

**A  
REPORT  
ON**

**8 x 8 UNSIGNED ARRAY MULTIPLIER AND  
'LEFT' & 'RIGHT' SHIFT ALGORITHMS FOR  
MULTIPLICATION**

**By**

<b>Guduru Sai Suresh</b>	<b>2023H1230183H</b>
<b>Radhanath Mishra</b>	<b>2023H1230185H</b>
<b>Samudrala Lakshmi Srilekha</b>	<b>2023H1230164H</b>
<b>Thathapudi Sanjeev Paul Joel</b>	<b>2020AAPS0120H</b>

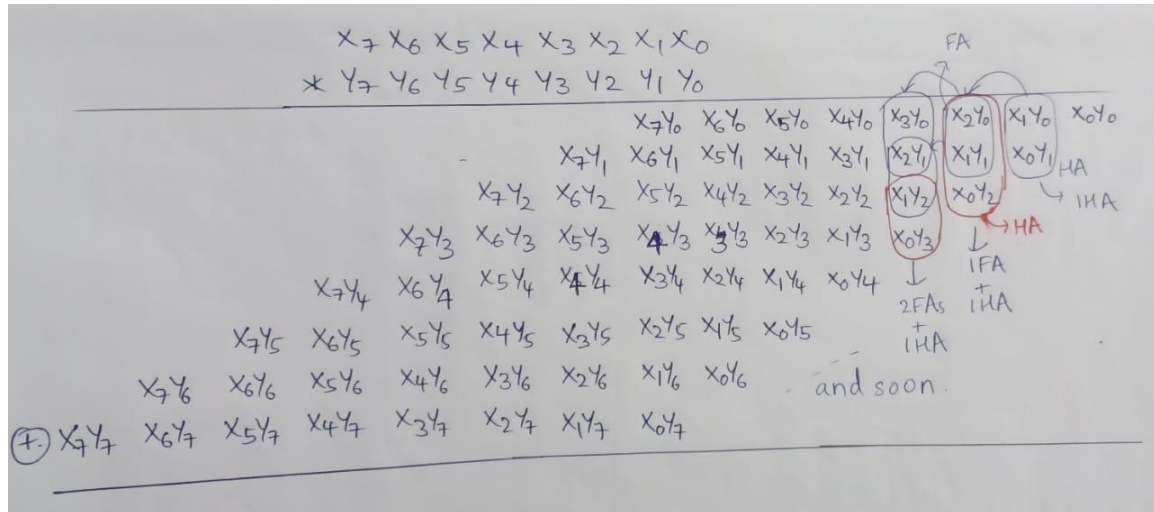
**Prepared as credit fulfillment for the  
Course VLSI Design, Course No: MEL G621**



**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,  
PILANI (Rajasthan)  
(November, 2023)**

## PART - A

### DESIGN FLOW



### Hardware used:

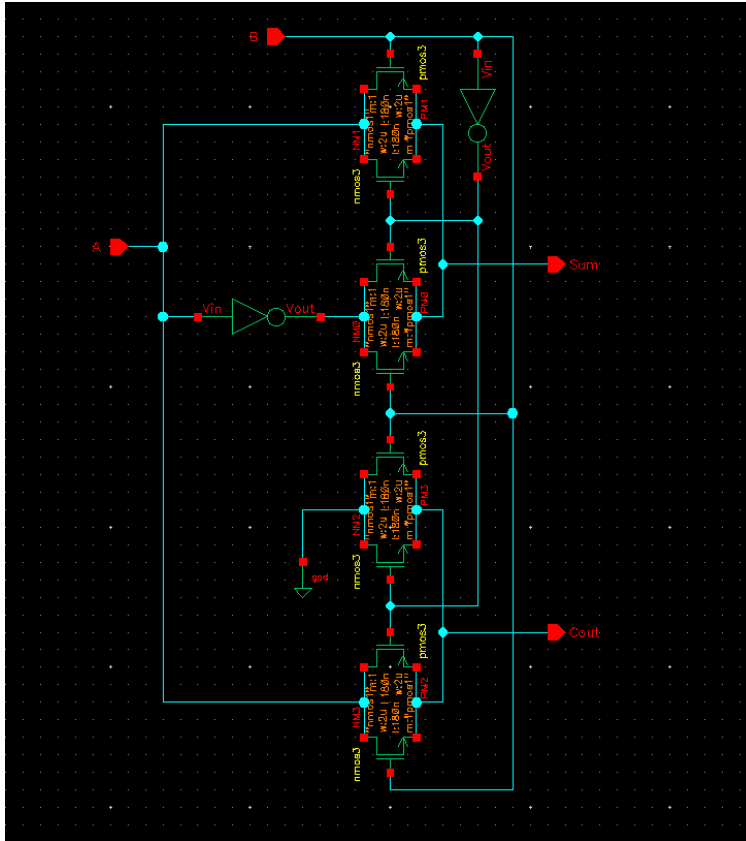
No of half adder modules = 8

No of full adder modules = 48

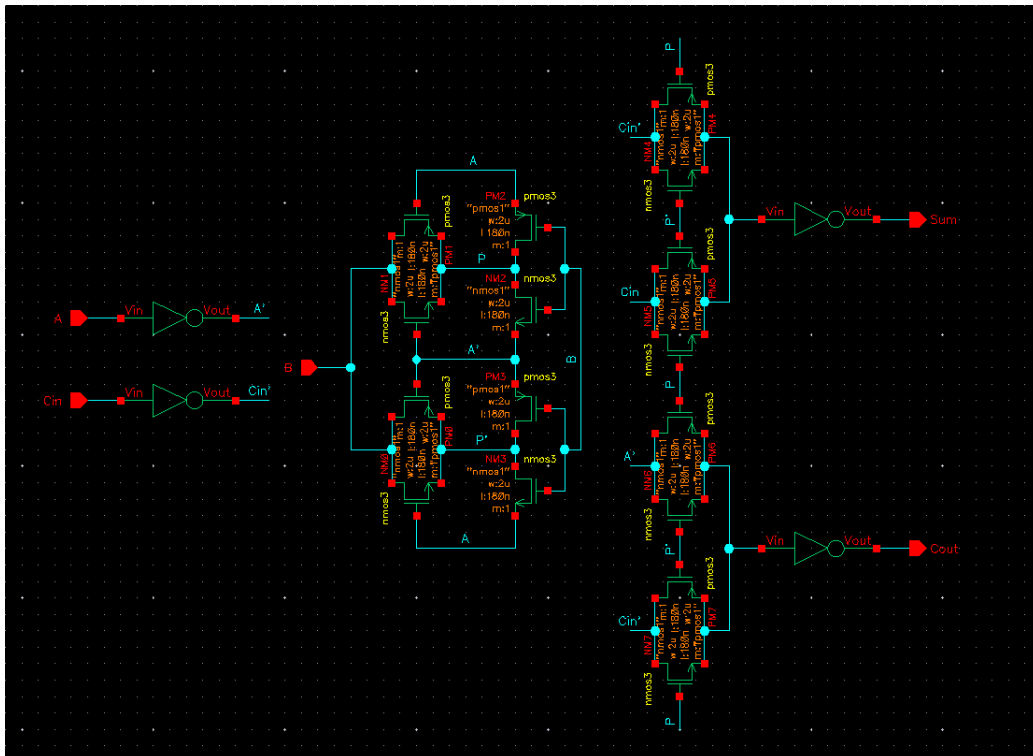
No of AND gates = 64

### SCHEMATICS

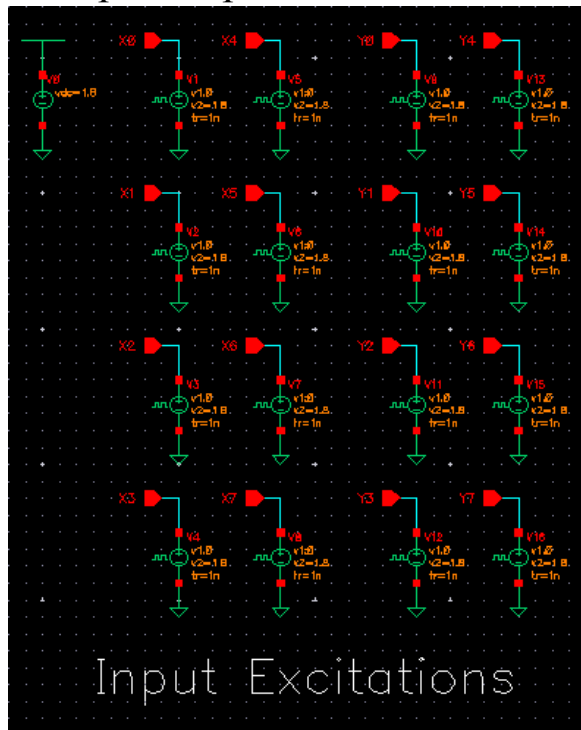
#### Half Adder



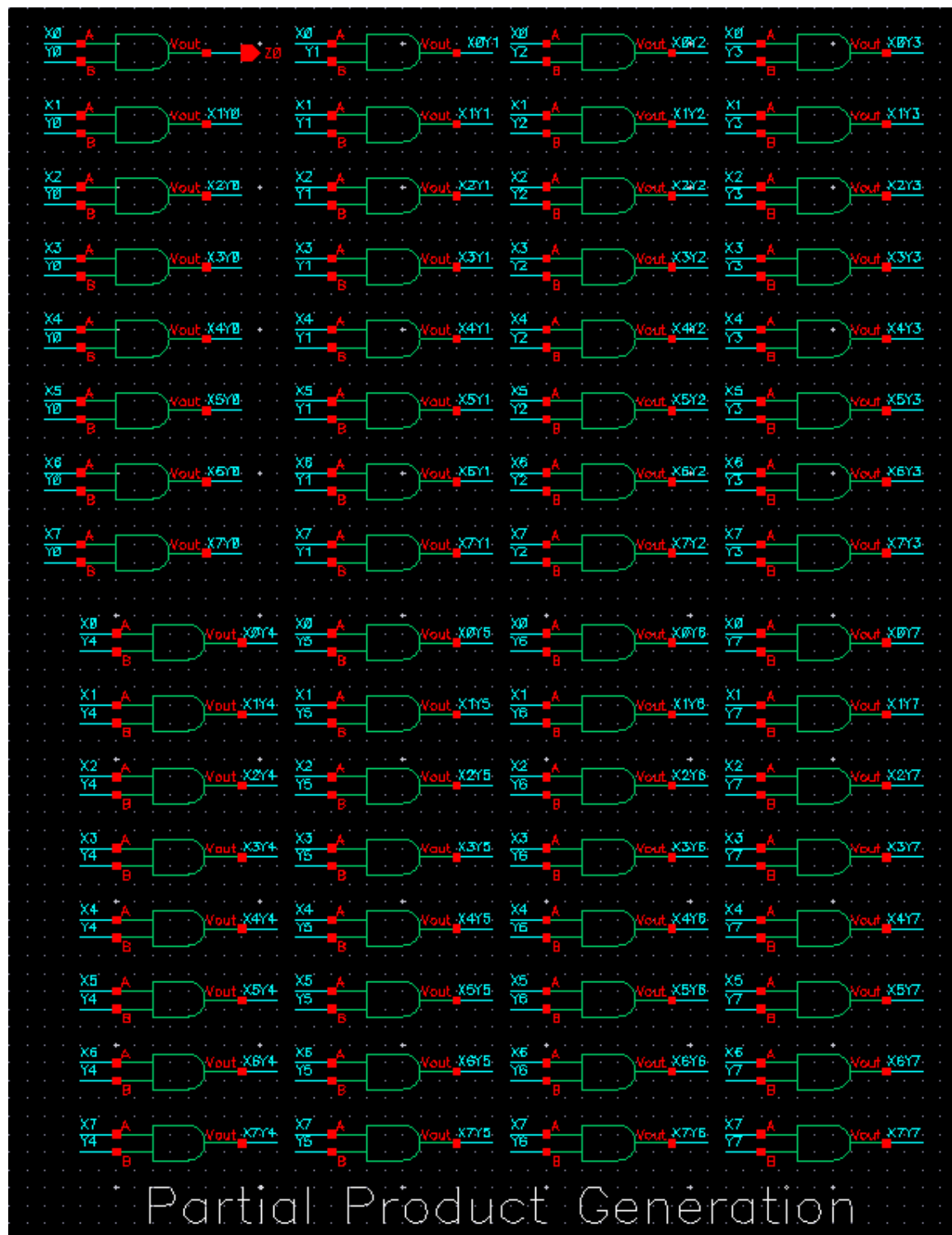
## Full Adder



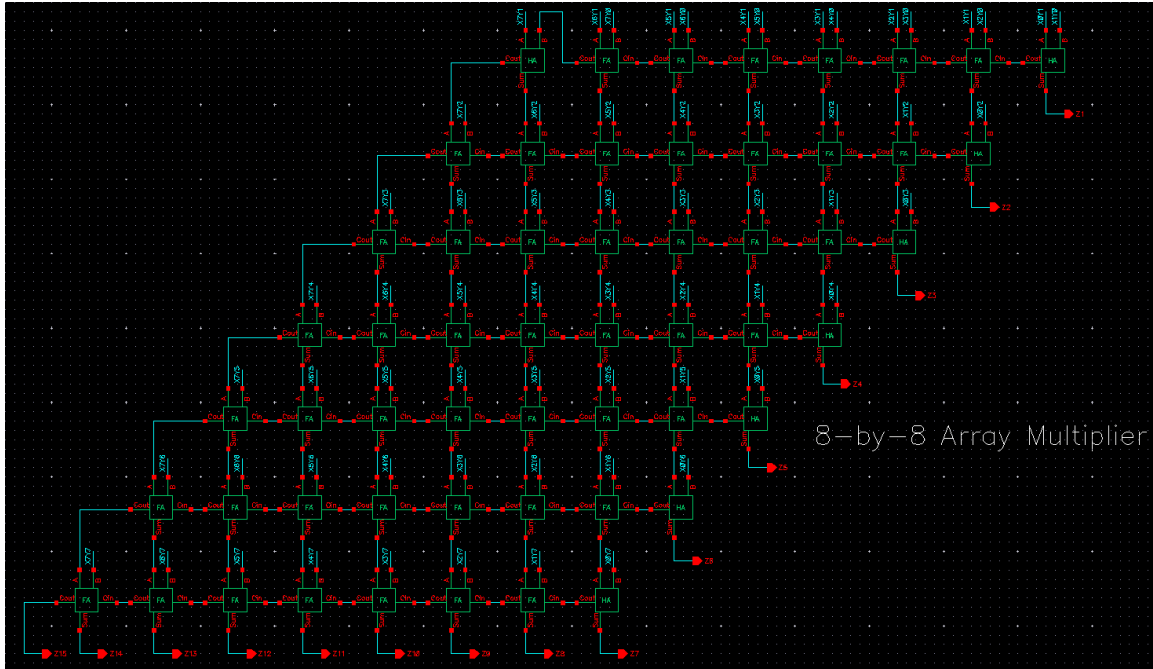
## Multiplier Input Excitations



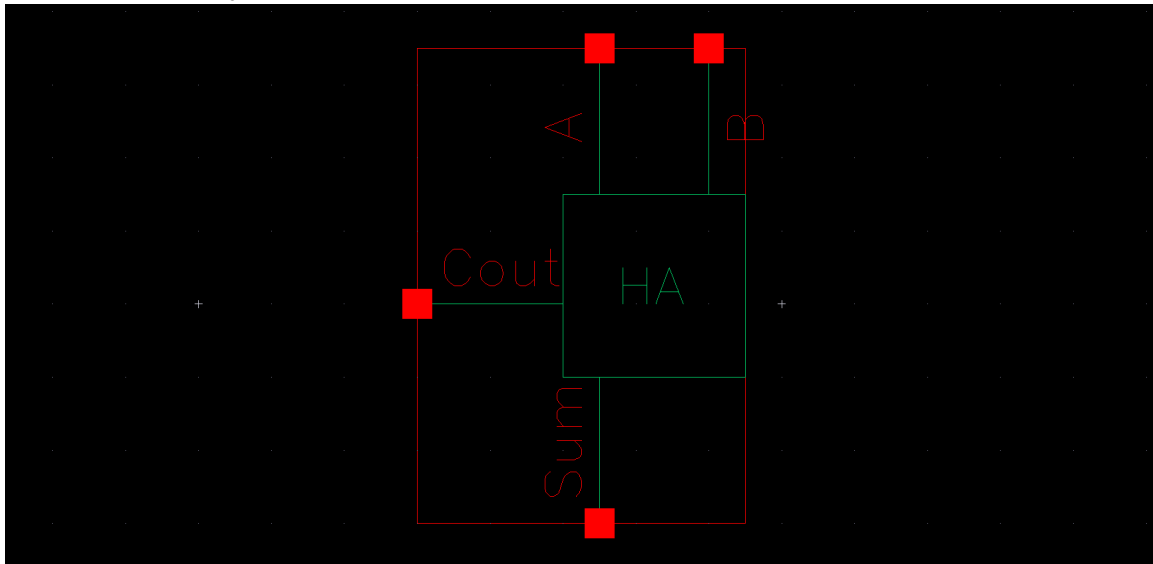
## Multiplier Partial Product Generation



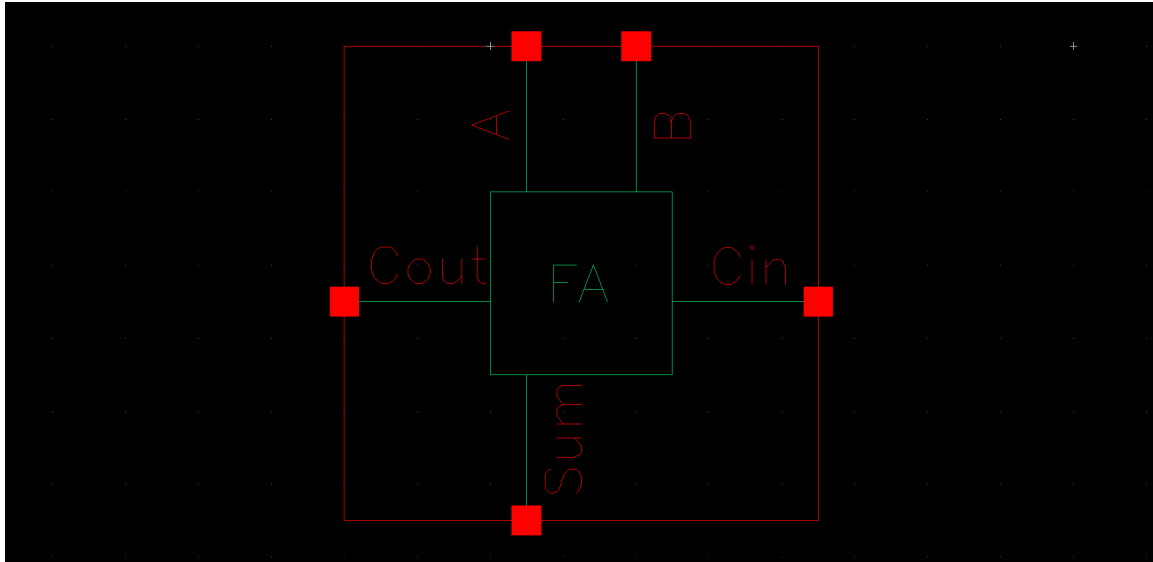
## Multiplier Architecture



## Half Adder Symbol

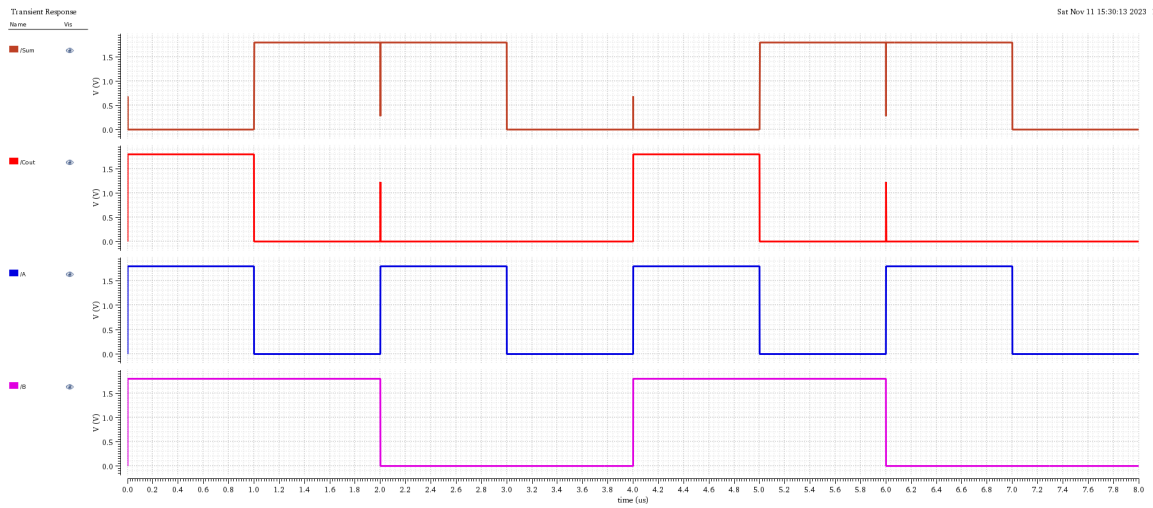


## Full Adder Symbol

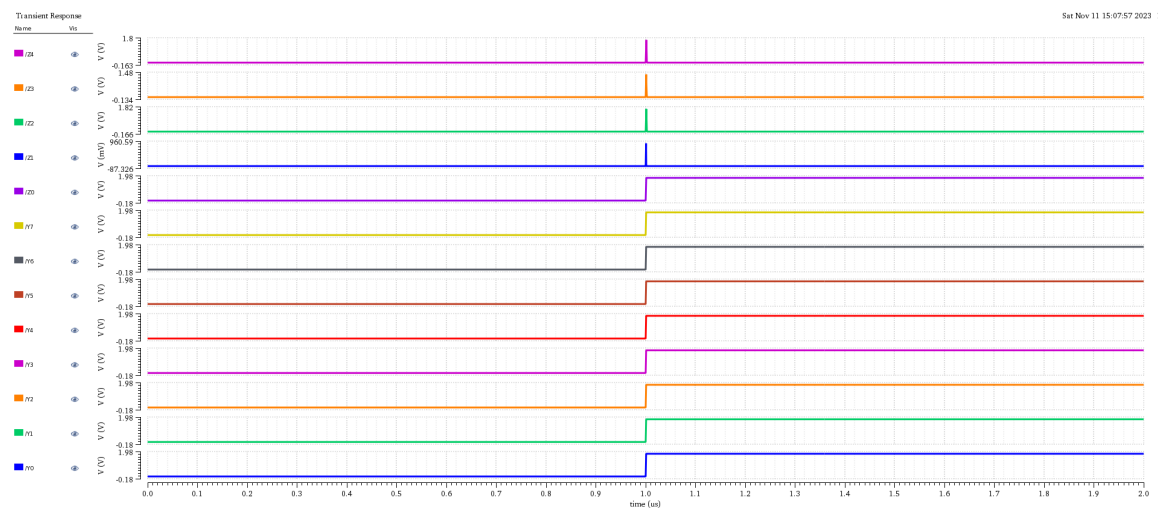


# WAVEFORMS

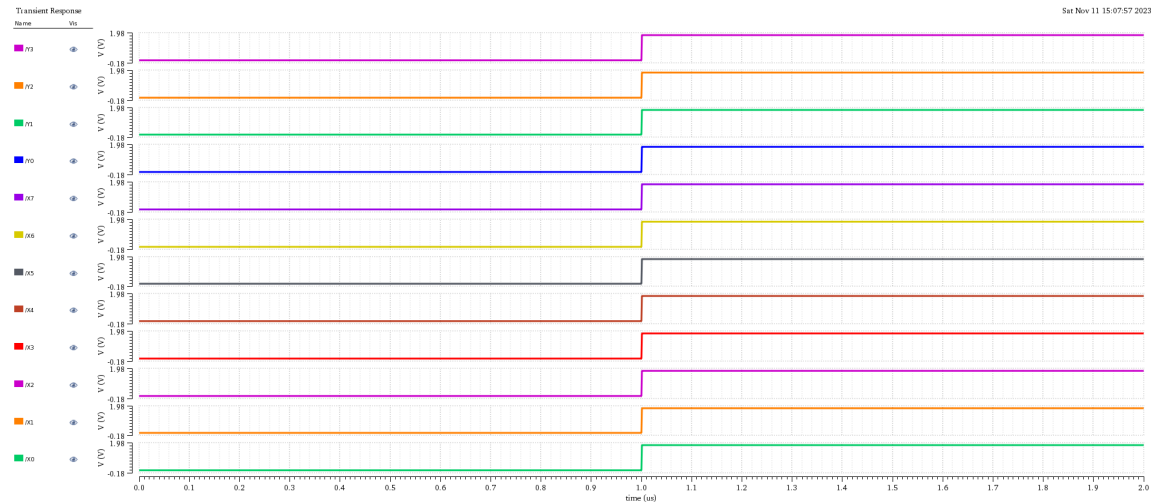
## Half Adder



## Full Adder

[illegible]





## METRICS MEASURED

### Half Adder

Delay(Sum(0->1)) = 493.5 ps

Delay(Sum(1->0)) = 26.12 ps

Delay(C<sub>out</sub>(0->1)) = 162.6 ps

Delay(C<sub>out</sub>(1->0)) = 22.02 ps

$\therefore t_{\text{sum}}(\text{HA}) = 493.5 \text{ ps}, t_{\text{carry}}(\text{HA}) = 162.6 \text{ ps}$

Average Power Dissipation = 65.86 nW

### Full Adder

Delay(Sum(0->1)) = 1.031 ns

Delay(Sum(1->0)) = 303.6 ps

Delay(C<sub>out</sub>(0->1)) = 155.7 ps

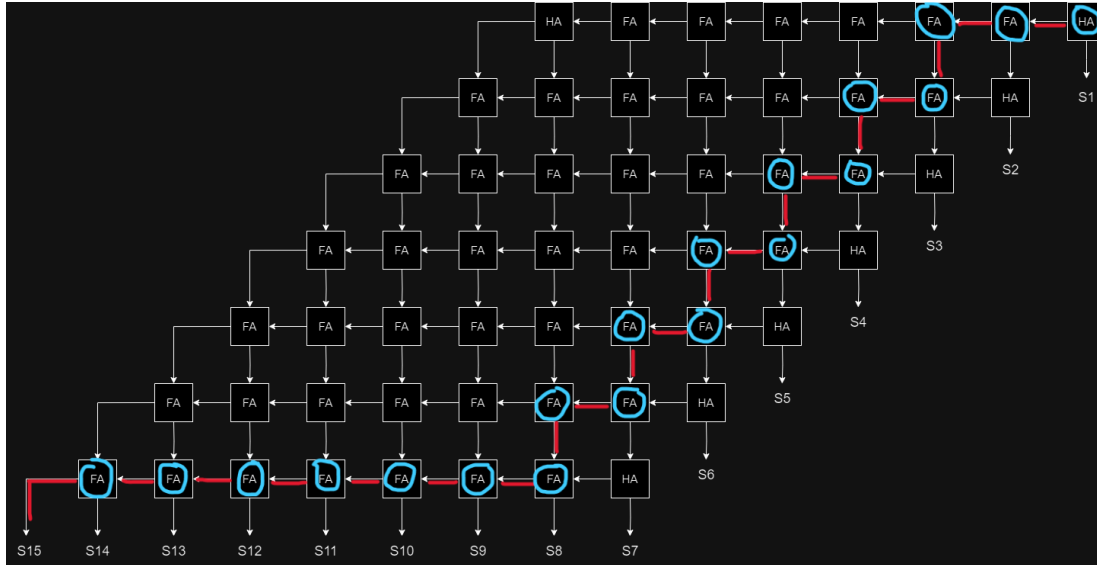
Delay(C<sub>out</sub>(1->0)) = 792.5 ps

$\therefore t_{\text{sum}}(\text{FA}) = 1.031 \text{ ns}, t_{\text{carry}}(\text{FA}) = 792.5 \text{ ps}$

Average Power Dissipation = 196.5 nW

## Array Multiplier

The critical path for finding out an theoretical estimate for the worst case delay is as follows:



$$t_{\text{Theoretical}} = t_{\text{sum}}(\text{HA}) + 13 * t_{\text{carry}}(\text{FA}) + 6 * t_{\text{sum}}(\text{FA})$$

$$= 493.5 + 13 * 792.5 + 6 * 1031 = 11.827 \text{ ns}$$

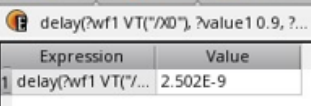
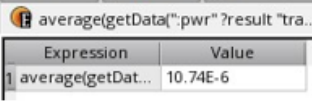
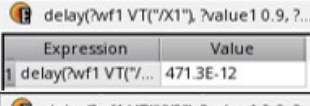
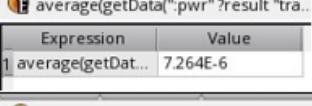


*Delay and power calculation for different input combinations:*

Sl. No.	Input X	Input Y	$t_{S14}$ (in ns)	$t_{S15}$ (in ns)	Power (in $\mu\text{W}$ )
1	11111111	10101010	2.3	2.5	10.74
2	10101010	10101010	0.47	-	7.26
3	11111111	11111111	1.66	1.2	14.66
4	10000011	11111111	6.02	6.01	9.3
5	01000011	11111111	5.49	-	9.46
6	00000000	00000000	0.52	1.5	12.02
7	10000001	11111111	6.12	6.33	5.49

Worst Case Delay: 6.33 ns

Maximum Power Dissipation: 14.66  $\mu$ W

### Cadence Images:

Sl. No.	$t_{s14}$	$t_{s15}$	Power
1			
2		-	
3			
4			
5		-	
6			
7			

Waveforms for worst case delay ( X = 1000 0001, Y = 1111 1111):

X \* Y = 1000 0000 0111 1111





## INFERENCES

1. The maximum power dissipation is  $14.66 \mu\text{W}$ , which is obtained when all the 16 input bits are 1, as the switching activity is maximum in this case.
2. The worst case propagation delay is  $6.33 \text{ ns}$ , which is obtained for  $X = 10000001$  and  $Y = 11111111$ . For this input combination, the sum and carry have to propagate through the longest path making the delay maximum.
3. The propagation delays obtained are dependent on the input combinations and there is a significant variation in the values based on the inputs. This is because the propagation delay

depends on the path that the sum and carry bits have to travel to produce the MSBs of the output.

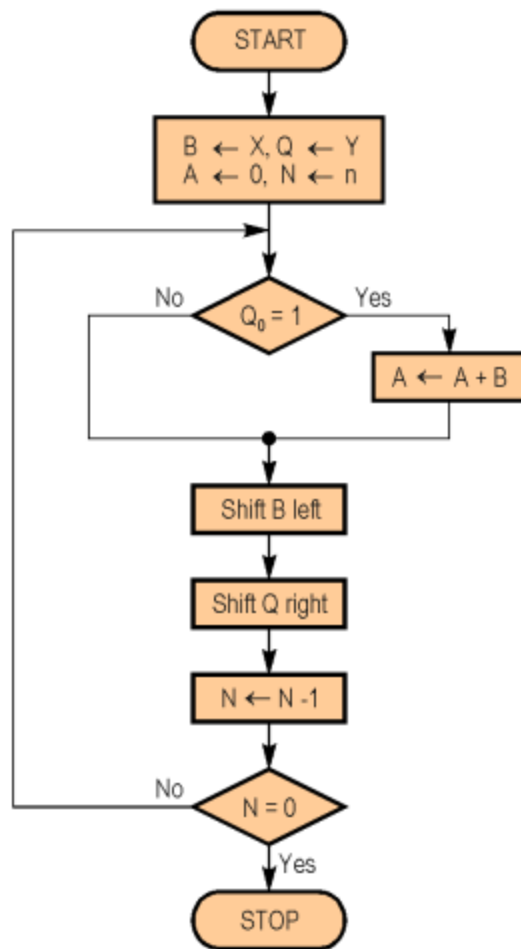
## **CONCLUSION**

In this section, we designed an 8-bit unsigned array multiplier using half adder, full adder, and other modules in Cadence. We **found out the power dissipated and propagation delays associated with different input patterns and found the input pattern associated with the worst case delay.** Also, we found out the **empirical worst case delay** of the multiplier by **tracing out the critical path not limited by input pattern dependencies.** Finally, we drew out some inferences from the results obtained from the multiplier.

## **PART - B**

### **LEFT-SHIFT MULTIPLICATION ALGORITHM**

The flowchart below shows the working of left shift multiplication algorithm:



where, A refers to the Accumulator register  
 B refers to the Multiplicand register  
 Q refers to the Multiplier register  
 N refers to the no of Multiplier bits

**Here, we must take the 'X + Y' no of bits not only for the Accumulator register, but also for the Multiplicand Register too.**

**EXAMPLE PROBLEM (Given)**



$$18 = 10010$$

$$21 = 10101$$

$$18 * 21 = ?$$

Left Shift Algorithm

Multiplicand (B) = 10010 (18)

Multiplier (Q) = 10101 (21)

Accumulator (A) = 0000000000 (Initially)

	A	$Q_4 Q_3 Q_2 Q_1$	$Q_0$	B
$N=5$	0000000000	1 0 1 0	1	0000010010
$N=4$	0000010010	1 0 1 0	1	0000010010 $A \leftarrow A+B$
	0000010010	0 1 0 1	0	0000100100 $Q \gg 1, B \ll 1$
$N=3$	0000010010	0 0 1 0	1	0001001000 $Q \gg 1, B \ll 1$
$N=2$	0001011010	0 0 1 0	1	0001001000 $A \leftarrow A+B$
	0001011010	0 0 0 1	0	0010010000 $Q \gg 1, B \ll 1$
$N=1$	0001011010	0 0 0 0	1	0100100000 $Q \gg 1, B \ll 1$
$N=0$ (stop after this!)	0101111010	0 0 0 0	1	0100100000 $A \leftarrow A+B$
	0101111010	0 0 0 0	0	1001000000



Result = 0101111010

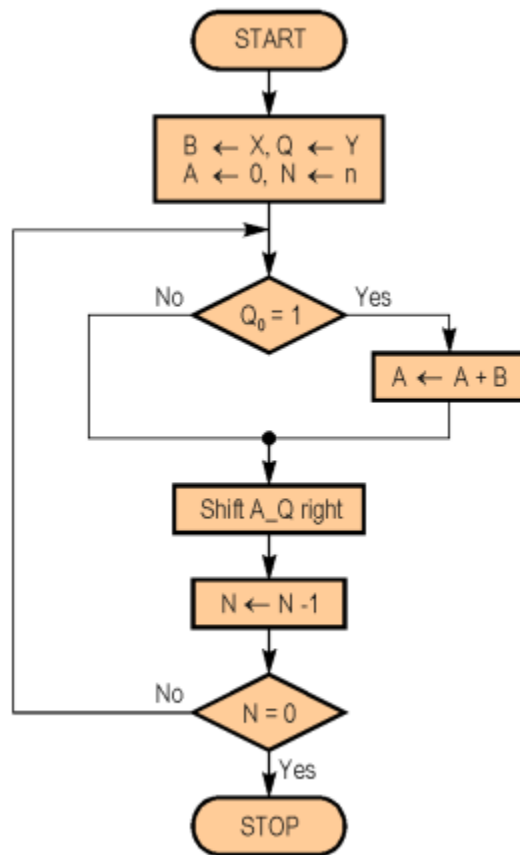
↓  
Decimal value = 378

(as expected)

## RIGHT-SHIFT MULTIPLICATION ALGORITHM



The flowchart below depicts how the right shift algorithm works:



where, A refers to the Accumulator register  
B refers to the Multiplicand register  
Q refers to the Multiplier register  
N refers to the no of Multiplier bits

In this algorithm, the **final result is stored as the combination of accumulator and multiplier registers.**

**EXAMPLE PROBLEM (Given)**

$$18 = 10010$$

$$21 = 10101$$

$$18 \times 21 = ?$$

### Right Shift Algorithm

Multiplicand (M) = 10010 (18)

Multiplier (Q) = 10101 (21)

Accumulator (A) = 00000 (Initially)

A	Q <sub>4</sub> Q <sub>3</sub> Q <sub>2</sub> Q <sub>1</sub>	Q <sub>0</sub>	M	
00000	1010	1	10010	
10010	1010	1	10010	A ← A + M } 1st cycle shift
01001	0101	0	10010	
00100	1010	1	10010	shift } 2nd cycle
10110	1010	1	10010	
01011	0101	0	10010	A ← A + M } 3rd cycle shift
00101	1010	1	10010	
10111	1010	1	10010	shift } 4th cycle
01011	1101	0	10010	
				A ← A + M } 5th cycle shift

$$\Downarrow$$

$$\text{Result} = 010111010$$

$$\downarrow$$

$$\text{Decimal Value} = 378$$

(as expected)

## CONCLUSION

In this section, we have understood the working of left-shift and right-shift methods of multiplication. Also, upon inspection it was clear that the **right shift algorithm** is **much faster** than the former **because for each addition only one right shift is required** for the right-shift method while for the left-shift method, both right-shift and left-shift operations are necessary. Also, the **right-shift method makes efficient use of the hardware by avoiding more number of bits allocated to the Accumulator and Multiplier Registers and using cascade of the two registers above for reporting the output.**