USING A Shift-and-Add METHOD DESIGNING 8-BIT Multiplier

An 8-bit multiplier is a digital circuit or component that is specifically designed to perform multiplication operations on two 8-bit binary numbers. It takes two 8-bit binary numbers as input and produces a 16-bit binary number as output, which represents the product of the two input numbers.

I design a multiplier for unsigned binary numbers. When we form the product A × B, the first operand (A) is called the multiplicand and the second operand (B) is called the multiplier.

Note that each partial product is either the multiplicand (11010000) shifted over by the appropriate number of places or zero. Instead of forming all the partial products first and then adding, each new partial product is added in as soon as it is formed, which eliminates the need for adding more than two binary numbers at a time.

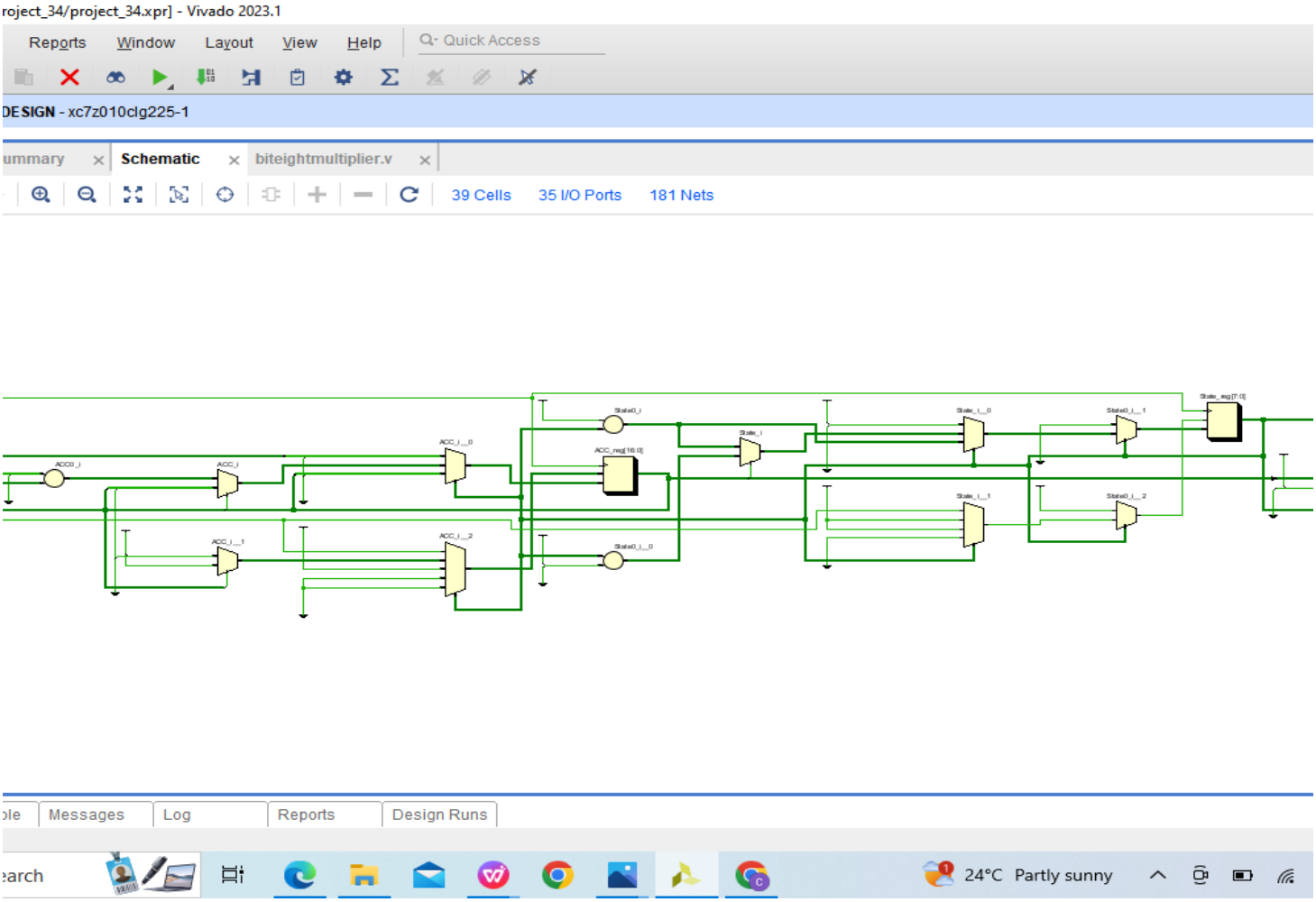
Multiplication of two 4-bit numbers requires a 4-bit multiplicand register, a 4-bit multiplier register, a 4-bit full adder, and an 8-bit register for the product. The product register serves as an accumulator to accumulate the sum of the partial products. If the multiplicand were shifted left each time before it was added to the accumulator, as was done in the previous example, an 8-bit adder would be needed. Therefore, it is better to shift the contents of the product register to the right each time.

Implementation of 8 bit multiplier:

```
`define M ACC[0]
module biteightmultiplier(Clk, St, Mplier, Mcand, Done, Result);
input Clk;
input St;
input[7:0] Mplier;
input[7:0] Mcand;
output Done;
output[15:0] Result;
reg[7:0] State;
reg[16:0] ACC;
initial
begin
State = 0;
ACC = 0;
end
always @(posedge Clk)
begin
case (State)
0 :
begin
if (St ==1'b1)
begin
ACC[16:8] <= 5'b00000 ;
ACC[7:0] <= Mplier ;
State <= 1 ;
end
end
1, 3, 5, 7, 9, 11, 13, 15:
    begin
if (`M == 1'b1)
begin
ACC[16:8] <= {1'b0, ACC[15:8]} + Mcand ;
State <= State + 1 ;
end
```

```
 else
 begin
 ACC <= {1'b0, ACC[16:1]} ;
State <= State + 2 ;
 end
 end
 2, 4, 6, 8, 10, 12, 14, 16 :
 begin
 ACC <= {1'b0, ACC[16:1]} ;
 State <= State + 1 ;
 end
 17 :
 begin
 State <= 0 ;
 end
 endcase
 end
 assign Done = (State == 17) ? 1'b1 : 1'b0 ;
 assign Result = (State == 17) ? ACC[15:0] : 16'b0101010100000 ;
endmodule
```

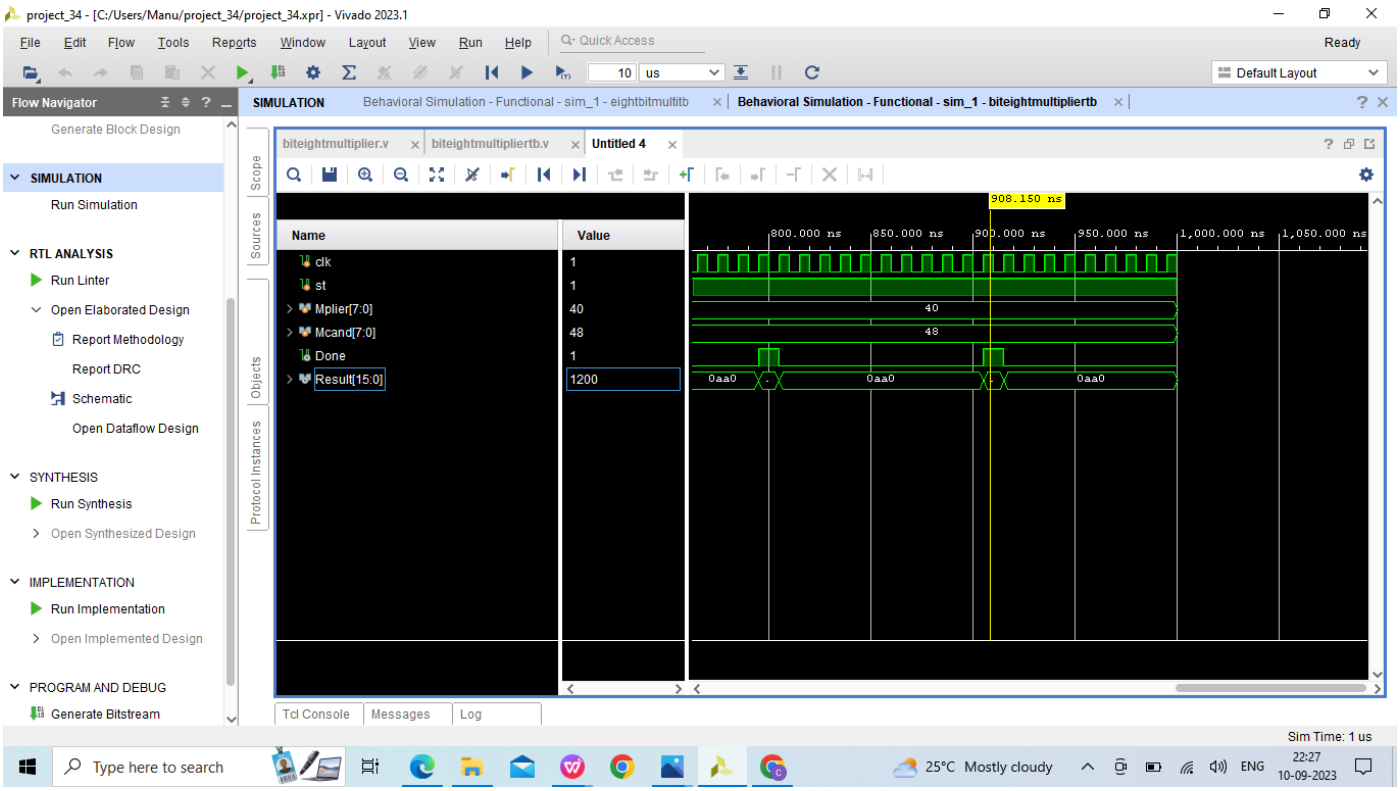
SCHEMATIC:



Testbench for 8 bit multiplier:

```
module biteightmultipliertb();
reg clk,st;
reg [7:0]Mplier,Mcand;
wire Done;
wire [15:0]Result;
biteightmultiplier uut (.Clk(clk),.St(st),.Mplier(Mplier),.Mcand(Mcand),.Done(Done),.Result(Result));
initial begin
clk=0;
st=0;
Mplier=0;
Mcand=0;
#20 st=1;
Mplier=10;
Mcand=5;
#150 Mplier=64;
Mcand=72;
#1000 $finish;
end
always
#5 clk=~clk;

endmodule
```

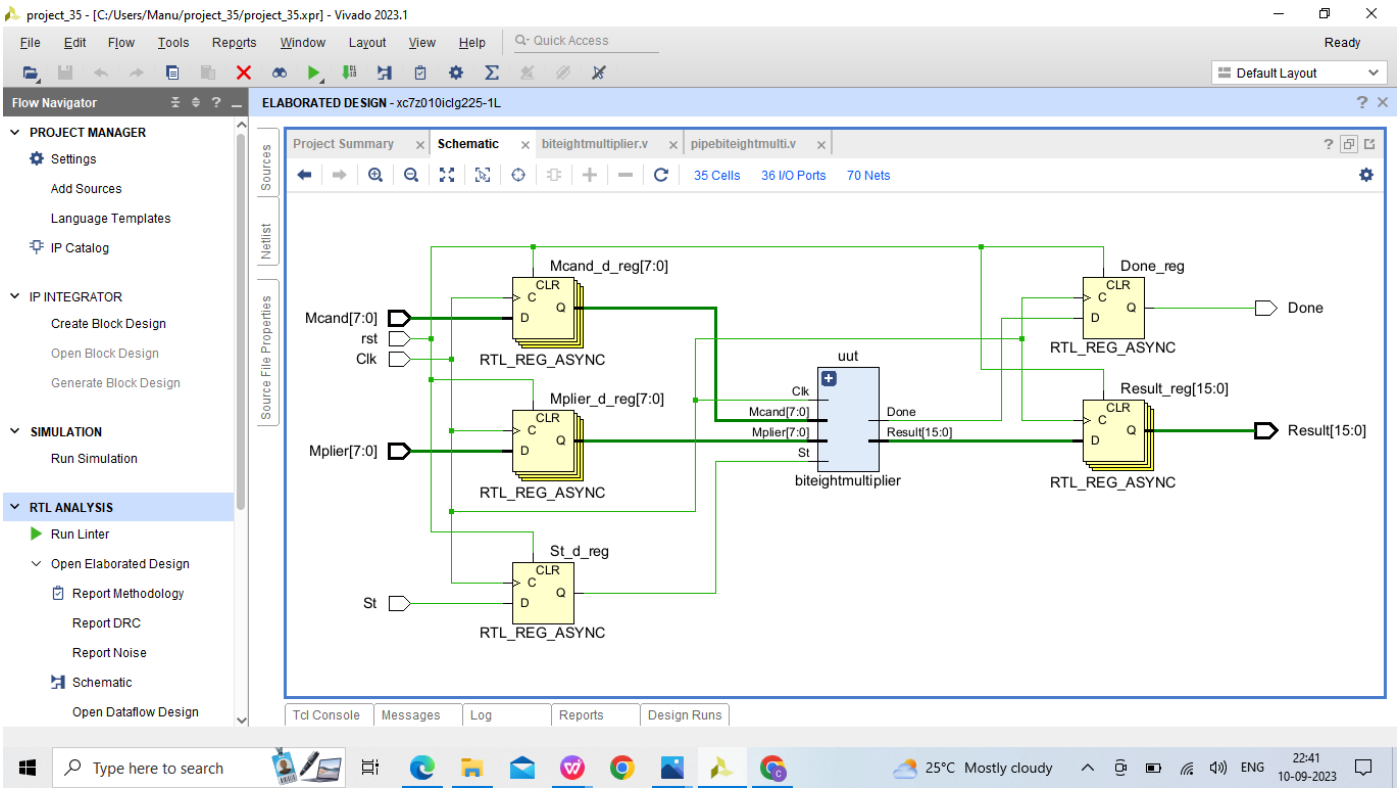
Waveform:



Implementation of 8 bit multiplier with pipeline:

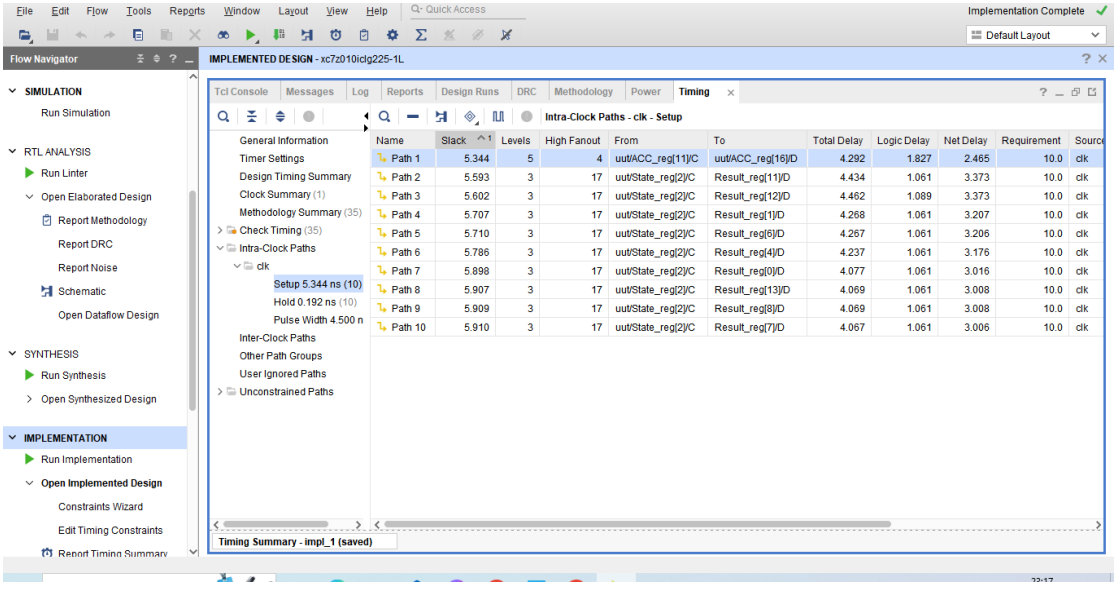
```
module pipebiteightmulti(
input Clk,
input rst,
input St,
input [7:0]Mplier,
input [7:0]Mcand,
output reg Done,
output reg [15:0]Result
            );

        reg St_d;
        reg [7:0]Mplier_d, Mcand_d;
        wire done_d;
        wire [15:0]result_d;

 multiplier uut(.Clk(Clk),.St(St_d),.Mplier(Mplier_d),.Mcand(Mcand_d),.Done(done_d),.Result(result_d));

    always @(posedge Clk or posedge rst) begin

    if(rst) begin

    St_d<=0;
    Mplier_d<=0;
    Mcand_d<=0;
    Done<=0;
    Result<=0;

    end

    else begin

    St_d<=St;
    Mplier_d<=Mplier;
    Mcand_d<=Mcand;
    Done<=done_d;
    Result<=result_d;

    end

    end

endmodule
```

SCHEMATIC:



Timing Summary :

There are total 10 setup timing paths and 10 hold timing paths. For the designed shown above the clock is kept constant that is 10ns.(**clock =10ns**)



1. **Path-1 :**

Source(Launch FF): uut/ACC_reg[11]/C

Destination (Capture FF):uut/ACC_ reg[15]/D

Source clock path: clock clk rise edge + net + buffer +net+buffer+ FDRE = 4.888 ns

Data path : FDRE (Prop_fdre_C_Q) + LUT2+CARRY4(prop c…4 S[3] CO[3])+ CARRY4(prop carry4 CI CO[3])+CARRY4(prop carry4 CI CO[0]) +LUT3 = 4.292ns.

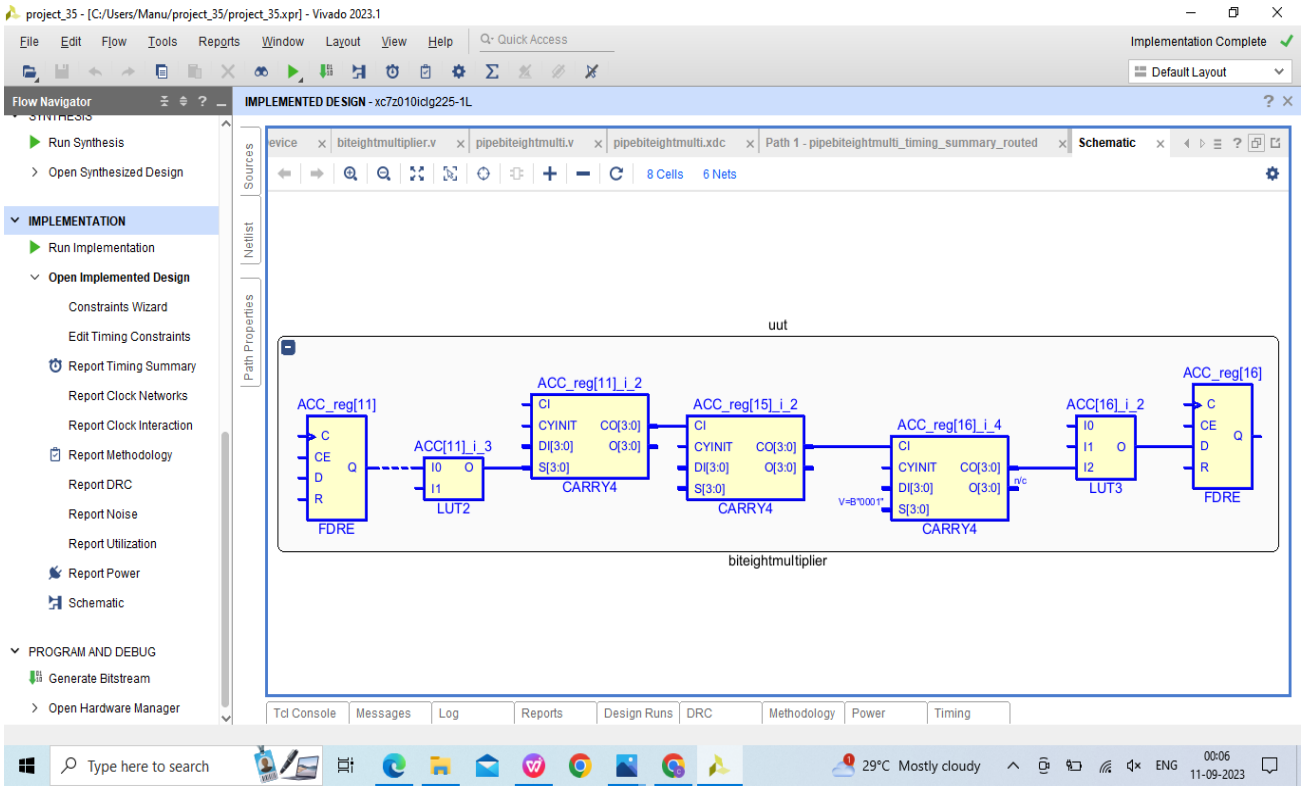
Arrival time : 5.158 + 6.928 = **9.180ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.524ns.**

Slack = Required time - Arrival time

= 14.524 – 9.180

= **5.344ns**



2. Path-2 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[11]/D

Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdre_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.443ns.

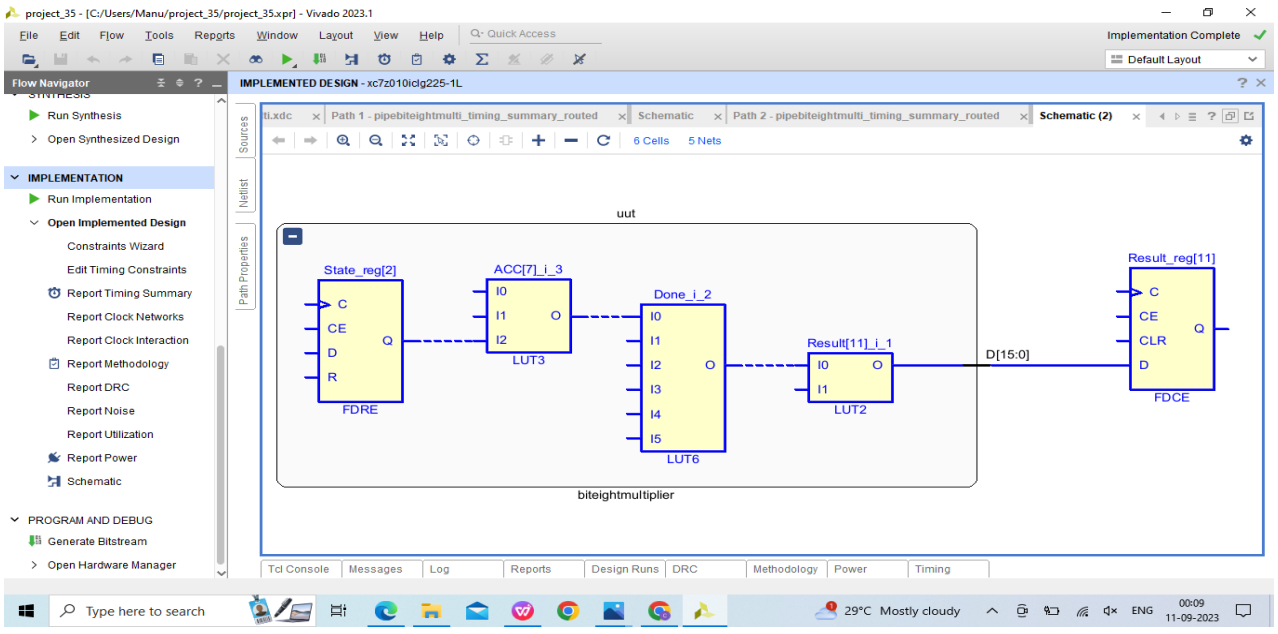
Arrival time : 4.886 + 4.443 = **9.320ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.913ns.**

Slack = Required time - Arrival time

= 14.913 – 9.320

= **5.593ns**



3. Path-3 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[12]/D

Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdre_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.462ns.

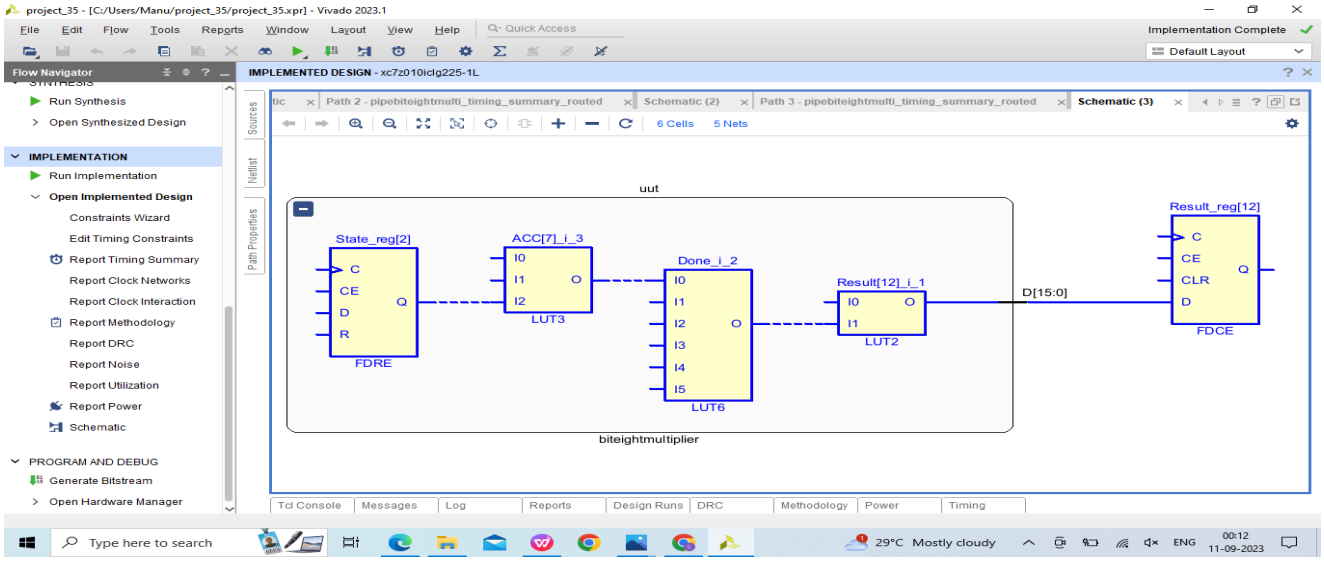
Arrival time : 4.886+ 4.462 = **9.348ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.950ns.**

Slack = Required time - Arrival time

= 14.950– 9.348

= **5.602ns**



4. Path-4 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[12]/D

Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.268ns.

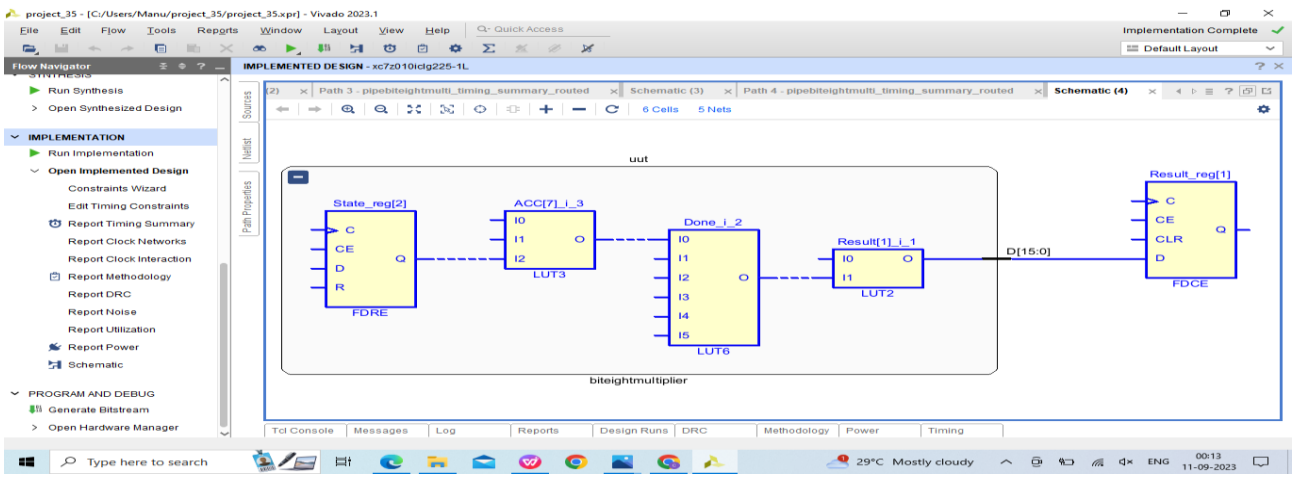
Arrival time : 4.886 + 4.268 = **9.154ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.861ns.**

Slack = Required time - Arrival time

= 14.861 – 9.154

= **5.707ns**



5. Path-5 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[6]/D

Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.267ns.

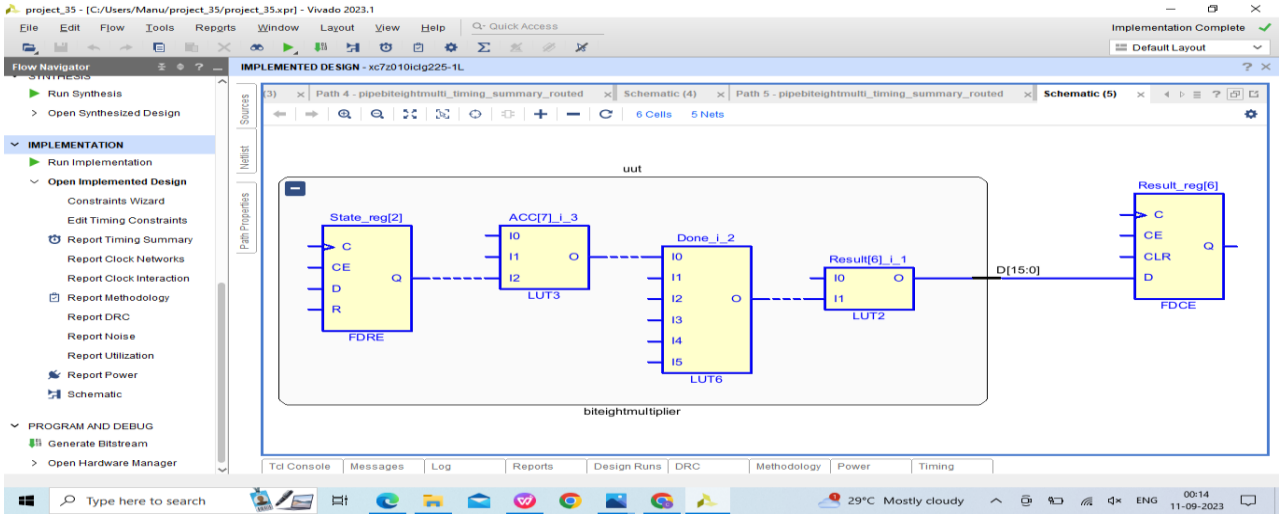
Arrival time : 4.886+ 4.267 = **9.153ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.863ns.**

Slack = Required time - Arrival time

= 14.863 – 9.153

= **5.710ns**



6. Path-6 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[6]/D

Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.237ns.

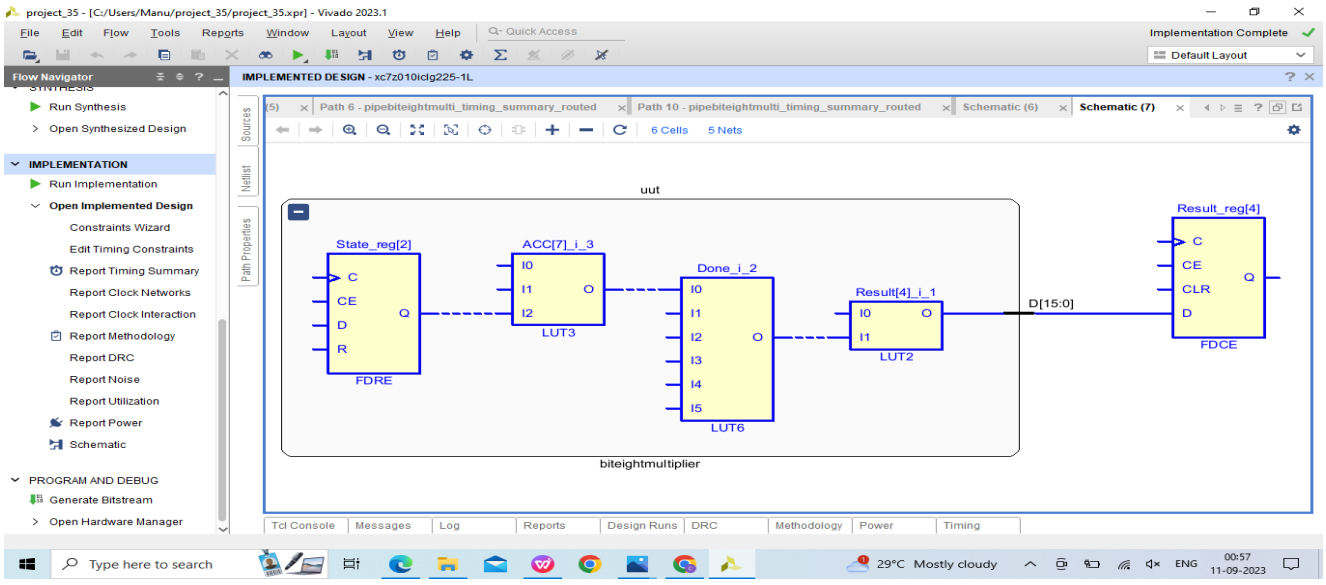
Arrival time : 4.886+ 4.237 = **9.123ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.864ns.**

Slack = Required time - Arrival time

= 14.864 – 9.123

= **5.786ns**



7. Path-7 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[0]/D

Source clock path: clock clk rise edge + net + buffer + FDRE (Prop_fdre_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.077ns.

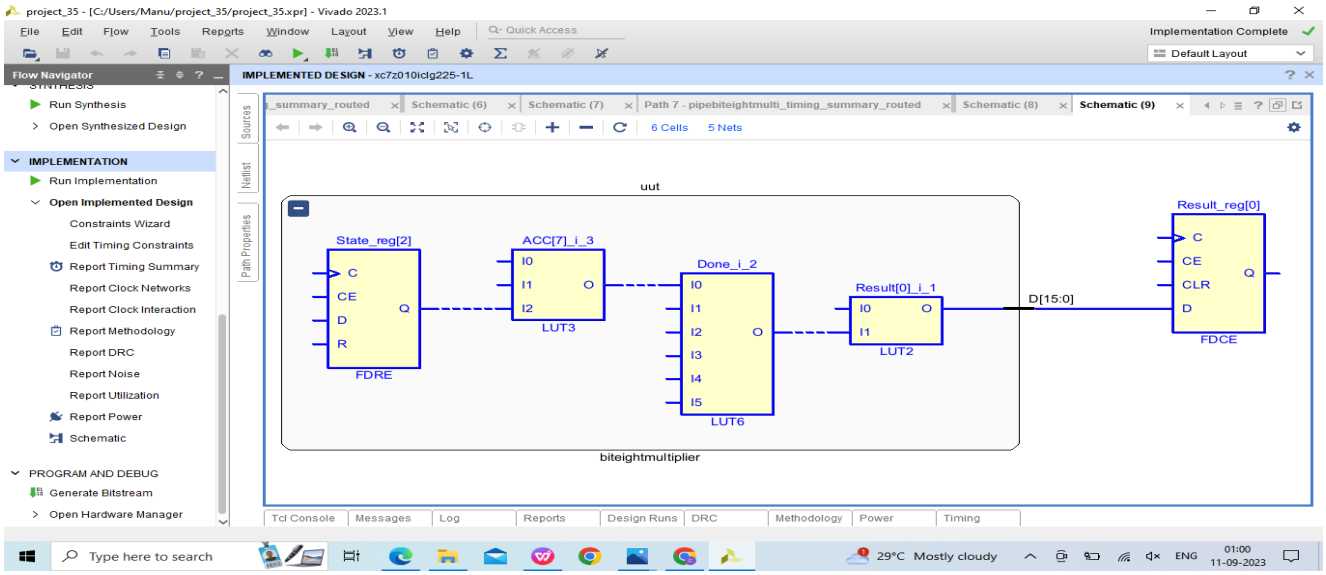
Arrival time : 4.886 + 4.077 = **8.963ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.861ns.**

Slack = Required time - Arrival time

= 14.861 – 8.963

= **5.898ns**



8. Path-8 :

Source(Launch FF): uut/ACC_reg[2]/C

Destination (Capture FF): uut/ACC_ reg[0]/D

Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdce_C) = 4.886ns

Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.069ns.

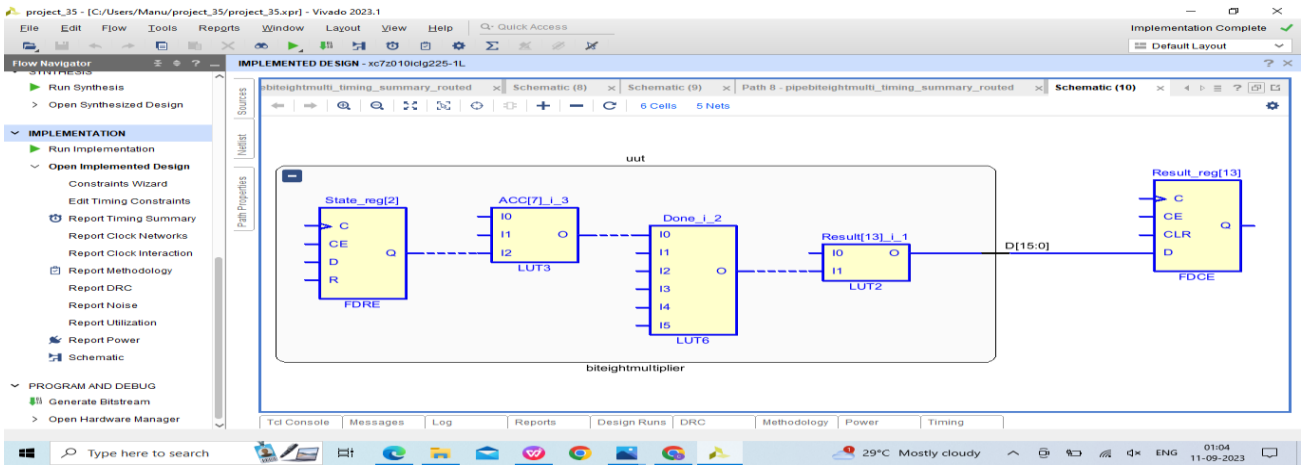
Arrival time : 4.886 + 4.069 = **8.955ns.**

Required time : clock clk rise edge + net + buff + cp + cu = **14.863ns.**

Slack = Required time - Arrival time

= 14.863 – 8.955

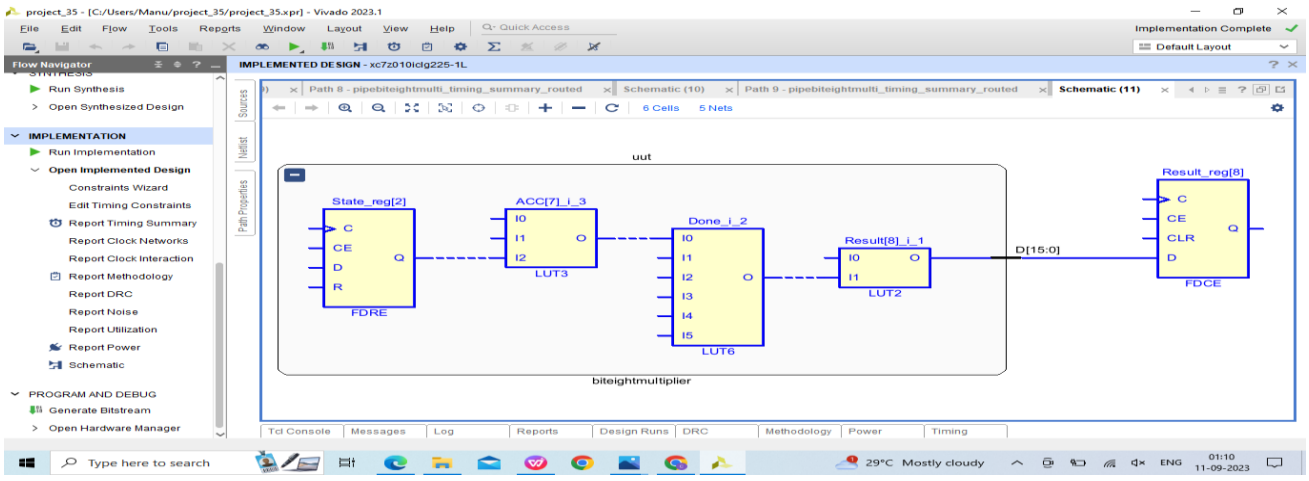
= **5.907ns**



9. Path-9 :

Source(Launch FF): uut/ACC_reg[2]/C
Destination (Capture FF):uut/ACC_ reg[8]/D
Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdre_C) = 4.886ns
Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.069ns.
Arrival time : 4.886 + 4.069 = **8.955ns.**
Required time : clock clk rise edge + net + buff + cp + cu = **14.864ns.**

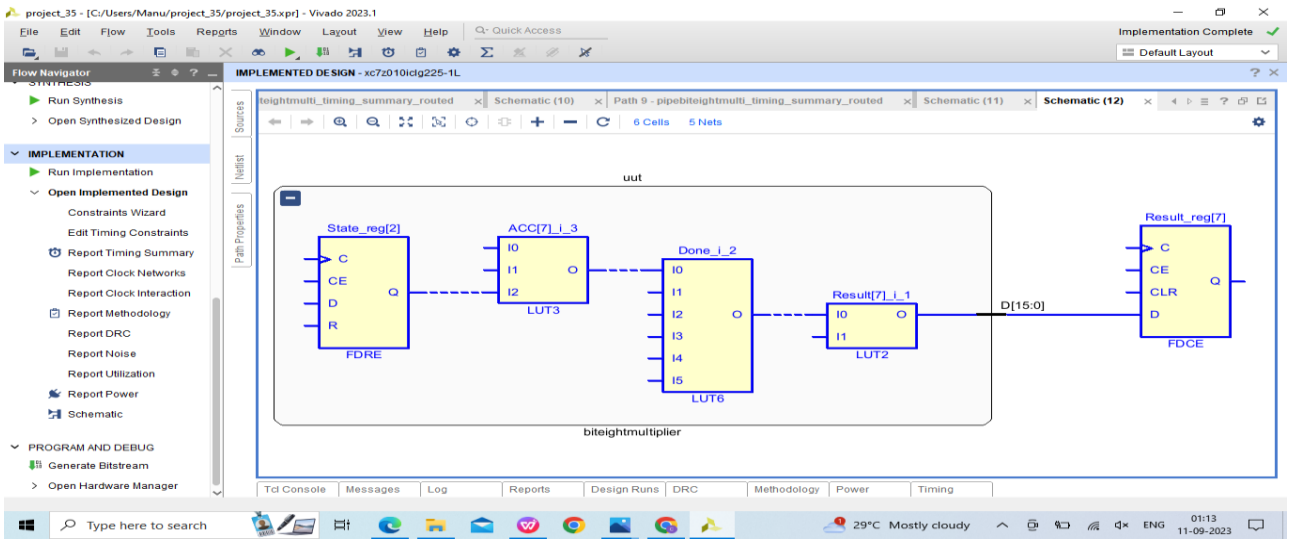
Slack = Required time - Arrival time
= 14.864 – 8.955
= **5.909ns**



10. Path-10 :

Source(Launch FF): uut/ACC_reg[2]/C
Destination (Capture FF): uut/ACC_reg[7]/C
Source clock path: clock clk rise edge + net + buffer + FDCE (Prop_fdre_C) = 4.886ns
Data path : FDRE (Prop_fdre_C_Q) + net + LUT 3+net+LUT6+net+ LUT2 = 4.067ns.
Arrival time : 4.886 + 4.067 = **8.953ns.**
Required time : clock clk rise edge + net + buff + cp + cu = **14.863ns.**

Slack = Required time - Arrival time
= 14.863 – 4.067
= **5.910ns**



Conclusion :

The goal of the project,first understanding the concept of add and shift to design an 8-bit multiplier , was achieved. The multiplier was designed, coded in VHDL and simulated using the Xilinx Vivado tools. As an added value to the project, several designs were implemented in order to compare speed and area. The design can also expanded to a 32-bit version .

References:

1. Digital Systems Design Using Verilog Charles H. Roth, Jr., Lizy Kurian John, and Byeong Kil Lee

2.ChatGPT https://chat.openai.com/