LAB 1 Report: Design of 8-bit Booth Multiplier

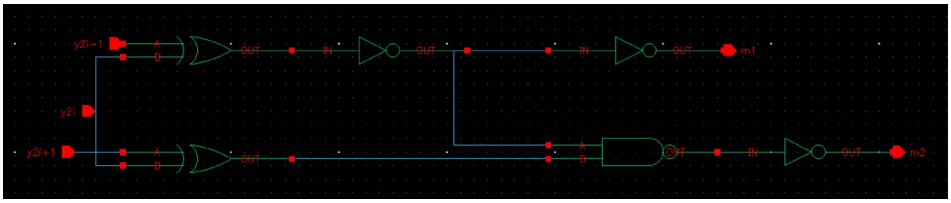
Yihao Wang

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.

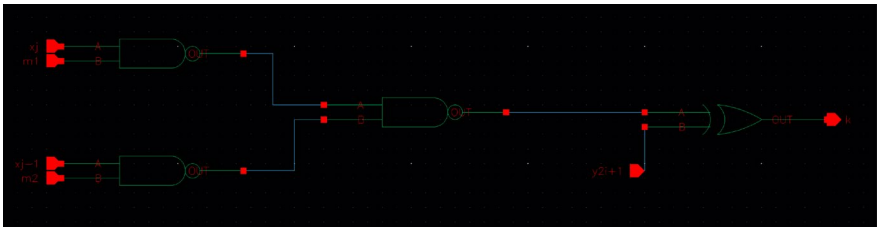
For partial product generator design, you should decide what should be selected among 0X, 1X and 2X based on Y2i-1, Y2i, Y2i+1. The circuit(Booth Encoder) shown below generates m1 and m2 signal to distinguish 0X, 1X and 2X three cases.



The truth table is as follows:

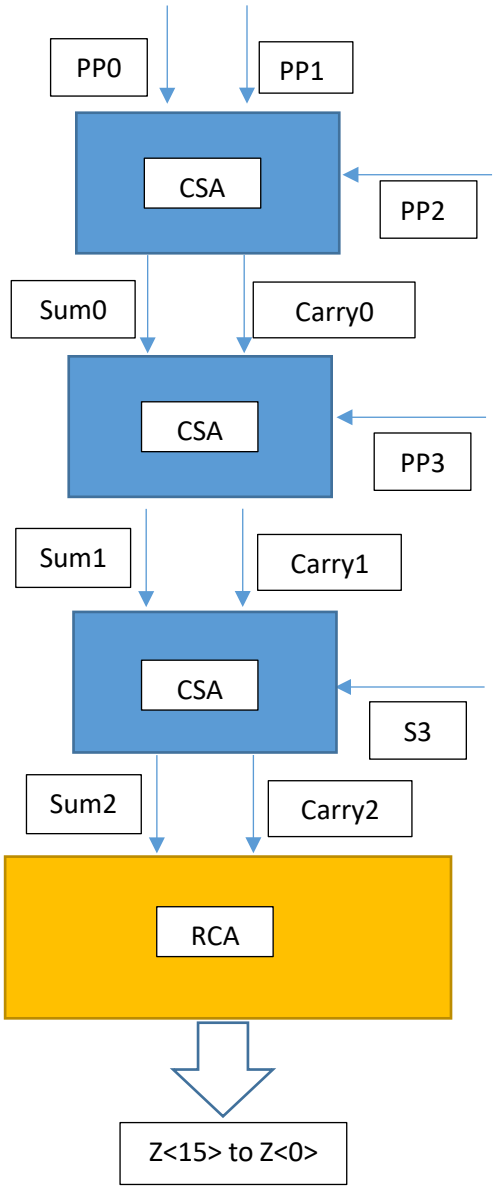
Y2i+1	Y2i	Y2i-1	m1	m2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	0

For each Ki,j, Xj, Xj-1 and 0 will be chosen by MUX based on m1 and m2, then if the sign(Y2i+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 Ki,0 to Ki,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):

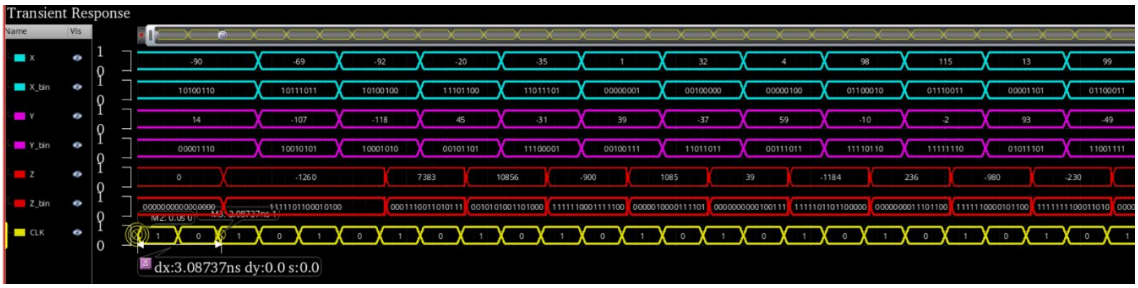
```
"""
    This program is used for generating some random binary input patterns
    to test the functionality of circuits
    By Yihao Wang
"""

import os
import random
## conversion from binary to decimal, returns a string(this function is not used by random generator)
def bin_to_dec (num):
    the_num = num
    count = 0
    dec = 0
    while the_num != 0 :
        ## note that **
        dec += (the_num %  10) * (2 ** count)

        ## division of python returns a float number
        the_num = int(the_num / 10)
        count = count + 1
    return str(dec)

## conversion from decimal to binary num, returns a string
def dec_to_bin (num,width):
    the_num = num
    bin_str = ''
    if (num < 0):
            print('the parameter must be [0,255]!')
    else:
        for num in range(width):
            bin_str = str(the_num % 2) + bin_str
            the_num = get_quotient(the_num, 2)
        return bin_str

## gets the remainder
def get_quotient (dividend, divider):
    the_dividend = dividend
    the_divider = divider
    quotient = 0
    while the_dividend >= the_divider :
        the_dividend -= the_divider
        quotient += 1
    return quotient

## Generates random input patterns based on user's configuration
## Parameters:
##      width: the width of binary input patterns (defalt value: 8)
##      number: the number of input patterns you want to generate (default value: 30)
##      clk: clk period (default value: 10)
##      unit: time unit (default: ns)
##      slope: the slope of input signal (default value: 0.01)
##      vih: voltage of logic high (default value: 1.8V)
##      vil: voltage of logic low (default value: 0V)
def random_generator(width = 8, number = 30, clk = 10, delay  = 2, unit = 'ns', slope = 0.01, vih = 1.8, vil = 0):
```

```
    time = delay
    ## gets user's work directory
    os.chdir(input('Please input your work directory: '))
    with open(input("Please enter output file name: "), "w") as the_file:

        count = width - 1
        print('radix', file = the_file, end = ' ')
        for i in range(width * 2) :
            print('1', file = the_file, end = ' ')

        print('\nio', file = the_file, end = ' ')
        for i in range(width *2) :
            print('i', file = the_file, end = ' ')
        print('\nvname', file = the_file, end = ' ')
        for i in range(width) :
            print('X' + str(count), file = the_file, end = ' ')
            count -= 1
        count = width - 1
        for i in range(width) :
            print('Y' + str(count), file = the_file, end = ' ')
            count -= 1

        print('\ntunit ' + str(unit), file = the_file)
        print('slope ' + str(slope), file = the_file)
        print('vih ' + str(vih), file = the_file)
        print('vil ' + str(vil), file = the_file)

        for num in range(number) :
            print(time, end = ' ', file = the_file)
            print(dec_to_bin(int(random.random() * (2 ** width)),width), end = ' ', fil
e = the_file)
            print(dec_to_bin(int(random.random() * (2 ** width)),width), end = '\n', fi
le = the_file)
            time += clk

        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):



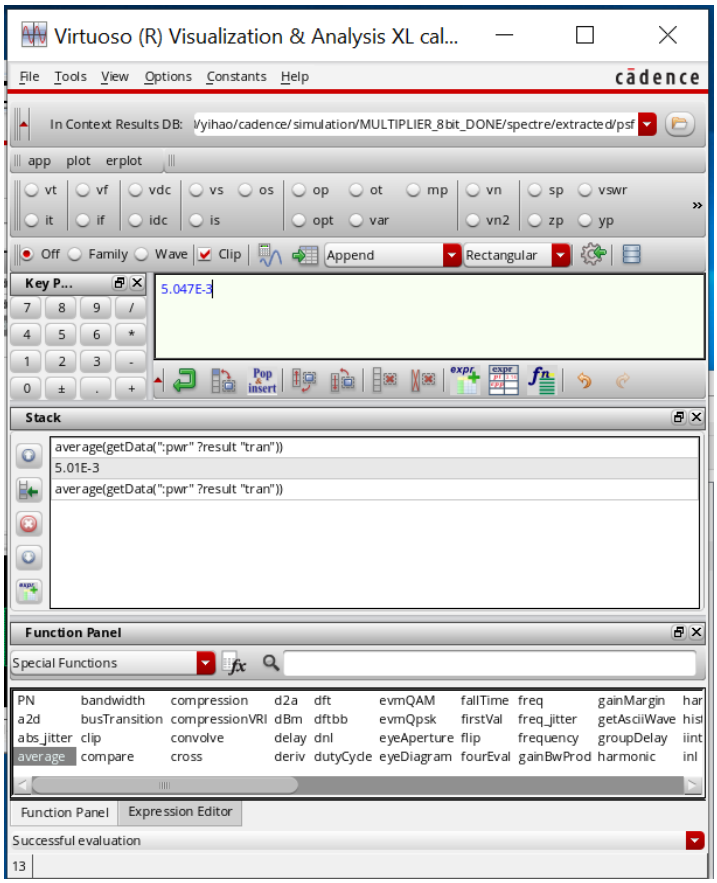
3. Measurement of Parameters

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

Width is 106.8um

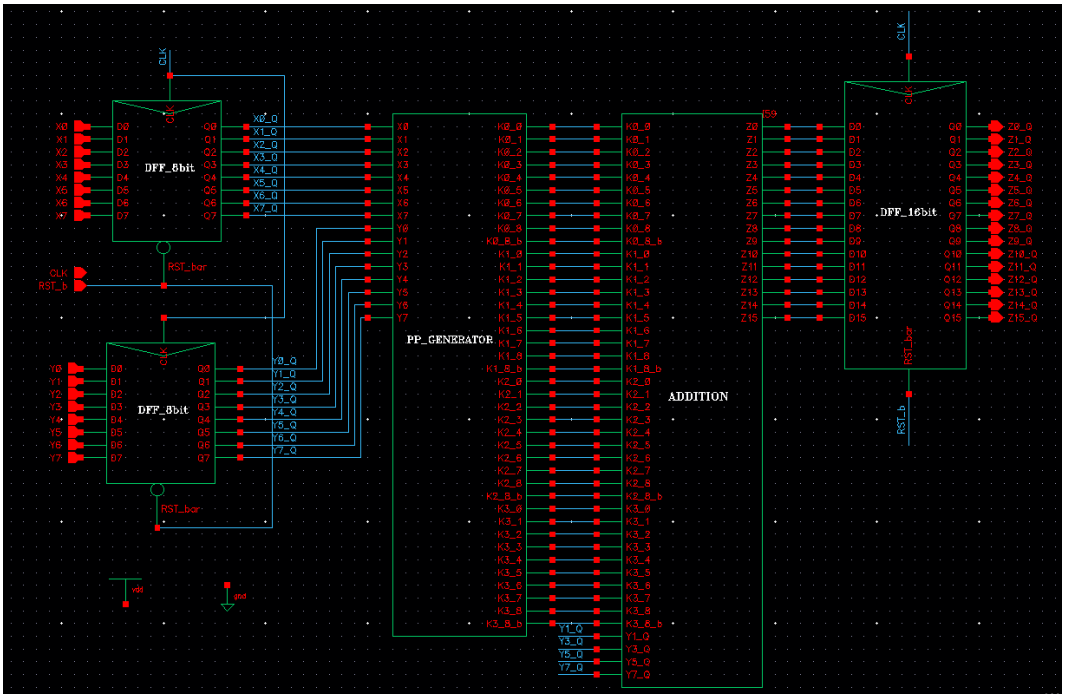
Height is 85.5um

Average Power Consumption is 5.047E-3 W

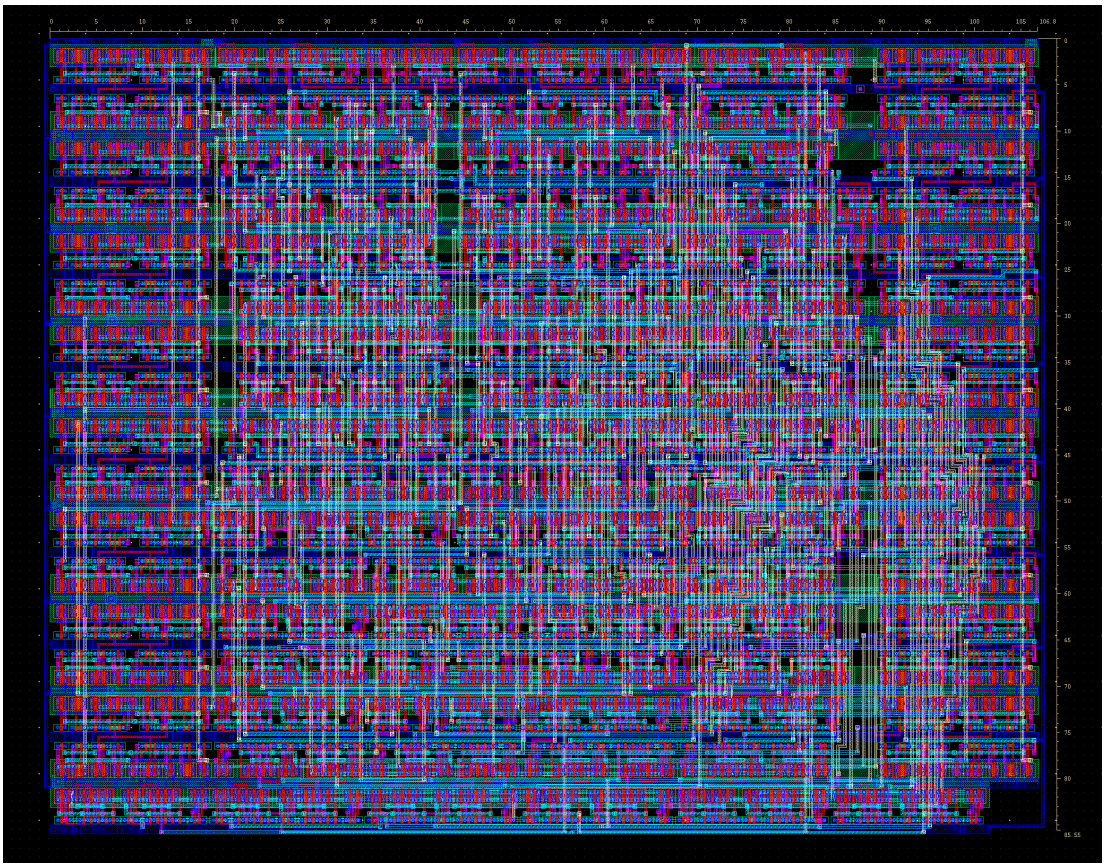


4. Screenshots

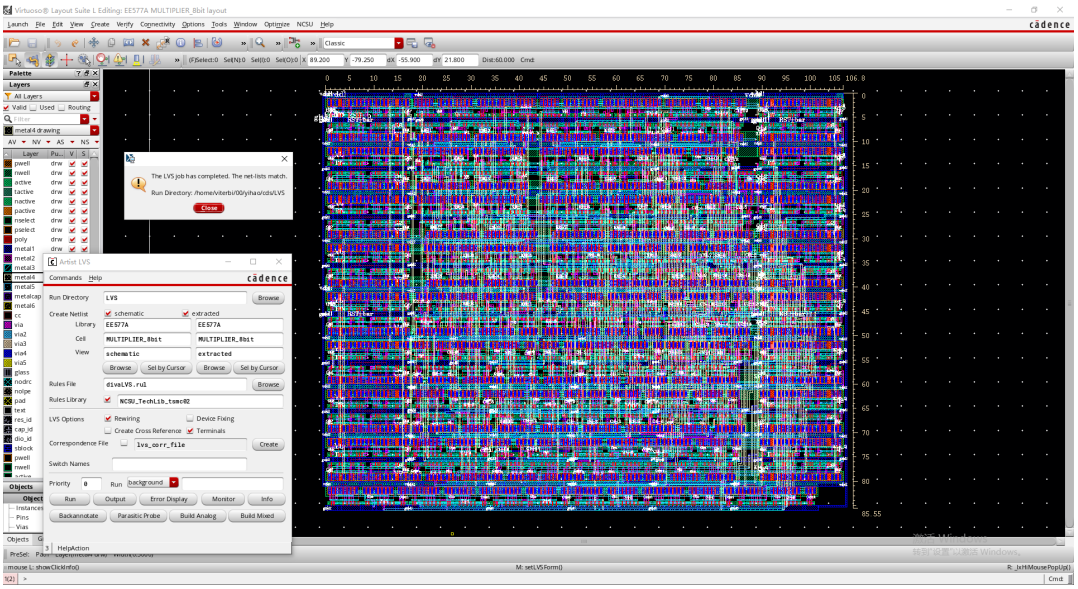
Schematic:



Layout:



LVS:



Functionality test of schematic:

Functionality test of layout: