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

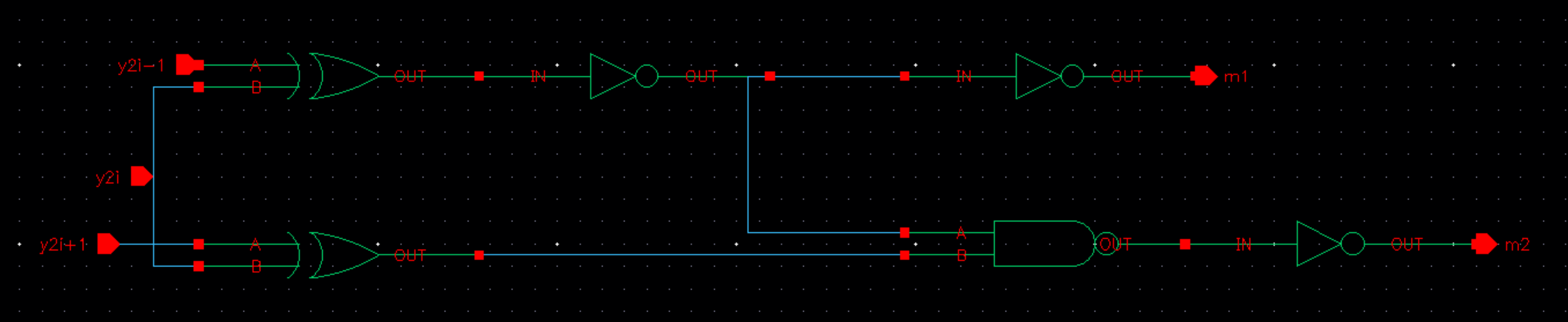
**Yihao Wang**

**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.

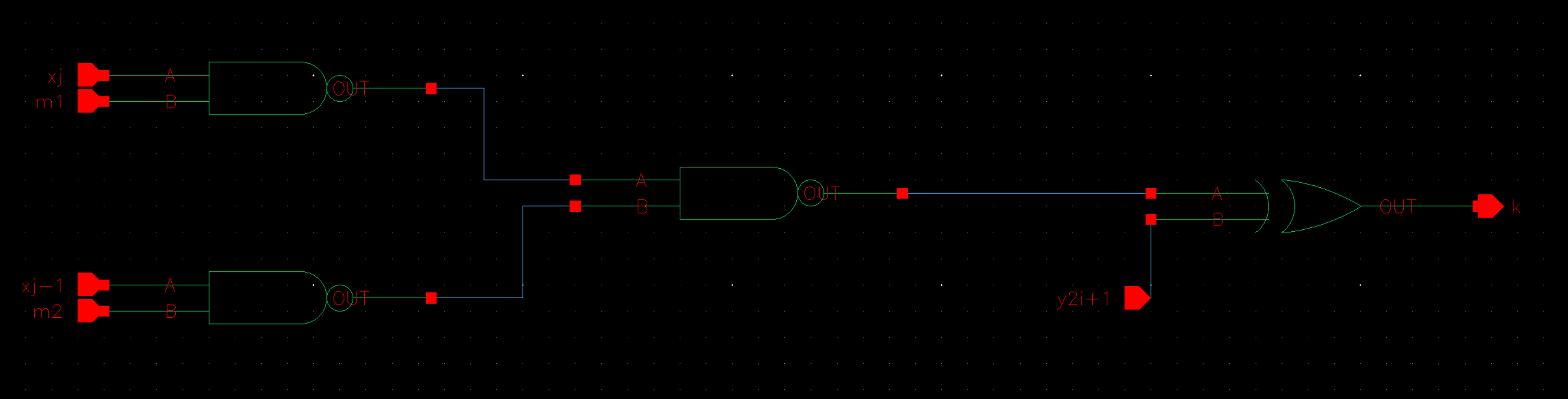
For partial product generator design, you should decide what should be selected among 0X, 1X and 2X based on Y2i-1, Y2i, Y2i+1. The circuit(Booth Encoder) shown below generates m1 and m2 signal to distinguish 0X, 1X and 2X three cases.



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

Z<15> to Z<0>

RCA

CSA

CSA

CSA

Sum2

Carry2

S3

Carry1

Sum1

Carry0

Sum0

PP3

PP2

PP1

PP0

1. **Functionality Test**

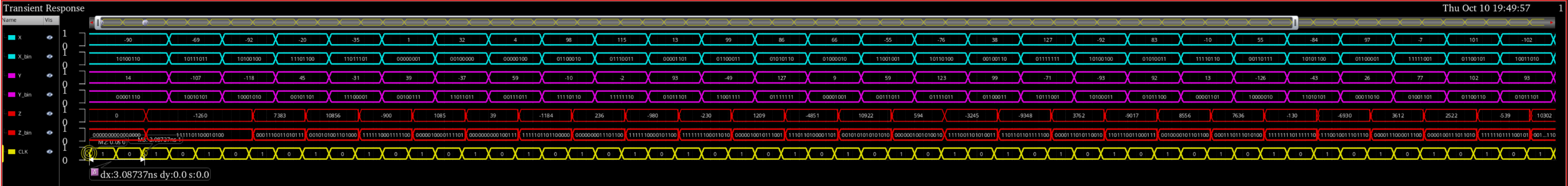
My Python scripts are shown as below( random\_generator function is used to generate a vector file with random testing patterns):

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. """
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)
18. ## division of python returns a float number
19. the\_num = int(the\_num / 10)
20. count = count + 1
21. **return** str(dec)
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
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
45. ## Generates random input patterns based on user's configuration
46. ## Parameters:
47. ##      width: the width of binary input patterns (defalt 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:
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 = ' ')
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
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)
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 = ' ', fil e = the\_file)
85. **print**(dec\_to\_bin(int(random.random() \* (2 \*\* width)),width), end = '\n', file = the\_file)
86. time += clk
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):



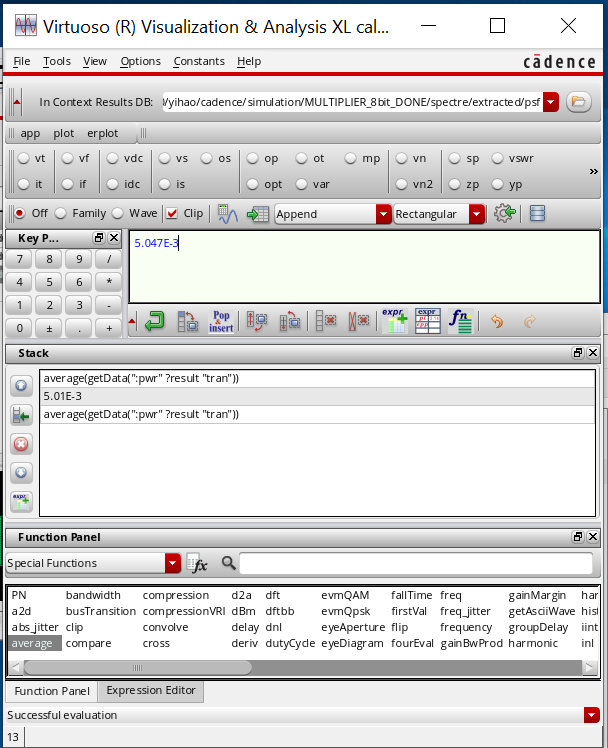
1. **Measurement of Parameters**

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

Width is 106.8um

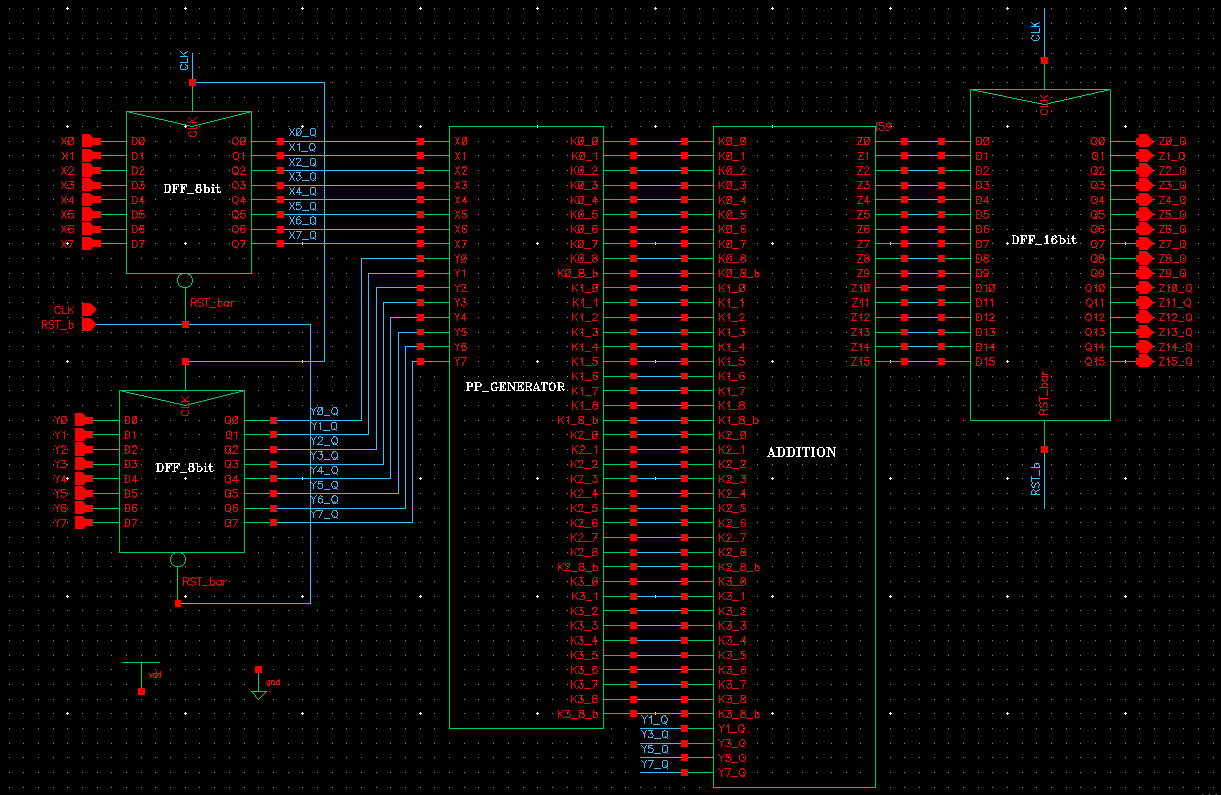
Height is 85.5um

Average Power Consumption is 5.047E-3 W

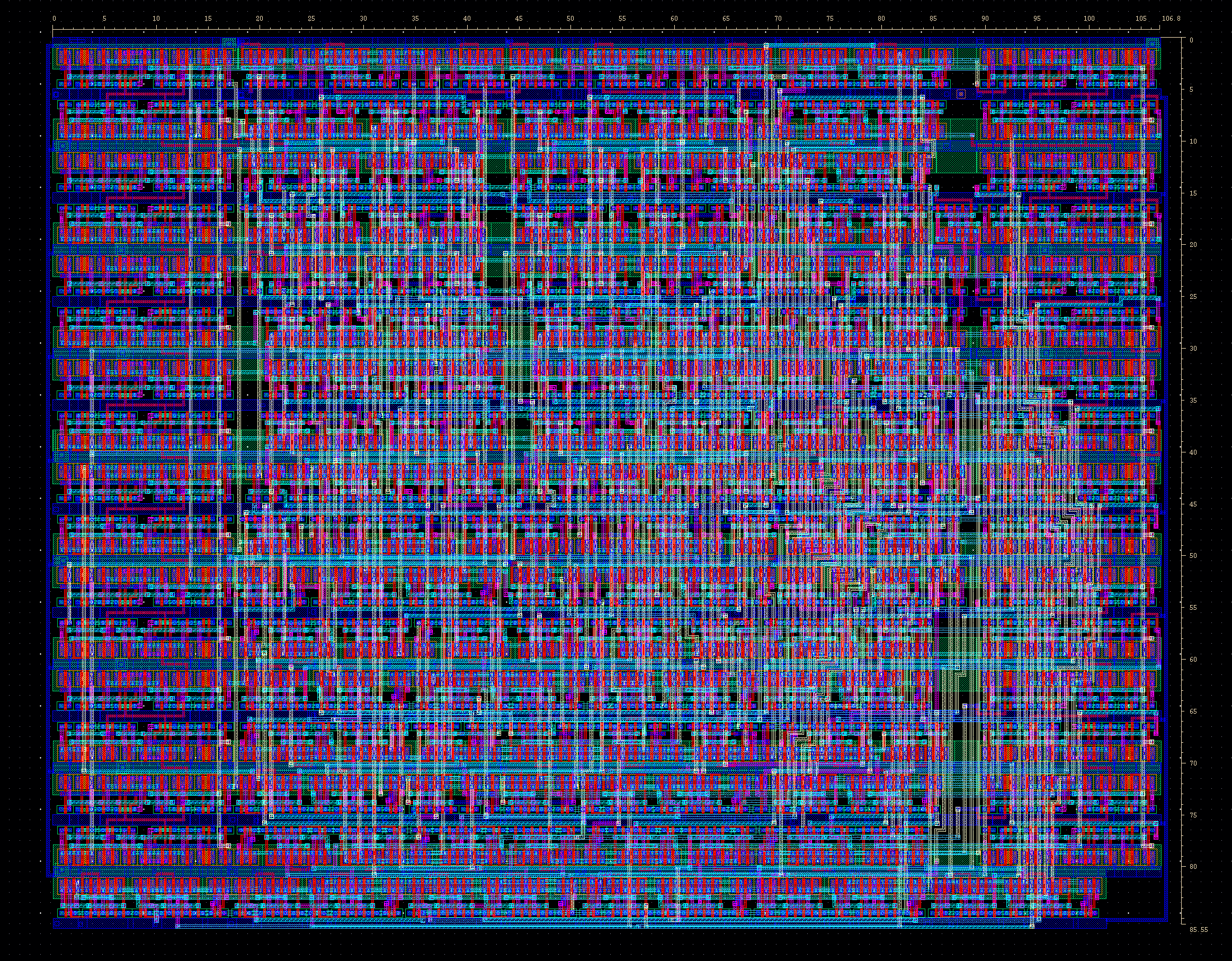


1. **Screenshots**

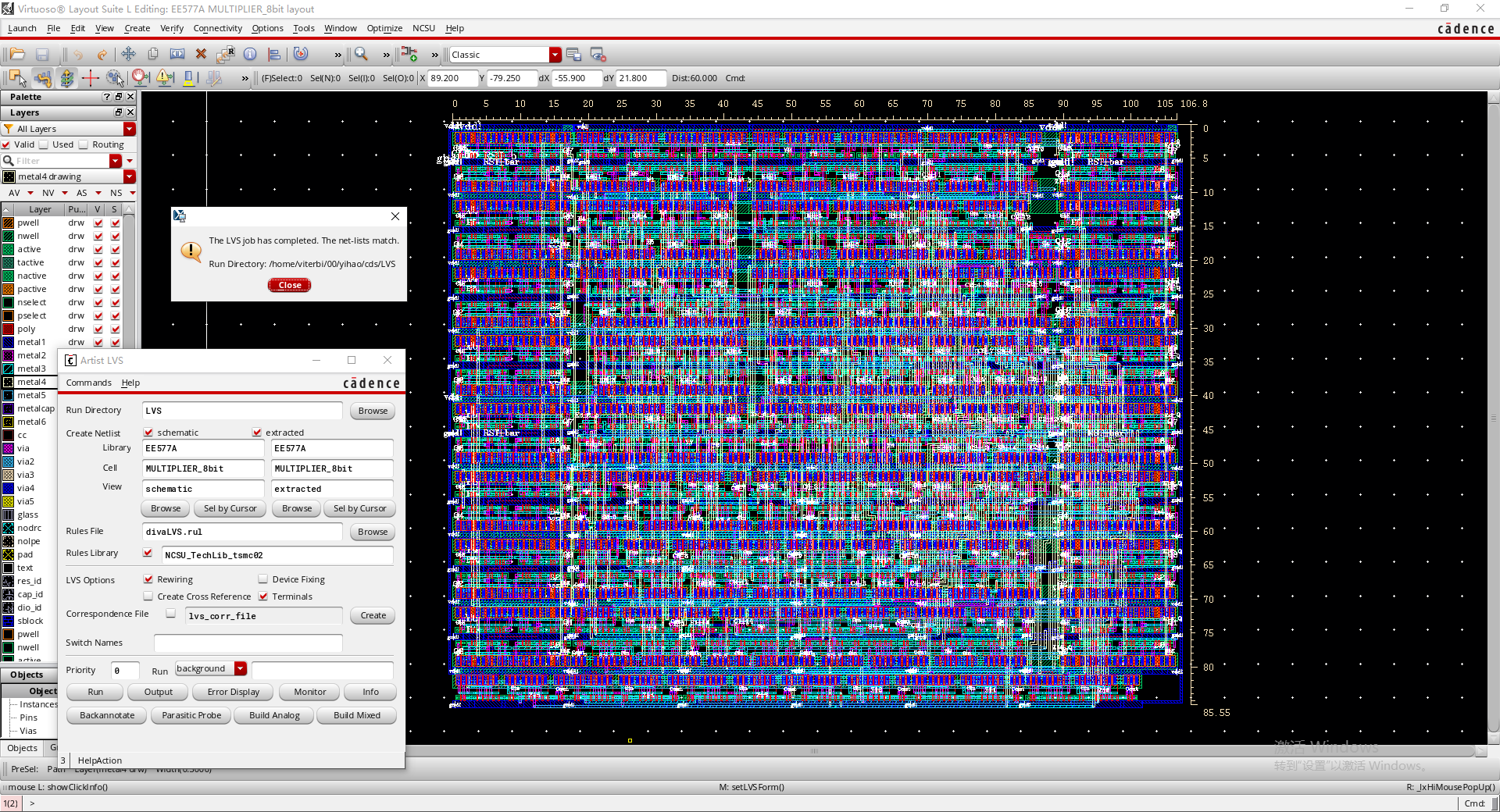
Schematic:



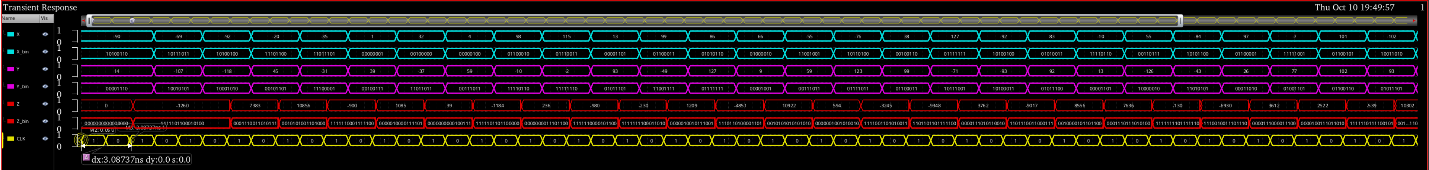
Layout:



LVS:



Functionality test of schematic:



Functionality test of layout:

