Implementation of Frequency Divider by Ring Counter with Design Constraint based on FPGA

by

Wenpei Li

B.Eng., Southern University of Science and Technology, 2021

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering

in the Department of Electrical and Computer Engineering

University of Victoria

Supervisory Committee

Implementation of Frequency Divider by Ring Counter with Design Constraint based on FPGA

By

Wenpei Li

B.Eng., Southern University of Science and Technology, 2021

Supervisory Committee

Dr. Mihai Sima (Department of Electrical and Computer Engineering)

Supervisor

Dr. Michael L. McGuire (Department of Electrical and Computer Engineering)

Co-Supervisor

Abstract

Shift registers have been widely used in the field of integrated circuits, which can be used to transport data from a flip-flop to another. A ring counter is composed of a shift register by connecting the input and output of it to form a ring. In this project, a type of frequency divider is implemented with a ring counter to divide the frequency of the clock, and its optimization can be achieved by timing constraints, placement constraints, and routing constraints. In this simulation, Xilinx Vivado is utilized as an optimization tool to refine the timing results of frequency divider, whereas the setup slack time and hold slack time are used as important references. The timing results measure the quality of the design at different stages of adding constraints and other modifications and optimizations are made based on them.

All the constraints are completed by adding constraint files in the source except that the routing design should be done by the router tool in Vivado to choose better net nodes for the critical path. After the modification of all constraints, it shows an improvement in the design timing summary.

Table of Contents

List of Figures

Acknowledgement

I would like to thank my supervisor, Dr. Mihai Sima first for his guidance through the whole program and all his valuable suggestions to my final project and future work.

I would also like to thank my mother for kindly supporting me constantly to help complete my graduate study in Canada, without her courage and love, I can't go this far.

I must also thank my aunt for helping me through my most difficult time.

I would also like to thank my best friends Araki and Sorachi for helping me a lot through my graduation.

I. Introduction

FPGAs were originally invented as a hardware platform to implement simple logic units and small digital circuits. Designers only use schematic tools to draw circuits, and the constraints were barely introduced or even did not exist [1]. Around 1990s, the function of FPGAs became more complex to implement more advanced functions. At that time, the significance of the utilization of design constraint for timing, routing, and placement became apparent [2].

This project focuses on the analysis of design constraint on Xilinx Vivado to optimize the implementation of a frequency divider with ring counter based on FPGA. The design constraint includes timing analysis, placement design, and routing design. According to the implementation result, the constraint is necessary to be added and modified to optimize the whole design [3]. In the project, timing constraint is required and mandatory for the timing analysis, because Vivado will never find out a better solution during the implementation process without specifying the timing constraint. Placement and routing design are both the options for the user to reach a goal of optimization and are highly recommended in such design case.

1.1 Ring Counter

A ring counter is a sequential circuit composed of flip-flops forming a circular shift register with the output of the last flip-flop fed into the first one. A shift register has

the property to shift data stored in the system, and consists of flip-flops that are connected in such a way that the output of one flip-flop can be used as an input to another, depending on the type of shift register created. A shift register is basically a register that is capable of transferring data from one location to another. FPGA registers are usually storage devices that are created by connecting a specific number of D flip-flops in series, and the amount of data (number of bits) that can be stored in the register is always proportional to the number of flip-flops, since each flip-flop can only store one bit at a time. When flip-flops in a register are connected in such a way that the output of one flip-flop becomes the input of another, a shift register is formed[4].

Ring counter uses the structure of a ring shift register to circulate a specific single bit or a group of bits through the ring. It has finite numbers of states to change in response to some inputs. When the circulation is in progress through the ring counter, the circuit outputs a specific cycle level signal which often used in creating finite-state machines. In this project, a ring counter functions as a frequency divider of the primary clock given[5].

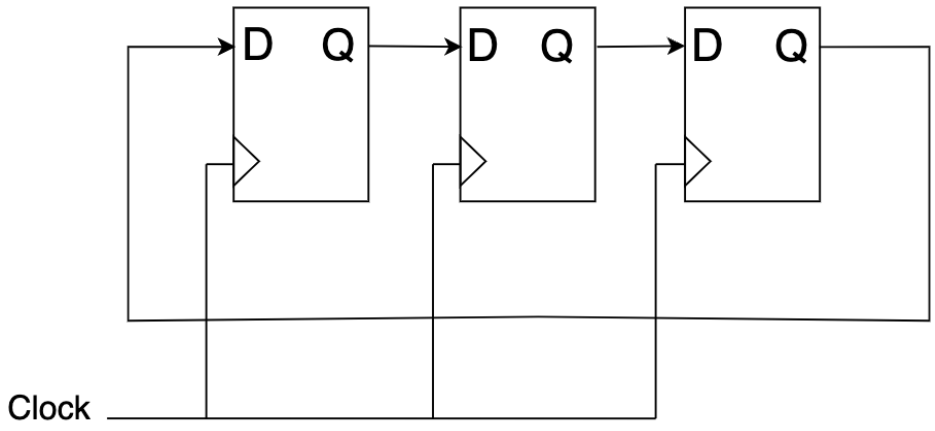


Figure 1.1.1 Ring counter diagram (without synchronous signals)

1.2 Frequency Divider

A frequency divider is a circuit which is also called a clock divider in this digital design. It generates the waveform with certain frequency which divides the frequency of the primary clock by number n (n is related to the number of flip-flops). In the normal design, the generated frequency f is calculated by $f_g = \frac{f}{n}$ (f_g is the output frequency, f is the frequency of the clock, and n is the number of flip-flops). The structure of it is shown in figure 1.2.1 (an example of ring counter with 4 flip-flops).

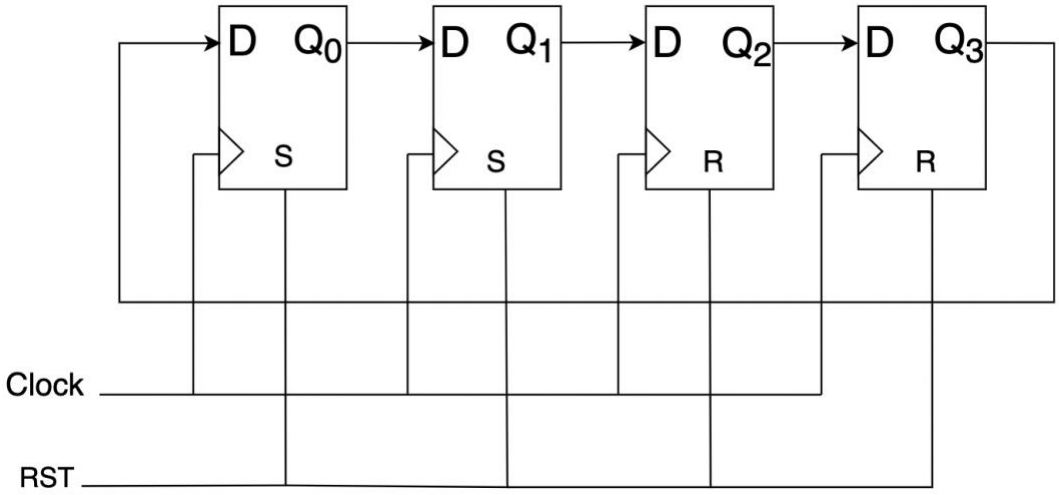


Figure 1.2.1 Normal frequency divider

In the design, ‘RST’ is a synchronous signal connected to each flip-flop while ‘S’ represents ‘SET’ signal to set Q as ‘1’ and ‘R’ is ‘Clear’ signal to set Q as ‘0’. It is defined that ‘RST’ sets the output of every flip-flop as ‘1100’ from Q_0 to Q_3 initially

and then the binary bits start to circulate in the loop. When the output of Q_3 is tested, the waveform generated should be like figure 1.2.2.

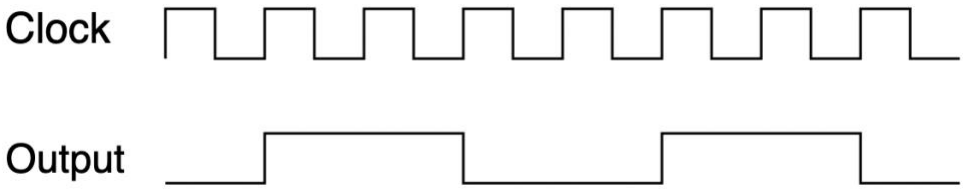


Figure 1.2.2 Output waveform diagram

From the waveform, it is shown that the primary frequency of the clock is divided by 4 which is exactly the number of flip-flops.

In this project, a new design of frequency divider with ring counter is tried, in which a lower frequency is to be obtained with the same number of flip-flops. The circuit diagram is shown as figure 1.2.3 (an example of ring counter with 3 flip-flops which will be the demo circuit in part II) [6].

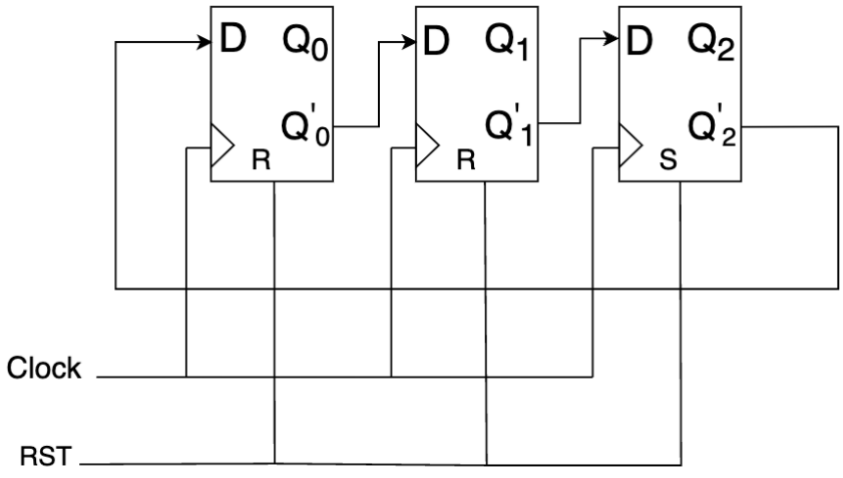


Figure 1.2.3 New design of a frequency divider

In this circuit, Q' represents the opposite level of Q which equals to adding an inverter after the output Q, and every output of Q' is connected to the next input of a flip-flop to form a ring. The initialization of the output of Q' from Q'_0 to Q'_2 is '110'. After the binary bits start to circulate and the output of Q'_2 is tested, the sequences table and waveform generated should be like the way as figure 1.2.4 and figure 1.2.5 present respectively.

Frequency Divider			
State	Q0	Q1	Q2
0	1	1	0
1	1	0	0
2	1	0	1
3	0	0	1
4	0	1	1
5	0	1	0
0	1	1	0
1	1	0	0
2	1	0	1
3	0	0	1
4	0	1	1
5	0	1	0
0	1	1	0
1	1	0	0
2	1	0	1
3	0	0	1
4	0	1	1
5	0	1	0

Figure 1.2.4 Sequences table

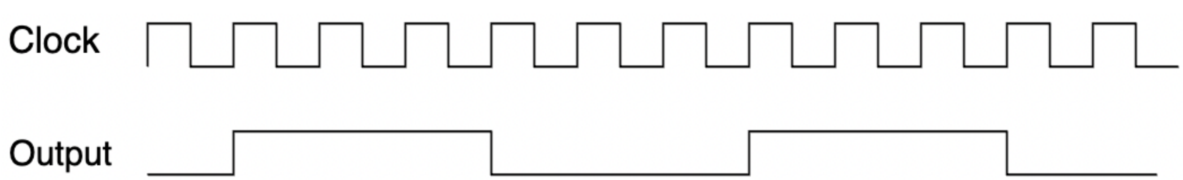


Figure 1.2.5 Output waveform diagram

From the output, it is shown that the new generated frequency can be calculated by $f_g = \frac{f}{2n}$, which is half the frequency of the previous generation for the first frequency divider. In this project, a frequency divider with 5 flip-flops is designed to generate a wave that is 1/10 of the primary clock frequency and the constraint is added to improve its timing results.

Besides, an extra frequency counter module is used connected to the output to ensure if the generated frequency matches the formula.

1.3 Design Constraint

Design constraint is of vital importance in a digital circuit mapped on to FPGA. Here are some main types of constraints:

Timing design constraint specifies the timing requirements for a circuit design and commonly include clock frequency, input/output delay requirements, and maximum allowable clock-to-clock paths etc. These constraints ensure that the target design will meet the timing closure requirements. The results can be used to derive the way the constraint be modified through the timing report given, so that the circuit is optimized for each timing requirement [9].

Placement constraints guide the placement of logic components (such as flip-flops, LUT cells) and other logic blocks within an FPGA device. The specific implementation is done by specifying TCL commands, and the device will have arrays of logic that have been designed and named in regions. These constraints can

specify preferred locations for critical components or enforce physical constraints to optimize signal integrity and performance [3]. There are some useful properties to define a specific location on the layout: ‘BEL’ and ‘LOC’ are the main keywords to determine where a cell should be placed, a reference name of a cell is assigned by Vivado to execute ‘*get_cells*’ command, ‘LUT’ and ‘FF’ are the type names of look-up tables and flip-flops to describe ‘BEL’, and the number of ‘SLICE’ is the basic statement to decide which tile the element should be in. More detailed steps are to be illustrated in part II.

The Vivado router tool performs routing on the design after placement and implements optimization on the routed design to resolve hold time and setup time violations. On the default layout view of the device, if the button ‘Routing Resources’ is turned on, a complex routing layout will be shown. Switch boxes are distributed in various locations where all the wires can use them to connect the signals from other locations and pip junctions are placed around the edges of switch boxes to implement the routing inside the boxes, so the utilization of switch boxes will affect the delays on the paths if there are too many boxes through a specific path or is a congestion inside them. Net nodes are also important in routing design because there are different kinds of nets all over the layout which form many nodes when the nets intersect. In this project, the choice of nodes will be discussed in part II and the routing tool on Vivado will guide the design constraint. In summary, Routing constraint controls the routing of signals between different logic cells and I/O ports of the FPGA. They can define routing routes, specify routing delays, and modify the location of routing resources, which minimizes wire congestion and signal delays [10].

II. Design and Analysis

In this part, the design of frequency divider on FPGA and the timing results will be discussed. The schematic of circuit is to be presented and there will be a testbench added to simulate the code. After the basic design work is finished, constraints are used to improve the performance, and some modifications need to be done according to the potential results.

2.1 Basic Circuit and Code Interpretation

The code for the frequency divider needs to be implemented by the Vivado tool. By adding design source files, the code of different modules will automatically form a complete circuit. After synthesizing successfully, a schematic is displayed. A test file for simulation can also be added to the source files to test whether the circuit outputs the results according to the specified function, and based on the simulated waveforms, we can determine whether there is any error in the code.

2.1.1 Demo Circuit Simulation

In this project, a demo is first designed to make sure the digital design is feasible. It is a frequency divider of three flip-flops with a testbench to test its function. The figure 2.1.1.1 shows the schematic of the demo frequency divider:

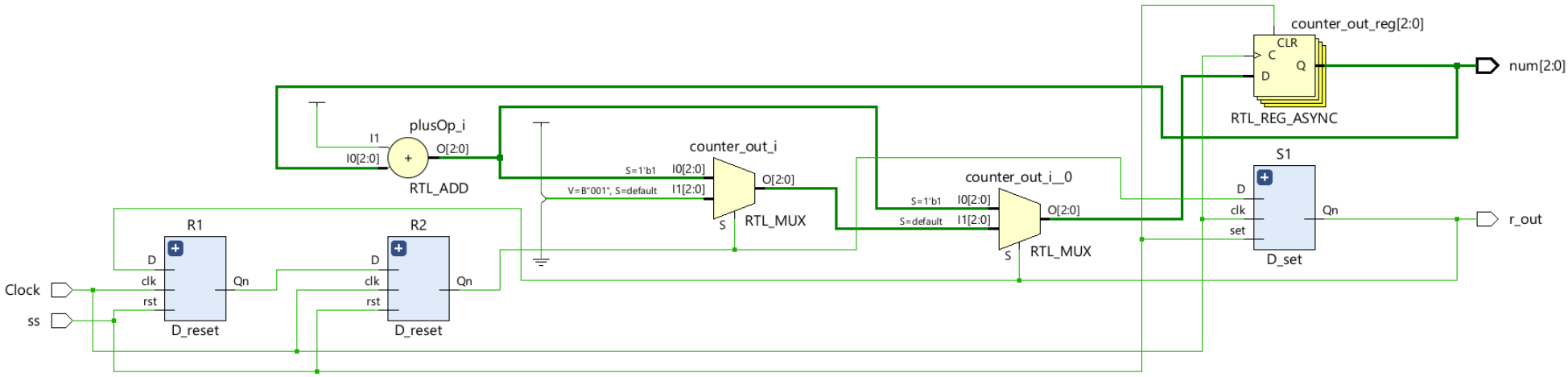


Figure 2.1.1.1 Frequency divider schematic

The main structure is as follows: ‘Clock’ is the general input port of clock, it functions to specify the time for a signal to pass through a D flip-flop from ‘D’ to ‘Q’, ‘ss’ represents the input port of set and reset signals, ‘r_out’ is the output port for the generated wave, and ‘num’ port is the output for the counter connected inside the circuit to calculate the frequency of the generated wave.

The cells inside the circuit are flip-flop modules which we need three of them to implement the circuit: two of them have ‘reset’ signal inputs which would be initialized with binary signal ‘0’while the other one has a ‘set’ signal input which would be initialized with ‘1’. A frequency counter implemented automatically with RTL_ADD, RTL_MUX, and RTL_REG_ASYNC by Vivado. It functions by counting the number of the periods of the output wave within a clock cycle, and then outputting the summed three bits of binary data to the output port. The relative code is presented in figure 2.1.1.2 – 4.

```
D_flip_flop_set : process (clk, set)
    begin
        if (set = '1') then
            Q <= '1'; Qn <= '0';
        elsif (clk'event and clk='1') then
            Q <= D; Qn <= not D;
        end if;
end process;
```

Figure 2.1.1.2 Code of D flip-flop (set)

```
D_flip_flop_reset : process (clk, rst)
    begin
        if (rst = '1') then
            Q <= '0'; Qn <= '1';
        elsif (clk'event and clk='1') then
            Q <= D; Qn <= not D;
        end if;
end process;
```

Figure 2.1.1.3 Code of D flip-flop (reset)

```
frequency_count: process(Clock, ss, Qn1, Qn3)
    begin
    if (ss='1') then
        counter_out <= "000";
    elsif rising_edge(Clock) then
        if (Qn1='1') then
            counter_out <= counter_out + 1;
        elsif (Qn3='1') then
            counter_out <= counter_out + 1;
            else
                counter_out <= "001";
        end if;
     end if;

end process;
```

Figure 2.1.1.4 Code of frequency Counter

A testbench helps to check if the code operates correctly, and a test module file is added in the source to connect the design modules. The test waveform is presented as figure 2.1.1.5, from which a wave with a regular frequency is generated.

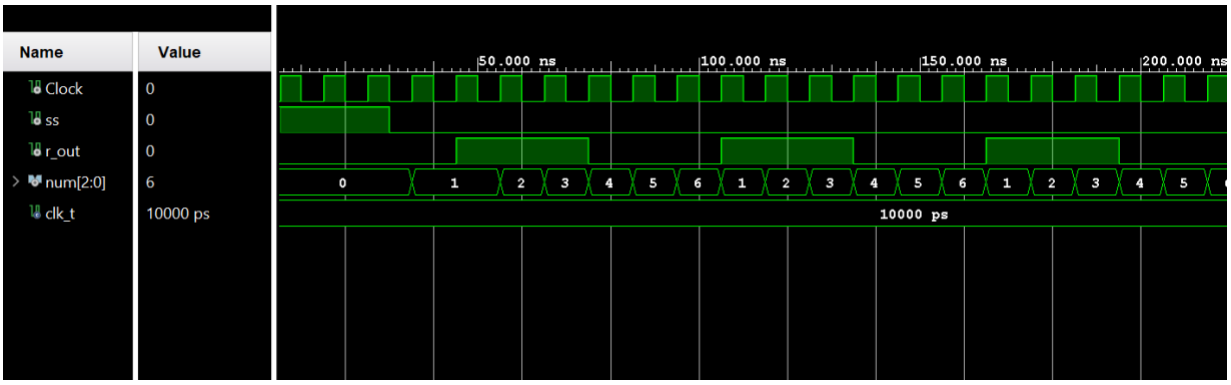


Figure 2.1.1.5 Behavior simulation result

In the simulation result, a clock with the period of 10 ns (name the period as T) is added to test the output. From the result, it can be observed the circuit works correctly that the generated waveform has a stable period of 6T (frequency reduced to 1/6 of the original clock) which is consistent with the original design intent.

2.1.2 Circuit design

After the simulation, a frequency divider of 5 flip-flops is to be designed for implementation. The structure of it is generally the same to the demo one but the synchronous signal set is a little bit different. In this design, the initialized synchronous signals set the output as '01010' from Q_1' to Q_5' before the clock starts.

```
S1: D_set port map (clk => Clock, set => ss, D => Qn5, Q=>Q1, Qn=>Qn1);
R1: D_reset port map (clk => Clock, rst => ss, D => Qn1, Q=>Q2, Qn=>Qn2);
S2: D_set port map (clk => Clock, set => ss, D => Qn2, Q=>Q3, Qn=>Qn3);
R2: D_reset port map (clk => Clock, rst => ss, D => Qn3, Q=>Q4, Qn=>Qn4);
S3: D_set port map (clk => Clock, set => ss, D => Qn4, Q=>Q5, Qn=>Qn5);
```

Figure 2.1.2.1 synchronous signals

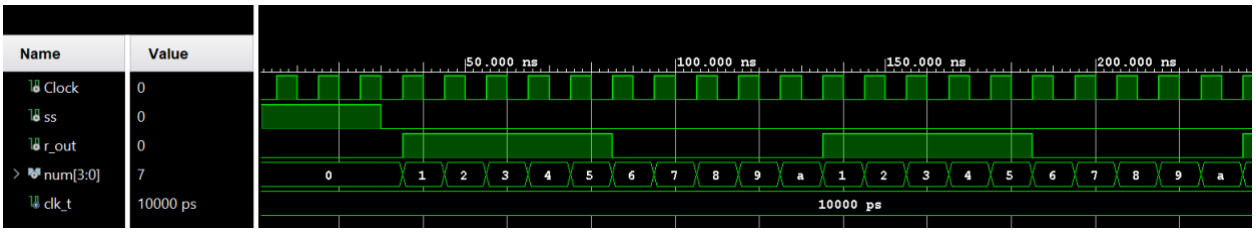


Figure 2.1.2.2 Simulation result

The simulation result is shown as figure 2.1.2.2. The circuit also works correctly to divide the frequency of the original clock by 10 which is twice the number of flip-flops.

After the source code has been added, next step is to synthesize and complete implementation process, by which the schematic diagram and layout diagram on device can be generated. All the locations of cells, wires arrangement and pin choices can be witnessed on the device view, which are automatically generated by Vivado itself in terms of the module layout of the chip.

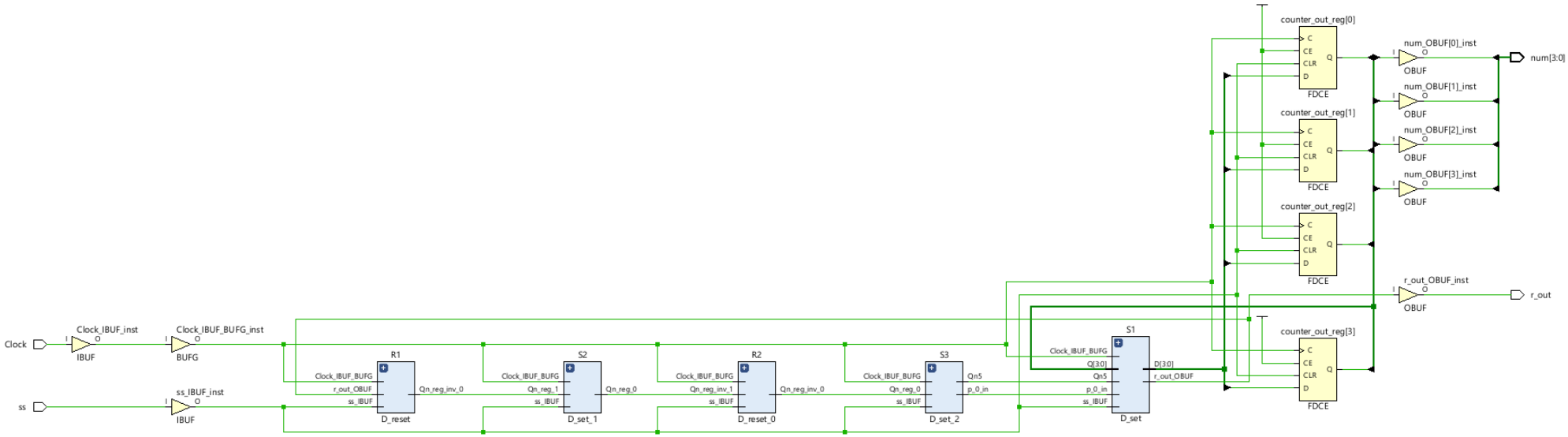


Figure 2.1.2.3 Schematic diagram

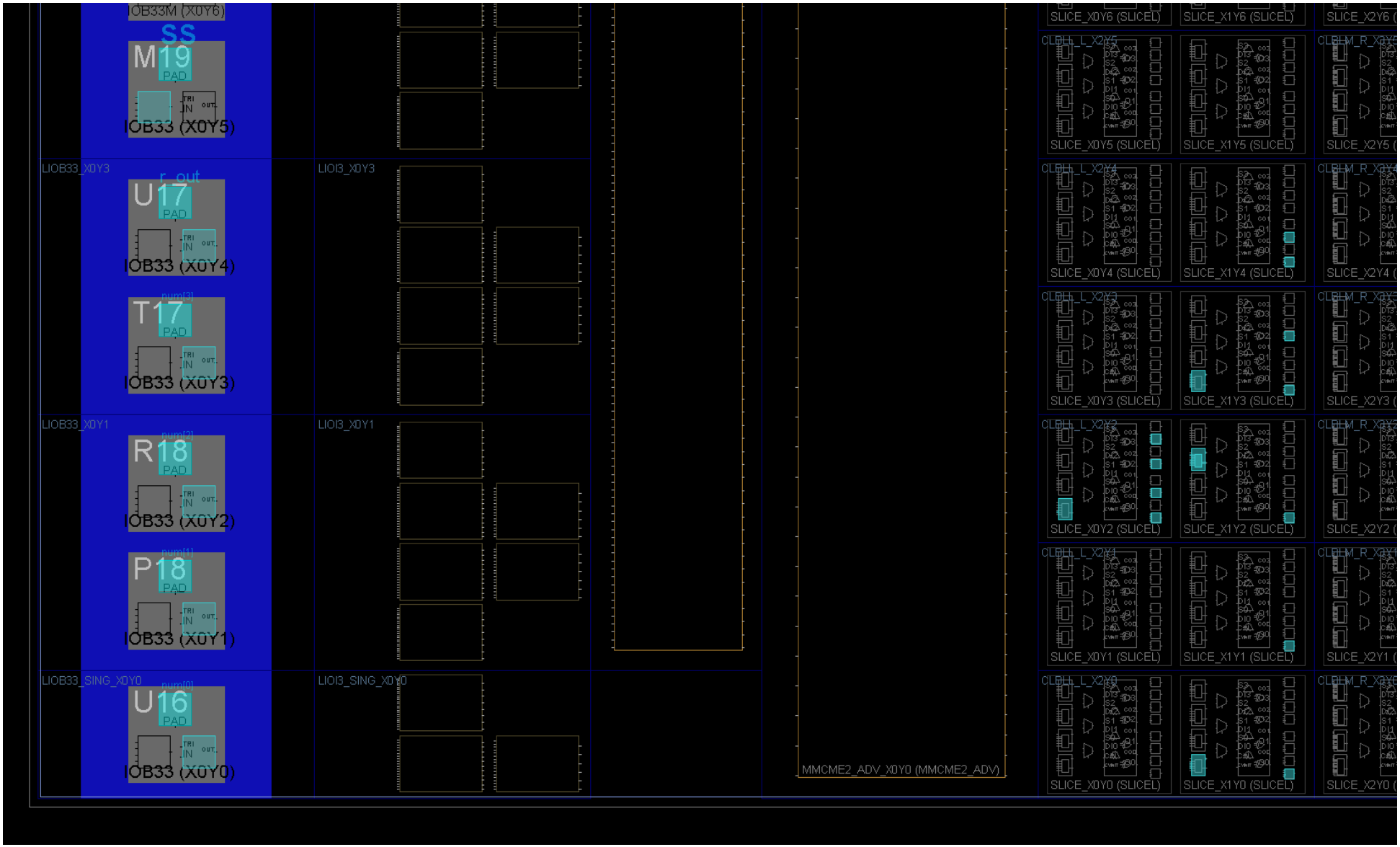


Figure 2.1.2.4 Placement after implementation (part)

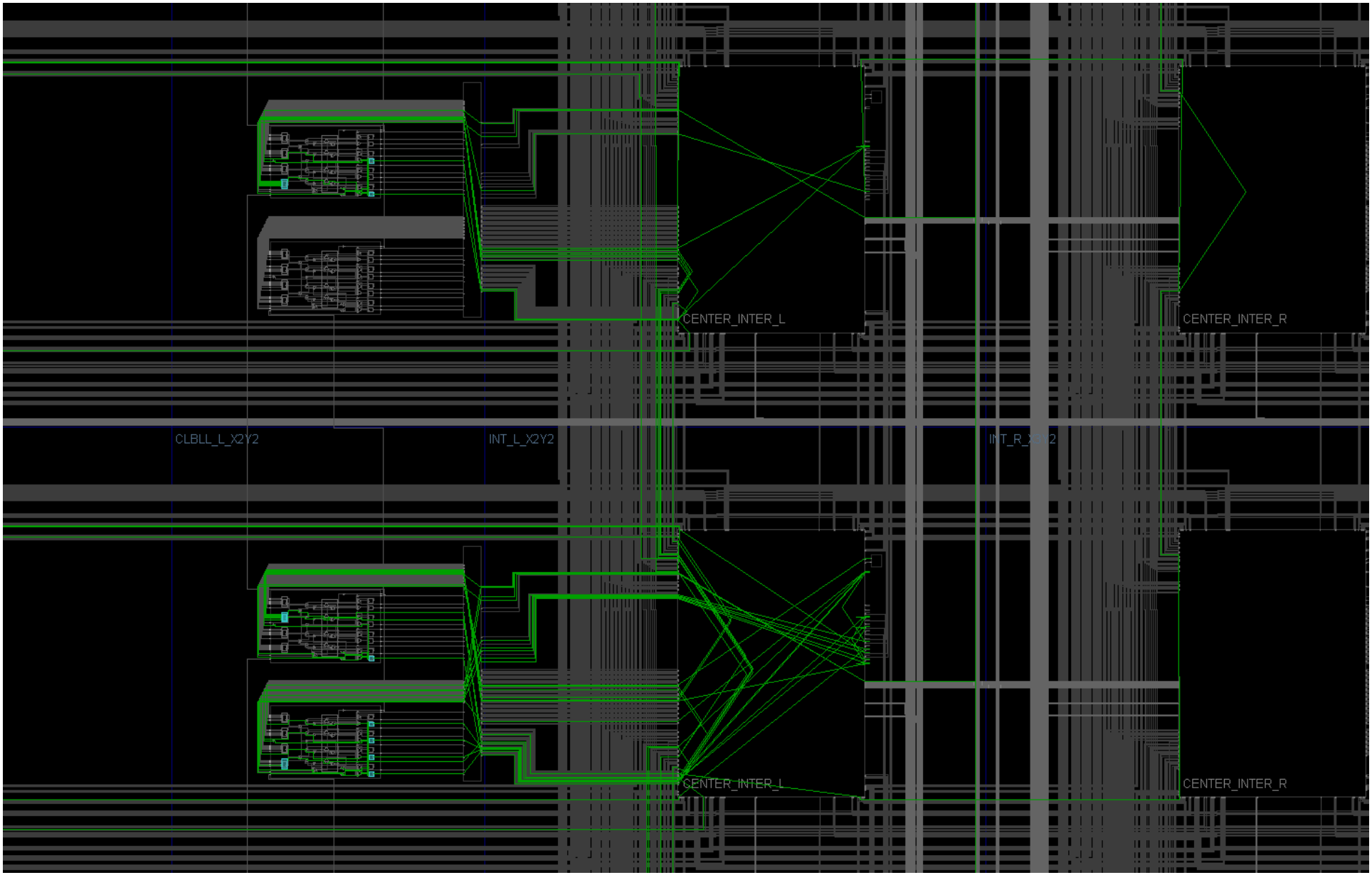


Figure 2.1.2.5 Routing after implementation (part)

Placement is automatically generated which can be seen that all the cells are placed inside one clock region randomly which is a proper way Vivado can find out to place them. The circuit cells are gathered in the zone of slice tiles and the I/O pins are located by the side of device in a column while the clock is placed in another region that can be fixed in the sequential constraint. Part of the routing layout is shown in figure 2.1.2.5, the situation inside each switch box and all the connection of nets can be observed in it. Work needs to be done to implement rational planning of cell locations and routing choices to reduce wire congestion, thereby reducing delay on the critical path and ultimately achieving optimized timing results.

2.2 Timing Design Constraint

In Vivado, which is a popular tool for FPGA design by Xilinx, timing constraints are crucial for ensuring that the design meets the required performance specifications and operates reliably. Timing constraints define the timing requirements for signals within a design. They specify the desired timing characteristics such as clock frequency, setup time, hold time, maximum delay, minimum delay, etc., that the signals must meet for proper operation of the design [3].

2.2.1 Introduction to Timing Constraint

Typically, the timing constraints wizard in Vivado can perfectly guide us to complete the basic timing design. Primary clocks are the first constraint type for timing analysis which enter the design through input ports, it will specify the period, a name, and waveform (rising and falling of the clock signal) to describe the duty

cycle if it is not 50%. The clock signal controls the launch and capture time of those flip-flops connected to the clock buffer. By creating a primary clock on the clock input port, the timer can check the setup and hold slack on the timing path from the input port of a register to the input port of next register [14].

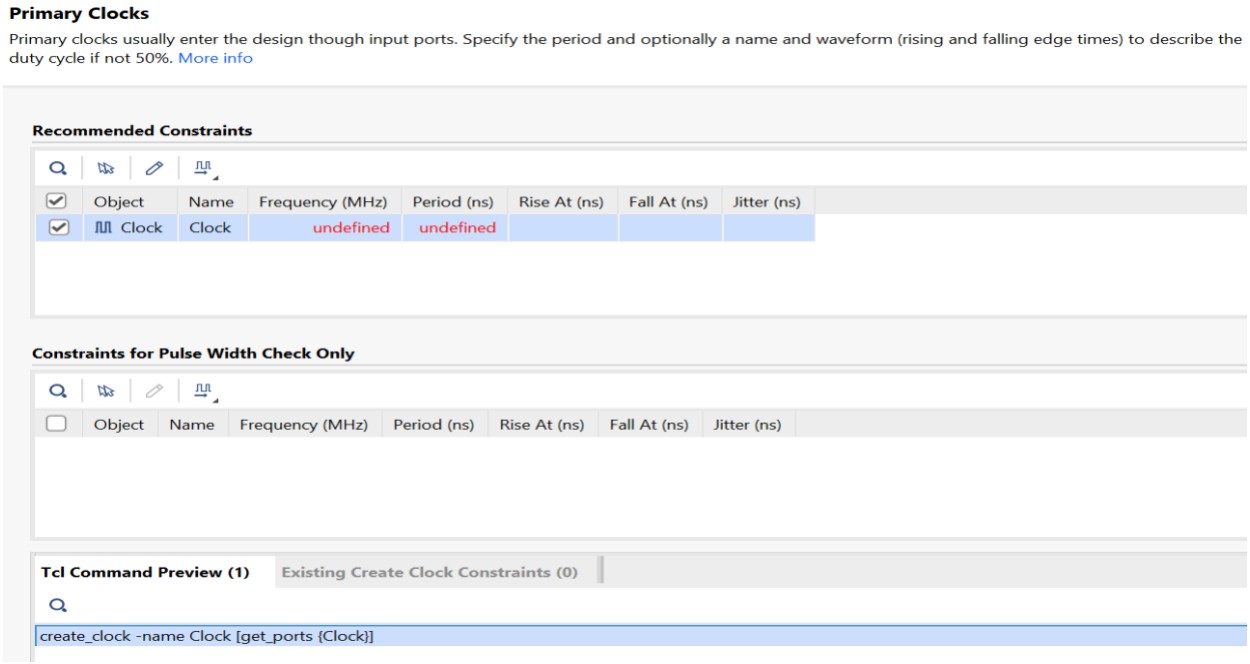


Figure 2.2.1.1 Primary clocks guide

The corresponding directives form for example: *create_clock -period 12.000 -name clk -waveform {0.000 6.000} [get_ports clk]* [3]

Input delays set are necessary in the circuit. It describes the phase between input signals from the FPGA boundary and the reference clocks of the board. If a register has both the clock input signal and another input signal (for example: ‘in’) which will be captured on the rising edge of the clock, then the input delay constraint defines the delay between the clock launch edge and the time the signal ‘in’ transitions at the input of the device. Specifying a minimum input delay ensures that input signals arrive at the designated flip-flops or storage elements after a certain amount of time. This helps avoid setup time violations, where the data at the input

may transition too close to the clock edge, causing potential timing failures. Accordingly, specifying a maximum input delay constraint ensures that input signals remain stable for a certain amount of time after the clock edge. This helps prevent hold time violations, where the data at the input changes too slowly after the clock edge, leading to potential timing failures [9].

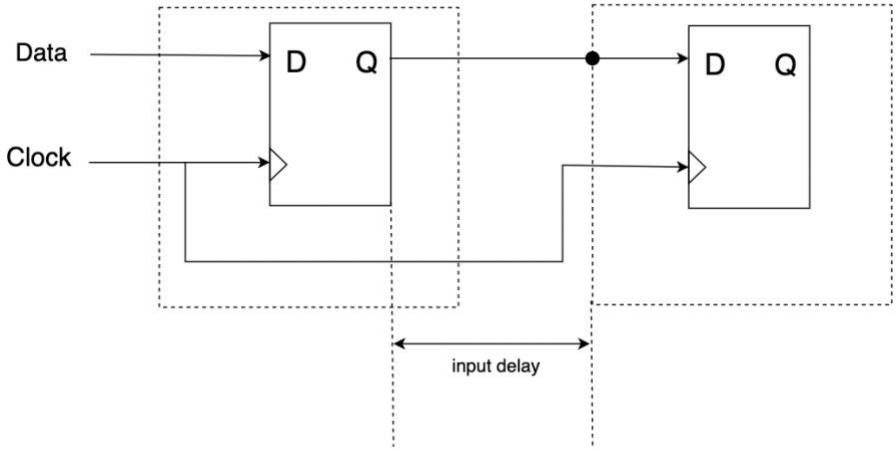
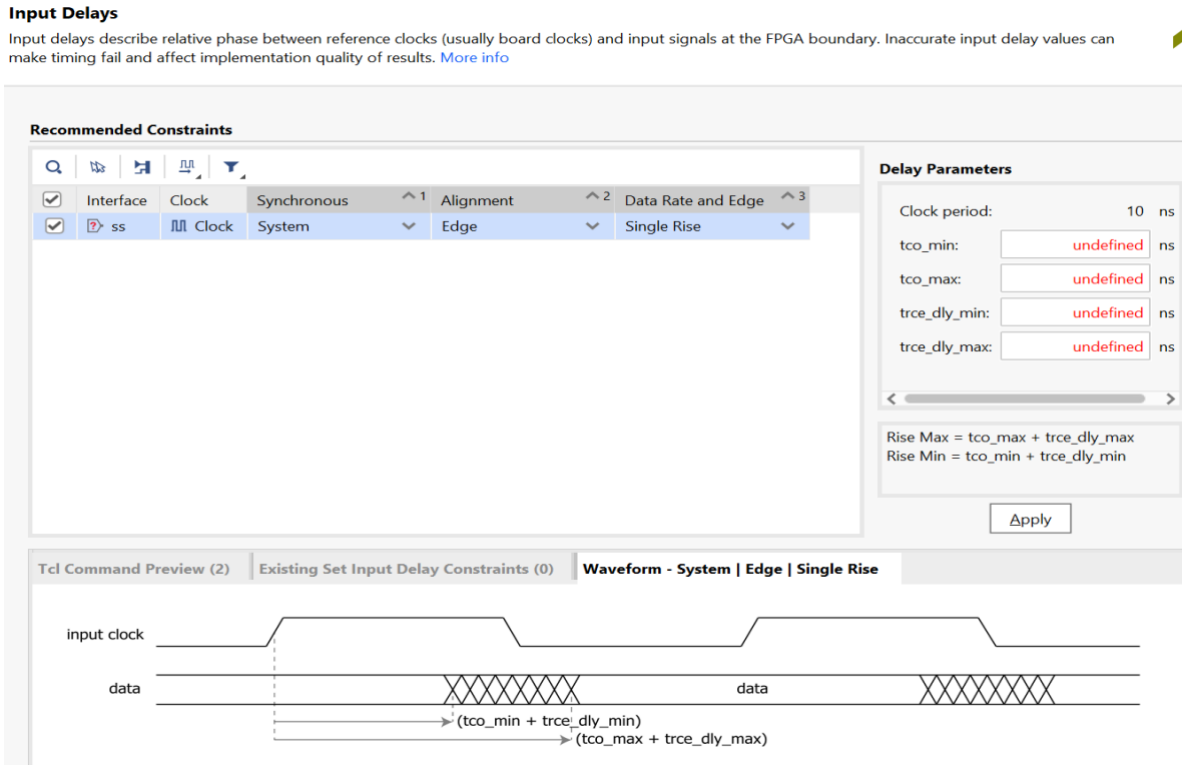


Figure 2.2.1.2 Input delay diagram



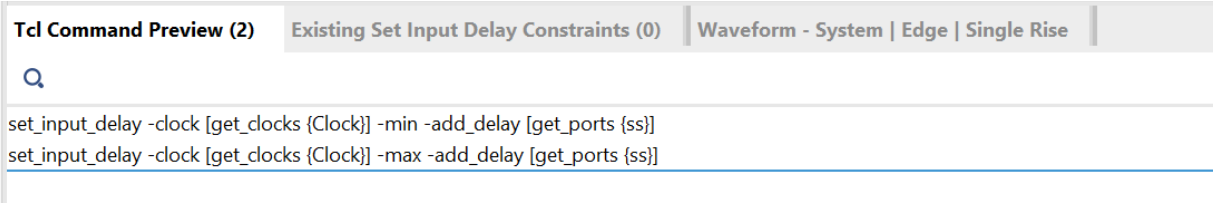


Figure 2.2.1.3 Input delays guide

The corresponding directives form for example: *set_input_delay -clk [get_clocks clk] -min add_delay 1.0 [get_ports in]*
set_input_delay -clk [get_clocks clk] -max -add_delay 2.0 [get_ports in] [3]

Output delays define the relative phase between the output signals at the FPGA boundary and the reference clocks. If the output signal is named as ‘out’ and it is launched by a register, then the output delays describe the delay between the capture edge of the board clock and the time ‘out’ transitions at the boundary of device. Similar to the input delay, output delay time also needs to be specified on both the minimum and maximum [9].

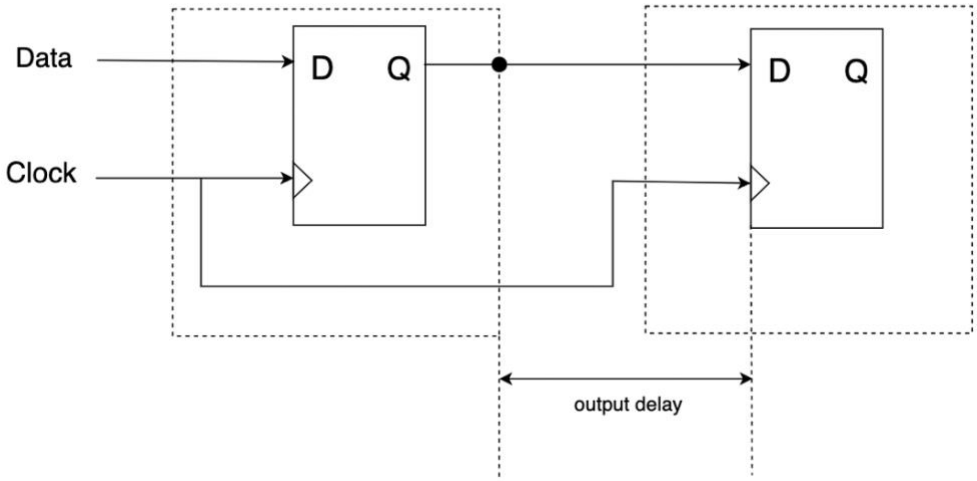


Figure 2.2.1.4 Output delay diagram

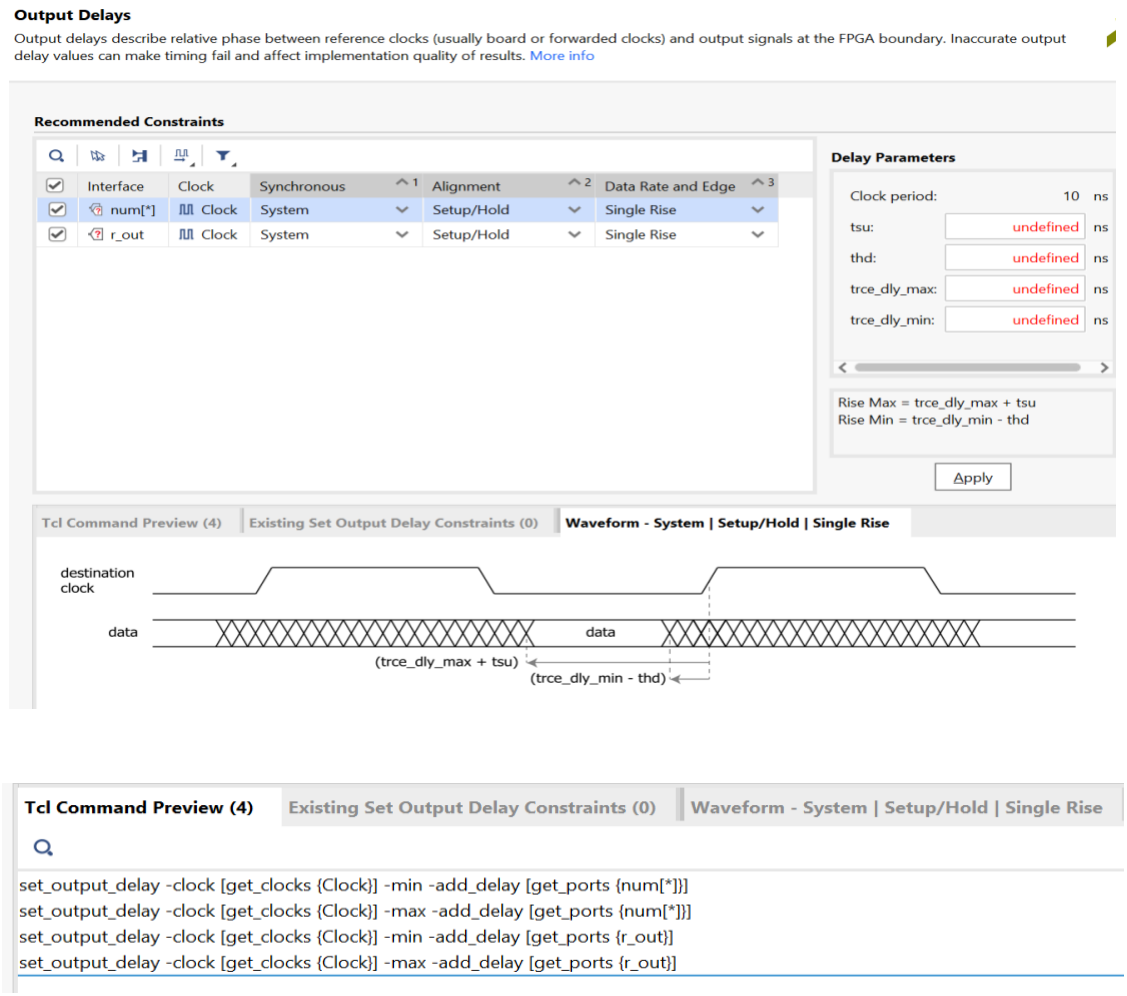


Figure 2.2.1.5 Output delays guide

The corresponding directives format for example: *set_output_delay -clk [get_clocks clk] -min add_delay 1.0 [get_ports out]*
set_output_delay -clk [get_clocks clk] -max -add_delay 2.0 [get_ports out] [3]

There are two significant timing indicators which measure the performance of the running result of the circuit: setup time slack and hold time slack.

Setup time refers to the required amount of time the input signal must be stable before the rising edge of clock arrives to ensure the output signal can be captured

reliably by the next flip-flop while hold time refers to the required amount of time the input signal must be stable after the rising edge of clock arrives [9].

Shown as figure 2.2.1.6, T_{clk1} is the propagation delay from clock to the first flip-flop while T_{clk2} is the delay to the second flip-flop, T_{dq} is the delay time from the clock launch to the output of Q, T_{tran} is the combinational logic delay between two flip-flops, and T_{setup} is the required stable time of data before the rising edge of the clock comes along. T_{cycle} represents a complete clock cycle.

In general, starting from the clock signal, $setup\ slack = data\ required\ time - data\ arrival\ time$. 'Data required time' specifies the maximum time allowed for the arrival of data and 'data arrive time' describes the actual time for the arrival of data. To meet the slack time requirement, $T_{setup\ slack}$ needs to be positive which means the data arrives before the required deadline. According to the given delay times, $data\ required\ time = T_{cycle} + T_{clk2} - T_{setup}$, that is, a new clock cycle plus the wire delay of the clock signal to the secondary flip-flop, and minus the required setup time. $data\ arrival\ time = T_{clk1} + T_{dq} + T_{tran}$, in which the data should wait for a T_{clk1} to let the clock signal arrive the pre-stage flip-flop, then go through it for T_{dq} and finally arrive to the secondary flip-flop [9].

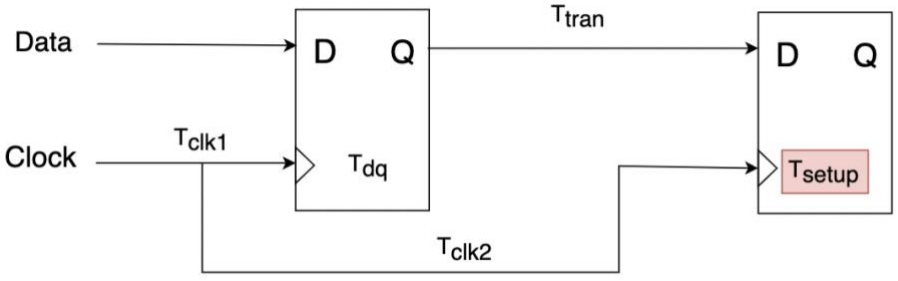


Figure 2.2.1.6 Setup time slack diagram

Similarly, $hold\ slack = data\ finish\ time - required\ finish\ time$. 'Data finish time' describes the actual time the data uses to finish transmission while 'required finish time' is the maximum time allowed for data to finish transmission. T_{hold} represents the time required for data to remain stable after the clock signal comes along. First, $data\ finish\ time = T_{clk1} + T_{dq} + T_{tran} + T_{cycle}$, in which the data waits until the clock signal launches and arrives at the pre-stage flip-flop, then starts to output from Q, goes through the wire delay to the secondary flip-flop, and ultimately waits for a clock cycle. Second, $required\ finish\ time = T_{cycle} + T_{clk2} + T_{hold}$. After signal arrives at the secondary flip-flop, a T_{hold} is necessary to meet the requirement besides a clock cycle. From which it is apparently to be seen that T_{cycle} is irrelevant to hold slack time [9].

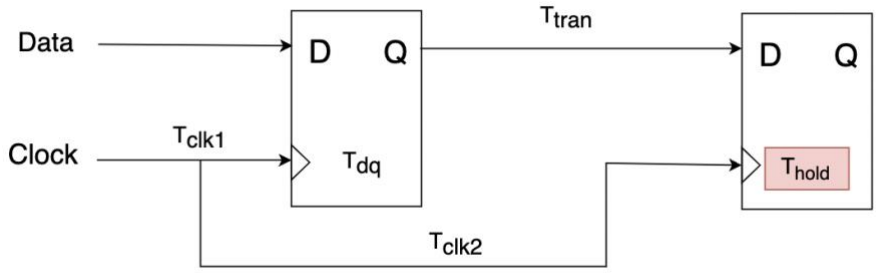


Figure 2.2.1.7 Hold time slack diagram

2.2.2 Timing Design Analysis

In this project, timing constraints are first to be added to test how they work to affect the timing results.

Here are the initial constraints presented:

```
create_clock -period 10.000 -name Clock -waveform {0.000 5.000} [get_ports Clock]
set_input_delay -clock [get_clocks Clock] -min -add_delay 0.000 [get_ports ss]
set_input_delay -clock [get_clocks Clock] -max -add_delay 4.000 [get_ports ss]
set_output_delay -clock [get_clocks Clock] -min -add_delay 0.000 [get_ports {num[*]}]
set_output_delay -clock [get_clocks Clock] -max -add_delay 4.000 [get_ports {num[*]}]
set_output_delay -clock [get_clocks Clock] -min -add_delay 0.000 [get_ports r_out]
set_output_delay -clock [get_clocks Clock] -max -add_delay 4.000 [get_ports r_out]
```

Figure 2.2.2.1 Timing constraints

First, the clock period 10ns is determined and all the delays are set to be 4ns. The results showed a successful hold slack time of 0.162 ns and a failed timing in setup time slack as a positive value is necessary for the worst negative slack to implement a successful timing.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-2.760 ns	Worst Hold Slack (WHS):	0.162 ns	Worst Pulse Width Slack (WPWS):	4.650 ns
Total Negative Slack (TNS):	-13.383 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	5	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	27	Total Number of Endpoints:	27	Total Number of Endpoints:	12

Timing constraints are not met.

Figure 2.2.2.2 Design timing summary

Intra-Clock Paths - Clock - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement
Path 1	-2.760	1	1	counter_out_reg[2]_replica/C	num[2]	3.964	2.715	1.249	10.0
Path 2	-2.704	1	5	counter_out_reg[0]/C	num[0]	3.907	2.703	1.204	10.0
Path 3	-2.666	1	4	counter_out_reg[1]/C	num[1]	3.868	2.720	1.149	10.0
Path 4	-2.632	1	2	counter_out_reg[3]/C	num[3]	3.836	2.688	1.148	10.0
Path 5	-2.620	1	1	S1/Qn_reg_lopt_replica/C	r_out	3.824	2.686	1.137	10.0

Figure 2.2.2.3 Failed critical paths

From the timing result of critical paths, the setup slack is affected by excessive delay time of output, so now we try to lower the maximum value to 3ns.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	-1.760 ns	Worst Hold Slack (WHS):	0.162 ns	Worst Pulse Width Slack (WPWS):	4.650 ns
Total Negative Slack (TNS):	-8.383 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	5	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	27	Total Number of Endpoints:	27	Total Number of Endpoints:	12

Timing constraints are not met.

Figure 2.2.2.4 New timing summary

From which, an apparent improvement is shown on the worst negative slack, but the timing constraints are still not met. The next step is to test the maximum extent to which we can optimize by modifying the maximum output delay.

In the process of lowering the maximum output delay, the setup and hold slack both vary with it. From the diagram of running result below, the hold time slack is always stable around 0.16ns, and lower than 1ns around is better for the choice of maximum

output delay time value to obtain proper hold and setup time slack for the worst critical path.

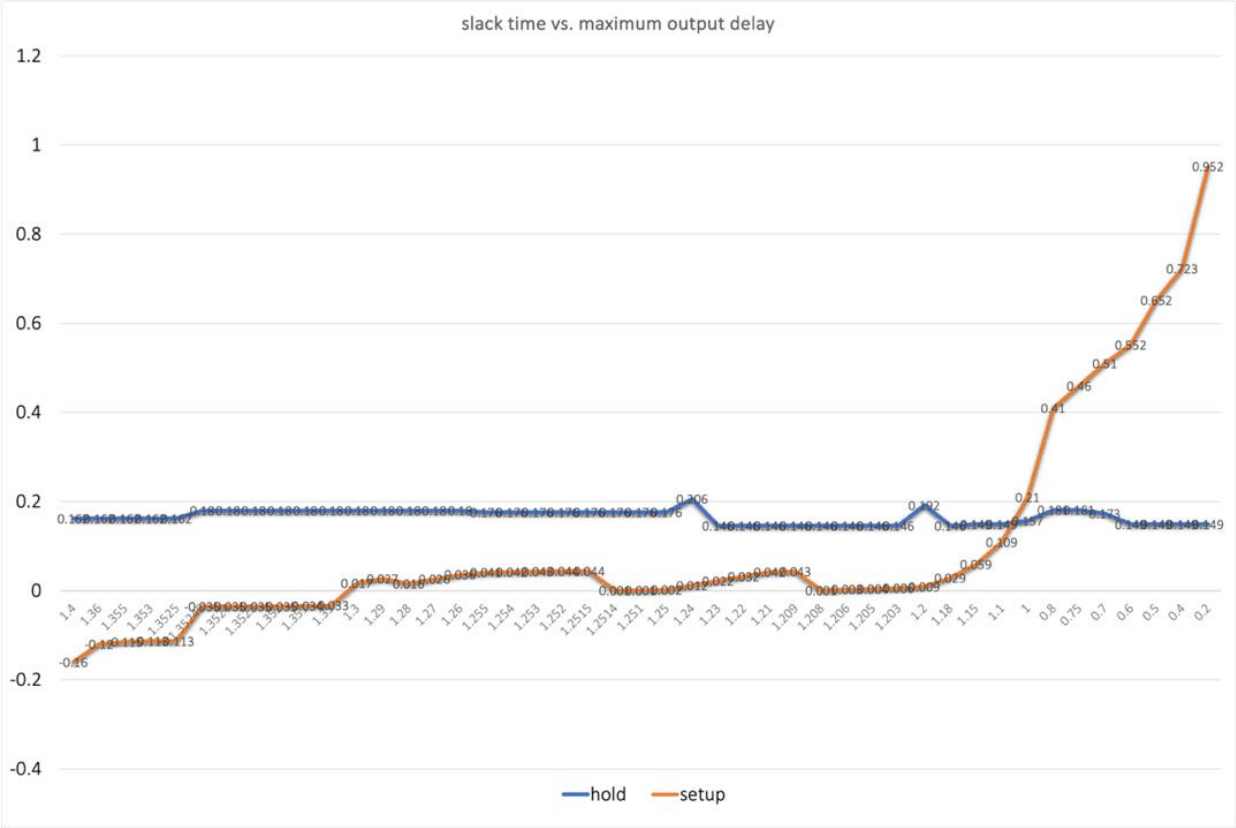


Figure 2.2.2.5 Timing variation diagram

After selecting 0.75ns as the timing constraint for maximum output delay, the result after implementation is as figure 2.2.2.6 shows.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.460 ns	Worst Hold Slack (WHS):	0.181 ns	Worst Pulse Width Slack (WPWS):	4.600 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	25	Total Number of Endpoints:	25	Total Number of Endpoints:	11

All user specified timing constraints are met.

Figure 2.2.2.6 Final design timing summary

2.3 Placement Design Constraint

Placement constraints are directives provided to the Vivado tool during the implementation phase of FPGA design. These constraints guide the tool in deciding where to place specific elements of the design, such as logic cells, I/O pins, and memory blocks, on the physical FPGA device [3].

2.3.1 Introduction to Placement Constraint

In general, placement is a way to rearrange the logic cells on the device to optimize the timing results. There are several steps to modify the placement for a circuit.

First, a placement constraint file is required to be added into the project sources. One thing to keep in mind is that all the constraint files can be selected to be executed in either synthesis phase or implementation phase, and in the case of placement design, only the box of implementation should be checked because the placement behavior only relates to the process after netlist generation: Placement constraints are inherently related to the physical layout of the design on the FPGA device, which is determined during implementation while synthesis phase deals with logical representations and optimizations, whereas implementation deals with physical aspects such as placement and routing [12].

Second, the divisions and concepts of the modules and areas on the device must be clarified first: On a targeted device, a basic unused logic cell on it has many properties to locate it. ‘Name’ defines the cell type or the position number on FPGA. The types include multiplexer, LUT (Look-up Table), flip-flop, other logic gates etc.

‘BEL’ stands for basic element label which describes the type of a specific logic element (for example: ‘A6LUT’ means a particular LUT cell). ‘SLICE’ is a repeated array zone contains the combination of different types of logic cells and each slice is named by its position as there are rows and columns numbered on the device according to location (for example: ‘SLICE_X38Y82’ is a slice on column 38 and row 82). The site of slice is the key information to specify the location of a logic cell. ‘Tile’ is a module which contains two slices right next to each other and it has the same naming convention as ‘SLICE’ does. Pins are also able to be assigned manually for logic units if it is necessary to try different options instead of the choice of Vivado.

Placement design constraint has many operations of directives which are to be demonstrated as below:

Logic cells placement: If a flip-flop named ’counter_out_reg [1]’ is placed on the slice ‘SLICE_X0Y3’ of labeled type ‘AFF’, then it is shown to be like figure 2.3.1.1:

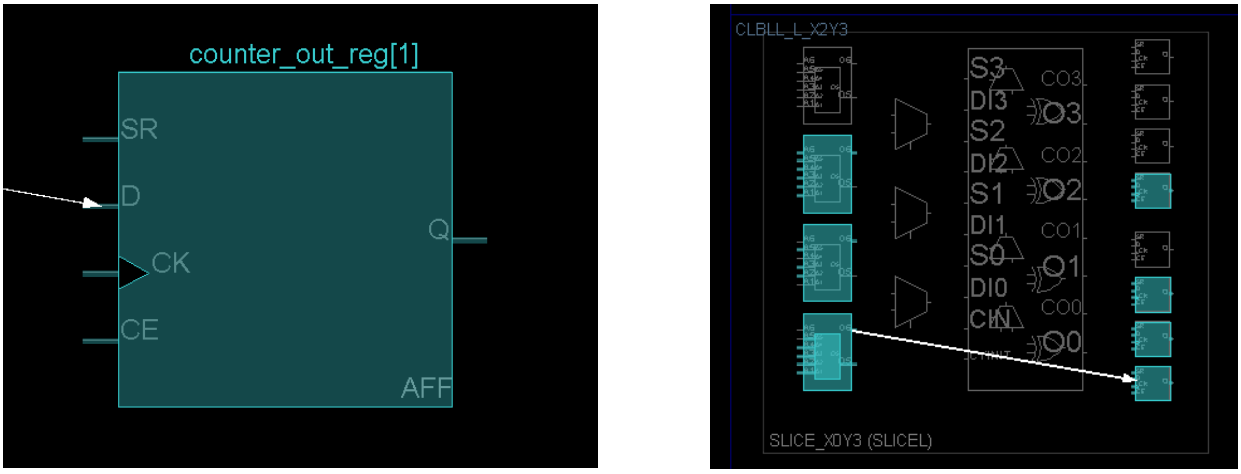


Figure 2.3.1.1 (a) Former Element label (left); (b) Former slice site (right)

Now a placement is done to locate it on a new slice ‘SLICE_X1Y3’:

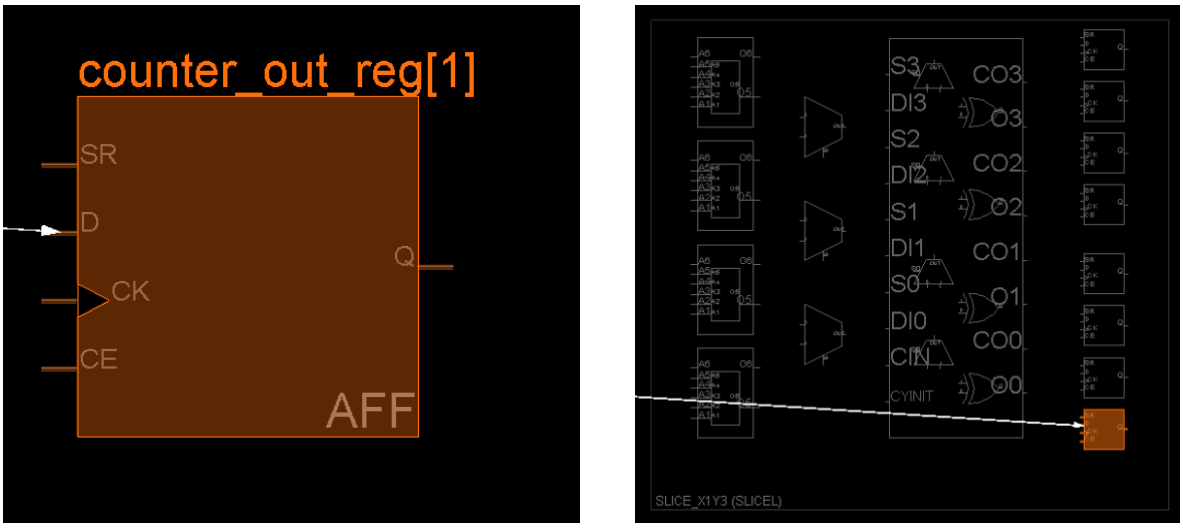


Figure 2.3.1.2 (a) Element label (left); (b) New slice site (right)

Corresponding directives: *set_property LOC SLICE_X1Y3 [get_cells {counter_out_reg [1]}]*
set_property BEL AFF [get_cells {counter_out_reg [1]}] [3]

The reason why the color of the cell turns orange is that the cell is fixed now which means any other changes to the layout shall not affect the existing position of it until it is cancelled in the cell properties window.

Clock/I/O ports placement: If a clock port and an I/O port needs to be relocated to slice ‘U6’, the directives are slightly different, and it is presented as below in the layout:

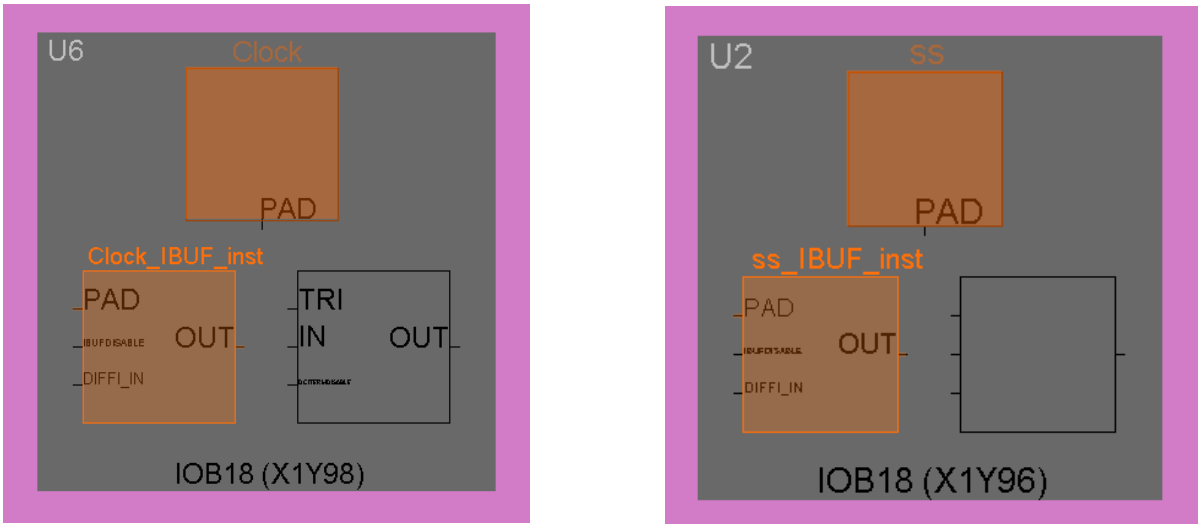
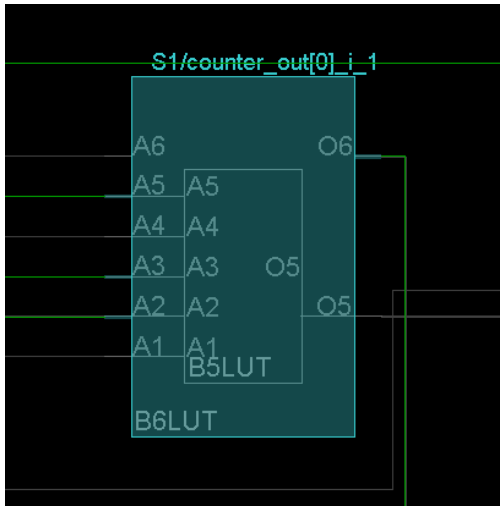


Figure 2.3.1.3 (a) New slice site (clock)(left); (b) New slice site (output pin) (right)

Corresponding directives: *set_property PACKAGE_PIN U6 [get_ports Clock]*
Set_property PACKAGE_PIN U2 [get_ports ss] [3]

Within the above placement, clock has its own requirement that if a relocation happens to a clock port, the system will warn that it is a dedicated routing for the signal which results in errors. In this situation, directive '*set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clock_IBUF]*' needs to be added to the constraints to instruct Vivado not to dedicate routing resources exclusively for clock signals. This constraint essentially allows clock signals to share routing resources with other signals, rather than having dedicated routing paths. Sharing means allowing multiple signals within an FPGA design to use the same physical routing pathways (include metal tracks, wires, switches and other physical structures) on the FPGA chip [3].

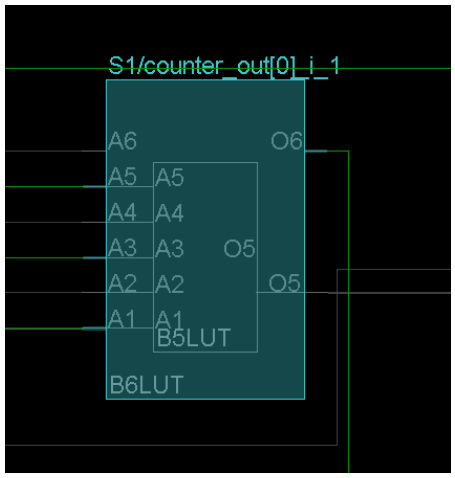
The last placement type in this project is pin choice of logic elements (especially LUTs). The meaning for selecting different pins is to influence the routing paths and reduce routing congestion which may improve signal integrity and timing closure. By changing the used pin of a LUT in the constraints, we can optimize the placement of the LUT within the FPGA device to improve performance and meet design requirements. Here is an example for a cell 'S1/counter_out [0] _i_1' in figure 2.3.1.4 and 2.3.1.5:



counter_out[0]_i_1

Name	Dir	BEL Pin	Net
I0	Input	A5	S1/r_out_OBUF
I1	Input	A3	S1/Q[0]
I2	Input	A2	S1/Qn5
O	Output	O6	S1/D[0]

Figure 2.3.1.4 (a) Initial used pins layout (left); (b) Cell pins table (right)



counter_out[0]_i_1

Name	Dir	BEL Pin	Net
I0	Input	A5	S1/r_out_OBUF
I1	Input	A3	S1/Q[0]
I2	Input	A1	S1/Qn5
O	Output	O6	S1/D[0]

Figure 2.3.1.5 (a) Reselected pins layout (left); (b) Cell pins table (right)

Corresponding directive: *set_property LOCK_PINS {I0:A5 I1:A3 I2:A1} [get_cells {S1/counter_out [0] _i_1}]* [3]

The three pins can also be specified separately in three lines.

2.3.2 Placement Design Analysis

Placement design for the circuit is necessary due to some long wire routing between different cells which excessive delays derive from it. The purpose of replacement is to minimize the physical distance between interconnected units to lower delay time on the wires.

As there are various types of units located on the device, the work is supposed to be undertaken part by part. First, some clock cells are fixed like ‘Clock_IBUF_BUFG_inst’ because the tile region for ‘BUFGCTRL’ provided by the device is fixed and rare, therefore all the placement of other cells are to be near this part.

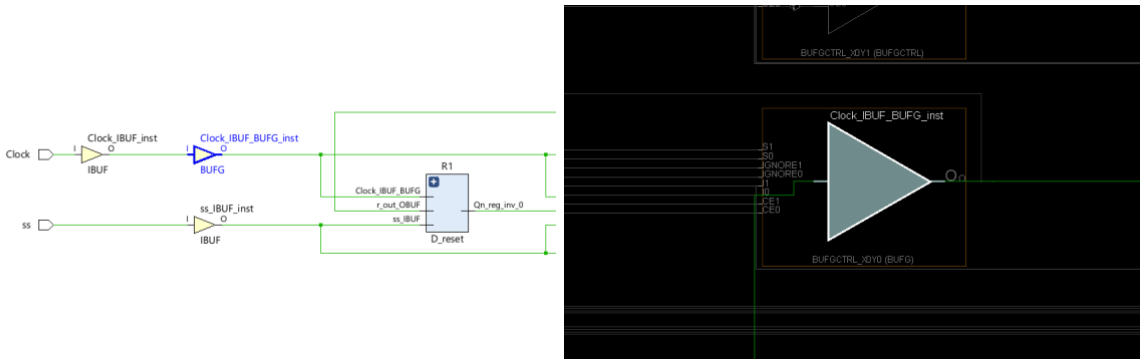


Figure 2.3.2.1 ‘Clock_IBUF_BUFG_inst’ in schematic and on device

The clock region is relocated to ‘X1Y1’ and the permission of changing the location of the clock is supposed to be allowed. The constraints statement is as below.

```
#Clock placement
set_property LOC U6 [get_cells Clock_IBUF_inst]
set_property PACKAGE_PIN U6 [get_ports Clock]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets Clock_IBUF]

set_property BEL BUFG [get_cells Clock_IBUF_BUFG_inst]
set_property LOC BUFGCTRL_X0Y2 [get_cells Clock_IBUF_BUFG_inst]
```

Figure 2.3.2.2 Clock region constraints [13]

Second, I/O ports are relocated to one side of the clock region area as the clock signal does.

```
#I/O Ports placement
set_property LOC W3 [get_cells {num_OBUF[0]_inst}]
set_property PACKAGE_PIN W3 [get_ports {num[0]}]
set_property LOC V3 [get_cells {num_OBUF[1]_inst}]
set_property PACKAGE_PIN V3 [get_ports {num[1]}]
```

Figure 2.3.2.3 I/O ports constraints (part)

Third, flip-flops are placed as a ring and they are all connected to the input ports, so it is better to relocate them in order as the schematic presents and try to make them close to input ports.

```
#flip-flops
set_property BEL A5FF [get_cells R1/Qn_reg_inv]
set_property LOC SLICE_X51Y94 [get_cells R1/Qn_reg_inv]
set_property LOC SLICE_X52Y94 [get_cells S2/Qn_reg]
set_property LOC SLICE_X53Y94 [get_cells R2/Qn_reg_inv]
set_property LOC SLICE_X54Y94 [get_cells S3/Qn_reg]
```

Figure 2.3.2.4 Flip-flops constraints (part)

The rest of units are all related to the output ports and may affect the delay slack in the timing analysis reports, so trying to relocate them near the I/O ports column in a proper position is vital.

```
#Cell placement__counter
set_property BEL A5FF [get_cells {counter_out_reg[0]}]
set_property LOC SLICE_X57Y91 [get_cells {counter_out_reg[0]}]
set_property LOC SLICE_X57Y92 [get_cells {counter_out_reg[1]}]
set_property LOC SLICE_X57Y93 [get_cells {counter_out_reg[2]}]
```

Figure 2.3.2.5 Cells constraints (part)

After implementation, the new layout is shown as below.

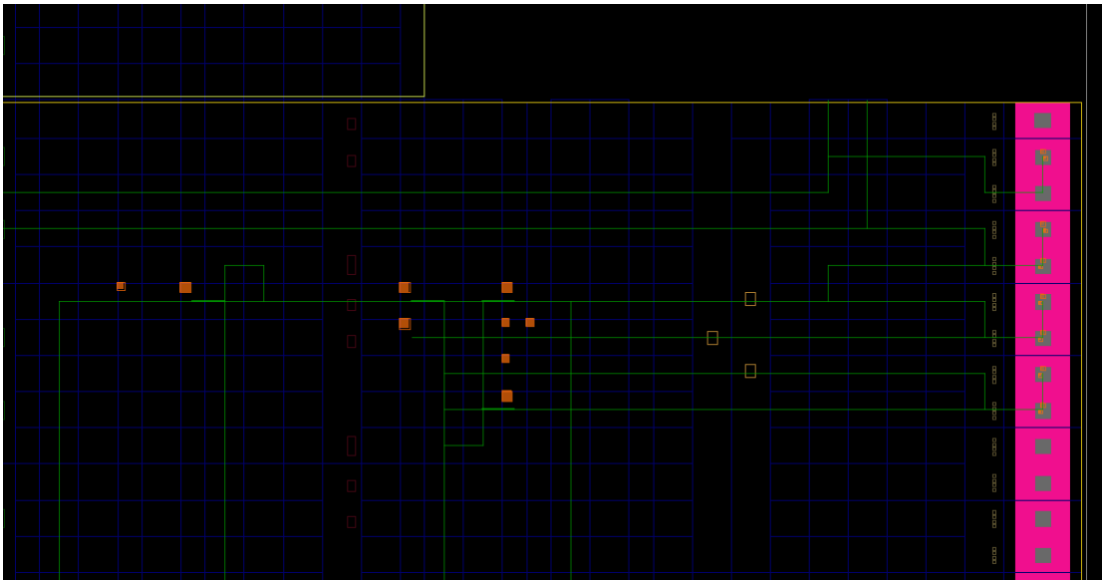
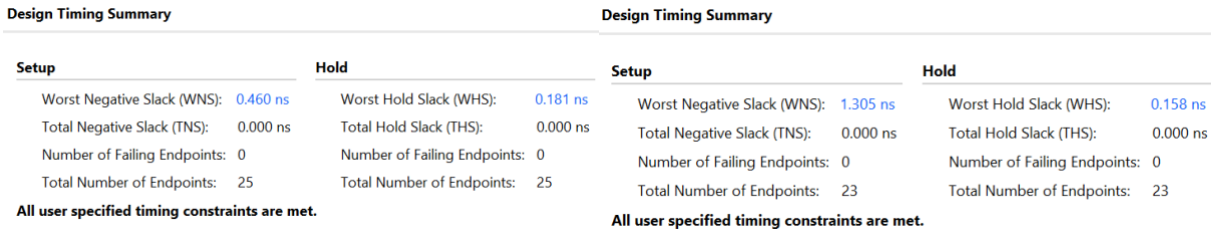


Figure 2.3.2.6 New placement layout

Design timing summary presents an optimization on worst negative slack, and figure 2.3.2.7 shows a comparison of timing results before and after adding placement constraints with the same timing constraints.



Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	0.460 ns	Worst Hold Slack (WHS):	0.181 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	25	Total Number of Endpoints:	25

All user specified timing constraints are met.

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	1.305 ns	Worst Hold Slack (WHS):	0.158 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	23	Total Number of Endpoints:	23

All user specified timing constraints are met.

Figure 2.3.2.7 Comparison of timing results

From which it is shown that the worst hold slack has decreased due to excessive switch boxes in the critical path of the wires. To increase the hold slack, a modification of placement for the cells in the path needs to be completed. After reducing the number of switch boxes to one, the placement and routing of them are shown in figure 2.3.2.8 and 2.3.2.9 respectively, and the final implementation results are shown in figure 2.3.2.10. The worst hold slack has increased back to 0.185ns.

Name	Slack ^1	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay
Path 11	0.185	0	1	R2/Qn_reg_inv/C	S3/Qn_reg/D	0.196	0.091	0.105

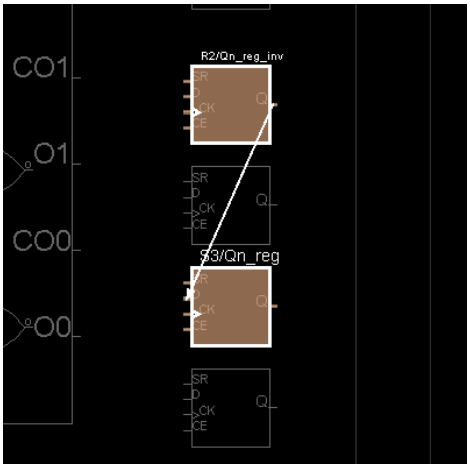


Figure 2.3.2.8 Critical path after modification

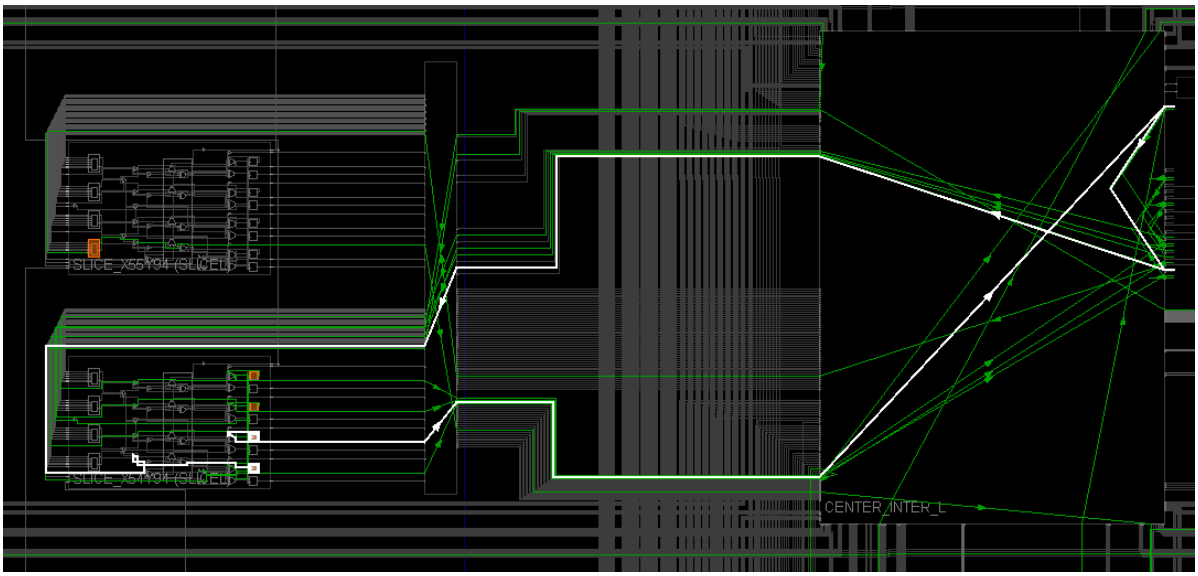


Figure 2.3.2.9 Routing layout after modification

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	1.305 ns	Worst Hold Slack (WHS):	0.185 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	23	Total Number of Endpoints:	23

All user specified timing constraints are met.

Figure 2.3.2.10 Design timing summary after modification

2.4 Routing Design Constraint

In Vivado, routing constraints are essential for guiding the placement and routing process to meet specific design requirements. Routing is a part to rearrange the layout of wires to improve the performances of results about power, timing, or

utilization statistics etc. These constraints ensure that signals in your FPGA design are routed efficiently and meet timing requirements. Different from what are completed in the placement and timing constraints, routing part is quite limited to work on due to the strict settings on the connection nodes of each net which can be utilized on Vivado [3].

2.4.1 Introduction to Routing Constraint

Routing design is a significant part to find out the potential paths between two cells depending on the permission of system. Here the complete steps of example process are to be presented below:

First, to see which path is specified to be routed, the wire layout view is necessary to be turned on to present all the routing paths clearly. In figure 2.4.1.1, the green lines are the normal wires the system chooses automatically to form the whole circuit, and the highlighted white line are the wire we select to make a change if it is possible.

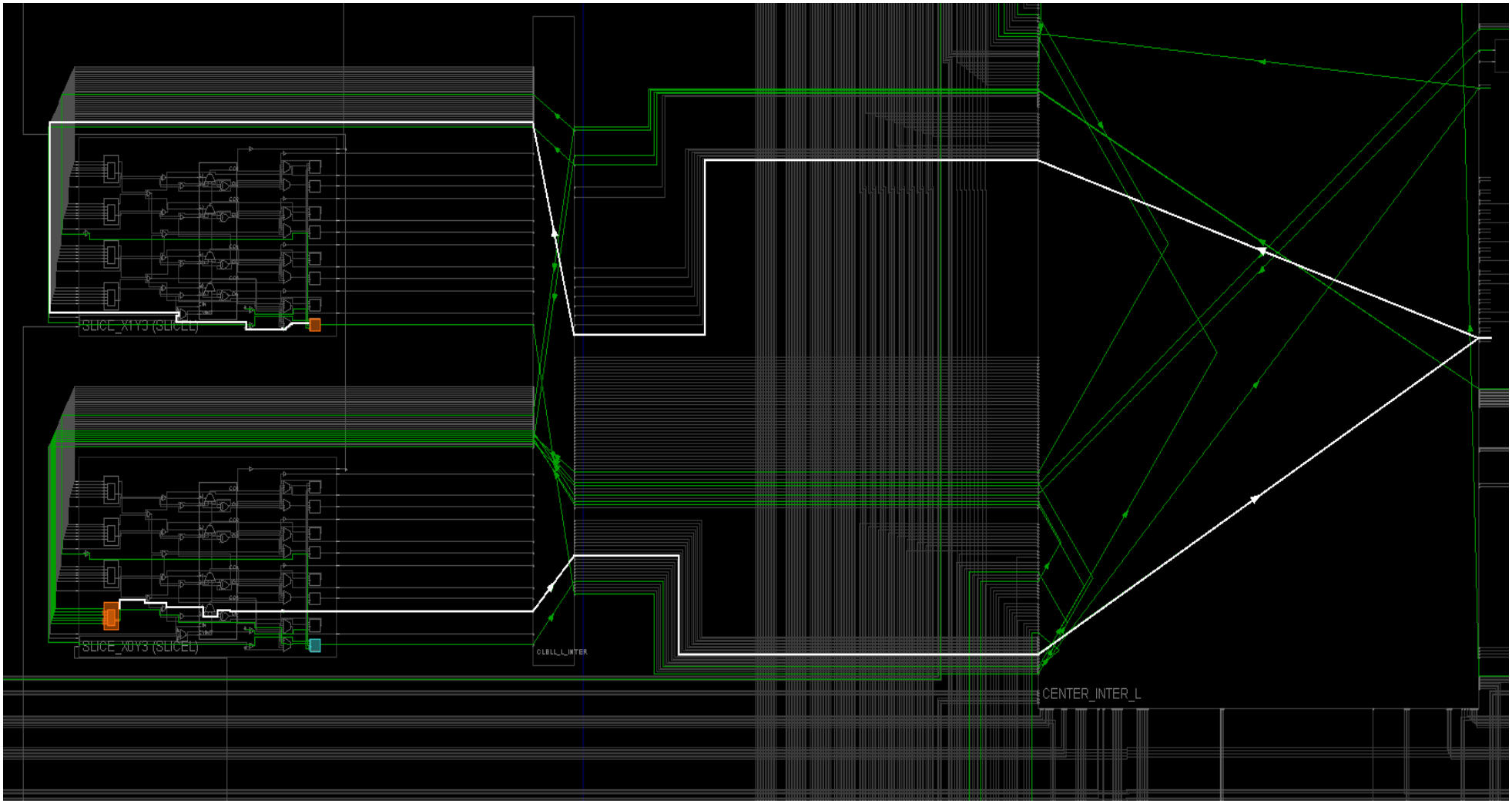


Figure 2.4.1.1 Routing diagram

After the wire of net is determined, assigning routing is what to do next to figure out if there is another way to route between two pins or if there is a better choice at each node. ‘Enter Assign Routing Mode…’ is an option to check the routing information of this particular net which is already highlighted white. This mode is not the only method to reassign routing, but it is the most effective way to obtain the whole list of nodes in order and to know if the new routing is feasible [10].

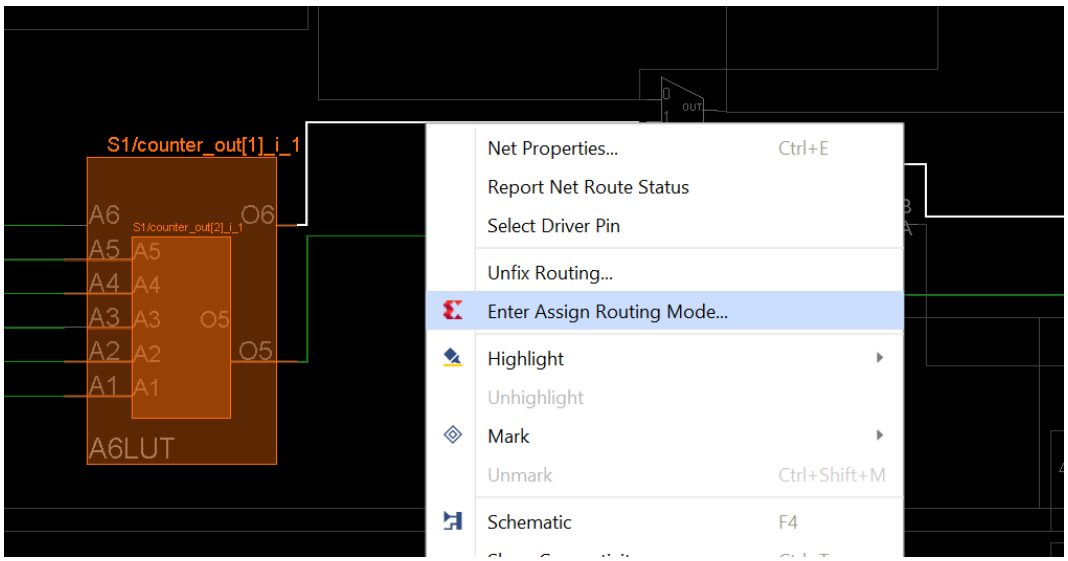


Figure 2.4.1.2 Assign Routing Mode option

Before entering this mode, targeting load cell pin is the last step to specify the exact pins of the two units. Sometimes there are several pins listed below and there is only one which can be dealt with for one time. After selecting the row of the cell pin, click ‘OK’ to officially enter the mode.

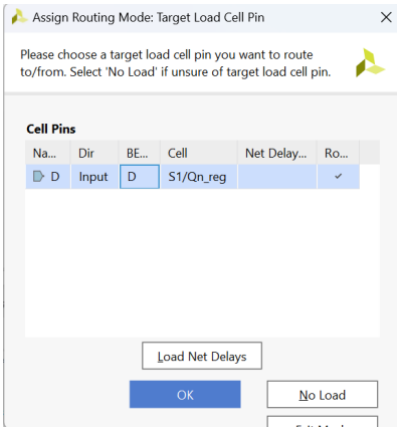


Figure 2.4.1.3 Cell pins

In Figure 2.4.1.4, the routing assignment interface is presented. The selected net is highlighted orange this time on left, and there are ‘Assigned Nodes’ and ‘Neighbor Nodes’ listed on right. Assigned nodes are the ones already on the routing line of this net and neighbor nodes are ones listed when a specific assigned node is selected. Not all the assigned nodes possess neighbor nodes, and this depends on the device layout. In general, adding and deleting of neighboring nodes are the target for routing design.

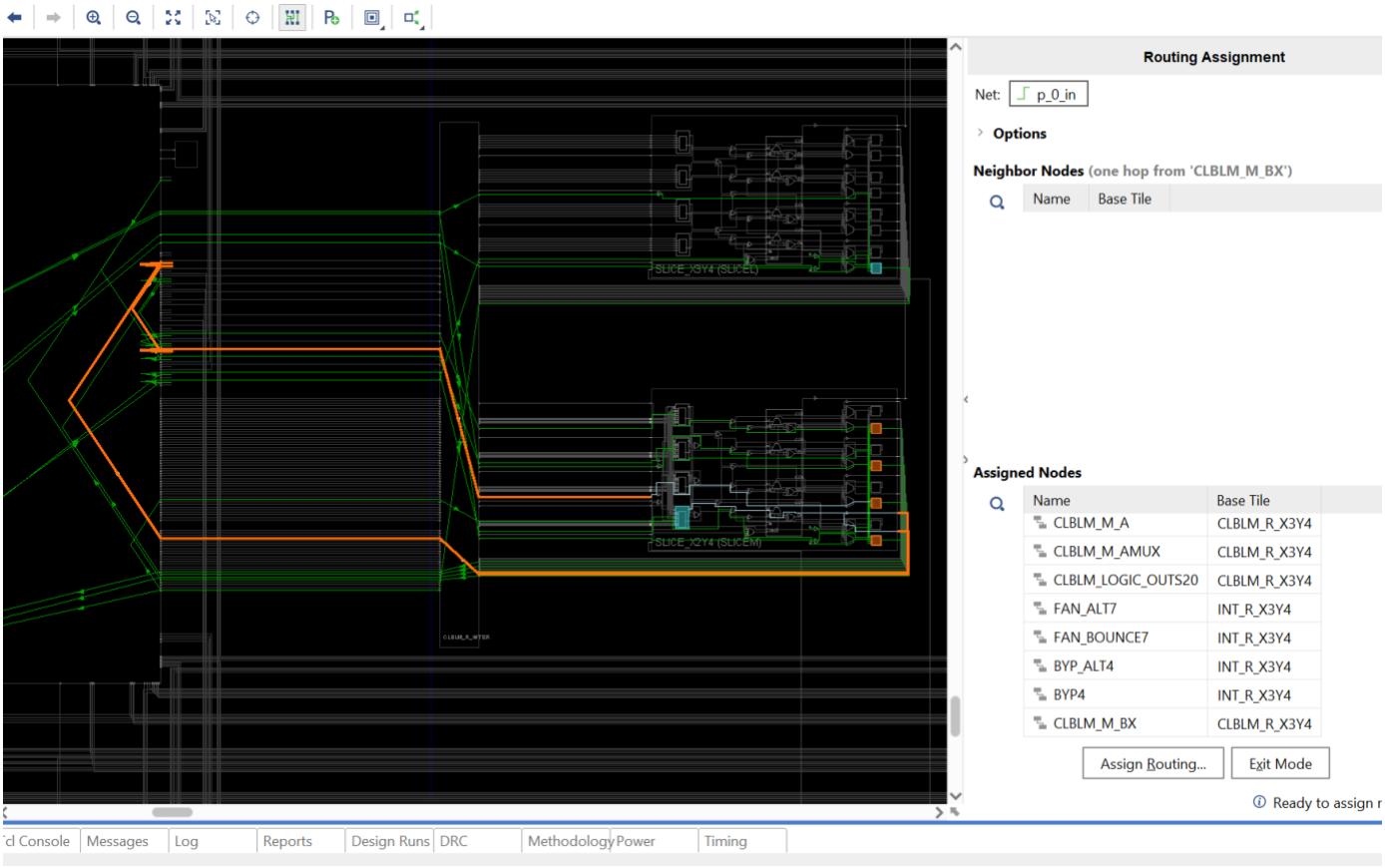


Figure 2.4.1.4 Routing assignment interface

Net gaps are vital to be utilized in routing as they provide the net with multiple choices to choose any wire between two nodes in a gap. It is to be added anywhere on the list of assigned nodes and once a gap is needed, at least one wire is better to be deleted between two nodes to offer spaces for other routing ways.

In Figure 2.4.1.5, there are two breaking points in the orange line because the wire connecting them has been deleted and a net gap is formed afterwards. The next step is to search for other reasonable routes between two points even if it has a possibility to result in worse result.

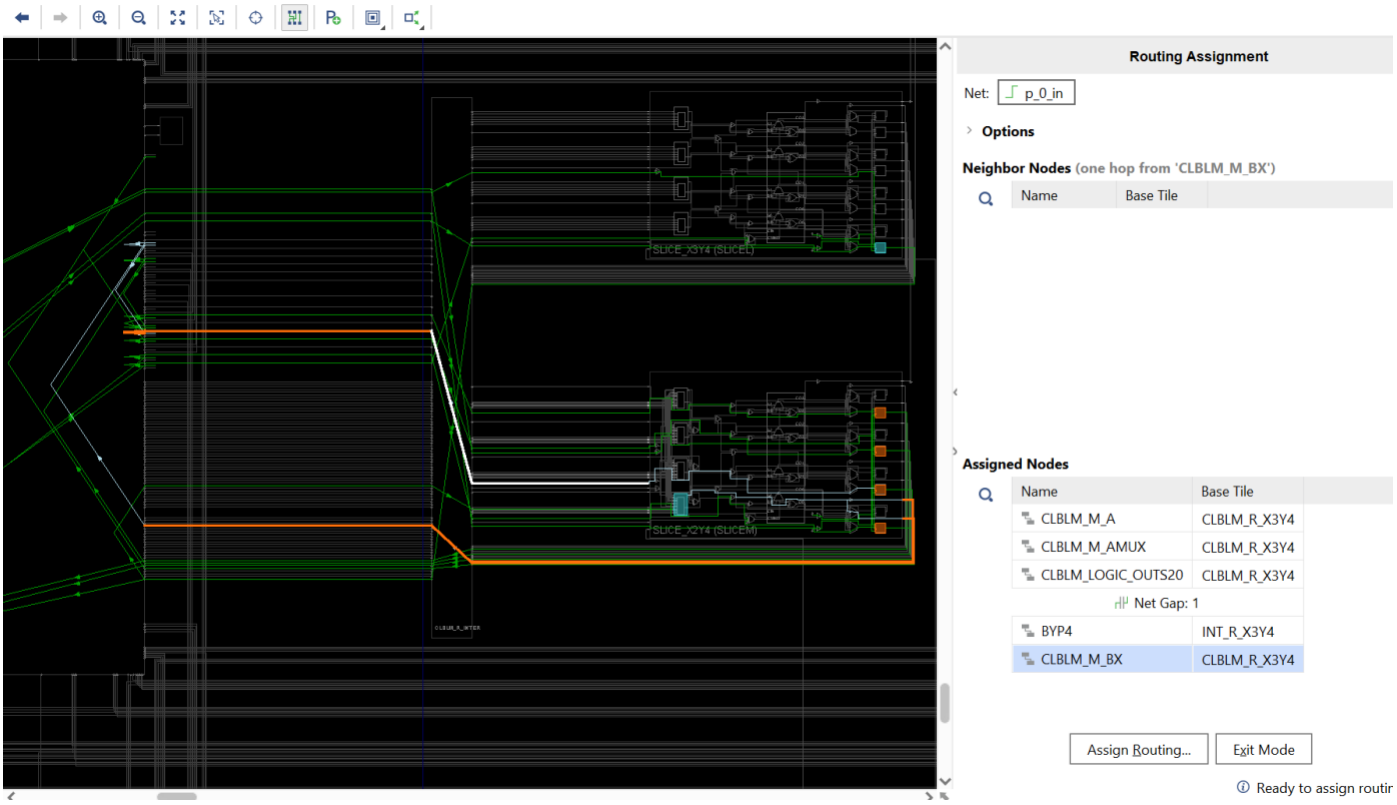


Figure 2.4.1.5 Net gaps

If an existing assigned node is selected as shown on Figure 2.4.1.6, all the potential choices after the node are noted as highlighted dotted-white on the routing diagram. There are quite a lot of wires shown but not all of them are helpful to connect two points. The work next is to find out the proper wires other than what the system automatically generates. The method of adding them can be from the neighbor nodes or select the dotted-white lines directly. In this part, system errors are prone to occur due to the set wiring regulations of Vivado, so it is probable to take several attempts to route the wires successfully.

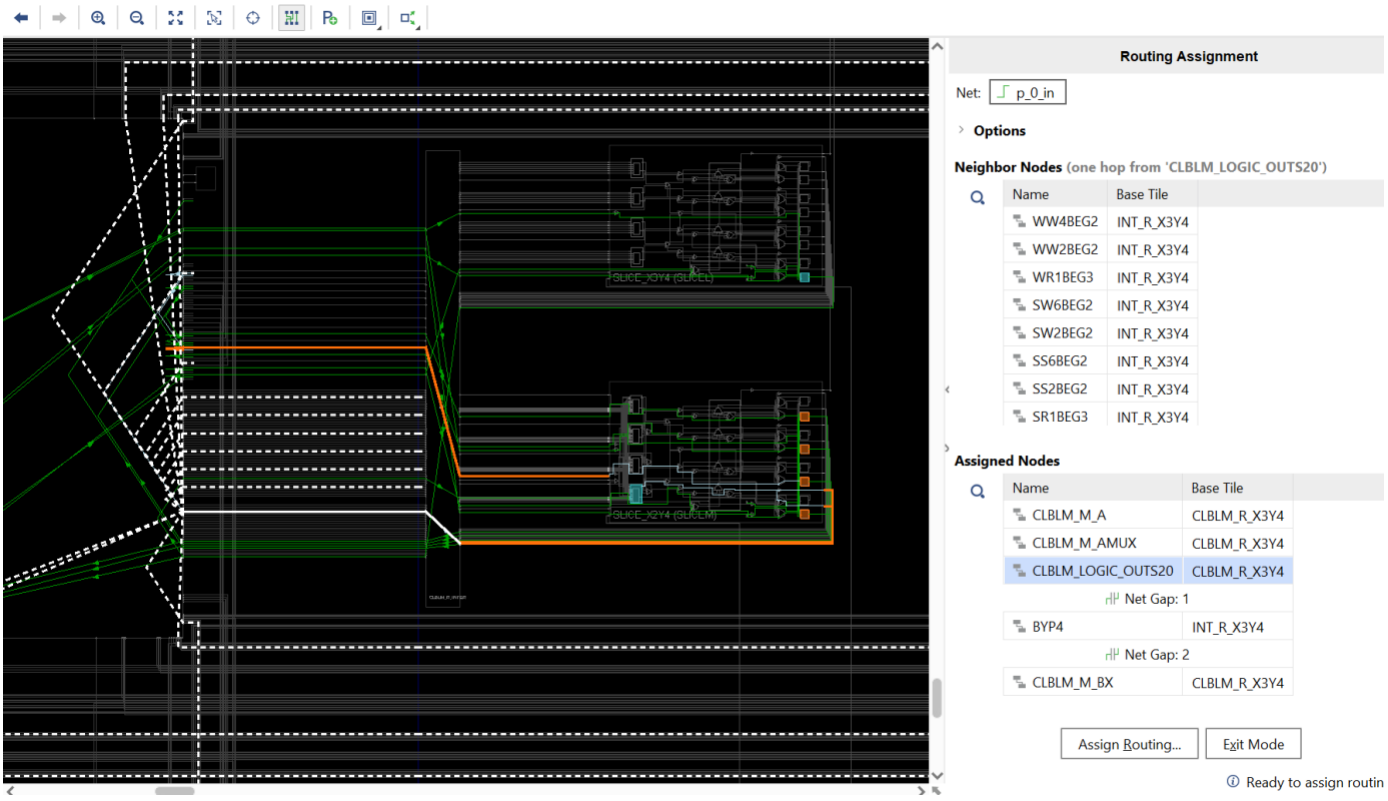


Figure 3.4.1.6 Neighbor nodes

After attempting for new routes, the assigned nodes list is updated as well as the routing diagram on the left shown as Figure 2.4.1.7.

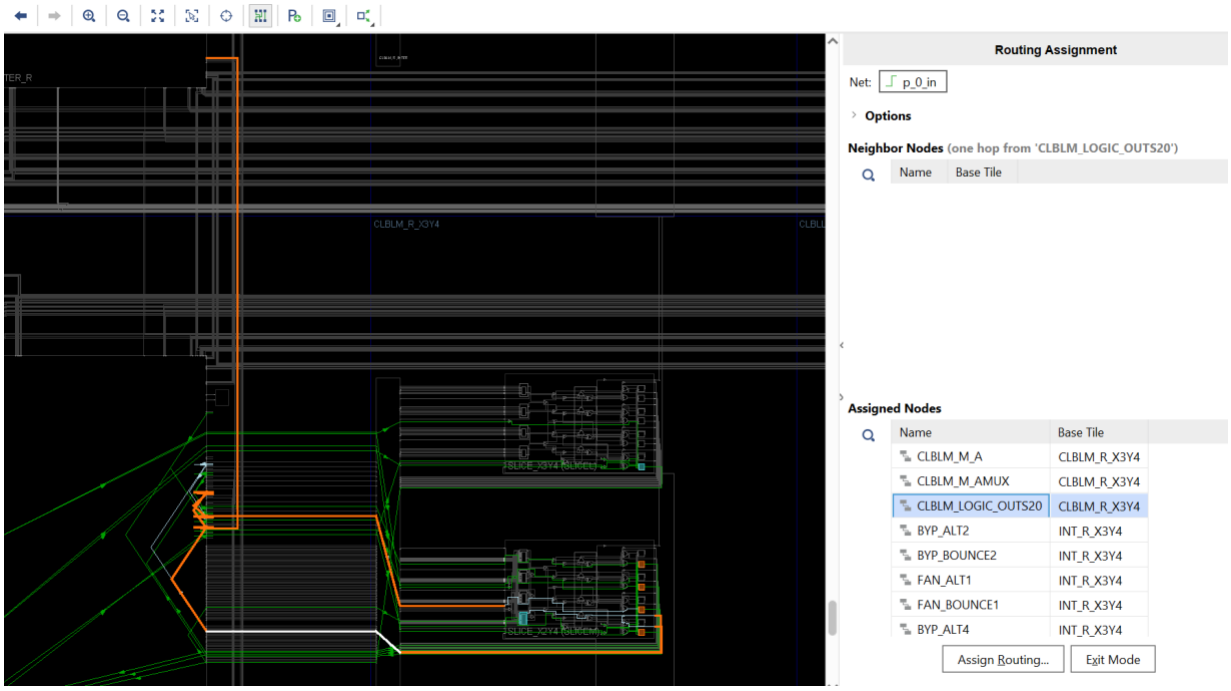


Figure 2.4.1.7 Assigned nodes completed

The full list is shown as below and the box of 'Fix Routing' can be checked if we do not hope a re-implementation operation affects the manual work by routing the circuit automatically again.

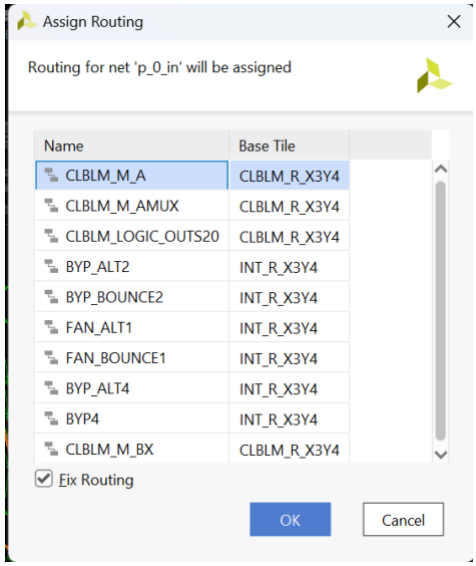


Figure 2.4.1.8 Complete routing

By clicking ‘OK’, a formal routing for net ‘p_0_in’ is completed if there is no errors or warnings popping up. The ultimate design is presented with dotted-white lines in the final diagram. A statement will be added to the target constraint file before an implementation process, or it can be added manually by entering status query under the tcl console.

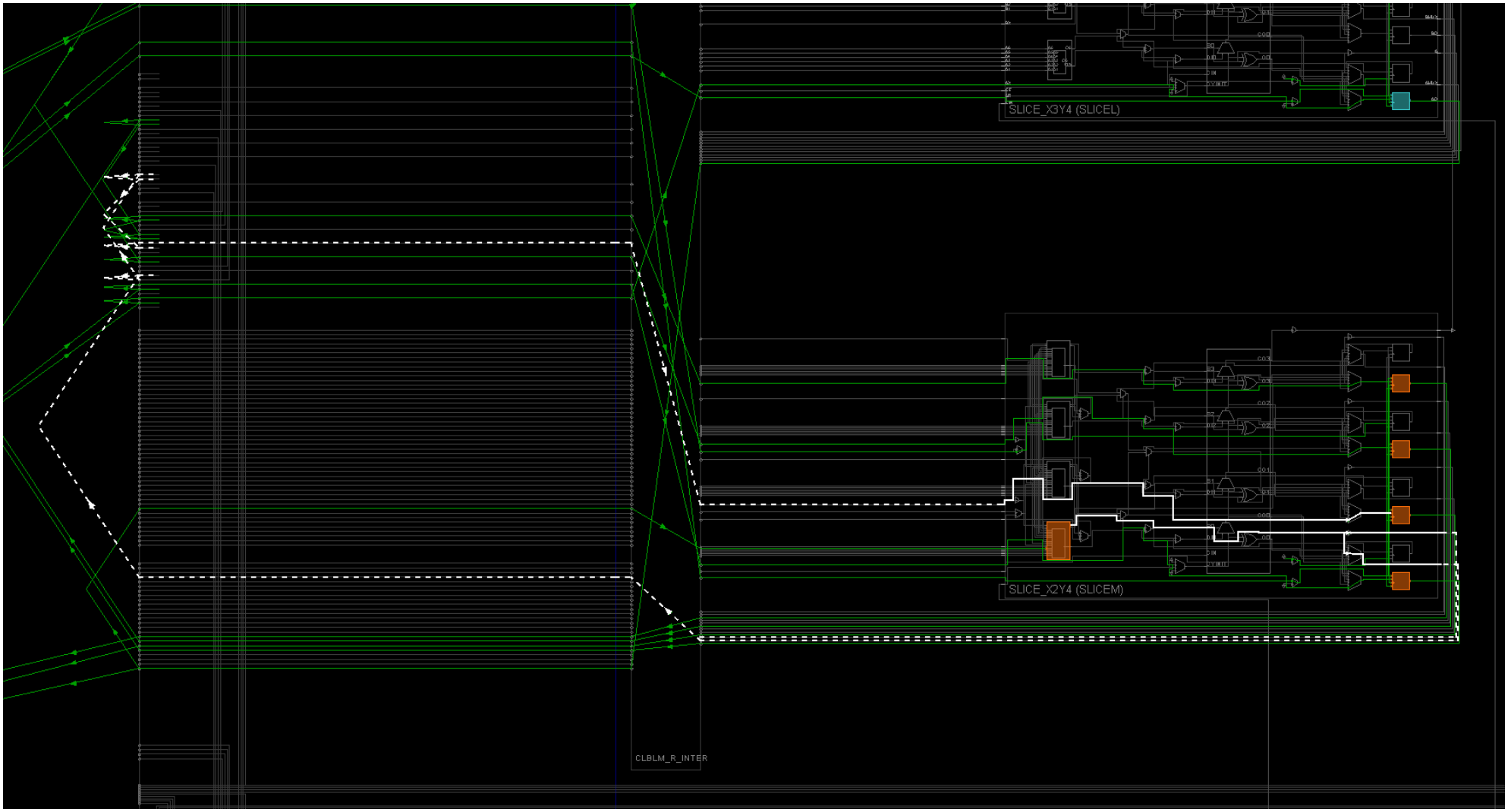


Figure 2.4.1.9 Final diagram

A completement statement of routing nodes details is shown in the console to be copied and added in the constraint file. '*set net* [*get_nets p_0_in*]' specifies the target net and '*get_property ROUTE $net*' obtains the current nodes list of the path [3] [14].

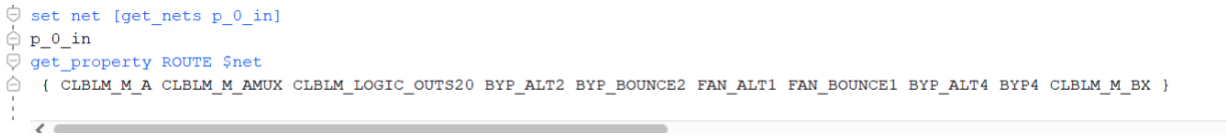


Figure 2.4.1.10 Path query

2.4.2 Routing Design Analysis

Routing design is hard on Vivado even if the router gives you complete control over the routing paths. When the nets chosen by the system are deleted and the user wants an 'auto-route' or to select the nets manually, there will be two failure scenarios: Routing remains unchanged due to limited nets nodes between the two cells, or a more lengthy and complex routing method. After completion of placement, the next task is attempting to see if the setup or hold time slack can be optimized further.

First, the critical path of the worst hold slack is showed in figure 2.3.2.9 and the assigned nodes list is presented in figure 2.4.2.1.

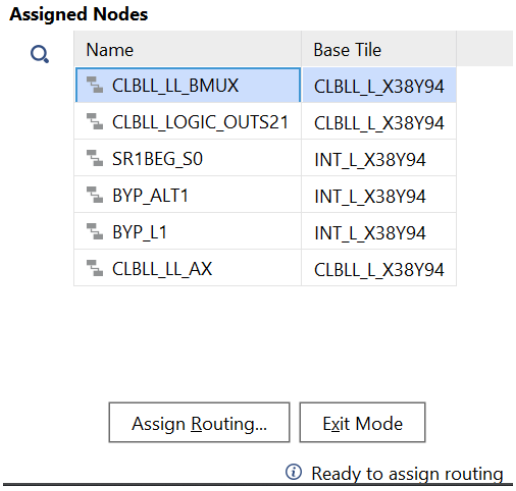


Figure 2.4.2.1 Assigned nodes of critical path

After removing the uncertain wires and nodes in the switch boxes, the rest of them are all fixed and unable to be replaced.

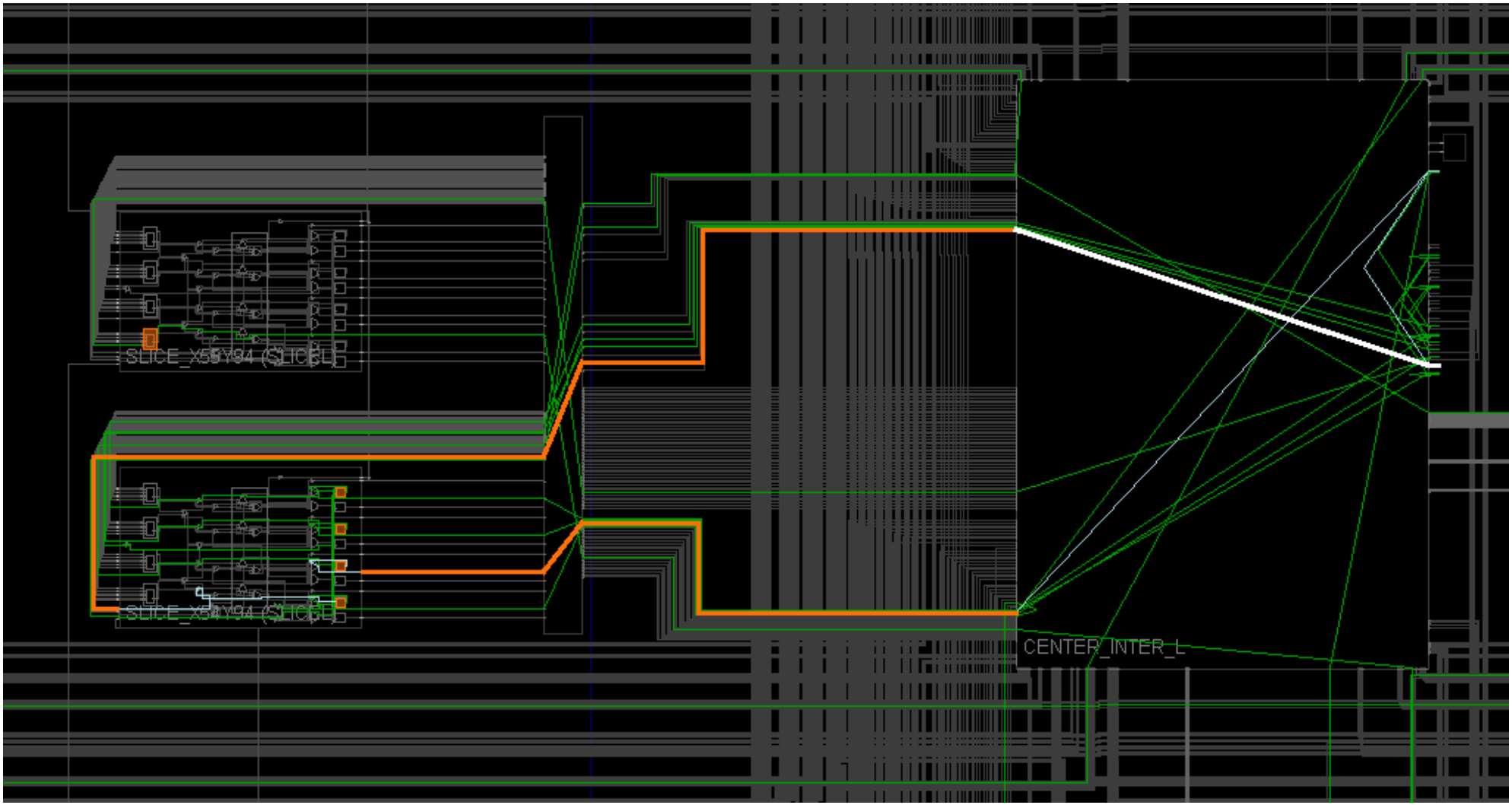


Figure 2.4.2.2 Rest of the assigned wires

All the routing plans in the switch box indicate that the automatically assigned nodes are the only proper way that can be chosen because the junction pins on the box transferring the signal over the path have been fixed already. The optimization of the worst hold slack cannot be implemented in the process of routing constraining.

For the worst setup slack, the critical path shows a routing utilizing eight switch boxes. To avoid congestion and high delay in the switch box, a better routing plan can be figured out in the layout, and the new routing reduced the number of switch boxes to six.

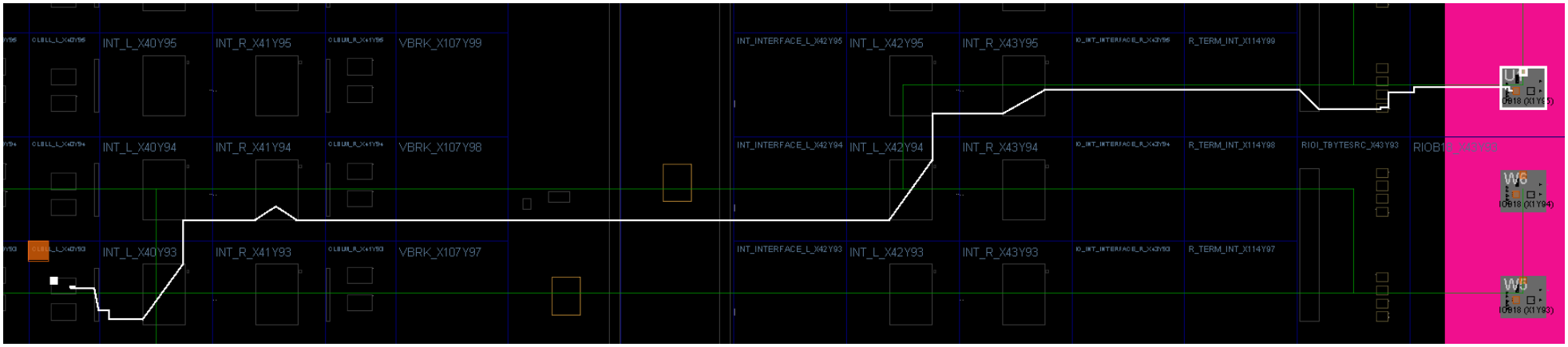


Figure 2.4.2.3 Critical path of the worst setup slack

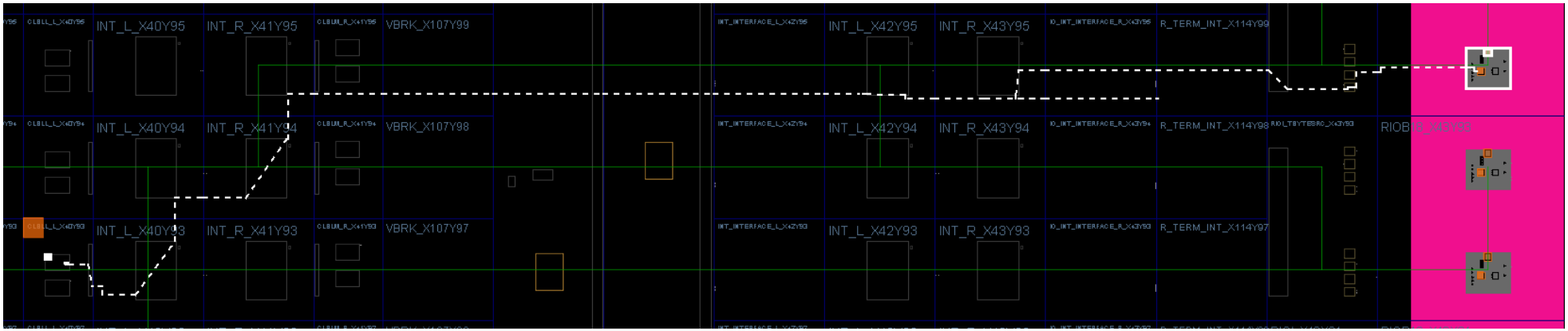


Figure 2.4.2.4 New routing for the worst critical path

The result after implementation is improved indeed with smaller increase of the setup slack time.

Design Timing Summary

Setup		Hold	
Worst Negative Slack (WNS):	1.307 ns	Worst Hold Slack (WHS):	0.185 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	23	Total Number of Endpoints:	23

All user specified timing constraints are met.

Figure 2.4.2.5 Design timing summary after routing

III. Discussion

3.1 Timing result

Through the entire simulation, timing, placement, and routing all function to improve the timing results after implementation. Timing constraints includes input and output delays settings and, in this project, only output delays affect the timing results rather than input delays due to the logic architecture of the output circuit. First, in the design, the critical paths are more dependent on the logic of data transmission rather than the arrival time of input signals. Second, the worst path of output has the highest fan-out of 6 so that the required delay time must meet the requirements of the slowest load on the output path, in this way, it can directly affect the worst slack time. More commands can be used in the timing constraint to analyse the timing result. '*set_false_path'* is a statement to tell the system not to consider or ignore specific paths during timing analysis in the process of implementation. This command can help reduce the time of analysis by excluding the unnecessary paths. '*set_data_check'* can also be used to define the setup and hold time requirements for the paths especially in the data captured process for flip-flops in the circuit, this statement ensures the destination registers or flip-flops sample the data reliably [9].

In the placement constraints, all the units are relocated to the region where the delays on wires are lower than they are when the layout is generated automatically by Vivado. Due to the limited size of the circuit, the cells cannot be placed very even on the device so that there are always some long wires in the clock region which cannot be evitable. Instead of completing the placement manually, the placement

can be done by another constraint command ‘group related logic’. This command aims to group the related logic elements together spatially during placement which helps ensure that related logic elements are placed close to each other on the FPGA device, which provides multiple benefits, including reduced routing delays, improved timing, and minimized congestion. The standard statement is ‘*creat_area_group <group_name>*’ and ‘*add_cells_to_area_group <group_name> [get_cells <cell_list>]*’ where ‘*group_name’* is a user-define name and ‘*cell_list*’ is the list of cells composing the circuit that are assigned to belong to the group. The first benefit for using the command is that the signal propagation delays can be reduced especially for high-speed design. Second, it makes the length of routing paths reduced to some extents leading to a less congestion and a better routing quality [3].

Routing is rather tricky among all the design constraint in Vivado. First, the choices given by the system is limited when a routing needs to be undertaken. The reason for it is that the distance between each cell is not long enough for us to choose which switch boxes or routing nets are more proper for the path. Second, although there are multiple paths to select in a complete net, most of them are quite similar that cannot affect much to the results owing to a circuit which is not very congested. Third, some choices of routing path may lead to higher delayed results because the complexity of routing is difficult to determine in some large-scale projects.

3.2 Power result

In addition to the timing results, power reports also reveal some different information before and after the placement constraints are added.

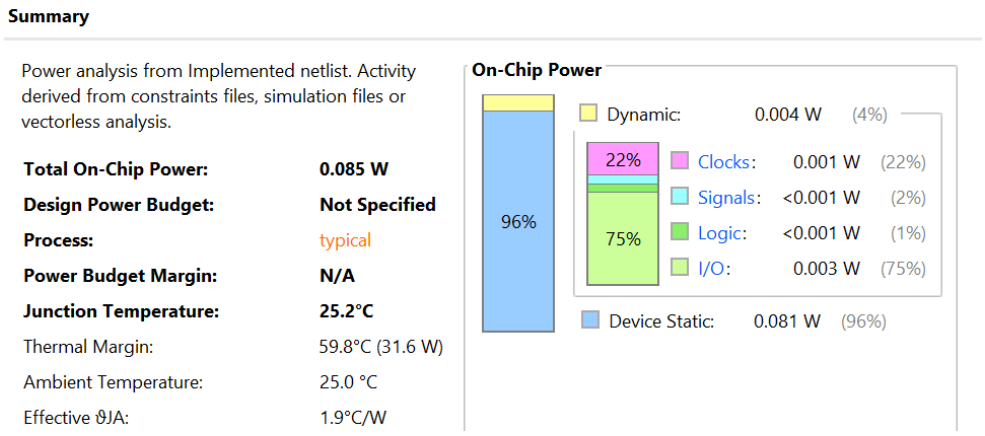


Figure 3.2.1 On-Chip power distribution (before placement)

From figure 3.2.1, it can be observed that the device static power includes clocks power, signals power, logic power, and I/O power in which the clocks and I/O power dominate. After adding placement constraints, the I/O power consumed increases while the clocks power consumed reduces, at the meantime, their total amount remains constant.

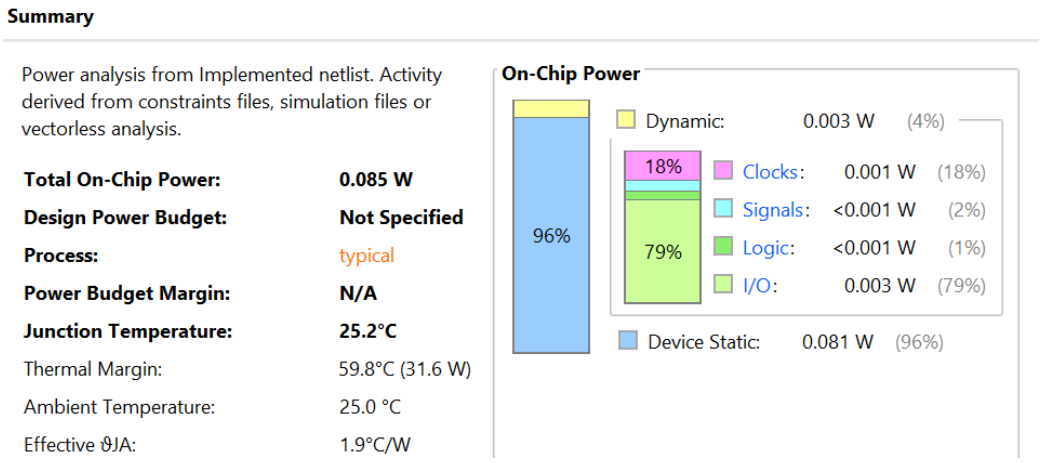


Figure 3.2.2 On-Chip power distribution (after placement)

The device static power is the power generated by the leakage of all connected voltage rails and transistors on the circuit required for normal operation of the device after configuration. Design changes or optimizations may have increased the activity levels on the I/O pins which leads to a higher power consumption. Even if the placement and routing constraints modification gives an optimal timing result, the increased activity of I/O ports will result in higher power consumption. Besides, the reason for this kind of power variation of clocks and I/O pins may result from the compensation effect caused by the power estimation tools of Vivado. It chooses to automatically adjust one component to maintain a relatively constant power consumption [15].

The methods for attempting to reduce the device static power are multiple. First, the power management constraints are the top choice for us to specify power-saving modes and features through constraints. The principle of it is to reduce the total power consumption by setting the unused resources into a low-power state. Second, the voltage scaling techniques can be utilized to reduce power consumption by driving the device in a low voltage level. The related voltage constraint '*CONFIG_VOLTAGE'* can specify the voltage levels in the clock region which is occupied on the device [16].

IV. Conclusion

During the process of design constraint for a frequency divider with ring counter, the timing, placement, and routing commands are all utilized to improve the running result of timing. Timing constraints have multiple clocks-defining statements and in this project, a clock with a cycle duty of 50% is used to function as the primary clock of the whole circuit. Then the input and output delays are added to specify the timing requirement to the circuit. Input delays have no effect on the implementation result in this project while the output delays affect a lot on the timing results. The slack time varies sensitively as the variation of maximum output delays varies especially for the worst setup slack time. According to the curve diagram of the timing results, an appropriate interval is selected to set the maximum output delay.

Placement constraint part focuses on the connection of every module and the positional relationship between each cell including LUTs, flip-flops, I/O ports, and clock. The schematic shows a straightforward logic structure to help understand how to place these sources in a proper way within a clock region. There are multiple commands for placement to choose the site of a specific slice, and the type of the source. In this process, output and input path are significant to pay attention to due to the critical path analysis. After the placement, an obvious improvement can be observed on the timing analysis report.

Routing design constraint relies on much on the system tool of Vivado. It is always plenty ways to figure out a wiring optimization on a critical path while some of them does not commit to improving the timing results. Too many Switches boxes and

congestion are both the factors which are decided to be avoided. For the worst hold slack, the critical path has no room for improvement because the input pin and the output pin on the two cells have been fixed on the device. For the worst setup slack, a better way can be found for the critical path to improve its slack even though it is a slight variation.

Bibliography

[1] Chu, P. P. (2011). FPGA prototyping by VHDL examples: Xilinx MicroBlaze MCS SoC. John Wiley & Sons.

[2] Woods, R., McAllister, J., & Lightbody, G. (2011). FPGA-based implementation of signal processing systems. John Wiley & Sons.

[3] Xilinx, Inc. (n.d.). Vivado Design Suite User Guide: Using Constraints (UG903) [PDF]. Retrieved from https://www.xilinx.com/support/documentation/sw_manuals/xilinx2021_2/ug903-vivado-using-constraints.pdf.

[4] Williams, Marshall, "Shift register delay circuit", issued 1985-07-16.

[5] Pedroni, Volnei A. (2013). Finite State Machines in Hardware: Theory and Design. MIT Press. ISBN 978-0-26201966-8.

[6] Holdsworth, Brian; Woods, Clive (2002). Digital Logic Design *(*4 ed.). Newnes Books / Elsevier Science. pp. 191–192. ISBN 0-7506-4588-2.

[7] G. Baek and H. Jeong, "All-Digital Time-Domain Temperature Sensor for Energy Efficient On-Chip Thermal Management," 2022 International Conference on Electronics, Information, and Communication (ICEIC), Jeju, Korea, Republic of, 2022, pp. 1-4, doi: 10.1109/ICEIC54506.2022.9748344.

[8] Brown, S. J., & Rose, J. M. (1994). ″ Field Programmable Gate Arrays: Architecture and Tools for Rapid Prototyping. ″ IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(12), 1521–1537.

[9] Sridhar Gangadharan, & Sanjay Churiwala (2013). ″ Constraining Designs for Synthesis and Timing Analysis A Practical Guide to Synopsys Design Constraints (SDC) for the paper. ″ Springer Science+Business Media, New York, 2013, doi: 10.1007/978-1-4614-3269-2.

[10] Xilinx, Inc. (n.d.). Vivado Design Suite User Guide: Implementation (UG904) [PDF]. Retrieved from https://docs.amd.com/r/en-US/ug904-vivado-implementation/Routing.

[11] K. Sreedharan and A. J. Raghunathan, "Timing constraint optimization for field-programmable gate arrays," in IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, vol. 11, no. 6, pp. 1086-1097, Dec. 2003.

[12] M. Maniatakos, S. Xu and W. L. Miranker, "Constraint-Based Placement and Routing for FPGAs Using Self-Organizing Maps," 2008 20th IEEE International Conference on Tools with Artificial Intelligence, Dayton, OH, USA, 2008, pp. 465-469, doi: 10.1109/ICTAI.2008.55.

[13] Xilinx, Inc. (n.d.). Vivado Design Suite User Guide: I/O and Clock Planning (UG899) [PDF]. Retrieved from https://docs.amd.com/r/2021.1-English/ug899-vivado-io-clock-planning.

[14] Xilinx, Inc. (n.d.). Vivado Design Suite Tutorial: Using Constraints (UG945) [PDF]. Retrieved from https://docs.amd.com/r/en-US/ug945-vivado-using-constraints-tutorial.

[15] Xilinx, Inc. (n.d.). Vivado Design Suite User Guide: Power Analysis and Optimization (UG907) [PDF]. Retrieved from https://docs.amd.com/r/2021.1-English/ug907-vivado-power-analysis optimization.

[16] Xilinx, Inc. (n.d.). Vivado Design Suite User Guide: I/O and Clock Planning (UG899) [PDF]. Retrieved from https://docs.amd.com/r/2021.2-English/ug899-vivado-io-clock-planning.