ASSIGNMENT

16 × 16 FIXED POINT MULTIPLICATION AND IT'S PIPELINING

Master of Technology

IN THE
FACULTY OF ENGINEERING

BY

GANESH KUMAR SHAW

GUIDED BY

PROF. KURUVILLA VARGHESE

DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING
INDIAN INSTITUTE OF SCIENCE, BANGALORE

FEBRUARY 2022

Contents

List of Figures

Chapter 1

Pre-study

In recent years, power consumption, as well as area and speed, are the most important issues in VLSI design. Passtransistor logic has been intensively studied as a breakthrough for high-speed and low-power digital circuits. Most modern arithmetic processors are built with architectures that have been well-established in the literature, with many of the latest innovations devoted to special logic circuits and the use of advanced technologies. Specifically, the design of multipliers is critical in digital signal processing applications, where a high number of multiplications are required.

1.1 Background

In this project I am going to design 16$\times$ 16 unsigned fixed point multiplication based on shift and adder method in which It would be tried to perform addition and shift operation in a *single clock* so that speed can be improved.

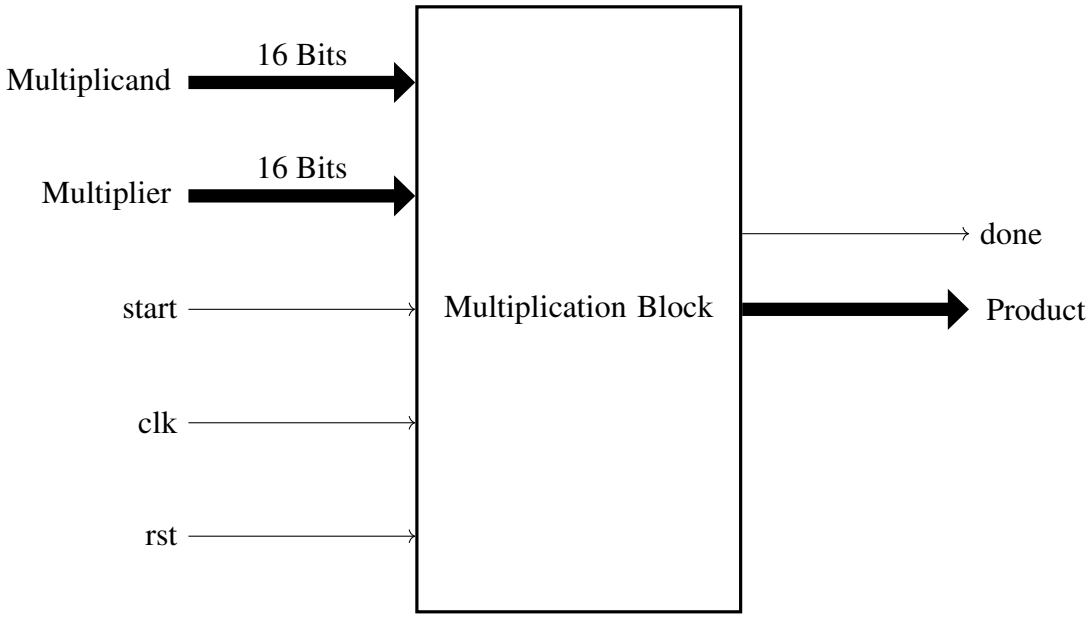

Figure 1.1: Block

Chapter 2

Design

2.1 Flow Chart

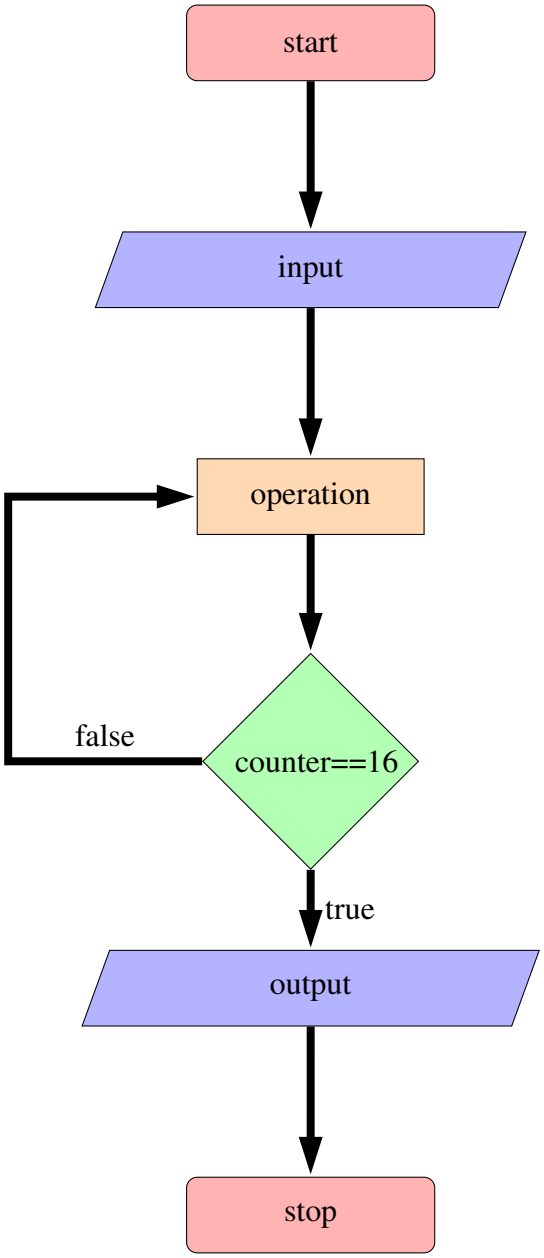

Figure 2.1: Flow Chart

2.2 Data Path

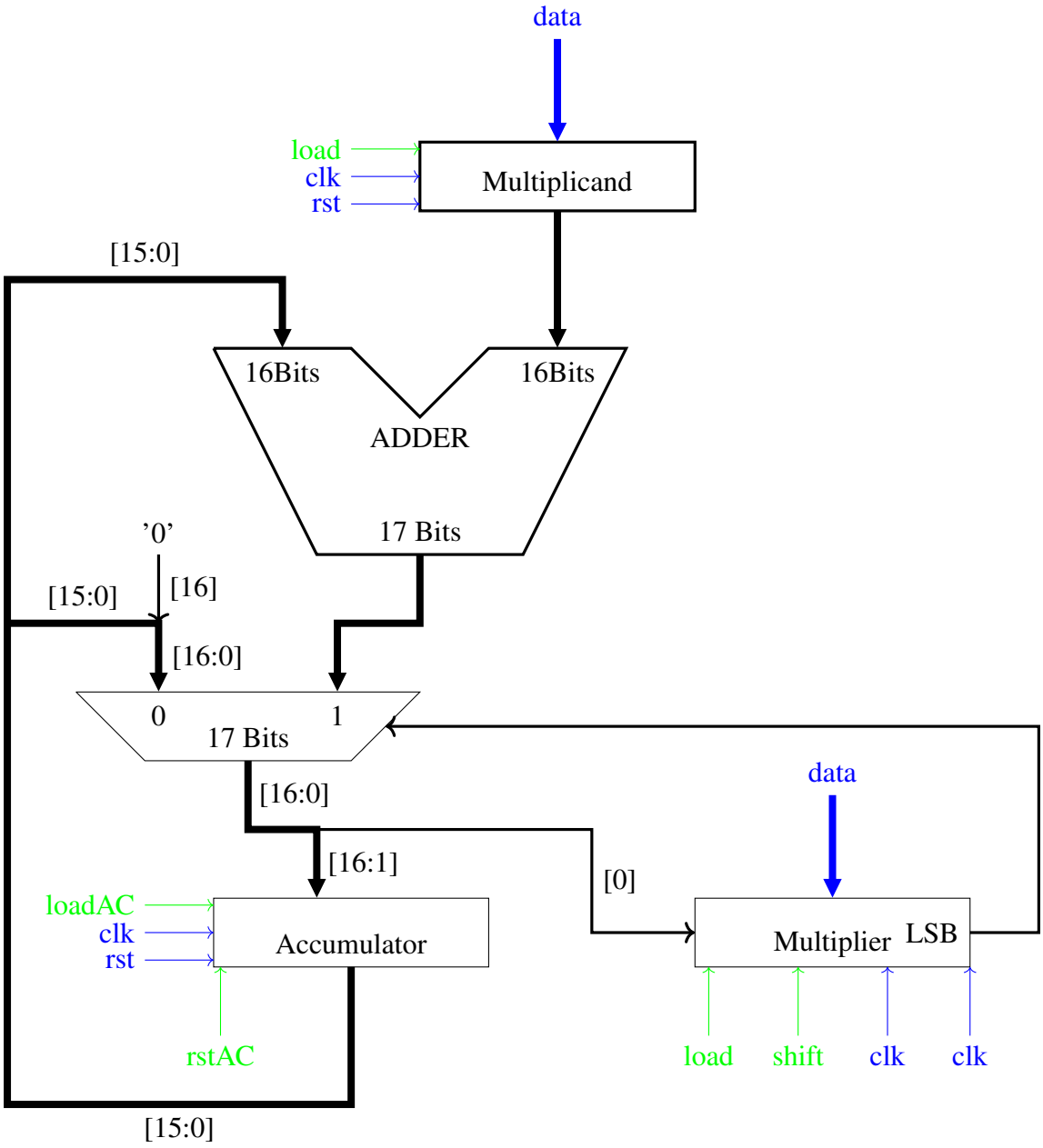

Figure 2.2: Data Path

2.3 Controller

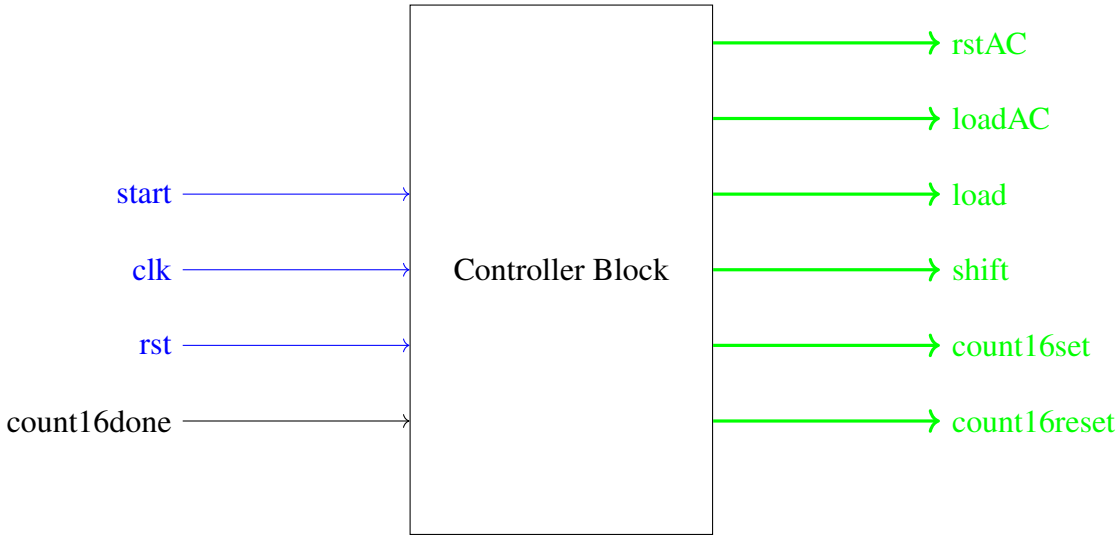

Figure 2.3: Controller

2.4 State Diagram

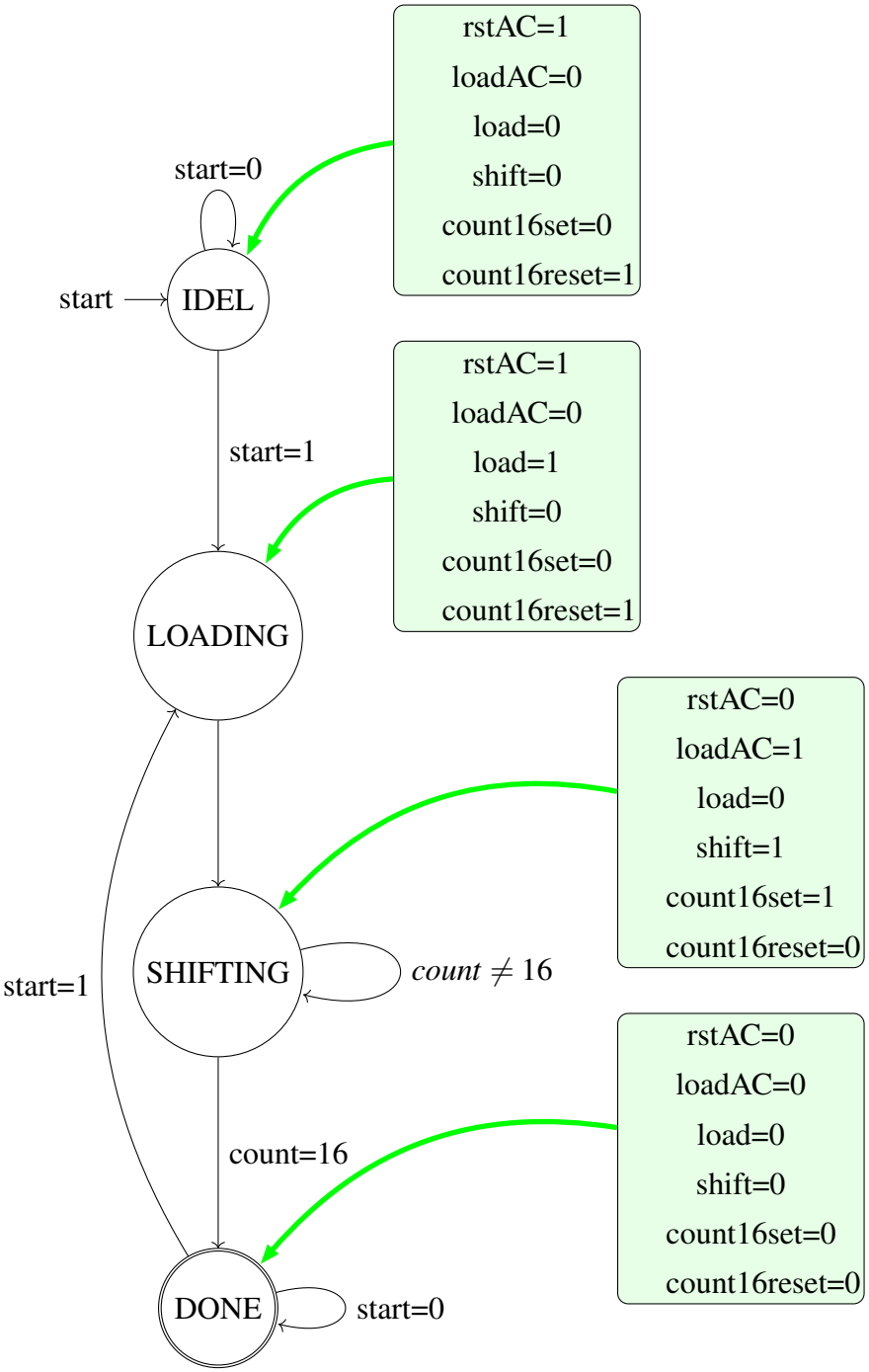

Figure 2.4: State Diagram

2.5 Timing Wave Form

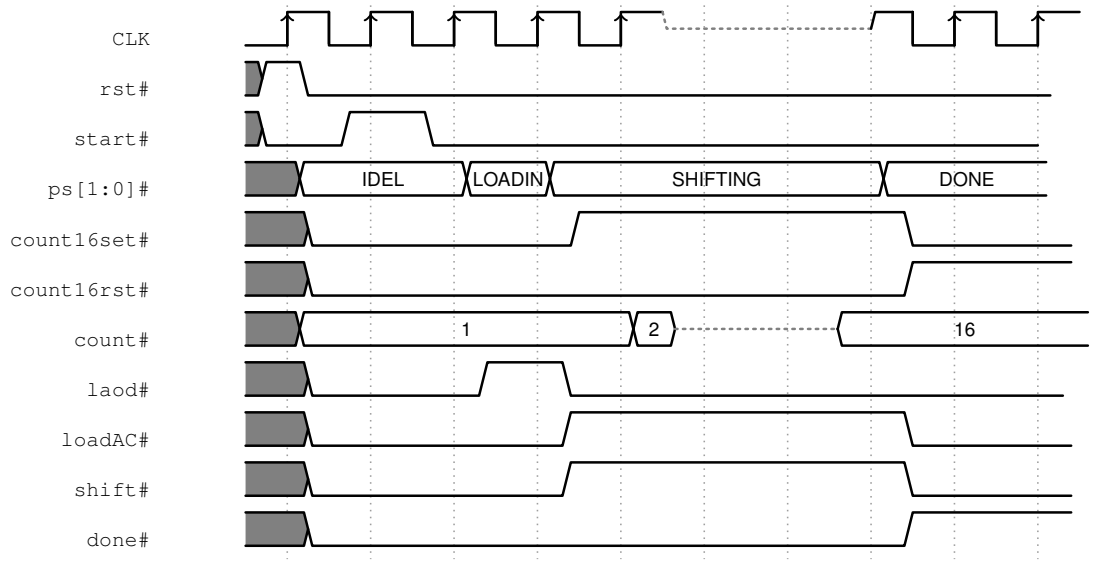

Figure 2.5: Timing Waveform

2.6 Verilog Code

- Verilog Code for Data Path

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  //////////////////////////////////////////////////////////////////////////////////
4
5
6  module DATAPATH(Product,
7                  done,
8                  clk,
9                  multiplicand,
10                 multiplier,
11                 start,
12                 rst);
13 parameter n=16,     //n:multiplicand width
14           m=16 ;  // m:multiplier width
15 output[m+n-1:0] Product;
16 output reg done;
17 input[n:0] multiplicand;
18 input[m:0] multiplier;
19 input clk,
20       start,
21       rst;
22
23 // controller wire instantiation
24 wire cr_load,
25      cr_loadAC,
26      cr_rstAC,
27      cr_shift,
```

```
28        cr_count16set,
29        cr_count16reset,
30        cr_clk,
31        cr_start,
32        cr_rst,
33        cr_count16done;
34  //counter wire instantiation
35  wire ct_counter16done,
36        ct_count16set,
37        ct_count16reset,
38        ct_rst,
39        ct_clk;
40
41  // multiplicand register wire instantiation
42  wire[n-1:0] md_data_out,
43              md_data_in;
44  wire md_clk,
45       md_load,
46       md_rst;
47  // adder wire instantiation
48  wire[n:0] add_data_out;
49  wire[n-1:0] add_data0,
50              add_data1;
51  // mux wire instantiatiion
52  wire[n:0] mux_data_out,
53            mux_data0,
54            mux_data1;
55  // accumulator wire instantiation
56  wire[n-1:0] ac_data_out,
57            ac_data_in;
58  wire ac_clk,
59       ac_rstAC,
60       ac_loadAC,
61       ac_rst;
62  // multiplier wire instantiation
63  wire[m-1:0] mr_data_out,
64              mr_data_in;
65  wire mr_clk,
66       mr_shift_in,
67       mr_load,
68       mr_shift,
69       mr_rst;
70  // controller instantiation
71  CONTROLLER controller(.load(cr_load),
72                        .loadAC(cr_loadAC),
73                        .rstAC(cr_rstAC),
74                        .shift(cr_shift),
75                        .count16set(cr_count16set),
76                        .count16reset(cr_count16reset),
77
78                        .clk(cr_clk),
79                        .start(cr_start),
80                        .rst(cr_rst),
```

```
                    .count16done(cr_count16done));
assign cr_clk=clk;
assign cr_rst=rst;
assign cr_start=start;
assign cr_count16done=ct_count16done;


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


// multiplicand register  instantiation
PIPO16BITS md(.data_out(md_data_out),
                  .clk(md_clk),
                  .data_in(md_data_in),
                  .load(md_load),
                  .rst(md_rst));
assign md_clk=clk;
assign md_data_in=multiplicand;
assign md_load=cr_load;
assign md_rst=rst;


// adder  instantiation
ADDER16BIT adder(.data_out(add_data_out),
                  .data0(add_data0),
                  .data1(add_data1));
assign add_data0=md_data_out;
assign add_data1=ac_data_out;


// mux  instantiatiion
MUX17BIT2_1 mux(.data_out(mux_data_out),
                   .data0(mux_data0),
                   .data1(mux_data1),
                   .sel(mux_sel));
assign mux_data0={1'b0,ac_data_out};
assign mux_data1=add_data_out;
assign mux_sel=mr_data_out[0];


// accumulator  instantiation
RPIPO16BITS ac(.data_out(ac_data_out),
                   .clk(ac_clk),
                   .data_in(ac_data_in),
```

```
134                     .Rrst(ac_rstAC),
135                     .load(ac_loadAC),
136                     .rst(ac_rst));
137 assign ac_clk=clk;
138 assign ac_data_in=mux_data_out[n:1];
139 assign ac_rstAC=cr_rstAC;
140 assign ac_loadAC=cr_loadAC;
141 assign ac_rst=rst;
142
143 // multiplier  instantiation
144 PISO16BITS mr(.data_out(mr_data_out),
145                 .clk(mr_clk),
146                 .data_in(mr_data_in),
147                 .shift_in(mr_shift_in),
148                 .load(mr_load),
149                 .shift(mr_shift),
150                 .rst(mr_rst));
151 assign mr_clk=clk;
152 assign mr_data_in=multiplier;
153 assign mr_shift_in=mux_data_out[0];
154 assign mr_load=cr_load;
155 assign mr_shift=cr_shift;
156 assign mr_rst=rst;
157
158 //outpur declaration
159 assign Product={ac_data_out,mr_data_out};
160
161 always @(posedge clk)
162 if(rst)
163  done<=1'b0;
164 else
165  done<=ct_count16done;
166
167 endmodule
```

- Verilog Code for Controller

```
1  `timescale 1ns / 1ps
2
3  //////////////////////////////////////////////////////////////////////////////////
4
5
6  module CONTROLLER(load,
7                    loadAC,
8                    rstAC,
9                    shift,
10                   count16set,
11                   count16reset,
12
13                   clk,
14                   start,
15                   rst,
```

```
16                    count16done);
17  output reg load,
18             loadAC,
19             rstAC,
20             shift,
21             count16set,
22             count16reset;
23  input clk,
24        rst,
25        start,
26        count16done;
27  parameter S0=2'b00,S1=2'b01,S2=2'b10,S3=2'b11;
28  reg[1:0] ps,ns;
29  always @(posedge clk)
30  begin
31   if(rst)
32    ps<=S0;
33   else
34    ps<=ns;
35  end
36
37  always @(*)
38  begin
39  case(ps)
40  S0: begin
41        rstAC=1'b1;
42        loadAC=1'b0;
43        load=1'b0;
44        shift=1'b0;
45        count16set=1'b0;
46        count16reset=1'b1;
47        if(start)
48         ns=S1;
49        else
50         ns=S0;
51      end
52  S1: begin
53        rstAC=1'b1;
54        loadAC=1'b0;
55        load=1'b1;
56        shift=1'b0;
57        count16set=1'b0;
58        count16reset=1'b1;
59
60        ns=S2;
61      end
62  S2: begin
63        rstAC=1'b0;
64        loadAC=1'b1;
65        load=1'b0;
66        shift=1'b1;
67        count16set=1'b1;
68        count16reset=1'b0;
```

```
69        if(count16done)
70         ns=S3;
71        else
72         ns=S2;
73       end
74  S3: begin
75        rstAC=1'b0;
76        loadAC=1'b0;
77        load=1'b0;
78        shift=1'b0;
79        count16set=1'b0;
80        count16reset=1'b1;
81        if(start)
82         ns=S1;
83        else
84         ns=S3;
85       end
86  default: begin
87             rstAC=1'b1;
88             loadAC=1'b0;
89             load=1'b0;
90             shift=1'b0;
91             count16set=1'b0;
92             count16reset=1'b1;
93
94             ns=S0;
95            end
96  endcase
97  end
98
99  endmodule
```

- Paralle in Parallel out 16 Bits Mulitplicand Register Verilog Code

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3
4   module PIPO16BITS(data_out,
5                     clk,
6                     data_in,
7                     load,
8                     rst);
9   parameter n=16;         // multiplicand width size
10
11  output reg[n-1:0] data_out;
12  input[n-1:0] data_in;
13  input load,
14        clk,
15        rst;
16  always @(posedge clk)
17   begin
18    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
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/30/2022 05:21:02 PM
// Design Name:
// Module Name: RPIPO
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


`timescale 1ns / 1ps

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


module RPIPO16BITS(data_out,
                   clk,
                   data_in,
                   Rrst,
                   load,
                   rst);
parameter n=16;          // multiplier width size

output reg[n-1:0] data_out;
input[n-1:0] data_in;
input load,
      Rrst,
      clk,
      rst;
always @(posedge clk)
```

```
 begin
  if(rst)
   data_out<=16'b0;
  else if(Rrst)
   data_out<=16'b0;
  else if(load)
   data_out<=data_in;
  else
   data_out<=data_out;
 end
endmodule
```

- Verilog code for Adder

```
`timescale 1ns / 1ps

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


module ADDER16BIT(data_out,
                  data0,
                  data1);
parameter n=16;   // multiplicand width size
output[n:0] data_out;
input[n-1:0] data0,
             data1;
assign data_out={1'b0,data0}+{1'b0,data1};
endmodule
```

- Paralle in Parallel out 16 Bits Multiplier Register Verilog Code

```
`timescale 1ns / 1ps

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


module PISO16BITS(data_out,
                  clk,
                  data_in,
                  shift_in,
                  load,
                  shift,
                  rst);
parameter m=16;         // multiplier width size

output reg[m-1:0] data_out;
input[m-1:0] data_in;
input load,
      shift,
      shift_in,
      clk,
```

```
21        rst;
22  always @(posedge clk)
23   begin
24    if(rst)
25     data_out<=16'b0;
26    else if(load)
27     data_out<=data_in;
28    else if(shift)
29     data_out<={shift_in,data_out[15:1]};
30    else
31     data_out<=data_out;
32   end
33  endmodule
```

- 16 Bits 2 to 1 Multiplexer Verilog Code

```
1   `timescale 1ns / 1ps
2
3   //////////////////////////////////////////////////////////////////////////////////
4
5
6   module MUX17BIT2_1(data_out,
7                      data0,
8                      data1,
9                      sel);
10  parameter n=16;  // multiplicand bits + 1
11  output[n:0] data_out;
12  input[n:0] data0,
13             data1;
14  input sel;
15
16  assign  data_out=sel?data1:data0;
17  endmodule
```

- Verilog Code for Counter to count 16

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////////////////////
3
4   //////////////////////////////////////////////////////////////////////////////////
5
6
7   module COUNTER16(count16done, // one bit output
8                    count16set,  // one bit input
9                    count16reset,
10                   rst,
11                   clk);
12  output  count16done;
13  input count16set,
14        count16reset,
15        rst,
```

```
      clk;
reg[4:0] count;
assign count16done=count[4];
always @(posedge clk)
begin
 if(rst)
  count<=5'b00001;
 else if(count16reset)
  count<=5'b00001;
 else if(count16set)
  count<=count+1;
 else
  count<=count;
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<=1;i=i+1)
 begin

 tb_multiplicand={$random}%65000;
```

```
35    tb_multiplier={$random}%75000;
36    #400;
37    end
38
39    $finish;
40  end
41  endmodule
```

2.7 Report

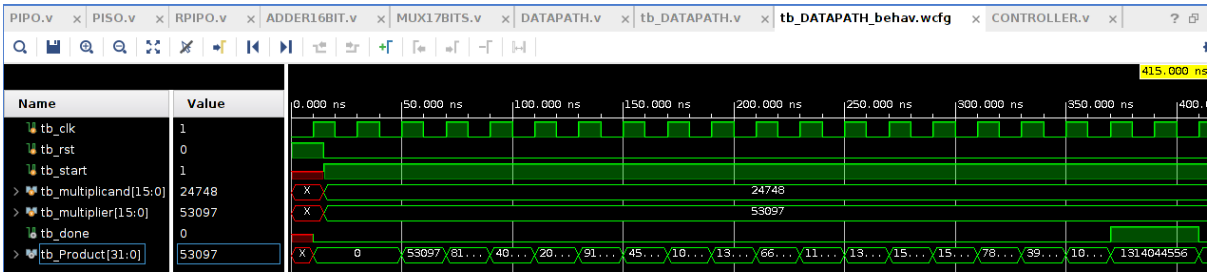

Figure 2.6: Behavioural Timing Simulation

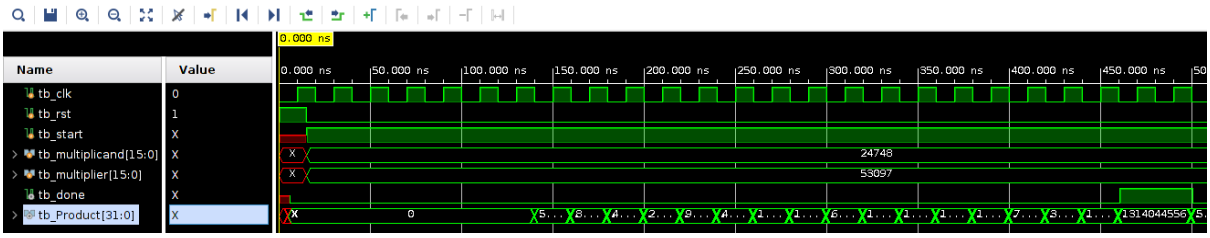

Figure 2.7: Post Implementation Timing Simulation

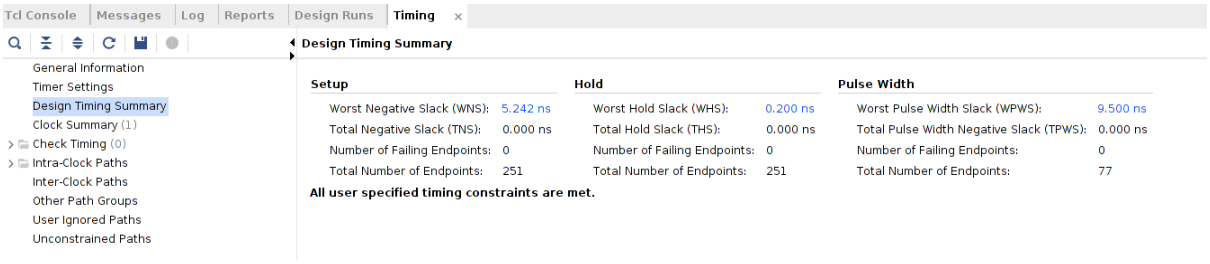

Figure 2.8: Timning Constraints

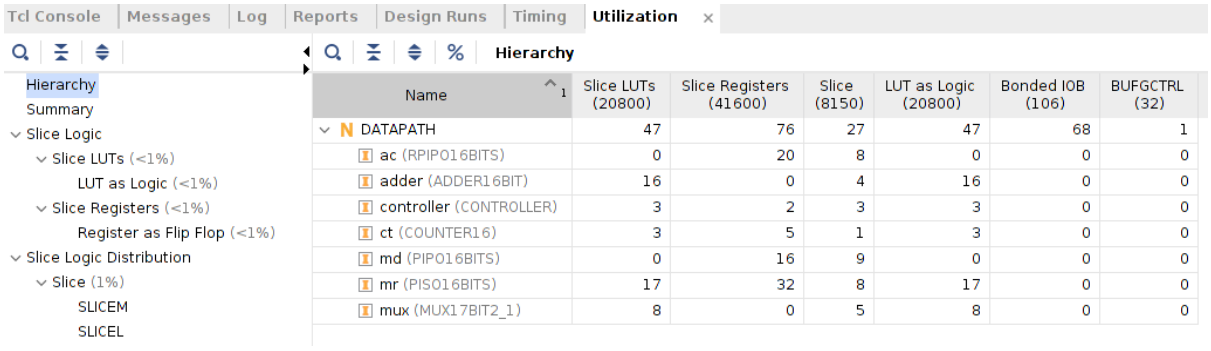

Figure 2.9: Utilization

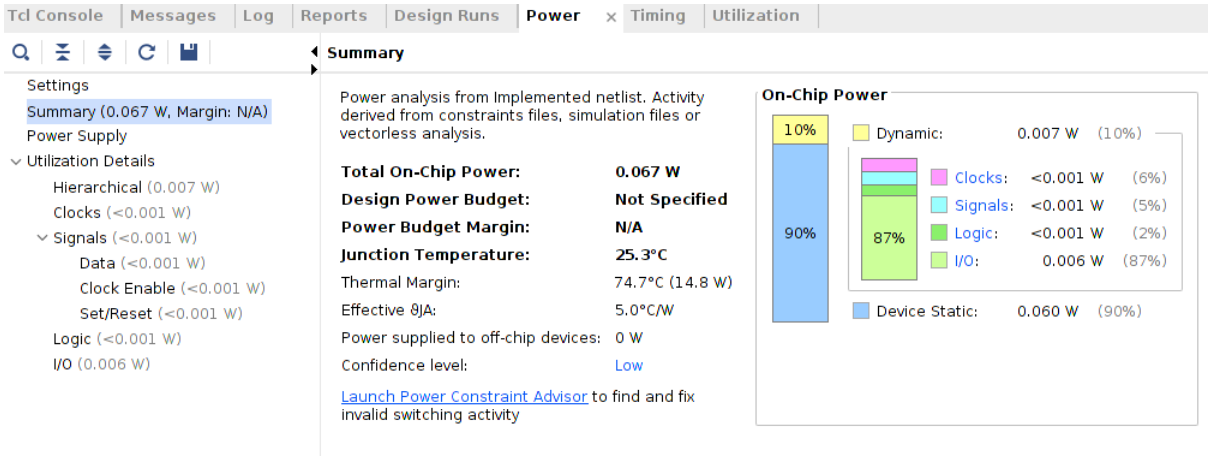

Figure 2.10: Power

Figure 2.11: Project Summery

- Maximum frequency calculation

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

  $T_{slak} = 5.424ns$ shown in the fig 2.8

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

  $T_{min} = 14.758ns$ // Hence $f_{max} \approx 68MHz$

  Latency=17 clock cycle

2.8 Pipelining

The pipelining is the process in which successive steps of an instruction sequence are executed in turn by a sequence of modules able to operate concurrently, so that another instruction can be begun before the previous one is finished.

Example:Y=A*B+C In the following fig 2.12 it can seen that the latency of the A and B input

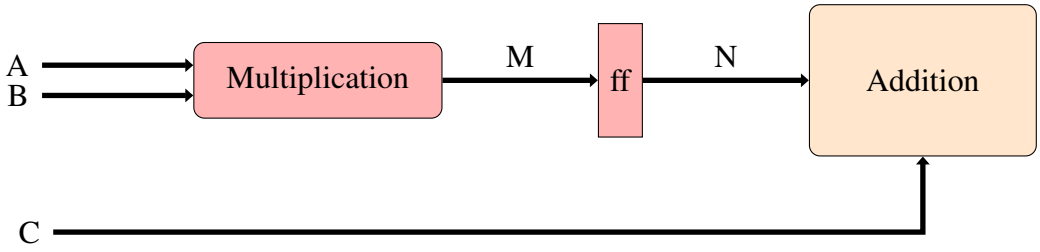

Figure 2.12: PipelineExample

compare to C input is more to reach at Addition block. Because of that throughput is not 1 but after insertion of a flip flop in the path of C. Throughput can be increased.

2.8.1 Data Path

2.8.2 Verilog Codes

- Verilog Code for Data Path

```
1  `timescale 1ns / 1ps
2
3  //////////////////////////////////////////////////////////////////////////////////
4
5  module PIPELINED_MUL(Product,
6                       //done,
7                         clk,
8                         rst,
```

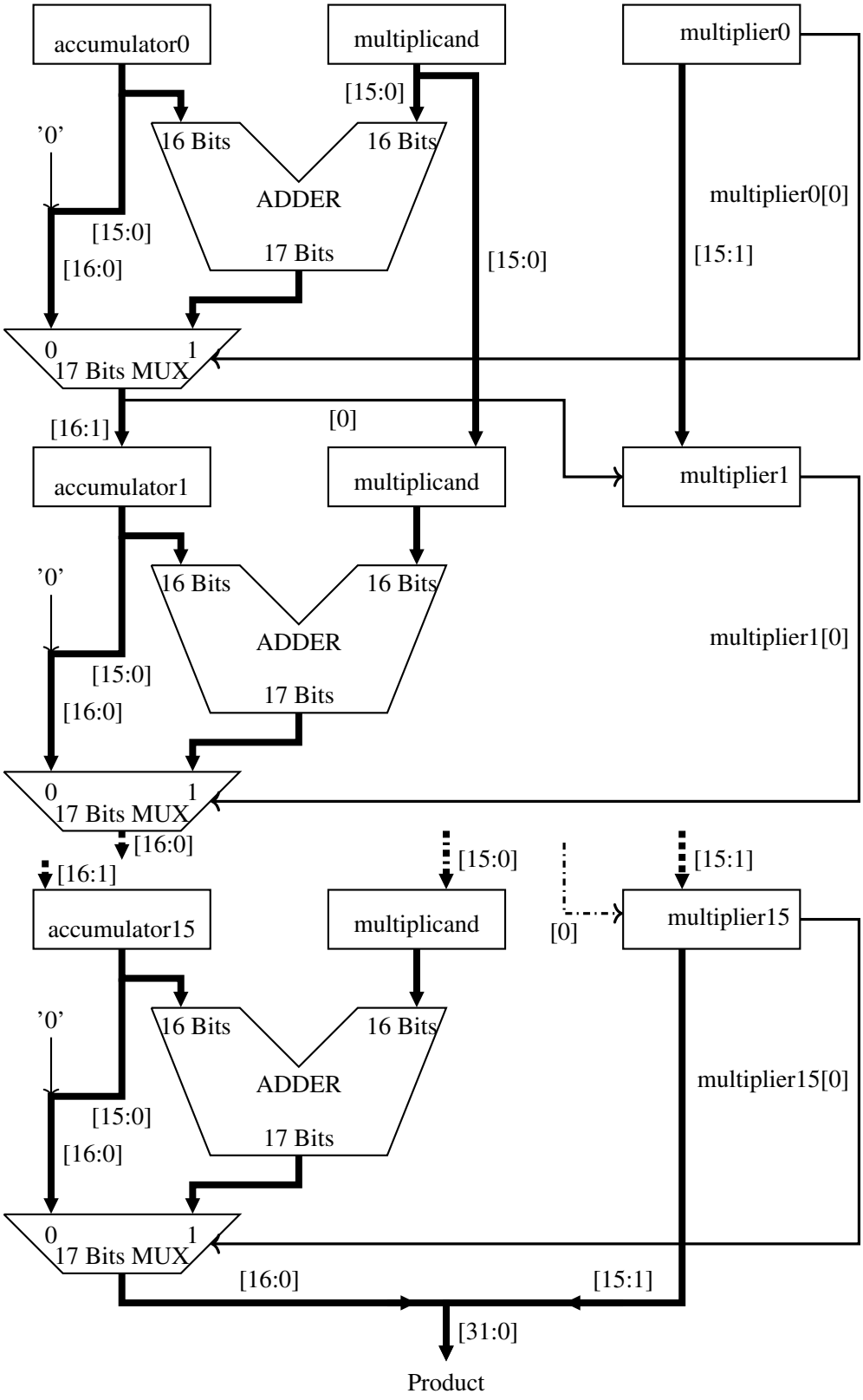

Figure 2.13: Data Path with Pipelining

```
9                       multiplicand,
10                      multiplier);
11  parameter n=16,   // multiplicand width size
12            m=16;  //  multiplier  width size
13  output[n+m-1:0] Product;
14  //output done
15  input[n-1:0] multiplicand;
16  input[m-1:0] multiplier;
17  input clk,
18        rst;
19  // submodule wire instantiation
20  wire[n-1:0] m0_acc_in,
21              m0_md_in,
22              m0_acc_out,
23              m0_md_out;
24  wire[m-1:0] m0_mr_in,
25              m0_mr_out;
26  wire m0_clk,
27       m0_rst;
28
29  // wire for loop
30  wire[0:m-1] m_acc_out[n-1:0];
31  wire[0:m-1] m_md_out[n-1:0];
32  wire[0:m-1] m_mr_out[m-1:0];
33
34
35  SUBMODULE m0(.acc_in(m0_acc_in),
36                  .md_in(m0_md_in),
37                  .mr_in(m0_mr_in),
38                  .clk(m0_clk),
39                  .rst(m0_rst),
40                  .acc_out(m0_acc_out),
41                  .md_out(m0_md_out),
42                  .mr_out(m0_mr_out));
43  assign m0_acc_in=16'b0;
44  assign m0_md_in=multiplicand;
45  assign m0_mr_in=multiplier;
46  assign m0_clk=clk;
47  assign m0_rst=rst;
48
49
50  assign m_acc_out[0]=m0_acc_out;
51  assign m_md_out[0]=m0_md_out;
52  assign m_mr_out[0]=m0_mr_out;
53
54  genvar i;
55  generate
56   for(i=1;i<m;i=i+1)
57    begin
58      SUBMODULE m( m_acc_out[i-1], //acc_in,
59                   m_md_out[i-1], //md_in,
60                   m_mr_out[i-1], //mr_in,
61                   clk,           //clk,
```

```
                rst,           //rst,
                m_acc_out[i],  //acc_out,
                m_md_out[i],   // md_out,
                m_mr_out[i]);   // mr_out);

  end
endgenerate

// output initialization
assign Product = {m_acc_out[m-1],m_mr_out[m-1]};

endmodule
```

- Verilog Code for submodule

```
`timescale 1ns / 1ps

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


module SUBMODULE(acc_in,
                 md_in,
                 mr_in,
                 clk,
                 rst,
                 acc_out,
                 md_out,
                 mr_out);
parameter n=16, // multiplicand width size
          m=16; // multiplier width size
output [n-1:0] acc_out;
output [n-1:0] md_out;
output [m-1:0] mr_out;

input clk,
      rst;
input[n-1:0] acc_in;
input[n-1:0] md_in;
input[m-1:0] mr_in;

//adder wire instantiation
wire[n:0] ad_data_out;
wire[n-1:0] ad_data0,
            ad_data1;

// mux wire instantiation
wire[n:0] mux_data_out,
          mux_data0,
          mux_data1;
wire mux_sel;

///
```

```
reg[n-1:0] accumulator,
           multiplicand;
reg[m-1:0] multiplier;


///
// adder instantaition
ADDER16BIT ad(.data_out(ad_data_out),
                  .data0(ad_data0),
                  .data1(ad_data1));
assign ad_data0=accumulator;
assign ad_data1=multiplicand;
// mux instantiation
MUX17BIT2_1 mux(.data_out(mux_data_out),
                   .data0(mux_data0),
                   .data1(mux_data1),
                   .sel(mux_sel));
assign mux_data0={1'b0,accumulator};
assign mux_data1=ad_data_out;
assign mux_sel=multiplier[0];

always @(posedge clk)
begin
if(rst)
begin
 accumulator<=16'b0;
 multiplicand<=16'b0;
 multiplier<=16'b0;
end
else
begin
 accumulator<=acc_in;
 multiplicand<=md_in;
 multiplier<=mr_in;
end
end

// out declarattion
assign acc_out=mux_data_out[16:1];
assign md_out=multiplicand;
assign mr_out={mux_data_out[0],multiplier[15:1]};
endmodule
```

- Test bench for Data Path

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

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


module tb_PIPELINED_MUL;
```

```
8  parameter n=16,m=16;
9  wire[n+m-1:0] tb_Product;
10 reg tb_clk,
11     tb_rst;
12 reg[15:0] tb_multiplicand,
13           tb_multiplier;
14 PIPELINED_MUL DUT(.Product(tb_Product),
15                  .clk(tb_clk),
16                  .rst(tb_rst),
17                  .multiplicand(tb_multiplicand),
18                  .multiplier(tb_multiplier));
19
20 initial tb_clk=1'b0;
21 always #10 tb_clk=~tb_clk;
22 integer i;
23 initial
24 begin
25 tb_rst=1'b1;
26 #15 tb_rst=1'b0;
27 $monitor($time,"tb_Product=%b",tb_Product);
28 for(i=1;i<=25;i=i+1)
29  begin
30
31  tb_multiplicand={$random}%65000;
32  tb_multiplier={$random}%75000;
33  #20;
34  end
35
36  #500 $finish;
37 end
38 endmodule
39
40
41
42
43
44
45 `timescale 1ns / 1ps
46 //////////////////////////////////////////////////////////////////////////////////
47
48 //////////////////////////////////////////////////////////////////////////////////
49
50
51 module tb_PIPELINED_MUL;
52 parameter n=16,m=16;
53 wire[n+m-1:0] tb_Product;
54 reg tb_clk,
55     tb_rst;
56 reg[15:0] tb_multiplicand,
57           tb_multiplier;
58 PIPELINED_MUL DUT(.Product(tb_Product),
59                  .clk(tb_clk),
60                  .rst(tb_rst),
```

```
                .multiplicand(tb_multiplicand),
                .multiplier(tb_multiplier));

initial tb_clk=1'b0;
always #10 tb_clk=~tb_clk;
integer i;
initial
begin
tb_rst=1'b1;
#15 tb_rst=1'b0;
$monitor($time,"tb_Product=%b",tb_Product);
for(i=1;i<=25;i=i+1)
 begin

 tb_multiplicand={$random}%65000;
 tb_multiplier={$random}%75000;
 #20;
 end

 #500 $finish;
end
endmodule
```

- Test Bench for sub module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/30/2022 05:21:02 PM
// Design Name:
// Module Name: RPIPO
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


`timescale 1ns / 1ps

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


```

```
28  module RPIPO16BITS(data_out,
29                     clk,
30                     data_in,
31                     Rrst,
32                     load,
33                     rst);
34  parameter n=16;          // multiplier width size
35
36  output reg[n-1:0] data_out;
37  input[n-1:0] data_in;
38  input load,
39        Rrst,
40        clk,
41        rst;
42  always @(posedge clk)
43   begin
44    if(rst)
45     data_out<=16'b0;
46    else if(Rrst)
47     data_out<=16'b0;
48    else if(load)
49     data_out<=data_in;
50    else
51     data_out<=data_out;
52   end
53  endmodule
```

2.8.3 Report

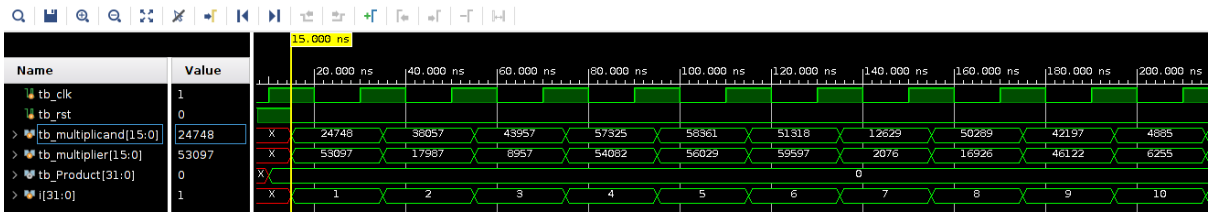

Figure 2.14: Behavioural Timing Simulation For Input Data

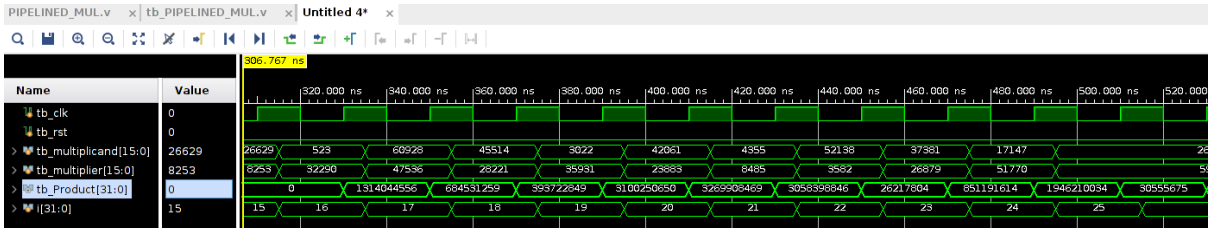

Figure 2.15: Behavioural Timing Simulation For Output Data

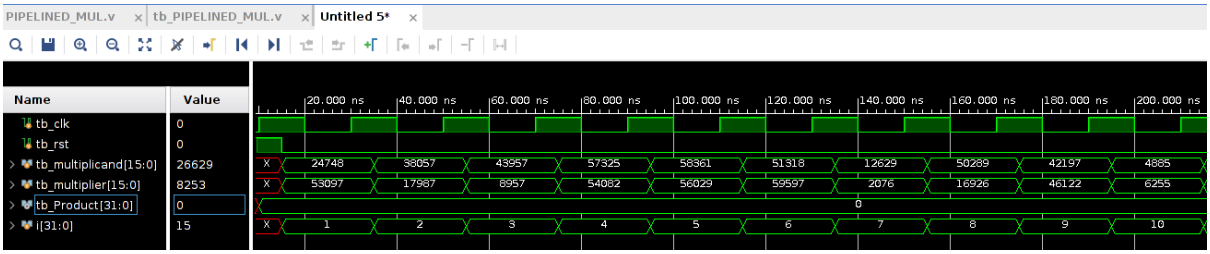

Figure 2.16: Post Implementation Timing Simulation For Input

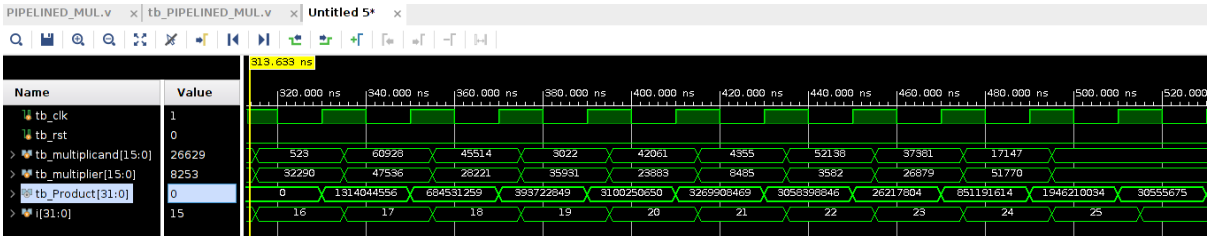

Figure 2.17: Post Implementation Timing Simulation For Output

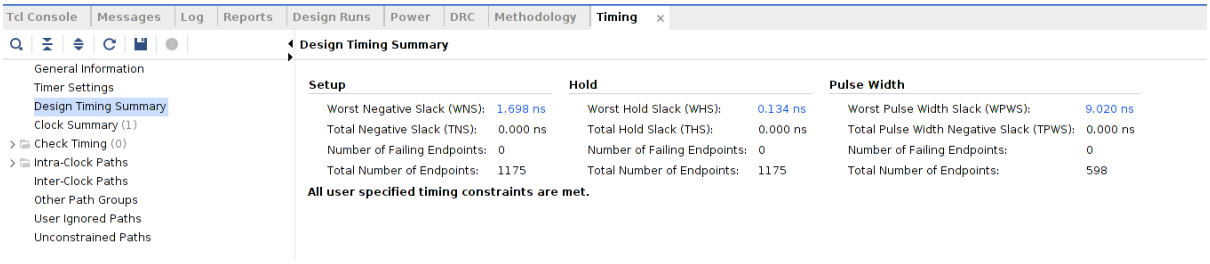

Figure 2.18: Timning Constraints

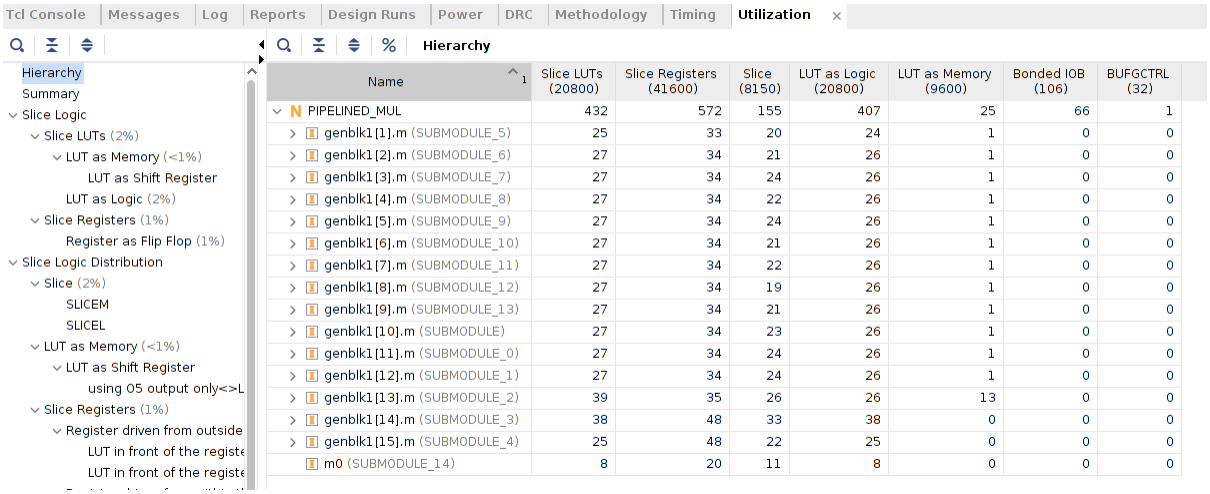

Figure 2.19: Utilization

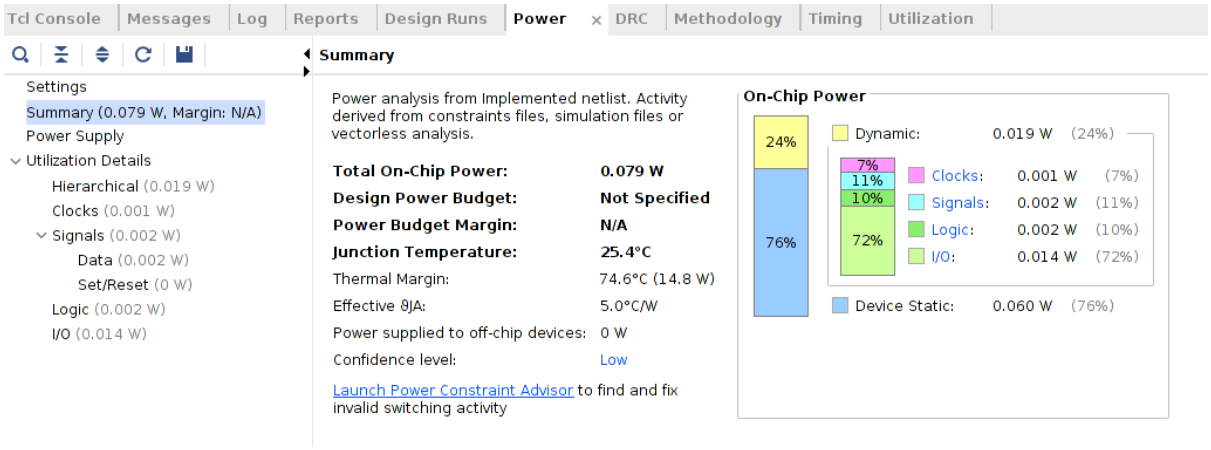

Figure 2.20: Power

- Maximum frequency calculation

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

  $T_{slak} = 1.698ns$ shown in the fig 2.8

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

  $T_{min} = 18.302ns$ // Hence $f_{max} \approx 55MHz$

  Throughput=1 per clock cycle

  Latency=16 clock cycle

Lerning from this project

- pipeling

Bibliography