1 Bit Full Adder and 5 Bit Multiplier Layout Simulation

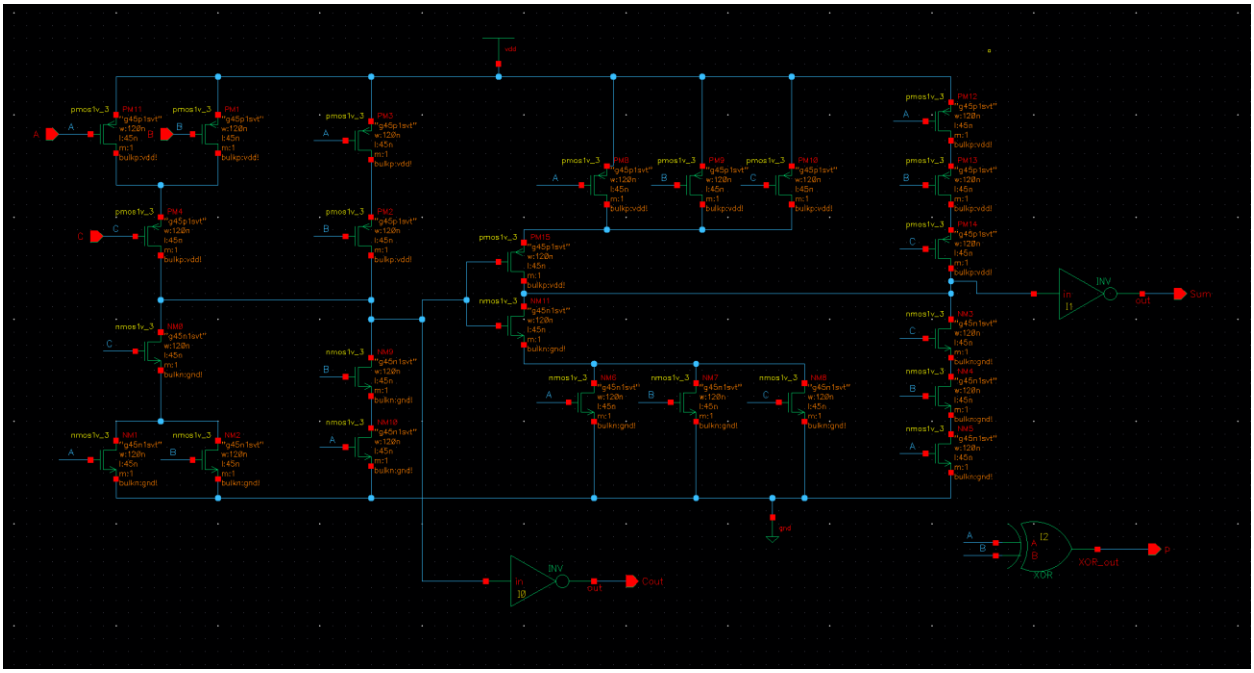
Submitted by –

Rohit Kumar Singh

Content:

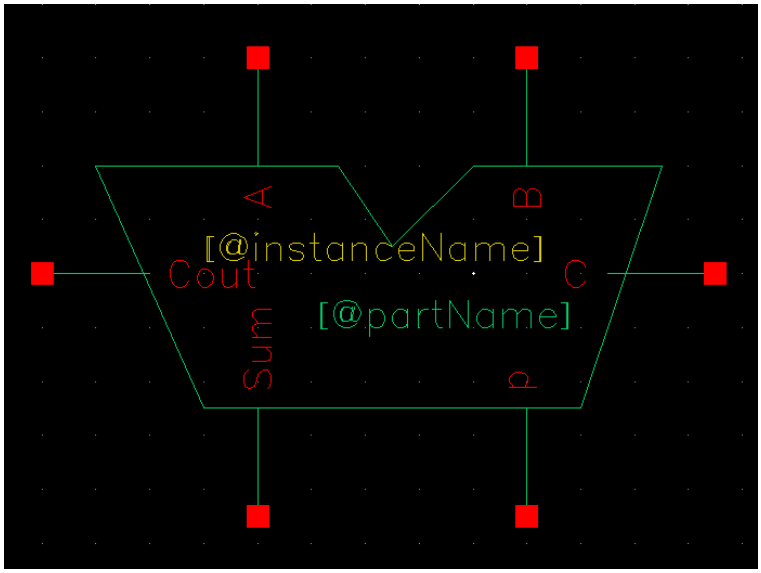
1-Bit Full Adder –

The 1-bit Full Adder is the fundamental building block of our 5 Bit Full Adder. We have constructed it using Duality.



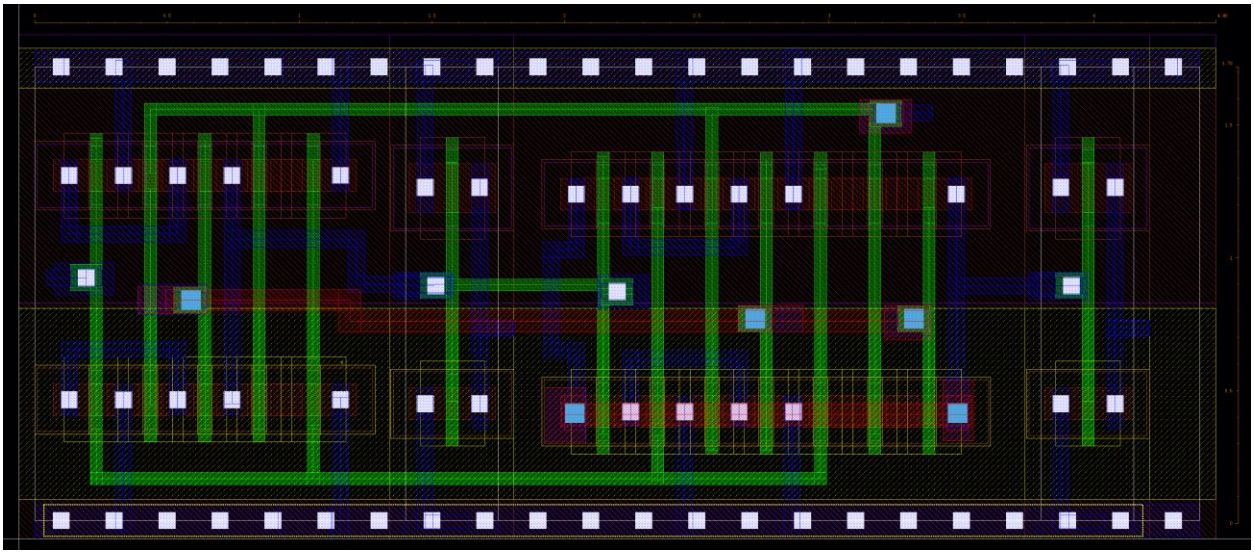
Fiq 1 : 1- Bit Full Adder Schematic

Below is the symbol of our 1-bit Full Adder, we will use it to instantiate in our top-level design in 5-bit Half adder and Multiplier.

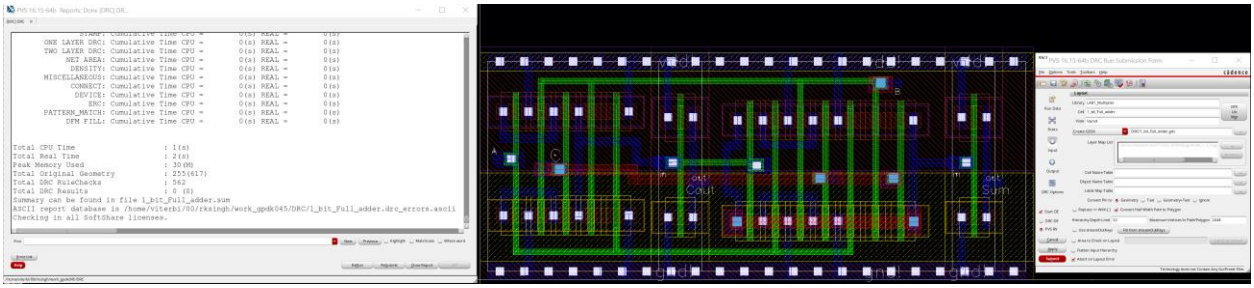


Fiq 2: 1- Bit Full Adder Symbol

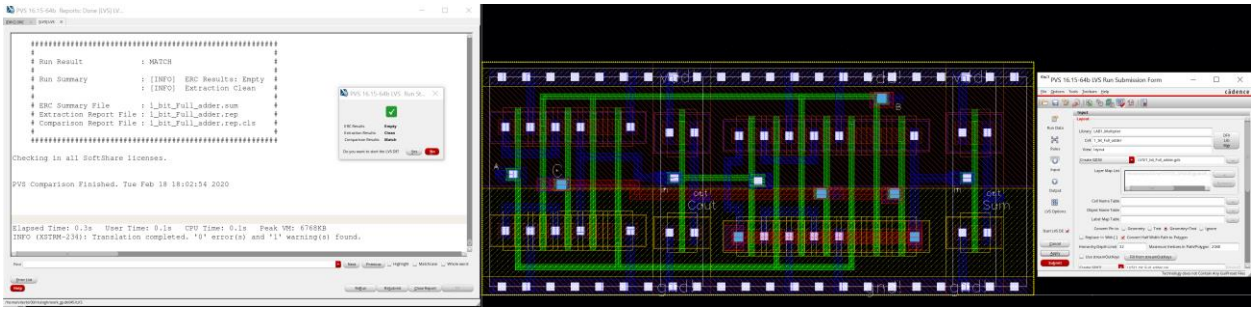
To design the layout of our Full Adder, I have divided it into smaller chunks of Euler path. I have calculated the eular path of Carry Out and then utilized that for the input of the sum calculator. Below is the layout of my 1-bit full adder.



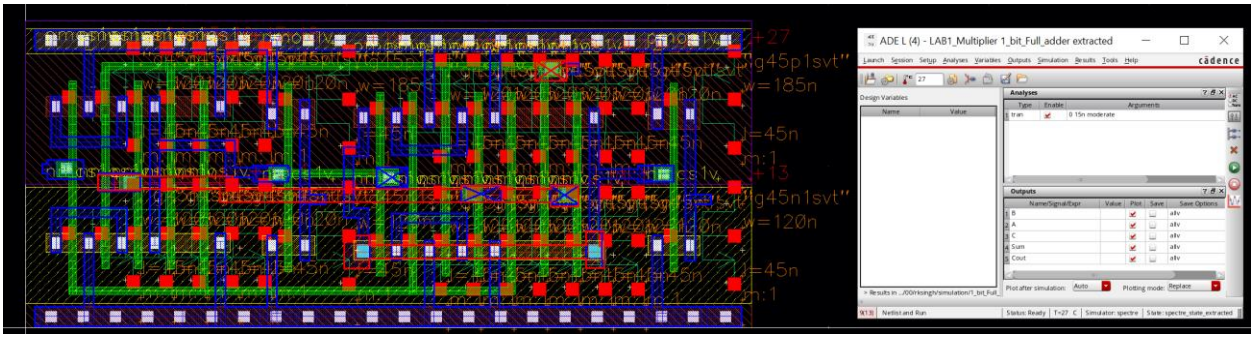
Fiq 3 : 1- Bit Full Adder Layout



Fiq 4 : 1- Bit Full Adder Layout – DRC Check

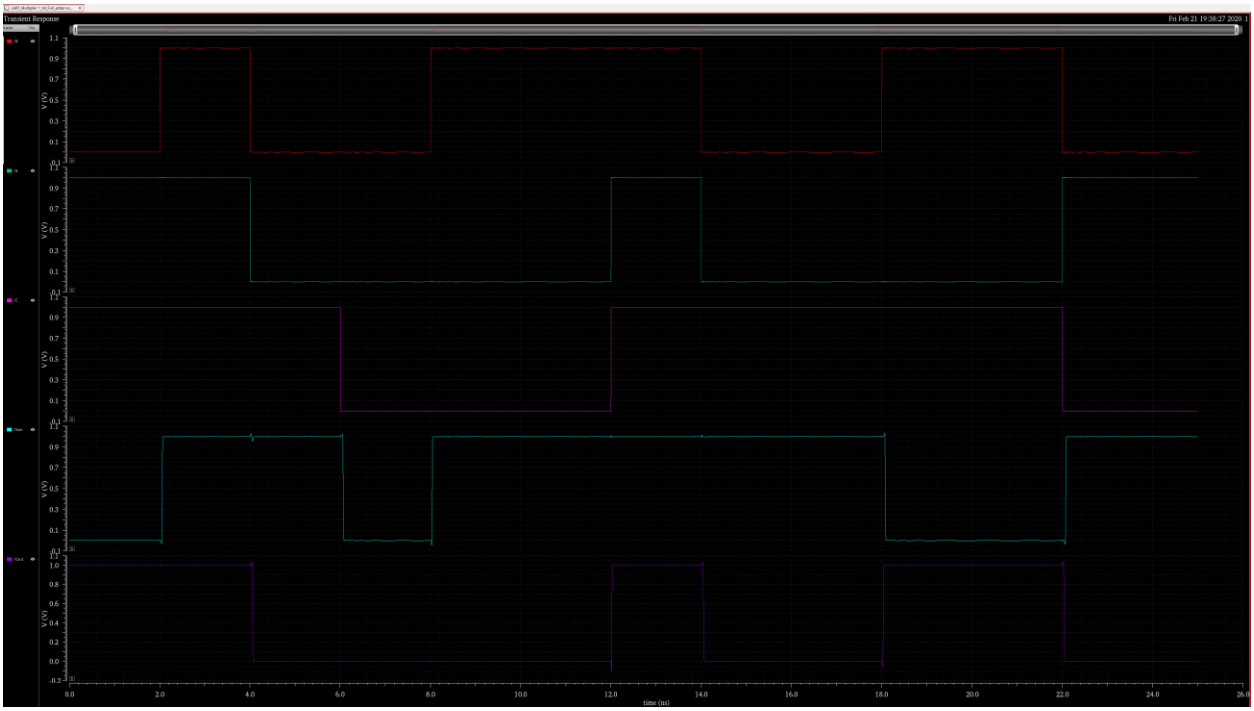


Fiq 5 : 1- Bit Full Adder Layout – LVS Check



Fiq 6 : 1- Bit Full Adder Layout – Extracted ADL

We are using python to generate the vector file. The vector file is inserted in our Specter manually by the user. After executing the vector file, the output waveform is verified manually by the user with the golden outputs generated by the python script.



Fiq 7 : 1- Bit Full Adder Layout Simulation

Functional Verification:

```
radix 1 1 1
io i i i
vname A B C
tunit ns
slope 0.005
vih 1.0
vil 0.0
trise 0.005
tfall 0.005

0 1 0 1
2 1 1 1
4 0 0 1
6 0 0 0
8 0 1 0
10 0 1 0
12 1 1 1
14 0 0 1
16 0 0 1
18 0 1 1
20 0 1 1
22 1 0 0
```

Fiq 8 : 1- Bit Full Adder Vector File

Below is the simulation of the schematic. It is a visual comparison of the waveform generated by Schematic and extracted layout.

vector_file_Add_1bit.py

Vector_add_final_1bit.vec

Add_golden_1bit.txt

Golden Output :

A(DEC)	B(DEC)	Cin(DEC)	SUM(DEC)	A(BIN)	B(BIN)	Cin(BIN)	SUM(BIN)
1	0	1	2	1	0	1	010
1	1	1	3	1	1	1	011
0	0	1	1	0	0	1	001
0	0	0	0	0	0	0	000
0	1	0	1	0	1	0	001
0	1	0	1	0	1	0	001
1	1	1	3	1	1	1	011
0	0	1	1	0	0	1	001
0	0	1	1	0	0	1	001
0	1	1	2	0	1	1	010
0	1	1	2	0	1	1	010
1	0	0	1	1	0	0	001

Functional Verification For all inputs:

```
radix 1 1 1
io i i i
vname A B C
tunit ns
slope 0.005
vih 1.0
vil 0.0
trise 0.005
tfall 0.005


0   0 0 0
2   0 0 1
4   0 1 0
6   0 1 1
8   1 0 0
10  1 0 1
11  1 1 0
12  1 1 1
```

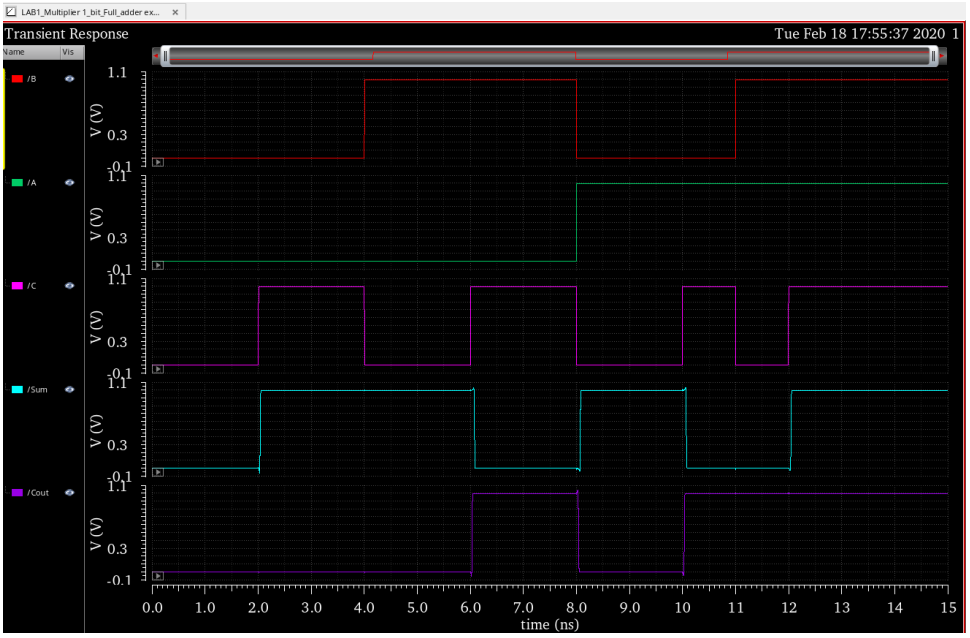


Table:

Length	Width	Area
4.2um	1.7um	7.14um2

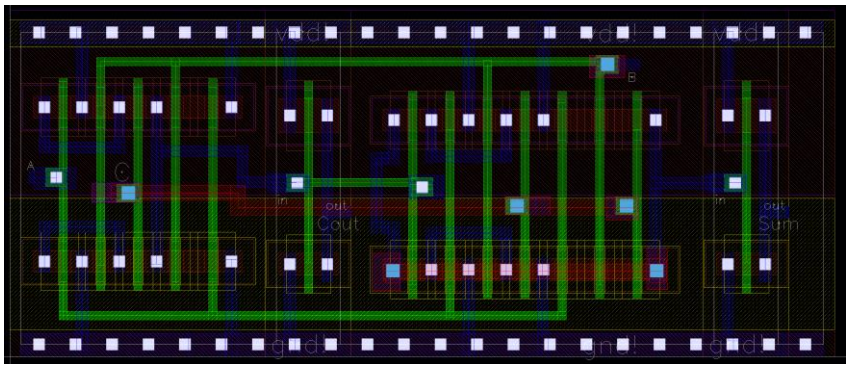
Result and Summary –

Hence the 1 bit Carry Skip Ahead Adder was designed and the proper golden files were generated using python scripting. The implementation of the vector files was executed manually and the verification of the results, showed positive correlation between the golden files and the generated output waveforms. We calculated the Beta ration of our inverter and found out the critical path. Hence the Layout and Verification of the 1_Bit Carry Skip Ahead Adder is successful.

5- Bit Multiplier:

1-Bit Full-Adder –

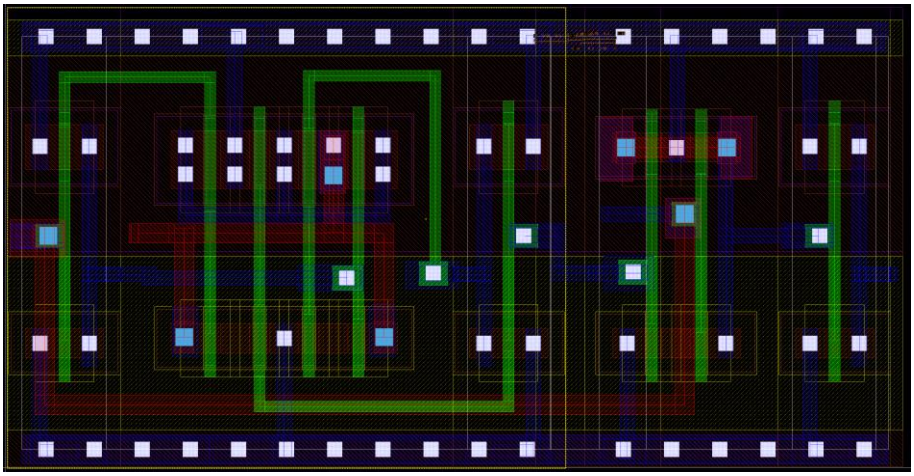
Below is the symbol of our 1 bit Full Adder, we will use it to instantiate in our top level design in 5 bit Multiplier.



Fiq 1 : 1- Bit Full Adder Layout

1-Bit Half Adder –

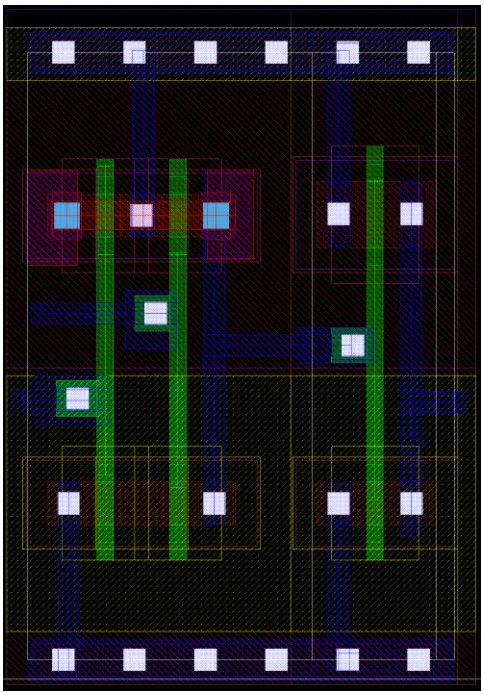
By utilizing a Xor and And gate we are forming a four 1-bit Full Adder, we are generating sum and carry out from the Xor and And Gates.



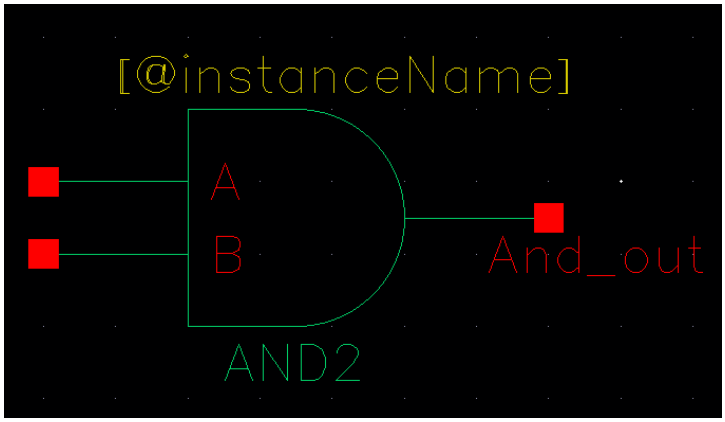
Fiq 2: 4- Bit Full Adder Layout

Basic Gates –

We are using pmos and nmos transistors to create an AND gate, we are using an inverter which we created on our own using a CMOS inverter circuit.

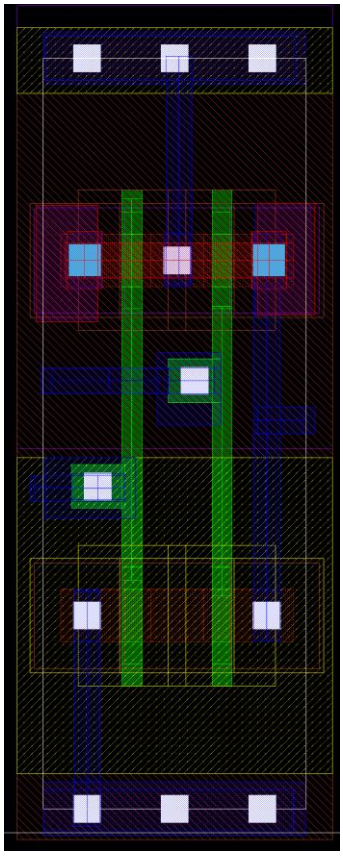


Fiq 3 : AND Gate Layout

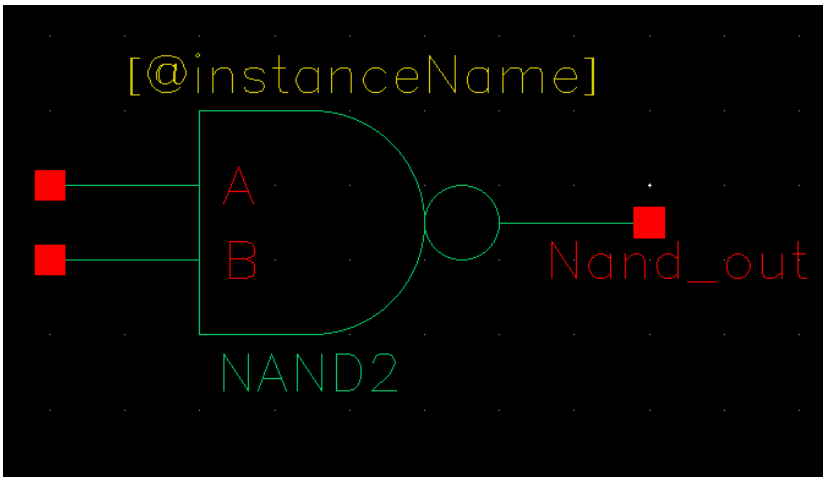


Fiq 4 : AND Gate Symbol

We are using pmos and nmos transistors to create an NAND gate.

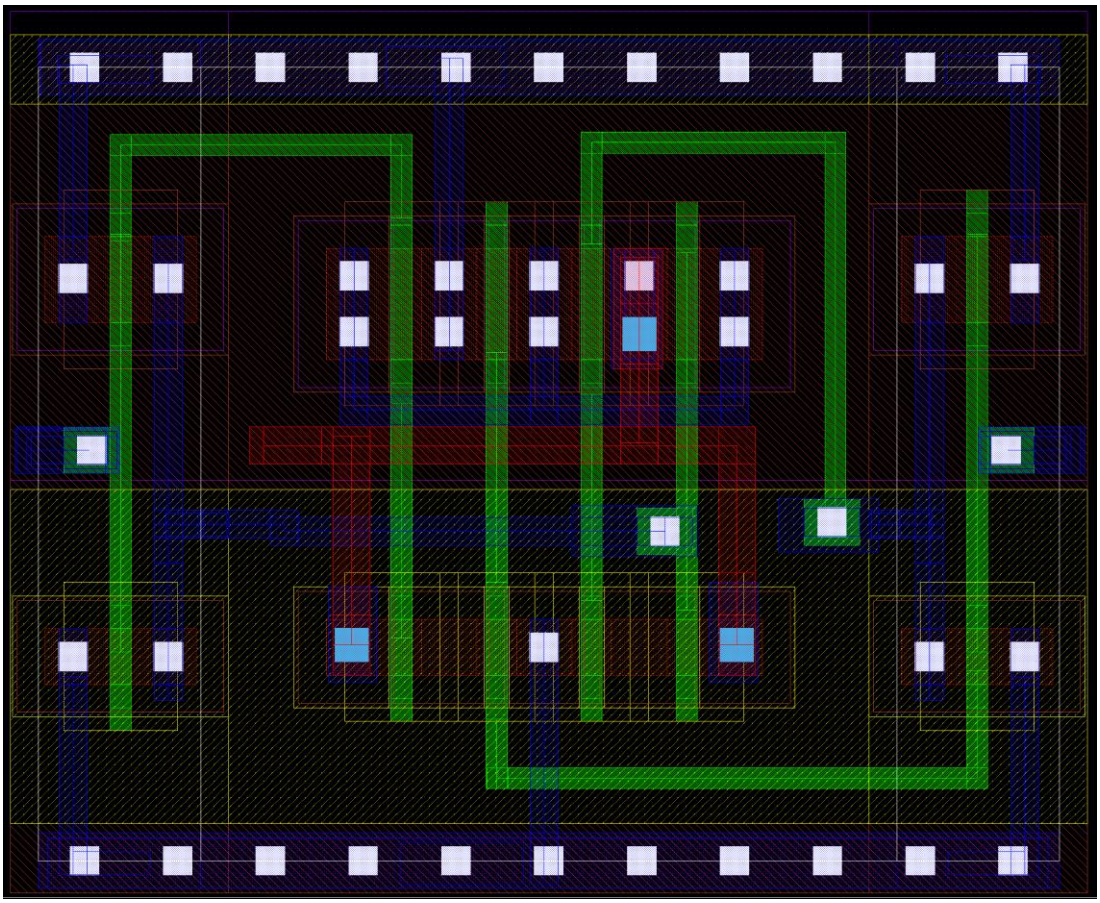


Fiq 5: NAND Gate

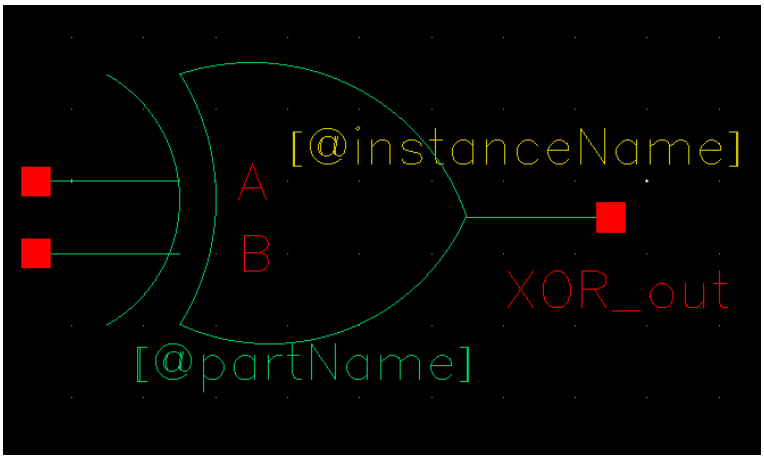


Fiq 6 : NAND Gate Symbol

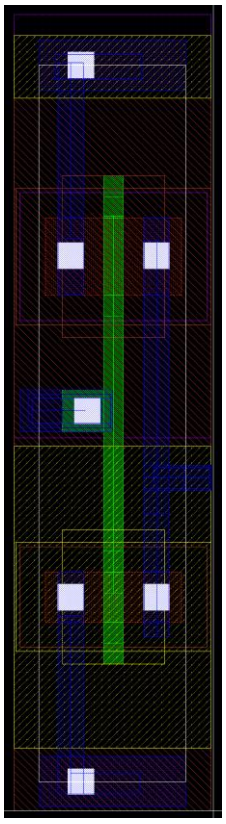
We are using pmos and nmos transistors to create an XOR gate. I have increased my width to reduce the trise and tfall.



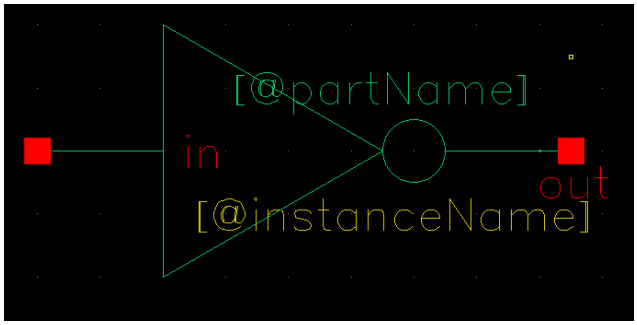
Fiq 7 : NAND Gate Symbol



Fiq 10 : NAND Gate Symbol



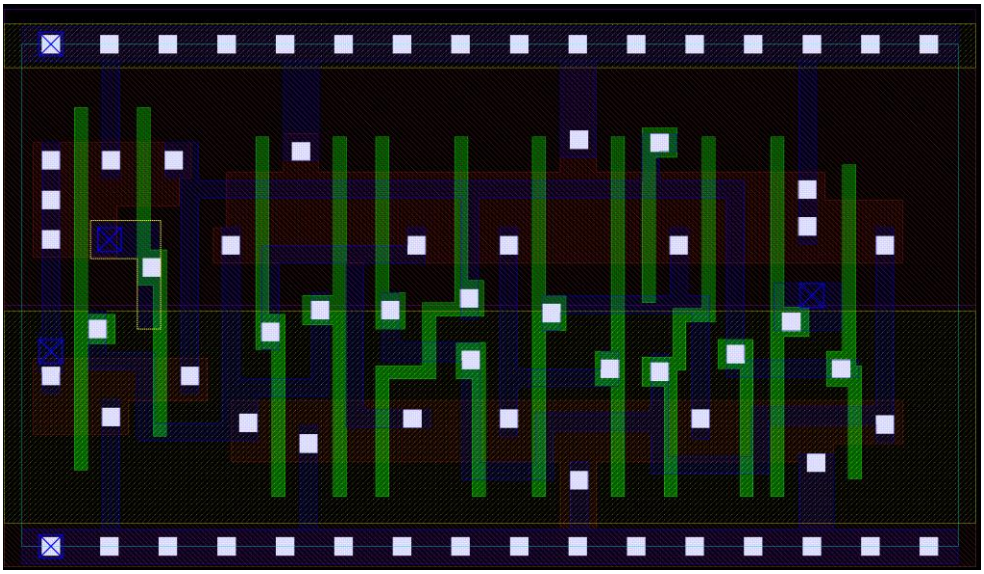
Fiq 11 : Inverter Gate Schematic



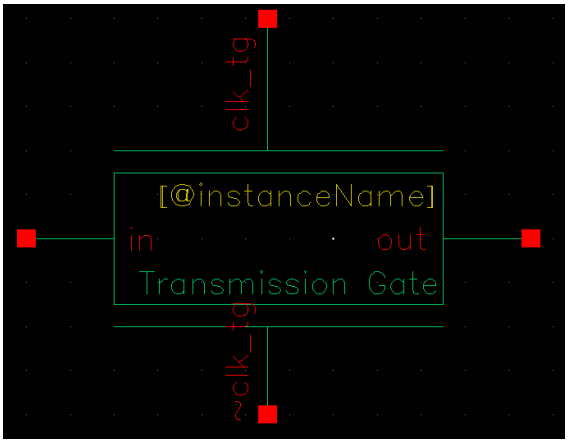
Fiq 12 : Inverter Gate Symbol

D Flip Flop –

The D flip flop was instantiated from the standard library, to save the area of our layout.

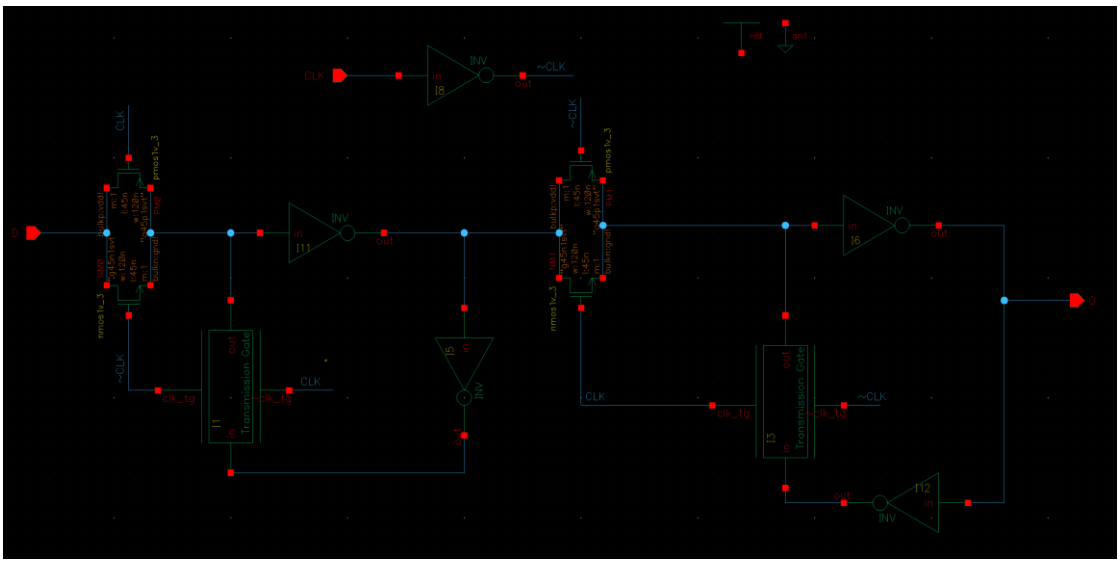


Fiq 13 : D Flip Flop Layout



Fiq 14 : Transmission Gate Symbol

Instantiating the Inverter and the transmission gate, we are creating the Master Slave D Flip Flop, to test the functionality of the D Flip Flop, I am providing a PWL Signal as an input just before the set-up time.

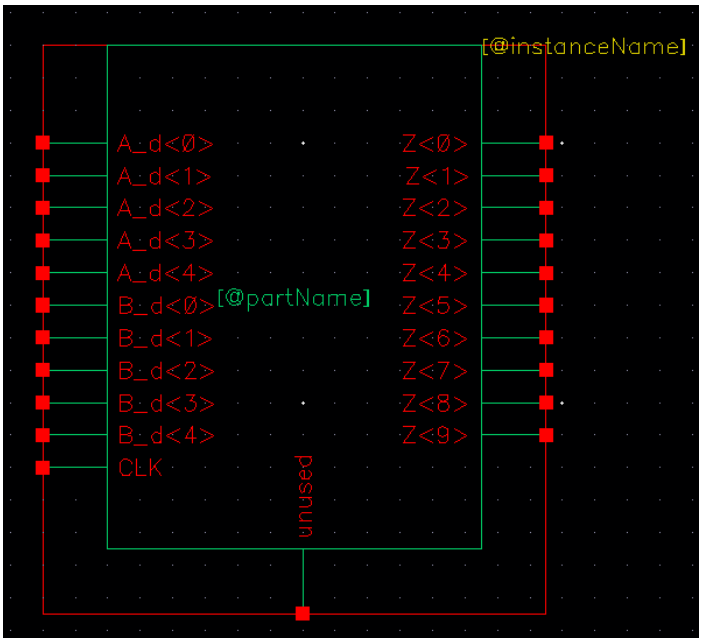


Fiq 15 : D Flip Flop Schematic

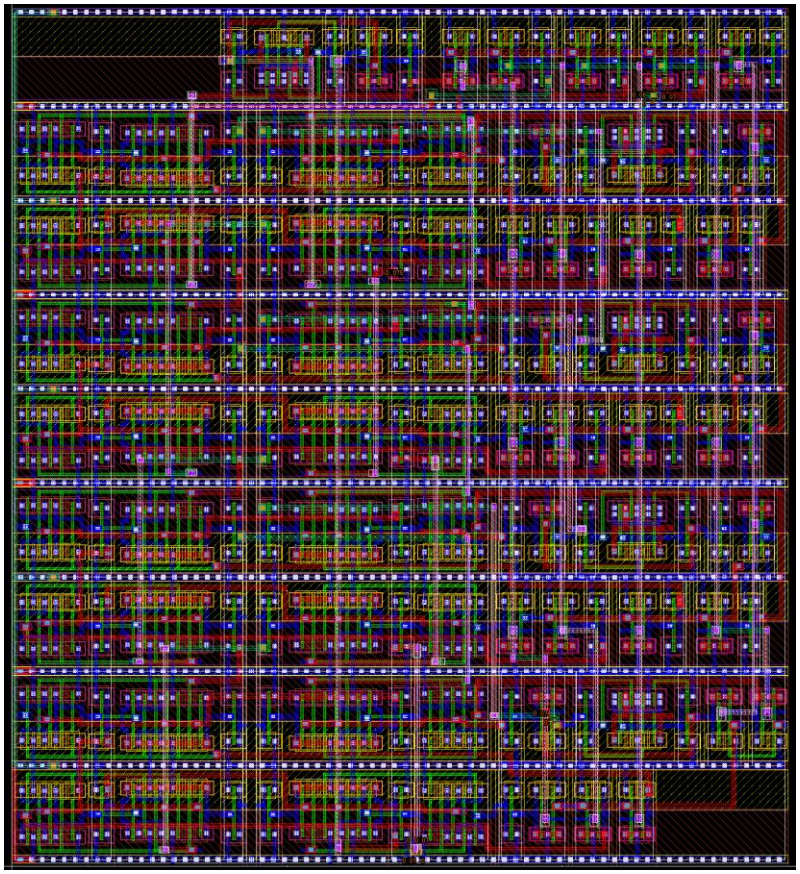
Fiq 16 : D Flip Flop Symbol

Multiplier -

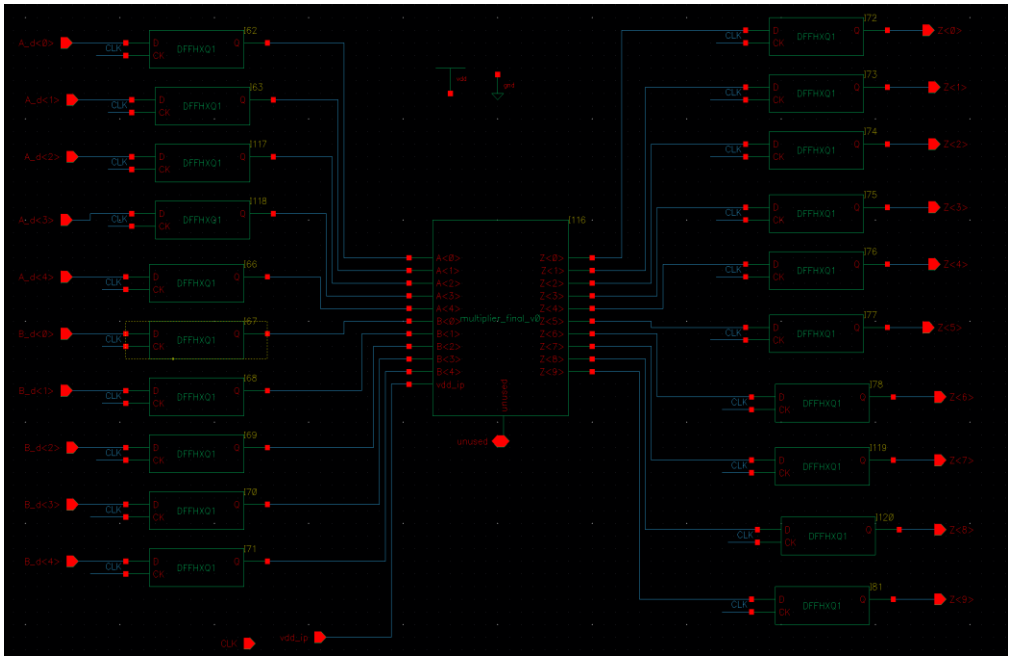
As instructed we have used multiple D Flip Flops, for every input we are using one D Flip Flop and for every output we are using one D Flip Flop. The Flip Flop syncs the input and outputs with the clock. I divided the whole layout into multiple rows, which helped me figure out how to place my instances, I have placed all my first row on the top most part of my layout, then the second, third and the fourth row. Every row comprises of 1 half adder, followed fy 4 full adders and, nand gates. They are filled to save the area and to share vdd and ground.



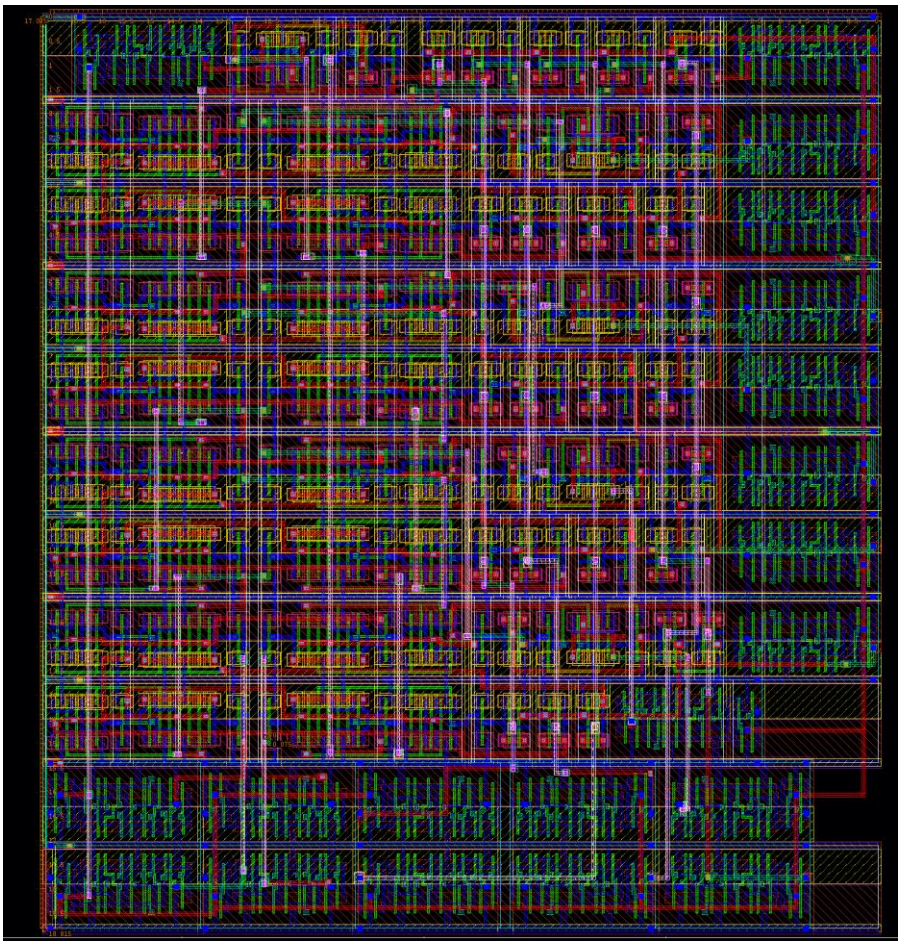
Fiq 17 : Multiplier Schematic with D Flip Flop Symbol



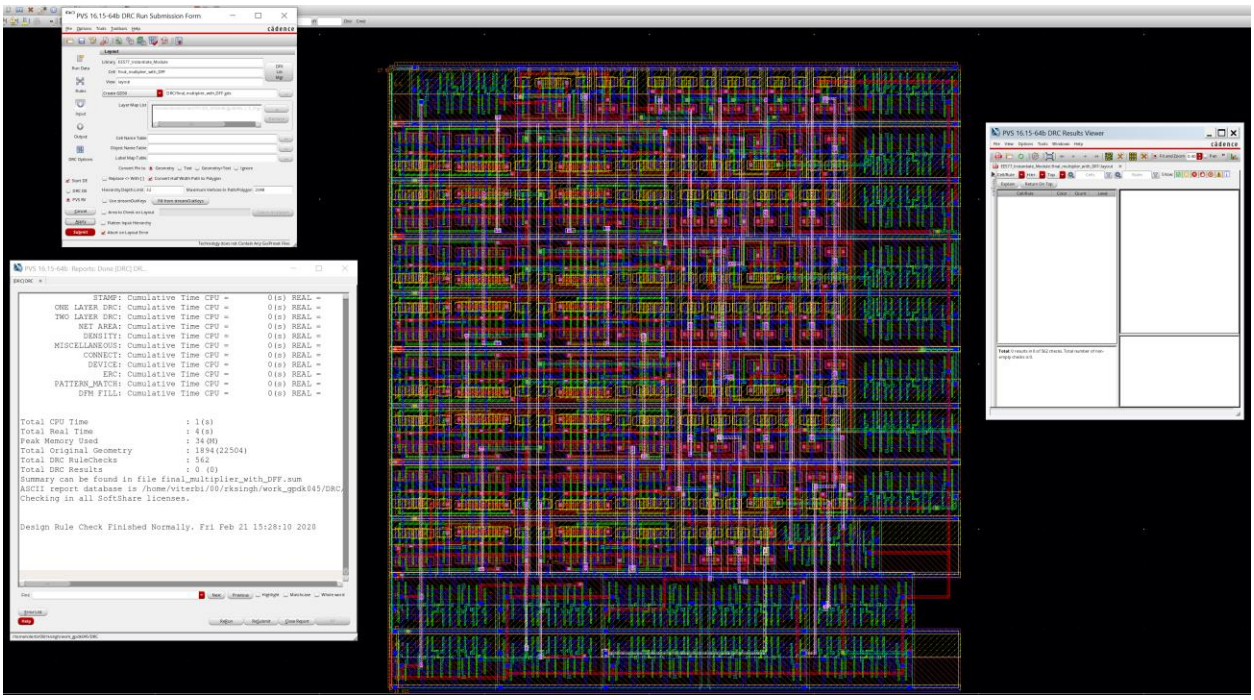
Fiq 18 : Multiplier Layout without D Flip Flop



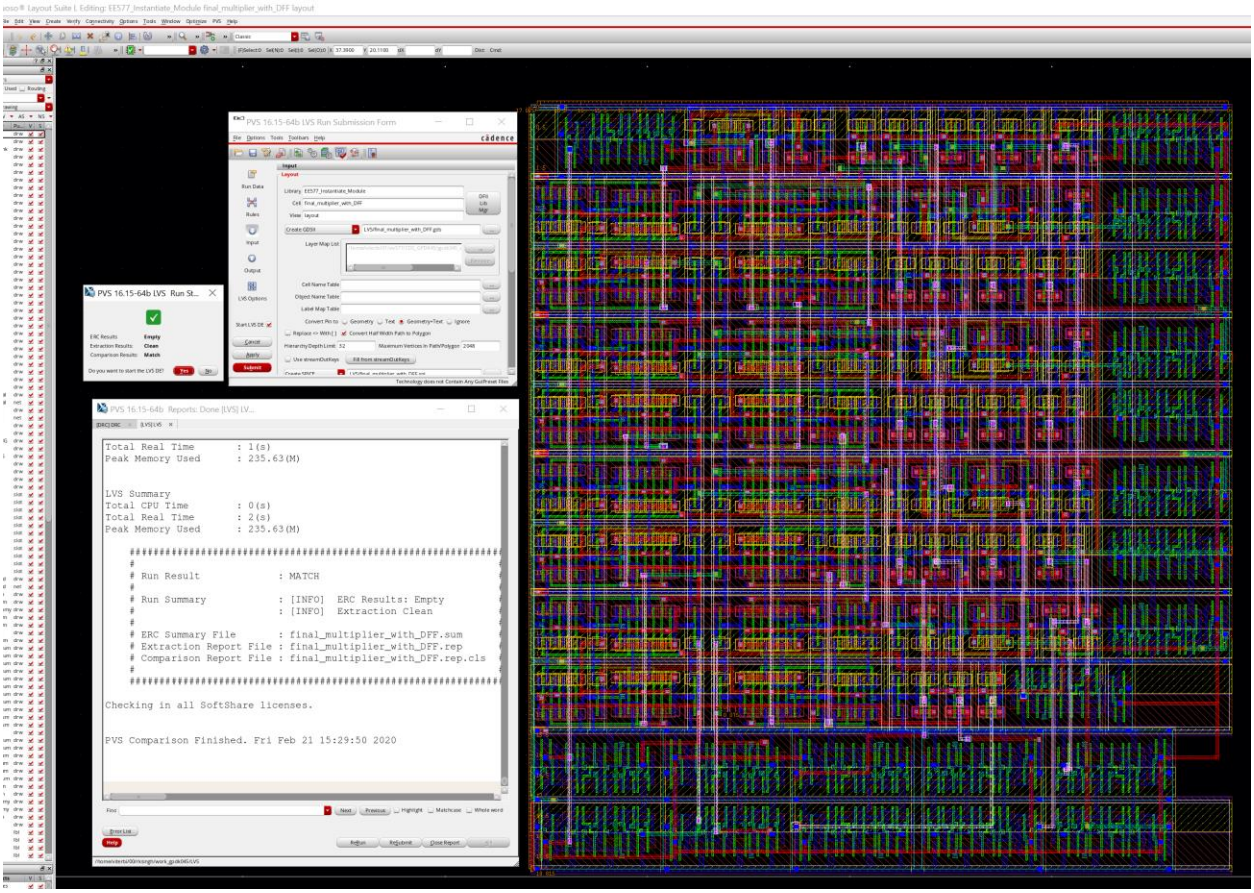
Fiq 19 : Multiplier Schematic with D Flip Flop



Fiq 20 : Multiplier Layout with D Flip Flop



Fiq 21 : Multiplier Layout DRC Check



Fiq 22: Multiplier Layout LVS Check

Functional Verification:

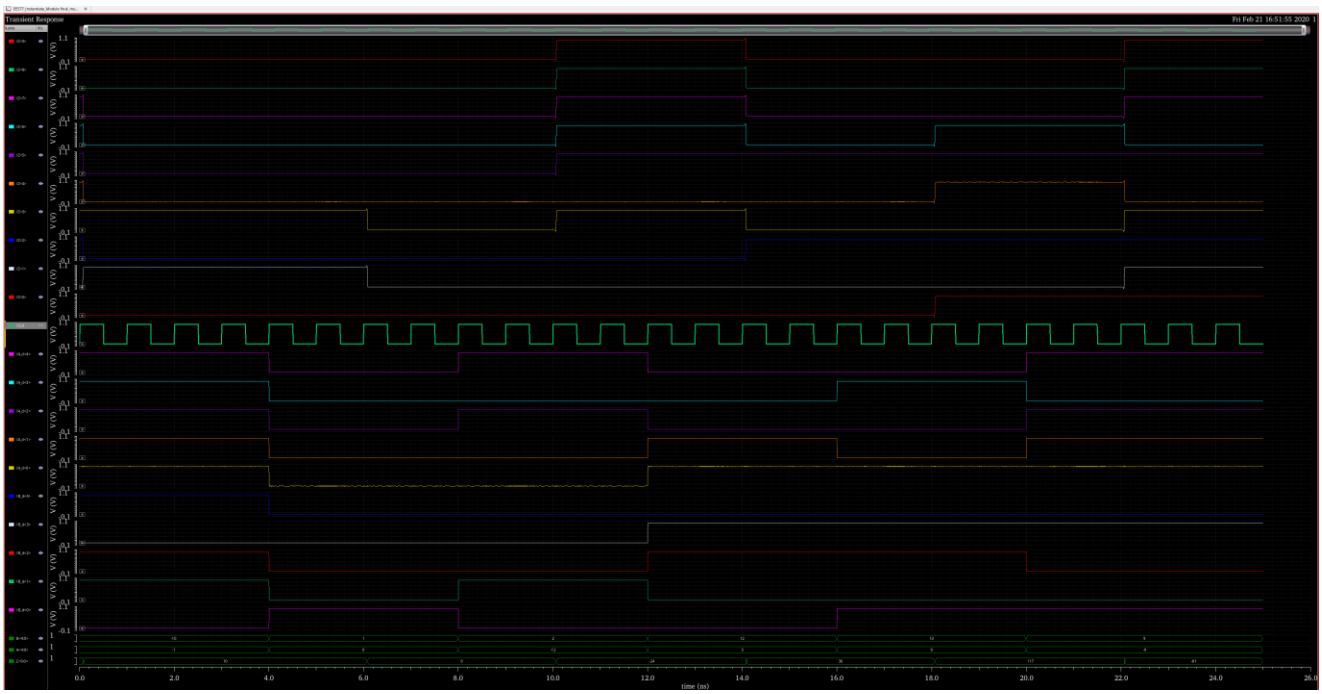
```
radix 1 4 1 4
io i i i i
vname B_d<4> B_d<[3:0]> A_d<4> A_d<[3:0]>
tunit ns
slope 0.005
vih 1.0
vil 0.0
trise 0.005
tfall 0.005

0  1 6 1 F
4  0 1 0 0
8  0 2 1 4
12 0 C 0 3
16 0 D 0 9
20 0 9 1 7
```

Fiq 23 : Multiplier with D Flip Flop Vector File

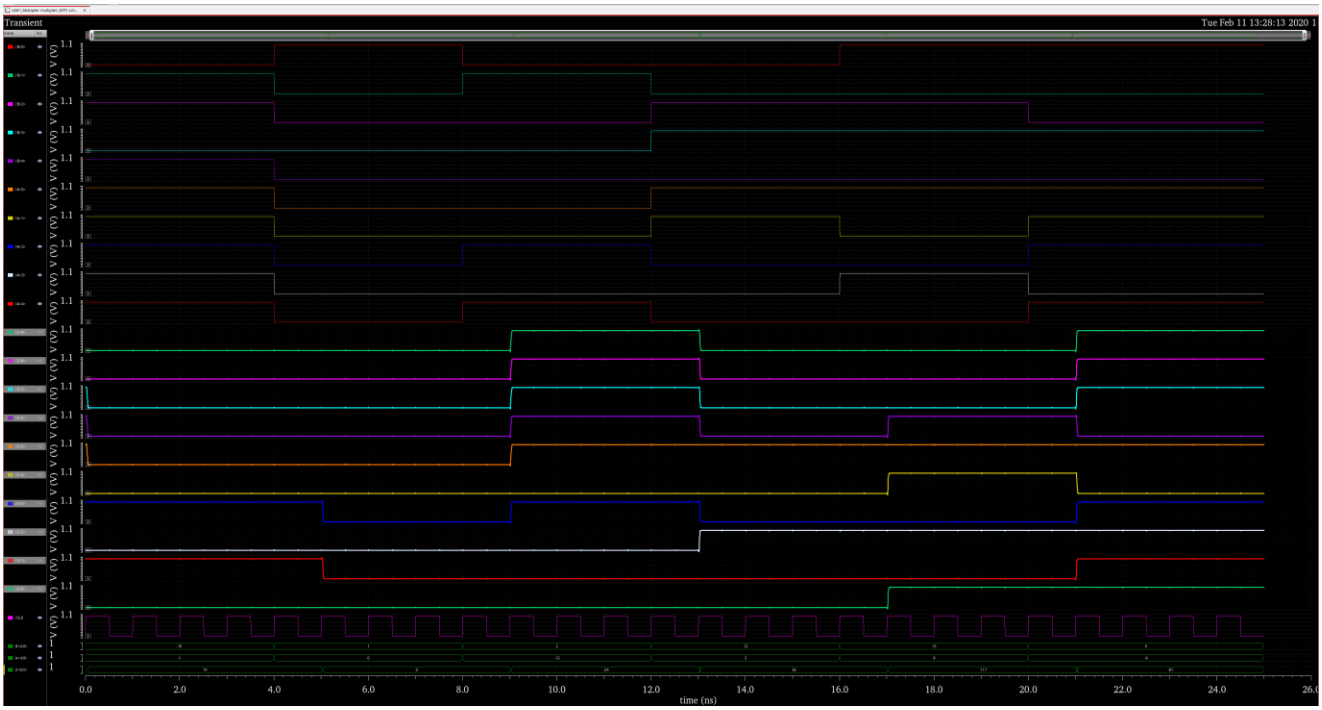
vector_file_Mul.py

Vector_mul_final.vec

We are using python to generate the vector file. The vector file is inserted in our Specter manually by the user. After executing the vector file, the output waveform is verified manually by the user with the golden outputs generated by the python script



Fiq 24 : Multiplier Layout Output Waveform



Fiq 25 : Multiplier Schematic Output Waveform

Golden Output:

A(DEC)	B(DEC)	PROD(DEC)	A(BIN)	B(BIN)	PROD(BIN)
-10	-1	10	10110	11111	0000001010
1	0	0	00001	00000	0000000000
2	-12	-24	00010	10100	1111101000
12	3	36	01100	00011	0000100100
13	9	117	01101	01001	0001110101
9	-9	-81	01001	10111	1110101111

Fiq 26 : Multiplier Output Golden File

Mul_golden.txt

Table:

Length	Width	Area
10.8um	17.12um	184.96um2

Result and Summary –

Hence the 5-bit Multiplier was designed and the proper golden files were generated using python scripting. The layout is designed to be a perfect square in order to save as much area as possible. The implementation of the vector files was executed manually and the verification of the results, showed positive correlation between the golden files and the generated output waveforms. All the connects were performed, using within Metal 4. Hence the Design and Verification of the 5 Bit Multiplier is successful.