EE577A Lab #4
Pipelined 5-bit 2's Complement Multiplier Design

SHANG ZHOU 2725583204
ANH VU 2396504143
HONGXIANG GAO 8095639536
University of Southern California
April 26, 2019

Contents

1 Introduction

In this lab, we design a 5-bit pipelined 2's complement multiplier (the inputs are 5-bit 2's complement numbers and the output is a 10-bit 2's complement number). Registers are inserted between each row so that the multiplier is divided into 4 stages. The 5 bit multiplier is built and tested based on the architecture provided for the 4-bit multiplier.

In the later section, the design is optimized to reach a higher performance in clock timing.

2 Pipelined 5-bit Multiplier Design

2.1 Design Concept

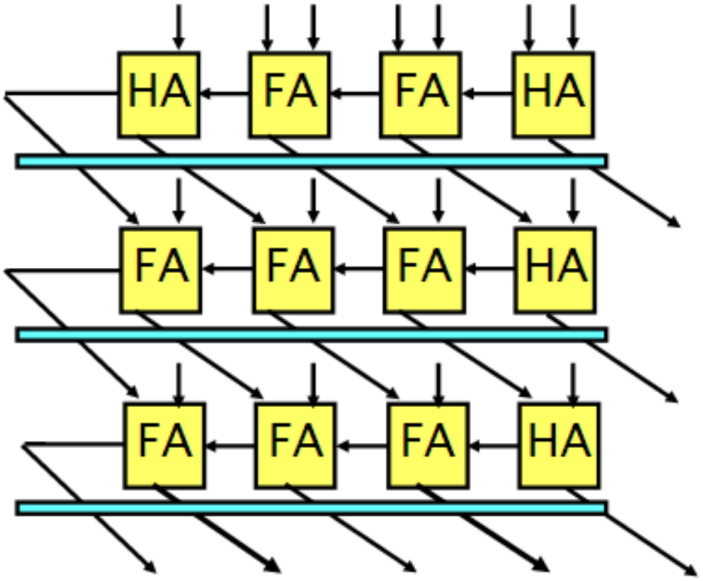

Figure 1: Pipelined Multiplier Diagram

Here illustrates the structure of a 4-bit pipelined 2's compliment multiplier. The main concept is inserting DFF's between each Adder level so that the complete process could be divided into several parts. For 5-bit pipelined 2's compliment multiplier, three level of DFF's are needed. And then the new clock is 1/4 of the original clock.

2.2 Schematic Design

The figure below show the schematic for the 5-bit 2's compliment multiplier.

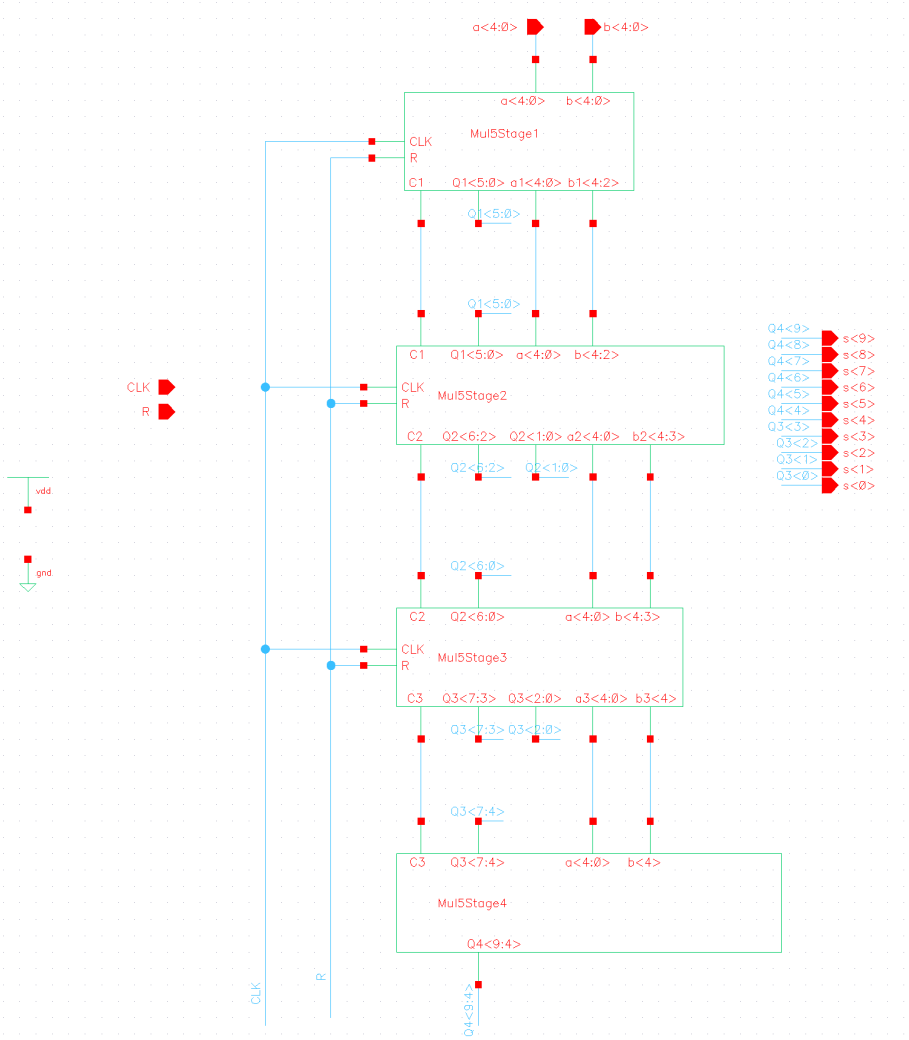

Figure 2: Multiplier schematic

- Stage 1

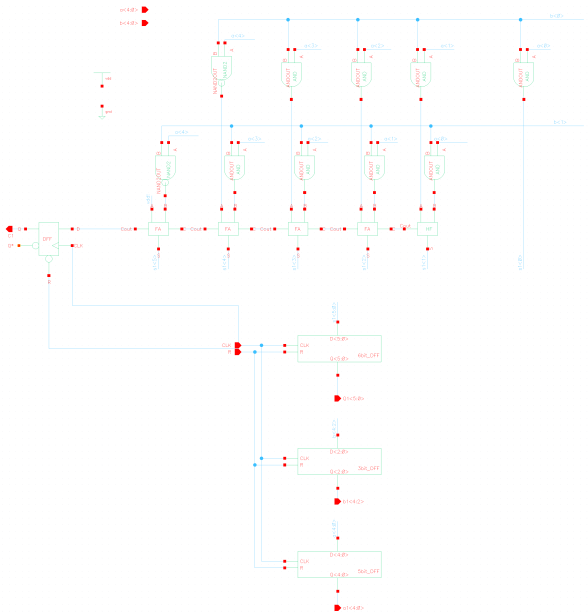

Figure 3: Multiplier stage 1 schematic

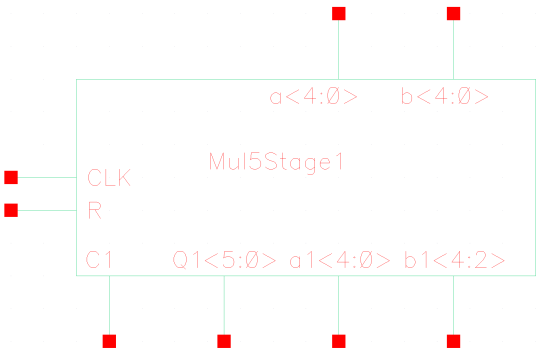

Figure 4: Multiplier schematic stage 1 schematic symbol

Here shows the diagram of both schematics and symbol of Stage 1.

- Stage 2

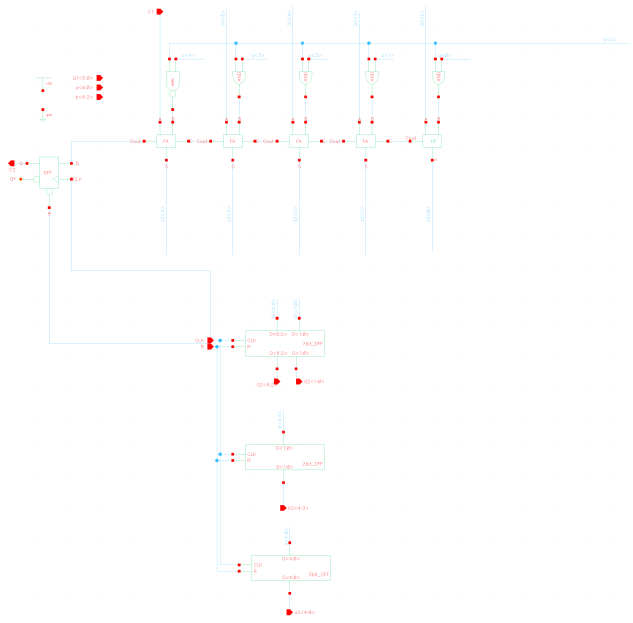

Figure 5: Multiplier stage 2 schematic

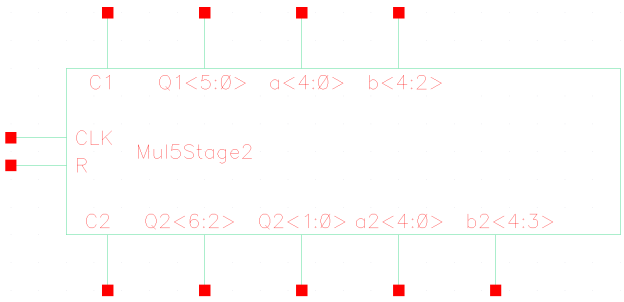

Figure 6: Multiplier schematic stage 2 schematic symbol

Here shows the diagram of both schematics and symbol of Stage 2.

- Stage 3

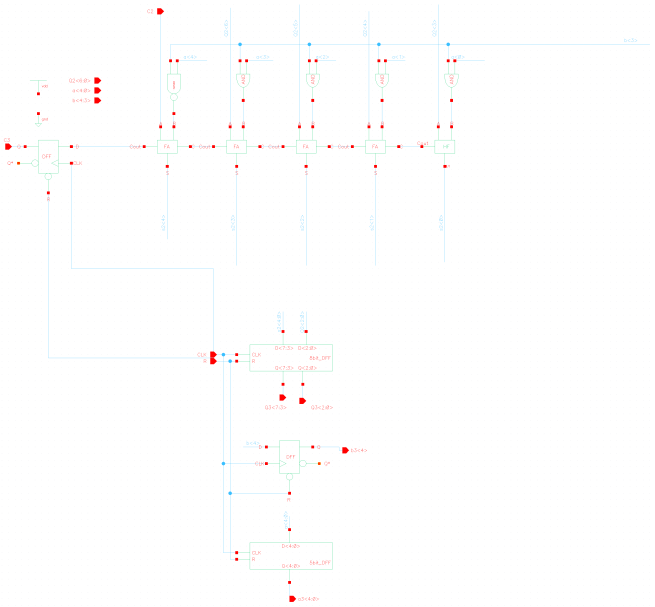

Figure 7: Multiplier stage 1 schematic

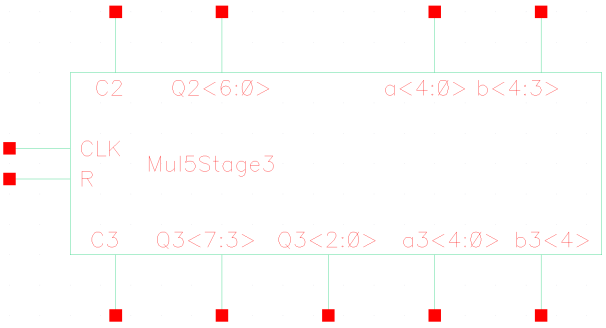

Figure 8: Multiplier schematic stage 1 schematic symbol

Here shows the diagram of both schematics and symbol of Stage 3.

- Stage 4

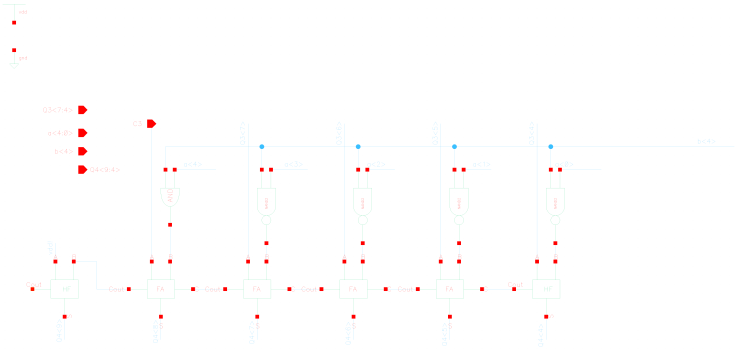

Figure 9: Multiplier stage 4 schematic

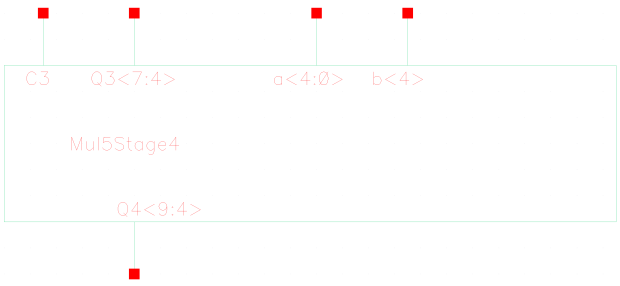

Figure 10: Multiplier schematic stage 4 schematic symbol

Here shows the diagram of both schematics and symbol of Stage 4. The outputs of stage 4 is the final outputs of the multiplier.

3 Front-end Python Code for Vector File Generation and Result Test

Here shows the Python code for generating the vector file and comparing the difference between golden result and output signals.

3.1 Vector File Generation

Here shows the Python code for vector file generation.

```
#This is a script to generate the random test cases for lab4
#Created by Shang

```

```
import random

fp=open("5bit_Mul_test.vec","a+")
fp.truncate()
fp2=open("golden_result.txt","a+")
fp2.truncate()

fp.write("radix 2 3 2 3\n")
fp.write("io i i i i\n")
fp.write("vname a<[4:3]> a<[2:0]> b<[4:3]> b<[2:0]>\n")
fp.write("vih 1.8\n")
fp.write("slope 0.01\n")
fp.write("tunit 1ns\n")


#The following part is 10 random test cases:

random.seed()
for i in range(10):
  randtest=[]
  randtest.append(i*8)
  randtest.append(random.randint(0,3))
  randtest.append(random.randint(0,7))
  randtest.append(random.randint(0,3))
  randtest.append(random.randint(0,7))
  print(randtest)
  for item in range(len(randtest)):
    fp.write(str(randtest[item])+' ')
  fp.write("\n")
  s1=0
  s2=0
  if (randtest[1]==0):
    s1=0+randtest[2]
  elif (randtest[1]==1):
    s1=8+randtest[2]
  elif (randtest[1]==2):
    s1=(-16)+randtest[2]
  elif (randtest[1]==3):
    s1=(-8)+randtest[2]
  print s1

  if (randtest[3]==0):
    s2=0+randtest[4]
  elif (randtest[3]==1):
    s2=8+randtest[4]
  elif (randtest[3]==2):
    s2=(-16)+randtest[4]
  elif (randtest[3]==3):
    s2=(-8)+randtest[4]
  print s2

```

```
  result=0
  result=s1*s2
  print result
  fp2.write(str(result)+"\n")
#The following part is 5 directed test cases:

fp.write("50 0 0 0 0\n")
fp2.write("0\n")
#result =0

fp.write("55 0 1 0 3\n")
fp2.write("3\n")
#result =3

fp.write("60 0 2 0 6\n")
fp2.write("12\n")
#result=12
fp.write("65 0 3 0 3\n")
fp2.write("9\n")
#result= 9
fp.write("70 1 0 0 2\n")
fp2.write("16\n")
#result=16

fp.close()
fp2.close()
```

Code Listing 1: Vector File Generation

3.2 Golden Result Comparison

Here shows the Python code for result comparison.

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
# Author: Shang Zhou

import csv

def twos_comp(val):
    result=int(val[1])*2**8 + int(val[2])*2**7- int(val[0])*2**(9)
    result=result+int(val[3])*2**6+int(val[4])*2**5+int(val[5])
    *2**4
    result=result+int(val[6])*2**(3)+int(val[7])*2**2+int(val[8])
    *2**1+int(val[9])*2**0
    return result
# csv file name
filename = "Lab4P1.csv"

# initializing the titles and rows list
fields = []
```

```
rows = []
fp2=open("sim.txt","w")
fp2.truncate()

# reading csv file
with open(filename, 'r') as csvfile:
    # creating a csv reader object
    csvreader = csv.reader(csvfile)

    # extracting field names through first row
    fields = next(csvreader)

    # extracting each data row one by one
    for row in csvreader:
        rows.append(row)

    for row in rows:
        print(row[0]+' ',end='')
        m=twos_comp(row[1])
        print('%d' %m)
        if(row[0]=="7.406n" or row[0]=="9.559n" or row[0]=="12.10n
    " or row[0]=="14.25n" or row[0]=="16.14n" or row[0]=="18.27n"
    or row[0]=="21.13n" or row[0]=="23.13n" or row[0]=="25.45n" or
     row[0]=="27.58n" or row[0]=="29.10n" or row[0]=="31.05n" or
    row[0]=="33.24n" or row[0]=="35.44n" or row[0]=="37.86n"):
            fp2.write(str(m)+'\n')


fp2.close()
csvfile.close()

f3=open("sim.txt","r")
f4=open("golden_result.txt","r")
f5=open("compare_result.txt","w")
for line1 in f3:
    for line2 in f4:
        if line1==line2:
            f5.write("SAME\n")
        else:
            f5.write(line1 + line2)
        break
f3.close()
f4.close()
f5.close()
```

Code Listing 2: Result Comparison

4 Pipelined 5-bit Multiplier Optimization

4.1 Schematic Design

Here shows the optimized version of the 5-bit pipelined 2's compliment multiplier. The optimization is focus on changing the location of registers to achieve a more balanced pipeline design. The total number of DFFs in the registers remain unchanged.

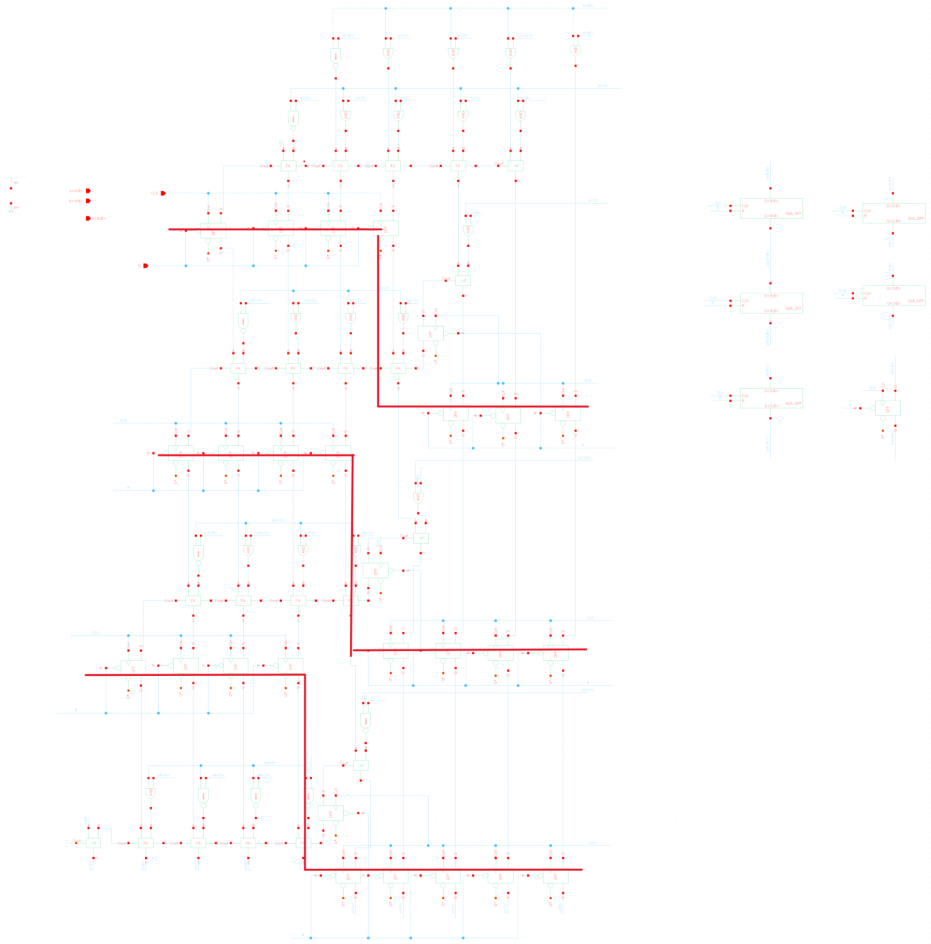

Figure 11: Optimized Multiplier Schematic

From Figure 11, the first level registers are placed as a twisted line. The left part is between first level adder and next level of AND gates, the right part is between second level adder and next level of AND gates. And it's similar for the configuration of other three levels. The reason for this optimization is based on the concept of "Look Ahead", clock is saved because

the parallel computation of the right part when transferring signals to the left part.

4.2 Layout Design

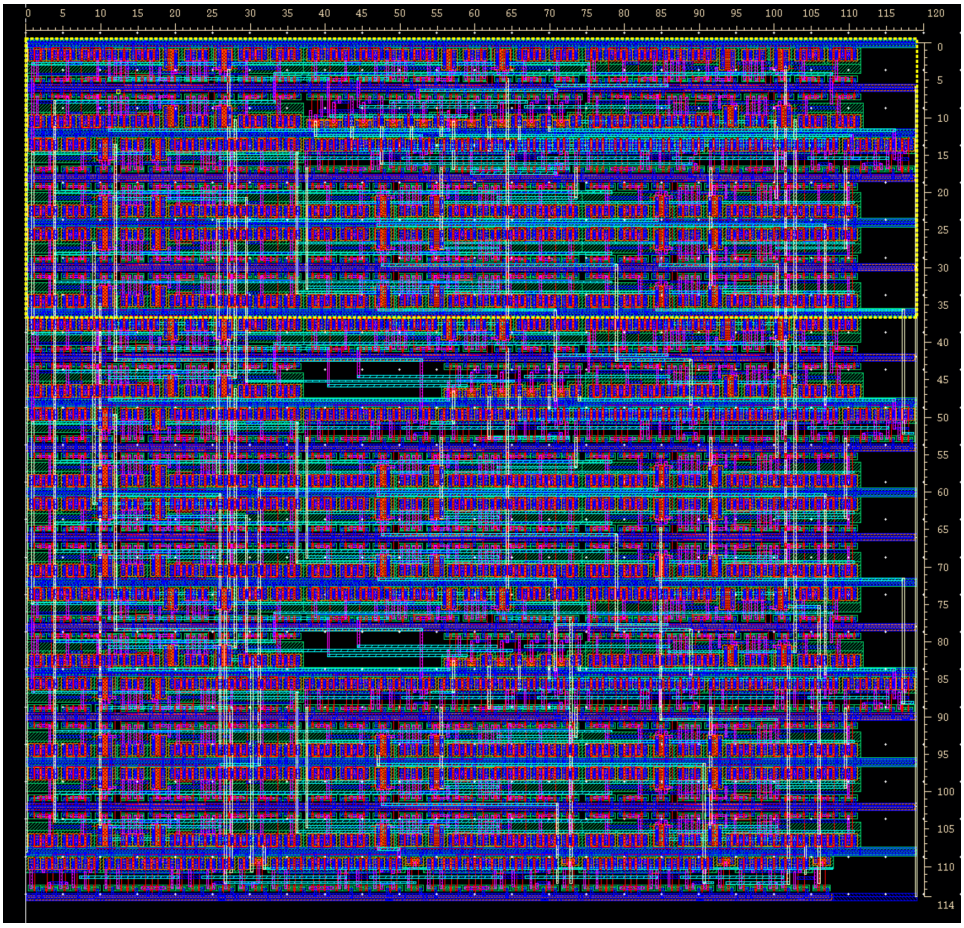

Figure 12: Optimized Multiplier Layout

Here shows the layout design of the optimized circuit. Its height is 114, and its width is 120.

```
@(#)$CDS: LVS version 6.1.7-64b 07/05/2016 20:10 (sjfhw313) $

Command line: /usr/local/cadence/IC617/tools.lnx86/dfII/bin/64bit/
    LVS -dir /home/scf-11/anhv/EE477_VLSI/cds/LVS -l -s -t /home/
    scf-11/anhv/EE477_VLSI/cds/LVS/layout /home/scf-11/anhv/
    EE477_VLSI/cds/LVS/schematic
Like matching is enabled.
Net swapping is enabled.
Using terminal names as correspondence points.
```

```
Compiling Diva LVS rules...

    Net-list summary for /home/scf-11/anhv/EE477_VLSI/cds/LVS/
    layout/netlist
        count
         880              nets
         24               terminals
         1676             pmos
         1676             nmos

    Net-list summary for /home/scf-11/anhv/EE477_VLSI/cds/LVS/
    schematic/netlist
        count
         880              nets
         24               terminals
         956              pmos
         956              nmos


    Terminal correspondence points
    N866        N55         CLK
    N861        N14         R
    N863        N33         a<0>
    N859        N48         a<1>
    N856        N49         a<2>
    N878        N50         a<3>
    N875        N64         a<4>
    N876        N46         b<0>
    N873        N2          b<1>
    N871        N26         b<2>
    N869        N11         b<3>
    N865        N59         b<4>
    N858        N1          gnd!
    N864        N53         s<0>
    N860        N19         s<1>
    N857        N51         s<2>
    N879        N15         s<3>
    N877        N47         s<4>
    N874        N4          s<5>
    N872        N25         s<6>
    N870        N3          s<7>
    N867        N37         s<8>
    N862        N57         s<9>
    N868        N0          vdd!

Devices in the netlist but not in the rules:
        pcapacitor
Devices in the rules but not in the netlist:
        cap nfet pfet nmos4 pmos4

The net-lists match.
```

```
                              layout   schematic
                                 instances
        un-matched               0        0
        rewired                  0        0
        size errors              0        0
        pruned                   0        0
        active                   3352     1912
        total                    3352     1912

                                   nets
        un-matched               0        0
        merged                   0        0
        pruned                   0        0
        active                   880      880
        total                    880      880

                                 terminals
        un-matched               0        0
        matched but
        different type           0        0
        total                    24       24


Probe files from /home/scf-11/anhv/EE477_VLSI/cds/LVS/schematic

devbad.out:

netbad.out:

mergenet.out:

termbad.out:

prunenet.out:

prunedev.out:

audit.out:


Probe files from /home/scf-11/anhv/EE477_VLSI/cds/LVS/layout

devbad.out:

netbad.out:

mergenet.out:

termbad.out:
```

```
107 prunenet.out:
108
109 prunedev.out:
110
111 audit.out:
```

Code Listing 3: Netlist

5 Schematic Simulation Result

5.1 Functional Waveforms and results of the Schematic

For this part, we just need to check the functionality of our schematic and report the shortest time it required. For the simple pipelined version, the best clock period is 2.2ns.The following pictures show the simulation results.

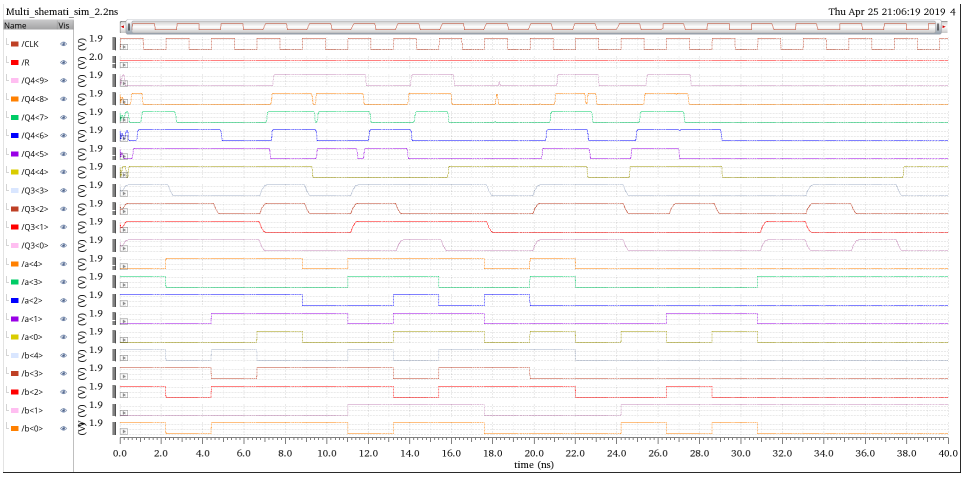

Figure 13: Final simulation for the simple pipelined version

In order to give the direct view of our result, We create a bus for our input a and b, and output Q3.

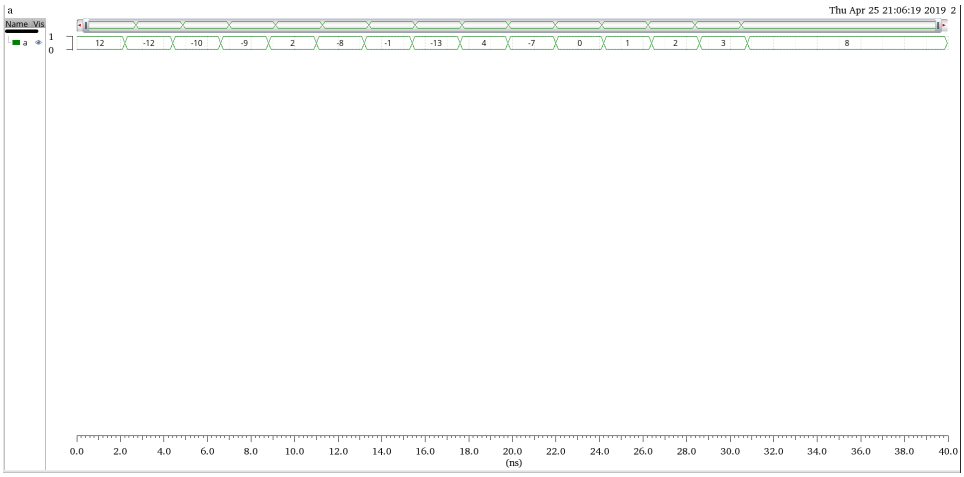

Figure 14: Input a

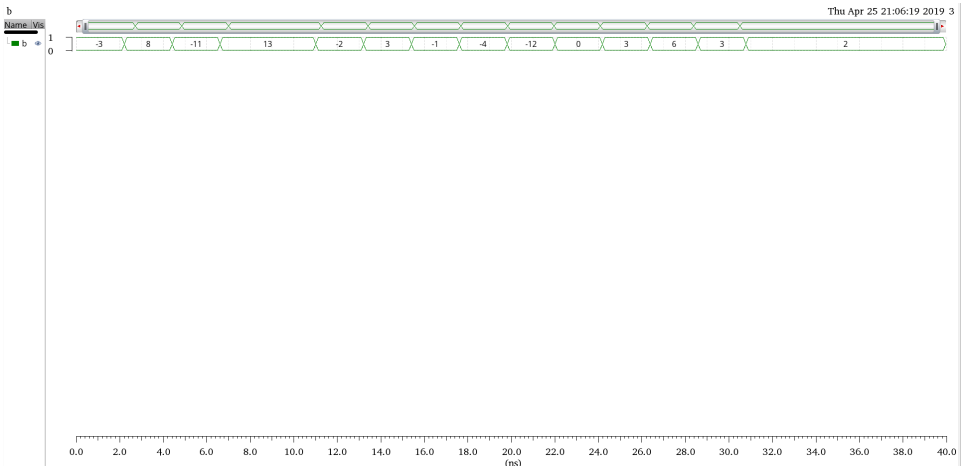

Figure 15: Input b

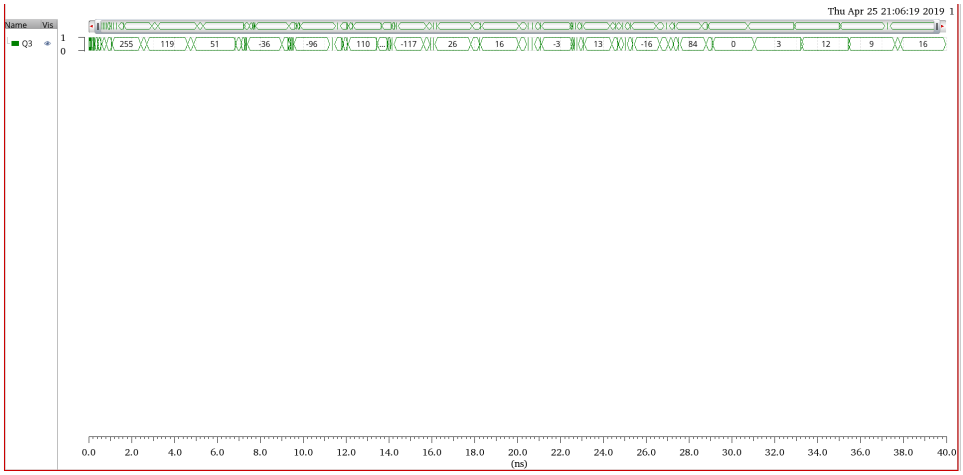

Figure 16: Output Q3

5.2 Functional Waveforms and results of Optimized Schematic

For this part, we need to check the functionality of our schematic. Then report the shortest time it required. For the optimized pipelined version, the best clock period is 2ns.The following pictures show the schematic simulation results.

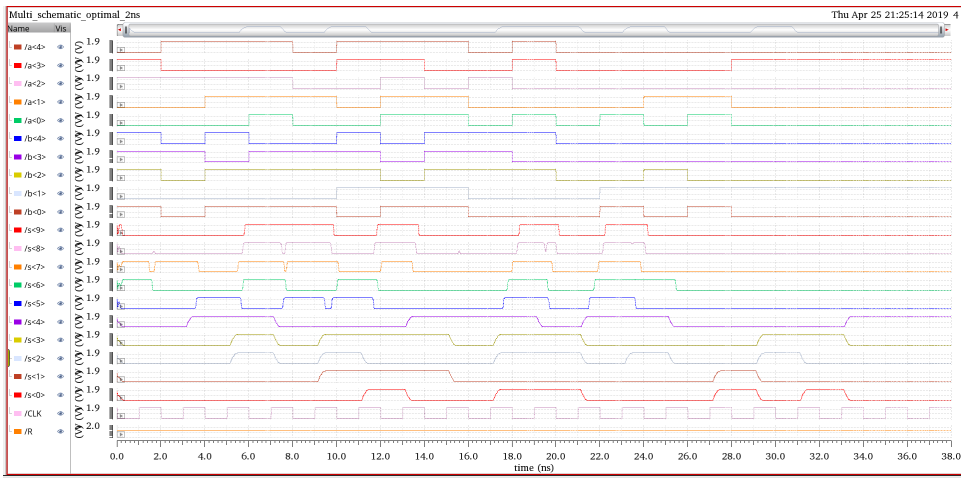

Figure 17: Final simulation for the optimal pipelined version

In order to give the direct view of our result, We create a bus for our input a and b, and output s.

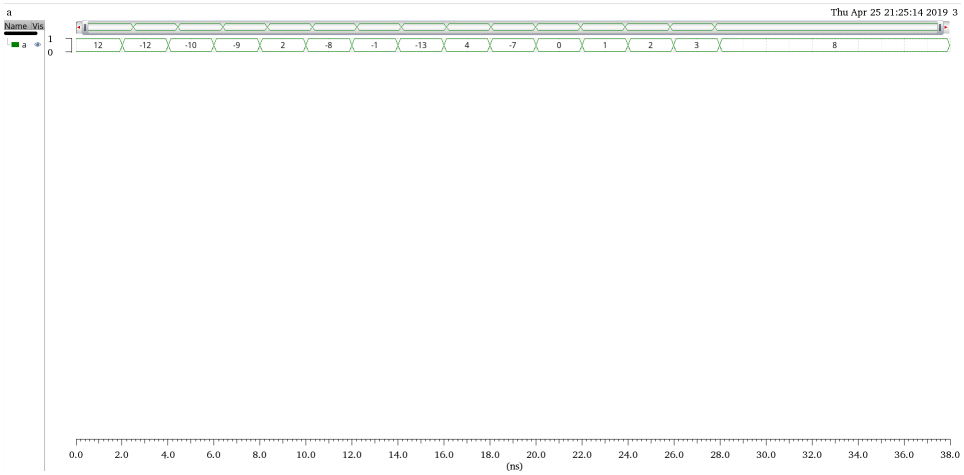

Figure 18: Input a

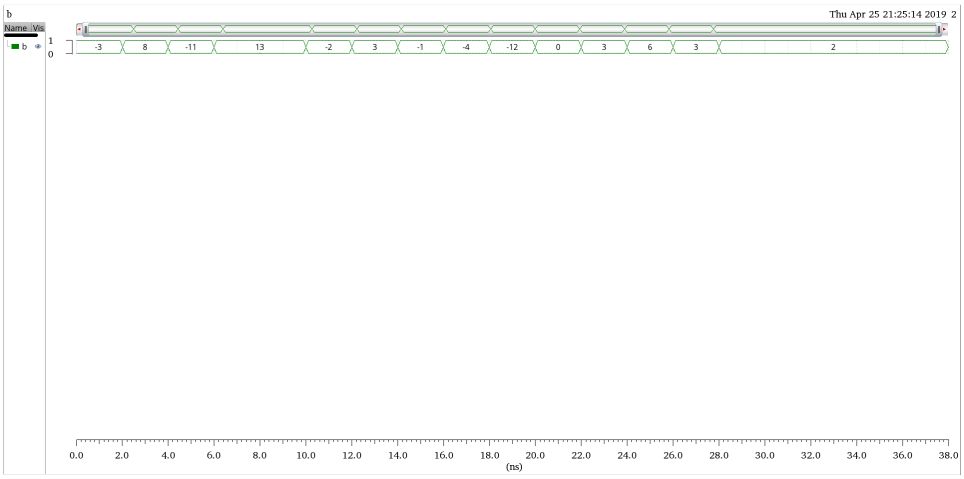

Figure 19: Input b

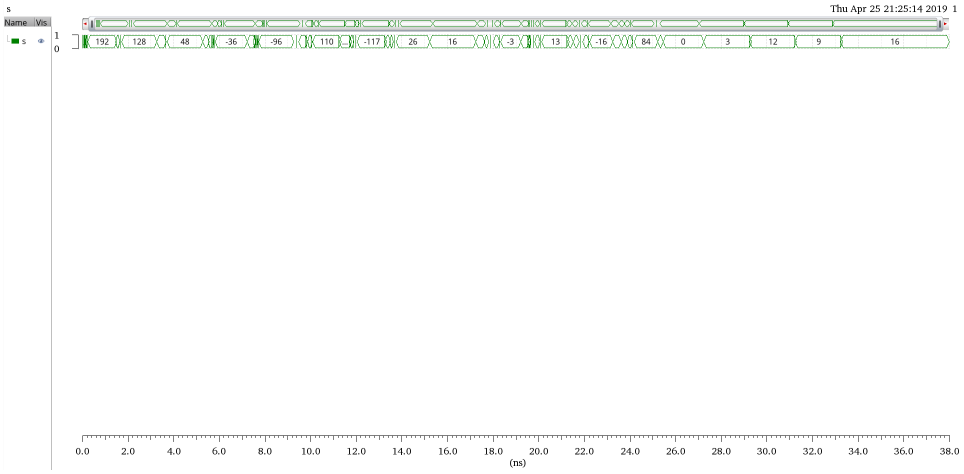

Figure 20: Output Q3

5.3 Functional Waveforms and results of Optimized Layout

For this part, we need to check the functionality of our layout. Then report the shortest time it required. For the optimized pipelined version, the best clock period is 2ns.The following pictures show the layout simulation results.

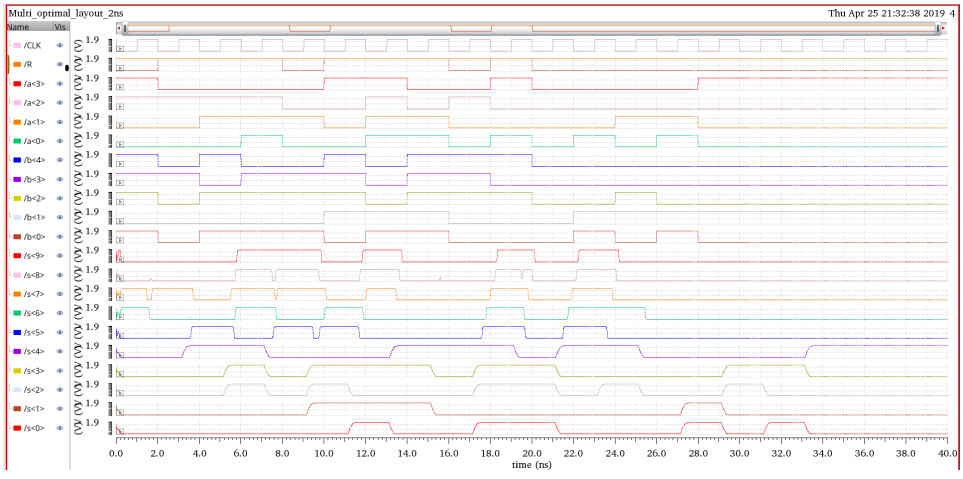

Figure 21: Final simulation for the optimal pipelined version-layout

In order to give the direct view of our result, We create a bus for our input a and b, and output s.

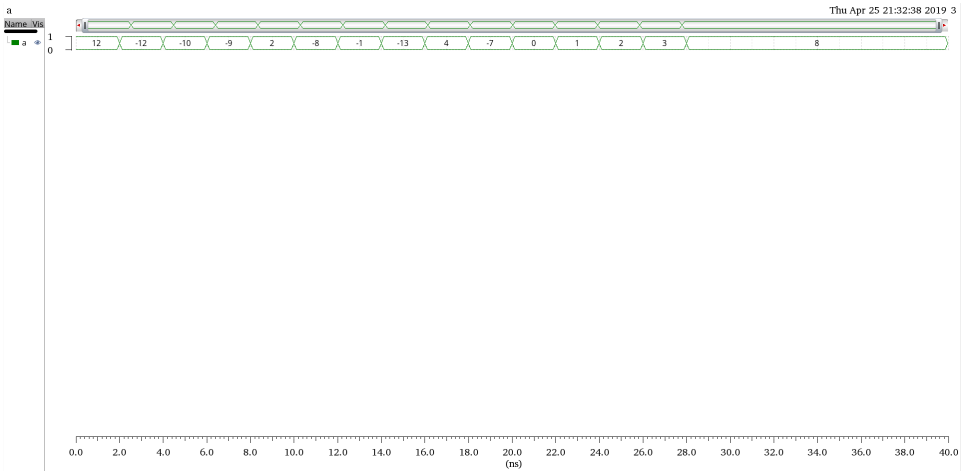

Figure 22: Input a

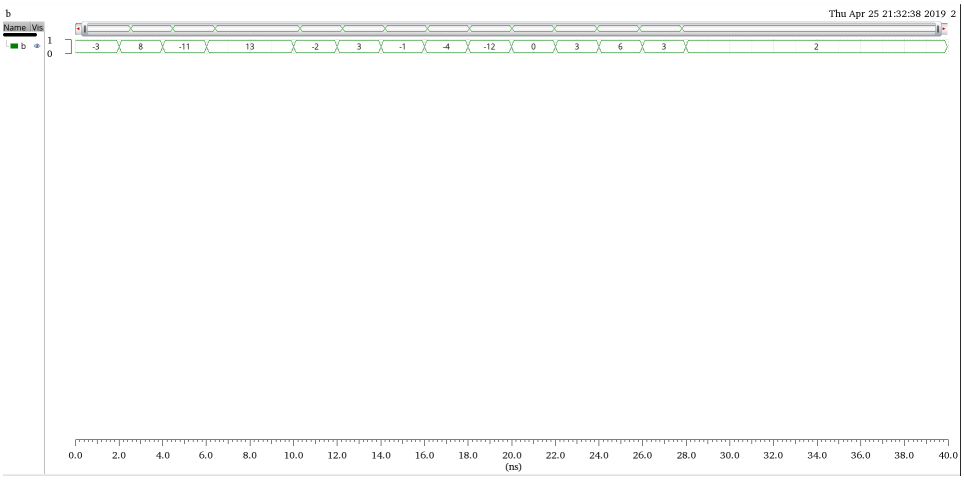

Figure 23: Input b

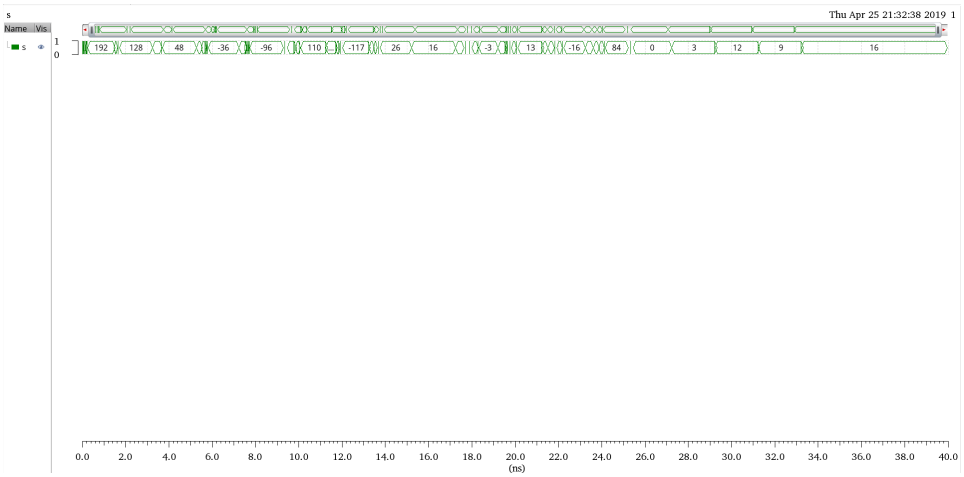

Figure 24: Output Q3

5.4 Data Analysis

The table below show the summary for the layout dimension.

Table 1: Layout dimension summary

Height	114u
Width	120u

Table 2: Clock for Multiplier

	Schematic	**Optimized Schematic**	**Optimized Layout**
Clock(ns)	2.2	2.0	2.0

After optimization, the clock is improved by 0.2ns.
Our golden result for input a and b is :

```
−36
−96
110
−117
26
16
−3
13
−16
84
0
3
```

```
12
9
16
```

Code Listing 4: golden result from python

Our simulation result which we extract from our CSV file is:

```
−36
−96
110
−117
26
16
−3
13
−16
84
0
3
12
9
16
```

Code Listing 5: simulation result from CSV file

For the comparison with the CSV file, we generate the result file, which will contain all the SAME characters. Here is the CSV file exported from cadence and the result content:

```
(s),Q3
0.000,0000000000
2.643p,1111110000
2.807p,1111111111
11.00p,0000001111
21.15p,0000000000
66.91p,0111100000
77.10p,1111100000
82.61p,1111110000
118.1p,1111010000
136.3p,1010011111
148.6p,1000011111
215.4p,1010011111
232.8p,0010011111
263.7p,0011011111
297.6p,0011001111
397.4p,0010001111
408.1p,0010011111
416.8p,0000011111
541.9p,0100011111
619.8p,0100111111
833.5p,0101111111
1.044n,0111111111
```

```
1.087n,0011111111
2.420n,0011110111
2.709n,0001110111
4.614n,0001110011
4.901n,0000110011
6.813n,0000110000
6.841n,0000111100
7.103n,0010111100
7.258n,0010011100
7.327n,0110011100
7.337n,0111011100
7.406n,1111011100
9.013n,1111010000
9.293n,1111000000
9.299n,1011000000
9.417n,1001000000
9.471n,1101000000
9.500n,1100000000
9.516n,1100100000
9.559n,1110100000
11.24n,1110101110
11.47n,1110001110
11.77n,1010001110
11.79n,1010101110
11.88n,0010101110
12.04n,0011101110
12.10n,0001101110
13.41n,0001101010
13.45n,0001101011
13.90n,0001001011
13.93n,0101001011
14.07n,1101001011
14.09n,1100001011
14.25n,1110001011
15.62n,1110001010
15.86n,1100011010
16.02n,1000011010
16.14n,0000011010
17.81n,0000010000
18.17n,0100010000
18.27n,0000010000
20.05n,0000011101
20.41n,0000111101
20.60n,0001111101
20.81n,0011111101
21.04n,0111111101
21.13n,1111111101
22.50n,1011111101
22.59n,1011101101
22.60n,1111101101
22.62n,1110101101
```

```
22.71n,1110001101
22.84n,1100001101
23.03n,1000001101
23.13n,0000001101
24.41n,0000000000
24.63n,0000010000
24.71n,0000110000
24.95n,0001110000
25.13n,0011110000
25.35n,0111110000
25.45n,1111110000
26.64n,1111110100
27.03n,1111010100
27.27n,1101010100
27.48n,1001010100
27.58n,0001010100
28.81n,0001010000
29.07n,0000010000
29.10n,0000000000
31.05n,0000000011
33.22n,0000000000
33.24n,0000001100
35.42n,0000001000
35.44n,0000001001
37.62n,0000000000
37.86n,0000010000
40.00n,0000010000
```

Code Listing 6: CSV File from Cadence

```
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
SAME
```

Code Listing 7: Comparison Result