ELECTRIC VLSI PROJECT

NAME: Servet Çelik

STUDENT ID: 21947071

8 BIT MULTIPLIER & 8 BIT ALU PROJECT

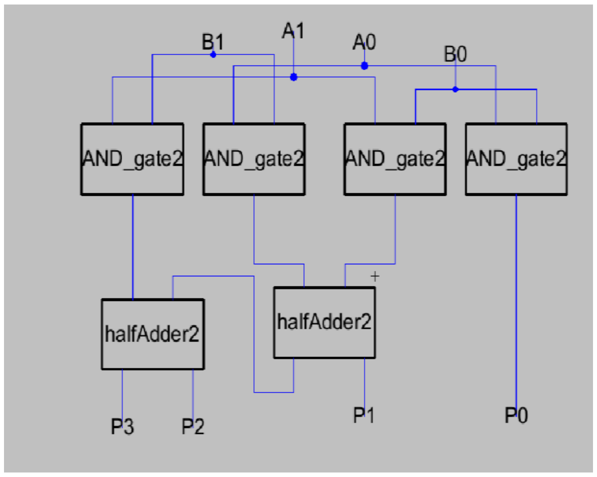
1) INTRODUCTION:

In this project, an 8-bit multiplier and an 8-bit ALU is designed. According to article of Akshata R., Prof. V.P. Gejji, Prof. B.R. Pandurangi called "ANALYSIS OF VEDIC MULTIPLIER", Vedic multipliers have simpler architecture and smaller path delay than array multipliers. That's why Vedic multiplier architecture is preferred for the 8-bit multiplier. The ALU in designed in this project handles 8 different operations. These operations are 8-bit addition, 8-bit subtraction, 4-bit multiplication, 8-bit inversion, 8-bit one digit left shift, 8-bit one-digit right shift, 8-bit two digits right shift and 8-bit buffer. The 8-bit multiplier gets two 8-bit numbers as input and gives 16-bit output, but the multiplexer in the 8-bit ALU gets an 8-bit input. So, 4-bit multiplier is implemented instead of 8-bit multiplier in the ALU.

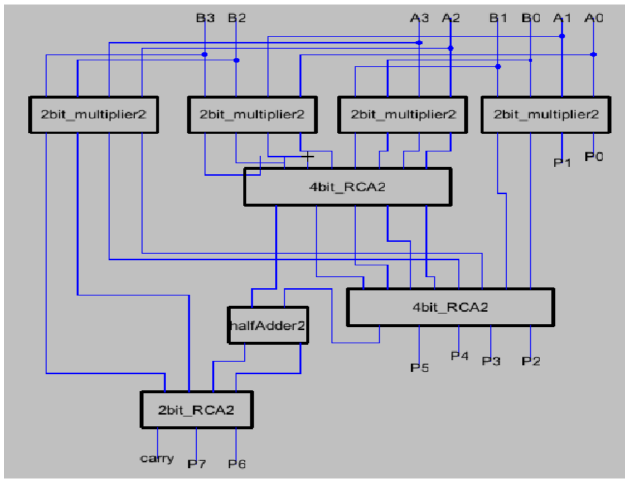
2) TRANSISTOR LEVEL DESIGN:

Ikons of each type of conventional gates were created by using NMOSs and PMOSs with proper sizing.

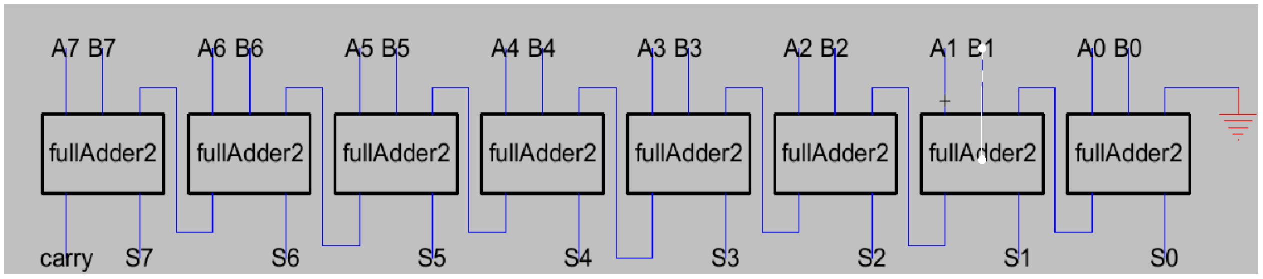
2-bit Multiplier:



4-bit Multiplier:

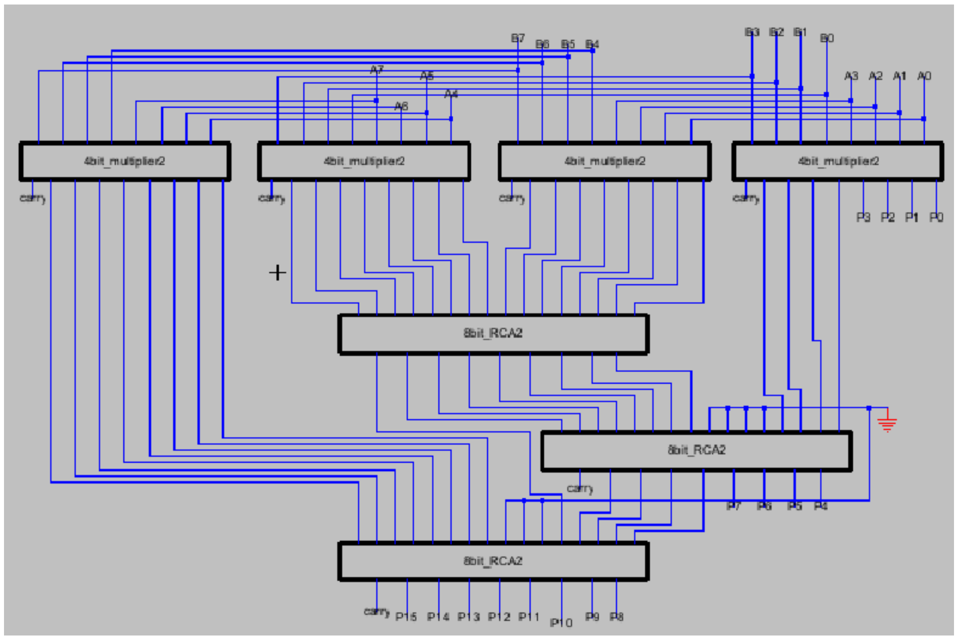


8-bit Ripple Carry Adder (RCA):

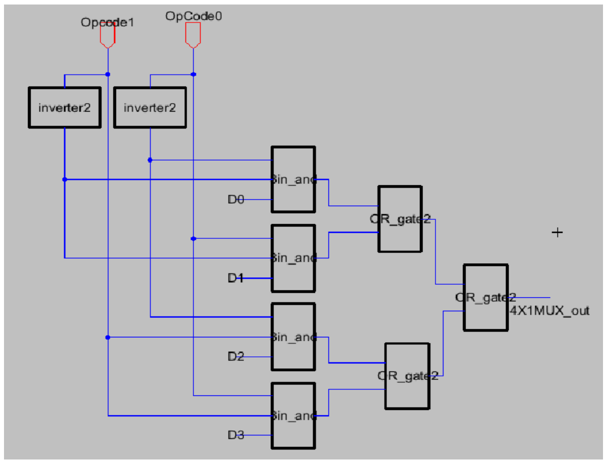


The 2-bit RCA and 4-bit RCA are not shown since they have similar architecture with 8-bit RCA.

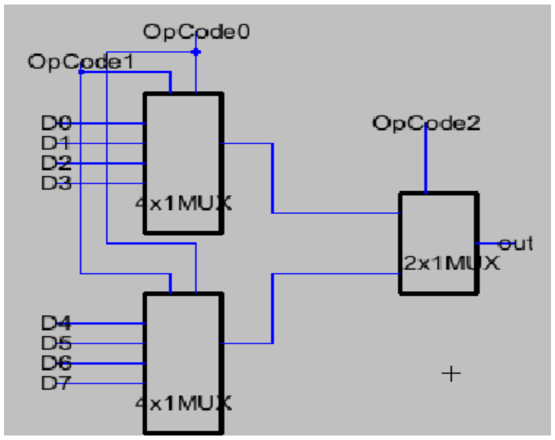
8-bit multiplier:



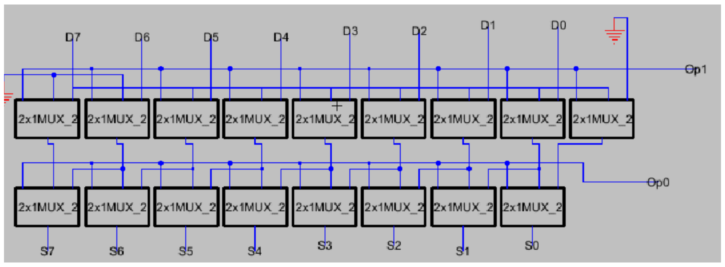
4x1 MUX:



8x1 MUX:



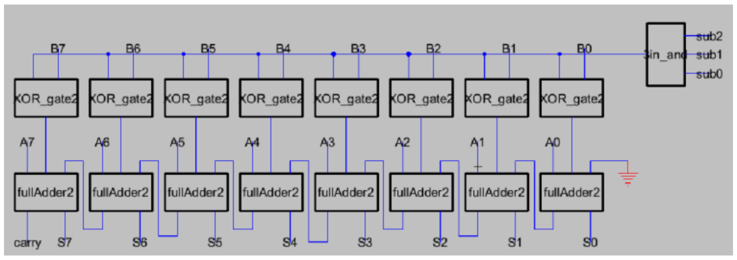
8-bit Shifter:



This shifter was designed by using 2x1 MUXs. The mode of shifting is determined by Op1 and Op0.

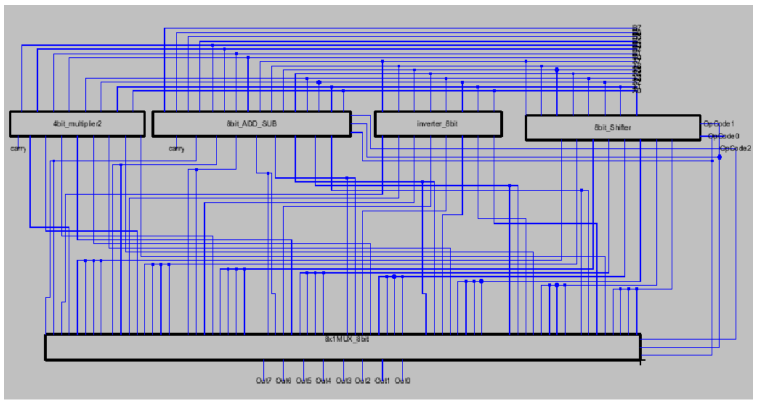
Op1	Op0	Function
0	0	1 bit left
0	1	buffer
1	0	1 bit right
1	1	2 bits right

8-bit Adder & Subtractor:



This 8-bit adder & subtractor was designed by using fulladders and XOR gates. It makes subtraction (A-B) when sub2, sub1 and sub0 are high, otherwise it makes an addition operation (A+B).Sub2, sub1 and sub0 are determined by the OpCodes in the ALU.

8-bit ALU:

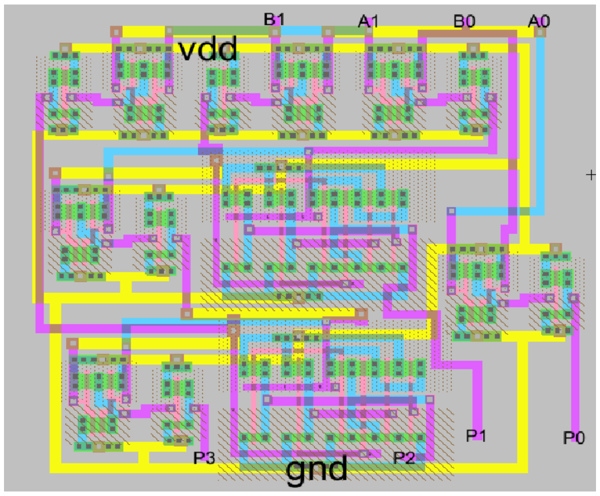


Function table of 8-bit ALU

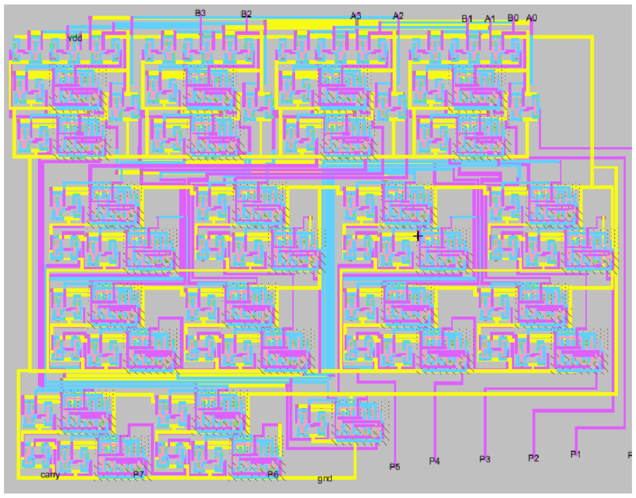
OpCode2	OpCode1	OpCode0	Function
0	0	0	1 left (A<<1)
0	0	1	Buffer (A)
0	1	0	1 right (A>>1)
0	1	1	2 right (A>>2)
1	0	0	Multiplier (A*B)
1	0	1	Inverter (~A)
1	1	0	Add (A+B)
1	1	1	Subtraction (A-B)

3) LAYOUT DESIGN:

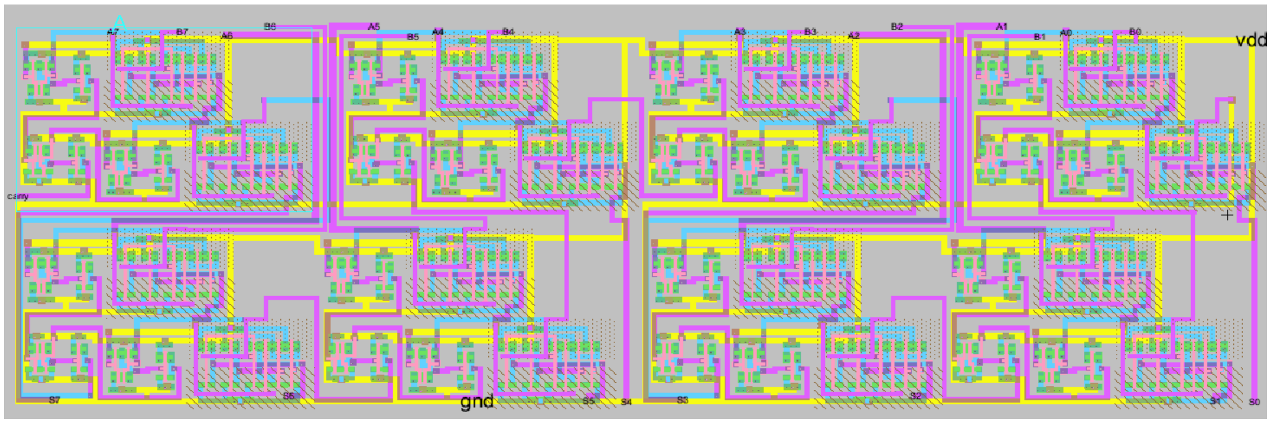
2-bit Multiplier:



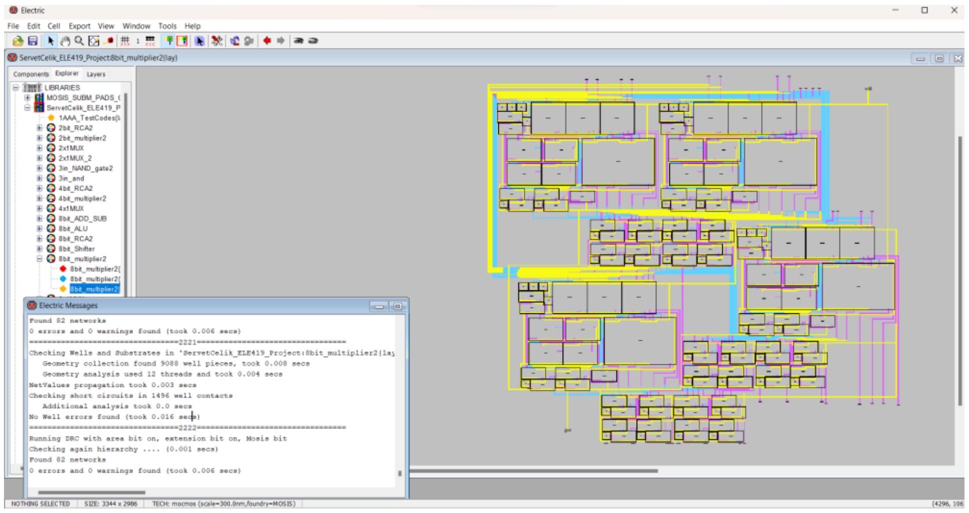
4-bit Multiplier:



8-bit Ripple Carry Adder:



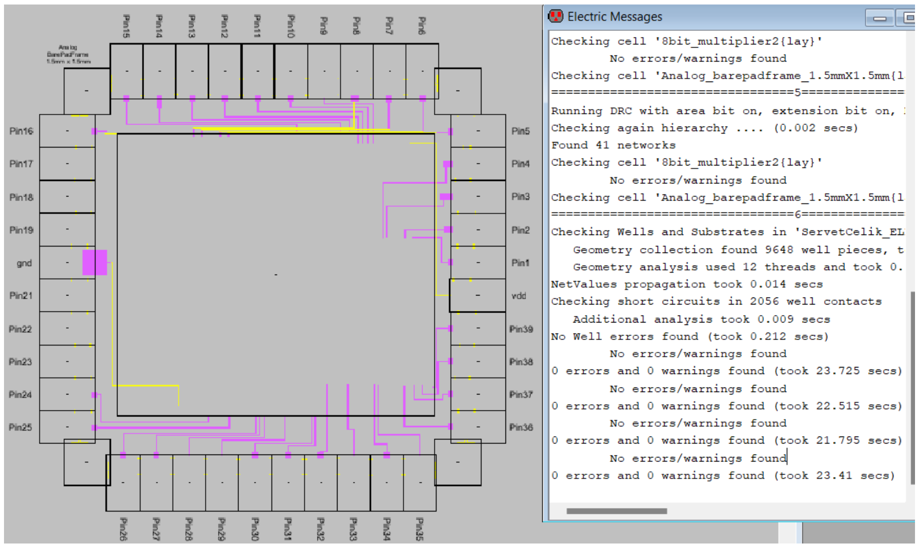
8-bit multiplier:



```
Found 82 networks
0 errors and 0 warnings found (took 0.006 secs)
=================================2221=================================
Checking Wells and Substrates in 'ServetCelik_ELE419_Project:8bit_multiplier2{la
   Geometry collection found 9088 well pieces, took 0.008 secs
   Geometry analysis used 12 threads and took 0.004 secs
NetValues propagation took 0.003 secs
Checking short circuits in 1496 well contacts
   Additional analysis took 0.0 secs
No Well errors found (took 0.016 secs)
=================================2222=================================
Running DRC with area bit on, extension bit on, Mosis bit
Checking again hierarchy .... (0.001 secs)
Found 82 networks
0 errors and 0 warnings found (took 0.006 secs)
```

This 8-bit multiplier is created by using four 4-bit multipliers and three 8-bit RCAs. In the picture, I could not go down to transistor level because of my computer quality. It shows just green blocks when I go down to first level.

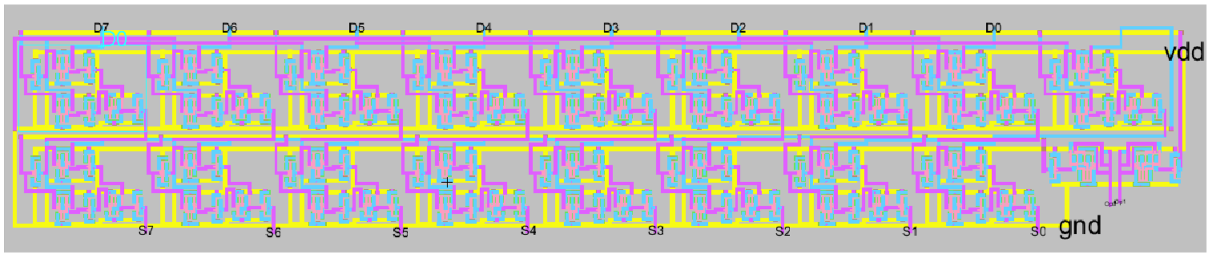
Pin connections of 8-bit multiplier:



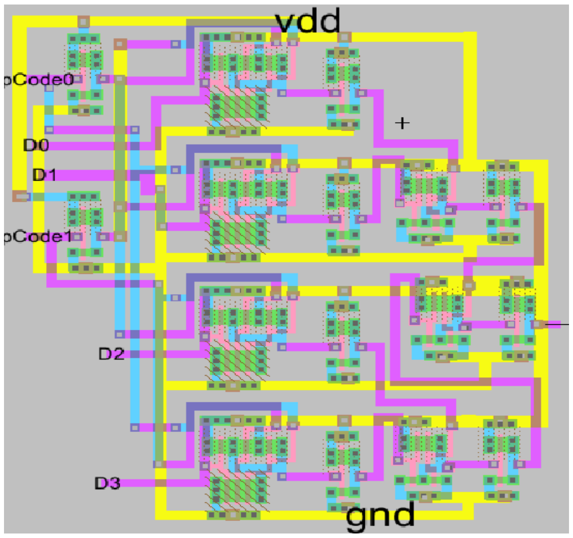
```
Checking cell '8bit_multiplier2{lay}'
        No errors/warnings found
Checking cell 'Analog_barepadframe_1.5mmX1.5mm{l
=====================================5===============
Running DRC with area bit on, extension bit on, 
Checking again hierarchy .... (0.002 secs)
Found 41 networks
Checking cell '8bit_multiplier2{lay}'
        No errors/warnings found
Checking cell 'Analog_barepadframe_1.5mmX1.5mm{l
=====================================6===============
Checking Wells and Substrates in 'ServetCelik_EL
   Geometry collection found 9648 well pieces, t
   Geometry analysis used 12 threads and took 0.
NetValues propagation took 0.014 secs
Checking short circuits in 2056 well contacts
   Additional analysis took 0.009 secs
No Well errors found (took 0.212 secs)
        No errors/warnings found
0 errors and 0 warnings found (took 23.725 secs)
        No errors/warnings found
0 errors and 0 warnings found (took 22.515 secs)
        No errors/warnings found
0 errors and 0 warnings found (took 21.795 secs)
        No errors/warnings found
0 errors and 0 warnings found (took 23.41 secs)
```

vdd	vdd	**gnd**	gnd
A7	Pin8	**Out15**	Pin24
A6	Pin7	**Out14**	Pin25
A5	Pin6	**Out13**	Pin26
A4	Pin5	**Out12**	Pin27
A3	Pin4	**Out11**	Pin28
A2	Pin3	**Out10**	Pin29
A1	Pin2	**Out9**	Pin30
A0	Pin1	**Out8**	Pin31
B7	Pin16	**Out7**	Pin32
B6	Pin15	**Out6**	Pin33
B5	Pin14	**Out5**	Pin34
B4	Pin13	**Out4**	Pin35
B3	Pin12	**Out3**	Pin36
B2	Pin11	**Out2**	Pin37
B1	Pin10	**Out1**	Pin38
B0	Pin9	**Out0**	Pin39

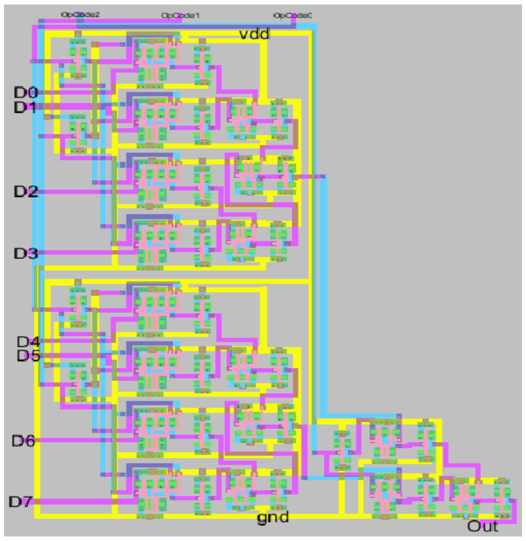
8-bit Shifter:



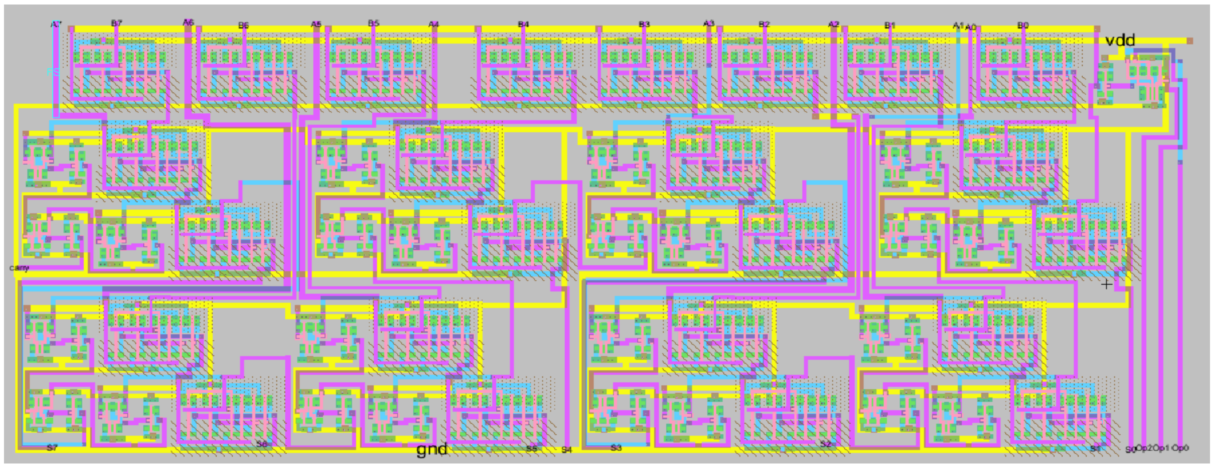
4x1 MUX:



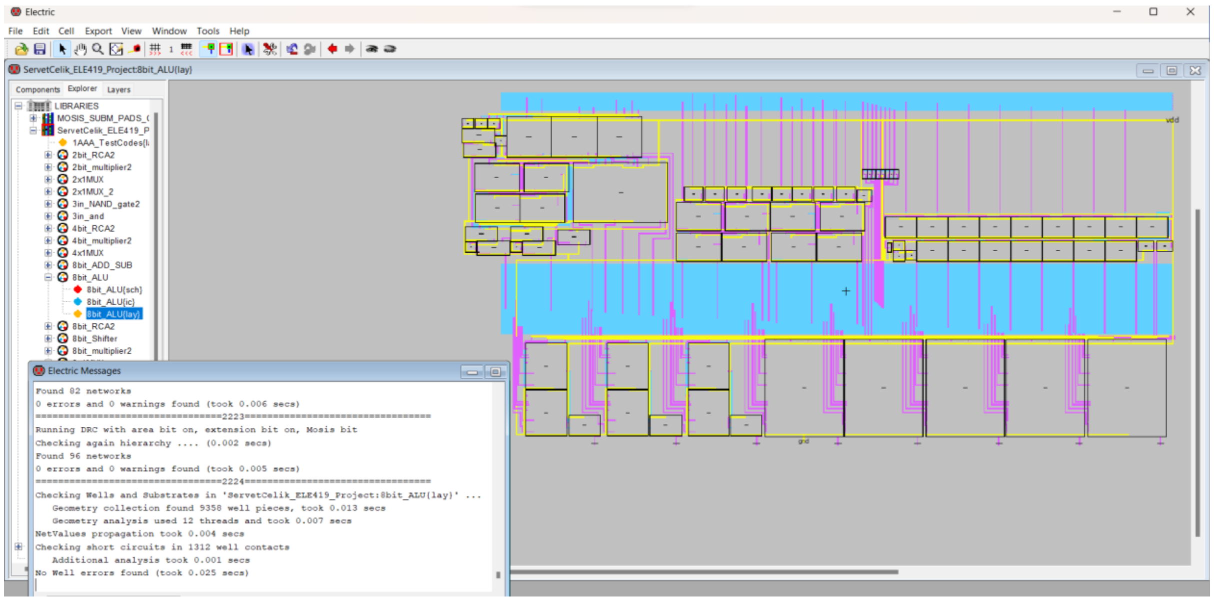
8x1 MUX:



8-bit Adder & Subtractor:



8-bit ALU:



```
Found 82 networks
0 errors and 0 warnings found (took 0.006 secs)
=================================2223=================================
Running DRC with area bit on, extension bit on, Mosis bit
Checking again hierarchy .... (0.002 secs)
Found 96 networks
0 errors and 0 warnings found (took 0.005 secs)
=================================2224=================================
Checking Wells and Substrates in 'ServetCelik_ELE419_Project:8bit_ALU(lay)' ...
   Geometry collection found 9358 well pieces, took 0.013 secs
   Geometry analysis used 12 threads and took 0.007 secs
NetValues propagation took 0.004 secs
Checking short circuits in 1312 well contacts
   Additional analysis took 0.001 secs
No Well errors found (took 0.025 secs)
```

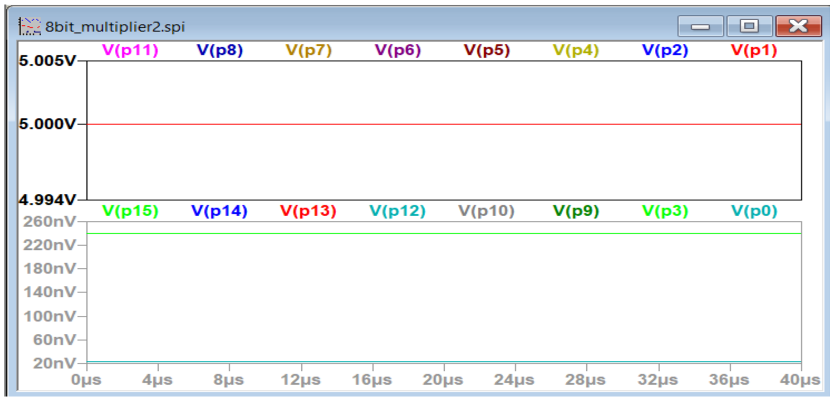
4) SIMULATION RESULTS:

- In the simulation part, the outputs are separated as 1s and 0s to have a better visualization.

a. Multiplication:

i. A=00001111, B=10101010

```
vdd vdd 0 DC 5
VinA7 A7 0 DC 0
VinA6 A6 0 DC 0
VinA5 A5 0 DC 0
VinA4 A4 0 DC 0
VinA3 A3 0 DC 5
VinA2 A2 0 DC 5
VinA1 A1 0 DC 5
VinA0 A0 0 DC 5
VinB7 B7 0 DC 5
VinB6 B6 0 DC 0
VinB5 B5 0 DC 5
VinB4 B4 0 DC 0
VinB3 B3 0 DC 5
VinB2 B2 0 DC 0
VinB1 B1 0 DC 5
VinB0 B0 0 DC 0
.tran 10u 40u
```

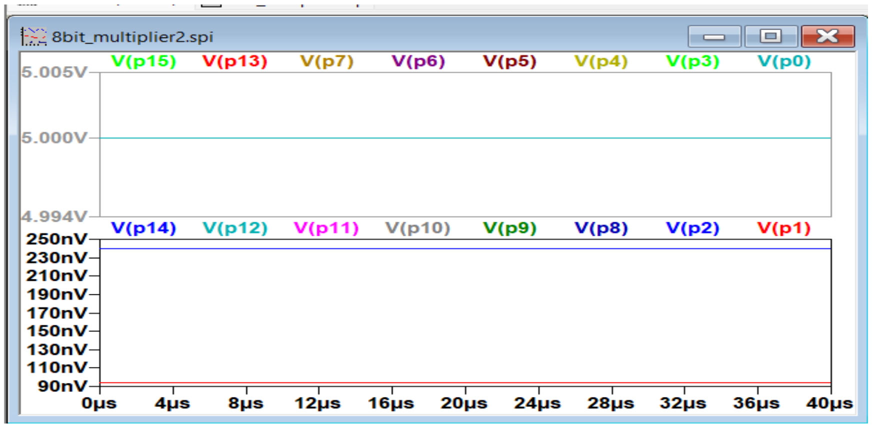


The output is separated as 1s and 0s. So, the above output shows that: A * B = 0000 1001 1111 0110

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	0

ii. A=11001011, B=11001011

```
vdd vdd 0 DC 5
VinA7 A7 0 DC 5
VinA6 A6 0 DC 5
VinA5 A5 0 DC 0
VinA4 A4 0 DC 0
VinA3 A3 0 DC 5
VinA2 A2 0 DC 0
VinA1 A1 0 DC 5
VinA0 A0 0 DC 5
VinB7 B7 0 DC 5
VinB6 B6 0 DC 5
VinB5 B5 0 DC 0
VinB4 B4 0 DC 0
VinB3 B3 0 DC 5
VinB2 B2 0 DC 0
VinB1 B1 0 DC 5
VinB0 B0 0 DC 5
.tran 10u 40u
```

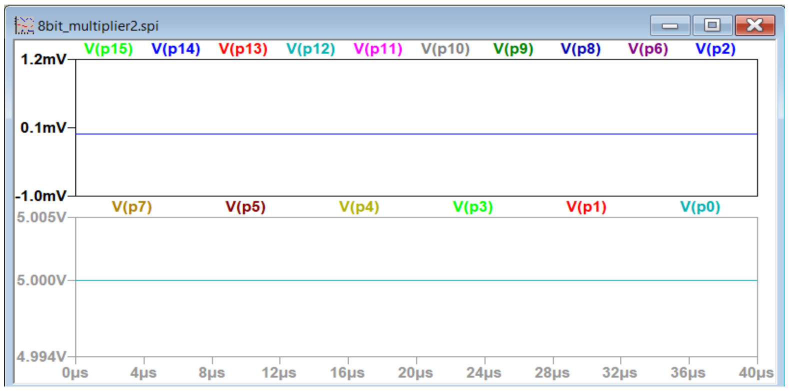


The output is separated as 1s and 0s. So, the above output shows that: A * B = 1010 0000 1111 1001.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	0	1	1	1	1	1	0	0	1

iii. A=10111011, B=00000001

```
vdd vdd 0 DC 5
VinA7 A7 0 DC 5
VinA6 A6 0 DC 0
VinA5 A5 0 DC 5
VinA4 A4 0 DC 5
VinA3 A3 0 DC 5
VinA2 A2 0 DC 0
VinA1 A1 0 DC 5
VinA0 A0 0 DC 5
VinB7 B7 0 DC 0
VinB6 B6 0 DC 0
VinB5 B5 0 DC 0
VinB4 B4 0 DC 0
VinB3 B3 0 DC 0
VinB2 B2 0 DC 0
VinB1 B1 0 DC 0
VinB0 B0 0 DC 5
.tran 10u 40u
```



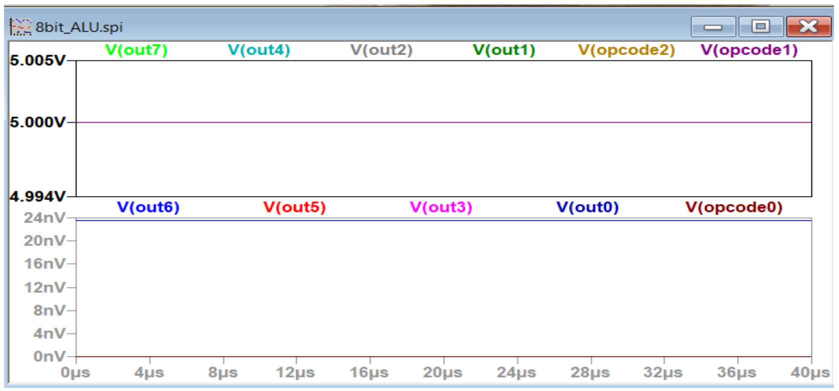
The output is separated as 1s and 0s. So, the above output shows that: A * B = 0000 1001 1111 0110.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	1	1	1	0	1	1

b. ALU:

i. Sum of A=11001011, B=11001011 (A+B)

```
vdd vdd 0 DC 5
VinOpCode0 OpCode0 0 DC 0
VinOpCode1 OpCode1 0 DC 5
VinOpCode2 OpCode2 0 DC 5
VinA7 A7 0 DC 5
VinA6 A6 0 DC 5
VinA5 A5 0 DC 0
VinA4 A4 0 DC 0
VinA3 A3 0 DC 5
VinA2 A2 0 DC 0
VinA1 A1 0 DC 5
VinA0 A0 0 DC 5
VinB7 B7 0 DC 5
VinB6 B6 0 DC 5
VinB5 B5 0 DC 0
VinB4 B4 0 DC 0
VinB3 B3 0 DC 5
VinB2 B2 0 DC 0
VinB1 B1 0 DC 5
VinB0 B0 0 DC 5
.tran 10u 40u
```

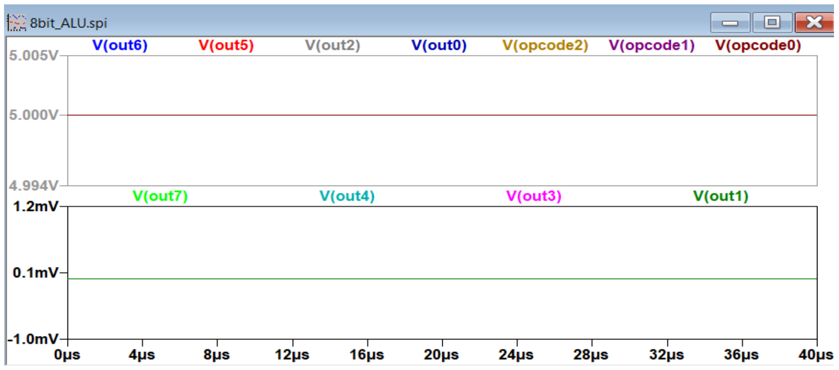


In this spice code, OpCodes are determined by using the function table of ALU. The function code of summation is 110. This summation creates an overflow, but I could not show the overflow flag since I used an 8-bit multiplexer in the ALU. So, the output is shown as: A+B = 1001 0110 instead of A+B = 1 1001 0110.

Digits								**OpCodes**		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Op2**	**Op1**	**Op0**
1	0	0	1	0	1	1	0	1	1	0

ii. Subtraction of A=00001111, B=10101010 (A-B)

```
vdd vdd 0 DC 5
VinOpCode0 OpCode0 0 DC 5
VinOpCode1 OpCode1 0 DC 5
VinOpCode2 OpCode2 0 DC 5
VinA7 A7 0 DC 0
VinA6 A6 0 DC 0
VinA5 A5 0 DC 0
VinA4 A4 0 DC 0
VinA3 A3 0 DC 5
VinA2 A2 0 DC 5
VinA1 A1 0 DC 5
VinA0 A0 0 DC 5
VinB7 B7 0 DC 5
VinB6 B6 0 DC 0
VinB5 B5 0 DC 5
VinB4 B4 0 DC 0
VinB3 B3 0 DC 5
VinB2 B2 0 DC 0
VinB1 B1 0 DC 5
VinB0 B0 0 DC 0
.tran 10u 40u
```

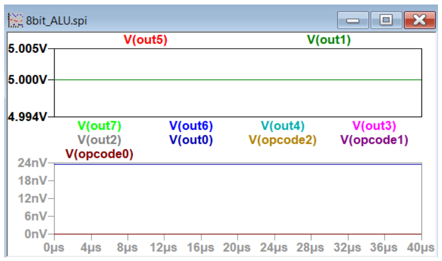


In this spice code, OpCodes are determined by using the function table of ALU. The function code of subtraction is 111. This subtraction gives a negative output, but it is not shown in the output since sign bit wasn't used in the operations. So, the output shown as the 2's complement of the number. So, the output is shown as : A-B = 0110 0101 instead of A-B = -1001 1011.

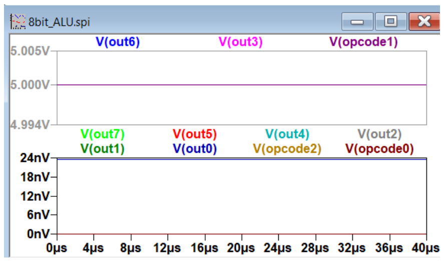
Output digits								**OpCodes**		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Op2**	**Op1**	**Op0**
0	1	1	0	0	1	0	1	1	1	1

iii. 1 bit shifting of A=10010001

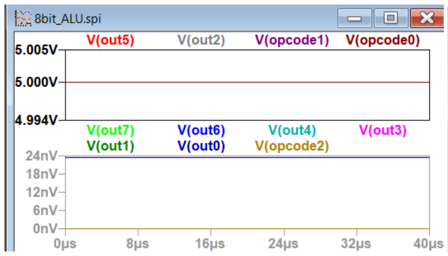
```
vdd vdd 0 DC 5
VinOpCode0 OpCode0 0 DC 5
VinOpCode1 OpCode1 0 DC 0
VinOpCode2 OpCode2 0 DC 0
VinA7 A7 0 DC 5
VinA6 A6 0 DC 0
VinA5 A5 0 DC 0
VinA4 A4 0 DC 5
VinA3 A3 0 DC 0
VinA2 A2 0 DC 0
VinA1 A1 0 DC 0
VinA0 A0 0 DC 5
.tran 10u 40u
```



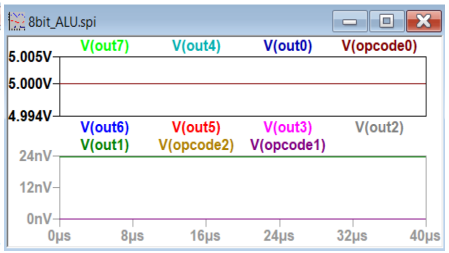
1-bit left shifting of A



1-bit right shifting of A



2-bit right shifting of A

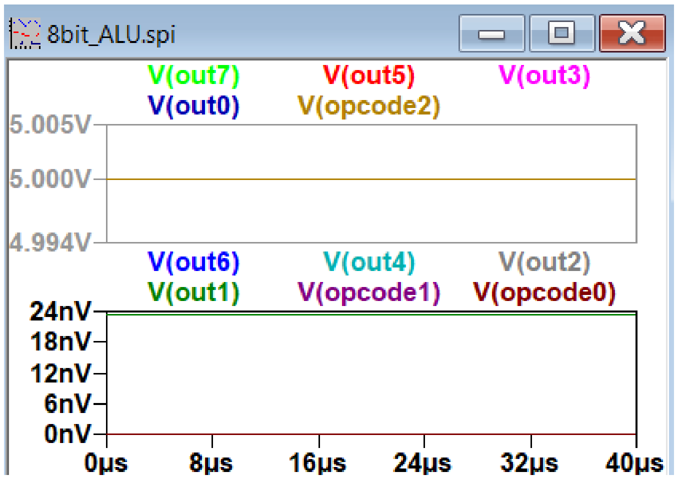


Buffered A

- For all the operations above same spice code is used by changing only the Opcodes, and the opcodes are determined by using function table of the ALU. Opcodes are 000 for 1-bit left shifting, 001 for buffer, 010 1-bit right shifting and 011 for 2-bit right shifting operations.

iv. 4-bit multiplication of A=1101, B=1101 (A*B)

```
vdd vdd 0 DC 5
VinOpCode0 OpCode0 0 DC 0
VinOpCode1 OpCode1 0 DC 0
VinOpCode2 OpCode2 0 DC 5
VinA3 A3 0 DC 5
VinA2 A2 0 DC 5
VinA1 A1 0 DC 0
VinA0 A0 0 DC 5
VinB3 B3 0 DC 5
VinB2 B2 0 DC 5
VinB1 B1 0 DC 0
VinB0 B0 0 DC 5
.tran 10u 40u
```

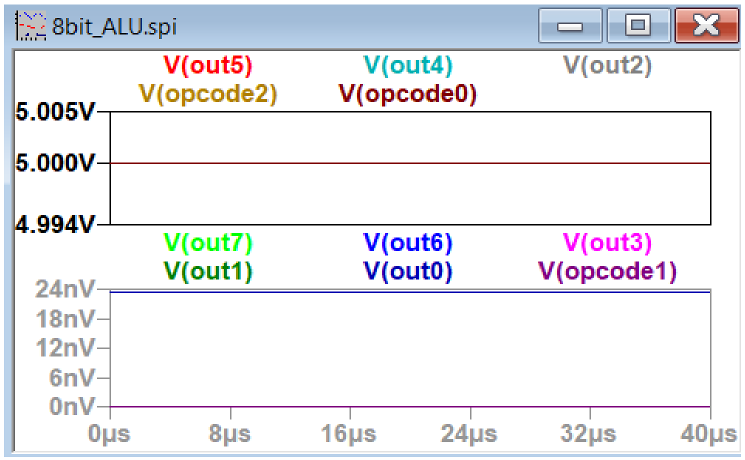


4-bit multiplier is implemented in the ALU instead of 8- bit multiplier since the output of the ALU is 8 bits. The above outputs show that: A * B = 10101001

Output digits								**OpCodes**		
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**	**Op2**	**Op1**	**Op0**
1	0	1	0	1	0	0	1	1	0	0

v. 8-bit inversion of A=1100 1011 (~A)

```
vdd vdd 0 DC 5
VinOpCode0 OpCode0 0 DC 5
VinOpCode1 OpCode1 0 DC 0
VinOpCode2 OpCode2 0 DC 5
VinA7 A7 0 DC 5
VinA6 A6 0 DC 5
VinA5 A5 0 DC 0
VinA4 A4 0 DC 0
VinA3 A3 0 DC 5
VinA2 A2 0 DC 0
VinA1 A1 0 DC 5
VinA0 A0 0 DC 5
.tran 10u 40u
```



5) NOVELTY & PROS-CONS:

a. **Novelty:** In the multiplier part, I used Vedic multiplier architecture. Before this design, I tried array multiplier, and I saw that it was lower than this architecture. In the ALU part, I designed a shifter with 2x1 multiplexers. That design allows me to use same design for buffer, 2-bit right shifting, and 1-bit right and left shifting by just changing the OpCodes. This reduces the number of transistors and power consumption in the design.

b. **Table of Pros-Cons:**

Pros	Cons
High speed: this design is faster than array multiplier.	**Transistor technology**: An old transistor technology is used in this project.
Multiplier size: The size of multiplier design can be considered small since it fits into the frame and even leaves space.	**ALU size**: ALU size couldn't be reduced enough because of time constraints.
Simplicity and systematicity: This design has simple and systematic structure. So, it can be edited and developed easily by someone else.	**Transistor Technology:** An old transistor technology is used in the project.

c. **Functionality:** It can be implemented to a clocked system with some additional components and a proper clock distribution. The current drawn from 5V power supply is about 8.86nA. So, power consumption of the circuit can be considered low. This design is efficient in terms of power consumption and speed since the number of transistors and size of circuit is reduced to minimum level, but this efficiency can be increased if the number of transistors could be further reduced by using different kind of methodologies such as sequential logic and transmission gates.

6) REFERENCES:

[1] R. Jacob Baker, " CMOSedu.com", CMOSedu. [Online]. Available: https://cmosedu.com/. [Accessed: Jan 17, 2024].

[2] Akshata R., Prof. V.P. Gejji, Prof. B.R. Pandurangi, " ANALYSIS OF VEDIC MULTIPLIER ", in Proceedings of the International Conference on computing, 2016