National Institute of Technology Karnataka,
Surathkal

EC372

VLSI Design Lab Project

4 bit Arithmetic Unit

Authors:
Anirudh BH 16EC105
Manan Sharma 16EC118

Supervisor:
Dr. Ramesh Kini

April 29, 2019

Contents

1 Introduction **2**

2 Specification Expected **2**

3 Implementation Flow **3**

4 Cells Used **4**
4.1 AND Gate 4
4.2 OR Gate 4
4.3 XOR Gate 5
4.4 Half Adder Cell 6
4.5 Multiplexer 6

5 Modular Approach **7**
5.1 Full Adder 7
5.2 Multiplier 9
5.3 8 Bit 2:1 Multiplexer 9
5.4 Complete Design 9
5.5 Testing using Python Script 10

6 Observation **14**

7 Conclusions Drawn and Further Improvements **14**

8 Results **15**

9 References **15**

1 Introduction

This project deals with the designing and implementation of a 4 Bit Arithmetic Unit. This arithmetic unit is capable of performing 3 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 4 bit numbers as the input and the output is defined by the following operations

sel	cin	Operation
0	X	a × 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 package. The pharosc technology file was also used during the implementation.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).

2 Specification Expected

1. Design of a 4 bit arithmetic unit capable of performing addition, subtraction and multiplication on a 4 bit input number.
2. The unit take in two 4 bit number
3. Based on the vales of the select line sel and carry in cin, the operation is selected.
4. A 4 bit multiplier based on 4 bit full adders and AND gates, provides an 8 bit output.
5. A 4 bit full adder/subtractor that take in two 4 bit number and a cin signal and gives a 4 bit sum and a 1 bit carry/borrow
6. A mux based 8 bit output selector, for choosing the necessary 8 output bits

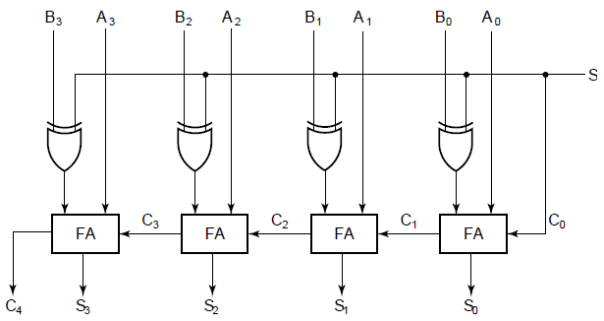

Figure 1: 4 Bit Full Adder/Subtractor using Full Adder and XOR gates

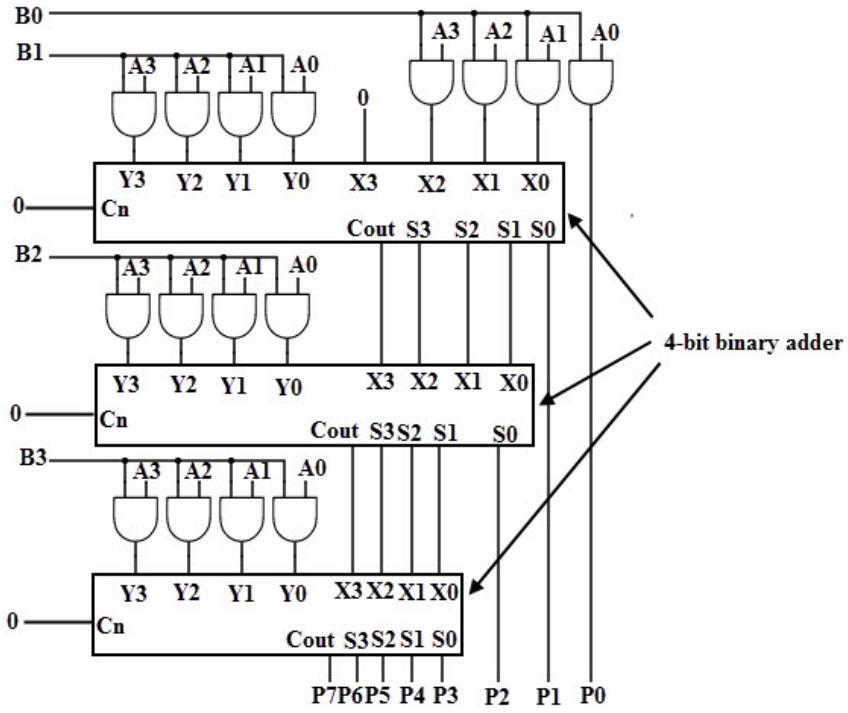

Figure 2: 4 Bit Multiplier using 4 bit Full Adders

3 Implementation Flow

1. Design of the project was arrived at and the software for the implementation was decided, MAGIC and IRSIM were chosen for designing and testing respectively.
2. Gate level circuit and the number of bits were arrived at for the entire Arithmetic Unit.
3. The Adder and Subtractor are implemented using a 4 bit ripple carry adder with auxiliary XOR gates for bit inversion (for 2's complement) in the case of subtraction.
4. The multiplication unit employs three 4 bit ripple carry adders and additional AND gate cells to perform the multiplication operation.
5. The operation is selected using a 8 bit 2:1 MUX. The sel signal selects between the multiplication operation and the addition/subtraction operation.
6. When sel = 1, then the cin flag/bit will decide bwteeen the addition and subtraction operation.
7. The XOR gates are used for performing 2's complement and the cin bit is the control signal
8. The circuit is layed out using MAGIC software and standard cells borrowed from Graham Petley's website (www.vlsitechnology.org) of pharosc technology
9. Simulation is done to test the logical operation using IRSIM
10. Script was written to automate the testing process for all possible input combinations and results tabulated

4 Cells Used

The following cells were used for the purpose of constructing the full adder and multiplier cells

4.1 AND Gate

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

A	B	Outpyt
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: AND gate Truth Table

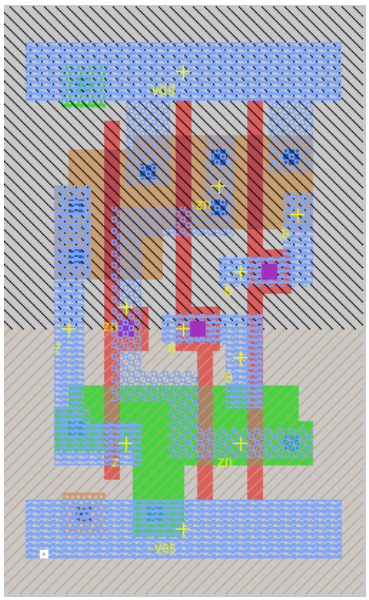

Figure 3: AND Gate Layout

4.2 OR Gate

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

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

Table 3: OR gate Truth Table

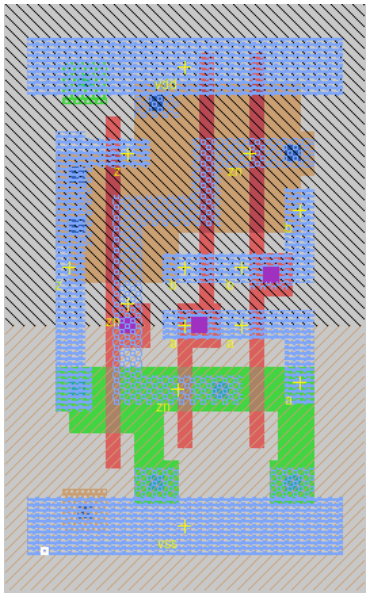

Figure 4: OR Gate Layout

4.3 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: XOR gate Truth Table

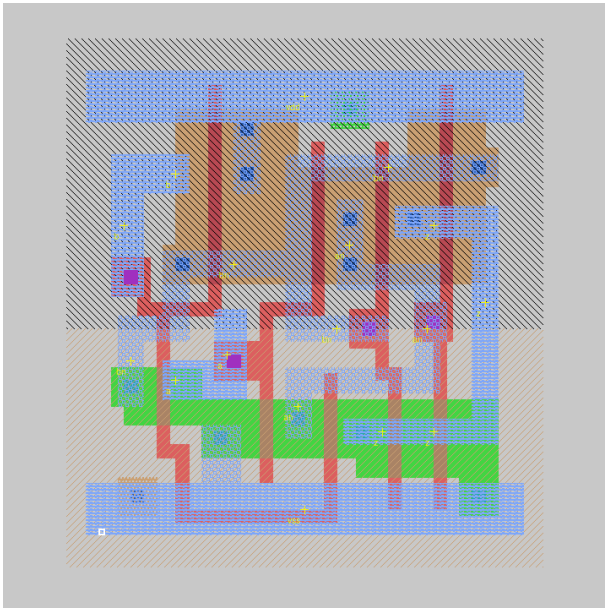

Figure 5: XOR Gate Layout

4.4 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: Half Adder gate Truth Table

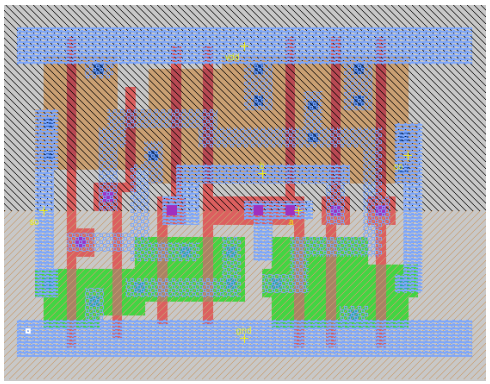

Figure 6: Half Adder Cell Layout

4.5 Multiplexer

A 2:1 MUX is used choose between multiplication and addition/subtraction operation.

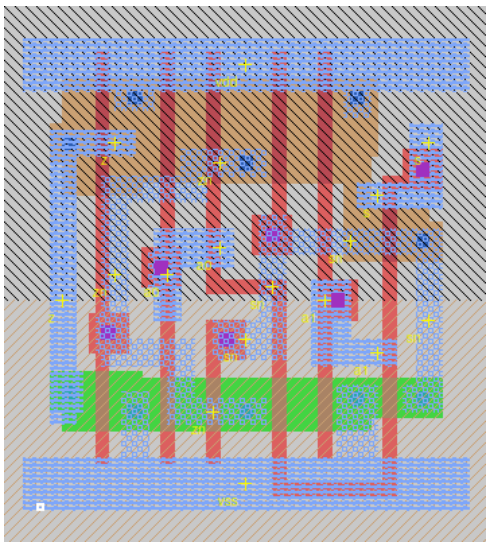

Figure 7: 2:1 MUX Layout

5 Modular Approach

In this project, we have a followed a bottom-up approach where the design was broken down into modules which were first constructed and tested, and then pieced together to form the target design. This section describes the various modules that are used in the construction of the 4 Bit Arithmetic Unit and the final design

5.1 Full Adder

As a first step, we have implemented a full adder with the help of 2 half adder cells and a an OR gate. The inputs A and B are fed in to the first HA. The sum of the first HA and the carry-in bit are fed as inputs to the next HA, whose Sum bit is the output Sum of the full adder. The carry-put is got by ORing the carry generated by the 2 Half-Adders.

A	B	Cin	Sum	Carry
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 6: Full Adder gate Truth Table

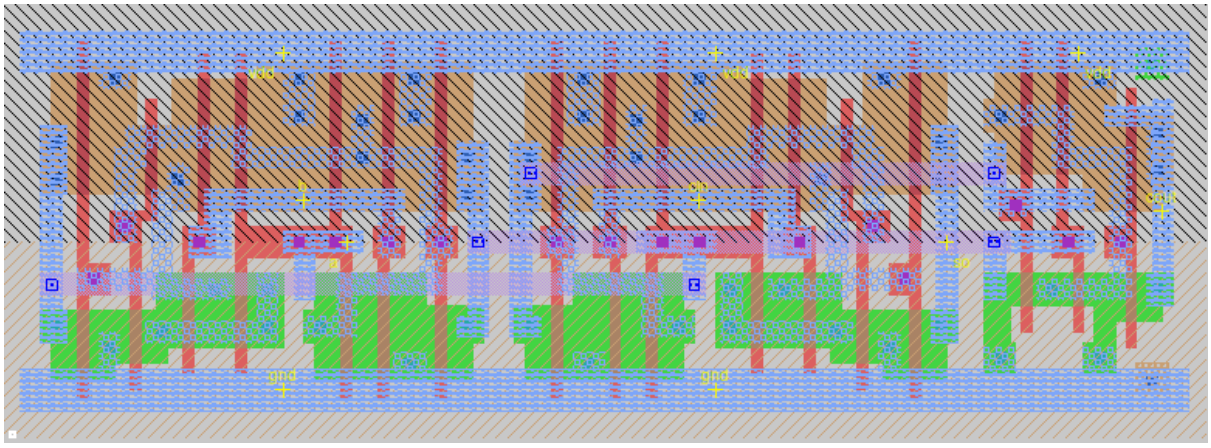

Figure 8: Full Adder Cell Layout

Continuing with the full adder, we have implemented a 4-Bit Ripple Carry adder. A and B are given as input s to the individual Full Adders. The cout of the first FA is fed as input to the second and the output of the second is fed as input to the third and so on

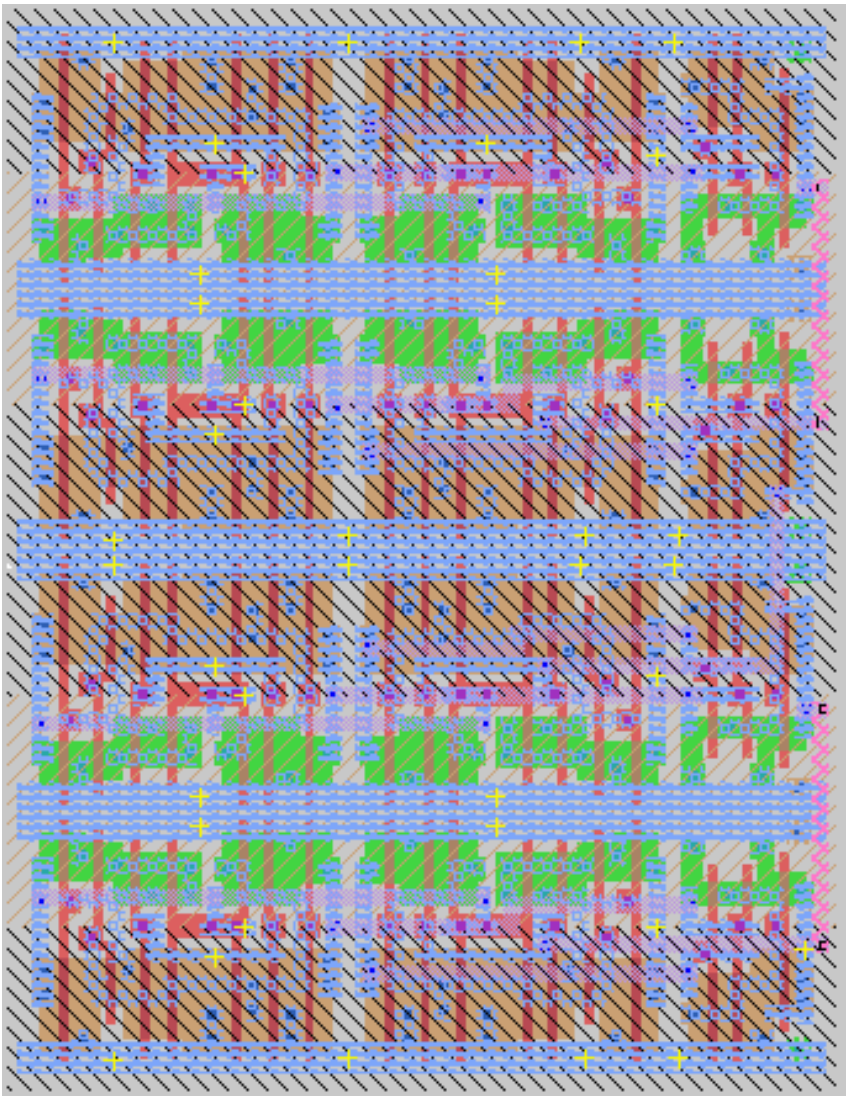

Figure 9: 4-Bit Ripple Carry Full Adder Cell Layout

The design pf the ripple carry block is then extended such that it can do both addition and subtraction. To implement the subtractor, we ca uses that fact that subtraction is the same as addition in 2's complement. The input and outputs of the new ripple carry adder/subractor are A3-A0, B3-B0, cin and S3-S0, cout respectively

This 2's complement is achieved using XOR gates. The input B are given to an XOR gate whose output is fed to the full adder cells. The other input to the full adder cells are the A bits. The cin is used a control signal for the XOR gates. When cin = 0, the output of the XOR gates will B3-B0 itself. Along with this the cin fed to the input as well is zero, resulting in the effective operation performed by the unit to be A+B. When cin = 1, The output of the XOR gates is the inverted version of B, $\overline{B3}$-$\overline{B0}$. The cin fed to the carry-in of the FA is also 1, resulting in the effective operation to be the 2's complement of the B input. This 2's compliment of B is added to A to get the Sum(Difference) and the carry-out(borrow).

5.2 Multiplier

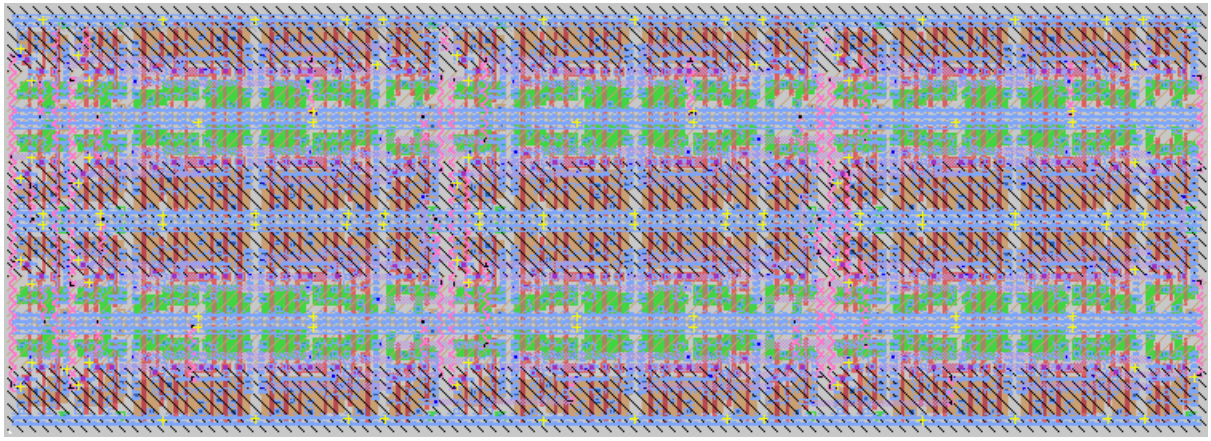

Figure 10: 4 bit Array Multiplier Layout

The layout for circuit diagram given in Figure (2) is shown in Figure (10). AND gates are used to multiply 1 bit to obtain partial products. These partial products are summed up using 4 bit Ripple Carry Adders to sum these partial products and compute product.

Three such stages are needed in order to compute the required 8 bit product term. The 0th partial product, 0^{th} bit of the first 2 ripple carry adders and the 5 outputs of the final ripple carry adder(4 sum bits and one carry-out) are together taken as the 8 bit product output

5.3 8 Bit 2:1 Multiplexer

This block is implemented using eight 2:1 MUXes. This takes in the sel signal and chooses between the outputs of the multiplications and addition operation.

5.4 Complete Design

The layout for the complete Arithmetic Unit is shown in Figure(11). The upper part in the layout is the 4 bit multiplier. The block below is the 4 bit Full Adder/Subtractor with the 8 bit MUX for output Selection. High metal widths were chosen to increase current carrying capacity and reduce wire resistance.

Regular wires were chosen of a shorter width to reduce the effects of the parasitics.

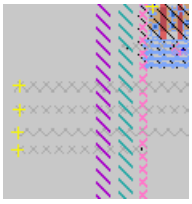

Figure 12: Input Output Pads used

Figure(12) shows the IO pads created for the inputs. Similar IO pads have been created for the outputs.

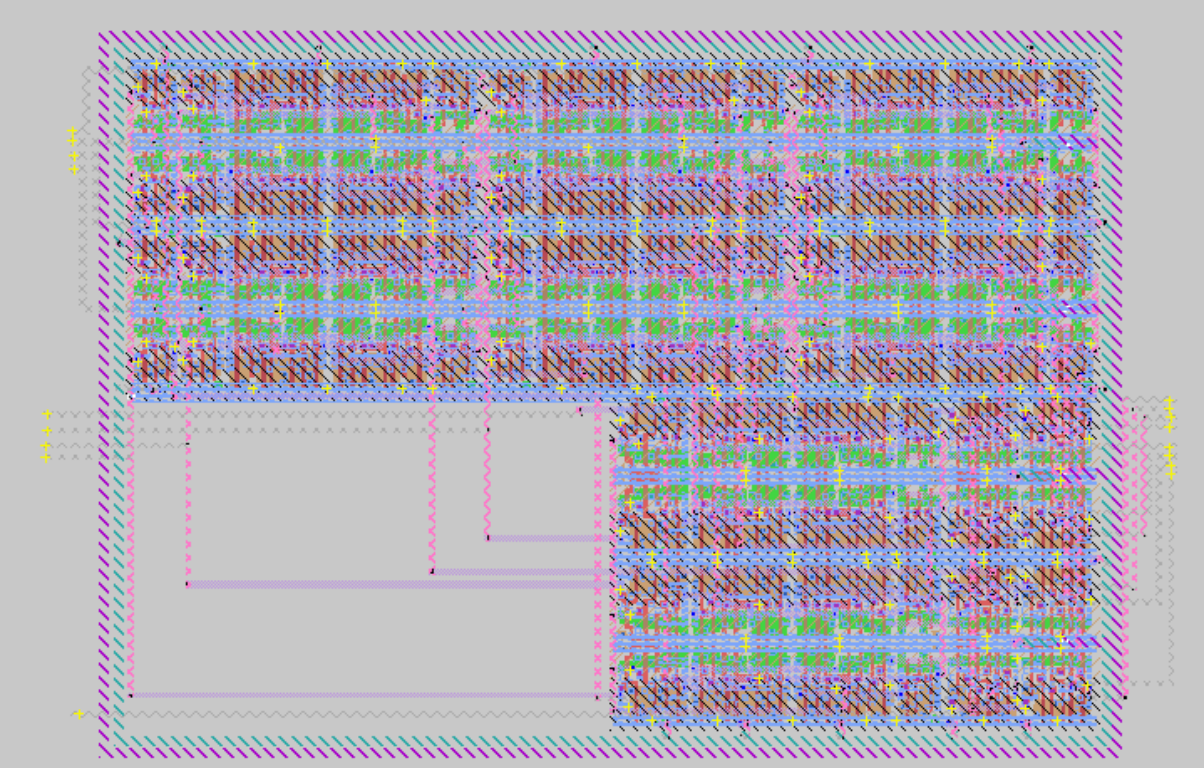

Figure 11: 4 bit Arithmetic Unit Layout

A power ring and power rails have been implemented for the layout. These thickness of these rails are taken to be higher to facilitate larger current flow. The inner rail is for the V_{DD} and the outer rail is for GND.

5.5 Testing using Python Script

In this subsection, we explain the testing procedures that were used in IRSIM to test our design. 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. We concentrate on these testing parameters in our project.

The following python code is used to generate a .cmd file containing the test vectors, and calls IRSIM to evaluate automatically and store all outputs in a .log file. Once IRSIM is done evaluating, python script then reads the .log file, and checks the accuracy of the whole system, by comparing the expected output based on inputs given and observed output.

```
# sel = 0 does multiplication
# cin=0 does addition, cin=1 does
import os
import time
import sys

if(len(sys.argv)<2):
```

```
    print("Error. Please provide the .sim file as command line
       argument")

cmdfile = "test_file.cmd"
sim_file = sys.argv[1]
logfile = "output.log"
in1_vector = "a_3 a_2 a_1 a_0"
in2_vector = "b_3 b_2 b_1 b_0"
sel_vector = "sel cin"
p_vector = "p_7 p_6 p_5 p_4 p_3 p_2 p_1 p_0"
out = "out_7 out_6 out_5 out_4 out_3 out_2 out_1 out_0"
s_vector = "s_3 s_2 s_1 s_0"
totalnumofvectors = 2**8
counter = 0
maxsteppower = 0
def evaluate(eachline):
    global counter
    a_0_txt = eachline[eachline.find('a_0')+4]
    a_1_txt = eachline[eachline.find('a_1')+4]
    a_2_txt = eachline[eachline.find('a_2')+4]
    a_3_txt = eachline[eachline.find('a_3')+4]
    b_0_txt = eachline[eachline.find('b_0')+4]
    b_1_txt = eachline[eachline.find('b_1')+4]
    b_2_txt = eachline[eachline.find('b_2')+4]
    b_3_txt = eachline[eachline.find('b_3')+4]
    cin_txt = eachline[eachline.find('cin')+4]
    sel_txt = eachline[eachline.find('sel')+4]
    out_0_txt = eachline[eachline.find('out_0')+6]
    out_1_txt = eachline[eachline.find('out_1')+6]
    out_2_txt = eachline[eachline.find('out_2')+6]
    out_3_txt = eachline[eachline.find('out_3')+6]
    out_4_txt = eachline[eachline.find('out_4')+6]
    out_5_txt = eachline[eachline.find('out_5')+6]
    out_6_txt = eachline[eachline.find('out_6')+6]
    out_7_txt = eachline[eachline.find('out_7')+6]
    a = int(a_3_txt+a_2_txt+a_1_txt+a_0_txt,2)
    b = int(b_3_txt+b_2_txt+b_1_txt+b_0_txt,2)
    if(out_7_txt == 'X' or out_6_txt == 'X' or out_5_txt == 'X' or
        out_4_txt == 'X' or out_3_txt == 'X' or out_2_txt == 'X'
       or out_1_txt == 'X' or out_0_txt == 'X'):
        print("Don't Cares present")
        print(out_7_txt+out_6_txt+out_5_txt+out_4_txt+out_3_txt+
           out_2_txt+out_1_txt+out_0_txt)

    else:
        if(sel_txt=='0'):
            result = int(out_7_txt+out_6_txt+out_5_txt+out_4_txt+
               out_3_txt+out_2_txt+out_1_txt+out_0_txt,2)
            product = int(a*b)
            if(result!=product):
                print("****************************************"
```

```
                )
            print("Error")
            print(str(a)+"*"+str(b)+ " failed")
            print("Obtained " + out_7_txt+out_6_txt+out_5_txt+
                out_4_txt+out_3_txt+out_2_txt+out_1_txt+
                out_0_txt + " = "+ str(result))
            print("Expected " + bin(product)[2:].zfill(8) + "
                = " + str(product))
            print("****************************************"
                )
            #sys.exit()
        else:
            print("Success " +  str(a) + "*" + str(b) + "=" +
                str(product))
            counter = counter + 1
    else:
        if(cin_txt=='0'):
            result = int(out_4_txt+out_3_txt+out_2_txt+
                out_1_txt+out_0_txt,2)
            add = int(a+b)
            if(result!=add):
                print("
                    ****************************************"
                    )
                print("Error")
                print(str(a)+"+"+str(b)+ " failed")
                print("Obtained " + out_7_txt+out_6_txt+
                    out_5_txt+out_4_txt+out_3_txt+out_2_txt+
                    out_1_txt+out_0_txt + " = "+ str(result))
                print("Expected " + bin(add)[2:].zfill(8) + "
                    = " + str(add))
                print("
                    ****************************************"
                    )
                #sys.exit()
            else:
                print("Success " +  str(a) + "+" + str(b) + "=
                    " + str(add))
                counter = counter + 1
        else:
            result = int(out_3_txt+out_2_txt+out_1_txt+
                out_0_txt,2)
            if(out_4_txt=='0'):
                result=result-(1<<4)
            sub = int(a-b)
            if(result!=sub):
                print("
                    ****************************************"
                    )
                print("Error")
                print(str(a)+"-"+str(b)+ " failed")
```

```
                    print('Obtained ' + out_7_txt+out_6_txt+
                       out_5_txt+out_4_txt+out_3_txt+out_2_txt+
                       out_1_txt+out_0_txt + " = "+ str(result))
                    print("Expected " + bin(sub)[2:].zfill(8) + "
                       = " + str(sub))
                    print("
                       ****************************************"
                       )
                    #sys.exit()
                else:
                    print("Success " +  str(a) + "-" + str(b) + "=
                       " + str(sub))
                    counter = counter + 1

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 + " " + sel_vector+" " +
       p_vector+" "+s_vector+" "+in1_vector + " "+ in2_vector+"\n"
       )
    file.write("logfile "+ logfile +"\n")
    file.write("stepsize 13.6\n")
    file.write("w " + out + " " + sel_vector + " "+p_vector + " "+
       s_vector+" "+in1_vector + " "+ in2_vector + "\n")
    file.write("vector In "+ sel_vector + " " + in1_vector + " "+
       in2_vector + "\n")
    file.write("powstep\n")
    file.write("set vlist {")
    for j in range(0,4,1):
        for i in range(0,totalnumofvectors,1):
            file.write(str(bin(j)[2:].zfill(2))+str(bin(i)[2:].
               zfill(8)) + " ")
    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 "+ sim_file + " -"+cmdfile+" &")
print("Waiting for irsim to write output log file....")
input("Press any key once 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):
            evaluate(eachline)
```

```
            line = fp.readline()
            eachline = ""
        elif('step = ' in line):
            steppower_txt = line[line.find('step = ')+7:line.find(
               'step = ')+14]
            print("Step power ="+ steppower_txt)
            steppower = float(steppower_txt)
            if(steppower>maxsteppower):
                maxsteppower=steppower
            line=fp.readline()
            eachline = ""
        else:
            eachline = eachline + line
            line = fp.readline()
    fp.close()

print("Accuracy " + str(counter/(4*totalnumofvectors)*100) + "%")
print("Maximum Dynamic Power Consumed = "+ str(maxsteppower) + "
   mW")
print("Power report: \n" + eachline)
```

6 Observation

1. Area occupied = 992 x 642 = 636864 square microns
2. Worst Case Dynamic Power Consumed = 1.39019 mW
3. Delay = 13.6 ns (worst case)
4. Number of cells used = 76
5. Sum of nodal Capacitance = 27.359362 pF
6. The 4 bit arithmetic unit is verified to work for all possible inputs using the python script.
7. As the time step increases, the power consumed increases exponentially.
8. One fourth of the space in the layout is unutilised.
9. The adder/subtracter is not capable of using the cin/borrow from previous operation. This however can be rectified by performing an XOR operation between the control cin and the previous cin.

7 Conclusions Drawn and Further Improvements

1. Shorter time steps have to be chosen to ensure lower dynamic power consumption. However, at the same time, time step should also be more than the time taken for output to appear and stabilise.
2. In order to further reduce the power consumption, an enable signal along with a decoder can be used to activate only the required blocks within the arithmetic unit.

3. The unutilised space can be used for further improvements like to incorparte a large size high fanout input buffers.
4. A carry save/carry look ahead adder can be used to improve the timing the system, at the loss of simplicity.

8 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.

9 References

1. IRSIM Usage Manual. Avaiable at http://opencircuitdesign.com/irsim/archive/IRSIM_manual.pdf(Last Accessed 28/04/2019)
2. Jan, M. Rabaey, Chandrakasan Anantha, and Nikolic Borivoje. "Digital Integrated Circuits–A Design Perspective." *Pearson Education* (2003).