects, and increases clock speed [3] (Tambi, 2013). This method is always used in long distance communication.

1.2 Aims of this Project:

In this case, in order to achieve the communication between different types of interface, a serial-to-parallel and parallel-to-serial converter is essential. This project will involve the establishment and simulation of a Parallel-to-Serial and Serial-to-Parallel converter in VHDL.

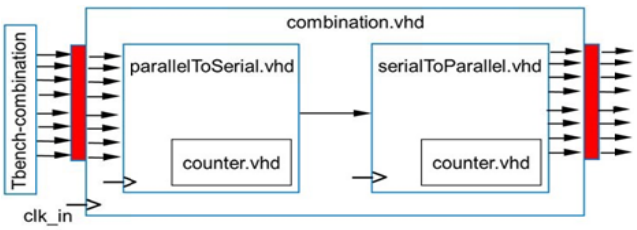

Figure 3 The diagram of two modules

1.3 Overview of the Content:

For the following sections, initially, it will sequentially illustrate how to design a counter, a parallel-to-serial converter, a serial-to-parallel converter and assembling of them, with corresponding testbenches of simulation to represent the results. Then, it will give the overall conclusions. Finally, it will give some recommendations for the future work.

II. Counter Designing

2.1 Introduction to Counter

The definition of counter is "a digital sequential logic device that will go through a certain predefined state (for example counting up or down) based on the application of the input pulses. [4] (Hou, 2005)". In this project, **4-bit counter** will be used to make the program do the various processes sequentially.

2.2 Code Analysis

4-bit up-counter, which can **increase its own value on each clock cycle** when count enable control input is 1, starts to count from "0000" to "1111" repeatedly [5] (Vahid, 2011). In this case, the code can be written as below:

```
--Counter Process--
process (clk, rst)    --clk is the input period clock cyle,
                      --rst is the enable control input
begin
if rst = '0' then Binout <= "0000"; --when rst is low, counter does not work
elsif (clk'event and clk='1') then  --when rst is high, it works
Binout <= Binout + 1;               --increase its own calue on each clock cyle
end if;
end process;

B (3 downto 0) <= Binout (3 downto 0); --send the final value to output port
```

Click here to view the whole entity if necessary

The schematic symbol of this process is shown below:

Figure 4 Counter Block

The sensitive list of this counter process includes two input signals, **clk** and **rst**. Once either of them changes, the process will start. The former is the **clock cycle** which can be set by **ALTPLL** or **manufacture**, and the latter one is the enable control input which means **only if** rst was **high**, the counter would work properly, **otherwise**, counter continuously outputs "0000" instead. In order to clearly reveal its function, the RTL viewer of Counter is provided below:

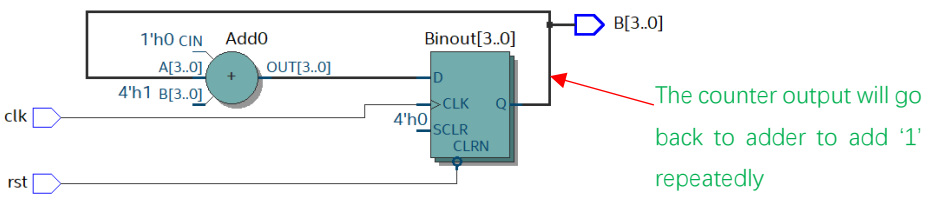

Figure 5 RTL Viewer of Counter

When the counter works, it will automatically increase its value (Binout) at the time when it meets the **up-edge** of the clock (clk'event and clk='1'). After it ends the calculating process, it will transmit the value of Binout to B (output port of the counter).

In addition, in order to test and show its function, a corresponding **testbench** is needed. Here show two kinds of its testbenches:

First approach:

```
inst1 : counter port map (clks, rsts, Bs);
Process
begin
rsts <= '0'; clks <= '0';--1st period, rst,clk='0'
wait for 10 ns;
rsts <= '1'; clks <= '0';--2nd period, rst='1' while clk='0'
wait for 10 ns;
rsts <= '1'; clks <= '1';--...
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
rsts <= '1'; clks <= '1';
wait for 10 ns;
rsts <= '1'; clks <= '0';
wait for 10 ns;
end process;
```

Click here to view the whole entity if necessary

Another approach:

```
inst1 : counter port map (clks, rsts, Bs);
Process
begin
rsts <= '0'; clks <= '0';
wait for 10 ns;
for i in 1 to 16 loop --do the following statements 16 times
clks <= '0'; rsts <= '1';
wait for 10 ns;
clks <= '1'; rsts <= '1';
wait for 10 ns;
end loop;
end process;
```

Click here to view the whole entity if necessary

What the main difference between two approaches is that the latter one uses **loop structure** to substitute the copy-and-paste process, in order to avoid repeated code statements. Comparing to the former one, the second approach is obviously briefer and easier to operate.

2.3 Working Performance & Results

This report mainly uses two method to test the code performance, **University Program VWF** and **ModelSim,** and it will give their corresponding results separately. Compared to the figure 6, all the results are correct, the counter has a proper and right function.

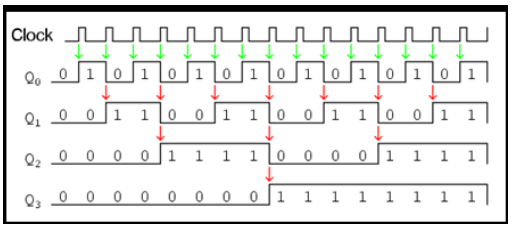

Figure 6 Counter Schematic Diagram

The result of simulation using University Program VWF:

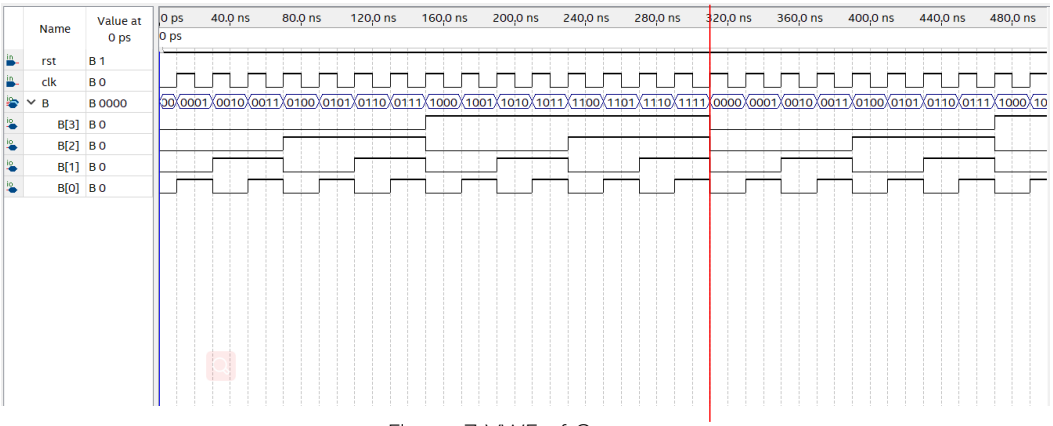

Figure 7 VWF of Counter

According to the figure 7, it reveals that the counter **dose the binary calculation repeatedly** and performs properly.

The result of simulation using ModelSim:

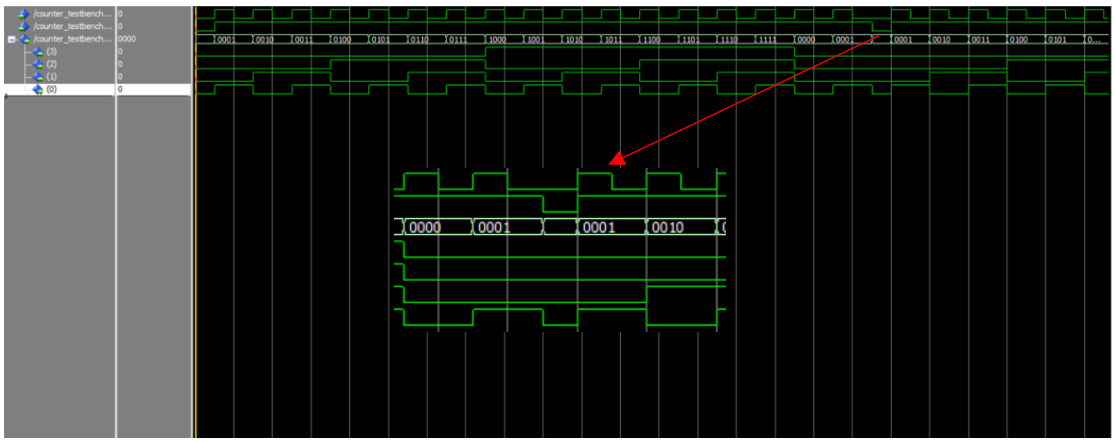

Figure 8 ModelSim of Counter for the 1st approach

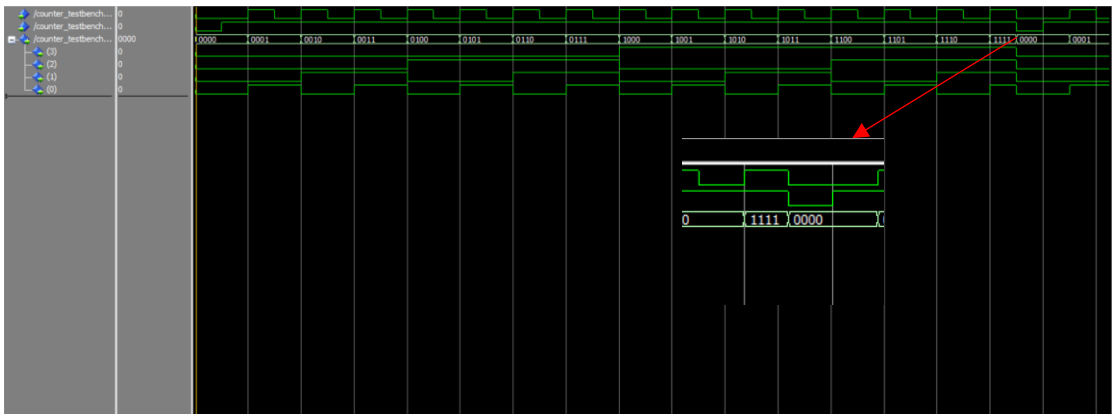

Figure 9 ModelSim of Counter for the 2nd approach

According to the figure 8 and figure 9, it can observe that the counter is reset when rst = '0' as mentioned. In realistic transmitting, just keep rst = '1' (high), then the signals can be converted properly.

2.4 Section Conclusion

This 4-bit up-counter is a sequential component that can increase (add 1) its own value on each up-edge clock cycle when count enable control input is '1', from "0000" to "1111" repeatedly. It is evident that the counter **wraps around** (also known as rolling over) from the highest value ("1111") to 0 [5] (Vahid, 2011).

In terms of the code of the counter, its function can be achieved using **binary adding calculation**. Turning to its corresponding testbench, it can use both **VWF** or **ModelSim** to simulate the results, while the former one is only the **functional simulation**, and the latter one is used more common which can show the **transient signals** at the same time if necessary. In addition, for the designers, **loop structure** can be adapted to take the place of repeated process, if a briefer structure is required.

As the figures of this section shown above, the results and the performance are all correct.

As for the next section, it will illustrate how to design a **parallel-to-serial converter** using VHDL.

III. Parallel-to-Serial Converter Design

3.1 Introduction to P2S Converter

As mentioned in section I, in order to make **parallel interface devices communicate with serial interface devices**, it needs a kind of converter which function is like a bridge between the serial data and parallel data. This bridge is to achieve **parallel-to-serial conversion**: "conversion of a stream of multiple data elements, received simultaneously, into a stream of data elements transmitted in time sequence, i.e., one at a time" [6] (bldrdoc, 1996). In this project, **8-bit P2S converter** will be designed.

3.2 Code Analysis

The 8 bits signals waiting to transmit are parallel, which required to be sent one after another, which needs a conversion using **sequential logic**. In this situation, **counter** will be used as sequential control, which will control the circuit to send one bit when it meets the next binary value. The code can be written as below:

```
architecture Behavior of parallelToSerial is
begin
   with Counter select     --"0000" is for initialization
   Q  <= D(0) when "0001", --"0001" the 1st bit will be transmitted
         D(1) when "0010", --"0010" the 2nd bit will be transmitted
         D(2) when "0011", --"0011" the 3rd bit will be transmitted
         D(3) when "0100", --...
         D(4) when "0101",
         D(5) when "0110",
         D(6) when "0111",
         D(7) when "1000",
         '0' when others;
end architecture;
```

Click here to view the whole entity if necessary

The schematic symbol of this process is shown below:

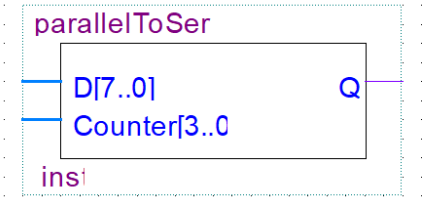

Figure 10 Parallel-to-Serial Block

It uses "**with … select**" statement (object can only be single, and here is Q) similar to **case statement** (the object can be multiple), which selects for execution one of several alternative sequences of statements; the alternative is chosen **based on the value of the associated expression**. It uses counter as associated expression; thus, Q will receive the value from D (8-bit) sequentially, taking 8 cycles. For instance, when counter = "0001", Q will output D (0); when counter = "0110", it will output D (5). It should be noticed that **Q should be initialized** in other time to avoid sending incorrect values. Furthermore, it can implement a **buffer** as

receiver to hold the value from transmitter (more details will be shown in the final section, click if want to see right now). In order to clearly reveal its function, the RTL viewer of P2S converter is provided below:

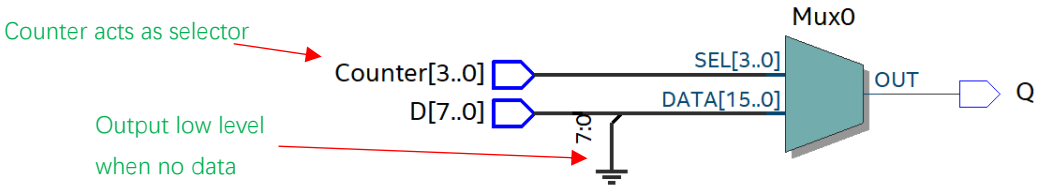

Figure 11 RTL Viewer of P2S Converter

In addition, in order to test and show its function, a corresponding **testbench** is needed. Here shows its testbench code:

```
inst1 : parallelToSerial port map (Data_input, Counters, Qs);
Process
begin
Data_input <= "00100110";  --the input data is "00100110"
for i in 1 to 8 loop
Counters <= Counters + 1;
wait for 10 ns;
end loop;
end process;
```

Click here to view the whole entity if necessary

As mentioned in section 2.2, loop structure is adapted to use here.

3.3 Working Performance & Results

Both methods, **University Program VWF** and **ModelSim**, will be used in this part of the section. The corresponding results will be shown independently.

The result of simulation using University Program VWF:

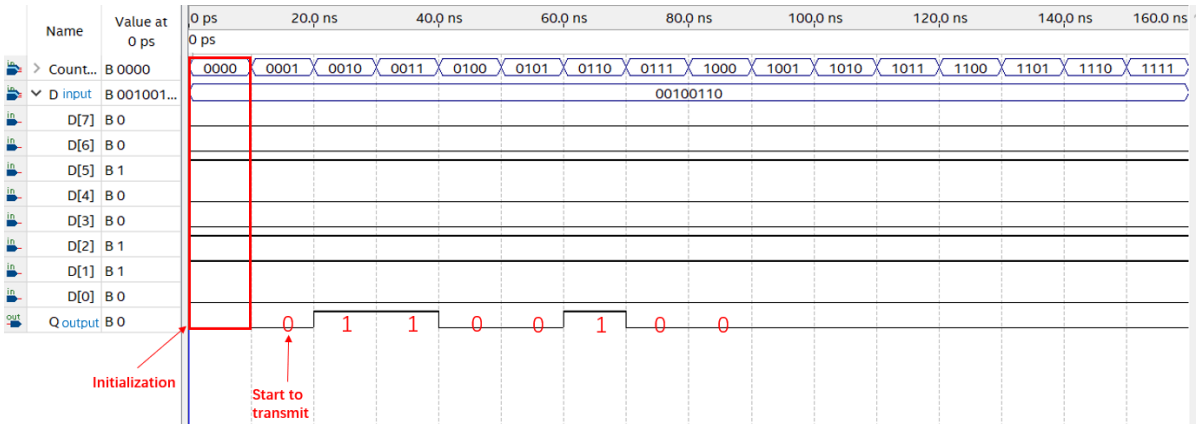

Figure 12 VWF of P2S Converter

The result of simulation using ModelSim:

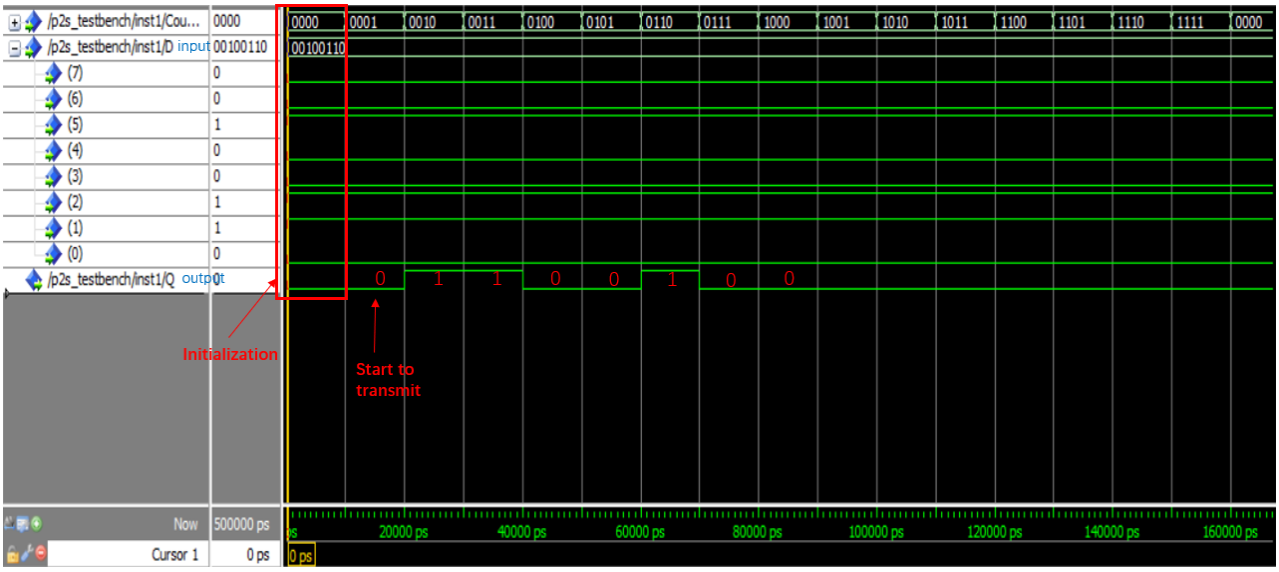

Figure 13 ModelSim of P2S Converter

As figure 12 and figure 13 shown, it does **not** start to transmit the parallel data **until** the value of counter equals to "0001", before that it **initializes** the value of Q (sets Q = '0', meanwhile does not receive the parallel signals).

In one period:

#1 When counter starts count from "0000", the process **will initially set Q = '0'** (without this step, the output will be unknown), and it will not receive the data from parallel input.

#2 When counter's value equals to "0001", the output port receives the first bit from parallel data input and sent it out.

#3 When counter's value equals to "0010", the output port receives the second bit from parallel data input and sent it out.

...

#9 When counter's value equals to "1000", the output port receives the last bit from parallel data input and sent it out. The transition is done.

#10-#16 Keep Q being low. (**For these empty cycles, it will waste the sources; thus, the process can be promoted to some extent**. more details will be shown in the final section, click if want to see right now).

More examples will be shown below. It will try to firstly send "10101010", then send "01010101", **from lower bit to higher bit**, in two neighbour periods:

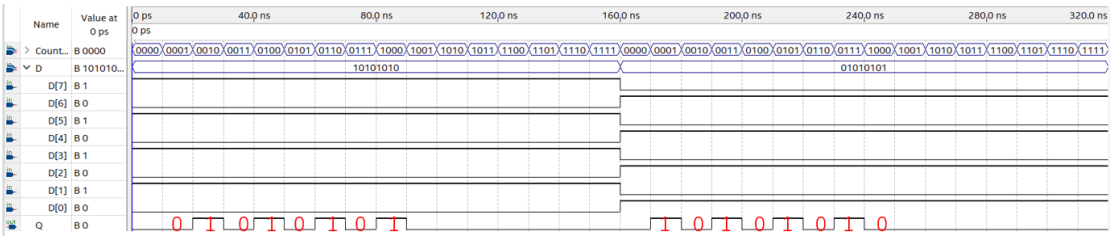

Figure 14 one example of serialization converter

Figure 15 another example of serialization converter

The repeated illustration will not be demonstrated again.

By the way, the testbench code of figure 15 is shown below:

```
inst1 : parallelToSerial port map (Data_input, Counters, Qs);
Process
begin
Data_input <= "10101010";   --the frst input data is "10101010"
for i in 1 to 16 loop       --the reason choosing 16 is one total period consitituted by small 16 cycles
Counters <= Counters + 1;
wait for 10 ns;
end loop;
Data_input <= "01010101";   --the second input data is "01010101"
for i in 1 to 16 loop
Counters <= Counters + 1;
wait for 10 ns;
end loop;
end process;
```

Click here to view the whole entity if necessary

At very beginning, the input data **from high bit to low bit** is “10101010”, and then process will enter the transmitted period (160 ns), transmitting the bit from high to low. Then, the input data changes into “01010101”, and then again will enter the transmitted period. The reason why it chooses 16 here is that **counter counts from “0000” to “1111” will spend 16 cycles**.

3.4 Section Conclusion

In this section, it has demonstrated the **function** of sterilization converter, and at the same time, it has provided the both **code of the parallel-to-serial converter and its corresponding testbench**. Furthermore, it has shown the **results** of two methods to simulate, also with proper **illustration**.

In terms of the next section, it will illustrate how to design a **serial-to-parallel converter**, also known as de-sterilization converter, using VHDL.

IV. Serial-to-Parallel Converter Design

4.1 Introduction to S2P Converter

In the section III, the parallel has been converted into serial form to transmit, which is not the original parallel form. In order to make the serial data recover back to its original form, the **de-serialization process** is needed. In this section, it will provide **de-serialization converter** for the receiver, which function is to do the **serial-to-parallel conversion**. This conversion means that a stream of data elements received in time sequence, i.e., one at a time, into a data stream consisting of multiple data elements transmitted simultaneously [7] (bldrdoc, 1996). In this project, **8-bit S2P converter** will be designed.

4.2 Code Analysis

The **signals waiting to receive are serial**, which required to be received in sequence, and not until the time when all bits are received, it will output the 8-bit signals **as a whole**. In this situation, **counter** will be used as sequential control, which will control the circuit to receive the signal one by one. In addition, a **buffer** is needed to store the signals for at least 8 cycles (receiving time). After all bits are received, it will **export the data in parallel**. The code can be written as below:

```
architecture Behavior of serialToParallel is

signal x : std_logic_vector(7 downto 0);   --X will be acted as a buffer to
                                           --store the data temporarily in one period
attribute keep: boolean;                   --keep VHDL synthesis attribute

begin

   process(Counter,D)                      --when either of counter or D changes, the process will begin
   begin
   if Counter = "0000" then x <= "00000000"; --the first cycle is used for initialization
   elsif Counter = "0001" then x(0) <= D;    --"0001" x(0) will receive and store the 1st bit
   elsif Counter = "0010" then x(1) <= D;    --"0010" x(1) will receive and store the 2nd bit
   elsif Counter = "0011" then x(2) <= D;    --"0011" x(1) will receive and store the 3rd bit
   elsif Counter = "0100" then x(3) <= D;    --...
   elsif Counter = "0101" then x(4) <= D;
   elsif Counter = "0110" then x(5) <= D;
   elsif Counter = "0111" then x(6) <= D;
   elsif Counter = "1000" then x(7) <= D;    --"1000" the last bit has been received
   else null;
   end if;
   end process;

   with Counter select
   Q <= x when "1001",                       --"1001" transmit the data from buffer to output
   "00000000" when others;                   --export "0000_0000" in other situations

end architecture;
```

Click here to view the whole entity if necessary

The schematic symbol of this process is shown below:

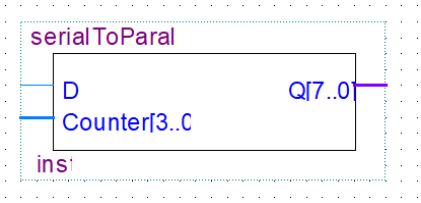

Figure 16 Serial-to-Parallel Block

The code uses **"if" statement** and put it into a **process** which **sensitive list** includes "**Counter**" and "**D**". When the change of **Counter** or **D** happens, it will enter the process, and runs the statement sequentially. "if…then…" is **event statement** which is an important concept in VHDL. It relates to signals and it occurs on a signal if the current value of that signal changes. In other words, an event on a signal is a change of the signal's value. **In this process, if at a time when counter = "xxxx" was true, it would do the corresponding event, passing the value of D to x(i)**. For instance, if the value of counter at this time was "0111", it would pass the value of the seventh bit to x(6), in order to temporally latch it to wait all bits arriving. When in other time outside transmitting period, it will do nothing.

"with…select" is also used here for **setting the proper time** to export the parallel data as a whole. As is shown, it ends the receiving process when all bits are accepted. That is when counter is "1000", **Q can export all bits in next cycle ("1001")**, as 8-bit parallel data, while in any other time, **Q** will consistently output "0000_0000". **Without** ""0000_0000" when others", it will persistently export the result in the rest of one period. (it can be set depending on the situations and requirements)

In order to clearly reveal its function, the RTL viewer of S2P converter is provided below:

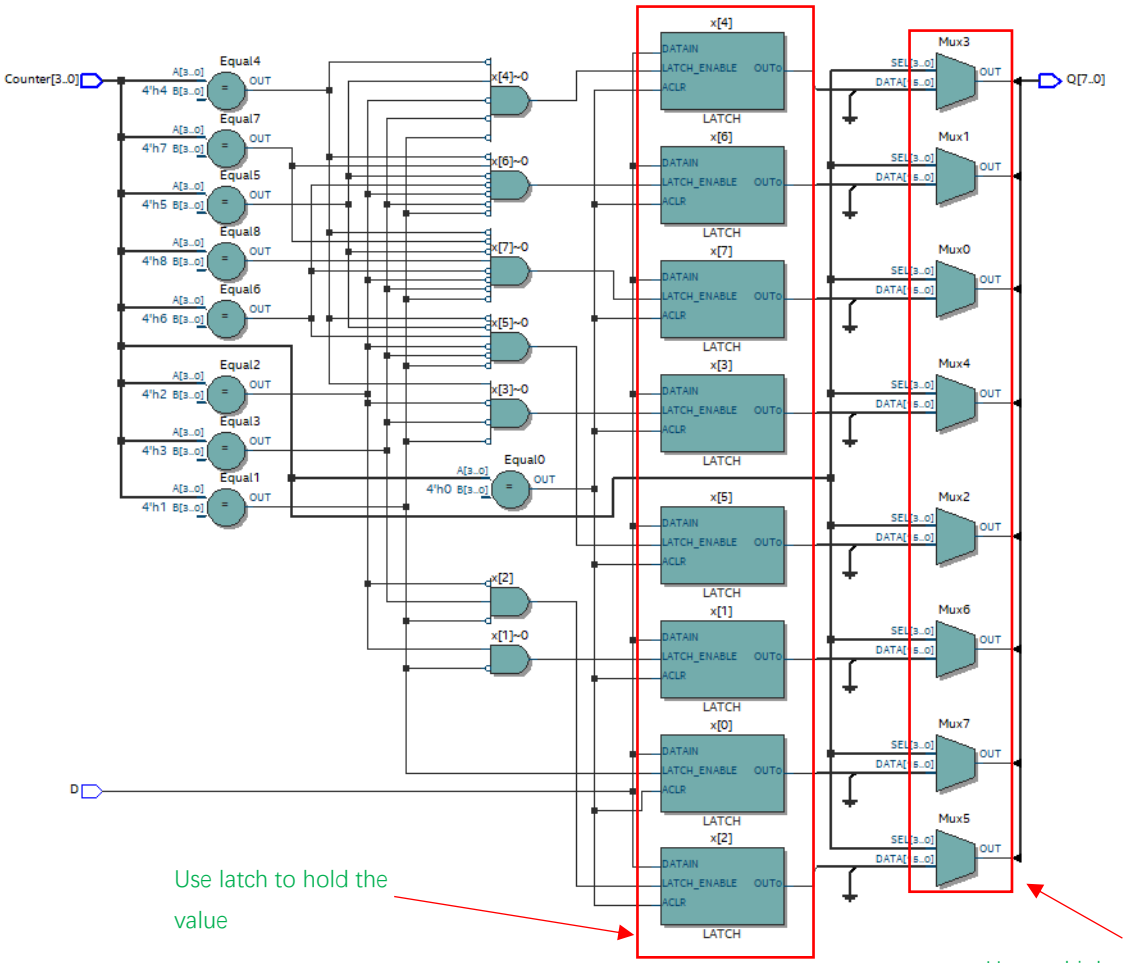

Figure 17 RTL viewer of S2P converter

In addition, in order to test and show its function, a corresponding **testbench** is needed. Here shows its testbench code:

```
inst1 : serialToParallel port map (Data_input, Counters, Qs);
Process
begin
Data_input <= '0'; Counters <= "0000";        --initialize both serial data input and counter
wait for 10 ns;
Data_input <= '1'; Counters <= Counters + 1; --serial input is '1';counter is "0001"
wait for 10 ns;
Data_input <= '0'; Counters <= Counters + 1; --serial input is '0';counter is "0010"
wait for 10 ns;
Data_input <= '1'; Counters <= Counters + 1; --serial input is '1';counter is "0011"
wait for 10 ns;
Data_input <= '0'; Counters <= Counters + 1; --...
wait for 10 ns;
Data_input <= '1'; Counters <= Counters + 1;
wait for 10 ns;
Data_input <= '0'; Counters <= Counters + 1;
wait for 10 ns;
Data_input <= '1'; Counters <= Counters + 1;
wait for 10 ns;
Data_input <= '0'; Counters <= Counters + 1; --serial input is '0';counter is "1000"
wait for 10 ns;
end process;
```

Serial data input is "01010101"

Click here to view the whole entity if necessary

The first cycle is for initialization, and for the next 8 cycles, it will transmit "01010101".

4.3 Working Performance and Results

Only **ModelSim** will be used in this part of the section, because University Program VWF's result is the same which is not necessary. The corresponding results will be shown below:

The result of simulation using ModelSim:

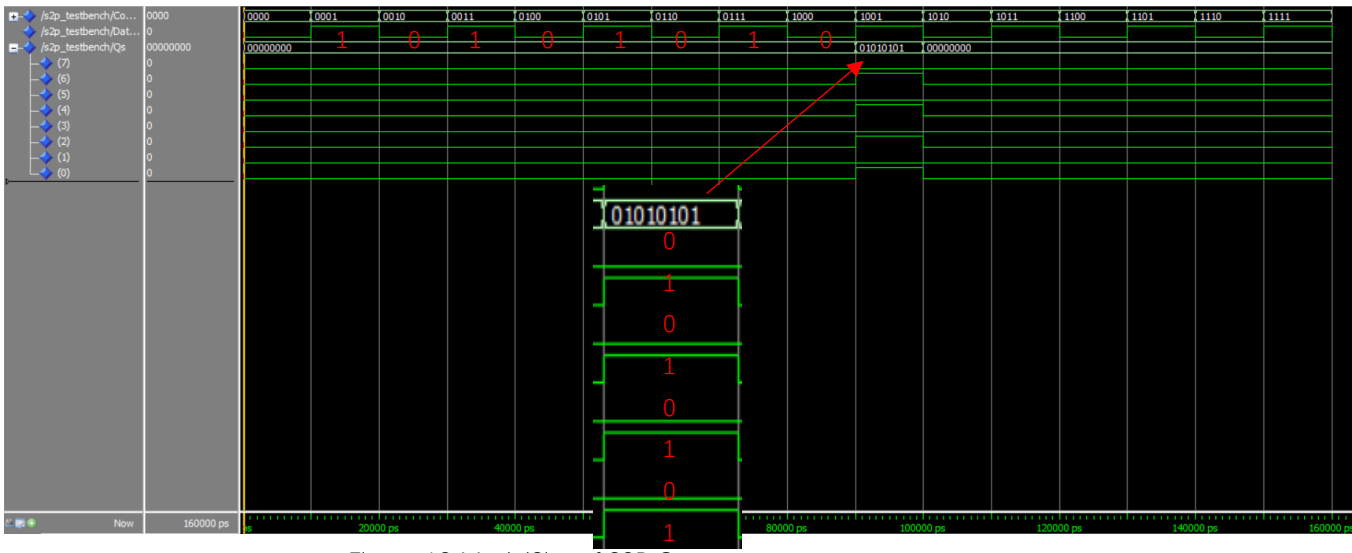

Figure 18 ModelSim of S2P Converter

As figure 18 shown, it does **not** start to export the parallel data **until** the value of counter equals to "1001", before that, it **initializes** the value of Q and x (buffer) (sets Q, x =

"0000_0000", when counter equals to "0000").
In one period:
#1 When counter starts count from "0000", the process **will initially set Q, x = "0000_0000"** (without this step, the output will be unknown), and it will not receive the data from serial input line.
#2 When counter's value equals to "0001", the buffer receives the first bit from serial data input and store it, meanwhile it will not send it to output port.
#3 When counter's value equals to "0010", the buffer receives the second bit from serial data input and store it, meanwhile it will not send it to output port.
...
#9 When counter's value equals to "1000", the buffer receives the last bit from serial data input and store it. The transition is done.
#10 When counter's value equals to "1001", the buffer will transmit all the stored values to output port, which will export all data in parallel form, in other words, transmission as a whole.
#11-#16 Keep Q, x being low. (**For these empty cycles, it will waste the sources; thus, the process can be promoted to some extent**. more details will be shown in the final section, click if want to see right now).

One more example will be shown below. It will try to firstly send "01010101", then send "10101010" in two close period:

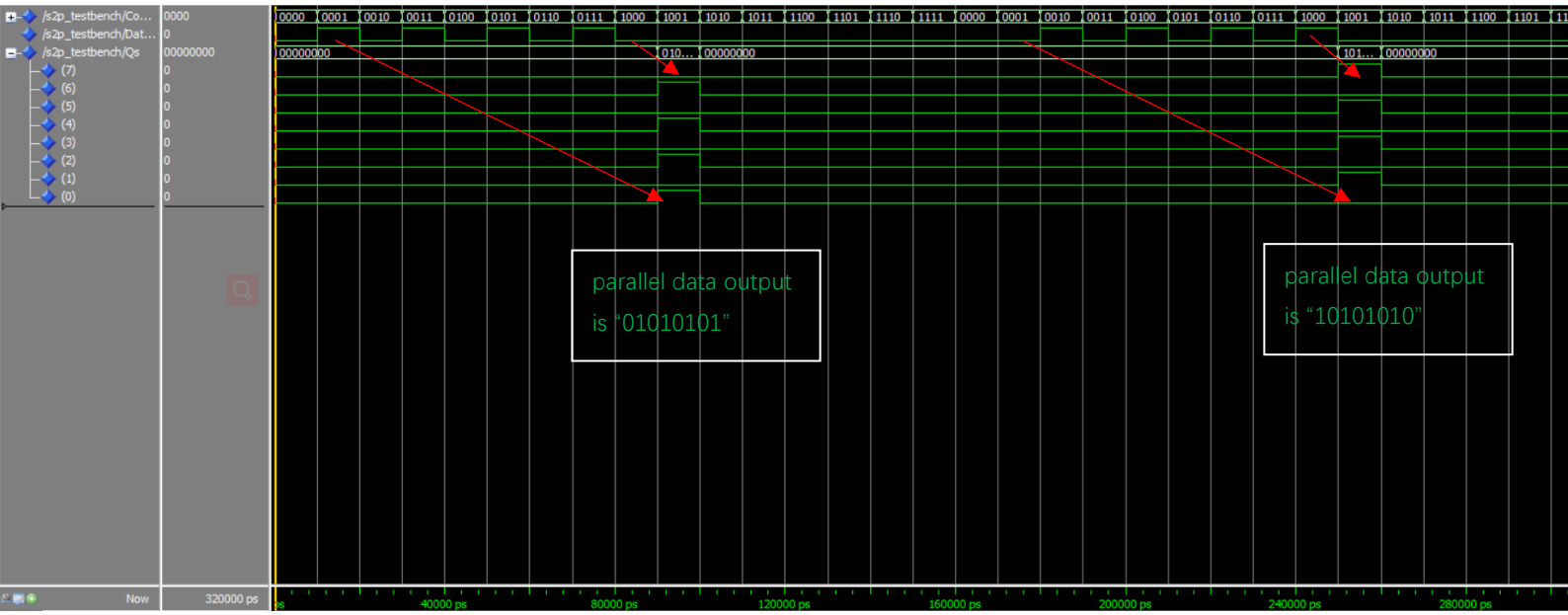

Figure 19 another example of de-serialization converter

According to the figure 19, its function is correct.

4.4 Section Conclusion

In this section, it has demonstrated the **function** of de-sterilization converter, and at the same time, it has provided the both **code of the serial-to-parallel converter and its corresponding testbench**. Furthermore, it has shown the **results** of using ModelSim to simulate, also with proper **illustration**.

In terms of the next section, it will illustrate how to combine all the components, **counter**, **serial-to-parallel converter** and **parallel-to-serial converter** together to constitute the final integrated circuit shown in **figure 3**.

V. Combination Design

5.1 Introduction to Combination Circuit

In section II, III, and IV, it has shown how to build **a counter**, **a serialization converter** and **a de-serialization converter**, respectively. In this section, it will show how to connect them into a whole circuit named **combination circuit** in a proper way, and test its function. In addition, it will also briefly demonstrate **how to download the program onto FPGA board**.

5.2 Connection of Components

In the previous sections, it has created the blocks of a counter and two converters. These bocks (figure.4, figure.10, figure.16) can be connected together in **Block Diagram/Schematic File** in Quartus Prime. The connected blocks' diagram has shown below:

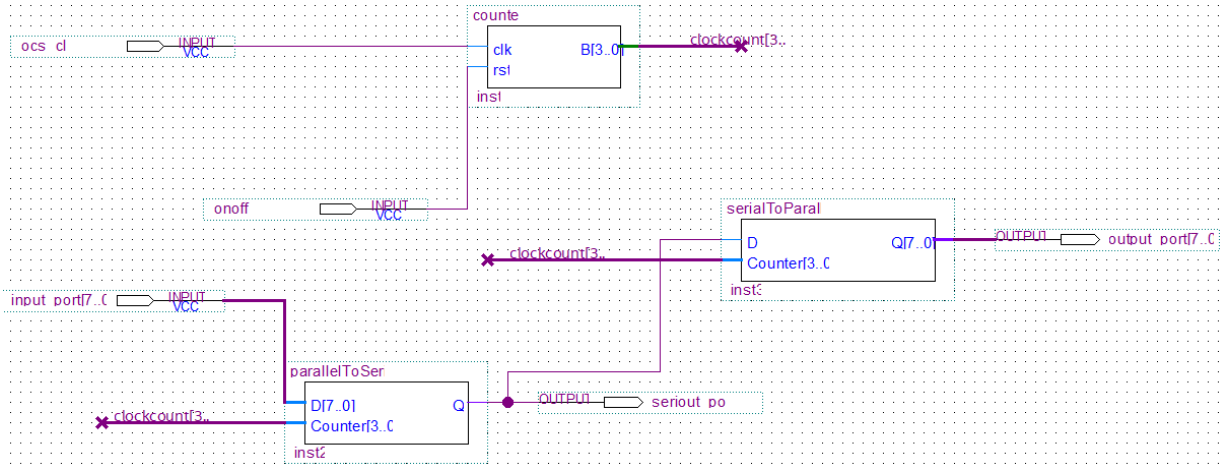

Figure 20 Combination Circuit Diagram

As figure 20 shown, both converters are controlled by a common counter, therefore, their counter's ports should be connected with the counter output port.
Serialization Converter should be connected with De-serialization Converter in series, because the former output is the latter input.
Data should be input from Serialization Converter.
In fact, the connection using this method is not complicated, but it also can be connected using component declaration using VHDL code instead, which seems more complicated. (click to view this alternative method if necessary)

The schematic symbol of this process is shown below:

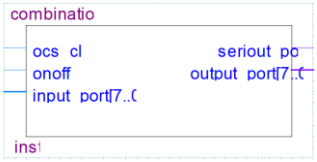

Figure 21 Combination Block

In order to clearly reveal its function, the RTL viewer of S2P converter is provided below:

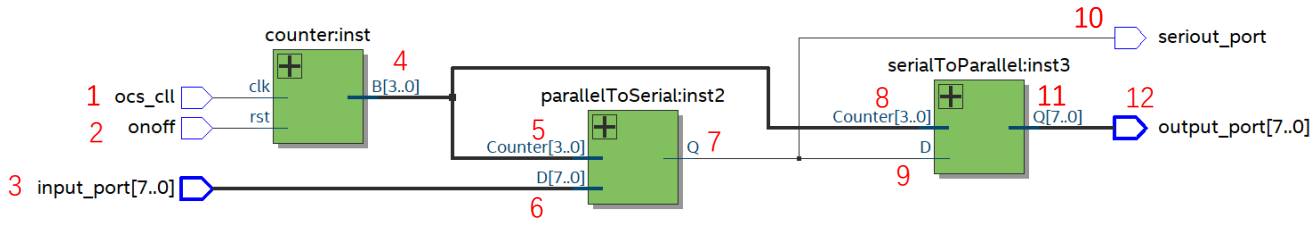

Figure 22 RTL viewer of combination circuit

Instruction:

1. **ocs_cll** is the **clock input**. It should be set by manufacture, but when it is used on FPGA board, it should be set by **ALTPL** to send the input clock.

2. **onoff** is an **on/off switch**, also a reset bottom, to determine whether the circuit will work or not.

3. **input_port[7..0]** is the **parallel data input port** which is the initial point of the data.

4. **B[3..0]** is the **output port of counter** which will export the counter signals.

5. **Counter[3..0]** is the **counter input port** of P2S converter.

6. **D[7..0]** is the **data input port** of P2S converter.

7. Q is the serial data output port of P2S converter.

8. **Counter[3..0]** is the **counter input port** of S2P converter.

9. **D** is the **serial data input port** of S2P converter.

10. **seriout_port** is the **serial data output port**.

11. **Q[7..0]** is the **parallel data output port** of S2P converter.

12. **output_port[7..0]** is the **parallel data output port** of the whole circuit.

In addition, in order to test and show its function, a corresponding **testbench** is needed. Here shows its testbench code:

```
inst1 : combination port map (osc_clls, onoffs, input_ports, seriout_ports, output_ports);
input_ports <= "10001011";        --the data waiting to transmit
Process
begin
onoffs <= '0'; osc_clls <= '0';   --initialization
wait for 10 ns;
for i in 1 to 16 loop             --one transmitting period
onoffs <= '1'; osc_clls <= '1';
wait for 10 ns;
onoffs <= '1';osc_clls <= '0';
wait for 10 ns;
end loop;
end process;
```

Click here to view the whole entity if necessary

For this test bench, it still uses loop structure. The data waiting to transmit is “1000_1011”.

5.3 Working Performance and Results

Only **ModelSim** will be used in this part of the section where the corresponding results will be shown below.

The result of simulation using ModelSim:

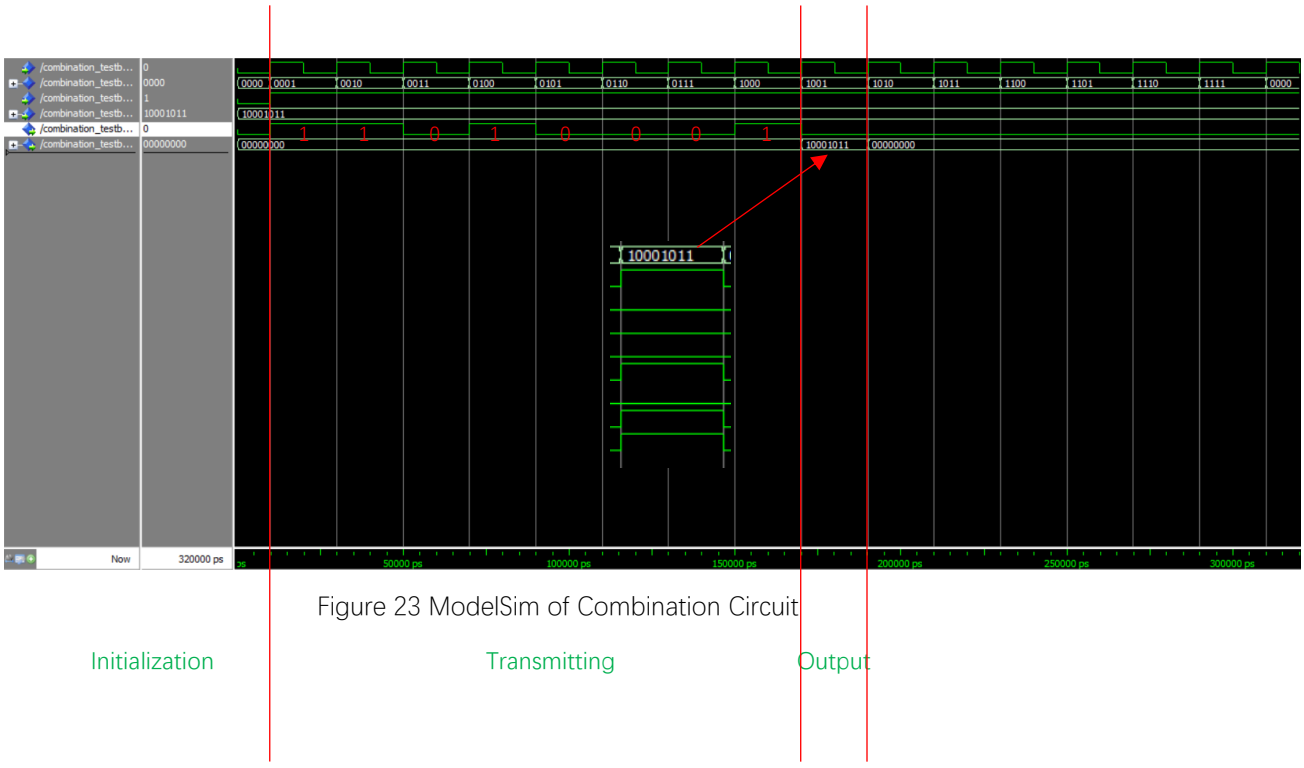

Figure 23 ModelSim of Combination Circuit

According to figure 23, it performs well.

Intially, the process will initialize all the signals’ value, in order to avoid outputting unknown value.

In terms of **sender**, the input parallel data, as mentioned, from higher bit to lower bit is “1000_1011”, then it will be serialized and be transmitted in sequence, ‘1’, ‘1’, ‘0’, ‘1’, ‘0’, ‘0’,

'0', '1', which will take 8 cycles.

Meanwhile, turning to the **receiver**, it will receive the serial stream and store all the bits in a buffer before counter counts to "1001". When the value of counter equals to "1001", the de-serialization converter will be triggered to send all the bits out in parallel, "1000_1011", which is the same to the original input.

The rest of the period will do nothing but send low level. This is the end of one working period.

One more example will be shown below. It will try to firstly send "01010101", then send "10101010" in two neighbour periods:

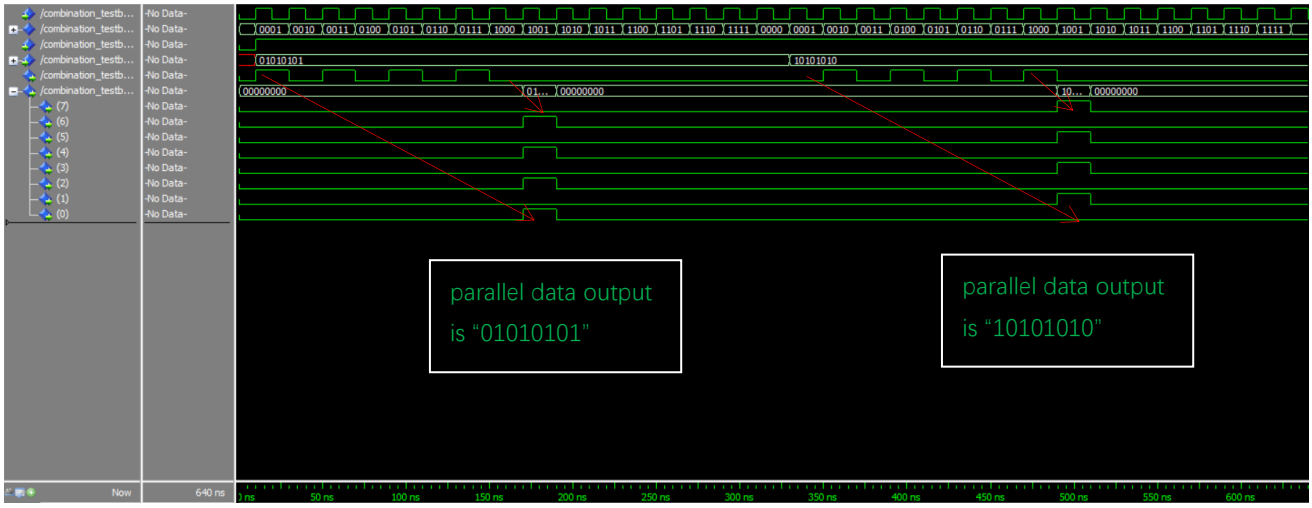

Figure 24 another example of combination circuit

5.4 Download onto FPGA Board

Before downloading the program on FPGA, it needs to be added one more component which is shown below:

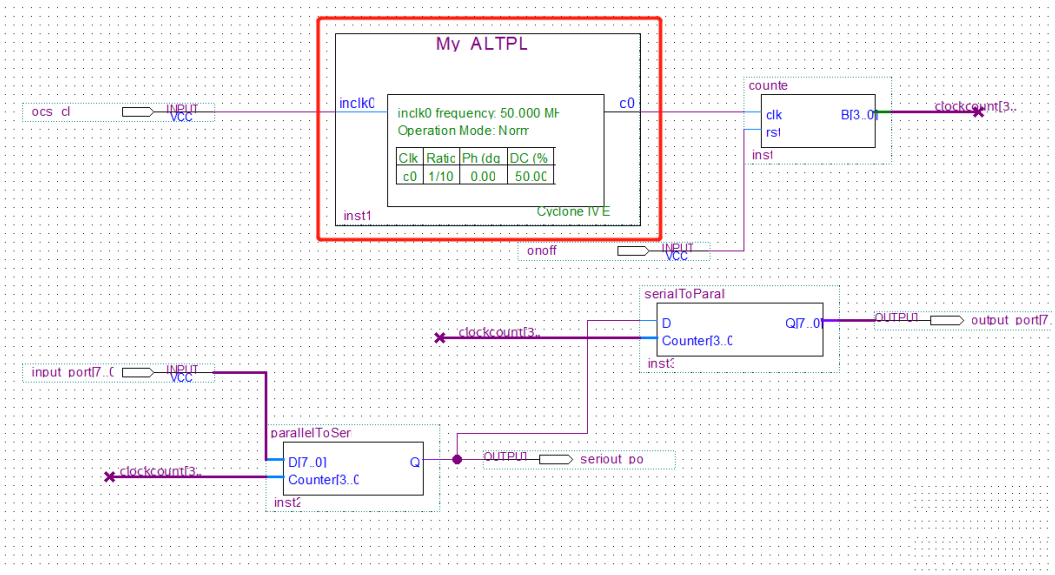

Figure 25 ALTPLL Block

ALTPL can be used to control the clock input from FPGA. The clock and counter should be set properly, otherwise the output phenomenon cannot be captured by human eyes. The light will be on constantly. The counter is suggested to set in following value:

```
process (clk, rst)
begin
if rst = '0' then Binout <= x"000000";
elsif (clk'event and clk='1') then
Binout <= Binout + 1;
end if;
end process;

B (3 downto 0) <= Binout (23 downto 20);
```

Besides that, it needs to set **PIN assignment** before being converted to FPGA.

The test video made by writer has uploaded to the internet for readers to view:
https://youtu.be/NgIef0AVLOc

5.5 Section Conclusion

In this section, it has initially demonstrated how to build the combination circuit using the components' blocks created in the above sections. In addition, it has illustrated the connection methods to connect the components. Then, it has shown the testbench of the whole circuit. At last, it also gives some results of simulation with some interpretations.

In the following section, it will give an overall conclusion for what the project has done till now.

VI. Overall Conclusion

Section I. It has illustrated the background of this project, the difference between serial communication and parallel communication, the advantages and drawbacks of these various method to transmit the data. It has given the main aim of this project is to build a **serialization and de-serialization converter** using VHDL code.

Section II. It has shown how to set up a **4-bit up-counter** which function is to count from "0000" to "1111" using binary addition. It has also provided the testbench with simulating result of this counter. One of the testbenches is used loop structure, which will be easier for the operator to pre-set before using ModelSim to simulate.

Section III. It has demonstrated how to construct a **parallel-to-serial converter** using "with···select" structure. When it is at the proper time (controlled by the counter), the parallel bits from sender will be sent to a single transmitting line, which means that the data will be converted sequentially. It has also provided the testbench with simulating result of this module.

Section IV. It has demonstrated how to construct a **serial-to-parallel converter** using "if" and "with···select" structures. The data transmitted from the line will be received in order and be stored in a buffer, waiting the right time to convert to the output port (controlled by the counter). It has also provided the testbench with simulating result of this module.

Section V. It has interpreted how to constitute all the components together to be a **combination circuit**. Either of connection using block diagram or component declaration can achieve the links among the different components. It has also provided the testbench with simulating result of this module.

This project has illustrated how to design a counter, both serialization and de-serialization converter using VHDL. In addition, it has also demonstrated how to combine all the aspects together into an integrated circuit. For each part, it has revealed the corresponding testbench whit its result. All the results have shown that the function of this design works properly.

In this report, it has also given some flaws of this design which will be discussed in the following section which will identify future work that can be done including show the limitations of this project design and how could it become more efficient.

VII. Recommendations for Future Work

7.1 Introduction to Recommendations for Future Work

In this section, it will illustrate some improvement work which will be done in the future, mainly divided into three part. The first one is **implementing a buffer in serialization converter**. The next one is that there are some **resources wasted during transmitting and receiving data bits**. The final part will conclude the future work and give an overall assessment.

7.2 Buffer in Serialization

As it mentioned in section 3.2, a **buffer,** as receiver, has the ability to hold the value from sender. If the transmitting data was disturbed from outside, the bits waiting to serialize would be adversely influenced. For instance, “10101010” changed into “11101010” due to the outside disturbance when counter = “0101”, but the seventh bit will **only** be received when the value of counter equals to “0111”, which would cause the **error seventh bit receiving**.
In this situation, if a buffer was implemented in this process, which function was only received the data when counter = “0000” and **locked values until next period**, it would avoid the above problem.

The code is provided below:

```
process(Counter)
begin
if Counter = "0000" then buff_er <= D; --"0000" buffer will receive and lock the values
else null;
end if;
end process;

with Counter select           --"0000" is for initialization
Q  <= buff_er(0) when "0001", --"0001" the 1st bit will be transimtted
      buff_er(1) when "0010", --"0010" the 2nd bit will be transimtted
      buff_er(2) when "0011", --"0011" the 3rd bit will be transimtted
      buff_er(3) when "0100", --...
      buff_er(4) when "0101",
      buff_er(5) when "0110",
      buff_er(6) when "0111",
      buff_er(7) when "1000",
      '0' when others;
```

Click here to view the whole entity if necessary

In order to clearly reveal its function, the RTL viewer of S2P converter is provided below:

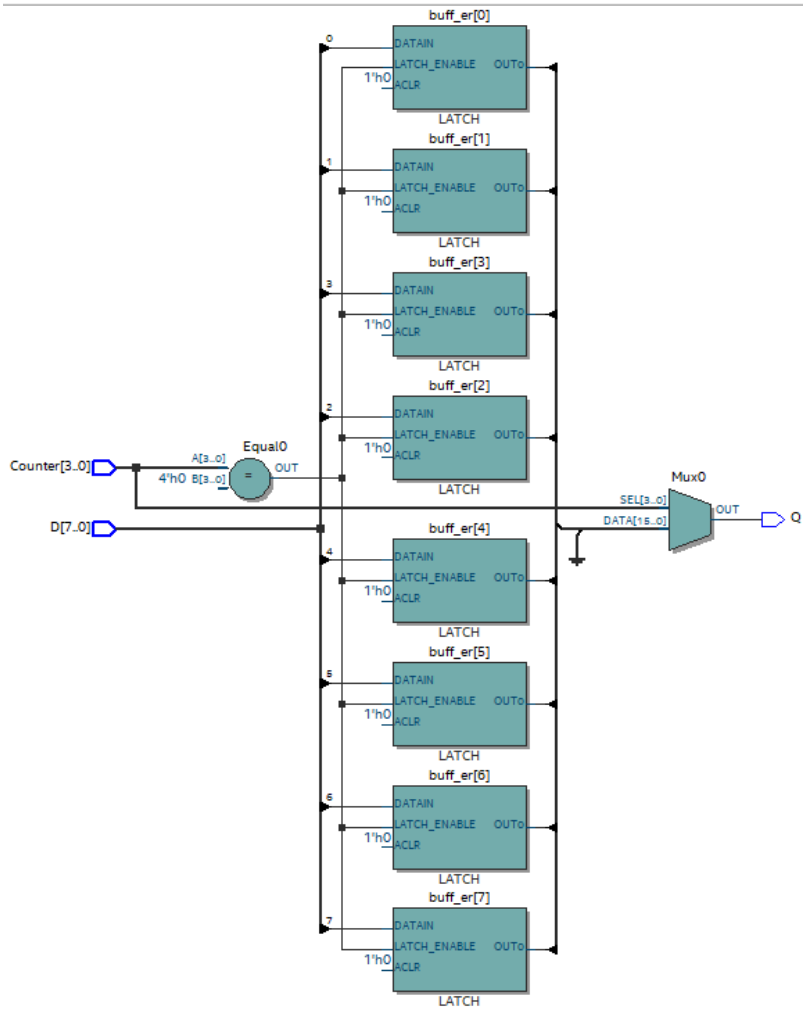

Figure 26 RTL viewer of P2S Converter with Buffer

Compared with P2S converter without buffer shown in figure 11, it has an extra part constituted by eight latches which have the ability to hold the received bits in one period.

In order to test its new function, a new testbench has been created shown below:

```
inst1 : parallelToSerial port map (Data_input, Counters, Qs);
Process
begin
Data_input <= "10101010";
for i in 1 to 5 loop
Counters <= Counters + 1;
wait for 10 ns;
end loop;
Data_input <= "11101010";
for i in 1 to 11 loop
Counters <= Counters + 1;
wait for 10 ns;
end loop;
end process;
```

Click here to view the whole entity if necessary

It will initially convert “10101010” in the first five cycles, but will change its seventh bit from

'0' to '1' in next cycle (it has not been received).

The result of simulation using ModelSim:

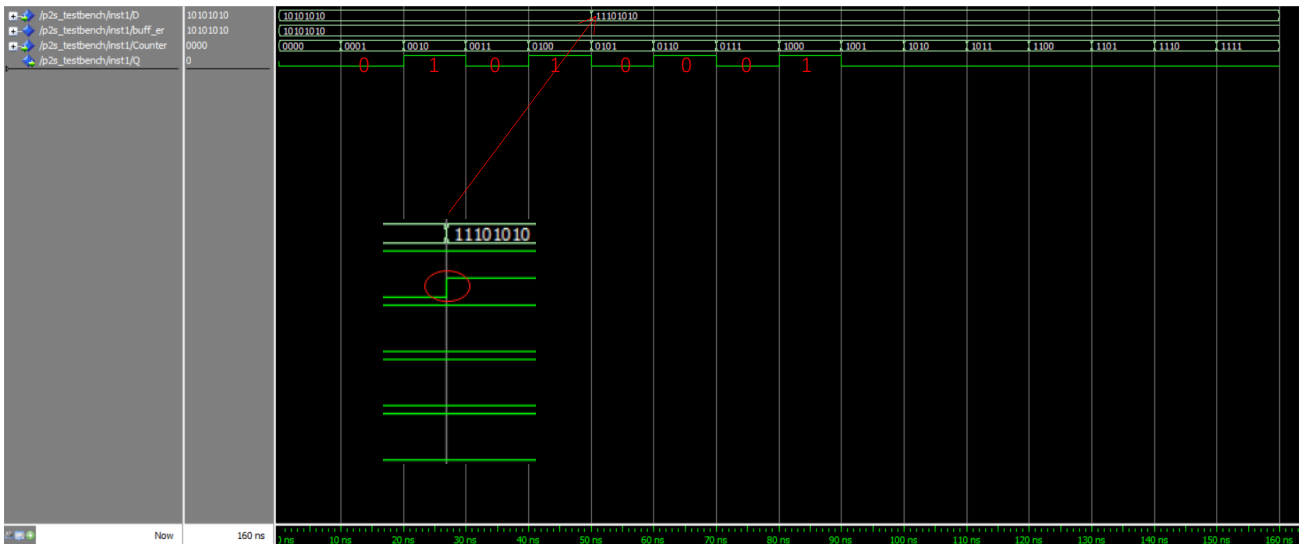

Figure 27 ModelSim of S2P Converter with Buffer

According to the figure 27, the data waiting to received changes its seventh bit from '0' to '1' when "0101", but the result of the serial output is still correct due to having a buffer.

This serialization converter with buffer can be used when the sender lacking the ability to hold the transmitting bits.

7.3 Resources Waste

In section 3.3 and section 4.3, it has mentioned about the resources waste due to empty clock cycle. According to the figure 23, the total transmitting period **only occupies 10 cycles**, leading to the **waste of the rest 6 cycles** from "1010" to "1111". It only makes the use of 5/8 period, which means the next transmitting data needs to wait extra 3/8 period. This inefficient phenomenon should be undermined in the future work.

A multibit memory buffer between serialization and de-serialization converter (stores the data bits from sender until it is filled), **a feedback system** (informs the condition of the buffer) and **asynchronous clock** (controls the serialization and de-serialization converter separately) can be added into this system to settle down the above problem. However, due to the requirement of synchronous clock input, this method may take into consideration in the future work.

7.4 Conclusion of Design Project

In the last section, it has interpreted some flaws of this design and give some suggestions to promote its performance.
In the next two sections, before the end of this report, it will initially give the reference list, and then all the code used in this project for readers who can put theory into practice.

VIII. Reference List

[1] Blom, J. '*Serial Communication*' Journal on. Sparkfun, 2009
Retrieved from
https://learn.sparkfun.com/tutorials/serial-communication/all

[2] Naskar, S., Basuli, K, and Sarma, S. '*SERIAL PORT DATA COMMUNICATION USING MODBUS PROTOCOL*' Journal on. Ubiquity, Vol 9, issue 3, Jan 28, 2008.
Retrieved from
https://ubiquity.acm.org/article.cfm?id=1348477

[3] Tambi, Y. '*Serial Communication*' Journal on. Atml AVR, Sep 3, 2013
Retrieved from
http://maxembedded.com/2013/09/serial-communication-introduction/#ParallelComm

[4] Hou, E., '*ECE 394 Digital Systems Laboratory*', p.7, New Jersey Institute of Technology, 2005
Retrieved from
https://web.njit.edu/~gilhc/ECE394/ECE394-VII.htm

[5] Vahid, F. '*Digital Design with RTL Design, VHDL, and Verilog*', pp 215-218, Wiley, 2011.

[6] BLDRDOC, '*parallel-to-serial conversion*', Aug 23, 1996
Retrieved from
https://www.its.bldrdoc.gov/fs-1037/dir-026/_3840.htm

[7] BLDRDOC, '*serial-to-parallel conversion*', Aug 23, 1996
Retrieved from
https://www.its.bldrdoc.gov/fs-1037/dir-032/_4778.htm

IX. Appendix of Code

```
1   /* Project Name: counter
2      Author: Timo
3      Date: 2018/10/30 */
4
5   library IEEE;
6   use IEEE.STD_LOGIC_1164.all;
7   use IEEE.STD_LOGIC_ARITH.all;
8   use IEEE.STD_LOGIC_UNSIGNED.all;
9
10  entity counter is
11     Port ( clk : in std_logic;                  --input clock signal
12     rst : in std_logic;                         --ON/OFF switch
13     B : out std_logic_vector (3 downto 0));     --counter output
14  end counter;
15
16  architecture Behavioral of counter is
17
18  signal Binout : std_logic_vector (3 downto 0);
19
20  begin
21     --Counter Process--
22     process (clk, rst)   --clk is the input period clock cyle,
23                          --rst is the enable control input
24     begin
25     if rst = '0' then Binout <= "0000"; --when rst is low, counter does not work
26     elsif (clk'event and clk='1') then  --when rst is high, it works
27     Binout <= Binout + 1;               --increase its own calue on each clock cyle
28     end if;
29     end process;
30
31     B (3 downto 0) <= Binout (3 downto 0); --send the final value to output port
32
33  end architecture Behavioral;
```

Click to go back

```vhdl
/* Project Name: Counter_TestBench1
   Author: Timo
   Date: 2018/10/30 */

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity Counter_TestBench1 is
end Counter_TestBench1;

architecture test of Counter_TestBench1 is

component counter is
   Port ( clk : in std_logic;
   rst : in std_logic;
   B : out std_logic_vector (3 downto 0));
end component;
signal clks, rsts : std_logic;
signal Bs: std_logic_vector (3 downto 0);

begin

   inst1 : counter port map (clks, rsts, Bs);
   Process
   begin
   rsts <= '0'; clks <= '0';--1st period, rst,clk='0'
   wait for 10 ns;
   rsts <= '1'; clks <= '1';--2nd period, rst='1' while clk='0'
   wait for 10 ns;
   rsts <= '1'; clks <= '0';--...
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
   rsts <= '1'; clks <= '1';
   wait for 10 ns;
   rsts <= '1'; clks <= '0';
   wait for 10 ns;
```

```
78      rsts <= '1'; clks <= '1';
79      wait for 10 ns;
80      rsts <= '1'; clks <= '0';
81      wait for 10 ns;
82      rsts <= '1'; clks <= '1';
83      wait for 10 ns;
84      rsts <= '1'; clks <= '0';
85      wait for 10 ns;
86      rsts <= '1'; clks <= '1';
87      wait for 10 ns;
88      rsts <= '1'; clks <= '0';
89      wait for 10 ns;
90      rsts <= '1'; clks <= '1';
91      wait for 10 ns;
92      rsts <= '1'; clks <= '0';
93      wait for 10 ns;
94      rsts <= '1'; clks <= '1';
95      wait for 10 ns;
96      rsts <= '1'; clks <= '0';
97      wait for 10 ns;
98      end process;
99
100   end architecture;
```

Click to go back

```
1   /* Project Name: Counter_TestBench
2      Author: Timo
3      Date: 2018/10/30 */
4
5   library IEEE;
6   use IEEE.STD_LOGIC_1164.all;
7   use IEEE.STD_LOGIC_ARITH.all;
8   use IEEE.STD_LOGIC_UNSIGNED.all;
9
10  entity Counter_TestBench is
11  end Counter_TestBench;
12
13  architecture test of Counter_TestBench is
14
15  component counter is
16     Port ( clk : in std_logic;
17     rst : in std_logic;
18     B : out std_logic_vector (3 downto 0));
19  end component;
20  signal clks, rsts : std_logic;
21  signal Bs: std_logic_vector (3 downto 0);
22
23  begin
24
25     inst1 : counter port map (clks, rsts, Bs);
26     Process
27     begin
28     rsts <= '0'; clks <= '0';
29     wait for 10 ns;
30     for i in 1 to 16 loop --do the following statements 16 times
31     clks <= '0'; rsts <= '1';
32     wait for 10 ns;
33     clks <= '1'; rsts <= '1';
34     wait for 10 ns;
35     end loop;
36     end process;
37
38  end architecture;
```

Click to go back

```vhdl
/* Project Name: parallelToSerial converter without buffer
   Author: Timo
   Date: 2018/10/28 */

library ieee;
use ieee.std_logic_1164.all;

entity parallelToSerial is
port(
   D: in std_logic_vector(7 downto 0);
   Counter: in std_logic_vector(3 downto 0);
   Q: out std_logic
);
end parallelToSerial;

architecture Behavior of parallelToSerial is
begin
   with Counter select     --"0000" is for initialization
   Q  <= D(0) when "0001", --"0001" the 1st bit will be transimtted
         D(1) when "0010", --"0010" the 2nd bit will be transimtted
         D(2) when "0011", --"0011" the 3rd bit will be transimtted
         D(3) when "0100", --...
         D(4) when "0101",
         D(5) when "0110",
         D(6) when "0111",
         D(7) when "1000",
         '0' when others;
end architecture;
```

Click to go back

```vhdl
/* Project Name: P2S_TestBench
   Author: Timo
   Date: 2018/10/30 */

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity P2S_TestBench is
end P2S_TestBench;

architecture test of P2S_TestBench is

component parallelToSerial is
   port(
      D: in std_logic_vector(7 downto 0);
      Counter: in std_logic_vector(3 downto 0);
      Q: out std_logic
   );
end component;

   signal Data_input : std_logic_vector(7 downto 0);
   signal Counters : std_logic_vector(3 downto 0) := "1111";
   signal Qs : std_logic;

begin

   inst1 : parallelToSerial port map (Data_input, Counters, Qs);
   Process
   begin
   Data_input <= "00100110";  --the input data is "00100110"
   for i in 1 to 8 loop
   Counters <= Counters + 1;
   wait for 10 ns;
   end loop;
   end process;

end architecture;
```

Click to go back

```
1   /* Project Name: P2S_TestBench_plus
2      Author: Timo
3      Date: 2018/10/30 */
4
5
6   library IEEE;
7   use IEEE.STD_LOGIC_1164.all;
8   use IEEE.STD_LOGIC_ARITH.all;
9   use IEEE.STD_LOGIC_UNSIGNED.all;
10
11  entity P2S_TestBench_plus is
12  end P2S_TestBench_plus;
13
14  architecture test of P2S_TestBench_plus is
15
16  component parallelToSerial is
17     port(
18        D: in std_logic_vector(7 downto 0);
19        Counter: in std_logic_vector(3 downto 0);
20        Q: out std_logic
21     );
22  end component;
23
24     signal Data_input : std_logic_vector(7 downto 0);
25     signal Counters : std_logic_vector(3 downto 0) := "1111";
26     signal Qs : std_logic;
27
28  begin
29
30     inst1 : parallelToSerial port map (Data_input, Counters, Qs);
31     Process
32     begin
33     Data_input <= "10101010";  --the frst input data is "10101010"
34     for i in 1 to 16 loop      --the reason choosing 16 is one total period consitituted
    by small 16 cycles
35     Counters <= Counters + 1;
36     wait for 10 ns;
37     end loop;
38     Data_input <= "01010101";  --the second input data is "01010101"
39     for i in 1 to 16 loop
40     Counters <= Counters + 1;
41     wait for 10 ns;
42     end loop;
43     end process;
44
45  end architecture;
```

Click to go back

```vhdl
/* Project Name: serialToParallel converter
   Author: Timo
   Date: 2018/10/29 */

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity serialToParallel is
port(
   D: in std_logic;
   Counter: in std_logic_vector(3 downto 0);
   Q: out std_logic_vector(7 downto 0)
);
end serialToParallel;

architecture Behavior of serialToParallel is

signal x : std_logic_vector(7 downto 0);  --X will be acted as a buffer to
                                          --store the data temporarily in one period
attribute keep: boolean;                  --keep VHDL synthesis attribute

begin

   process(Counter,D)                     --when either of counter or D changes, the
process will begin
   begin
   if Counter = "0000" then x <= "00000000"; --the first cycle is used for initialization
   elsif Counter = "0001" then x(0) <= D;    --"0001" x(0) will receive and store the 1st
bit
   elsif Counter = "0010" then x(1) <= D;    --"0010" x(1) will receive and store the 2nd
bit
   elsif Counter = "0011" then x(2) <= D;    --"0011" x(1) will receive and store the 3rd
bit
   elsif Counter = "0100" then x(3) <= D;    --...
   elsif Counter = "0101" then x(4) <= D;
   elsif Counter = "0110" then x(5) <= D;
   elsif Counter = "0111" then x(6) <= D;
   elsif Counter = "1000" then x(7) <= D;    --"1000" the last bit has been received
   else null;
   end if;
   end process;

   with Counter select
   Q <= x when "1001",                       --"1001" transmit the data from buffer to
output
   "00000000" when others;                   --export "0000_0000" in other situations

end architecture;
```

Click to go back

```
1   /* Project Name: S2P_TestBench
2      Author: Timo
3      Date: 2018/10/29 */
4
5   library IEEE;
6   use IEEE.STD_LOGIC_1164.all;
7   use IEEE.STD_LOGIC_ARITH.all;
8   use IEEE.STD_LOGIC_UNSIGNED.all;
9
10  entity S2P_TestBench is
11  end S2P_TestBench;
12
13  architecture test of S2P_TestBench is
14
15  component serialToParallel is
16  port(
17     D: in std_logic;
18     Counter: in std_logic_vector(3 downto 0);
19     Q: out std_logic_vector(7 downto 0)
20  );
21  end component;
22  signal Data_input : std_logic;
23  signal Counters : std_logic_vector(3 downto 0);
24  signal Qs : std_logic_vector (7 downto 0);
25
26  begin
27
28     inst1 : serialToParallel port map (Data_input, Counters, Qs);
29     Process
30     begin
31     Data_input <= '0'; Counters <= "0000";       --initialize both serial data input and
    counter
32     wait for 10 ns;
33     Data_input <= '1'; Counters <= Counters + 1; --serial input is '1';counter is "0001"
34     wait for 10 ns;
35     Data_input <= '0'; Counters <= Counters + 1; --serial input is '0';counter is "0010"
36     wait for 10 ns;
37     Data_input <= '1'; Counters <= Counters + 1; --serial input is '1';counter is "0011"
38     wait for 10 ns;
39     Data_input <= '0'; Counters <= Counters + 1; --...
40     wait for 10 ns;
41     Data_input <= '1'; Counters <= Counters + 1;
42     wait for 10 ns;
43     Data_input <= '0'; Counters <= Counters + 1;
44     wait for 10 ns;
45     Data_input <= '1'; Counters <= Counters + 1;
46     wait for 10 ns;
47     Data_input <= '0'; Counters <= Counters + 1;
48     wait for 10 ns;
49     Data_input <= '0'; Counters <= Counters + 1;
50     wait for 10 ns;
51     Data_input <= '0'; Counters <= Counters + 1;
52     wait for 10 ns;
53     Data_input <= '0'; Counters <= Counters + 1;
54     wait for 10 ns;
55     Data_input <= '0'; Counters <= Counters + 1;
56     wait for 10 ns;
57     Data_input <= '0'; Counters <= Counters + 1;
58     wait for 10 ns;
59     Data_input <= '0'; Counters <= Counters + 1;
60     wait for 10 ns;
61     Data_input <= '0'; Counters <= Counters + 1;
62     wait for 10 ns;
63     Data_input <= '0'; Counters <= Counters + 1;
64     wait for 10 ns;
65     Data_input <= '0'; Counters <= Counters + 1;
66     wait for 10 ns;
67     Data_input <= '1'; Counters <= Counters + 1;
68     wait for 10 ns;
69     Data_input <= '0'; Counters <= Counters + 1;
70     wait for 10 ns;
71     Data_input <= '1'; Counters <= Counters + 1;
72     wait for 10 ns;
73     Data_input <= '0'; Counters <= Counters + 1;
74     wait for 10 ns;
75     Data_input <= '1'; Counters <= Counters + 1;
76     wait for 10 ns;
```

```
77       Data_input <= '0'; Counters <= Counters + 1;
78       wait for 10 ns;
79       Data_input <= '1'; Counters <= Counters + 1;
80       wait for 10 ns;
81       Data_input <= '0'; Counters <= Counters + 1;
82       wait for 10 ns;
83       Data_input <= '0'; Counters <= Counters + 1;
84       wait for 10 ns;
85       Data_input <= '0'; Counters <= Counters + 1;
86       wait for 10 ns;
87       Data_input <= '0'; Counters <= Counters + 1;
88       wait for 10 ns;
89       Data_input <= '0'; Counters <= Counters + 1;
90       wait for 10 ns;
91       Data_input <= '0'; Counters <= Counters + 1;
92       wait for 10 ns;
93       Data_input <= '0'; Counters <= Counters + 1;
94       wait for 10 ns;
95       Data_input <= '0'; Counters <= Counters + 1;
96       wait;
97       end process;
98
99    end architecture;
```

Click to go back

```
1   /* Project Name: combination_TestBench
2      Author: Timo
3      Date: 2018/10/30 */
4
5   library IEEE;
6   use IEEE.STD_LOGIC_1164.all;
7   use IEEE.STD_LOGIC_ARITH.all;
8   use IEEE.STD_LOGIC_UNSIGNED.all;
9
10  entity combination_TestBench is
11  end combination_TestBench;
12
13  architecture test of combination_TestBench is
14
15  COMPONENT combination IS
16     PORT
17     (
18        ocs_cll :  IN  std_logic;
19        onoff :  IN  std_logic;
20        input_port :  IN  std_logic_vector (7 DOWNTO 0);
21        seriout_port :  OUT  std_logic;
22        output_port :  OUT  std_logic_vector (7 DOWNTO 0)
23     );
24  END COMPONENT;
25  signal osc_clls, onoffs, seriout_ports : std_logic;
26  signal input_ports, output_ports: std_logic_vector (7 downto 0);
27
28  begin
29
30     inst1 : combination port map (osc_clls, onoffs, input_ports, seriout_ports,
    output_ports);
31     Process
32     begin
33     onoffs <= '0'; osc_clls <= '0';  --initialization
34     wait for 10 ns;
35     input_ports <= "01010101";       --the data waiting to transmit
36     for i in 1 to 16 loop            --one transmitting period
37     onoffs <= '1'; osc_clls <= '1';
38     wait for 10 ns;
39     onoffs <= '1';osc_clls <= '0';
40     wait for 10 ns;
41     end loop;
42     input_ports <= "10101010";       --the data waiting to transmit
43     for i in 1 to 16 loop            --one transmitting period
44     onoffs <= '1'; osc_clls <= '1';
45     wait for 10 ns;
46     onoffs <= '1';osc_clls <= '0';
47     wait for 10 ns;
48     end loop;
49     end process;
50
51  end architecture;
```

Click to go back

```vhdl
/* Project Name: parallelToSerial converter with buffer
   Author: Timo
   Date: 2018/10/30 */

library ieee;
use ieee.std_logic_1164.all;

entity parallelToSerial is
port(
   D: in std_logic_vector(7 downto 0);
   Counter: in std_logic_vector(3 downto 0);
   Q: out std_logic
);
end parallelToSerial;

architecture Behavior of parallelToSerial is
signal buff_er: std_logic_vector(7 downto 0);
begin

   process(Counter)
   begin
   if Counter = "0000" then buff_er <= D; --"0000" buffer will receive and lock the values
   else null;
   end if;
   end process;

   with Counter select           --"0000" is for initialization
   Q  <= buff_er(0) when "0001", --"0001" the 1st bit will be transimtted
         buff_er(1) when "0010", --"0010" the 2nd bit will be transimtted
         buff_er(2) when "0011", --"0011" the 3rd bit will be transimtted
         buff_er(3) when "0100", --...
         buff_er(4) when "0101",
         buff_er(5) when "0110",
         buff_er(6) when "0111",
         buff_er(7) when "1000",
         '0' when others;
end architecture;
```

Click to go back

```vhdl
/* Project Name:P2Swithbuffer_TestBench
   Author: Timo
   Date: 2018/10/30 */

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity P2S_TestBench is
end P2S_TestBench;

architecture test of P2S_TestBench is

component parallelToSerial is
   port(
      D: in std_logic_vector(7 downto 0);
      Counter: in std_logic_vector(3 downto 0);
      Q: out std_logic
   );
end component;

   signal Data_input : std_logic_vector(7 downto 0);
   signal Counters : std_logic_vector(3 downto 0) := "1111";
   signal Qs : std_logic;

begin

   inst1 : parallelToSerial port map (Data_input, Counters, Qs);
   Process
   begin
   Data_input <= "10101010";
   for i in 1 to 5 loop
   Counters <= Counters + 1;
   wait for 10 ns;
   end loop;
   Data_input <= "11101010";
   for i in 1 to 11 loop
   Counters <= Counters + 1;
   wait for 10 ns;
   end loop;
   end process;

end architecture;
```

Click to go back

```
1   /* Project Name: combination
2      Author: Timo
3      Date: 2018/11/3 */
4
5   LIBRARY ieee;
6   USE ieee.std_logic_1164.all;
7
8   LIBRARY work;
9
10  ENTITY combination IS
11     PORT
12     (
13        ocs_cll :  IN  STD_LOGIC;
14        onoff :  IN  STD_LOGIC;
15        input_port :  IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
16        seriout_port :  OUT  STD_LOGIC;
17        output_port :  OUT  STD_LOGIC_VECTOR(7 DOWNTO 0)
18     );
19  END combination;
20
21  ARCHITECTURE bdf_type OF combination IS
22
23  COMPONENT counter
24     PORT(clk : IN STD_LOGIC;
25          rst : IN STD_LOGIC;
26          B : INOUT STD_LOGIC_VECTOR(3 DOWNTO 0)
27     );
28  END COMPONENT;
29
30  COMPONENT parralleltoserial
31     PORT(Counter : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
32          D : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
33          Q : OUT STD_LOGIC
34     );
35  END COMPONENT;
36
37  COMPONENT serialtoparallel
38     PORT(D : IN STD_LOGIC;
39          Counter : IN STD_LOGIC_VECTOR(3 DOWNTO 0);
40          Q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
41     );
42  END COMPONENT;
43
44  SIGNAL   clockcount :  STD_LOGIC_VECTOR(3 DOWNTO 0);
45  SIGNAL   SYNTHESIZED_WIRE_0 :  STD_LOGIC;
46
47
48  BEGIN
49
50  b2v_inst : counter
51  PORT MAP(clk => ocs_cll,
52          rst => onoff,
53          B => clockcount);
54
55
56  b2v_inst2 : parralleltoserial
57  PORT MAP(Counter => clockcount,
58          D => input_port,
59          Q => SYNTHESIZED_WIRE_0);
60
61  PROCESS(SYNTHESIZED_WIRE_0)
62  BEGIN
63  seriout_port <= SYNTHESIZED_WIRE_0;
64  END PROCESS;
65
66  b2v_inst3 : serialtoparallel
67  PORT MAP(D => SYNTHESIZED_WIRE_0,
68          Counter => clockcount,
69          Q => output_port);
70
71
72  END bdf_type;
```

Click to go back

Feedback comments

The structure of your report is excellent. You have used sections and section numbers correctly. You have used figures and figure captions effectively and referenced comprehensively. You have written correctly in the third person. Consequently, your report is easy to follow and of a professional standard. Your design strategies and choices are excellent, and you have provided justification for your choices. Your design is efficient, uses resources sparingly and is easy to debug and service. Your coding style is good. You have used some whitespace, indentation and comments correctly. Your variable names are also fairly meaningful. Your code can be understood and could be debugged, serviced or maintained. Your narrative is excellent. You have provided an introduction and background, you have explained your design choices and narrated your component testing and performance analysis, and you have provided conclusions, limitations and future work. The quality of your written English is excellent, and your report was a pleasure to read.