

# LAB 1 Report: Design of 8-bit Booth Multiplier

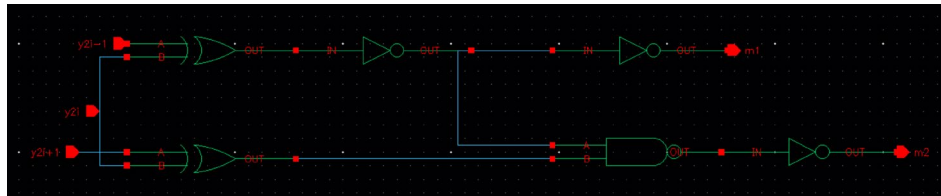
Yihao Wang

USCID: 7410178057

## 1. Design Description

This design is based on Modified Booth Coding Multiplication Algorithm. The circuit mainly consists of four parts: input register array, partial products generator, addition unit and output register array.

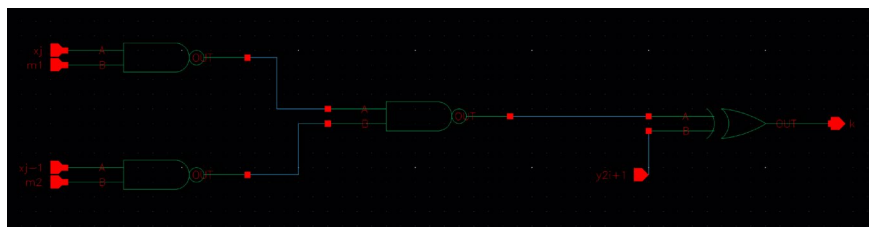
For partial product generator design, you should decide what should be selected among 0X, 1X and 2X based on  $Y_{2i-1}$ ,  $Y_{2i}$ ,  $Y_{2i+1}$ . The circuit(Booth Encoder) shown below generates  $m1$  and  $m2$  signal to distinguish 0X, 1X and 2X three cases.



The truth table is as follows:

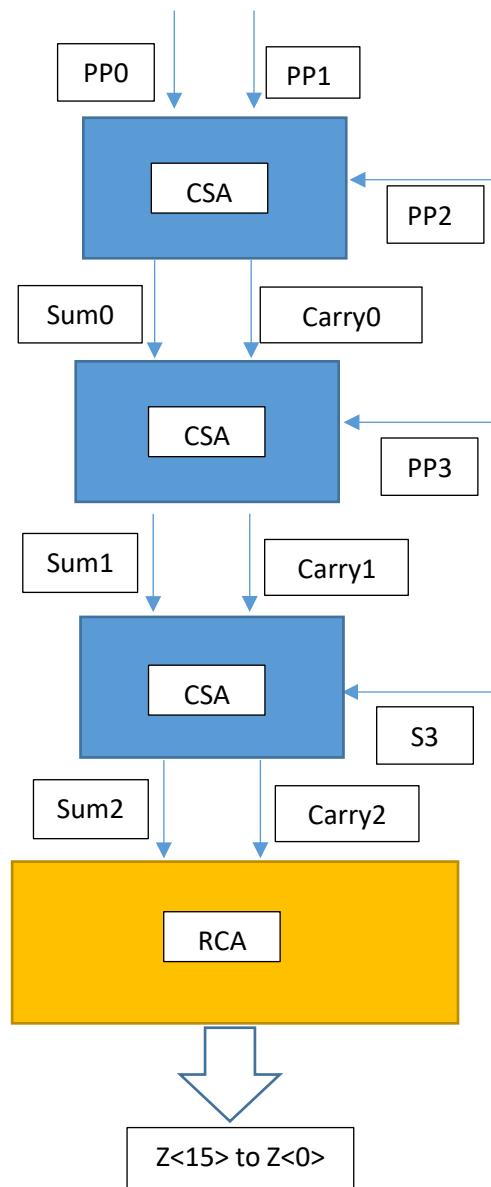
$Y_{2i+1}$	$Y_{2i}$	$Y_{2i-1}$	$m1$	$m2$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	0

For each  $K_{i,j}$ ,  $X_j$ ,  $X_{j-1}$  and 0 will be chosen by MUX based on  $m1$  and  $m2$ , then if the  $\text{sign}(Y_{2i+1})$  is 0, we just output the value chosen by mux. If sign is 1, we will invert the output. So, we should use a simplified 3 to 1 MUX and a controllable inverter(XOR gate) to implement this. The circuit(Booth Decoder) is as follows:



Three NAND gates forms a 3 to 1 MUX with a 0 input. If both m1 and m2 are 0, 0 will be selected and output. Therefore, for each row of partial product, we need one encoder circuit and 9 decoder circuits to generate  $K_{i,0}$  to  $K_{i,8}$

Since we get all four partial products, we need to use three rows of CSA and one row of RCA to add all partial products



## 2. Functionality Test

My Python scripts are shown as below( random\_generator function is used to generate a vector file with random testing patterns):

```
1. """
2.     This program is used for generating some random binary input patterns
3.     to test the functionality of circuits
4.     By Yihao Wang
5. """
6.
7. import os
8. import random
9. ## conversion from binary to decimal, returns a string(this function is not used by random generator)
10. def bin_to_dec (num):
11.     the_num = num
12.     count = 0
13.     dec = 0
14.     while the_num != 0 :
15.         ## note that **
16.         dec += (the_num % 10) * (2 ** count)
17.
18.         ## division of python returns a float number
19.         the_num = int(the_num / 10)
20.         count = count + 1
21.     return str(dec)
22.
23. ## conversion from decimal to binary num, returns a string
24. def dec_to_bin (num,width):
25.     the_num = num
26.     bin_str = ''
27.     if (num < 0):
28.         print('the parameter must be [0,255]!')
29.     else:
30.         for num in range(width):
31.             bin_str = str(the_num % 2) + bin_str
32.             the_num = get_quotient(the_num, 2)
33.         return bin_str
34.
35. ## gets the remainder
36. def get_quotient (dividend, divider):
37.     the_dividend = dividend
38.     the_divider = divider
39.     quotient = 0
40.     while the_dividend >= the_divider :
41.         the_dividend -= the_divider
42.         quotient += 1
43.     return quotient
44.
45. ## Generates random input patterns based on user's configuration
46. ## Parameters:
47. ##     width: the width of binary input patterns (default value: 8)
48. ##     number: the number of input patterns you want to generate (default value: 30)
49. ##     clk: clk period (default value: 10)
50. ##     unit: time unit (default: ns)
51. ##     slope: the slope of input signal (default value: 0.01)
52. ##     vih: voltage of logic high (default value: 1.8V)
53. ##     vil: voltage of logic low (default value: 0V)
54. def random_generator(width = 8, number = 30, clk = 10, delay = 2, unit = 'ns', slope = 0.01, vih = 1.8, vil = 0):
```

```

55.     time = delay
56.     ## gets user's work directory
57.     os.chdir(input('Please input your work directory: '))
58.     with open(input("Please enter output file name: "), "w") as the_file:
59.
60.         count = width - 1
61.         print('radix', file = the_file, end = ' ')
62.         for i in range(width * 2) :
63.             print('1', file = the_file, end = ' ')
64.
65.         print('\nio', file = the_file, end = ' ')
66.         for i in range(width * 2) :
67.             print('i', file = the_file, end = ' ')
68.         print('\nvname', file = the_file, end = ' ')
69.         for i in range(width) :
70.             print('X' + str(count), file = the_file, end = ' ')
71.             count -= 1
72.         count = width - 1
73.         for i in range(width) :
74.             print('Y' + str(count), file = the_file, end = ' ')
75.             count -= 1
76.
77.         print('\ntunit ' + str(unit), file = the_file)
78.         print('slope ' + str(slope), file = the_file)
79.         print('vih ' + str(vih), file = the_file)
80.         print('vil ' + str(vil), file = the_file)
81.
82.         for num in range(number) :
83.             print(time, end = ' ', file = the_file)
84.             print(dec_to_bin(int(random.random() * (2 ** width)),width), end = ' ', file = the_file)
85.             print(dec_to_bin(int(random.random() * (2 ** width)),width), end = '\n', file = the_file)
86.             time += clk
87.
88.         print('Successfully Generated!')

```

The input testing patterns table is as follows:

Op1 (dec)	Op2 (dec)	Result (dec)	Op1 (bin)	Op2 (bin)	Result (bin)
-90	14	-1260	10100110	00001110	1111101100010100
-69	-107	7383	10111011	10010101	0001110011010111
-92	-118	10856	10100100	10001010	0010101001101000
-20	45	-900	11101100	00101101	1111110001111100
-35	-31	1085	11011101	11100001	0000010000111101
1	39	39	00000001	00100111	0000000000100111
32	-37	-1184	00100000	11011011	1111101101100000
4	59	236	00000010	00111011	0000000011101100
98	-10	-980	01100010	11110110	1111110000101100
115	-2	-230	01110011	11111110	1111111100011010

The functionality test of schematic design is as follows (clock cycle is 3ns):



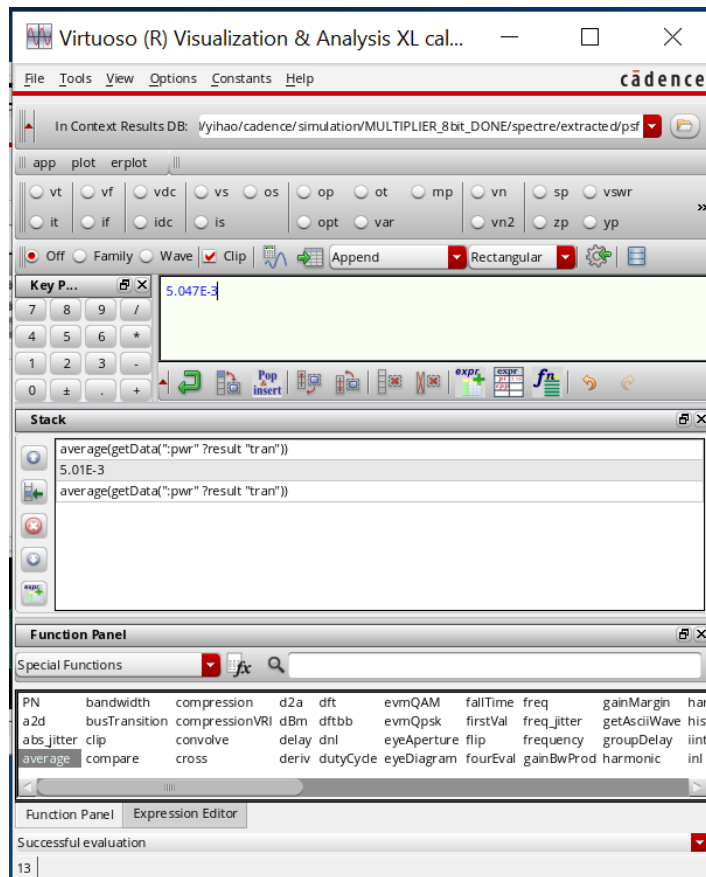
### 3. Measurement of Parameters

Minimum clock is 333MHz (clock cycle is 3ns)

Width is 106.8um

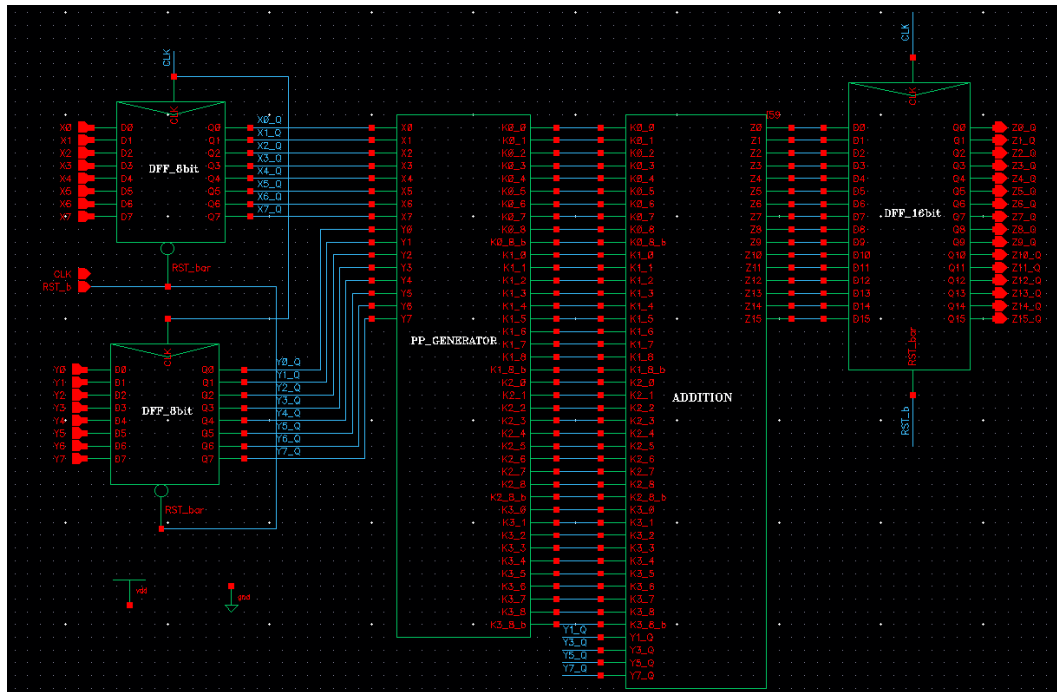
Height is 85.5um

Average Power Consumption is 5.047E-3 W

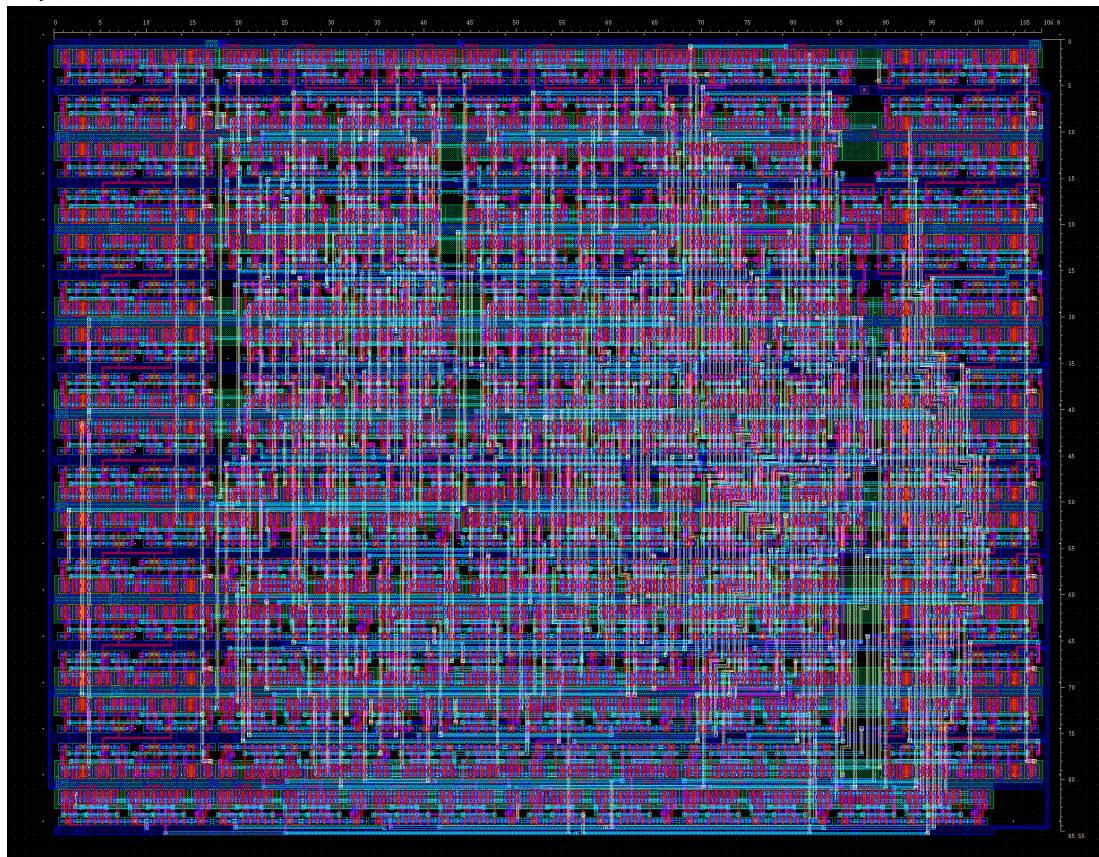


#### 4. Screenshots

Schematic:



Layout:





### Functionality test of schematic:

The screenshot displays the 'Functionality Response' of a 10-bit DAC. The top panel shows a green bar graph representing the output voltage for each input code from 0 to 1023. The bottom panel shows a table of the DAC's transfer characteristics, listing the input code, the corresponding output voltage (Vout), and the error (Vout - Ideal). The output voltage ranges from approximately 0.000V to 2.048V, and the error is consistently within  $\pm 0.001$  V.

Code	Vout	Vout - Ideal
0	0.000	0.000
1	0.001	0.000
2	0.002	0.000
3	0.003	0.000
4	0.004	0.000
5	0.005	0.000
6	0.006	0.000
7	0.007	0.000
8	0.008	0.000
9	0.009	0.000
10	0.010	0.000
11	0.011	0.000
12	0.012	0.000
13	0.013	0.000
14	0.014	0.000
15	0.015	0.000
16	0.016	0.000
17	0.017	0.000
18	0.018	0.000
19	0.019	0.000
20	0.020	0.000
21	0.021	0.000
22	0.022	0.000
23	0.023	0.000
24	0.024	0.000
25	0.025	0.000
26	0.026	0.000
27	0.027	0.000
28	0.028	0.000
29	0.029	0.000
30	0.030	0.000
31	0.031	0.000
32	0.032	0.000
33	0.033	0.000
34	0.034	0.000
35	0.035	0.000
36	0.036	0.000
37	0.037	0.000
38	0.038	0.000
39	0.039	0.000
40	0.040	0.000
41	0.041	0.000
42	0.042	0.000
43	0.043	0.000
44	0.044	0.000
45	0.045	0.000
46	0.046	0.000
47	0.047	0.000
48	0.048	0.000
49	0.049	0.000
50	0.050	0.000
51	0.051	0.000
52	0.052	0.000
53	0.053	0.000
54	0.054	0.000
55	0.055	0.000
56	0.056	0.000
57	0.057	0.000
58	0.058	0.000
59	0.059	0.000
60	0.060	0.000
61	0.061	0.000
62	0.062	0.000
63	0.063	0.000
64	0.064	0.000
65	0.065	0.000
66	0.066	0.000
67	0.067	0.000
68	0.068	0.000
69	0.069	0.000
70	0.070	0.000
71	0.071	0.000
72	0.072	0.000
73	0.073	0.000
74	0.074	0.000
75	0.075	0.000
76	0.076	0.000
77	0.077	0.000
78	0.078	0.000
79	0.079	0.000
80	0.080	0.000
81	0.081	0.000
82	0.082	0.000
83	0.083	0.000
84	0.084	0.000
85	0.085	0.000
86	0.086	0.000
87	0.087	0.000
88	0.088	0.000
89	0.089	0.000
90	0.090	0.000
91	0.091	0.000
92	0.092	0.000
93	0.093	0.000
94	0.094	0.000
95	0.095	0.000
96	0.096	0.000
97	0.097	0.000
98	0.098	0.000
99	0.099	0.000
100	0.100	0.000
101	0.101	0.000
102	0.102	0.000
103	0.103	0.000
104	0.104	0.000
105	0.105	0.000
106	0.106	0.000
107	0.107	0.000
108	0.108	0.000
109	0.109	0.000
110	0.110	0.000
111	0.111	0.000
112	0.112	0.000
113	0.113	0.000
114	0.114	0.000
115	0.115	0.000
116	0.116	0.000
117	0.117	0.000</

Functionality test of layout:

The screenshot displays a Genomic Browser interface. At the top, a coordinate scale ranges from 1 to 100,000,000. Below this, several tracks are visible, including a track labeled 'Transcript Browser' on the left. The main track shows a genomic region with various annotations, including gene models and transcript structures. The interface includes a search bar at the top right and a coordinate scale at the bottom.