EE518 LAB
Experiment 7

Design and implement a matrix convolution circuit with padding and N-bit parameterised integer ALU using Verilog

Submitted by,

Monish Nath

ROLL NO-224102409

March 19, 2023

Contents

List of Figures

List of Tables

1 Objective :

Design and implement Verilog modules to:
a. Modify the convolution module for inclusion of the padding function. The convolution block should contain a flag to enable/disable padding for convolution.

b. Design of a N bit parameterized integer arithmetic and logic unit, to perform addition, subtraction, right shift, left shift, arithmetic right shift, arithmetic left shift and logical operations as not, and, or and xor

2 Theory of Convolution:

Convolution is the treatment of a matrix by another one which is called "kernel".A convolution is a type of matrix operation, consisting of a kernel, a small matrix of weights, that slides over input data performing element-wise multiplication with the part of the input it is on, then summing the results into an output.

Intuitively, a convolution allows for weight sharing - reducing the number of effective parameters - and image translation (allowing for the same feature to be detected in different parts of the input space).

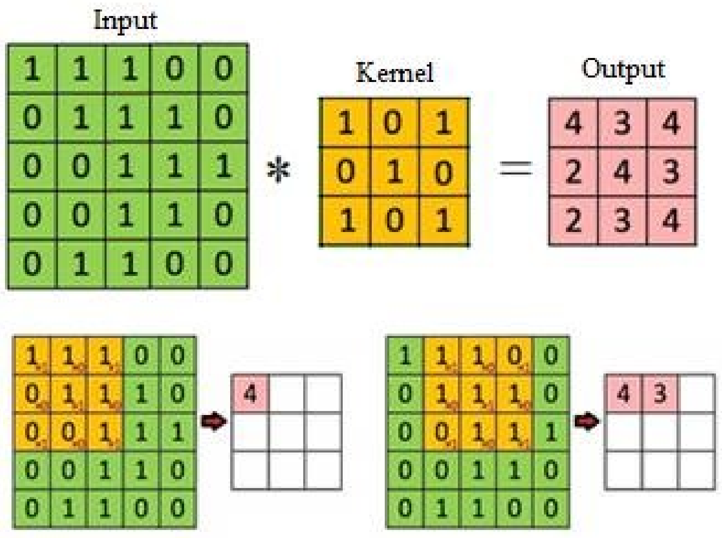


Figure 1: Basic Convolution Operation

2.1 Padding:

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one pixel border added to the image with a pixel value of zero.

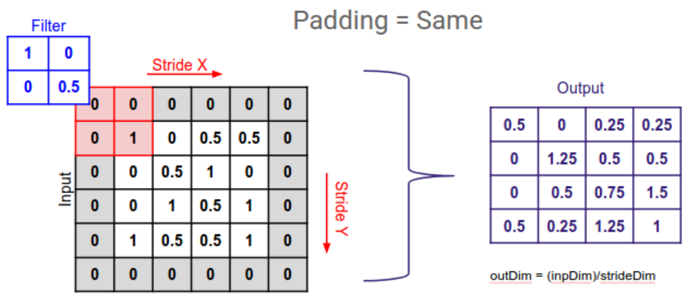


Figure 2: Basic Convolution Operation with padding

3 Matrix Convolution with padding :

3.1 Design Approach :

The design only has one input which accepts each input at the positive edge of clock. So, both the matrix elements are inserted to the design one by one. Initially the kernel inputs are send which is a mxm matrix (m=2) so first m^2 (4) elements are stored in a shift register which is our kernel. Now the main matrix input are fed one by one. We have such an arrangement of shift registers that we get a m*m window of the main n*n matrix at any instant of time and the convolution output is produced directly. The figure gives a rough representation of our algorithm. The output signals out_valid and end_conv indicate us which output values are correct and when the operation has ended.

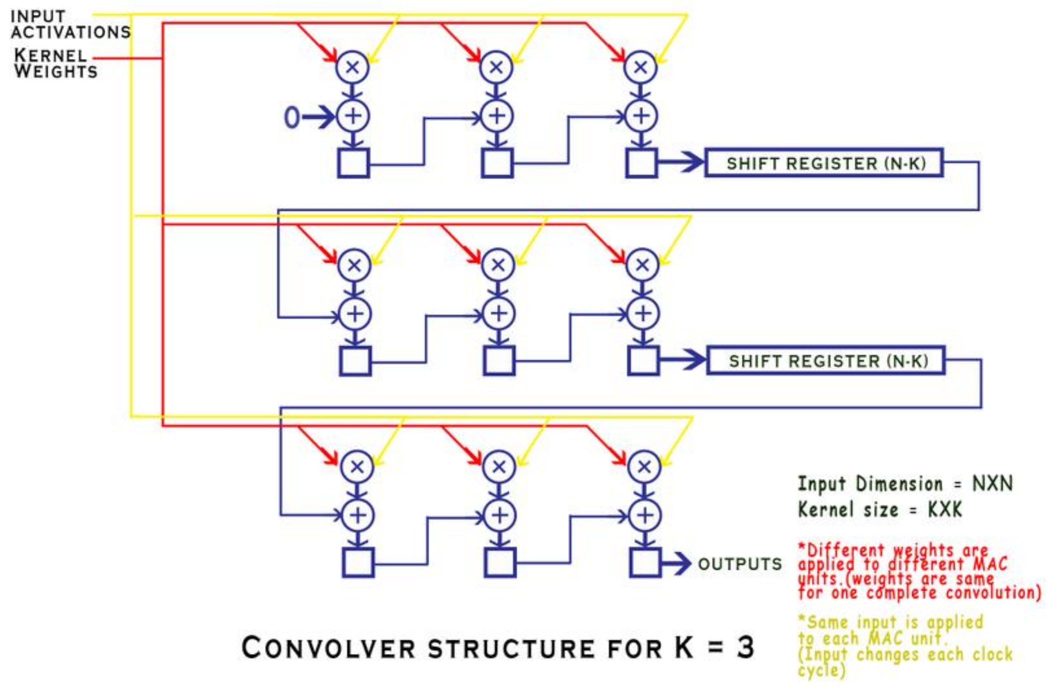

INPUT ACTIVATIONS
KERNEL WEIGHTS
0
SHIFT REGISTER (N-K)
SHIFT REGISTER (N-K)
OUTPUTS
Input Dimension = NXN
Kernel size = KXK
*Different weights are applied to different MAC units.(weights are same for one complete convolution)
*Same input is applied to each MAC unit. (Input changes each clock cycle)
CONVOLVER STRUCTURE FOR K = 3

Figure 3: Convolution Algorithm Schematic

The code is kept same just a border of pixel with 0 values are added to have the padding implementation. However we couldnot add a flag to enable or disable padding which can also be implemented with some modifications. Code is given for fixed point representation only. Floating point has same code with some modification in MAC unit.

3.2 Verilog Code :

```
module conv_fix(
    input clock,
    input rst,
    input [31:0] a_in,
    output [31:0] result,
    output reg out_valid,
    output reg end_conv
```

```
   );
   parameter n=3, m=2;
   reg [31:0] k [m*m-1:0];
   reg [31:0] w [m*m-1:0];
   reg [31:0] u [m*m-1:0];
   reg [31:0] v [(n+2)*(n+2)-1:0];
   reg [31:0] p [(m-1)*(n+2)-1:0];
   reg [5:0] count, count1, count2, count3;
   reg [5:0] y, i1=0, i2=0;

   always@(posedge clock) begin
   if(rst) count <= 'b0;
   else if(end_conv == 'b1) count <= 'b0;
   else count <= count + 1'b1;
   end

   always@(posedge clock)
   k[m*m-1] <= (count<m*m) ? a_in : k[m*m-1];

   genvar i;
   generate
   for(i=0;i<m*m-1;i=i+1)
   always@(posedge clock)
   k[m*m-1-i-1] <= (count<m*m) ? k[m*m-1-i] : k[m*m-1-i-1] ;
   endgenerate

//code change for padding
   always@(posedge clock) begin
   if(count<m*m) w[m*m-1] <= 'b0;
   else if(count-m*m<n+3) begin
   v[i1] <= a_in;
   i1 = i1 + 1;
   w[m*m-1] <= 'b0; end
   else if(count>m*m+n+3-1+n*(n+2)) w[m*m-1] <= 'b0;
   else if((count-m*m-n+3-1)%(n+2)<n) begin
   w[m*m-1] <= v[i2];
   i2 = i2 + 1;
   v[i1] <= a_in;
```

```
    i1 = i1 + 1;
    end
    else if((count-m*m-n+3-1)%(n+2)>=n) begin
    v[i1] <= a_in;
    i1 = i1 + 1;
    w[m*m-1] <= 'b0; end
    else w[m*m-1] <= 'b0;
    end
//padding code changes till here

    genvar j,l;
    generate
    for(i=0;i<m-1;i=i+1) begin

    for(j=0;j<m-1;j=j+1)
    always@(posedge clock)
    w[m*m-1-i*m-j-1] <= w[m*m-1-i*m-j];

    always@(*)
    p[0+i*m] <= w[m*m-1-i*m];
    for(l=0;l<(n+2)-1;l=l+1)
    always@(posedge clock) begin
    p[l+i*m+1] <= p[l+i*m];
    w[m*m-1-i*m-m] <= p[(n+2)-1];
    end
    end

    for(j=0;j<m-1;j=j+1)
    always@(posedge clock)
    w[m-1-j-1] <= w[m-1-j];
    endgenerate
    //inputs at w and k

    always@(*) u[m*m-1] <= w[m*m-1]*k[m*m-1];
    generate
    for(i=m*m-2;i>=0;i=i-1)
    always@(*) u[i] <= u[i+1] + w[i]*k[i];
    endgenerate
    assign result = u[0];

    // logic to generate valids
    //n-->n+2 for padding logic
    always@(posedge clock) begin
    if(rst) begin out_valid <= 'b0; end_conv <= 'b0; end
    else if(count < (m-1)*(n+2) + m - 'b1 + m*m );
    else if(count == (m-1)*(n+2) + m - 'b1 + m*m) begin
    out_valid <= 'b1; count1<='b1; count2<='b1; count3<='b1; end
    else if(end_conv == 'b1) begin out_valid <= 'b0; end_conv <= 'b0; end
    else if((count2 == (n+2)-m) && (count3 == (n+2)-m+'b1)) begin
    end_conv <= 'b1; count1<='b0; count2<='b0; count3<='b0; end
    else if(count2 == (n+2)-m+'b1) begin
    out_valid <= 'b0; count1<=count1+'b1; count2<='b0; end
    else if(count1%(n+2) == 'b0) begin
    out_valid <= 'b1; count1<=count1+'b1; count2<='b1; count3<=count3+'b1; end
    else begin count1<=count1+'b1; count2<=count2+'b1; end
    end

endmodule
```

Note: Decimal point is not defined in code. Just take proper care in test-bench to have x bits after decimal in input and 2x bits in output. Here we take 5 bits after decimal in input and 10bits in output for simulation. Code works for all cases.

3.3 Test bench ;

```
module tbconv_fix(    );

reg clock, rst;
reg [31:0] a;
wire [31:0] result;
wire out_valid, end_conv;

always #50 clock = ~clock;
```

```
conv_fix DUT (clock, rst, a, result, out_valid, end_conv);

always@(negedge clock) a = a+6'b100000;

initial begin
clock = 0; rst = 1;
#100 rst = 0; a = 6'b000000;
end
endmodule
```

3.4 RTL Schematic :

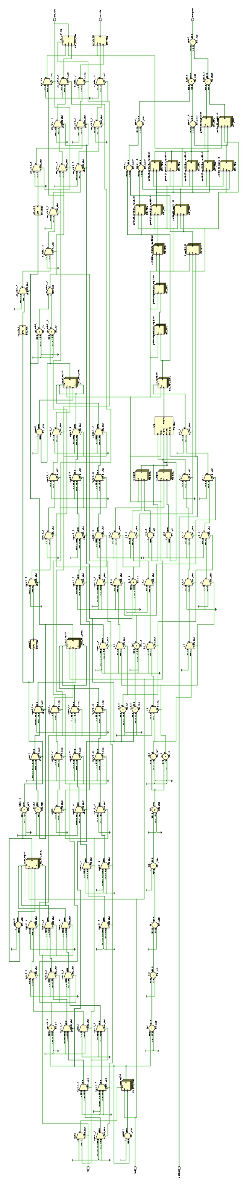


Figure 4: Schematic of convolution circuit with padding

3.5 Simulation :

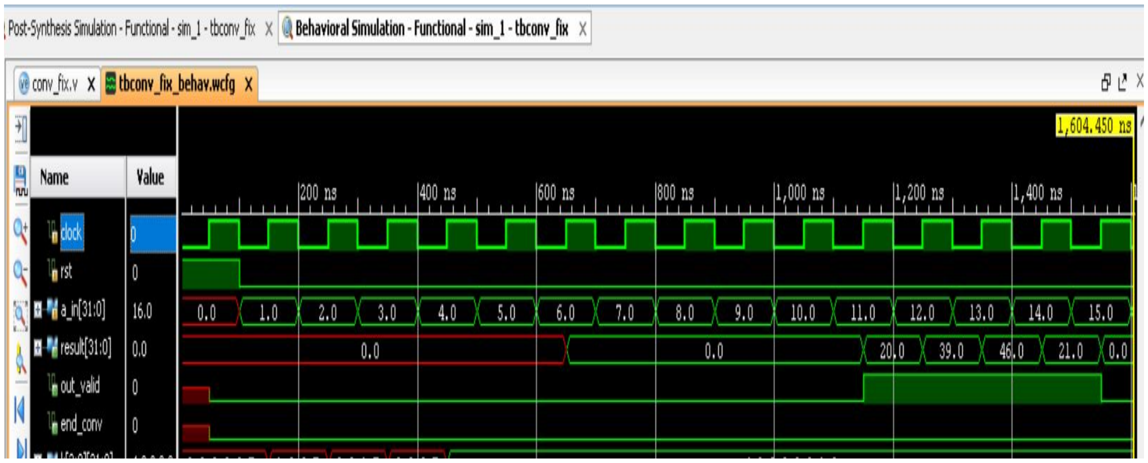


Figure 5: Behavioural simulation waveform 0-50

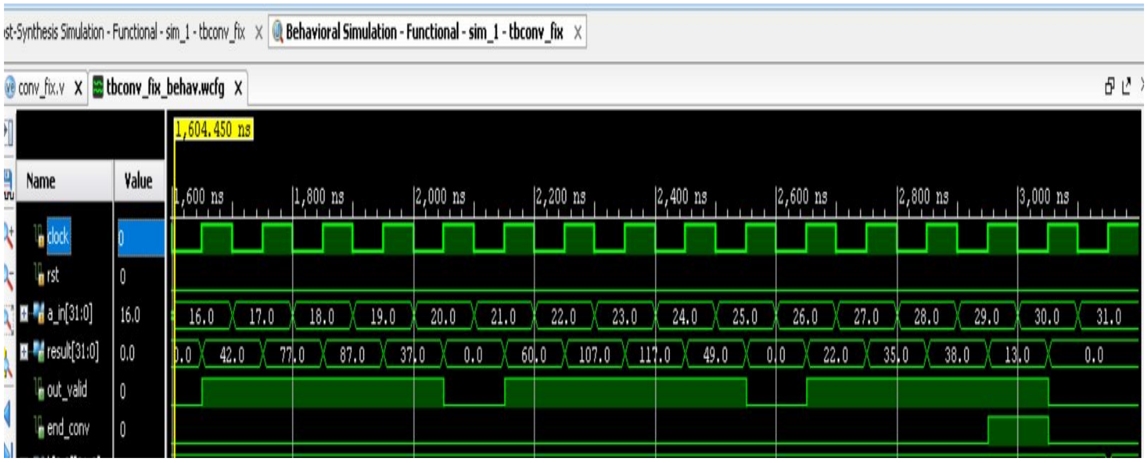


Figure 6: Behavioural simulation waveform 50-100

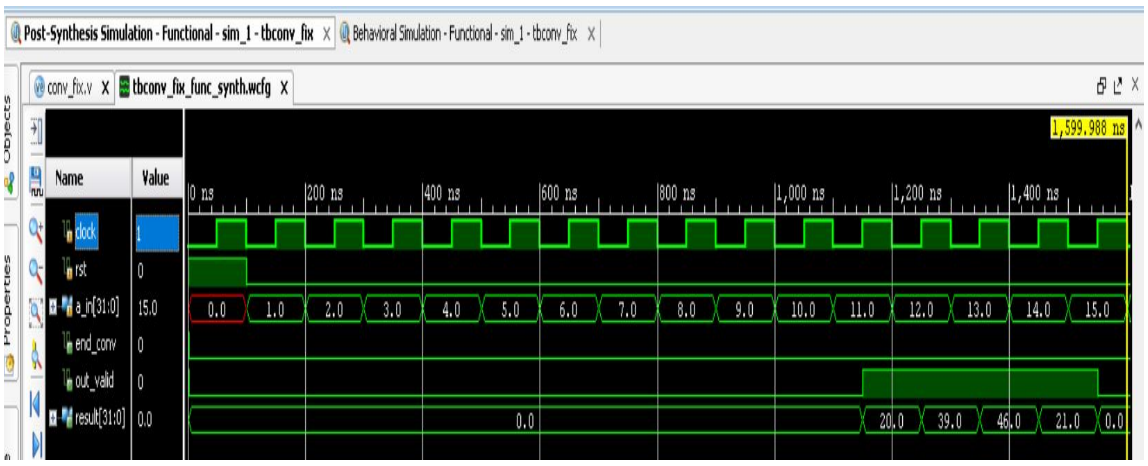

Post-Synthesis Simulation - Functional - sim_1 - tbconv_fix
Behavioral Simulation - Functional - sim_1 - tbconv_fix
conv_fix.v
tbconv_fix_func_synth.wcfg
1,599.988 ns
Name
Value
clock 1
rst 0
a_in[31:0] 15.0
end_conv 0
out_valid 0
result[31:0] 0.0

Figure 7: Post-synthesis functional simulation waveform 0-50

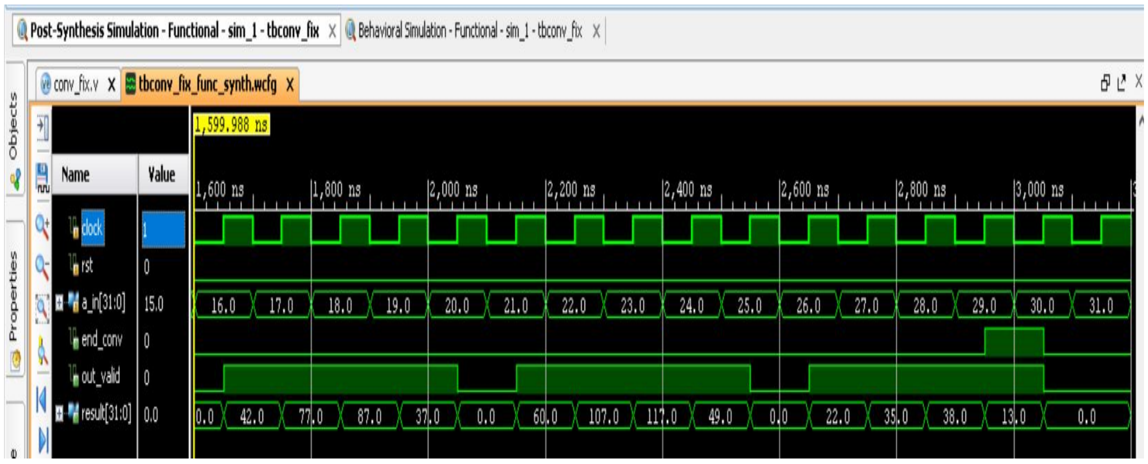

Post-Synthesis Simulation - Functional - sim_1 - tbconv_fix
Behavioral Simulation - Functional - sim_1 - tbconv_fix
conv_fix.v
tbconv_fix_func_synth.wcfg
1,599.988 ns
Name
Value
clock 1
rst 0
a_in[31:0] 15.0
end_conv 0
out_valid 0
result[31:0] 0.0

Figure 8: Post-synthesis functional simulation waveform 50-100

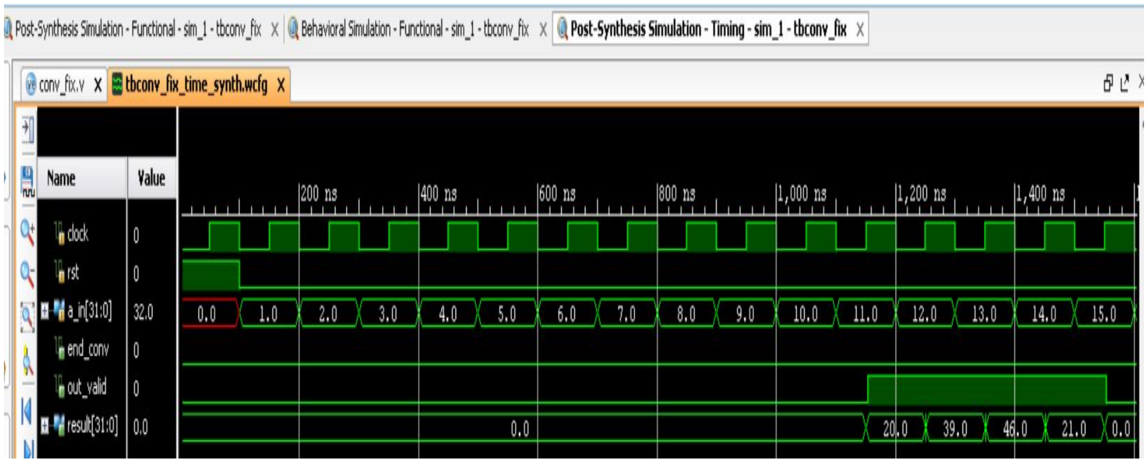


Figure 9: Post synthesis timing simulation waveform 0-50

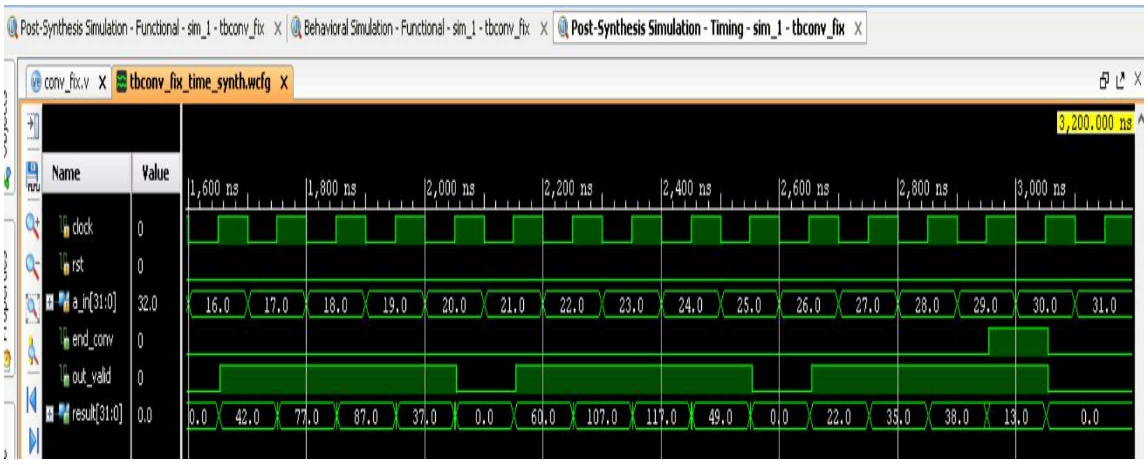


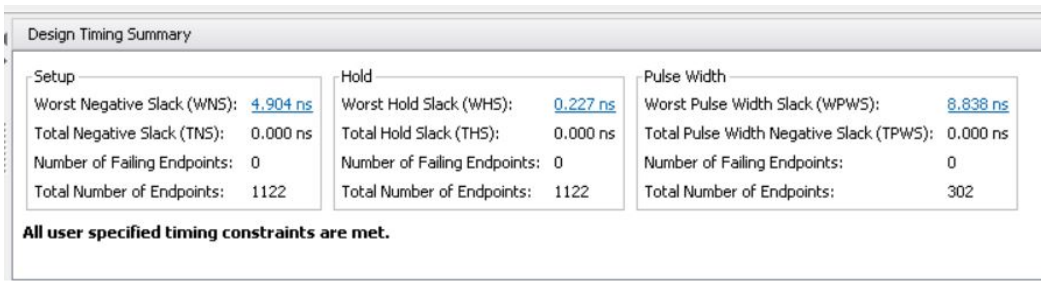
Figure 10: Post synthesis timing simulation waveform 50-100

Note: The representation of the input and output are kept in fixed point. The kernal matrix is 1,2,3,4 and main matrix is 5,6,7,8,9,10,11,12,13 which is padded with 0s in the border, so the outputs should be 20, 39, 46, 21, 42, **77, 87,** 37, 60, **107, 117,** 49, 22, 35, 38, 13. The results are as expected. The bold outputs were available without padding and others are added with padding.

3.6 Synthesis Reports :

The values have been found after synthesis of the corresponding designs. I have done the experiment on Vivado 2014.1. The FPGA board selected is Artix-7. The LUTs and Flops have been found from the utilization report. The delay has been found from the timing report and the power has been found from the power report.

We have added proper constraints and the synthesis results are shown below.

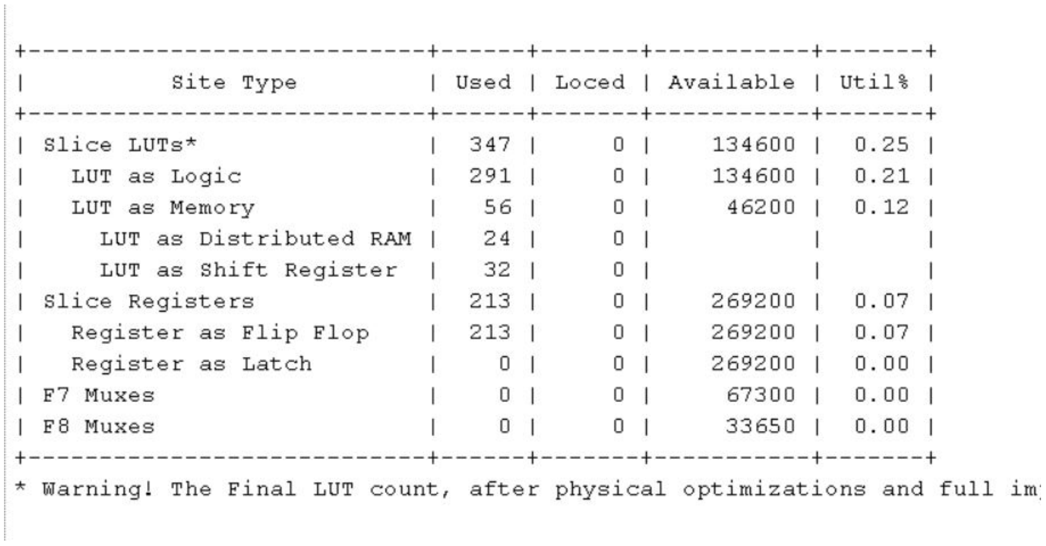


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.904 ns	Worst Hold Slack (WHS):	0.227 ns	Worst Pulse Width Slack (WPWS):	8.838 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:	1122	Total Number of Endpoints:	1122	Total Number of Endpoints:	302

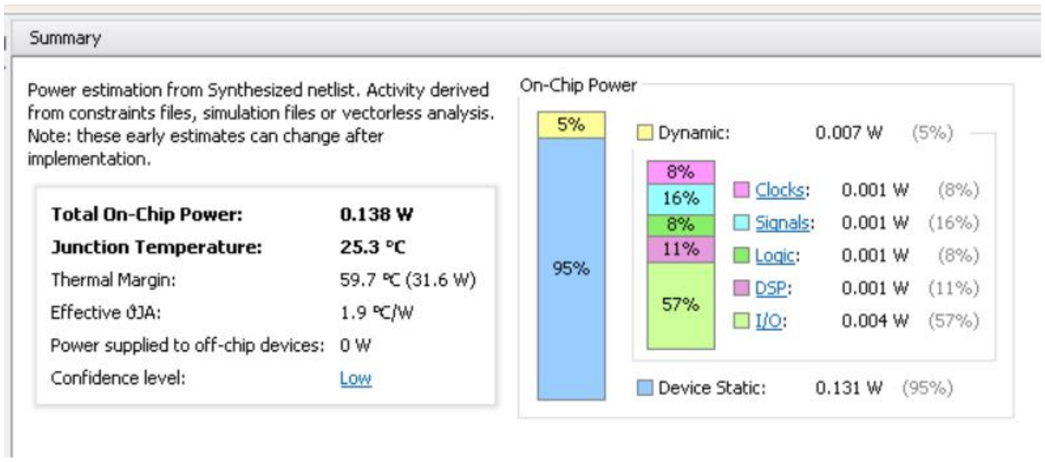
All user specified timing constraints are met.

Figure 11: Timing summary



```
+----------------------------+------+-------+-----------+-------+
|          Site Type         | Used | Loced | Available | Util% |
+----------------------------+------+-------+-----------+-------+
| Slice LUTs*                |  347 |     0 |    134600 |  0.25 |
|   LUT as Logic             |  291 |     0 |    134600 |  0.21 |
|   LUT as Memory            |   56 |     0 |     46200 |  0.12 |
|     LUT as Distributed RAM |   24 |     0 |           |       |
|     LUT as Shift Register  |   32 |     0 |           |       |
| Slice Registers            |  213 |     0 |    269200 |  0.07 |
|   Register as Flip Flop    |  213 |     0 |    269200 |  0.07 |
|   Register as Latch        |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                   |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                   |    0 |     0 |     33650 |  0.00 |
+----------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and full im
```

Figure 12: Utilization report


Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.138 W**
Junction Temperature:	**25.3 °C**
Thermal Margin:	59.7 °C (31.6 W)
Effective ϑJA:	1.9 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

On-Chip Power

🟨	Dynamic:	0.007 W	(5%)
🟪	Clocks:	0.001 W	(8%)
🟦	Signals:	0.001 W	(16%)
🟩	Logic:	0.001 W	(8%)
🟪	DSP:	0.001 W	(11%)
🟩	I/O:	0.004 W	(57%)
🟦	Device Static:	0.131 W	(95%)

Figure 13: Power report

4 N-bit parameterised integer ALU :

4.1 Design Approach :

A simple arithmetic and logical unit has been designed with muxing logic. The functions implemented are addition, subtraction, right shift, left shift, arithmetic right shift, arithmetic left shift and logical operations as not, and, or, xor and the corresponding control signals are 1, 2, 3, 4, 5, 6, 7, 8, 9, a. The code is shown below.

4.2 Verilog Code :

```
module alu_n
    #(parameter N=32)
        (rs1, rs2, clock, ctrl, outP);

input [N-1:0] rs1;
input [N-1:0] rs2;
input clock;
input [3:0] ctrl;
output reg [N-1:0] outP;
reg [N-1:0] out;
```

```
always@(posedge clock) outP <= out;

always@(rs1, rs2, ctrl) begin
case(ctrl)
4'h1 : out <= rs1 + rs2 ;   //add
4'h2 : out <= rs1 - rs2 ;   //sub
4'h3 : out <= rs1 >> rs2[4:0] ; //rightshift
4'h4 : out <= rs1 << rs2[4:0] ; //leftshift
4'h5 : out <= $signed(rs1) >>> rs2[4:0] ;   //rightshift arithmetic
4'h6 : out <= $signed(rs1) <<< rs2[4:0] ;   //leftshift arithmetic
4'h7 : out <= ~rs1 ;    //not
4'h8 : out <= rs1 & rs2 ;   //and
4'h9 : out <= rs1 | rs2 ;   //or
4'ha : out <= rs1 ^ rs2 ;   //xor
default : out <= 32'hxxxxxxxx ;
endcase
end

endmodule
```

4.3 Test bench ;

```
module tb_alu(    );

reg clock;
reg [31:0] rs1, rs2;
reg [3:0] ctrl;
wire [31:0] outP;

always #100 clock = ~clock;

alu_n DUT (rs1, rs2, clock, ctrl, outP);

initial begin
clock = 0;
#200 ctrl = 4'h1; rs1=$random; rs2=$random;
```

```
#200 ctrl = 4'h2; rs1=$random; rs2=$random;
#200 ctrl = 4'h3; rs1=$random; rs2=5'h4;
#200 ctrl = 4'h4; rs1=$random;
#200 ctrl = 4'h5; rs1=$random;
#200 ctrl = 4'h6; rs1=$random;
#200 ctrl = 4'h7; rs1=$random; rs2=$random;
#200 ctrl = 4'h8; rs1=$random; rs2=$random;
#200 ctrl = 4'h9; rs1=$random; rs2=$random;
#200 ctrl = 4'ha; rs1=$random; rs2=$random;

end
endmodule
```

4.4 RTL Schematic :

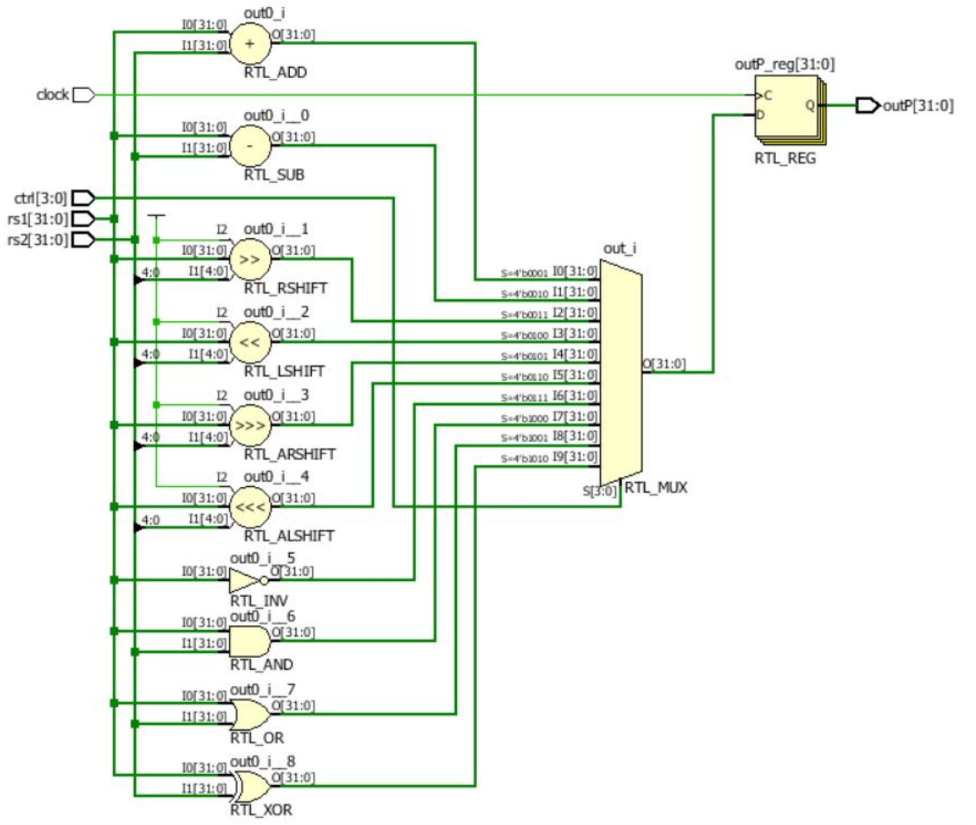


Figure 14: Schematic of ALU

4.5 Simulation :

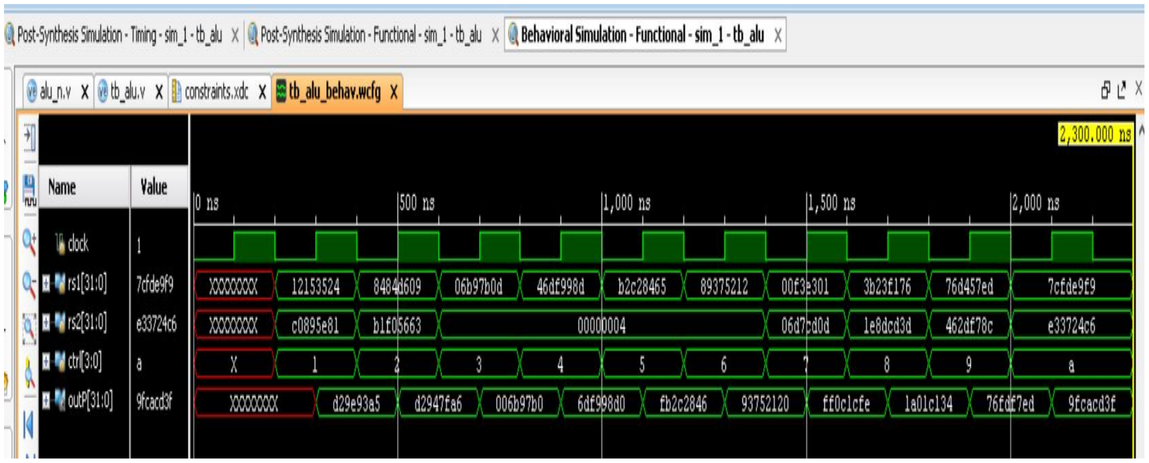

Post-Synthesis Simulation - Timing - sim_1 - tb_alu
Post-Synthesis Simulation - Functional - sim_1 - tb_alu
Behavioral Simulation - Functional - sim_1 - tb_alu
alu_n.v
tb_alu.v
constraints.xdc
tb_alu_behav.wcfg
2,300.000 ns
Name
Value
0 ns
500 ns
1,000 ns
1,500 ns
2,000 ns
clock 1
rs1[31:0] 7cfde9f9
XXXXXXXX 12153524 8484d609 06b97b0d 46df998d b2c28465 89375212 00f3e301 3b23f176 76d457ed 7cfde9f9
rs2[31:0] e33724c6
XXXXXXXX c0895e81 b1f05663 00000004 06d7cd0d 1e8dcd3d 462df78c e33724c6
ctrl[3:0] a
X 1 2 3 4 5 6 7 8 9 a
outP[31:0] 9fcacd3f
XXXXXXXX d29e93a5 d2947fa6 006b97b0 6df998d0 fb2c2846 93752120 ff0c1cfe 1a01c134 76fdf7ed 9fcacd3f

Figure 15: Behavioural simulation waveform

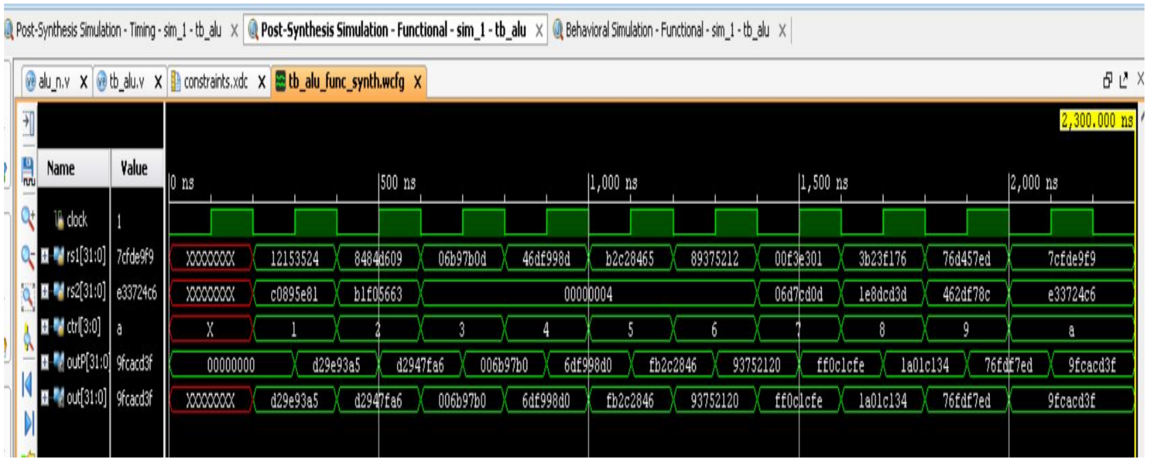

Post-Synthesis Simulation - Timing - sim_1 - tb_alu
Post-Synthesis Simulation - Functional - sim_1 - tb_alu
Behavioral Simulation - Functional - sim_1 - tb_alu
alu_n.v
tb_alu.v
constraints.xdc
tb_alu_func_synth.wcfg
2,300.000 ns
Name
Value
0 ns
500 ns
1,000 ns
1,500 ns
2,000 ns
clock 1
rs1[31:0] 7cfde9f9
XXXXXXXX 12153524 8484d609 06b97b0d 46df998d b2c28465 89375212 00f3e301 3b23f176 76d457ed 7cfde9f9
rs2[31:0] e33724c6
XXXXXXXX c0895e81 b1f05663 00000004 06d7cd0d 1e8dcd3d 462df78c e33724c6
ctrl[3:0] a
X 1 2 3 4 5 6 7 8 9 a
outP[31:0] 9fcacd3f
00000000 d29e93a5 d2947fa6 006b97b0 6df998d0 fb2c2846 93752120 ff0c1cfe 1a01c134 76fdf7ed 9fcacd3f
out[31:0] 9fcacd3f
XXXXXXXX d29e93a5 d2947fa6 006b97b0 6df998d0 fb2c2846 93752120 ff0c1cfe 1a01c134 76fdf7ed 9fcacd3f

Figure 16: Post-synthesis functional simulation waveform

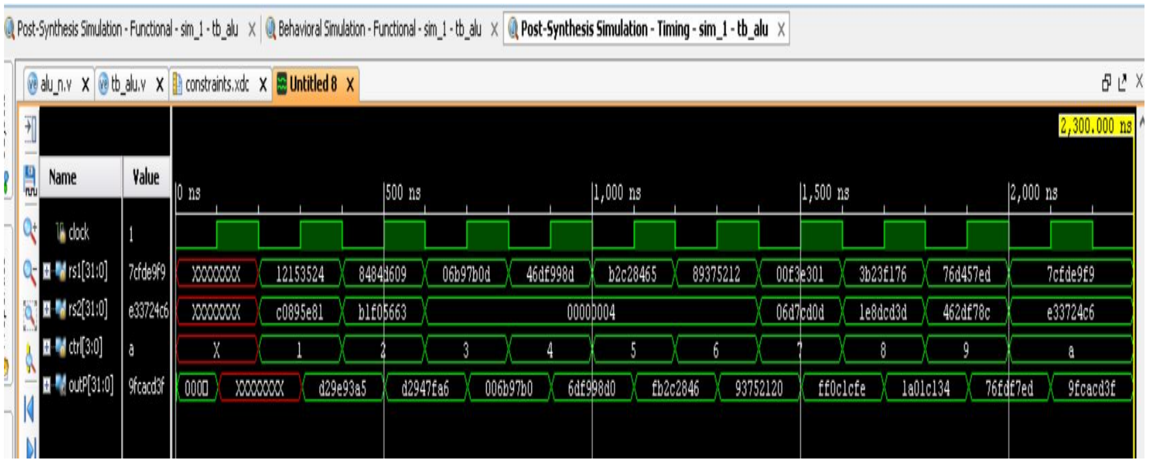
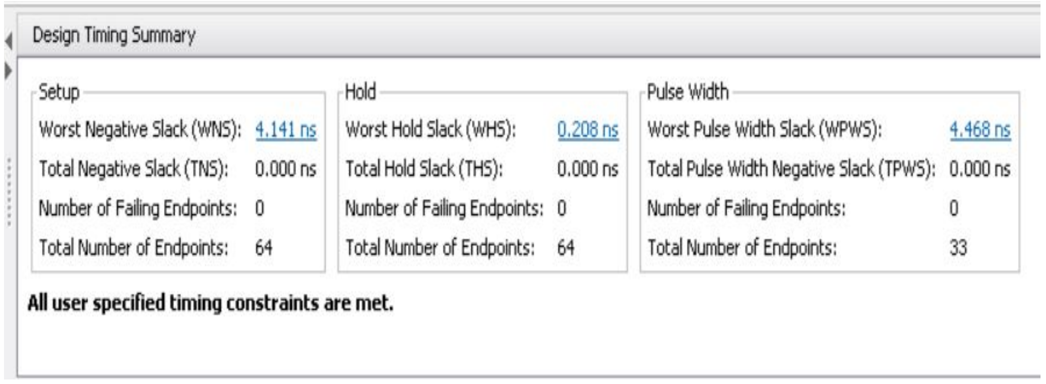

Post-Synthesis Simulation - Functional - sim_1 - tb_alu
Behavioral Simulation - Functional - sim_1 - tb_alu
Post-Synthesis Simulation - Timing - sim_1 - tb_alu
alu_n.v
tb_alu.v
constraints.xdc
Untitled 8
2,300.000 ns
Name
Value
0 ns
500 ns
1,000 ns
1,500 ns
2,000 ns
clock 1
rs1[31:0] 7cfde9f9
rs2[31:0] e33724c6
ctrl[3:0] a
outP[31:0] 9fcacd3f
XXXXXXXX 12153524 8484d609 06b97b0d 46df998d b2c28465 89375212 00f3e301 3b23f176 76d457ed 7cfde9f9
XXXXXXXX c0895e81 b1f05663 00000004 06d7cd0d 1e8dcd3d 462df78c e33724c6
X 1 2 3 4 5 6 7 8 9 a
000 XXXXXXXX d29e93a5 d2947fa6 006b97b0 6df998d0 fb2c2846 93752120 ff0c1cfe 1a01c134 76fdf7ed 9fcacd3f

Figure 17: Post synthesis timing simulation waveform

4.6 Synthesis Reports :

The values have been found after synthesis of the corresponding designs. I have done the experiment on Vivado 2014.1. The FPGA board selected is Artix-7. The LUTs and Flops have been found from the utilization report. The delay has been found from the timing report and the power has been found from the power report.

We have added proper constraints and the synthesis results are shown below.

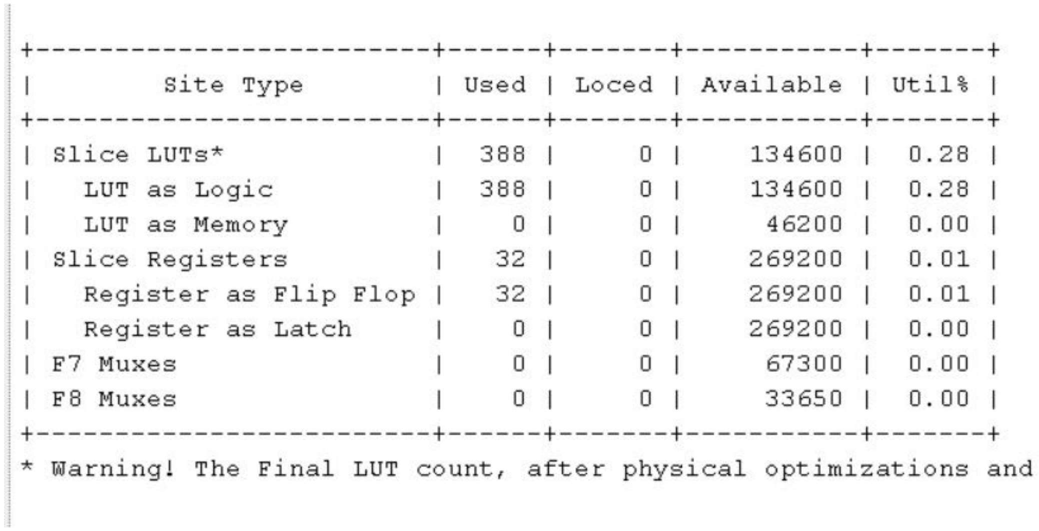


Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.141 ns	Worst Hold Slack (WHS):	0.208 ns	Worst Pulse Width Slack (WPWS):	4.468 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:	64	Total Number of Endpoints:	64	Total Number of Endpoints:	33

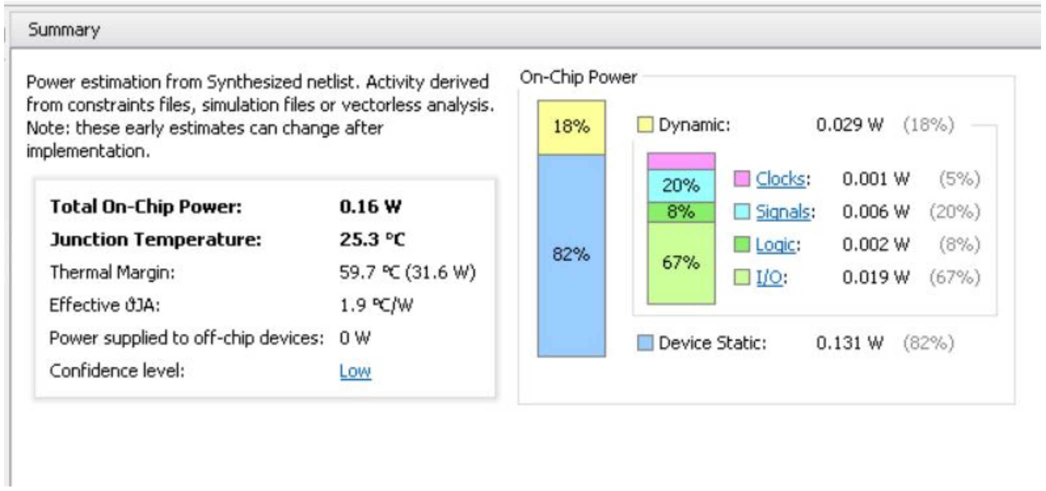
All user specified timing constraints are met.

Figure 18: Timing summary



```
+-------------------------+------+-------+-----------+-------+
|        Site Type        | Used | Loced | Available | Util% |
+-------------------------+------+-------+-----------+-------+
| Slice LUTs*             |  388 |     0 |    134600 |  0.28 |
|   LUT as Logic          |  388 |     0 |    134600 |  0.28 |
|   LUT as Memory         |    0 |     0 |     46200 |  0.00 |
| Slice Registers         |   32 |     0 |    269200 |  0.01 |
|   Register as Flip Flop |   32 |     0 |    269200 |  0.01 |
|   Register as Latch     |    0 |     0 |    269200 |  0.00 |
| F7 Muxes                |    0 |     0 |     67300 |  0.00 |
| F8 Muxes                |    0 |     0 |     33650 |  0.00 |
+-------------------------+------+-------+-----------+-------+
* Warning! The Final LUT count, after physical optimizations and
```

Figure 19: Utilization report



Summary

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**0.16 W**
Junction Temperature:	**25.3 °C**
Thermal Margin:	59.7 °C (31.6 W)
Effective ϑJA:	1.9 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

On-Chip Power

Dynamic:	0.029 W	(18%)
Clocks:	0.001 W	(5%)
Signals:	0.006 W	(20%)
Logic:	0.002 W	(8%)
I/O:	0.019 W	(67%)
Device Static:	0.131 W	(82%)

Figure 20: Power report

5 Conclusions :

- We have designed the circuit as per the requirement.

- The functionality of our design have been verified . The functionality are showing as expected.

- The different parameters of the design such as LUTs, delay and power have been calculated from the synthesis and tabulated.