ASSIGNMENT

16×16 FIXED POINT BOOTH'S MULTIPLICATION

Master of Technology

In The
Faculty Of Engineering

By

Ganesh Kumar Shaw

Guided By

Prof. Chetan Singh Thakur

Department of Electronic Systems Engineering

Indian Institute Of Science, Bangalore

October 2021

Contents

List of Figures

Chapter 1

Pre-study

Multiplication, the process of repeated addition is one of the earliest developments of mathematics. It is commutative and distributive over addition and subtraction. There are several computational approaches for multiplication. The two numbers subject to multiplication are called multiplicand and multiplier. Thus the result of multiplication is the number (product) that would be obtained by adding the multiplicand multiplier number of times. Booth's algorithm multiplies two signed binary numbers in two's complement notation. The algorithm was proposed by A.D Booth in 1951.

1.1 Background

In this project I am going to design 16×16 signed fixed point multiplication based on Booth's Algorithm in which shift and adder method would be performed in a *single clock* cycle so that speed can be improved.

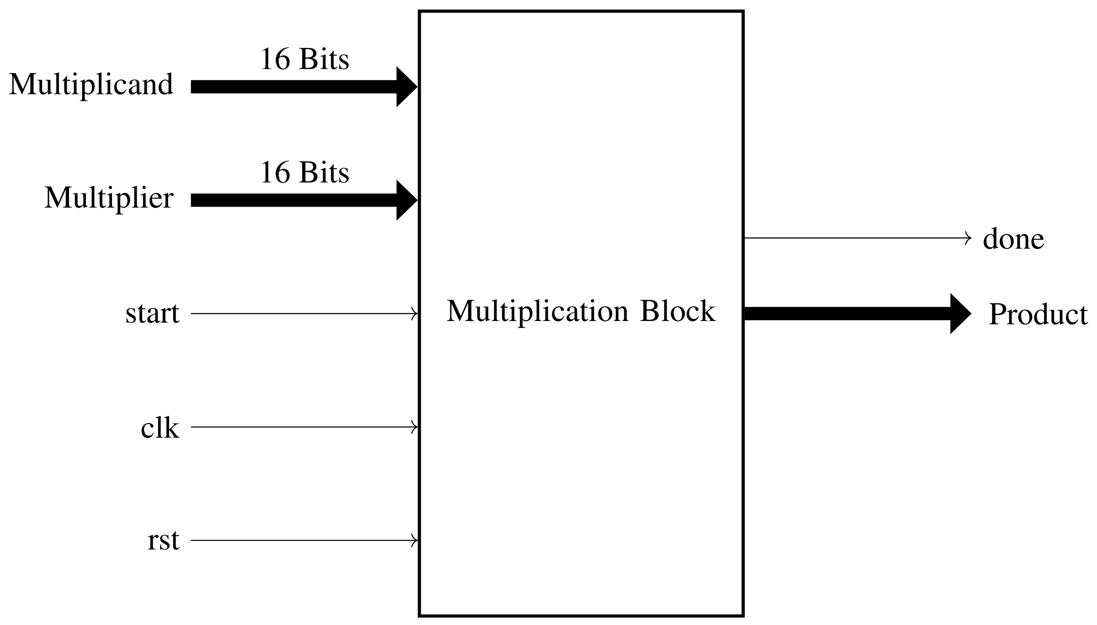


Figure 1.1: Block

Chapter 2

Design

2.1 Flow Chart

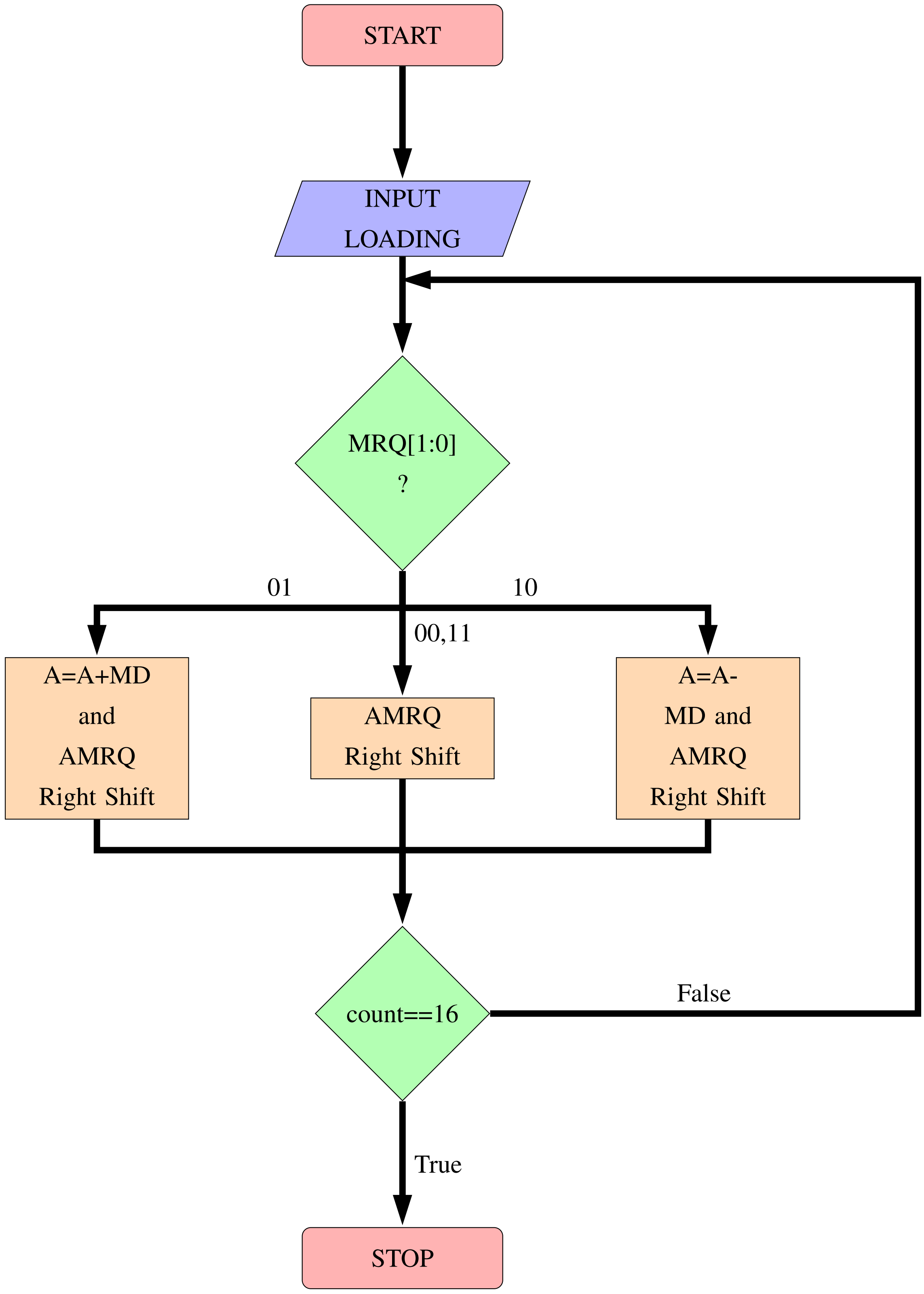


Figure 2.1: Caption

2.2 Data Path

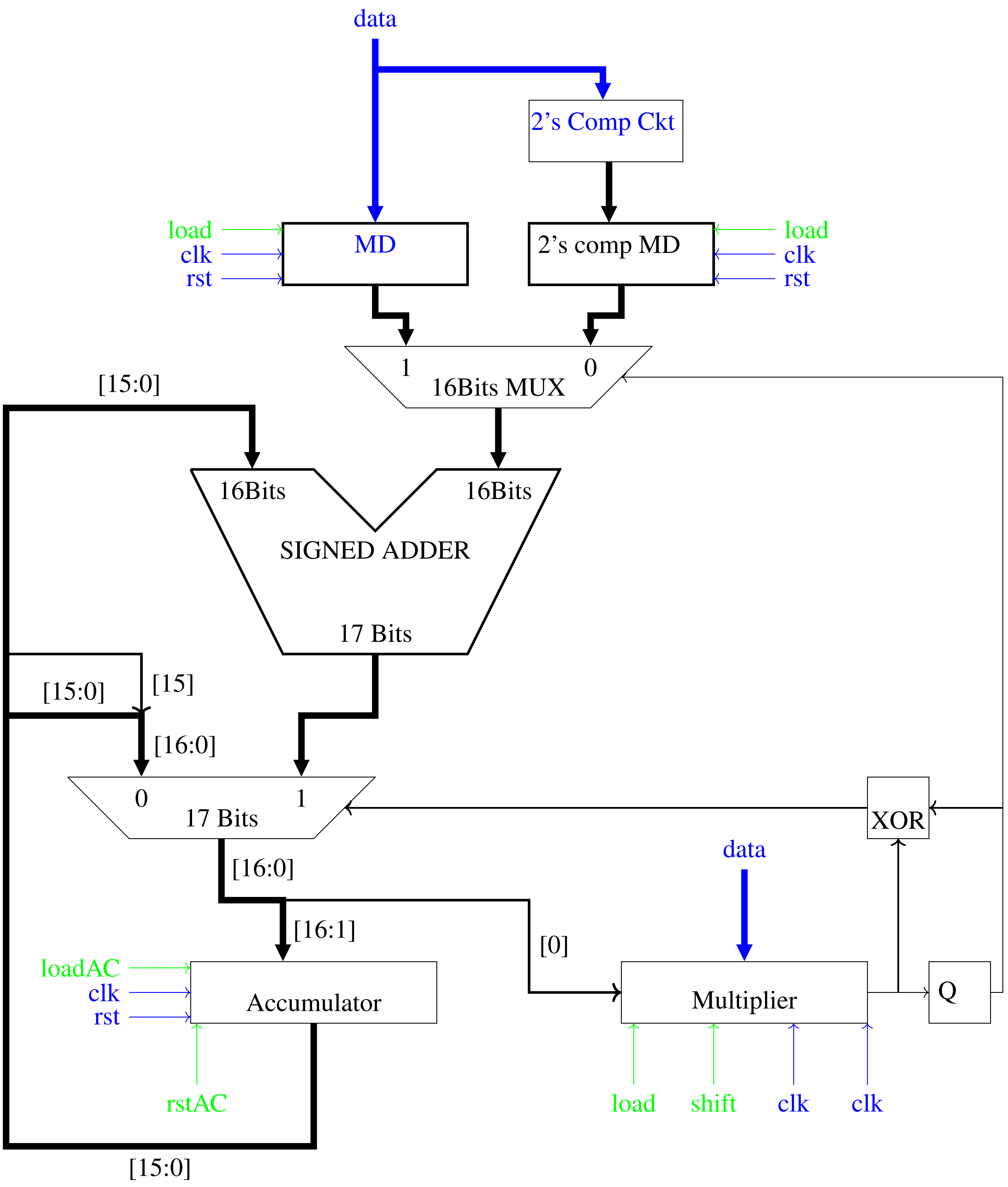


Figure 2.2: Data Path

2.3 Controller

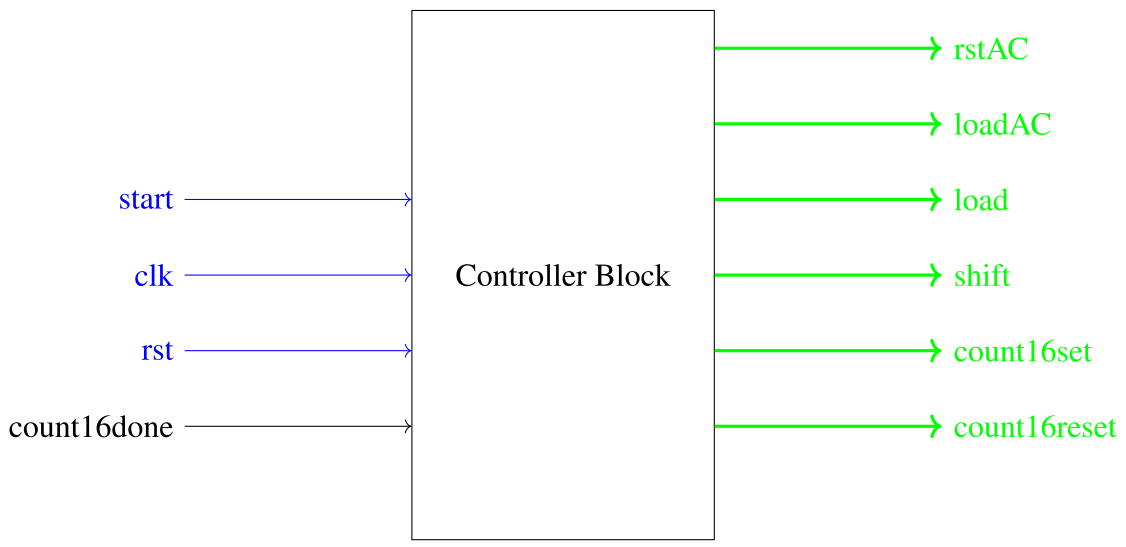


Figure 2.3: Controller

2.4 State Diagram

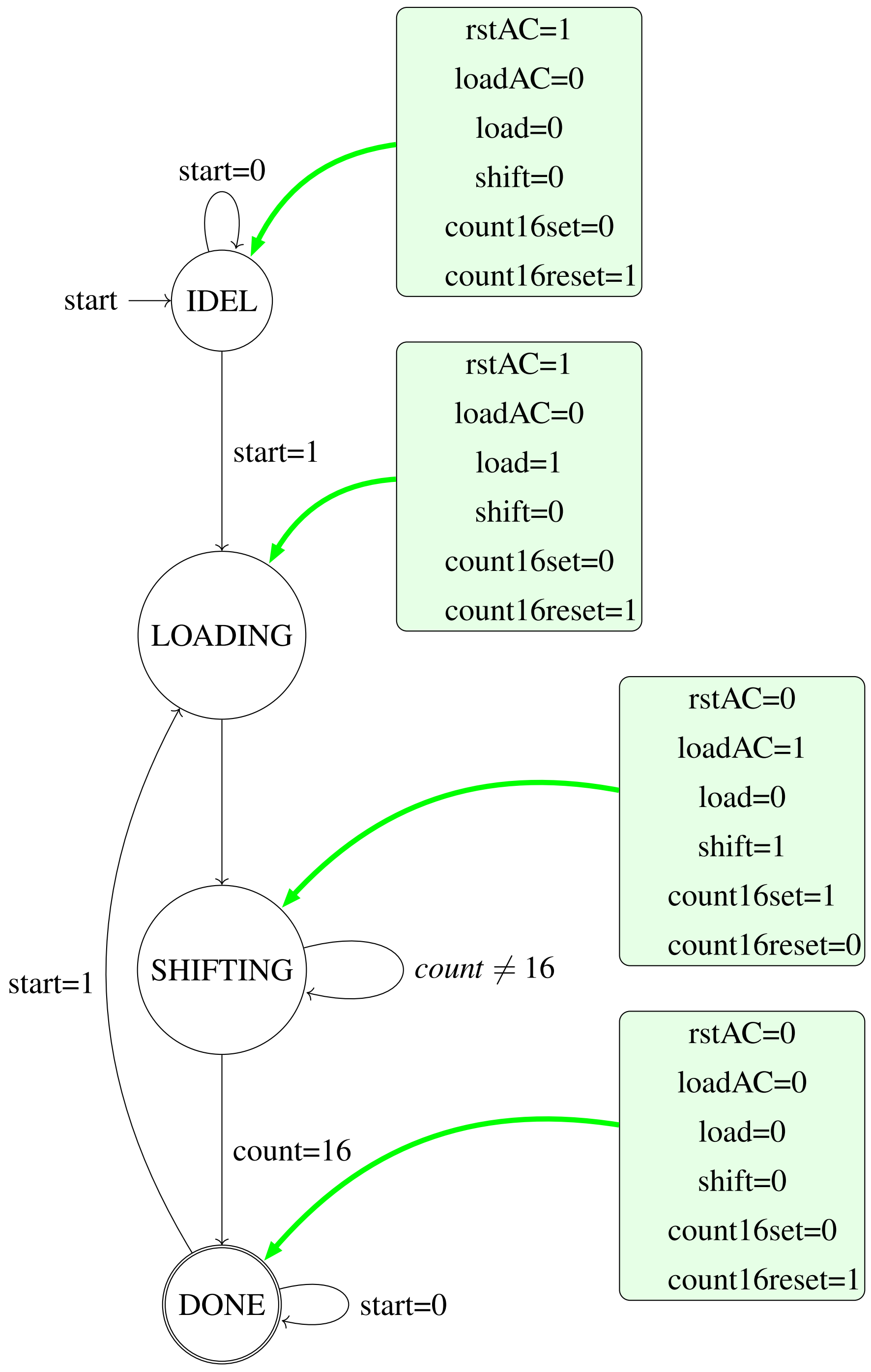


Figure 2.4: State Diagram

2.5 Timing Wave Form

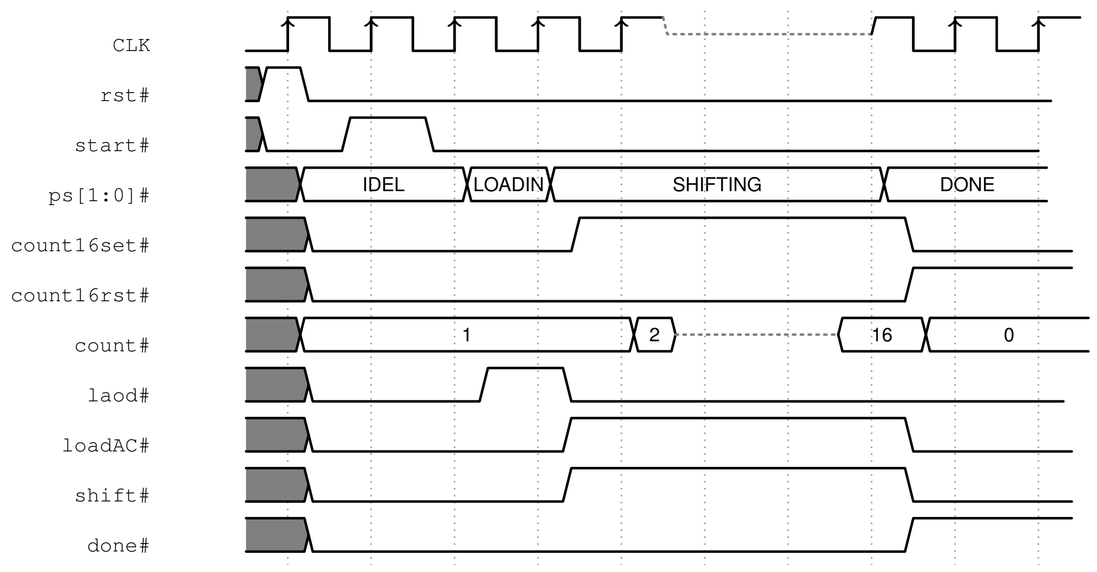


Figure 2.5: Timing Waveform

2.6 Verilog Code

- Verilog Code for Data Path

```
`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////


module DATAPATH(Product,
                done,
                clk,
                multiplicand,
                multiplier,
                start,
                rst);
parameter n=16,     //n:multiplicand width
          m=16 ;   // m:multiplier width
output[m+n-1:0] Product;
output reg done;
input[n:0] multiplicand;
input[m:0] multiplier;
input clk,
      start,
      rst;
// controller wire instantiation
wire cr_rstAC,
     cr_loadAC,
     cr_load,
     cr_shift,
     cr_count16set,
     cr_count16reset,
     cr_clk,
     cr_start,
     cr_count16done,
     cr_rst;

// wire instantiation for counter
wire ct_count16done,
     ct_clk,
     ct_count16set,
     ct_count16reset,
     ct_rst;

// 2's complement wire instantiation
wire[n-1:0] c2s_data_out,
            c2s_data_in;
// multiplicand register wire instantiaiton
wire[n-1:0] md_data_out;
wire[n-1:0] md_data_in;
wire md_clk,
     md_load,
     md_rst;
// 2's multiplicand wire instantition
wire[n-1:0] md2s_data_out;
wire[n-1:0] md2s_data_in;
wire md2s_clk,
     md2s_load,
     md2s_rst;
// mux1 wire instantiation
wire[n-1:0] mx1_data_out,
            mx1_data0,
            mx1_data1;
wire mx1_sel;

// adder wire instantiation
wire[n:0] add_sum;
wire[n-1:0] add_dataL,
            add_dataR;
// mux2 wire instantiation
wire[n:0] mx2_data_out,
          mx2_data0,
          mx2_data1;
wire mx2_sel;

// accumulator wire instantiation
wire[n-1:0] ac_data_out,
            ac_data_in;
wire ac_clk,
     ac_loadAC,
     ac_rstAC,
     ac_rst;
// multiplier wire instantiation
wire[m-1:0] mr_data_in;
```

```
wire[m-1:0] mr_data_out;
wire mr_clk,
     mr_load,
     mr_shift_data,
     mr_shift,
     mr_rst;


reg Q;
wire x;

// controller instantiaiton
CONTROLLER cr(.load(cr_load),
                  .loadAC(cr_loadAC),
                  .rstAC(cr_rstAC),
                  .shift(cr_shift),
                  .count16set(cr_count16set),
                  .count16reset(cr_count16reset),

                  .clk(cr_clk),
                  .start(cr_start),
                  .rst(cr_rst),
                  .count16done(cr_count16done));

assign cr_clk=clk;
assign cr_start=start;
assign cr_count16done=ct_count16done;
assign cr_rst=rst;


// counter instantition
COUNTER16 ct(.count16done(ct_count16done), // one bit output
                 .clk(ct_clk),
                 .count16set(ct_count16set),  // one bit input
                 .count16reset(ct_count16reset),
                 .rst(ct_rst));
assign ct_clk=clk;
assign ct_count16set=cr_count16set;
assign ct_count16reset=cr_count16reset;
assign ct_rst=rst;


// 2's complement instantiation
COMP2S comp2s(.data_out(c2s_data_out),
              .data_in(c2s_data_in));
assign c2s_data_in=multiplicand;


// multiplicand instantiaiton
PIPO md(.data_out(md_data_out),
            .clk(md_clk),
            .data_in(md_data_in),
            .load(md_load),
```

```verilog
            .rst(md_rst));
assign md_clk=clk;
assign md_data_in=multiplicand;
assign md_load=cr_load;
assign md_rst=rst;


// 2's multiplicand instantiation
PIPO md2s(.data_out(md2s_data_out),
            .clk(md2s_clk),
            .data_in(md2s_data_in),
            .load(md2s_load),
            .rst(md2s_rst));
assign md2s_clk=clk;
assign md2s_data_in=c2s_data_out;
assign md2s_load=cr_load;
assign md2s_rst=rst;


// mux1 instantiation
MUX16BITS mx1(.data_out(mx1_data_out),
                 .data0(mx1_data0),
                 .data1(mx1_data1),
                 .sel(mx1_sel));
assign mx1_data0=md2s_data_out;
assign mx1_data1=md_data_out;
assign mx1_sel=Q;


// adder instantiation
ADDER add(.sum(add_sum),
             .dataL(add_dataL),
             .dataR(add_dataR));
assign add_dataL=ac_data_out;
assign add_dataR=mx1_data_out;


// mux2 instantition i,e 17 Bits
MUX17BITS mx2(.data_out(mx2_data_out),
                 .data0(mx2_data0),
                 .data1(mx2_data1),
                 .sel(mx2_sel));
assign mx2_data0={ac_data_out[n-1],ac_data_out};
assign mx2_data1=add_sum;
assign mx2_sel=x;


// accumulator instantiation
SPIPO ac(.data_out(ac_data_out),
             .clk(ac_clk),
             .data_in(ac_data_in),
             .load(ac_loadAC),
             .rst(ac_rst),
              .Srst(ac_rstAC));
assign ac_clk=clk;
assign ac_data_in=mx2_data_out[n:1];
assign ac_loadAC=cr_loadAC;
assign ac_rst=rst;
assign ac_rstAC=cr_rstAC;


// multiplier instantiation
PISO mr(.data_out(mr_data_out),
              .clk(mr_clk),
              .data_in(mr_data_in),
               .load(mr_load),
              .shift_data(mr_shift_data),
              .shift(mr_shift),
              .rst(mr_rst));
assign mr_clk=clk;
assign mr_data_in=multiplier;
assign mr_load=cr_load;
assign mr_shift_data=mx2_data_out[0];
assign mr_shift=cr_shift;
assign mr_rst=rst;


 // Q instantiation
always @(posedge clk)
begin
 if(rst)
  Q<=1'b0;
 else if(cr_load)
  Q<=1'b0;
 else
  Q<=mr_data_out[0];
end

// xor gate instantiation
xor(x,mr_data_out[0],Q);

// output declaration

assign Product={ac_data_out,mr_data_out};

always @(posedge clk)
if(rst)
 done<= #3 1'b0;
else
 done<= #3 cr_count16done;

endmodule
```

- Verilog Code for Controller

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////


module CONTROLLER(load,
                  loadAC,
                  rstAC,
                  shift,
                  count16set,
                  count16reset,

                  clk,
                  start,
                  rst,
                  count16done);
output reg load,
           loadAC,
           rstAC,
           shift,
           count16set,
           count16reset;
input clk,
      rst,
      start,
      count16done;
parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
reg[1:0] ps,ns;
always @(posedge clk)
begin
 if(rst)
  ps<= #3 S0;
 else
  ps<=#3 ns;
end

always @(*)
begin
case(ps)
S0: begin
     rstAC=1'b1;
     loadAC=1'b0;
     load=1'b0;
     shift=1'b0;
     count16set=1'b0;
     count16reset=1'b1;
     if(start)
      ns=S1;
     else
      ns=S0;
    end
S1: begin
     rstAC=1'b1;
     loadAC=1'b0;
     load=1'b1;
     shift=1'b0;
     count16set=1'b0;
     count16reset=1'b1;

     ns=S2;
    end
S2: begin
     rstAC=1'b0;
     loadAC=1'b1;
     load=1'b0;
     shift=1'b1;
     count16set=1'b1;
     count16reset=1'b0;
     if(count16done)
      ns=S3;
     else
      ns=S2;
    end
S3: begin
     rstAC=1'b0;
     loadAC=1'b0;
     load=1'b0;
     shift=1'b0;
     count16set=1'b0;
     count16reset=1'b1;
     if(start)
      ns=S1;
     else
      ns=S3;
    end
default: begin
          rstAC=1'b1;
          loadAC=1'b0;
          load=1'b0;
          shift=1'b0;
          count16set=1'b0;
          count16reset=1'b1;

          ns=S0;
         end
endcase
end

endmodule

//`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////


```

```
//module CONTROLLER(rstAC,
//                  loadAC,
//                  load,
//                   shift,
//                   count16set,
//                   count16reset,
//                   clk,
//                   start,
//                   count16done,
//                   rst);
// output reg rstAC,
//        loadAC,
//        load,
//        shift,
//        count16set,
//        count16reset;
// input  clk,
//        start,
//        count16done,
//        rst;
//parameter n=16, // multiplicand width size
//          m=16; // mulitplier width size
//parameter[1:0] S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
//reg[1:0] ps,ns;
//always @(posedge clk)
//begin
// if(rst)
//  ps<=2'b00;
// else
//  ps<=ns;
//end

//always @(*)
// case(ns)
//S0: begin
//     if(start)
//      begin
//       rstAC=1'b1;
//       loadAC=1'b0;
//       load=1'b1;
//       shift=1'b0;
//       count16set=1'b0;
//       count16reset=1'b1;
//      end
//     else
//      begin
//       rstAC=1'b1;
//       loadAC=1'b0;
//       load=1'b0;
//       shift=1'b0;
//       count16set=1'b0;
//       count16reset=1'b1;
//      end
//    end
//S1: begin
//       rstAC=1'b0;
//       loadAC=1'b1;
//       load=1'b0;
//       shift=1'b1;
//       count16set=1'b1;
//       count16reset=1'b0;
//    end
//S2: begin
//     if(count16done)
//      begin
//       rstAC=1'b0;
//       loadAC=1'b0;
//       load=1'b0;
//       shift=1'b0;
//       count16set=1'b0;
//       count16reset=1'b0;
//      end
//     else
//      begin
//       rstAC=1'b0;
//       loadAC=1'b1;
//       load=1'b0;
//       shift=1'b1;
//       count16set=1'b1;
//       count16reset=1'b0;
//      end
//    end
//S3: begin
//     if(start)
//      begin
//       rstAC=1'b1;
//       loadAC=1'b0;
//       load=1'b1;
//       shift=1'b0;
//       count16set=1'b0;
//       count16reset=1'b1;
//      end
//     else
//      begin
//       rstAC=1'b0;
//       loadAC=1'b0;
//       load=1'b0;
//       shift=1'b0;
//       count16set=1'b0;
//       count16reset=1'b0;
//      end
//    end
//default: begin
//       rstAC=1'b1;
//       loadAC=1'b0;
//       load=1'b0;
//       shift=1'b0;
//       count16set=1'b0;
//       count16reset=1'b1;
//         end
// endcase

// always @(*)
//if(rst)
//  ns=S0;
//else
// case(ps)
//S0: begin
//     if(start)
//      ns=S1;
//     else
//      ns=S0;
//    end
//S1: begin
//     ns=S2;
//    end
//S2: begin
//     if(count16done)
//      ns=S3;
//     else
//      ns=S2;
//    end
//S3: begin
//     if(start)
//      ns=S1;
//     else
//      ns=S3;
//    end
//default: begin
//          ns=S0;
//         end
// endcase

//endmodule
```

- Verilog code for counter

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////


module COUNTER16(count16done, // one bit output
                 clk,
                 count16set,  // one bit input
                 count16reset,
                 rst);
```

```
output  count16done;
input count16set,
      count16reset,
      rst,
      clk;

reg[3:0] count;
wire[3:0] wire_in;
wire[4:0] wire_out;

assign wire_out={1'b0,wire_in}+1;
assign wire_in=count;
always @(posedge clk)
begin
 if(rst)
  count<=#3 4'b0000;
 else if(count16reset)
  count<=#3 4'b0000;
 else if(count16set)
  count<=#3 wire_out[3:0];
end
// output declaration
assign count16done=wire_out[4];

endmodule
```

- Verilog code for 2's complement ckt

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////


module COMP2S(data_out,
              data_in);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size
output[n-1:0] data_out;
input[n-1:0] data_in;
wire[n:0] comp;
assign comp=17'b1000_0000_0000_00000-{data_in[n-1],data_in};
assign data_out=comp[n-1:0];
endmodule
```

- Paralle in Parallel out 16 Bits Mulitplicand Register Verilog Code

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

module PIPO(data_out,
            clk,
            data_in,
            load,
            rst);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size
output reg[n-1:0] data_out;
input[n-1:0] data_in;
input clk,
      rst,
      load;

always @(posedge clk)
begin
 if(rst)
  data_out<=16'b0;
 else if(load)
  data_out<=data_in;
 else
  data_out<=data_out;
end

endmodule
```

- Paralle in Parallel out 16 Bits Accumulator Register Verilog Code

```
`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////////


module SPIPO(data_out,
             clk,
             data_in,
             load,
             rst,
             Srst);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size
output reg[m-1:0] data_out;
input[m-1:0] data_in;
input clk,
      rst,
      load,
      Srst;
always @(posedge clk)
begin
 if(rst)
  data_out<=16'b0;
 else if(Srst)
  data_out<=16'b0;
 else if(load)
  data_out<=data_in;
 else
  data_out<=data_out;
end
endmodule
```

- 16Bits Mux

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////


module MUX16BITS(data_out,
                 data0,
                 data1,
                 sel);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size
output[n-1:0] data_out;
input[n-1:0] data0,
             data1;
input sel;
assign data_out=sel?data1:data0;
endmodule
```

- Verilog code for Adder

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////
module ADDER(sum,
             dataL,
             dataR);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size

output[n:0] sum;
input[n-1:0] dataL,
             dataR;
wire[n-1:0] S;
assign S=dataL+dataR;
assign sum={S[n-1],S};
endmodule
```

- 17Bits Mux

```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////
```

```


module MUX17BITS(data_out,
                 data0,
                 data1,
                 sel);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size
output[n:0] data_out;
input[n:0] data0,
             data1;
input sel;
assign data_out=sel?data1:data0;
endmodule
```

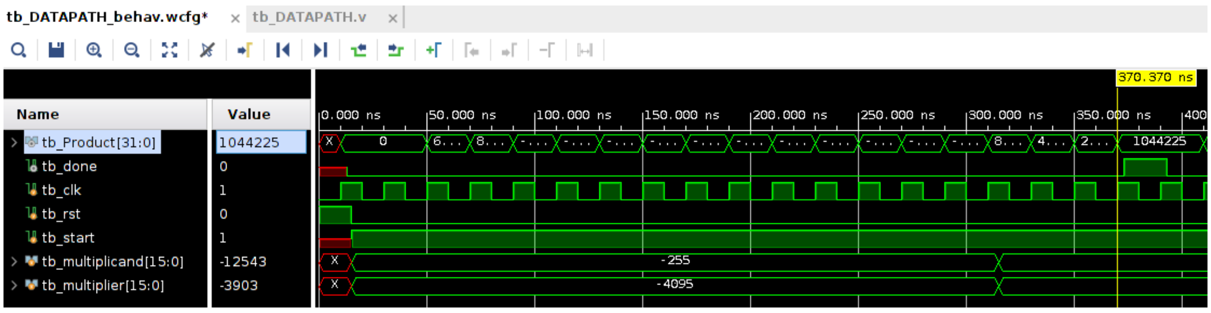
- Paralle in Parallel out 16 Bits Multiplier Register Verilog Code

```
`timescale 1ns / 1ps

/////////////////////////////////////////////////////////////////////////////////


module PISO( data_out,
             clk,
             data_in,
              load,
             shift_data,
             shift,
             rst);
parameter n=16, // multiplicand width size
          m=16; // mulitplier width size
output reg[m-1:0] data_out;
input[m-1:0] data_in;
input clk,
      load,
      shift_data,
      shift,
      rst;
always @(posedge clk)
begin
 if(rst)
  data_out<=16'b0;
 else if(load)
  data_out<=data_in;
 else if(shift)
  data_out<={shift_data,data_out[m-1:1]};
 else
  data_out<=data_out;
end

endmodule
```

- Test Bench

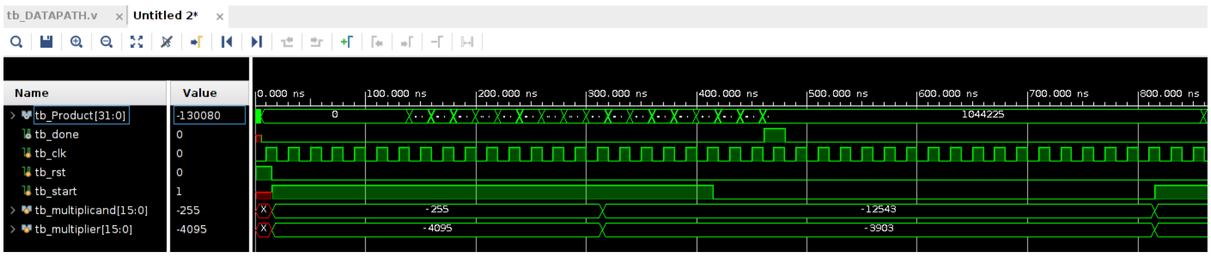
```
`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////


module tb_DATAPATH;
parameter n=16,m=16;
wire[n+m-1:0] tb_Product;
wire tb_done;
reg tb_clk,
    tb_rst,
    tb_start;
reg[15:0] tb_multiplicand,
          tb_multiplier;
DATAPATH DUT(.Product(tb_Product),
                .done(tb_done),
                .clk(tb_clk),
                .multiplicand(tb_multiplicand),
                .multiplier(tb_multiplier),
                .start(tb_start),
                .rst(tb_rst));

initial tb_clk=1'b0;
always #10 tb_clk=~tb_clk;
integer i;
initial
begin
tb_rst=1'b1;
#15 tb_rst=1'b0;tb_start=1'b1;
$monitor($time,"tb_Product=%b",tb_Product);
for(i=1;i<=10;i=i+1)
 begin
 tb_start=1'b1;
 tb_multiplicand=16'b1111_1111_0000_0001;//{$random}%65535;
 tb_multiplier=16'b1111_0000_0000_0001;//{$random}%1599;
 #300 tb_multiplicand=16'b1100_1111_0000_0001;//{$random}%65535;
  tb_multiplier=16'b1111_0000_1100_0001;//{$random}%1599;
 #100 tb_start=1'b0;
 #400;

 end

 $finish;
end
endmodule
```

2.7 Report



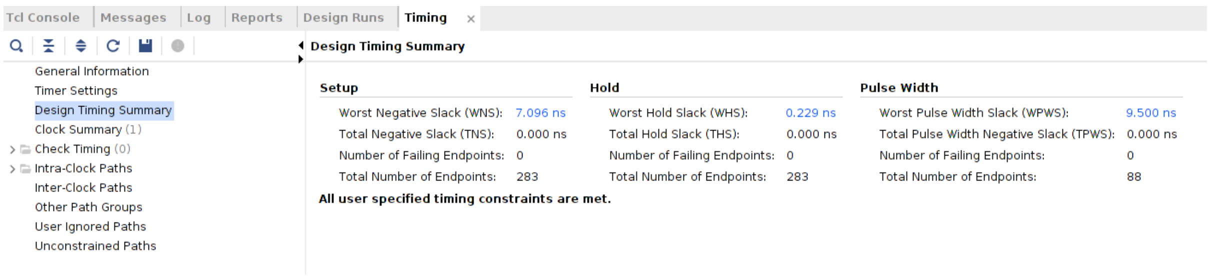
Name	Value
tb_Product[31:0]	1044225
tb_done	0
tb_clk	1
tb_rst	0
tb_start	1
tb_multiplicand[15:0]	-12543
tb_multiplier[15:0]	-3903

Figure 2.6: Behavioural Timing Simulation



Name	Value
tb_Product[31:0]	-130080
tb_done	0
tb_clk	0
tb_rst	0
tb_start	1
tb_multiplicand[15:0]	-255
tb_multiplier[15:0]	-4095

Figure 2.7: Post Implementation Timing Simulation

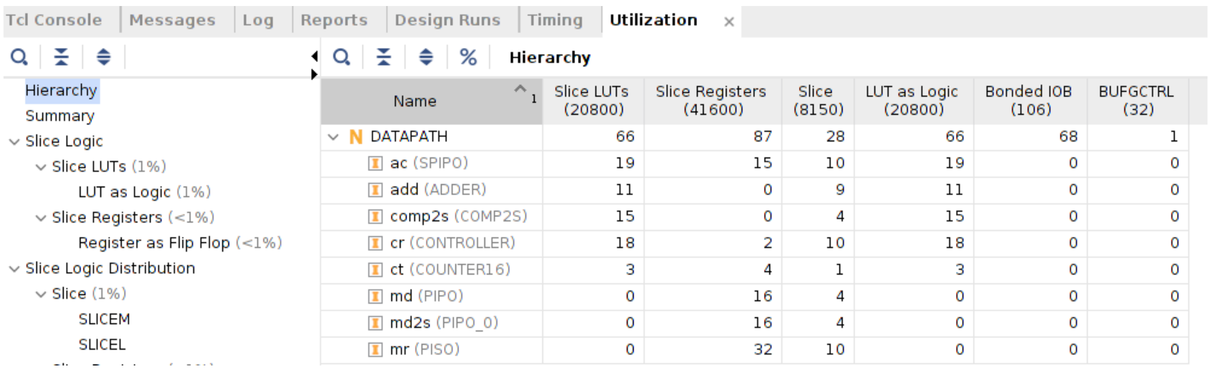


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.096 ns	Worst Hold Slack (WHS):	0.229 ns	Worst Pulse Width Slack (WPWS):	9.500 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:	283	Total Number of Endpoints:	283	Total Number of Endpoints:	88

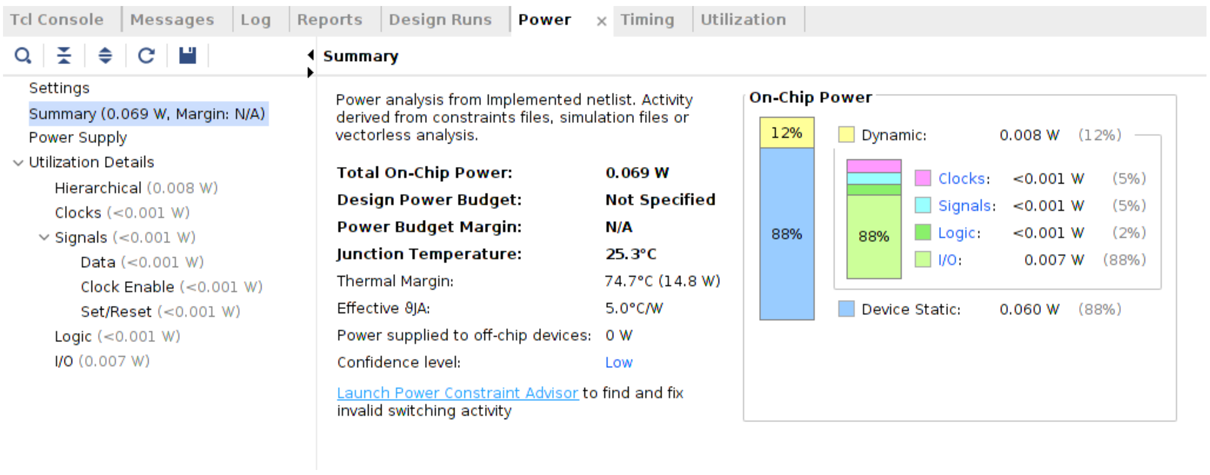
All user specified timing constraints are met.

Figure 2.8: Timning Constraints



Name	Slice LUTs (20800)	Slice Registers (41600)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)	BUFGCTRL (32)
DATAPATH	66	87	28	66	68	1
ac (SPIPO)	19	15	10	19	0	0
add (ADDER)	11	0	9	11	0	0
comp2s (COMP2S)	15	0	4	15	0	0
cr (CONTROLLER)	18	2	10	18	0	0
ct (COUNTER16)	3	4	1	3	0	0
md (PIPO)	0	16	4	0	0	0
md2s (PIPO_0)	0	16	4	0	0	0
mr (PISO)	0	32	10	0	0	0

Figure 2.9: Utilization



Settings
Summary (0.069 W, Margin: N/A)
Power Supply
Utilization Details
Hierarchical (0.008 W)
Clocks (<0.001 W)
Signals (<0.001 W)
Data (<0.001 W)
Clock Enable (<0.001 W)
Set/Reset (<0.001 W)
Logic (<0.001 W)
I/O (0.007 W)

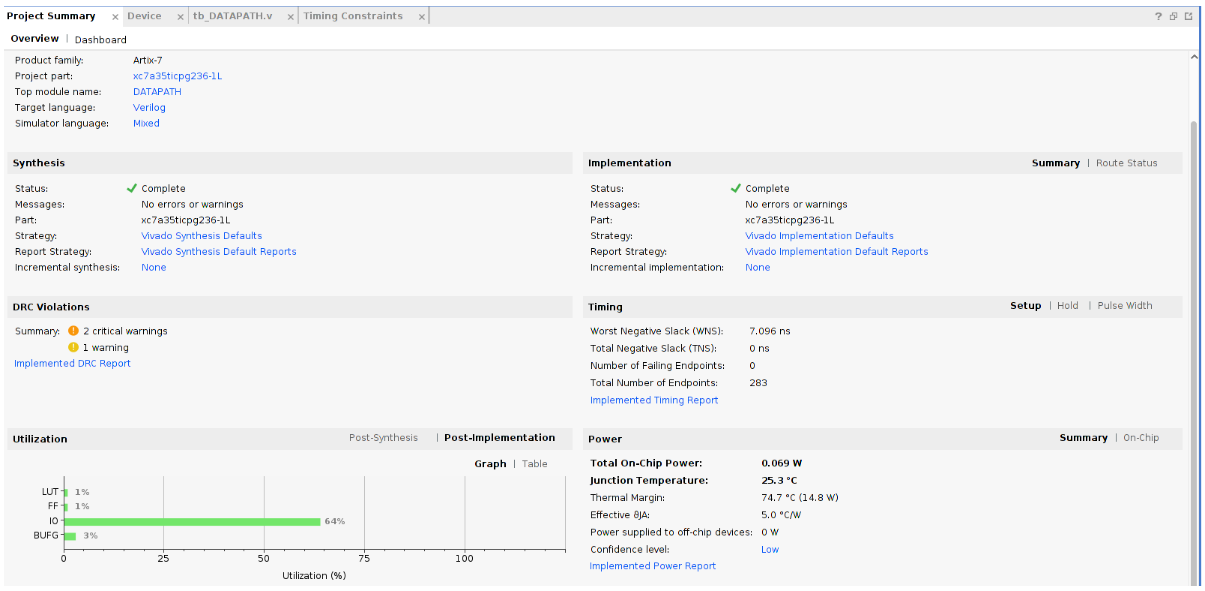
Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.069 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.3°C**
Thermal Margin:	74.7°C (14.8 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

🟨	Dynamic:	0.008 W	(12%)
🟪	Clocks:	<0.001 W	(5%)
🟦	Signals:	<0.001 W	(5%)
🟩	Logic:	<0.001 W	(2%)
🟩	I/O:	0.007 W	(88%)
🟦	Device Static:	0.060 W	(88%)

Figure 2.10: Power



Product family:	Artix-7
Project part:	xc7a35ticpg236-1L
Top module name:	DATAPATH
Target language:	Verilog
Simulator language:	Mixed

Synthesis

Status:	✓ Complete
Messages:	No errors or warnings
Part:	xc7a35ticpg236-1L
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports
Incremental synthesis:	None

Implementation

Status:	✓ Complete
Messages:	No errors or warnings
Part:	xc7a35ticpg236-1L
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental implementation:	None

DRC Violations

Summary: 2 critical warnings
1 warning
Implemented DRC Report

Timing

Worst Negative Slack (WNS):	7.096 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	283

Implemented Timing Report

Utilization

Power

Total On-Chip Power:	**0.069 W**
Junction Temperature:	**25.3 °C**
Thermal Margin:	74.7 °C (14.8 W)
Effective ϑJA:	5.0 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

Figure 2.11: Project Summery

- Maximum frequency calculation

  $T_{clk} = 20ns$ Considered in the test bench

  $T_{slak} = 7.096ns$ shown in the fig 2.8

  Therefore, $T_{min} = T_{clk} - T_{slack}$

  $T_{min} = 12.904ns$ // Hence $f_{max} \approx 77MHz$

  Latency=17 clock cycle

Bibliography