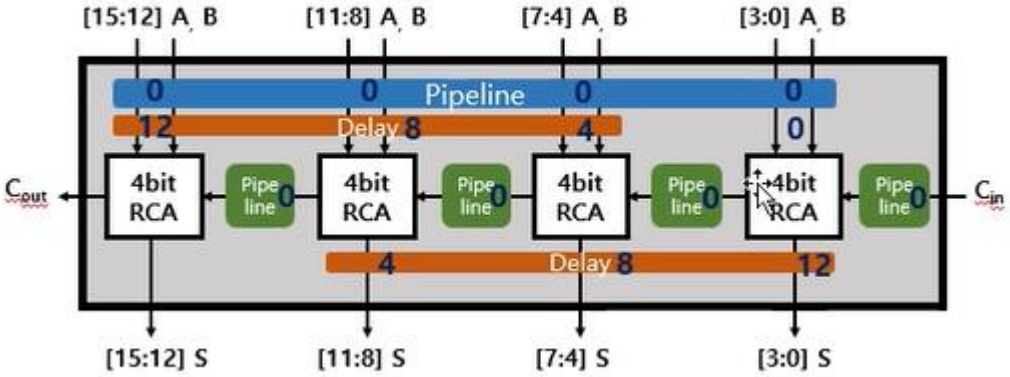
1. 16-bit Pipelined RCA 를 이용하여 Booth Multiplier(input: 8bit x2, output: 16bit)를 설계하고, Testbench 를 작성하여 5 가지 이상의 입력에 대한 시뮬레이션 결과를 보이시오.

우선 16-bit Pipielined RCA 는 다음과 같이 설계하였습니다.



저번 과제에서 설계한 4bit pipelined rca 4개를 이용하여 다음과 같이 delay를 주어 설계합니다.

4bit rca는 다음과 같습니다.

```
3   module full_adder(a, b, cin, s, cout);
4    input a, b, cin;
5    output s, cout;
6
7    assign s = a ^ b ^ cin;
8    assign cout = ( a & b ) | ( ( a ^ b ) & cin );
9
10  endmodule
```

```
12  module delay1(i,clk,o);
13   input i;
14   input clk;
15   output reg o;
16
17   always@(posedge clk)begin
18    o <= i;
19   end
20
21  endmodule
22
23  module delay2(i,clk,o);
24   input i;
25   input clk;
26   output reg o;
27
28   reg r_data;
29
30   always@(posedge clk)begin
31    r_data    <= i;
32    o         <= r_data;
33   end
34
35  endmodule
```

```
36
37  module delay3(i,clk,o);
38   input i;
39   input clk;
40   output reg o;
41
42   reg r_data[0:1];
43
44   always@(posedge clk)begin
45    r_data[0] <= i;
46    r_data[1] <= r_data[0];
47    o         <= r_data[1];
48   end
49
50  endmodule
51
52  module delay4(i,clk,o);
53   input i;
54   input clk;
55   output reg o;
56
57   reg r_data[0:2];
58
59   always@(posedge clk)begin
60    r_data[0] <= i;
61    r_data[1] <= r_data[0];
62    r_data[2] <= r_data[1];
63    o         <= r_data[2];
64   end
65
66  endmodule
```

```
68  module pprca_4bit(A,B,Cin,clk,S,Cout);
69
70   input [3:0]A,B;
71   input Cin;
72   input clk;
73   output [3:0]S;
74   output Cout;
75
76   wire [3:0]a, b;
77   wire [2:0]s;
78   wire [6:0]c;
79
80   delay1 dl0(A[0],clk,a[0]);
81   delay1 dl1(B[0],clk,b[0]);
82   delay2 dl2(A[1],clk,a[1]);
83   delay2 dl3(B[1],clk,b[1]);
84   delay3 dl4(A[2],clk,a[2]);
85   delay3 dl5(B[2],clk,b[2]);
86   delay4 dl6(A[3],clk,a[3]);
87   delay4 dl7(B[3],clk,b[3]);
88
89   delay1 dl8(Cin,clk,c[0]);
90   full_adder fa0(a[0],b[0],c[0],s[0],c[1]);
91   delay1 dl9(c[1],clk,c[2]);
92   full_adder fa1(a[1],b[1],c[2],s[1],c[3]);
93   delay1 dl10(c[3],clk,c[4]);
94   full_adder fa2(a[2],b[2],c[4],s[2],c[5]);
95   delay1 dl11(c[5],clk,c[6]);
96   full_adder fa3(a[3],b[3],c[6],S[3],Cout);
97
98   delay1 dl14(s[2],clk,S[2]);
99   delay2 dl13(s[1],clk,S[1]);
100  delay3 dl12(s[0],clk,S[0]);
101
102  endmodule
```

4개의 delay 모듈을 이용한 4bit pipelined rca입니다.

이제 16bit pipelined rca를 구성하기 위해, 4bit를 4, 8, 12clk 씩 delay 시켜주는 모듈이 필요합니다.

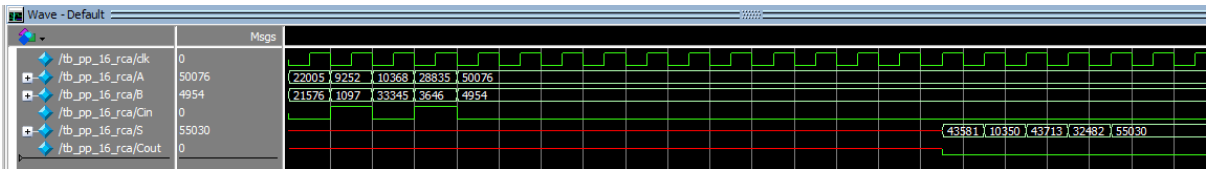
```
104 module delay4_4b(i,clk,o);
105  input [3:0]i;
106  input clk;
107  output reg [3:0]o;
108
109  reg [3:0]r_data[0:2];
110
111  always@(posedge clk)begin
112   r_data[0] <= i;
113   r_data[1] <= r_data[0];
114   r_data[2] <= r_data[1];
115   o         <= r_data[2];
116  end
117
118 endmodule
119
120 module delay8_4b(i,clk,o);
121  input [3:0]i;
122  input clk;
123  output reg [3:0]o;
124
125  reg [3:0]r_data[0:6];
126
127  always@(posedge clk)begin
128   r_data[0] <= i;
129   r_data[1] <= r_data[0];
130   r_data[2] <= r_data[1];
131   r_data[3] <= r_data[2];
132   r_data[4] <= r_data[3];
133   r_data[5] <= r_data[4];
134   r_data[6] <= r_data[5];
135   o         <= r_data[6];
136  end
137
138 endmodule
```

```
140 module delay12_4b(i,clk,o);
141  input [3:0]i;
142  input clk;
143  output reg [3:0]o;
144
145  reg [3:0]r_data[0:10];
146
147  always@(posedge clk)begin
148   r_data[0] <= i;
149   r_data[1] <= r_data[0];
150   r_data[2] <= r_data[1];
151   r_data[3] <= r_data[2];
152   r_data[4] <= r_data[3];
153   r_data[5] <= r_data[4];
154   r_data[6] <= r_data[5];
155   r_data[7] <= r_data[6];
156   r_data[8] <= r_data[7];
157   r_data[9] <= r_data[8];
158   r_data[10]<= r_data[9];
159   o         <= r_data[10];
160  end
161
162 endmodule
```

상위 모듈은 다음과 같이 설계합니다.

```
164  module pp_16_rca(clk, A, B, Cin, S, Cout);
165  input [15:0]A, B;
166  input clk;
167  input Cin;
168  output [15:0]S;
169  output Cout;
170
171  wire [2:0]c;
172  wire [15:4]a,b;
173  wire [11:0]s;
174
175   delay4_4b dly0(A[7:4],clk,a[7:4]);
176   delay4_4b dly1(B[7:4],clk,b[7:4]);
177   delay8_4b dly2(A[11:8],clk,a[11:8]);
178   delay8_4b dly3(B[11:8],clk,b[11:8]);
179   delay12_4b dly4(A[15:12],clk,a[15:12]);
180   delay12_4b dly5(B[15:12],clk,b[15:12]);
181
182   pprca_4bit pr0(A[3:0],B[3:0],Cin,clk,s[3:0],c[0]);
183   pprca_4bit pr1(a[7:4],b[7:4],c[0],clk,s[7:4],c[1]);
184   pprca_4bit pr2(a[11:8],b[11:8],c[1],clk,s[11:8],c[2]);
185   pprca_4bit pr3(a[15:12],b[15:12],c[2],clk,S[15:12],Cout);
186
187   delay12_4b dly6(s[3:0],clk,S[3:0]);
188   delay8_4b dly7(s[7:4],clk,S[7:4]);
189   delay4_4b dly8(s[11:8],clk,S[11:8]);
190
191  endmodule
```

Testbench를 이용한 시뮬레이션이 잘 동작하는 것을 다음과 같이 확인하였습니다.



12clk+4clk(4bit rca delay) = 16clk이 나오는 것을 확인할 수 있습니다.

이제 booth multiplier를 설계하기 위해 다음 사진을 참고합니다.

Ex) Radix-4 Booth Multiplier

000	001	010	011	100	101	110	111
0	+1	+1	+2	-2	-1	-1	0

+ : original
- : 2's com
1 : original
2 : left shift

```
        A =     1 0 1 1 0 1 0 1  (= -75)
X       B =     0 1 1 1 0 0 1 0  (= 114)
-------------------------------------------------
  0 0 0 0 0 0 0 0 1 0 0 1 0 1 1 0   (-2 2's of A & shift left)
  1 1 1 1 1 1 1 0 1 1 0 1 0 1       (+1 add A )
  0 0 0 0 0 1 0 0 1 0 1 1           (-1 2's of A)
  1 1 0 1 1 0 1 0 1 0               (+2 shift left A)
-------------------------------------------------
  1 1 0 1 1 1 1 1 0 1 0 0 1 1 0 1 0  (= -8550)
```

Check	Action
000	Add 0
001	Add multiplicand
010	Add multiplicand
011	Add 2*multiplicand (shift)
100	Subtract 2*multiplicand (shift)
101	Subtract multiplicand
110	Subtract multiplicand
111	Add 0

```
3  module booth_check(i,check,o,msb);
4          input [7:0]i;
5          input [2:0]check;
6          output reg [7:0]o;
7          output reg msb;
8
9          always @(*) begin
10                 case (check)
11                         3'b000, 3'b111 : begin
12                                 o = 8'b0000_0000;
13                                 msb = 1'b0;
14                         end
15
16                         3'b001, 3'b010 : begin
17                                 o = i;
18                                 msb = i[7];
19                         end
20
21                         3'b011 : begin
22                                 o = i <<1;
23                                 msb = i[7];
24                         end
25
26                         3'b100 : begin
27                                 o = (~i+1'b1) << 1;
28                                 msb = ~i[7];
29                         end
30
31                         3'b101, 3'b110 : begin
32                                 o = ~i+1'b1;
33                                 msb = ~i[7];
34                         end
35                 endcase
36         end
37 endmodule
```

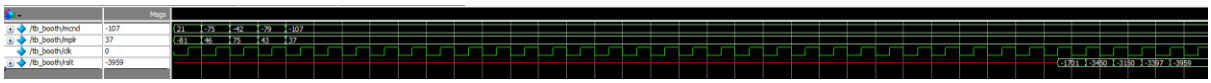
다음과 같이 case문을 이용합니다.

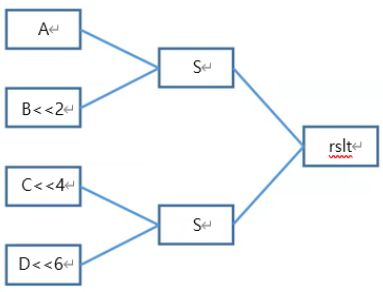
마지막으로 상위 모듈에서 16bit pipelined rca와 앞서 선언한 모듈을 이용하여 연산을 진행합니다.

```
39  module booth_top(a,b,clk,o);
40          input [7:0]a,b;
41          input clk;
42          output [15:0]o;
43
44          wire [7:0]pp0,pp1,pp2,pp3;
45          wire m0,m1,m2,m3;
46          wire [15:0] s0,s1;
47          wire cout1,cout2,cout3;
48
49          booth_check ck0(a,{b[1:0],1'b0},pp0,m0);
50          booth_check ck1(a,{b[3:1]},pp1,m1);
51          booth_check ck2(a,{b[5:3]},pp2,m2);
52          booth_check ck3(a,{b[7:5]},pp3,m3);
53
54          pp_16_rca bt_rca0(clk,{{8{m0}},pp0},{{6{m1}},pp1,2'b00},1'b0,s0,cout1);
55          pp_16_rca bt_rca1(clk,{{4{m2}},pp2,4'b0000},{{2{m3}},pp3,6'b000_000},1'b0,s1,cout2);
56          pp_16_rca bt_rca2(clk,s0,s1,1'b0,o,cout3);
57  endmodule
58
```

Testbench 결과는 다음과 같이 원활하게 값이 나오는 것을 확인할 수 있습니다.

```
59  module tb_booth();
60
61  reg [7:0]mcnd;
62  reg [7:0]mplr;
63  reg clk = 1'b0;
64
65  wire [15:0]rslt;
66
67  booth_top bt_n_0(mcnd, mplr, clk, rslt);
68
69  always #5 clk = ~clk;
70
71  initial begin
72  mcnd = 8'b0001_0101; mplr = 8'b1010_1111;
73  #10
74  mcnd = 8'b1011_0101; mplr = 8'b0010_1110;
75  #10
76  mcnd = 8'b1101_0110; mplr = 8'b0100_1011;
77  #10
78  mcnd = 8'b1011_0001; mplr = 8'b0010_1011;
79  #10
80  mcnd = 8'b1001_0101; mplr = 8'b0010_0101;
81
82  end
```

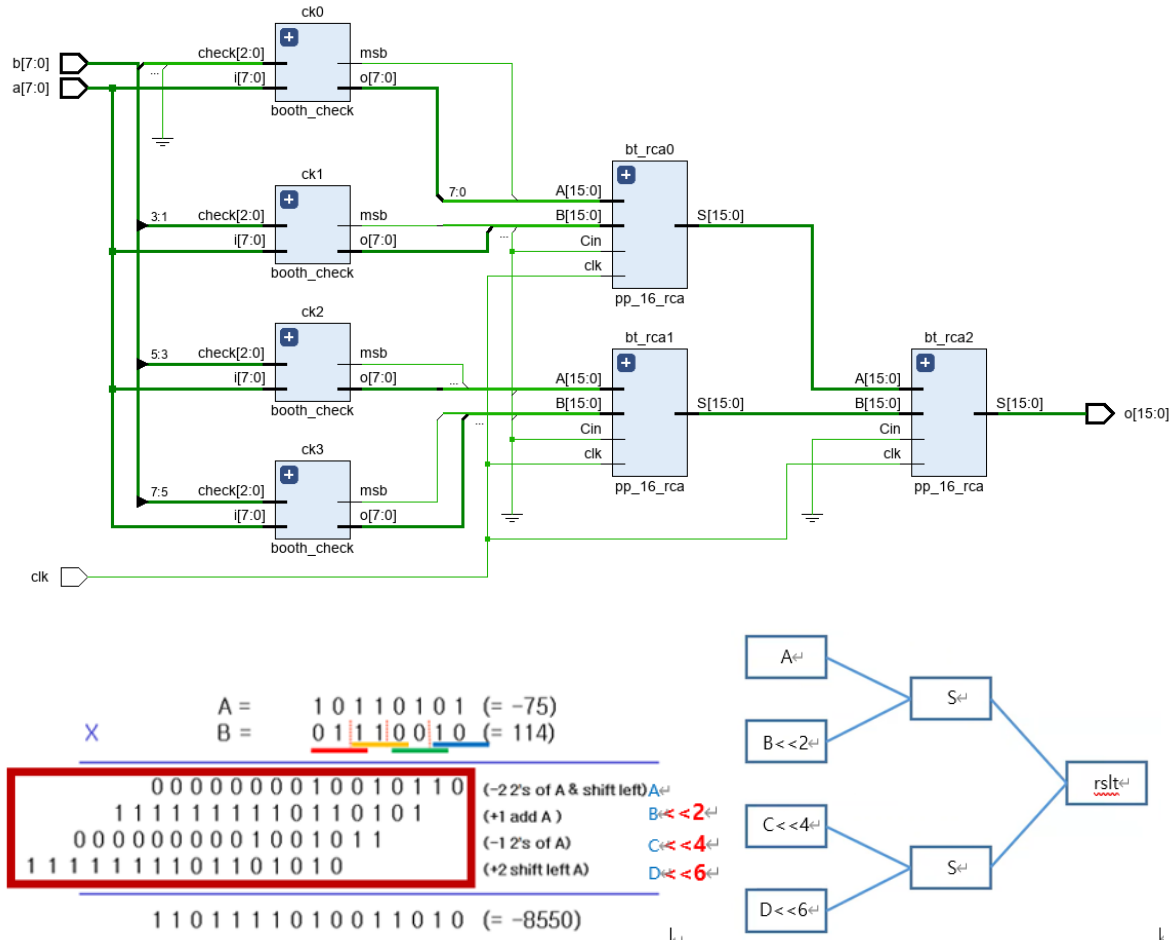




좌측의 그림처럼 2단계에 걸쳐 rca가 동작하기 때문에 16+16=32clk 지연이 생김을 확인할 수 있습니다.

2. Xilinx Vivado 를 이용하여 I 에서 설계한 Booth Multiplier 의 RTL Analysis(Elaborated Design) Schematic 을 얻고, 자신이 의도한 Architecture 와 일치하는지 최상위 모듈과 2 가지 하위 모듈을 선택하여 비교/분석 하시오. (3 점)

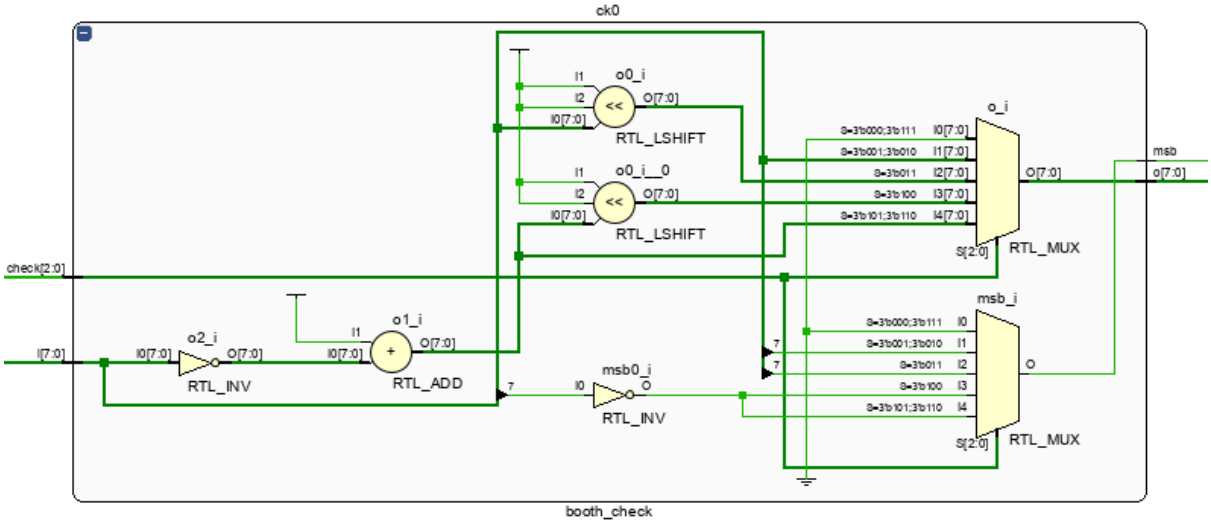
Vivado 를 이용한 schematic 은 다음과 같다.



위 그림처럼 상위모듈을 설계했기 때문에 schematic의 전체적인 모습을 봤을 때 원하는대로 설계되었음을 알 수 있다.

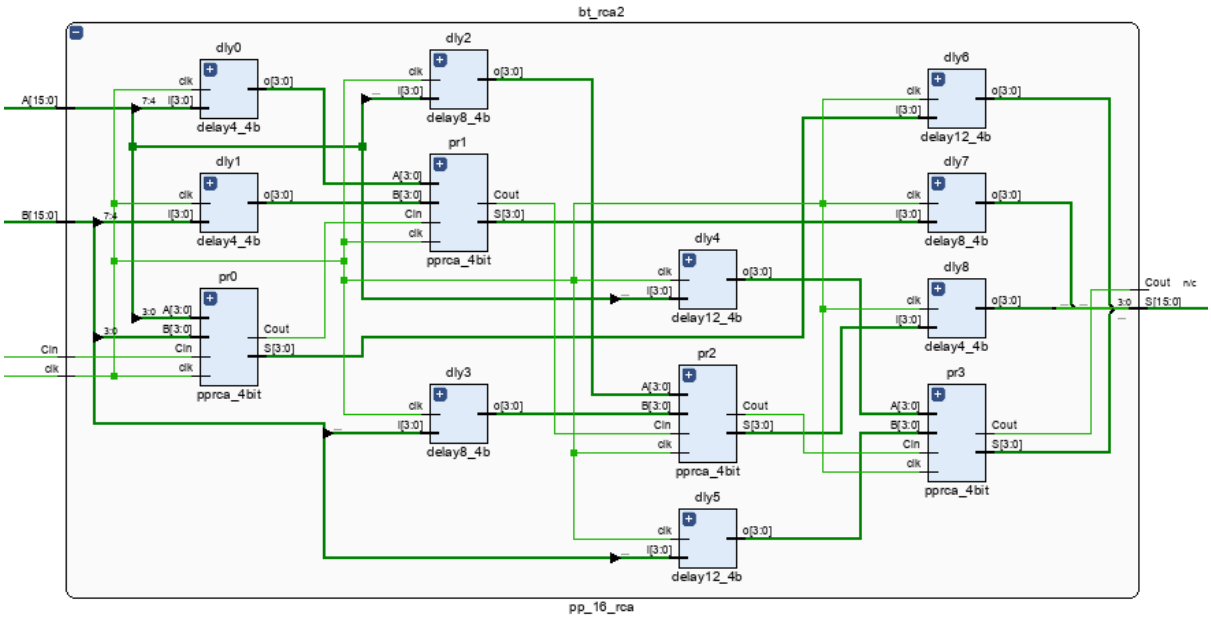
좌측 4개의 모듈을 통해 16bit의 값을 만들고 각각 2개씩 rca를 이용하여 연산한다. 이 과정에서 2,4,6 bit 씩 shift되고, 마지막에 다시 그 2값을 rca를 이용하여 연산하는 모습을 확인할 수 있다.

하위 모듈을 자세히 살펴보도록 하겠다.

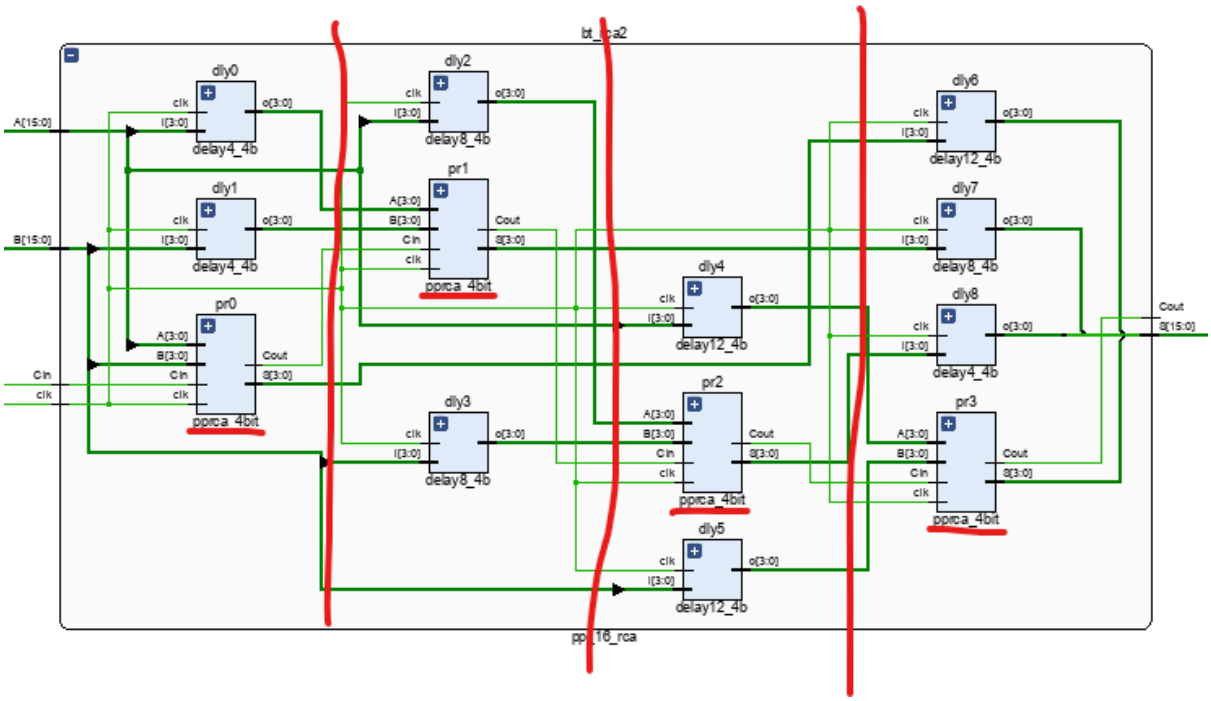


Case 문에서 1bit와 8bit짜리 data에 각각 값을 할당하기 때문에 2개의 mux가 생성된 것을 볼 수 있다. 5가지 경우에 대해 구분되어 있었으므로 MUX에 들어가는 bus가 5개가 존재하는 것을 확인할 수 있다. 각 경우에 따라 inverse를 취하거나 1bit를 더하는 등의 연산이 겹치는 부분은 공유되어 있음을 볼 수 있다.

다음은 16bit_pipelined rca를 살펴보겠다.



전체적인 모습은 위와 같다.



좌측부터 하나씩 4bit rca가 있는 모습을 확인할 수 있고 4bit rca의 위나 아래에 입력을 위한 delay 모듈이 존재하는 것을 볼 수 있습니다. 가장 우측에는 출력의 delay를 위한 모듈이 존재하는 것을 확인할 수 있습니다.

3. Xilinx Vivado 를 이용하여 I 에서 설계한 Booth Multiplier 의 Synthesis, Implementation 을 진행하시오. 모든 Critical Warning 은 해결하고, 필요시 Constraint 파일을 수정하시오. Synthesis 와 Implementation 각각에 해당하는 Timing, Power, Utilization Summary 를 분석하시오. (3 점)

Synthesis를 진행한 결과는 다음과 같습니다.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.896 ns	Worst Hold Slack (WHS):	0.100 ns	Worst Pulse Width Slack (WPWS):	4.020 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:	155	Total Number of Endpoints:	155	Total Number of Endpoints:	192

All user specified timing constraints are met.

Design Runs | **Utilization** | Timing | Package Pins | I/O Ports

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (285)	BUFGCTRL (32)
booth_top	179	122	33	1
bt_rca0 (pp_16_rca)	60	42	0	0
bt_rca1 (pp_16_rca_0)	57	29	0	0
bt_rca2 (pp_16_rca_1)	62	51	0	0

Design Runs | **Power** | Utilization | Timing | Package Pins | I/O Ports

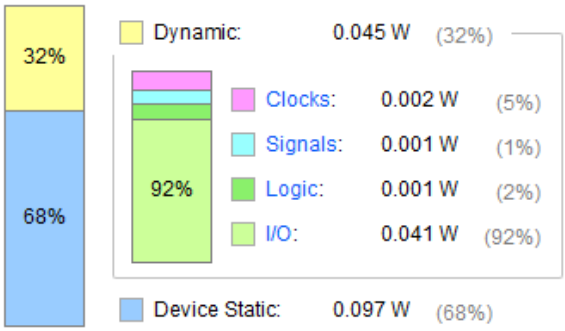
Summary

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:	**0.142 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (22.2 W)
Effective ϑJA:	2.7°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power



Implementation한 결과는 다음과 같습니다.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.384 ns	Worst Hold Slack (WHS):	0.042 ns	Worst Pulse Width Slack (WPWS):	4.020 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:	155	Total Number of Endpoints:	155	Total Number of Endpoints:	192

All user specified timing constraints are met.

Design Runs | Power | DRC | Methodology | Timing | **Utilization**

Hierarchy

Name	Slice LUTs (63400)	Slice Registers (126800)	Slice (15850)	LUT as Logic (63400)	LUT as Memory (19000)	Bonded IOB (285)	BUFGCTRL (32)
booth_top	171	122	58	102	69	33	1
bt_rca0 (pp_16_rca)	58	42	27	42	16	0	0
bt_rca1 (pp_16_rca_0)	55	29	28	39	16	0	0
bt_rca2 (pp_16_rca_1)	62	51	28	25	37	0	0

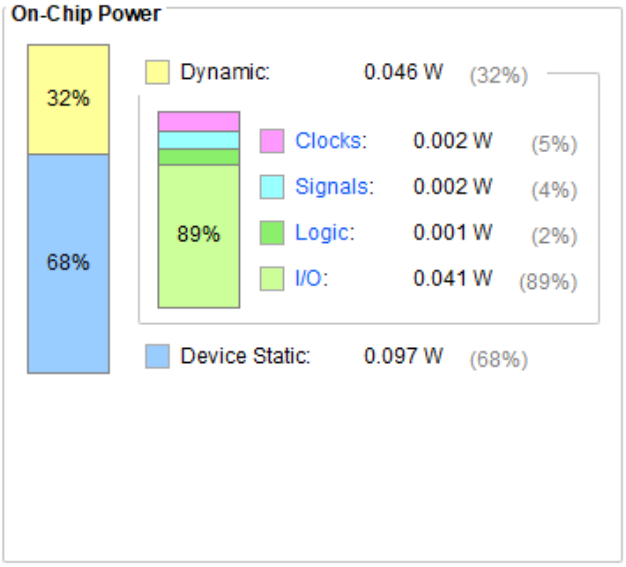
Design Runs | **Power** | DRC | Methodology | Timing | Utilization

Summary

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

Total On-Chip Power:	**0.143 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.4°C**
Thermal Margin:	59.6°C (22.2 W)
Effective ϑJA:	2.7°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



먼저 timing 부분에서는 TNS,THS가 0이므로 전체 negative slack이 없고, WNS, WHS가 양의 값을 갖는것으로 보아 설계가 잘 된 것으로 생각됩니다.

bounded IOB가 33인 이유는 8bit의 input 두개와 16bit output 하나와 clk신호 1개의 port를 합쳐서 33개임을 확인할 수 있습니다. BUFGCTRL는 clk을 위한 1개의 버퍼인 것 같습니다.

차이가 존재하는 부분을 비교하기 위에 다음과 같이 표를 만들었습니다.

<table>
<tr><th>schematic</th><th>implementation</th></tr>
<tr><td>

Hold

Worst Hold Slack (WHS):	0.100 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	155

</td><td>

Hold

Worst Hold Slack (WHS):	0.042 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	155

</td></tr>
<tr><td>

Name	Slice LUTs (63400)
booth_top	179
bt_rca0 (pp_16_rca)	60
bt_rca1 (pp_16_rca_0)	57
bt_rca2 (pp_16_rca_1)	62

</td><td>

Name	Slice LUTs (63400)
booth_top	171
bt_rca0 (pp_16_rca)	58
bt_rca1 (pp_16_rca_0)	55
bt_rca2 (pp_16_rca_1)	62

</td></tr>
<tr><td>

Total On-Chip Power: **0.142 W**

</td><td>

Total On-Chip Power: **0.143 W**

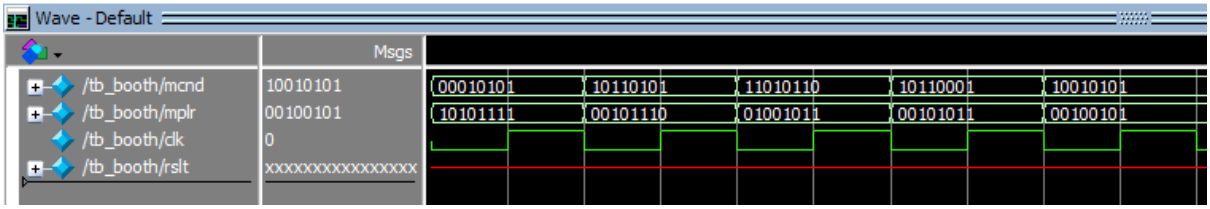
</td></tr>
</table>

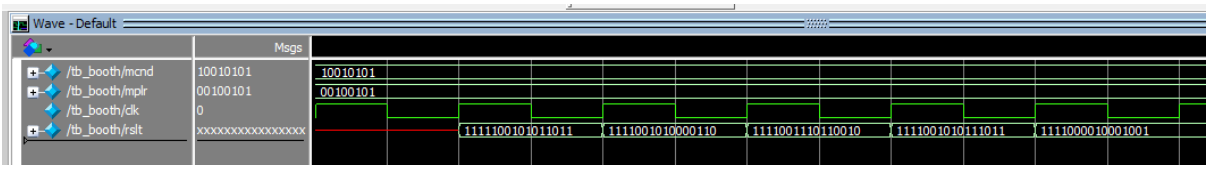
Slack이 증가하고 LUTs의 개수도 줄었으므로 implementation이 긍정적인 방향으로 이루어졌다고 생각합니다. 하지만 power는 미미하게 증가하였는데, implementation단계에서는 porting이 이루어졌기 때문에 power가 증가하지 않았나 추측해봅니다.

4. FPGA 보드의 DIP 스위치와 LED 를 이용하여 Booth Multiplier 의 입출력 결과를 확인하시오.

Name	Direction	Neg Diff Pair	Package Pin	Fixed	Bank	I/O Std	Vcco
a (8)	IN			☑	35	LVCMOS15*	1.500
a[7]	IN		J4	☑	35	LVCMOS15*	1.500
a[6]	IN		L3	☑	35	LVCMOS15*	1.500
a[5]	IN		K3	☑	35	LVCMOS15*	1.500
a[4]	IN		M2	☑	35	LVCMOS15*	1.500
a[3]	IN		K6	☑	35	LVCMOS15*	1.500
a[2]	IN		J6	☑	35	LVCMOS15*	1.500
a[1]	IN		L5	☑	35	LVCMOS15*	1.500
a[0]	IN		L4	☑	35	LVCMOS15*	1.500
b (8)	IN			☑	13	LVCMOS33*	3.300
b[7]	IN		Y16	☑	13	LVCMOS33*	3.300
b[6]	IN		AA16	☑	13	LVCMOS33*	3.300
b[5]	IN		AB16	☑	13	LVCMOS33*	3.300
b[4]	IN		AB17	☑	13	LVCMOS33*	3.300
b[3]	IN		AA13	☑	13	LVCMOS33*	3.300
b[2]	IN		AB13	☑	13	LVCMOS33*	3.300
b[1]	IN		AA15	☑	13	LVCMOS33*	3.300
b[0]	IN		AB15	☑	13	LVCMOS33*	3.300
o (16)	OUT			☑	(Multiple)	LVCMOS33*	3.300
o[15]	OUT		F15	☑	16	LVCMOS33*	3.300
o[14]	OUT		F13	☑	16	LVCMOS33*	3.300
o[13]	OUT		F14	☑	16	LVCMOS33*	3.300
o[12]	OUT		F16	☑	16	LVCMOS33*	3.300
o[11]	OUT		E17	☑	16	LVCMOS33*	3.300
o[10]	OUT		C14	☑	16	LVCMOS33*	3.300
o[9]	OUT		C15	☑	16	LVCMOS33*	3.300
o[8]	OUT		E13	☑	16	LVCMOS33*	3.300
o[7]	OUT		AA10	☑	13	LVCMOS33*	3.300
o[6]	OUT		AA11	☑	13	LVCMOS33*	3.300
o[5]	OUT		V10	☑	13	LVCMOS33*	3.300
o[4]	OUT		W10	☑	13	LVCMOS33*	3.300
o[3]	OUT		Y11	☑	13	LVCMOS33*	3.300
o[2]	OUT		Y12	☑	13	LVCMOS33*	3.300
o[1]	OUT		W11	☑	13	LVCMOS33*	3.300
o[0]	OUT		W12	☑	13	LVCMOS33*	3.300
Scalar ports (1)							
clk	IN		R4	☑	34	LVCMOS33*	3.300

위와 같이 보드에 연결하였습니다.





mcnd	0001_0101	1011_0101
Mplr	1010_1111	0010_1110
rslt	1111_1001_0101_1011	1111_0010_1000_0110
		
1101_0110		1011_0001
0100_1011		0010_1011
1111_0011_1011_0010		1111_0010_1011_1011
		
1001_0101		
0010_0101		
1111_0000_1000_1001		
		

결과가 위와 같이 testbench simulation과도 동일하게 나오는 것을 확인할 수 있습니다.