A Report on
DIGITAL ASSIGNMENT

Submitted in partial fulfilment of the course
EEE/INSTR F313: Analog and Digital VLSI design

By:

Parth Kalgaonkar	2016A3PS0268P
Pavni Jaiswal	2016A8PS0357P
Aditi Gupta	2016A3PS0262P

Birla Institute of technology and Science, Pilani
October 2018

Contents

1 Problem Statement

1.1 Problem 35

(a) Design of a XOR gate

Design and simulate Transmission gate based 3-input XOR gate at 1GHz.

(b) Implement an eight-bit multiplier using Radix-4 Booth's algorithm using behavioral modeling.

2 Design of XOR gate

We have designed an Exclusive-OR gate using transmission gates only. The steps followed in achieving the same are as follows:

2.1 Schematic in Cadence

Two schematics have been designed for the implementation of the XOR gate using transmission gates. In the schematic, the transmission gates themselves have been designed using NMOS and PMOS.

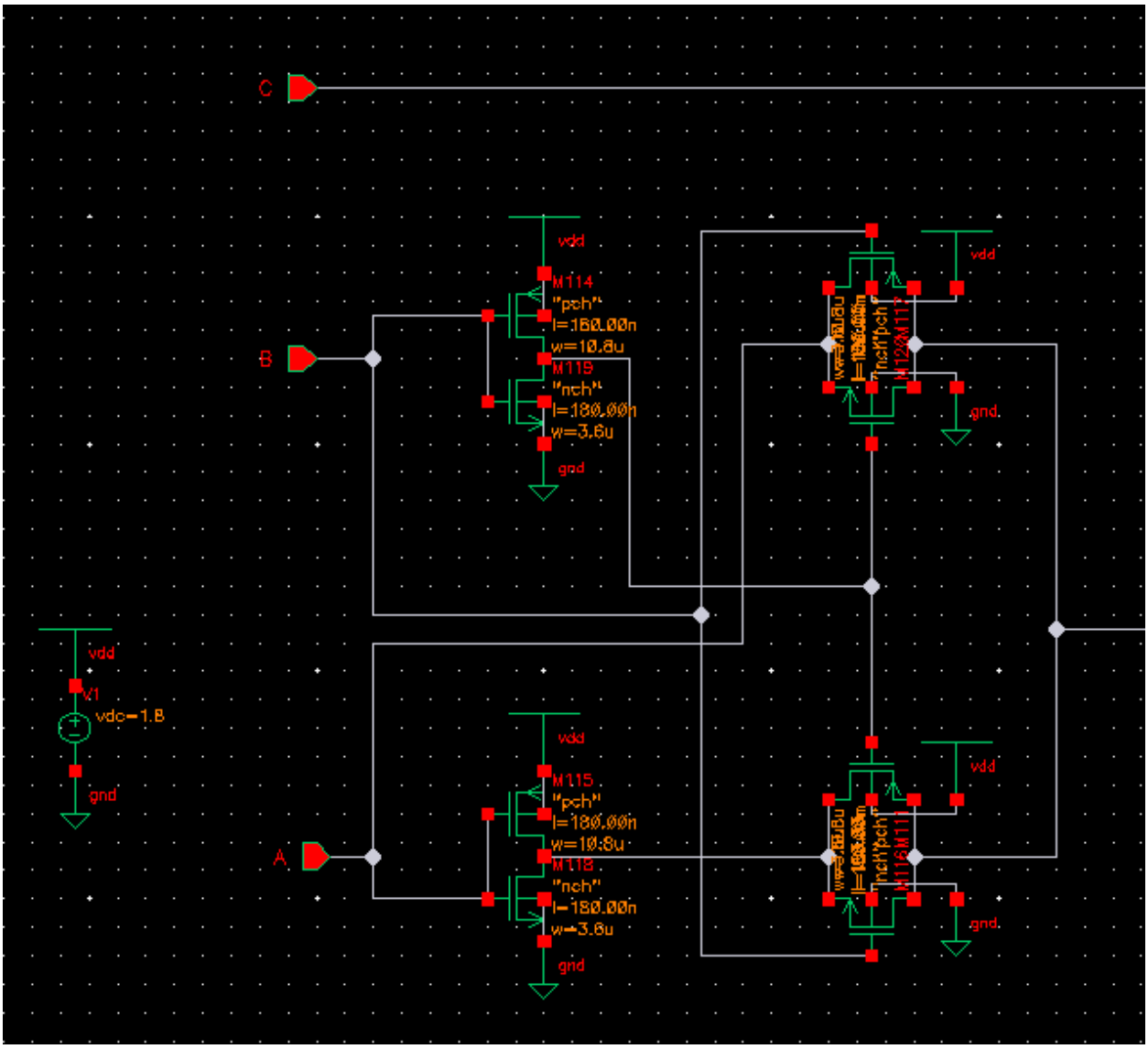
2.1.1 Specifications

For transmission gate:

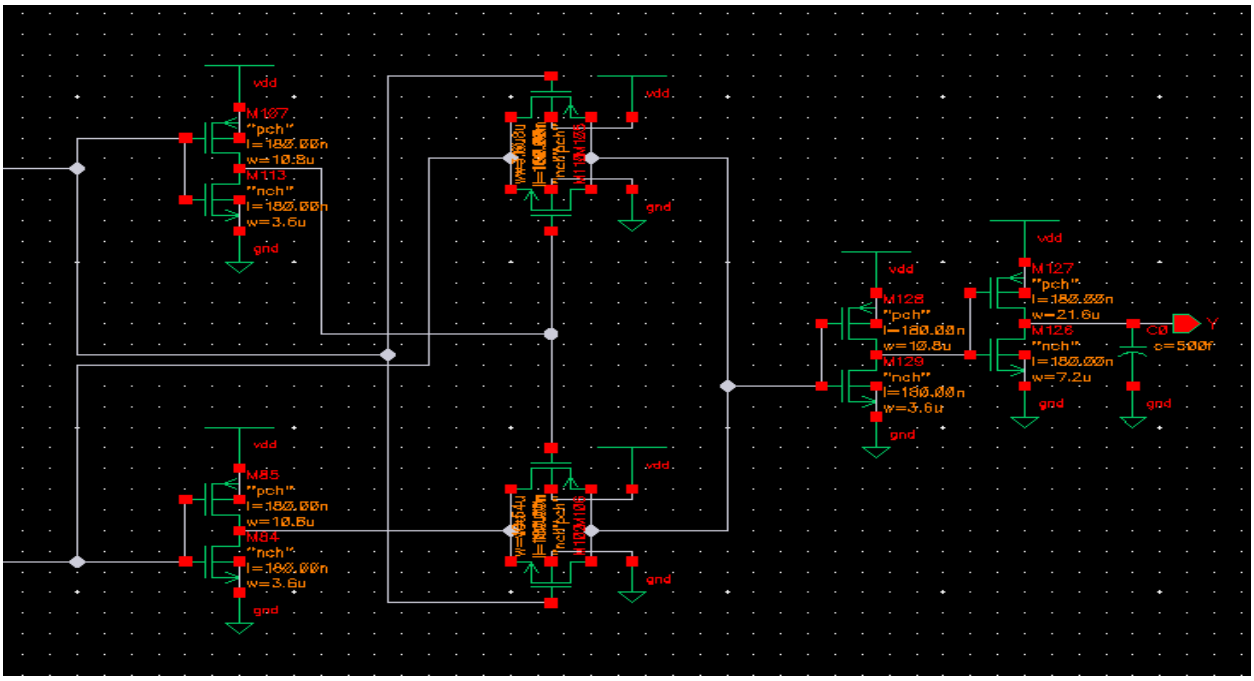
Each NMOS: W/L = 20

Each PMOS: W/L = 60

For L = 180 nm technology, WNMOS= 3.6 um, WPMOS= 10.80 um



Here, the two TGs take A and B as input and give AB+A’B’ as output.



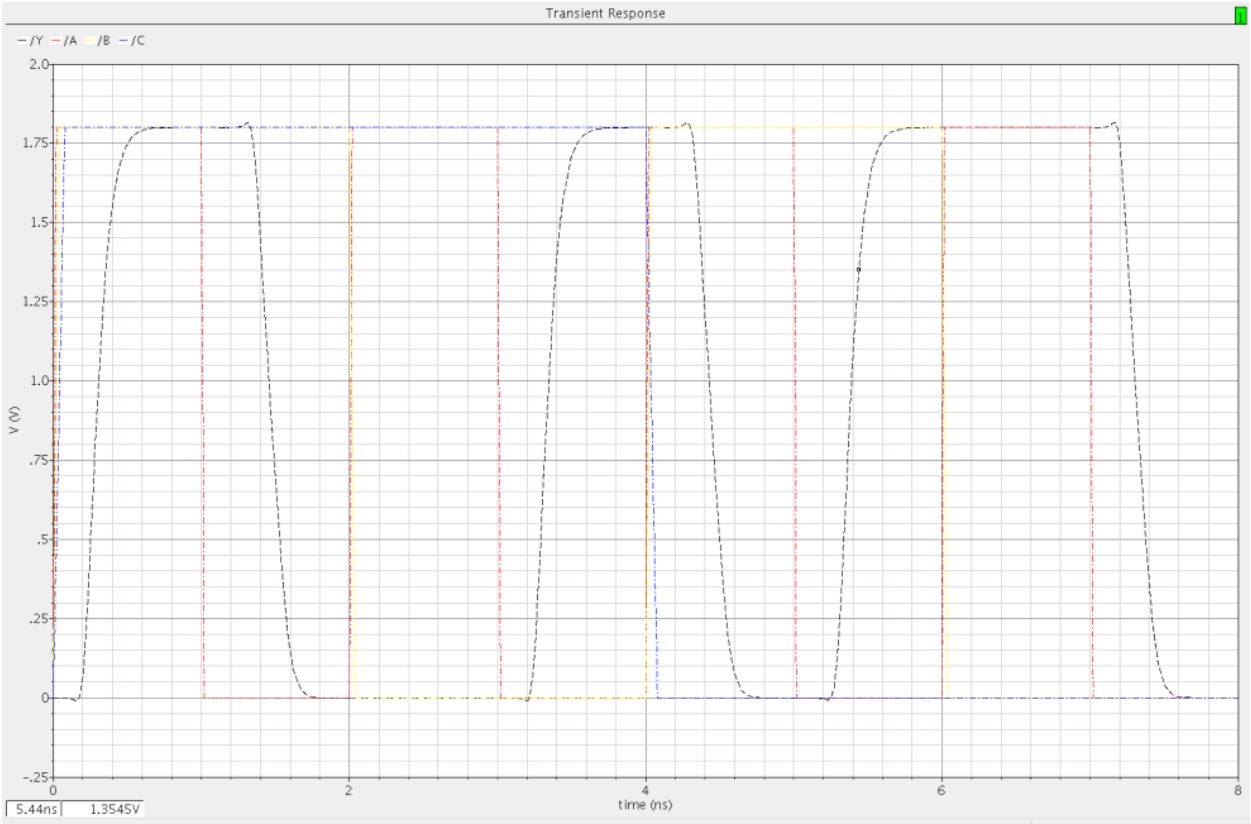
The next two TGs take C and result of first two TGs, as input and give AB'C'+A'BC'+A'B'C+ABC as output.

2.2 Truth Table

A	B	C	Output
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

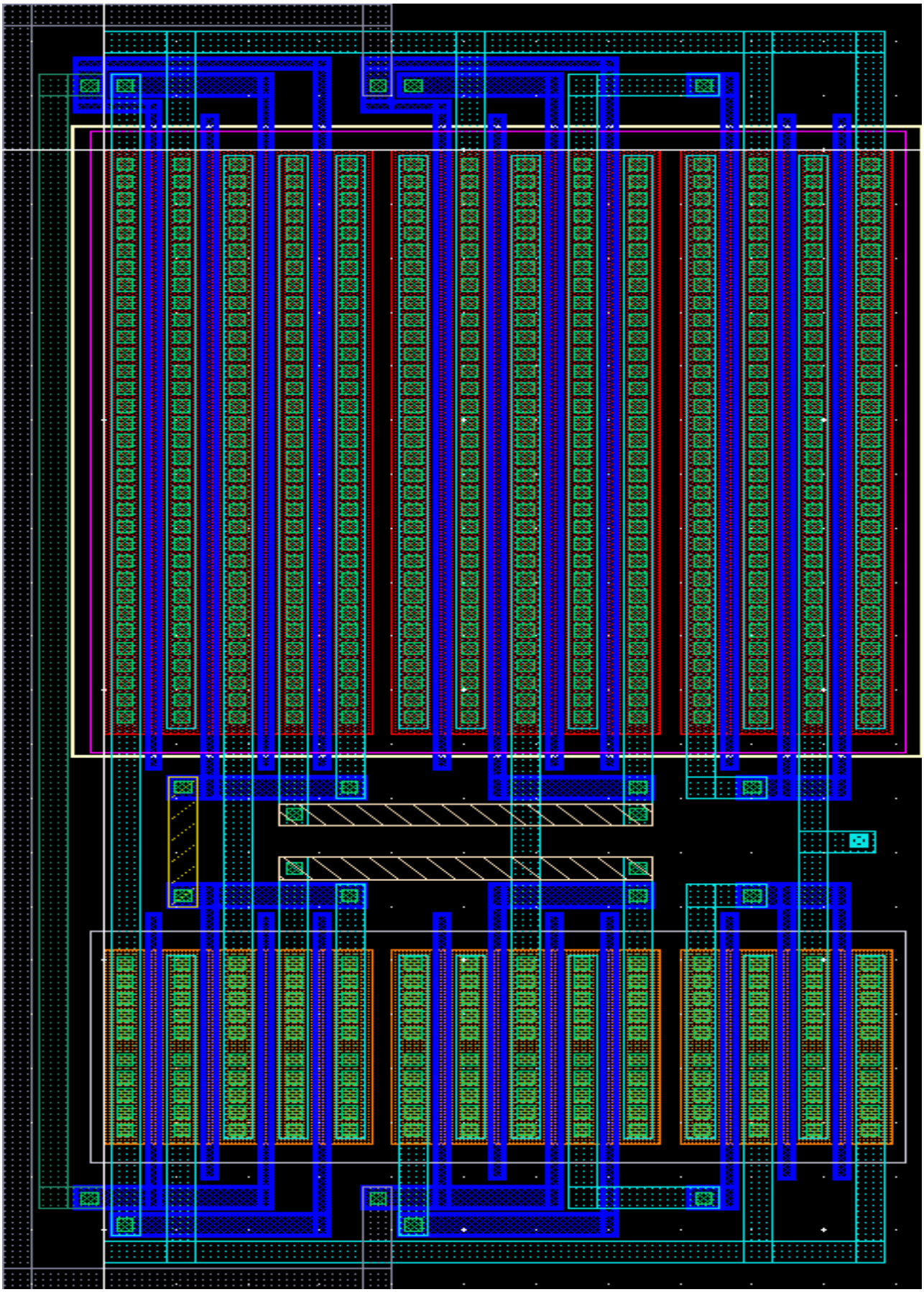
The schematic gives the final desired output as displayed in the timing diagram below:

2.3 Timing Diagram



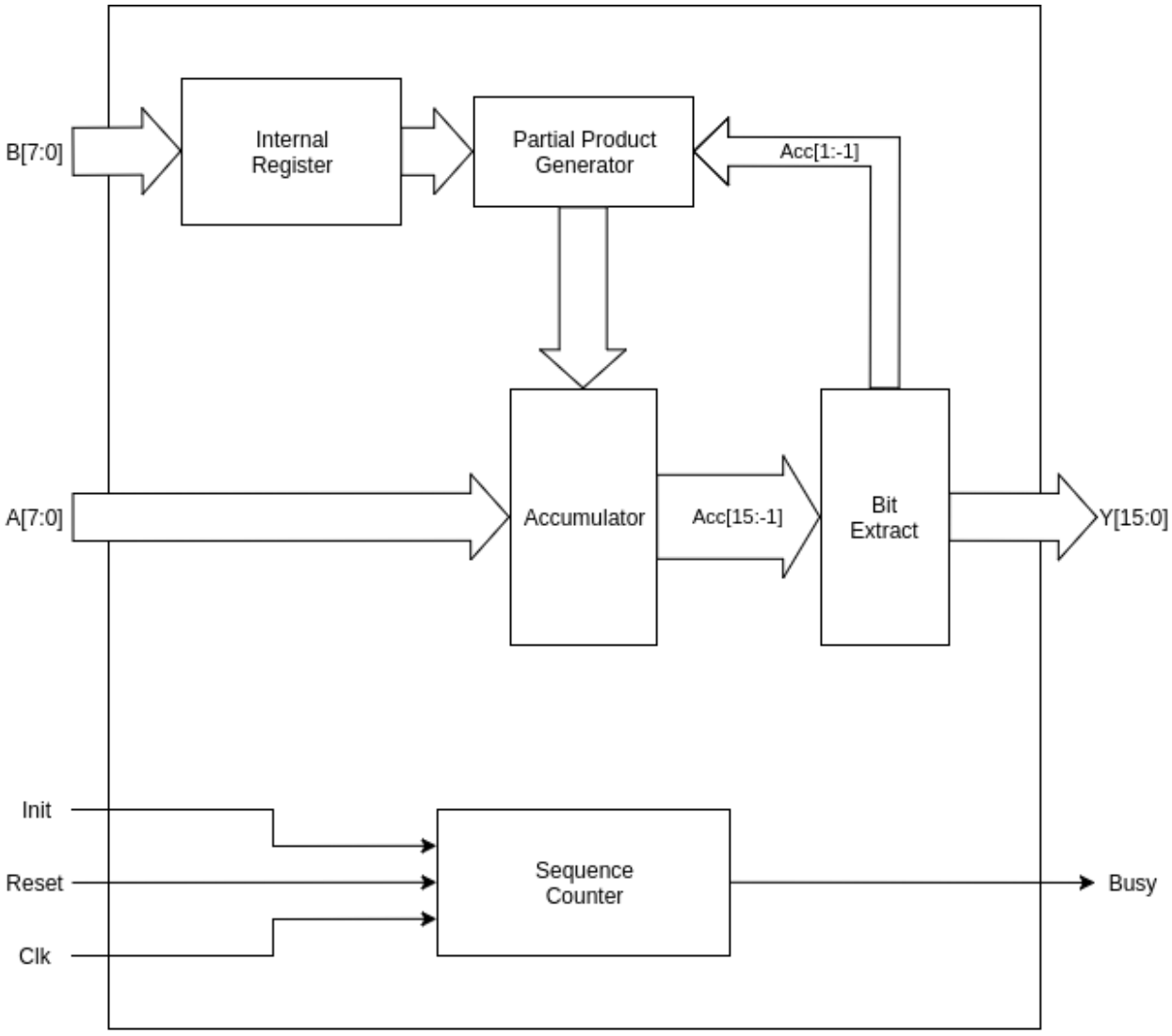
2.4 Layout in Cadence

Cadence software has been used to design the layout corresponding to the circuit shown in the schematic above.



3 Radix-4 Booth multiplier

We implemented booth multiplier using the following modules.



8-bit Radix-4 Booth Multiplier

3.1 Pin description

3.1.1 Inputs

- **A[7:0]** – This is the multiplier (or the first operand in multiplication).
- **B[7:0]** – The multiplicand (or second operand in multiplication).
- **Clk** – The input clock to the system for synchronization purposes. All system functions are positive edge triggered.

- **Reset** – System level active high reset signal. A logic **1** on this pin clears the sequence counter, **accumulator** and the internal registers. Logic **0** has no effect.
- **Init** – Start calculation. A logic **1** on this pin starts computation loads the internal register with the present value of **A** and the **accumulator** with the present value of **B**.

Note: Both **Reset** and **Init** are synchronous (changes are affected on the next rising edge of clock).

3.1.2 Outputs

- **Y[15:0]** – The result of multiplication. This value is only valid when the **busy** output is low. A reset clears the output.
- **Busy** – Logic **1** on this pin indicates that the multiplier is functioning to produce the correct output. This indicates that **sequence counter** is non-zero.

3.2 Internal modules description

3.2.1 Sequence Counter

This acts as the controller for the whole multiplier, generates internal signals for other modules, handles resets, start commands etc.

Contains a counter that counts down from four (the number of cycles req for 8-bit multiplication). As soon as count reaches 0, the **busy** signal is turned off indicating completion of operation.

3.2.2 Accumulator

Stores the intermediate result of adding partial products.

At every clock edge, the partial product (from the partial products generator) is added to the contents of the **accumulator** and the result is shifted (arithmetically) right by 2.

The implicit [-1] bit of the result is also stored in the **accumulator** and is used to find the partial products by the **partial product generator**.

3.2.3 Bit extract

The output of the **accumulator** is 17 bits (including the implicit [-1] bit). Of these, the most significant 16 are the actual bits of the result and the least significant three are used by the **partial product generator**. **Bit extract** separates the two sets of bits, assigns the most significant 16 to the output and the least significant 3 as an input to the **partial products generator**.

3.2.4 Partial product generator

Based on the least significant three bits of the accumulator it generates the partial product for each iteration of the algorithm.

Acc[1]	*Acc[0]*	*Acc[-1]*	*Partial Product*
0	0	0	*0*
0	0	1	*B*
0	1	0	*B*
0	1	1	*2*B*
1	0	0	*-2*B*
1	0	1	*-B*
1	1	0	*-B*
1	1	1	*0*

Table 1: Partial Products in Radix-4 Booth

3.2.5 Internal register

This stores the value of the second operand **B** so that the device can be directly connected to system buses and the bus can function freely while the multiplier is busy.

3.3 Verilog code

3.3.1 Top level module

```
module mult_8bit(
    input [7:0]a,
    input [7:0]b,
    input clk,
    input reset,
    input init,
    output [15:0]y,
    output busy
);

    wire [2:0]x;
    wire [7:0]m;
    wire [16:0]z;
    wire [7:0]c;

    seq_count sc(
        .clk(clk),
        .reset(reset),
        .init(init),
        .busy(busy)
    );

    int_reg ir(
        .clk(clk),
        .init(init),
        .reset(reset),
```

```
27          .a_in(a),
28          .a_out(c)
29      );
30
31      bit_extract be(
32          .a(z),
33          .x(x),
34          .y(y)
35      );
36
37      part_prod pp(
38          .x(x),
39          .a(c),
40          .m(m)
41      );
42
43      acc ac(
44          .q(b),
45          .m(m),
46          .clk(clk),
47          .init(init),
48          .busy(busy),
49          .reset(reset),
50          .a(z)
51      );
52  endmodule
```

3.3.2 Sequence counter

```
 1  module seq_count(
 2      input clk,
 3      input reset,
 4      input init,
 5      output reg busy
 6  );
 7
 8      reg [3:0]a;
 9
10      always @(posedge clk) begin
11          if(reset) a <= 0;
12          else if (init) a <= 4;
13          else if(a>0) a <= a-1;
14      end
15
16      always @(*) begin
17          if(a==0) busy = 0;
18          else busy = 1;
19      end
20  endmodule
```

3.3.3 Accumulator

```
 1  module acc(
 2      input [7:0]q,
```

```
    input [7:0]m,
    input clk,
    input init,
    input busy,
    input reset,
    output reg [16:0]a
);

    always @(posedge clk) begin
        if(init) begin
            a[8:1] <= q;
        end
        else if (reset) begin
            a <= 0;
        end
        else if (busy) begin
            a<=$signed(a+(m<<9))>>>2;
        end
    end
endmodule
```

3.3.4 Bit Extract

```
module bit_extract(
    input [16:0]a,
    output [2:0]x,
    output [15:0]y
);

    assign x = a[2:0];
    assign y = a[16:1];

endmodule
```

3.3.5 Partial Product Generator

```
module part_prod(
    input [2:0]x,
    input [7:0]a,
    output reg [7:0]m
);

    always @(*) begin
        case(x)
            0: m = 0;
            1: m = a;
            2: m = a;
            3: m = a<<1;
            4: m = -(a<<1);
            5: m = -a;
            6: m = -a;
            7: m = 0;
        default: m = 0;
        endcase
```

```
19      end
20  endmodule
```

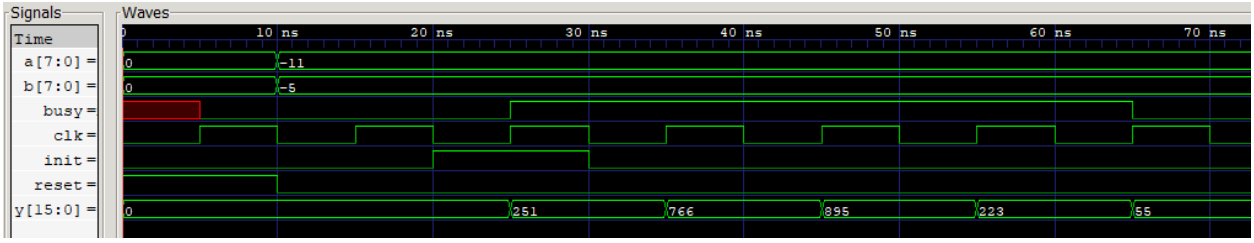
3.3.6 Internal Register

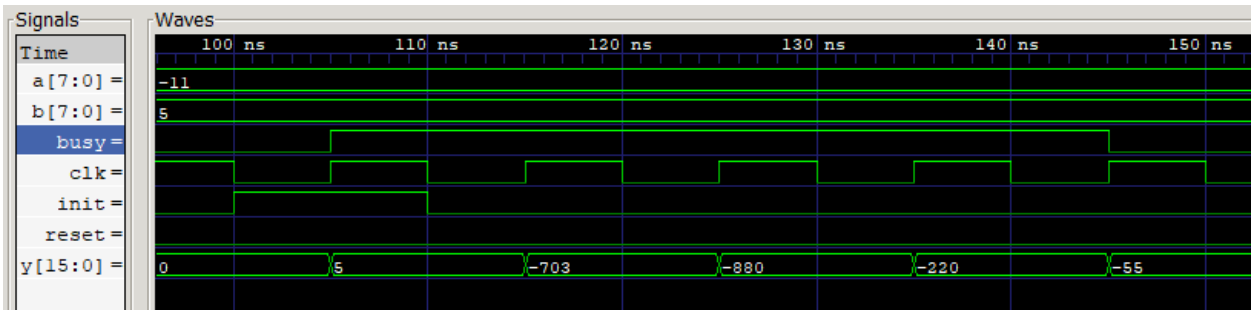
```
 1  module int_reg(
 2      input clk,
 3      input init,
 4      input reset,
 5      input [7:0]a_in,
 6      output reg [7:0]a_out
 7  );
 8
 9      always @(posedge clk) begin
10          if (reset) a_out<=0;
11          else if(init) a_out<=a_in;
12      end
13  endmodule
```

3.4 Simulation Results

The design was compiled using Icarus Verilog, and the simulation waveform is displayed using gtkwave.

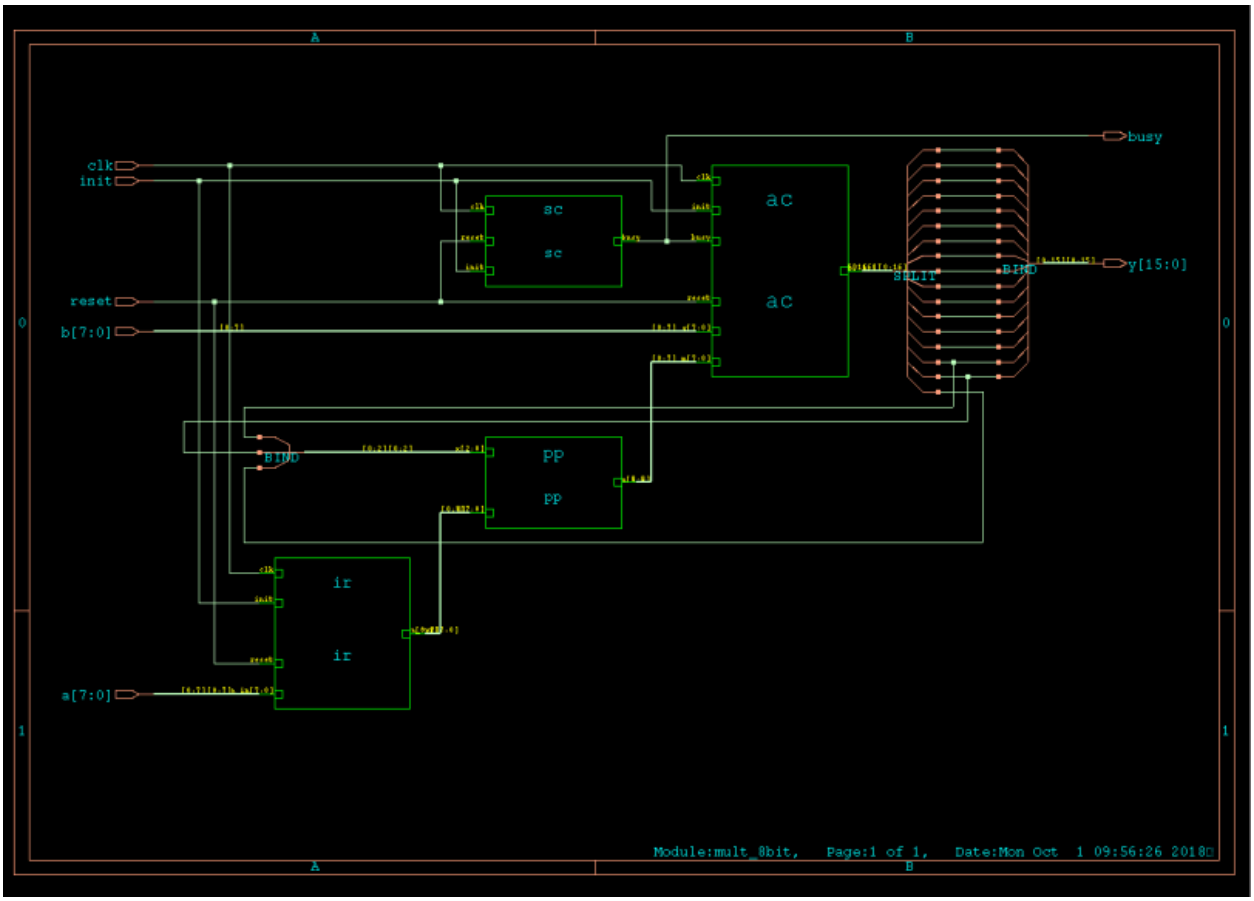
The simulation tests two input sets (-11x5) and (-11x-5) in succession. However, the results are shown separately for readability.





3.5 Synthesis Results

3.5.1 Synthesized circuit



3.5.2 Synthesis reports

3.5.2.1 Netlist

```
1  // Generated by Cadence Encounter(R) RTL Compiler v08.10-s116_1
2
3  module acc(q, m, clk, init, busy, reset, a);
4    input [7:0] q, m;
5    input clk, init, busy, reset;
6    output [16:0] a;
7    wire [7:0] q, m;
8    wire clk, init, busy, reset;
9    wire [16:0] a;
10   wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
11   wire n_8, n_9, n_10, n_11, n_12, n_13, n_14, n_15;
12   wire n_16, n_17, n_18, n_19, n_20, n_21, n_22, n_23;
13   wire n_24, n_25, n_26, n_27, n_28, n_29, n_30, n_31;
14   wire n_32, n_33, n_34;
15   QDFFRLX1 \a_reg[14] (.CK (clk), .D (n_33), .Q (a[14]));
16   QDFFRLX1 \a_reg[15] (.CK (clk), .D (n_34), .Q (a[15]));
17   QDFFRLX1 \a_reg[16] (.CK (clk), .D (n_32), .Q (a[16]));
18   QDFCLRBRLX1 \a_reg[13] (.CK (clk), .D (n_28), .LD (n_7), .RB (n_0),
```

```
19        .Q (a[13]));
20    AO12RLXLP g1172(.A1 (n_31), .B1 (n_5), .B2 (a[15]), .O (n_34));
21    AO12RLXLP g1173(.A1 (n_31), .B1 (n_5), .B2 (a[14]), .O (n_33));
22    AO12RLXLP g1174(.A1 (n_31), .B1 (n_5), .B2 (a[16]), .O (n_32));
23    AOI112RLXLP g1175(.A1 (n_30), .B1 (n_6), .C1 (n_29), .C2 (n_3), .O
24        (n_31));
25    QDFCLRBRLX1 \a_reg[12] (.CK (clk), .D (n_27), .LD (n_7), .RB (n_0),
26        .Q (a[12]));
27    NR2RLXLP g1177(.I1 (n_3), .I2 (n_29), .O (n_30));
28    FA1RLX1 g1178(.A (a[15]), .B (m[6]), .CI (n_26), .S (n_28), .CO
29        (n_29));
30    QDFCLRBRLX1 \a_reg[11] (.CK (clk), .D (n_25), .LD (n_7), .RB (n_0),
31        .Q (a[11]));
32    FA1RLX1 g1180(.A (a[14]), .B (m[5]), .CI (n_24), .S (n_27), .CO
33        (n_26));
34    QDFCLRBRLX1 \a_reg[10] (.CK (clk), .D (n_23), .LD (n_7), .RB (n_0),
35        .Q (a[10]));
36    FA1RLX1 g1182(.A (a[13]), .B (m[4]), .CI (n_22), .S (n_25), .CO
37        (n_24));
38    QDFCLRBRLX1 \a_reg[9] (.CK (clk), .D (n_20), .LD (n_7), .RB (n_0), .Q
39        (a[9]));
40    FA1RLX1 g1184(.A (a[12]), .B (m[3]), .CI (n_19), .S (n_23), .CO
41        (n_22));
42    QDFFRLX1 \a_reg[8] (.CK (clk), .D (n_21), .Q (a[8]));
43    AO222RLXLP g1186(.A1 (n_2), .A2 (q[7]), .B1 (n_17), .B2 (n_7), .C1
44        (n_4), .C2 (a[8]), .O (n_21));
45    FA1RLX1 g1187(.A (a[11]), .B (m[2]), .CI (n_16), .S (n_20), .CO
46        (n_19));
47    QDFFRLX1 \a_reg[7] (.CK (clk), .D (n_18), .Q (a[7]));
48    QDFFRLX1 \a_reg[1] (.CK (clk), .D (n_10), .Q (a[1]));
49    QDFFRLX1 \a_reg[2] (.CK (clk), .D (n_15), .Q (a[2]));
50    QDFFRLX1 \a_reg[3] (.CK (clk), .D (n_14), .Q (a[3]));
51    QDFFRLX1 \a_reg[4] (.CK (clk), .D (n_13), .Q (a[4]));
52    QDFFRLX1 \a_reg[5] (.CK (clk), .D (n_12), .Q (a[5]));
53    QDFFRLX1 \a_reg[6] (.CK (clk), .D (n_11), .Q (a[6]));
54    QDFCLRBRLX1 \a_reg[0] (.CK (clk), .D (a[2]), .LD (n_7), .RB (n_0), .Q
55        (a[0]));
56    AO222RLXLP g1196(.A1 (n_2), .A2 (q[6]), .B1 (n_9), .B2 (n_7), .C1
57        (n_4), .C2 (a[7]), .O (n_18));
58    FA1RLX1 g1197(.A (a[10]), .B (n_8), .CI (m[1]), .S (n_17), .CO
59        (n_16));
60    AO222RLXLP g1198(.A1 (n_7), .A2 (a[4]), .B1 (n_4), .B2 (a[2]), .C1
61        (n_2), .C2 (q[1]), .O (n_15));
62    AO222RLXLP g1199(.A1 (n_7), .A2 (a[5]), .B1 (n_4), .B2 (a[3]), .C1
63        (n_2), .C2 (q[2]), .O (n_14));
64    AO222RLXLP g1200(.A1 (n_4), .A2 (a[4]), .B1 (n_7), .B2 (a[6]), .C1
65        (n_2), .C2 (q[3]), .O (n_13));
66    AO222RLXLP g1201(.A1 (n_7), .A2 (a[7]), .B1 (n_4), .B2 (a[5]), .C1
67        (n_2), .C2 (q[4]), .O (n_12));
68    AO222RLXLP g1202(.A1 (n_4), .A2 (a[6]), .B1 (n_7), .B2 (a[8]), .C1
69        (n_2), .C2 (q[5]), .O (n_11));
70    AO222RLXLP g1203(.A1 (n_7), .A2 (a[3]), .B1 (n_4), .B2 (a[1]), .C1
71        (n_2), .C2 (q[0]), .O (n_10));
72    HA1RLX1 g1204(.A (m[0]), .B (a[9]), .S (n_9), .C (n_8));
73    INVRLX1 g1205(.I (n_6), .O (n_7));
74    ND2RLXLP g1206(.I1 (n_1), .I2 (n_0), .O (n_6));
75    NR2RLXLP g1207(.I1 (reset), .I2 (n_1), .O (n_5));
```

```
  NR3RLX1 g1208(.I1 (init), .I2 (reset), .I3 (busy), .O (n_4));
  MAOI1RLXLP g1209(.A1 (m[7]), .A2 (a[16]), .B1 (a[16]), .B2 (m[7]), .O
       (n_3));
  AN2RLXLP g1210(.I1 (n_0), .I2 (init), .O (n_2));
  AN2B1RLXLP g1211(.I1 (busy), .B1 (init), .O (n_1));
  INVRLX1 g1213(.I (reset), .O (n_0));
endmodule

module int_reg(clk, init, reset, a_in, a_out);
  input clk, init, reset;
  input [7:0] a_in;
  output [7:0] a_out;
  wire clk, init, reset;
  wire [7:0] a_in;
  wire [7:0] a_out;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  QDFFRLX1 \a_out_reg[0] (.CK (clk), .D (n_1), .Q (a_out[0]));
  QDFFRLX1 \a_out_reg[2] (.CK (clk), .D (n_0), .Q (a_out[2]));
  QDFFRLX1 \a_out_reg[3] (.CK (clk), .D (n_7), .Q (a_out[3]));
  QDFFRLX1 \a_out_reg[4] (.CK (clk), .D (n_6), .Q (a_out[4]));
  QDFFRLX1 \a_out_reg[1] (.CK (clk), .D (n_4), .Q (a_out[1]));
  QDFFRLX1 \a_out_reg[5] (.CK (clk), .D (n_5), .Q (a_out[5]));
  QDFFRLX1 \a_out_reg[6] (.CK (clk), .D (n_3), .Q (a_out[6]));
  QDFFRLX1 \a_out_reg[7] (.CK (clk), .D (n_2), .Q (a_out[7]));
  MUXB2RLXLP g26(.S (init), .A (a_out[3]), .B (a_in[3]), .EB (reset),
       .O (n_7));
  MUXB2RLXLP g27(.S (init), .A (a_out[4]), .B (a_in[4]), .EB (reset),
       .O (n_6));
  MUXB2RLXLP g28(.S (init), .A (a_out[5]), .B (a_in[5]), .EB (reset),
       .O (n_5));
  MUXB2RLXLP g29(.S (init), .A (a_out[1]), .B (a_in[1]), .EB (reset),
       .O (n_4));
  MUXB2RLXLP g30(.S (init), .A (a_out[6]), .B (a_in[6]), .EB (reset),
       .O (n_3));
  MUXB2RLXLP g31(.S (init), .A (a_out[7]), .B (a_in[7]), .EB (reset),
       .O (n_2));
  MUXB2RLXLP g32(.S (init), .A (a_out[0]), .B (a_in[0]), .EB (reset),
       .O (n_1));
  MUXB2RLXLP g33(.S (init), .A (a_out[2]), .B (a_in[2]), .EB (reset),
       .O (n_0));
endmodule

module part_prod(x, a, m);
  input [2:0] x;
  input [7:0] a;
  output [7:0] m;
  wire [2:0] x;
  wire [7:0] a;
  wire [7:0] m;
  wire n_0, n_1, n_2, n_3, n_4, n_5, n_6, n_7;
  wire n_8, n_9, n_10, n_12, n_13, n_14, n_15, n_16;
  wire n_17, n_18, n_19, n_20, n_21, n_23, n_25, n_26;
  wire n_28, n_29, n_31, n_32, n_34;
  OAI122RLXLP g570(.A1 (n_19), .B1 (n_34), .B2 (n_12), .C1 (n_7), .C2
       (n_32), .O (m[7]));
  OAI122RLXLP g571(.A1 (n_16), .B1 (n_32), .B2 (n_12), .C1 (n_7), .C2
       (n_29), .O (m[6]));
```

```
133   MAOI1RLXLP g572(.A1 (n_31), .A2 (a[7]), .B1 (a[7]), .B2 (n_31), .O
134         (n_34));
135   OAI122RLXLP g573(.A1 (n_15), .B1 (n_29), .B2 (n_12), .C1 (n_7), .C2
136         (n_26), .O (m[5]));
137   MAOI1RLXLP g574(.A1 (n_28), .A2 (a[6]), .B1 (a[6]), .B2 (n_28), .O
138         (n_32));
139   AN2B1RLXLP g575(.I1 (n_28), .B1 (a[6]), .O (n_31));
140   OAI122RLXLP g576(.A1 (n_14), .B1 (n_26), .B2 (n_12), .C1 (n_7), .C2
141         (n_23), .O (m[4]));
142   MAOI1RLXLP g577(.A1 (n_25), .A2 (a[5]), .B1 (a[5]), .B2 (n_25), .O
143         (n_29));
144   AN2B1RLXLP g578(.I1 (n_25), .B1 (a[5]), .O (n_28));
145   OAI122RLXLP g579(.A1 (n_18), .B1 (n_23), .B2 (n_12), .C1 (n_7), .C2
146         (n_13), .O (m[3]));
147   MAOI1RLXLP g580(.A1 (n_21), .A2 (a[4]), .B1 (a[4]), .B2 (n_21), .O
148         (n_26));
149   AN2B1RLXLP g581(.I1 (n_21), .B1 (a[4]), .O (n_25));
150   OAI122RLXLP g582(.A1 (n_17), .B1 (n_13), .B2 (n_12), .C1 (n_7), .C2
151         (n_3), .O (m[2]));
152   MAOI1RLXLP g583(.A1 (n_8), .A2 (a[3]), .B1 (a[3]), .B2 (n_8), .O
153         (n_23));
154   OAI122RLXLP g584(.A1 (n_20), .B1 (n_9), .B2 (n_0), .C1 (n_3), .C2
155         (n_12), .O (m[1]));
156   AN2B1RLXLP g585(.I1 (n_8), .B1 (a[3]), .O (n_21));
157   OAI12RLXLP g586(.A1 (a[0]), .B1 (n_6), .B2 (n_5), .O (n_20));
158   AOI22RLXLP g587(.A1 (n_10), .A2 (a[7]), .B1 (n_5), .B2 (a[6]), .O
159         (n_19));
160   AOI22RLXLP g588(.A1 (n_10), .A2 (a[3]), .B1 (n_5), .B2 (a[2]), .O
161         (n_18));
162   AOI22RLXLP g589(.A1 (n_10), .A2 (a[2]), .B1 (n_5), .B2 (a[1]), .O
163         (n_17));
164   AOI22RLXLP g590(.A1 (n_10), .A2 (a[6]), .B1 (n_5), .B2 (a[5]), .O
165         (n_16));
166   AOI22RLXLP g591(.A1 (n_10), .A2 (a[5]), .B1 (n_5), .B2 (a[4]), .O
167         (n_15));
168   AOI22RLXLP g592(.A1 (n_10), .A2 (a[4]), .B1 (n_5), .B2 (a[3]), .O
169         (n_14));
170   MAOI1RLXLP g593(.A1 (n_2), .A2 (a[2]), .B1 (a[2]), .B2 (n_2), .O
171         (n_13));
172   ND2RLXLP g594(.I1 (n_4), .I2 (x[2]), .O (n_12));
173   AN2RLXLP g595(.I1 (n_4), .I2 (a[0]), .O (m[0]));
174   INVRLX1 g596(.I (n_9), .O (n_10));
175   OR2B1RLXLP g597(.I1 (x[2]), .B1 (n_4), .O (n_9));
176   AN2B1RLXLP g598(.I1 (n_2), .B1 (a[2]), .O (n_8));
177   INVRLX1 g599(.I (n_6), .O (n_7));
178   AN3B2RLXLP g600(.I1 (x[2]), .B1 (x[0]), .B2 (x[1]), .O (n_6));
179   NR2RLXLP g601(.I1 (x[2]), .I2 (n_1), .O (n_5));
180   XOR2RLX1 g602(.I1 (x[0]), .I2 (x[1]), .O (n_4));
181   MXL2RLXLP g603(.S (a[0]), .A (a[1]), .B (n_0), .OB (n_3));
182   NR2RLXLP g604(.I1 (a[0]), .I2 (a[1]), .O (n_2));
183   ND2RLXLP g605(.I1 (x[0]), .I2 (x[1]), .O (n_1));
184   INVRLX1 g606(.I (a[1]), .O (n_0));
185 endmodule
186
187 module seq_count(clk, reset, init, busy);
188   input clk, reset, init;
189   output busy;
```

```
190   wire clk, reset, init;
191   wire busy;
192   wire \a[0] , \a[1] , \a[2] , n_0, n_1, n_2, n_4, n_5;
193   wire n_8, n_9, n_14, n_19;
194   OR3RLX1 g108(.I1 (\a[1] ), .I2 (\a[2] ), .I3 (\a[0] ), .O (busy));
195   INVRLX1 g109(.I (\a[0] ), .O (n_9));
196   QDFFRLX1 \a_reg[2] (.CK (clk), .D (n_8), .Q (\a[2] ));
197   MXL2RLXLP g222(.S (n_2), .A (reset), .B (n_14), .OB (n_8));
198   QDFCRBRLX1 \a_reg[1] (.CK (clk), .D (n_5), .RB (n_2), .Q (\a[1] ));
199   QDFFRLX1 \a_reg[0] (.CK (clk), .D (n_19), .Q (\a[0] ));
200   MOAI1RLXLP g227(.A1 (n_0), .A2 (\a[1] ), .B1 (n_0), .B2 (\a[1] ), .O
201         (n_5));
202   MOAI1RLXLP g228(.A1 (n_1), .A2 (\a[2] ), .B1 (n_1), .B2 (\a[2] ), .O
203         (n_4));
204   NR2RLXLP g230(.I1 (init), .I2 (reset), .O (n_2));
205   OR2RLXLP g231(.I1 (\a[1] ), .I2 (\a[0] ), .O (n_1));
206   ND2RLXLP g232(.I1 (busy), .I2 (n_9), .O (n_0));
207   ND2RLX1 g2(.I1 (busy), .I2 (n_4), .O (n_14));
208   AN3RLX1 g237(.I1 (n_2), .I2 (busy), .I3 (n_9), .O (n_19));
209 endmodule
210
211 module mult_8bit(a, b, clk, reset, init, y, busy);
212   input [7:0] a, b;
213   input clk, reset, init;
214   output [15:0] y;
215   output busy;
216   wire [7:0] a, b;
217   wire clk, reset, init;
218   wire [15:0] y;
219   wire busy;
220   wire \c[0] , \c[1] , \c[2] , \c[3] , \c[4] , \c[5] , \c[6] , \c[7] ;
221   wire \m[0] , \m[1] , \m[2] , \m[3] , \m[4] , \m[5] , \m[6] , \m[7] ;
222   wire \z[0] ;
223   acc ac(.q (b), .m ({\m[7] , \m[6] , \m[5] , \m[4] , \m[3] , \m[2] ,
224         \m[1] , \m[0] }), .clk (clk), .init (init), .busy (busy), .reset
225         (reset), .a ({y, \z[0] }));
226   int_reg ir(.clk (clk), .init (init), .reset (reset), .a_in (a),
227         .a_out ({\c[7] , \c[6] , \c[5] , \c[4] , \c[3] , \c[2] , \c[1] ,
228         \c[0] }));
229   part_prod pp(.x ({y[1:0], \z[0] }), .a ({\c[7] , \c[6] , \c[5] ,
230         \c[4] , \c[3] , \c[2] , \c[1] , \c[0] }), .m ({\m[7] , \m[6] ,
231         \m[5] , \m[4] , \m[3] , \m[2] , \m[1] , \m[0] }));
232   seq_count sc(.clk (clk), .reset (reset), .init (init), .busy (busy));
233 endmodule
```

3.5.2.2 Summary report

```
============================================================  Generated
by:           Encounter(R) RTL Compiler v08.10-s116_1
  Generated on:           Sep 22 2018  07:27:03 PM
  Module:                 mult_8bit
  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================
```

```
      Timing
      ------
        Tracing clock networks.
        Levelizing the circuit.
        Computing delays.
        Computing arrivals and requireds.
Warning : Possible timing problems have been detected in this design.
[TIM-11]
        : The design is 'mult_8bit'.
        : Use 'report timing -lint' for more information.

No paths found.


      Area
      ----

 Instance  Cells  Cell Area  Net Area   Wireload
-----------------------------------------------------
mult_8bit    111       1282         0      enG5K (S)

 (S) = wireload was automatically selected

      Design Rule Check
      -----------------
        Initializing DRC engine.

Max_transition design rule: no violations.

Max_capacitance design rule: no violations.

Max_fanout design rule: no violations.
```

3.5.2.3 Power report

```
=============================================================  Generated
by:           Encounter(R) RTL Compiler v08.10-s116_1
  Generated on:           Sep 22 2018  07:27:04 PM
  Module:                 mult_8bit
  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

                 Leakage   Dynamic    Total
 Instance Cells Power(nW) Power(nW) Power(nW)
-----------------------------------------------
mult_8bit   111     3.218 21539.089 21542.307
  ac         45     1.835 10342.762 10344.597
  ir         16     0.551  4064.339  4064.890
  pp         37     0.522  3647.662  3648.184
```

```
  sc          13      0.311   1806.326   1806.637
```

3.5.2.4 Area utilization report

```
================================================================  Generated
by:           Encounter(R) RTL Compiler v08.10-s116_1
  Generated on:           Sep 22 2018  07:27:03 PM
  Module:                 mult_8bit
  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
================================================================

 Instance  Cells  Cell Area  Net Area   Wireload
-----------------------------------------------------
mult_8bit    111       1282         0      enG5K (S)
  ac          45        705         0      enG5K (S)
  pp          37        249         0      enG5K (S)
  ir          16        216         0      enG5K (S)
  sc          13        112         0      enG5K (S)

 (S) = wireload was automatically selected
```

3.5.2.5 Gates utilization

```
================================================================  Generated
by:           Encounter(R) RTL Compiler v08.10-s116_1
  Generated on:           Sep 22 2018  07:27:04 PM
  Module:                 mult_8bit
  Technology library:     fsd0k_a_generic_core_1d0vtc 2007Q2v1.3
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
================================================================

    Gate     Instances    Area               Library
-------------------------------------------------------------
AN2B1RLXLP           6    30.000    fsd0k_a_generic_core_1d0vtc
AN2RLXLP             2    10.000    fsd0k_a_generic_core_1d0vtc
AN3B2RLXLP           1     7.000    fsd0k_a_generic_core_1d0vtc
AN3RLX1              1     7.000    fsd0k_a_generic_core_1d0vtc
AO12RLXLP            3    21.000    fsd0k_a_generic_core_1d0vtc
AO222RLXLP           8    96.000    fsd0k_a_generic_core_1d0vtc
AOI112RLXLP          1     7.000    fsd0k_a_generic_core_1d0vtc
AOI22RLXLP           6    42.000    fsd0k_a_generic_core_1d0vtc
FA1RLX1              6   180.000    fsd0k_a_generic_core_1d0vtc
HA1RLX1              1    15.000    fsd0k_a_generic_core_1d0vtc
INVRLX1              6    18.000    fsd0k_a_generic_core_1d0vtc
MAOI1RLXLP           7    63.000    fsd0k_a_generic_core_1d0vtc
MOAI1RLXLP           2    16.000    fsd0k_a_generic_core_1d0vtc
```

MUXB2RLXLP	8	80.000	fsd0k_a_generic_core_1d0vtc
MXL2RLXLP	2	14.000	fsd0k_a_generic_core_1d0vtc
ND2RLX1	1	4.000	fsd0k_a_generic_core_1d0vtc
ND2RLXLP	4	16.000	fsd0k_a_generic_core_1d0vtc
NR2RLXLP	5	20.000	fsd0k_a_generic_core_1d0vtc
NR3RLX1	1	6.000	fsd0k_a_generic_core_1d0vtc
OAI122RLXLP	7	63.000	fsd0k_a_generic_core_1d0vtc
OAI12RLXLP	1	6.000	fsd0k_a_generic_core_1d0vtc
OR2B1RLXLP	1	5.000	fsd0k_a_generic_core_1d0vtc
OR2RLXLP	1	5.000	fsd0k_a_generic_core_1d0vtc
OR3RLX1	1	7.000	fsd0k_a_generic_core_1d0vtc
QDFCLRBRLX1	6	156.000	fsd0k_a_generic_core_1d0vtc
QDFCRBRLX1	1	21.000	fsd0k_a_generic_core_1d0vtc
QDFFRLX1	21	357.000	fsd0k_a_generic_core_1d0vtc
XOR2RLX1	1	10.000	fsd0k_a_generic_core_1d0vtc
total	111	1282.000	

Type	Instances	Area	Area %
sequential	28	534.000	41.7
inverter	6	18.000	1.4
logic	77	730.000	56.9
total	111	1282.000	100.0