National Institute of Technology Karnataka, Surathkal

EC302
VLSI Design Lab Project

8-Bit Arithmetic Logic Unit

Project Report

Authors
Rahul M Hanchate – 181EC135
Supritha Harishankar – 181EC148

Supervisors
Dr. Ramesh Kini M.
Mrs. Kalpana Bhat

26 December 2020

Contents

Introduction

This project deals with the designing and implementation of an 8-Bit Arithmetic Logic Unit. This arithmetic unit is capable of performing three separate operations, namely multiplication, addition and subtraction. The layout for the Arithmetic Unit was created using the MAGIC software. The layout was then tested using the switch level logic simulator IRSIM.

The designed unit can take in two 8-bit numbers as the input, and the output is defined by the following operations

sel	cin	Operation
0	X	$a \times b$
1	0	$a + b$
1	1	$a - b$

Table 1: Operations

The standard cells that were used in the making of the unit were imported from the vsclib library package. The pharosc technology file was used during the implementation borrowed from Graham Petley's website (www.vlsitechnology.org). A scmos parameters file was also obtained. The testing of the design was performed with the assistance of a python script that generates the test-vectors, verifies the answer and prints the results, with the failed cases (if any).

Design Structure and Implementation Flow

- The structure of the 8-bit arithmetic logic unit has been designed to perform addition, subtraction and multiplication on 8-bit numbers.
- The unit can take in two 8-bit numbers.
- The unit also takes in two inputs, select line “sel” and carry in / control line “cin” for the selection of operation.
- An 8-bit multiplier has been designed using seven 8-bit ripple full adders, each providing eight sum outputs and one carry-out / borrow output using two 8-bit inputs, and AND gates. The individual “cin’s of FAs are grounded.
- An 8-bit ripple full adder has been designed using 8 full adders, each providing one sum output and one carry out output using two 1-bit inputs and one carry-in input.
- An 8-bit adder/subtractor has been designed using one 8-bit ripple full adder, and XOR gates.
- Sixteen 2:1 multiplexers were used to select outputs between multiplier and adder/subtractor units.

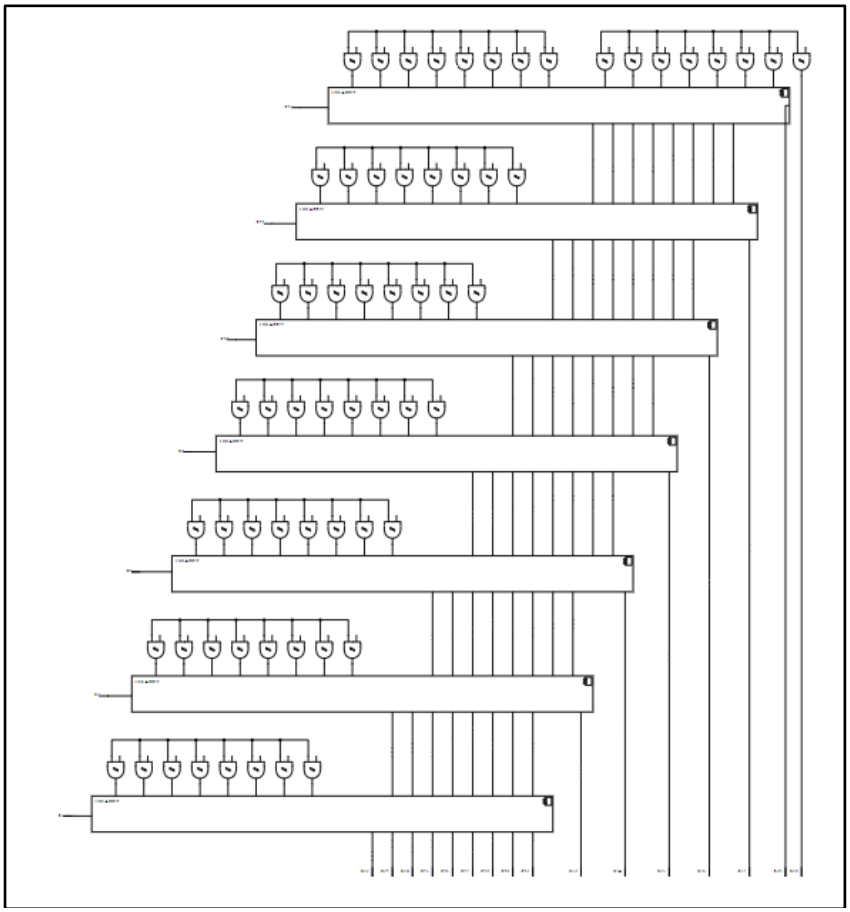

Figure 1: Circuit Schematic of 8-Bit Multiplier

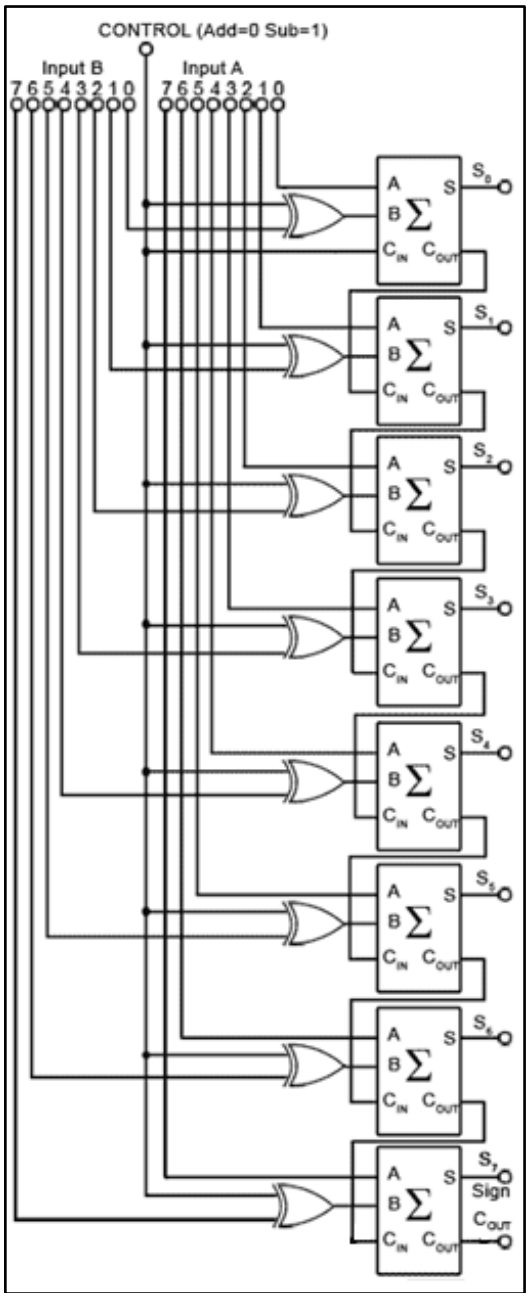

Figure 2: Circuit Schematic of 8-Bit Adder / Subtractor

- The gate-level circuit was finalized and the layout designed on MAGIC Layout Editor. A sim file was extracted and used for simulation on IRSIM using Python.
- Testing of cases was done using Python script on IRSIM Simulator. As the time required for the simulation of all of the test cases takes much time, random cases were selected and tested for accuracy.
- Results were obtained by reducing delay, dynamic power consumption and area occupied.

Cells Used

The following gates were used for designing the 8-Bit Adder / Subtractor and 8-Bit Multiplier.

AND Gate

A two-input gate that is high when both of its inputs are high.

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: Truth Table of AND gate

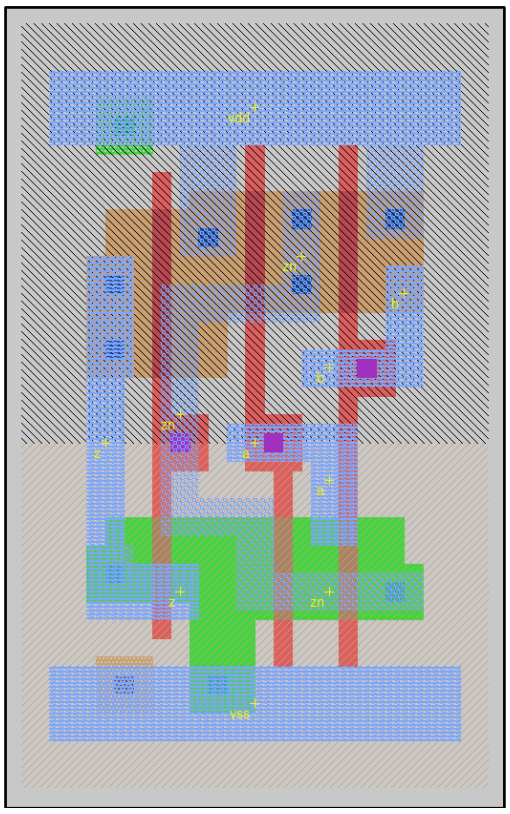

Figure 3: Layout of AND gate

OR Gate

A two-input gate that is high when any one of its inputs are high.

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

Table 3: Truth Table of OR gate

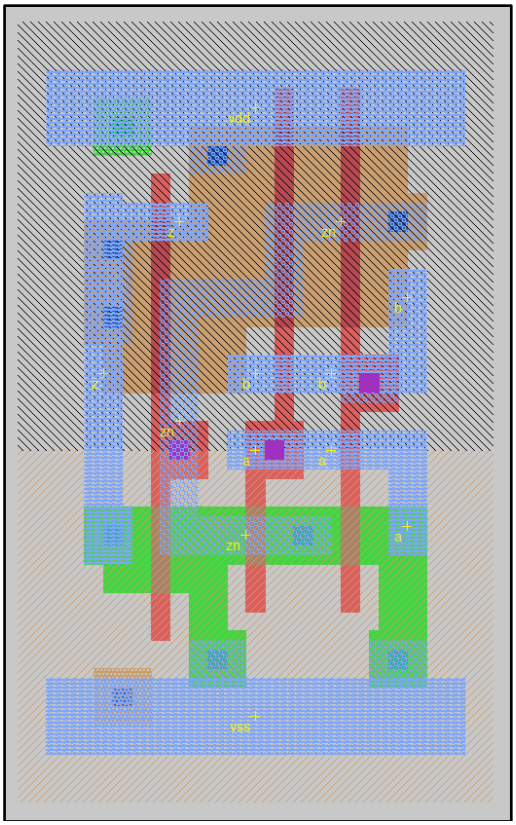

Figure 4: Layout of OR gate

XOR Gate

A two-input gate that is high when any either one of its inputs is high.

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 4: Truth Table of XOR gate

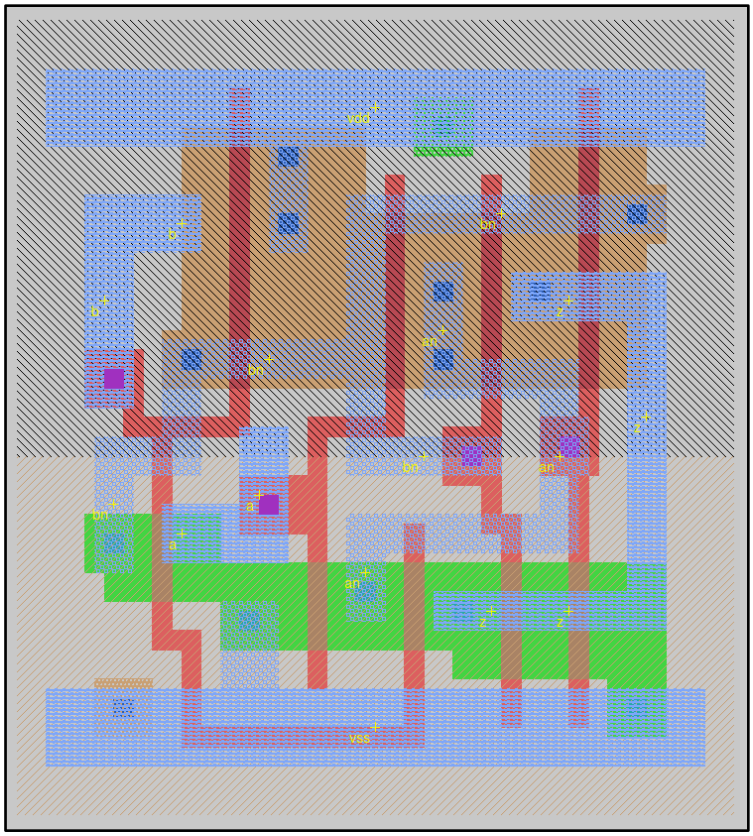

Figure 5: Layout of XOR gate

Half Adder Cell

A Half adder cell from the vsclib has been used for implementing a Full Adder.

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 5: Truth Table of Half Adder Cell

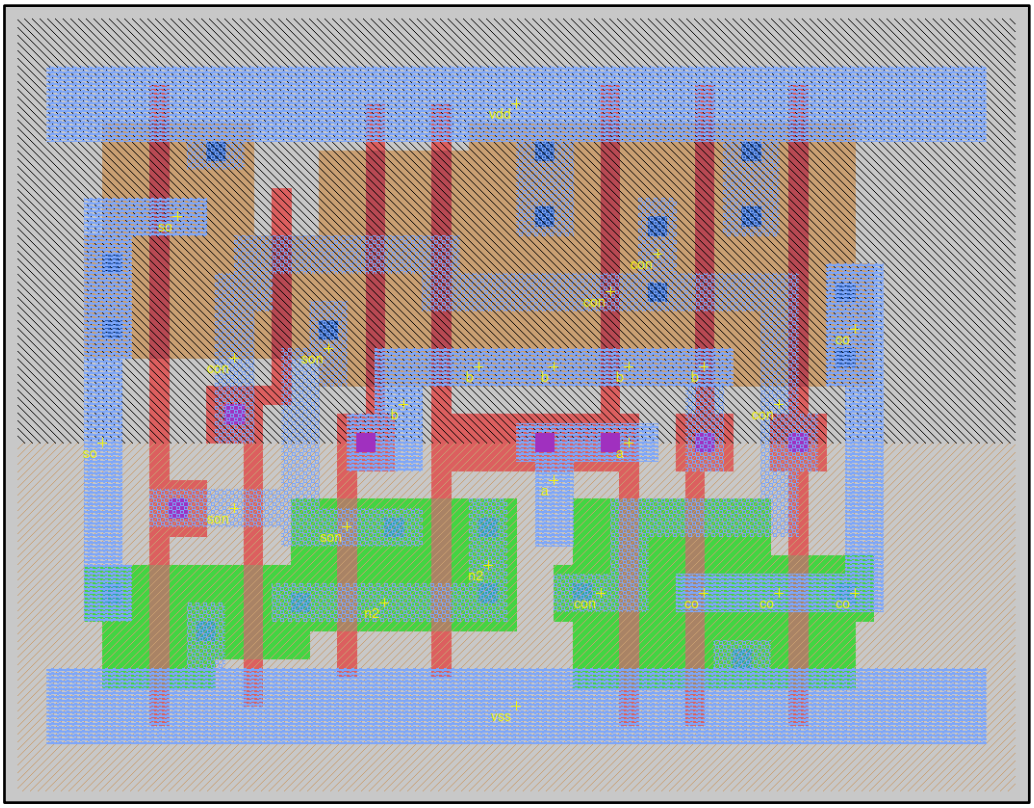

Figure 6: Layout of Half Adder Cell

Multiplexer Cell

A 2:1 Multiplexer selects between one input or the other depending upon the input select line.

A	B	Select	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Table 6: Truth Table of 2:1 Multiplexer

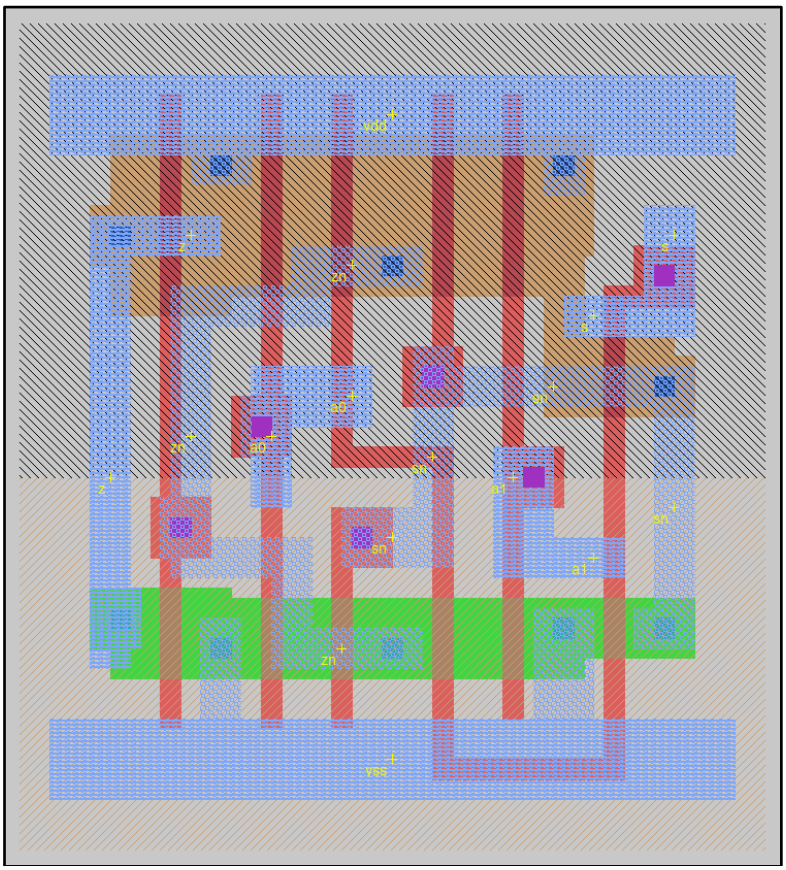

Figure 7: Layout of 2:1 Multiplexer

Modular Approach

Using the cells mentioned before, multiple cells were bought together and connected. The main modules were constructed to form the target design. The modules are discussed below.

Full Adder

A full adder has been implemented using two half adder cells and one OR gate cell. The inputs A and B are fed to the first half adder. The sum of the first half adder along with the carry-in is fed as inputs to the second half adder. The sum of the second half adder is the output of the full adder. The carry-out is generated from the OR gate using carry-out generated from both half adders.

A	B	Carry-In	Sum	Carry-Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 7: Truth Table of Full Adder cell

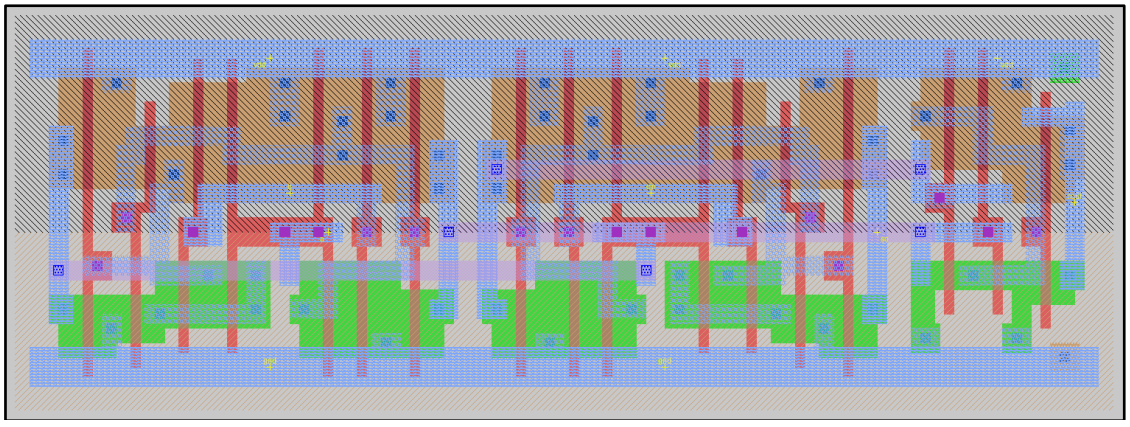

Figure 8: Layout of Full Adder cell

Continuing on with the full adder designed, eight such layouts were stacked together to form an 8-bit ripple full adder. The circuit schematic of an 8-bit ripple full adder is depicted in Figure 2.

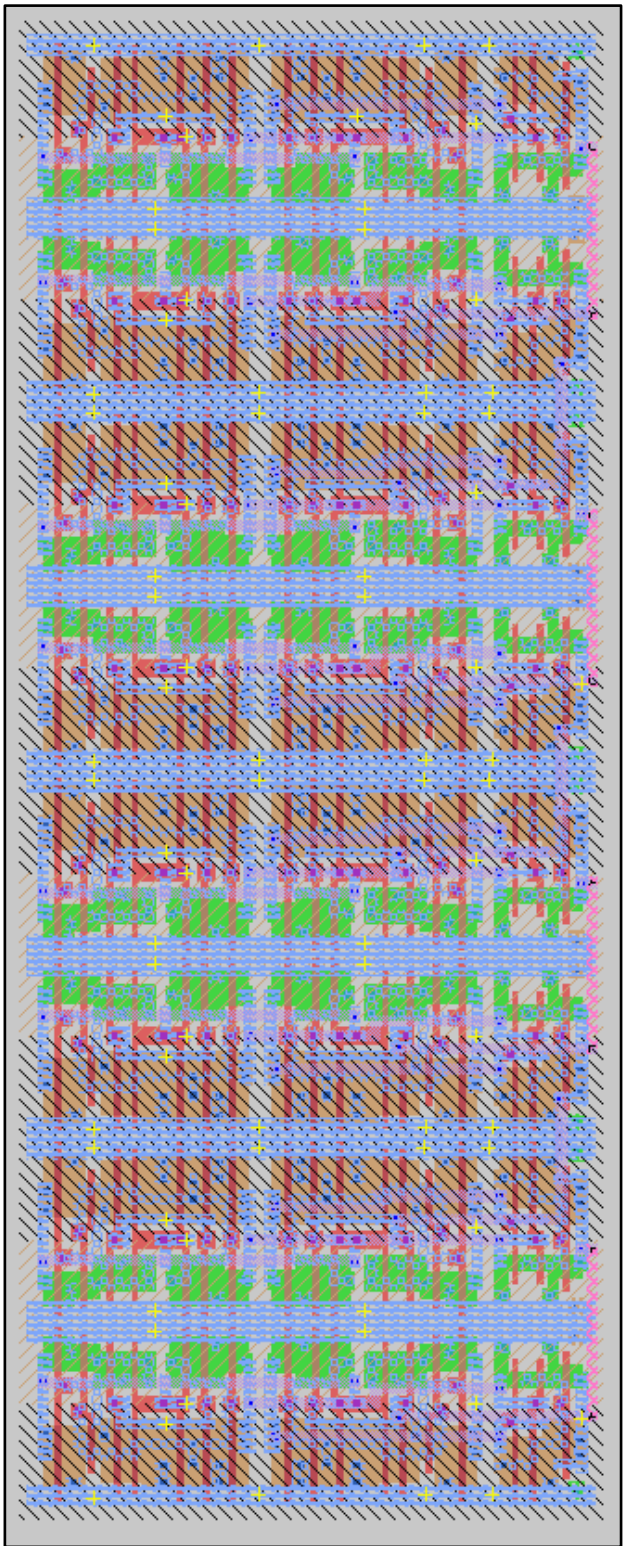

Figure 9: Layout of 8-Bit Ripple Full Adder

Adder / Subtractor

Continuing with the 8-bit ripple full adder, an XOR gate was added before each second input of the cell. One of the inputs of every XOR gate is connected to the control line, which decided which operation (addition or subtraction) to be performed. The other input to the XOR gate is one of the inputs of to the ALU. The control line is also connected to the carry-in of the first full adder cell. The formed cell will be an adder/subtractor cell. If the control line is LOW, the input connected through XOR is directly taken to the full adder, which results in addition. If the control line is high, then the input which is connected to the 8-bit ripple full adder inverts and is taken as input which results in subtraction.

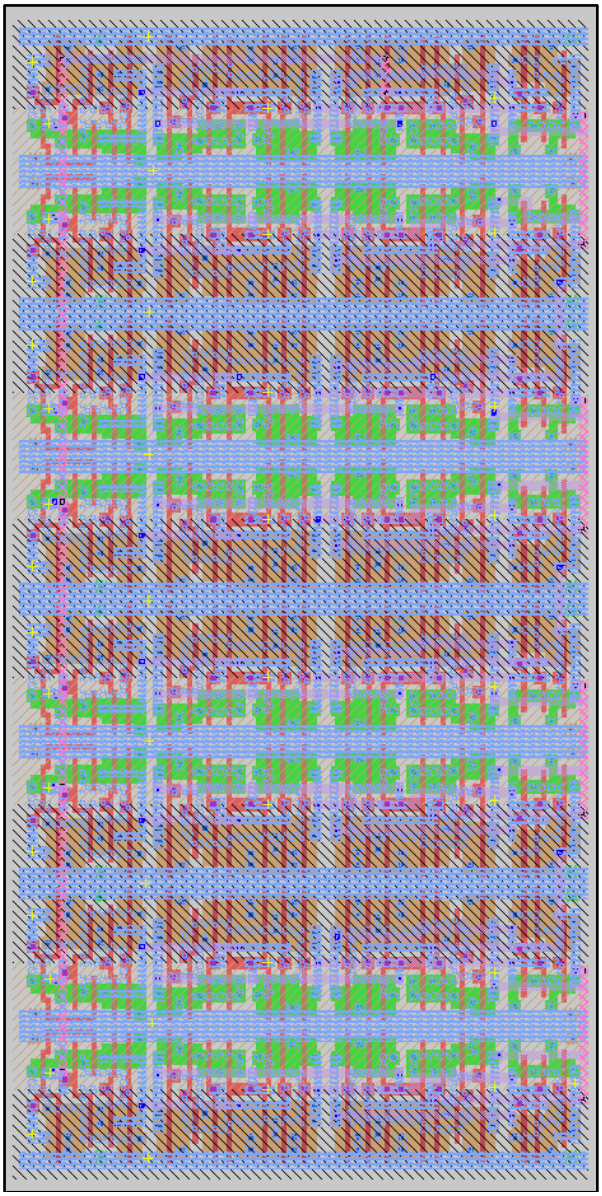

Figure 10: Layout of 8-Bit Adder / Subtractor

Multiplier

Using the 8-bit ripple full adder designed, 7 such layouts were stacked upon each other to form an 8-bit multiplier. AND gates are used to obtain partial products of the inputs. The circuit schematic of the 8-bit multiplier is given in Figure 1. The zeroth partial product, a zeroth bit of the first six ripple carry adders, and the nine outputs of the final ripple carry adder (8 sum bits and one carry-out) are together taken as the 8-bit product output.

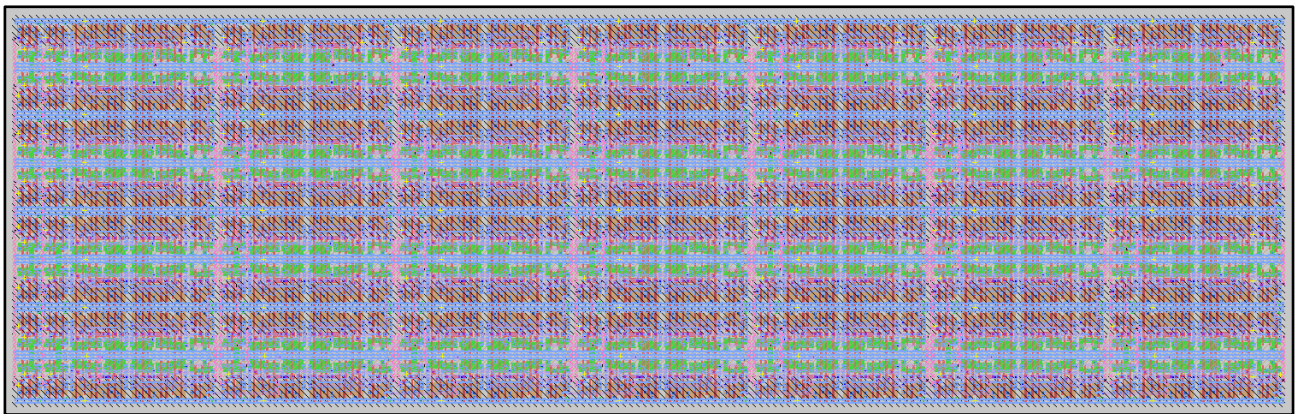

Figure 11: Layout of 8-Bit Multiplier

8-Bit 2:1 Multiplexer

This block of eight 2:1 multiplexers selects between 16 outputs of multiplier and nine outputs of adder/subtractor circuit based upon the value of the select signal “sin”.

Complete Design

The layout of the 8-bit Arithmetic Logic Unit is shown in Figure 12. The left part of the layout corresponds to the multiplier, and the right part corresponds to the adder/subtractor. The block of multiplexers selecting the outputs is placed at the rightmost part of the layout. A power ring and a power rail have been implemented for the layout. The thickness of these rails is taken to be higher to facilitate larger current flow. The inner rail is for $V_{DD,}$ and the outer rail is for GND. High metal widths were chosen to increase current carrying capacity and reduce wire resistance.

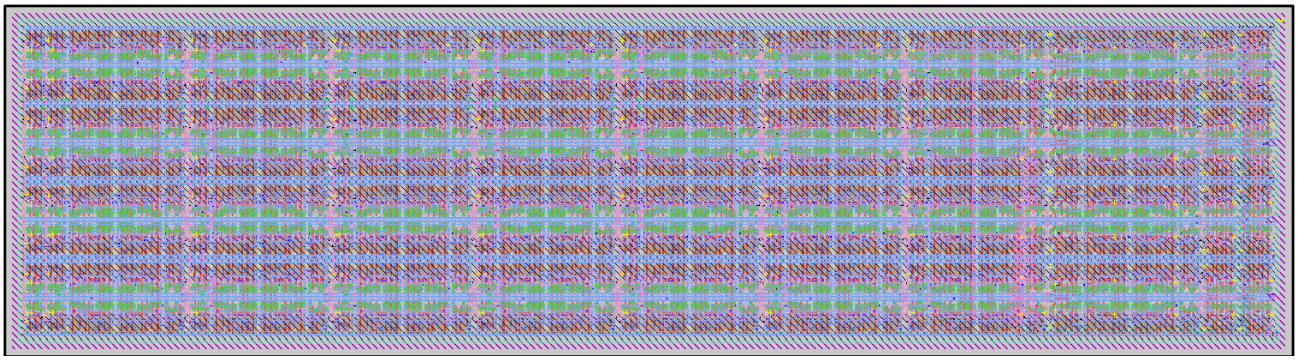

Figure 12: Layout of 8-Bit Arithmetic Logic Unit

Simulation

A sim file was extracted from the MAGIC layout of the 8-bit arithmetic logic unit. The sim file, along with the scmos parameters file, is used for testing the circuit on IRSIM. IRSIM is essentially a tool for simulating digital circuits. It treats transistors as ideal switches. Extracted capacitance and lumped resistance values are used to make the switch a little bit more realistic than the ideal, using the RC time constants to predict the relative timing of events. IRSIM is an effective platform to test the functioning of digital circuits, and it assesses them based on parameters such as dynamic power consumption and critical path delay.

The following Python code is used to generate a command file containing random test vectors and calls IRSIM automatically. It passes the simfile, the parameters file and the command file as command-line arguments to IRSIM. IRSIM the evaluates for the test cases and creates a log file and a powerlog file. The Python code then parses through the logfile and checks whether all test cases have been evaluated successfully by comparing the observed output from IRSIM with the expected output based on the inputs given. It then calculates the dynamic power consumption and total nodal capacitance of the layout.

```
# -*- coding: utf-8 -*-
"""
Created on Sun Dec  1 00:45:50 2020

@author: Rahul
"""

import os
import sys
import random

def evaluate(eachline, counter, anticounter):

     a0 = eachline[eachline.find('a0') + 3]
     a1 = eachline[eachline.find('a1') + 3]
     a2 = eachline[eachline.find('a2') + 3]
     a3 = eachline[eachline.find('a3') + 3]
     a4 = eachline[eachline.find('a4') + 3]
     a5 = eachline[eachline.find('a5') + 3]
     a6 = eachline[eachline.find('a6') + 3]
     a7 = eachline[eachline.find('a7') + 3]
     b0 = eachline[eachline.find('b0') + 3]
     b1 = eachline[eachline.find('b1') + 3]
     b2 = eachline[eachline.find('b2') + 3]
```

```
     b3 = eachline[eachline.find('b3') + 3]
     b4 = eachline[eachline.find('b4') + 3]
     b5 = eachline[eachline.find('b5') + 3]
     b6 = eachline[eachline.find('b6') + 3]
     b7 = eachline[eachline.find('b7') + 3]
     cin = eachline[eachline.find('cin') + 4]
     sel = eachline[eachline.find('sel') + 4]
     out0 = eachline[eachline.find('out0') + 5]
     out1 = eachline[eachline.find('out1') + 5]
     out2 = eachline[eachline.find('out2') + 5]
     out3 = eachline[eachline.find('out3') + 5]
     out4 = eachline[eachline.find('out4') + 5]
     out5 = eachline[eachline.find('out5') + 5]
     out6 = eachline[eachline.find('out6') + 5]
     out7 = eachline[eachline.find('out7') + 5]
     out8 = eachline[eachline.find('out8') + 5]
     out9 = eachline[eachline.find('out9') + 5]
     out10 = eachline[eachline.find('out10') + 6]
     out11 = eachline[eachline.find('out11') + 6]
     out12 = eachline[eachline.find('out12') + 6]
     out13 = eachline[eachline.find('out13') + 6]
     out14 = eachline[eachline.find('out14') + 6]
     out15 = eachline[eachline.find('out15') + 6]

     a = int(a7 + a6 + a5 + a4 + a3 + a2 + a1 + a0, 2)
     b = int(b7 + b6 + b5 + b4 + b3 + b2 + b1 + b0, 2)

     if (out15 == 'X' or out14 == 'X' or out13 == 'X' or out12
== 'X' or out11 == 'X' or out10 == 'X' or out9 == 'X' or out8 ==
'X' or out7 == 'X' or out6 == 'X' or out5 == 'X' or out4 == 'X'
or out3 == 'X' or out2 == 'X' or out1 == 'X' or out0 == 'X'):
          print (" Don't Cares present")
          print(out15 + out14 + out13 + out12 + out11 + out10 +
out9 + out8 + out7 + out6 + out5 + out4 + out3 + out2 + out1 +
out0)
          sys.exit()

     else:

          if(sel=='0'):

               result = int(out15 + out14 + out13 + out12 +
out11 + out10 + out9 + out8 + out7 + out6 + out5 + out4 + out3 +
out2 + out1 + out0,2)
               product = int(a*b)

               if(result!=product):
```

```
     print("*****************************************")
                    print("Error")
                    print(str(a) + "*" + str(b) + " failed")
                    print("Obtained " + out15 + out14 + out13 +
out12 + out11 + out10 + out9 + out8 + out7 + out6 + out5 + out4
+ out3 + out2 + out1 + out0 + " = " + str(result))
                    print("Expected " +
bin(product)[2:].zfill(16) + " = " + str(product))

     print("*****************************************")
                    anticounter = anticounter + 1
                    #sys.exit()

               else:

                    print("Success " +  str(a) + "*" + str(b) +
"=" + str(product))
                    counter = counter + 1

          else:

               if(cin=='0'):

                    result = int(out8 + out7 + out6 + out5 +
out4 + out3 + out2 + out1 + out0, 2)
                    add = int(a + b)

                    if(result!=add):

     print("*****************************************")
                         print("Error")
                         print(str(a) + " + " + str(b) + "
failed")
                         print("Obtained " + out15 + out14 +
out13 + out12 + out11 + out10 + out9 + out8 + out7 + out6 + out5
+ out4 + out3 + out2 + out1 + out0 + " = " + str(result))
                         print("Expected " +
bin(add)[2:].zfill(16) + " = " + str(add))

     print("*****************************************")
                         anticounter = anticounter + 1
                         #sys.exit()

                    else:
                         print("Success " +  str(a) + " + " +
str(b) + "=" + str(add))
                         counter = counter + 1
```

```
               else:

                    result = int(out7 + out6 + out5 + out4 +
out3 + out2 + out1 + out0, 2)

                    if(out8 == '0'):
                         result = result - (1<<8)

                    sub = int(a - b)

                    if(result!=sub):


     print("*****************************************")
                         print("Error")
                         print(str(a) + " - " + str(b) + "
failed")
                         print('Obtained ' + out15 + out14 +
out13 + out12 + out11 + out10 + out9 + out8 + out7 + out6 + out5
+ out4 + out3 + out2 + out1 + out0 + " = " + str(result))
                         print("Expected " +
bin(sub)[2:].zfill(16) + " = " + str(sub))

     print("*****************************************")
                         anticounter = anticounter + 1
                         #sys.exit()

                    else:

                         print("Success " +  str(a) + " - " +
str(b) + "=" + str(sub))
                         counter = counter + 1

     return counter, anticounter

def main():

     if(len(sys.argv)==1):
          print("Error. Please provide the .sim file as command
line argument")
     sim_file = sys.argv[1]

     cmdfile = "test.cmd"
     prmfile = "scmos100.prm"
     logfile = "output.log"
     input_vector_1 = "a7 a6 a5 a4 a3 a2 a1 a0"
     input_vector_2 = "b7 b6 b5 b4 b3 b2 b1 b0"
     select_vector = "sel cin"
```

```
     product_vector = "p15 p14 p13 p12 p11 p10 p9 p8 p7 p6 p5 p4
p3 p2 p1 p0"
     sum_vector = "s7 s6 s5 s4 s3 s2 s1 s0"
     out = "out15 out14 out13 out12 out11 out10 out9 out8 out7
out6 out5 out4 out3 out2 out1 out0"

     totalnumofvectors = 2**16
     global counter
     counter = 0
     global anticounter
     anticounter = 0
     maxsteppower = 0
     rand = 0

     with open (cmdfile , "w") as file :

          file.write("powlogfile powerlog.log\n")
          file.write("vsupply 5\n")
          file.write("h Vdd\n")
          file.write("l Gnd\n")
          file.write("powtrace Vdd Gnd " + out + " " +
select_vector + " " + product_vector + " " + sum_vector + " " +
input_vector_1 + " " + input_vector_2 + "\n")
          file.write("logfile " + logfile + " \n")
          file.write("stepsize 34.5\n")
          file.write("w " + out + " " + select_vector + " " +
input_vector_1 + " " + input_vector_2 + "\n")
          file.write("vector in " + select_vector + " " +
input_vector_1 + " " + input_vector_2 + "\n")
          file.write("powstep \n")
          file.write("set vlist {")

          for j in range(0,4,1):

               for i in range(0,totalnumofvectors,1):

                    if (random.randint(1, 256) < 3):

                         file.write(str(bin(j)[2:].zfill(2)) +
str(bin(i)[2:].zfill(16)) + " ")

          file.write("}\n")
          file.write("foreach vec $vlist {setvector in $vec ;
s}\n")
          file.write("powlogfile \n")
          file.write("sumcap \n")
          file.write("logfile \n")
          file.close()
```

```
     os.system("irsim " + prmfile + " " + sim_file + " -" +
cmdfile + " &")
     input("Press any key after irsim is done evaluating.")

     eachline = " "

     with open (logfile , "r") as fp :

          print ("Log file loaded. Evaluating...")
          line = fp.readline ()

          while line :

               if ('time' in line):

                    counter, anticounter = evaluate (eachline,
counter, anticounter)
                    line = fp.readline ()
                    eachline = " "

               elif ('step =' in line) :

                    steppower = line [ line.find ('step = ') + 7
: line.find ('step = ') + 14]
                    print ("Step power = " + steppower)
                    steppower = float(steppower)

                    if (steppower > maxsteppower):

                         maxsteppower = steppower

                    line = fp.readline ()
                    eachline = " "

               else :

                    eachline = eachline + line
                    line = fp.readline ()

          fp.close ()

     print("Accuracy " + str(counter/((counter +
anticounter))*100) + "%")
     print("Maximum Dynamic Power Consumed = " + str
(maxsteppower) + "mW")
     print("Power report : \n" + eachline)

if __name__ == '__main__':
     main()
```

Observation

The following points were observed after simulating the circuit.

- Area occupied = 2367 × 622 = 1472274 sq. microns
- Worst Case Dynamic Power Consumption = 2.8935 mW
- Worst Case Delay = 34.5 ns
- Number of cells used = 280
- Sum of nodal capacitances = 102.5914 pF
- Accuracy of all random test cases = 100 %
- Power consumption increases rapidly as the time step increase.
- One-fourth of the space is unutilized.
- The adder/subtracter is not capable of using the carry-in / borrow from the previous operation. This can be rectified by performing an XOR operation between the control line and the previous carry-in to obtain overflow.

Conclusion Drawn and Further Improvements

- The time steps were chosen to be as small as possible to reduce dynamic power consumption. The time step should also be above a certain threshold for the output to appear across the circuit due to delay in the operation.
- An enable signal along with a decoder can be added to activate only the required blocks within the ALU.
- A large size high fanout can be incorporated in the unutilized space.

Results

The experiment was performed, and all parameters were extracted and analyzed. The values obtained agreed with the theory, and hence simulations were verified. IRSIM was the simulator used in this task. The output of the script is noted down in **output.txt** which includes the results of the simulation and power analysis of the circuit.

References

1. IRSIM Usage Manual. Available at http://opencircuitdesign.com/irsim/archive/IRSIM_manual.pdf
2. Jan, M. Rabaey, Chandrakasan Anantha, and Nikolic Borivoje. “Digital Integrated Circuits–A Design Perspective.” *Pearson Education* (2003).