The University of Southern California

EE577A Project

Phase II Report

Huayu Fu

USC ID:4745-1598-48

Bowen Zhu

USC ID: 8364-5072-18

Brandon Butler

USC ID: 1002-6459-08

Contents

List of Figures

I. READ ME

The RF is designed with simple latch(a pass transistor and two inverters). It is able to perform write and read in the same register address within signal clock cycle.

ALU is designed to save OP code bit address, to be power efficient by using decoder to select functions, to save area by combining the functions. ADD and MIN are combined into one block with inverters to invert input B to perform ADD and MIN. The shift register can be reversely connected to switch between SFL/SFR. The Multiplier is pipelined. OR/AND functio is also combined and use transmission gate to save space and increase speed.

MEM stage contains the same design as lab2 and modified control signal, to precharge during the first half clock cycle and read or write during the second half clock cycle.

And the WB stage only contains a 2 to 1 16bit mux which is built from pass transistors.

II. Top-Level CPU

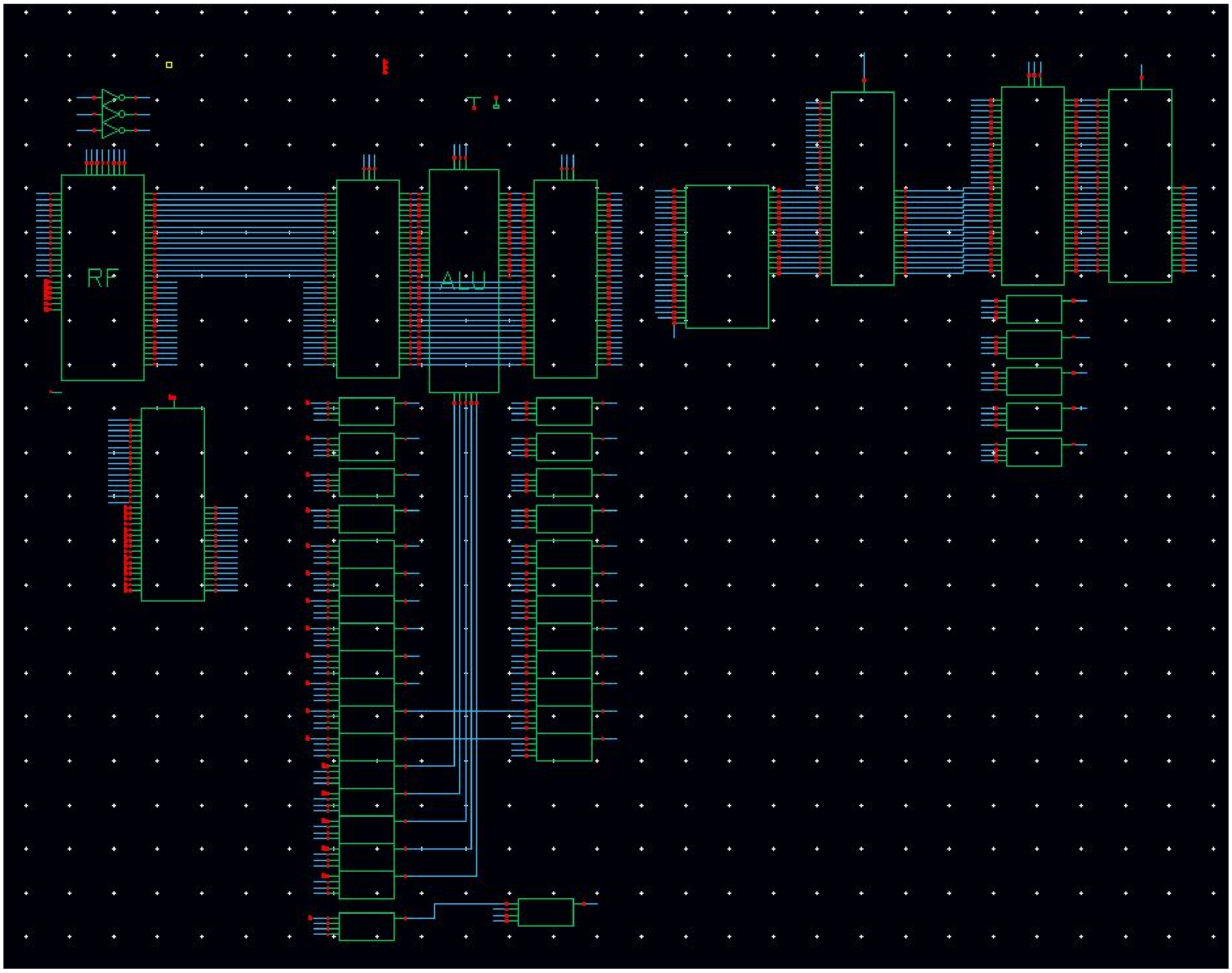


Figure 1. Top-Level CPU Schematic

III. IF/ID Stage Scheamtics

a. RF

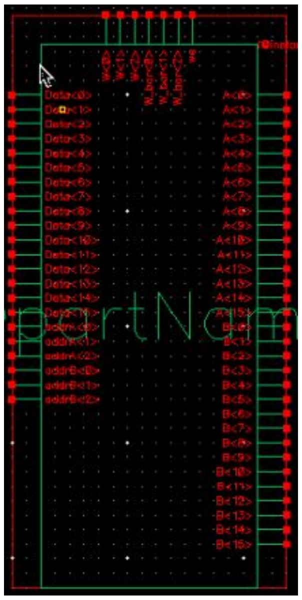


Figure 2. RF Symbol

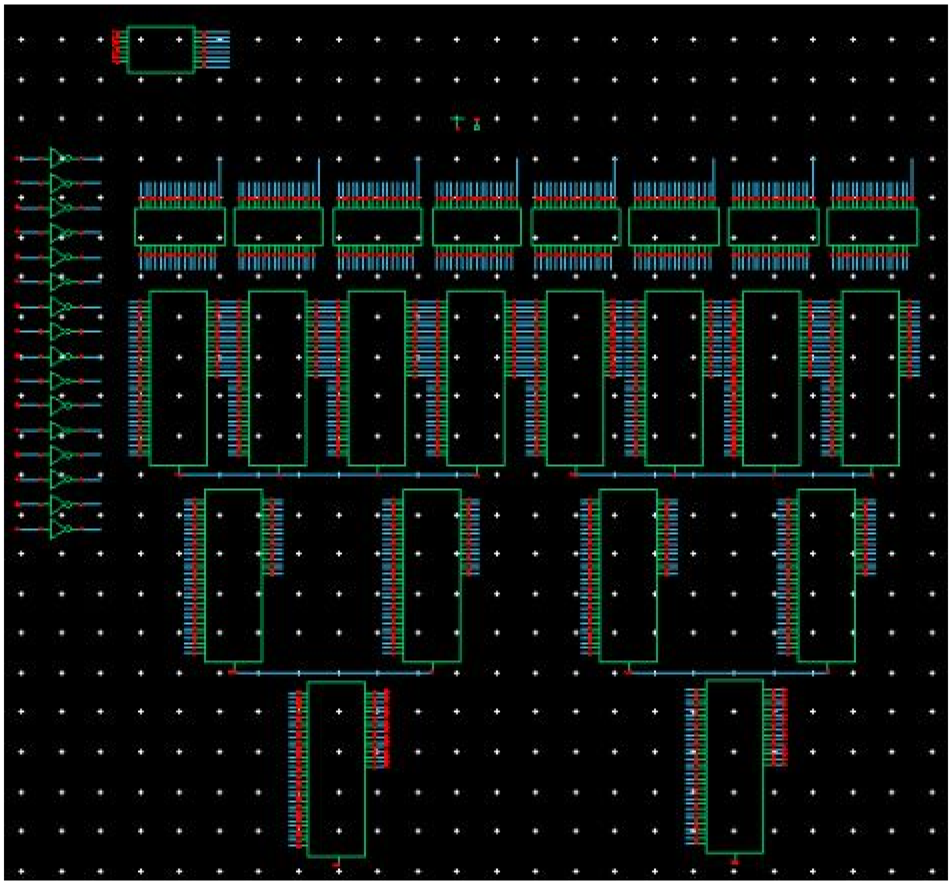


Figure 3. RF Schematic

i. 16-bit Register

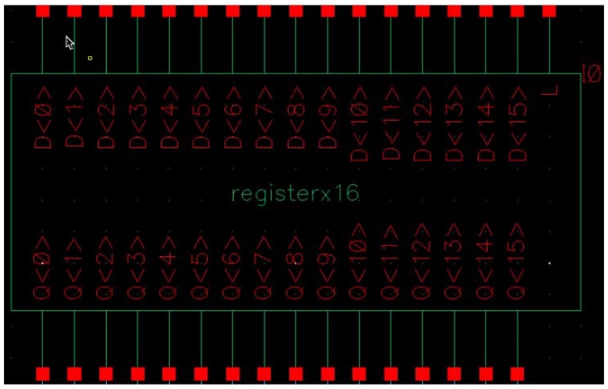


Figure 4. 16-Bit Register Symbol

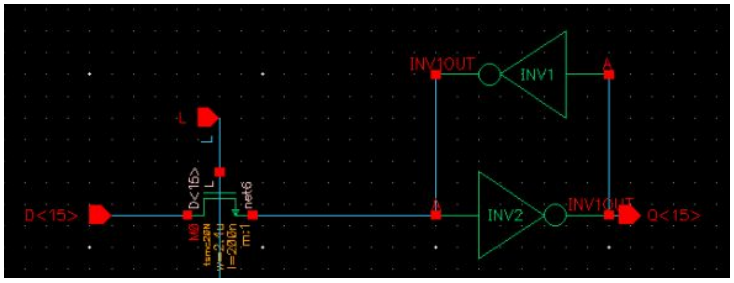


Figure 5. 16-Bit Register Schematic (1-bit shown)

ii. Inverter (INV3)

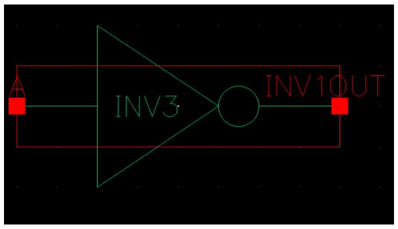


Figure 6. Inverter (INV3) Symbol

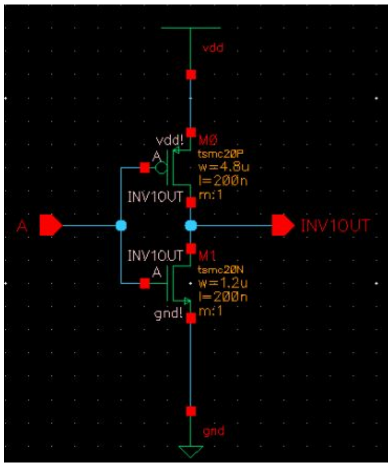


Figure 7. Inverter (INV3) Schematic

iii. Mux 2to1 16-bit

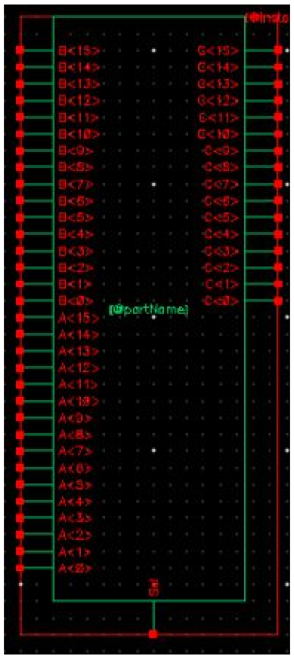
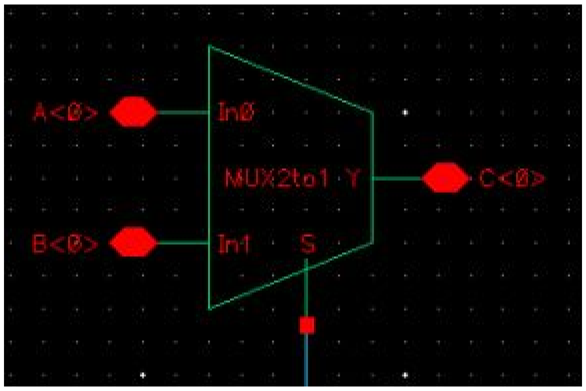



Figure 8,9. Mux 2to1 16-Bit Symbol and Schematic

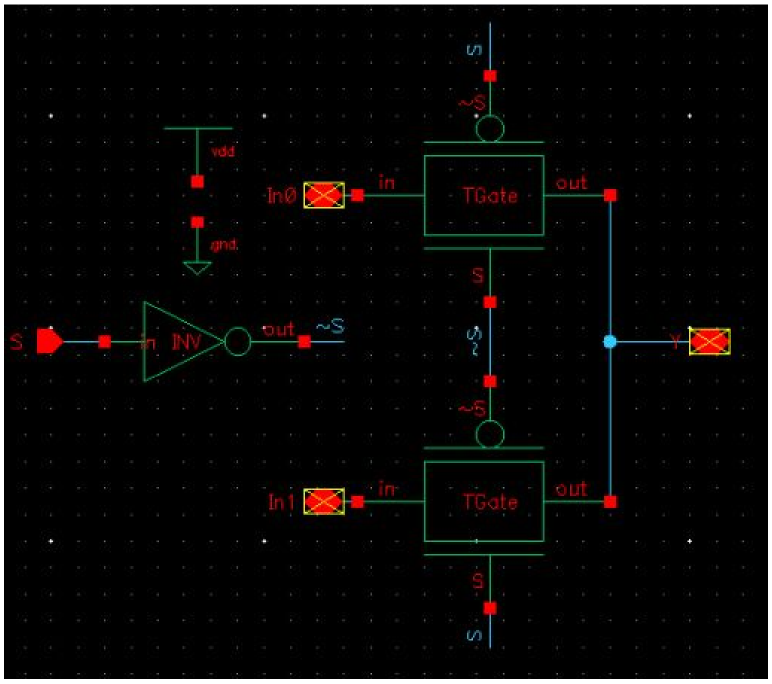


Figure 9. Mux 2to1 Schematic

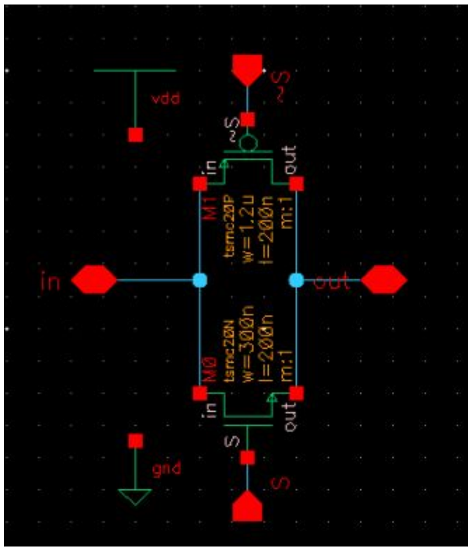


Figure 10. Transmission Gate Schematic

iv. 3-to-8 Decoder

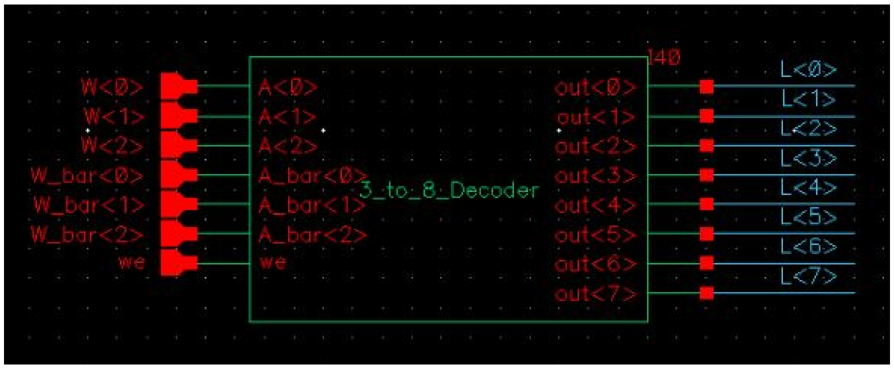


Figure 11. 3-to-8 Decoder Symbol

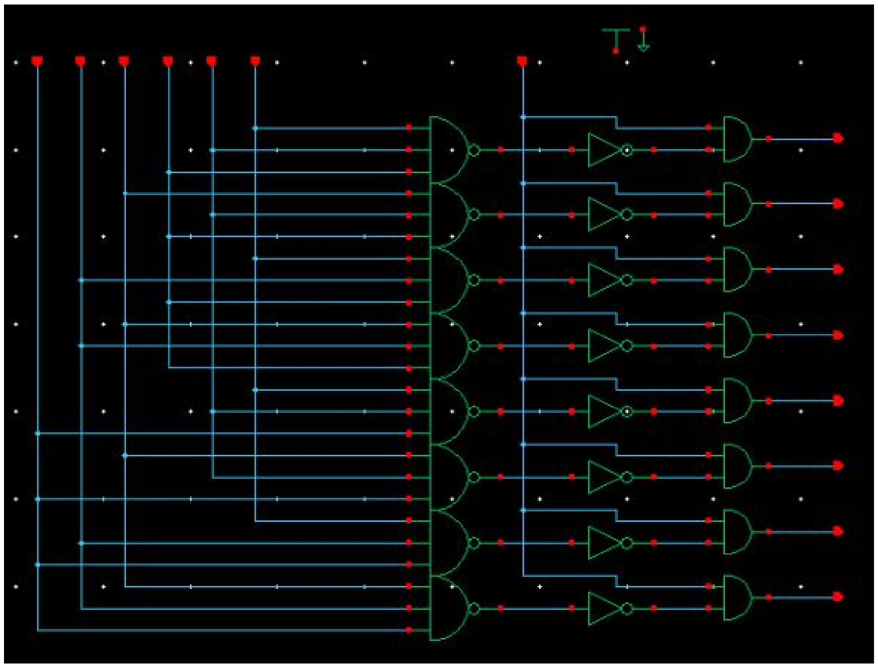


Figure 12. 3-to-8 Decoder Schematic

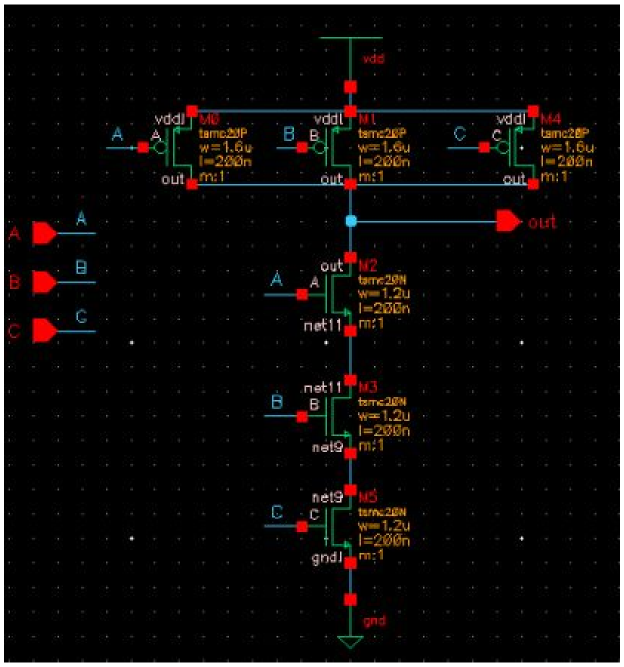


Figure 13. NAND3 Schematic

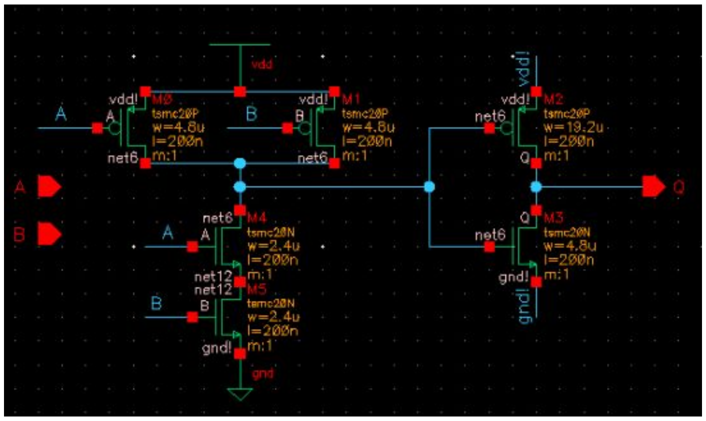


Figure 14. AND2 Schematic

IV. EX Stage

a. ALU

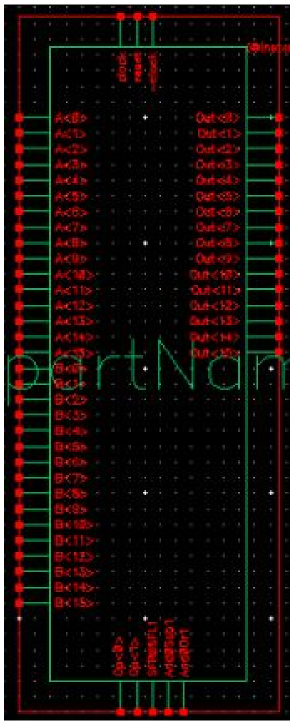


Figure 15. ALU Symbol

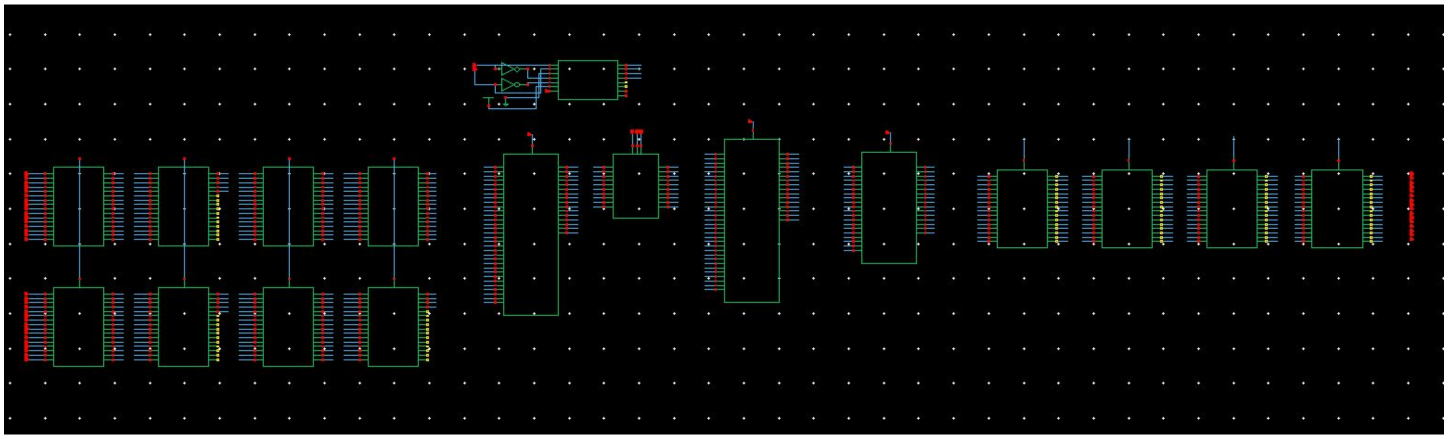


Figure 16. ALU Schematic

i. AND/OR 16-bit

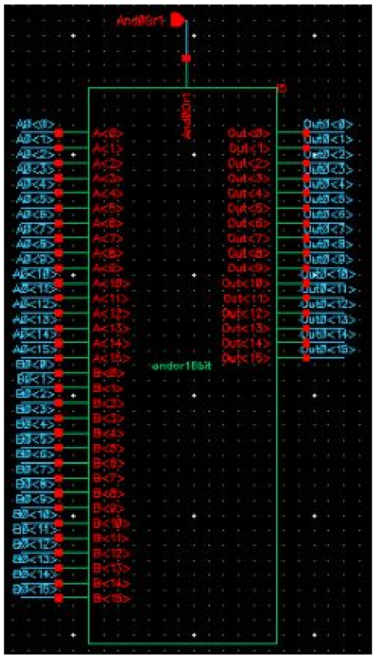


Figure 17. AND/OR 16-bit Symbol

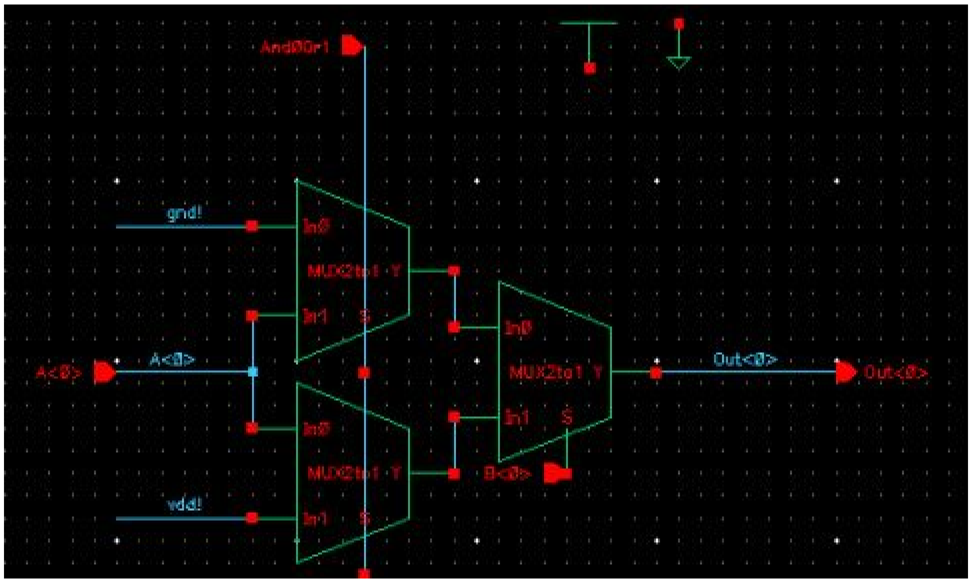


Figure 18. AND/OR 16-bit Schematic (1 bit shown)

ii. MUL

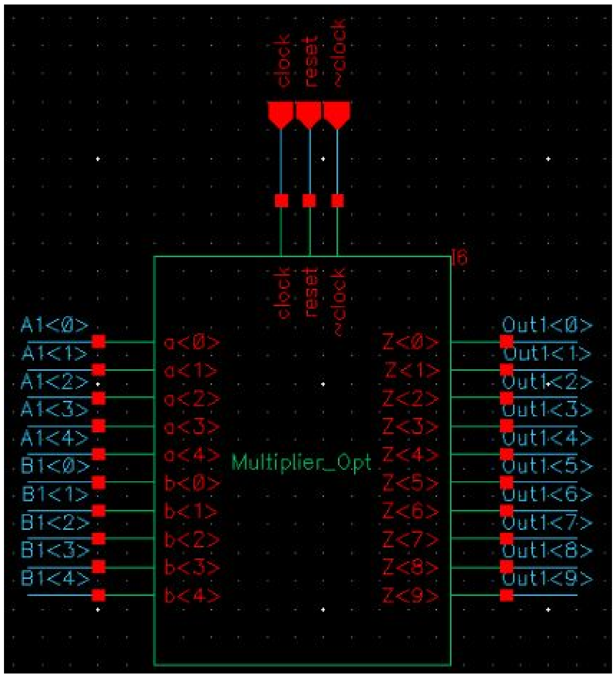


Figure 19. Multiplier Symbol

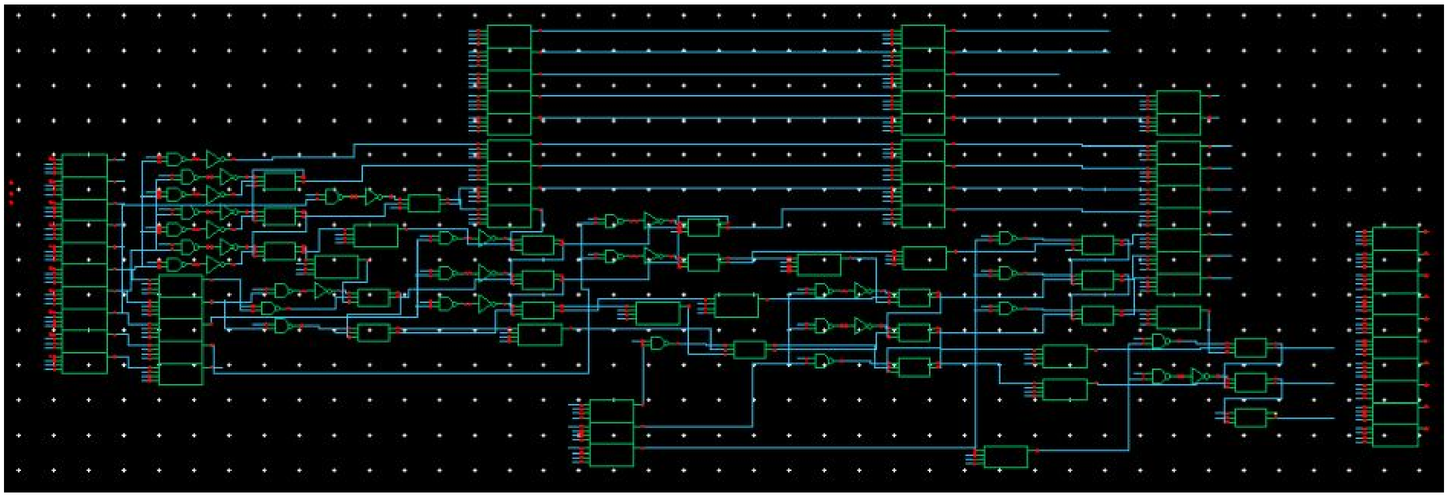


Figure 20. Multiplier Schematic (Pipelined)

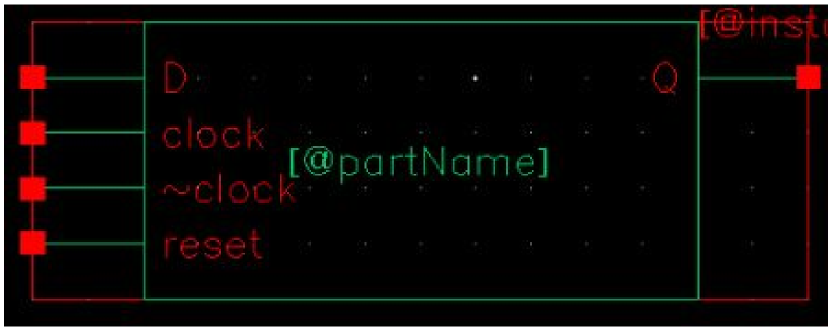


Figure 21. DFF Symbol

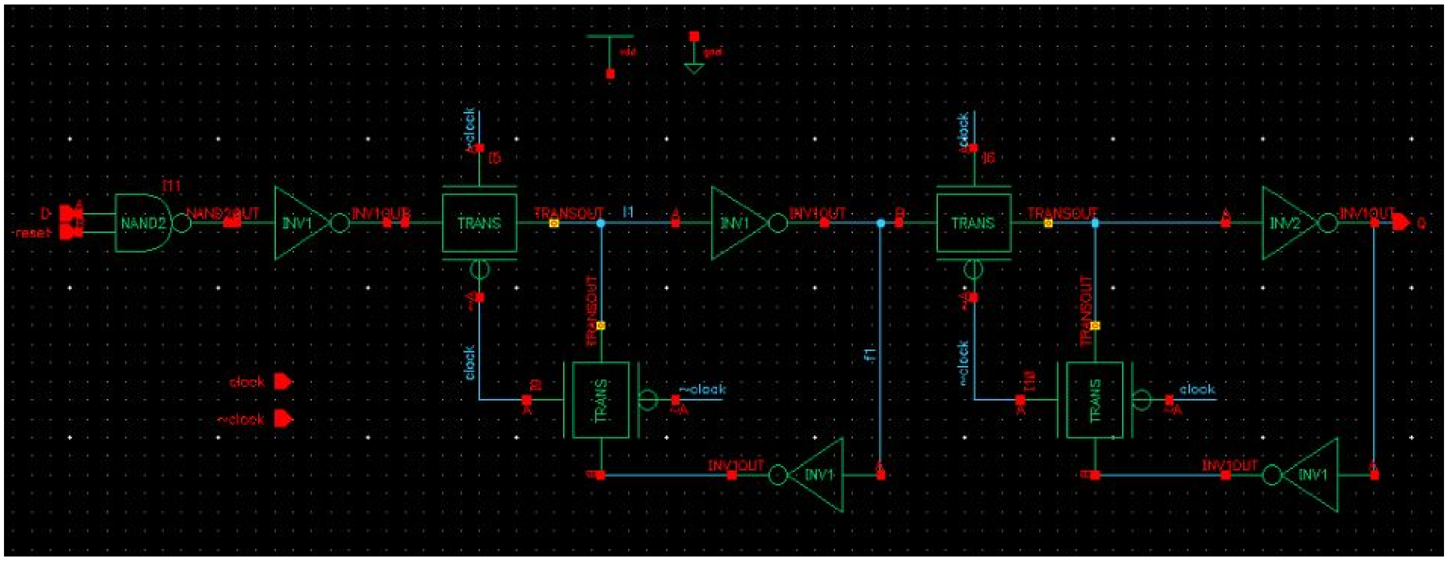


Figure 22. DFF Schematic

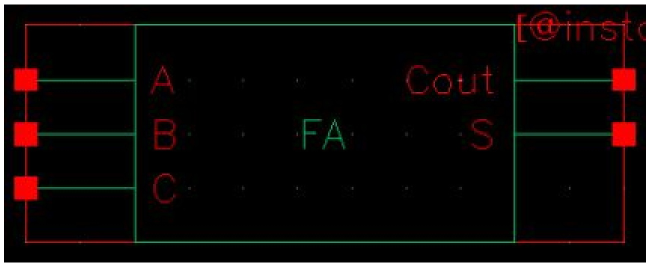


Figure 23. Standard Full Adder Symbol

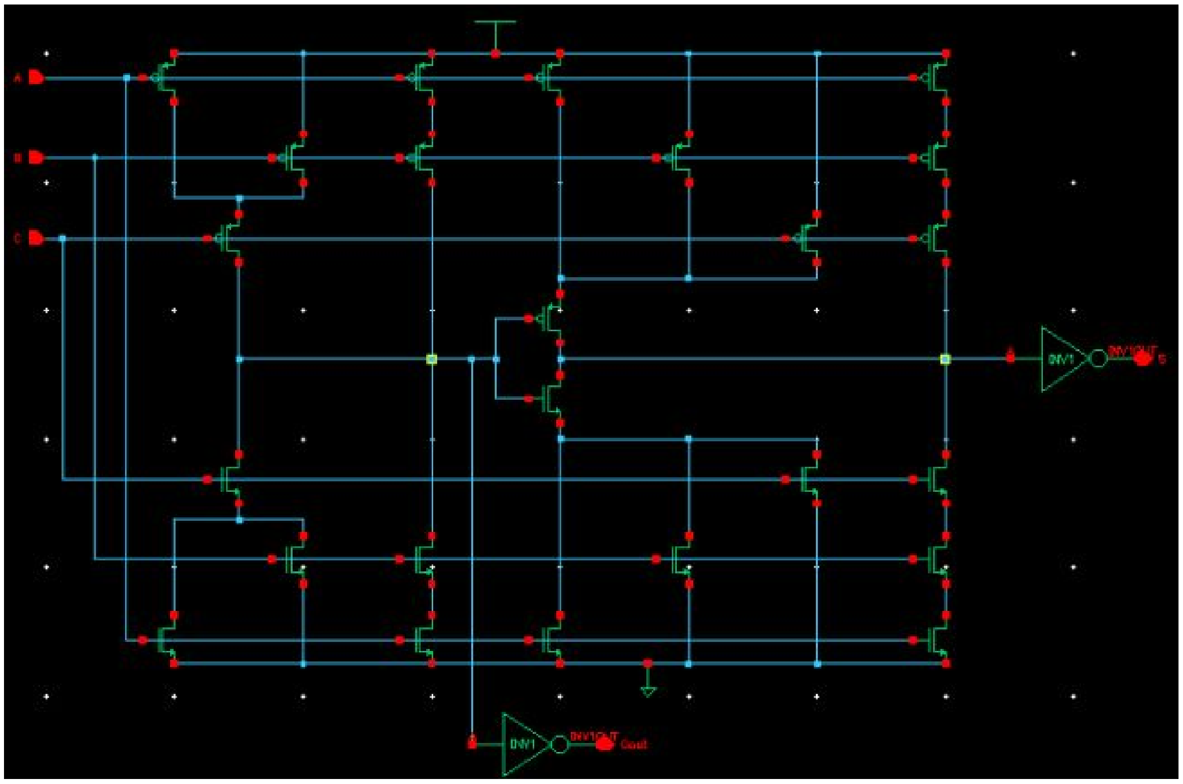


Figure 24. Standard Full Adder Schematic

iii. ADD

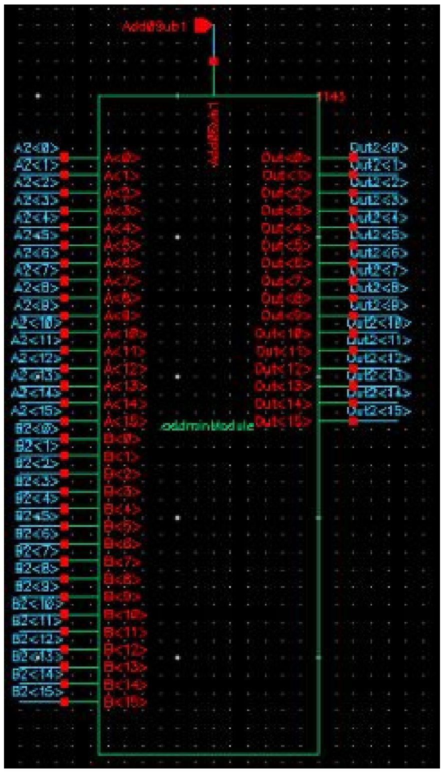


Figure 25. ADD Symbol

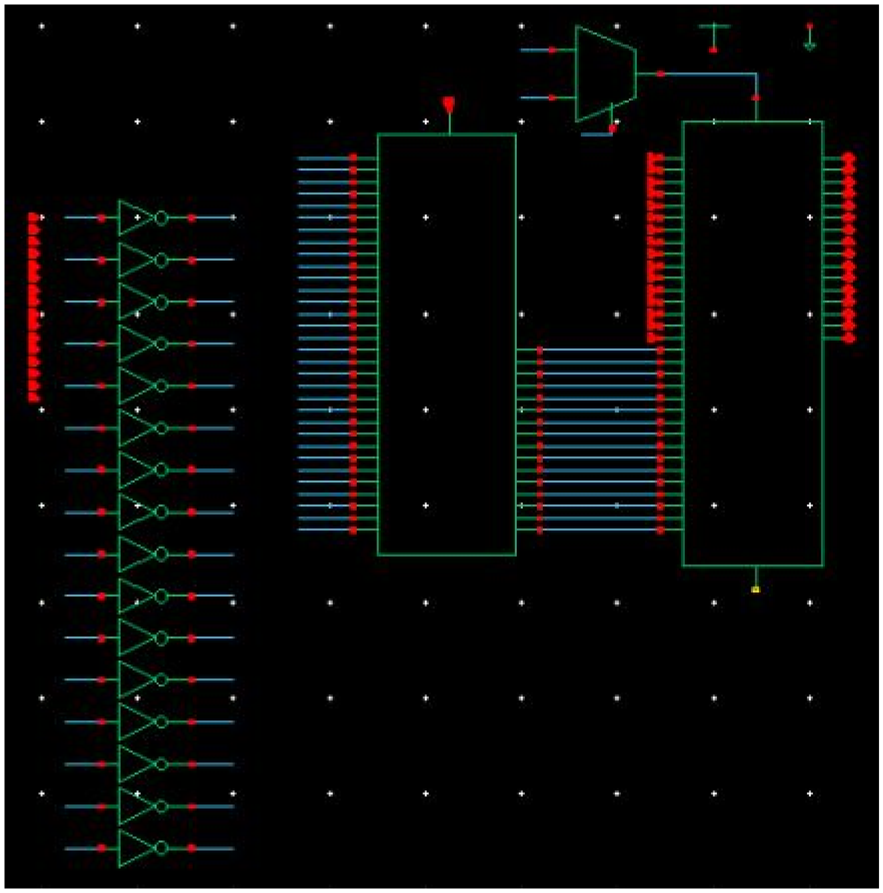


Figure 26. ADD Schematic

iv. SHIFT

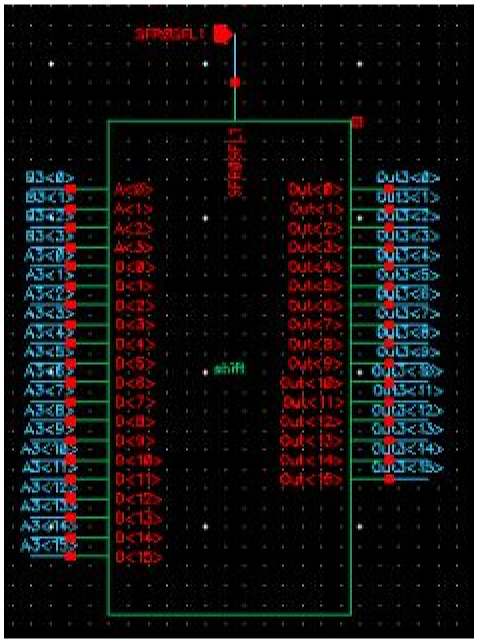


Figure 27. SFL/SFR Symbol

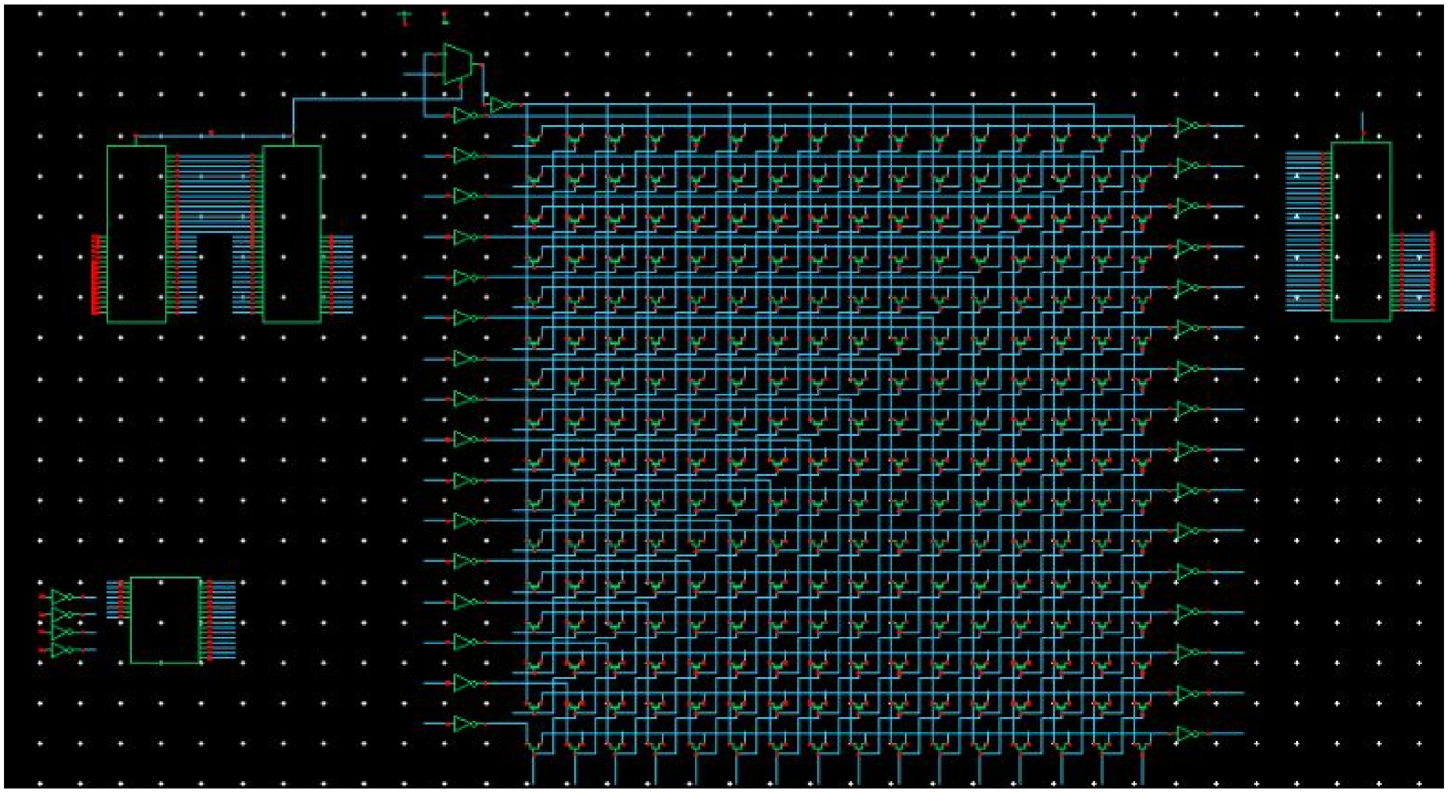


Figure 28. SFL/SFR Schematic

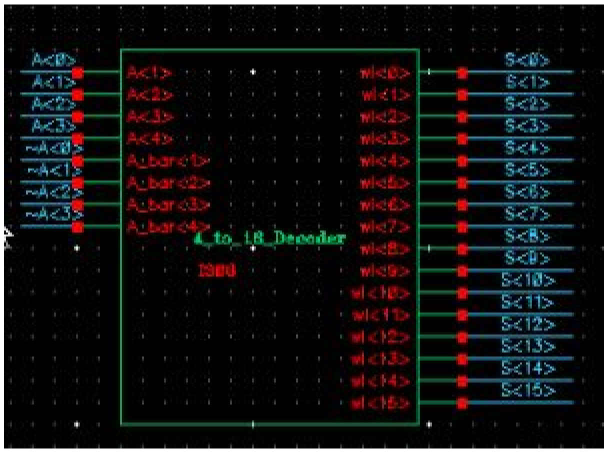


Figure 29. 4-to-16 Decoder Symbol

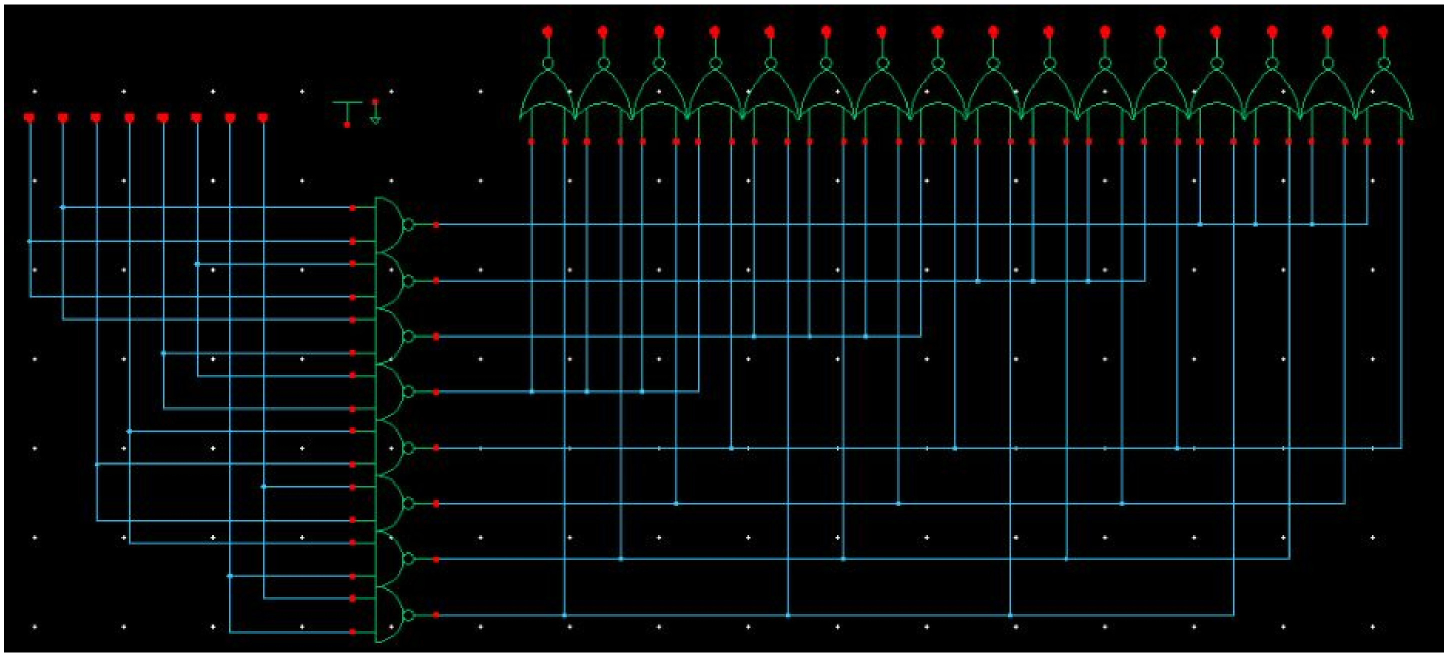


Figure 30. 4-to-16 Decoder Schematic

V. MEM Stage

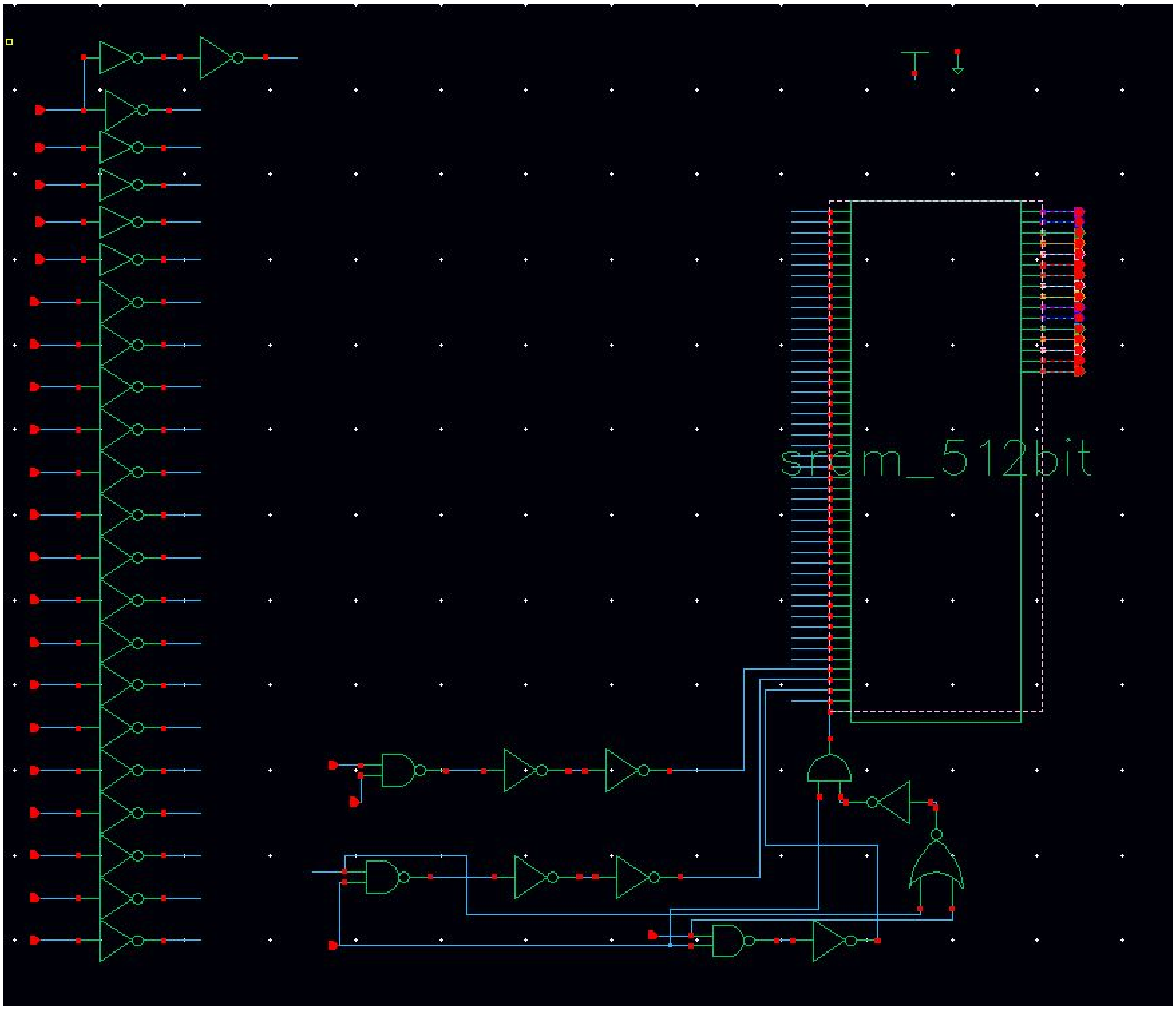


Figure 31. SRAM 512-Bit Symbol

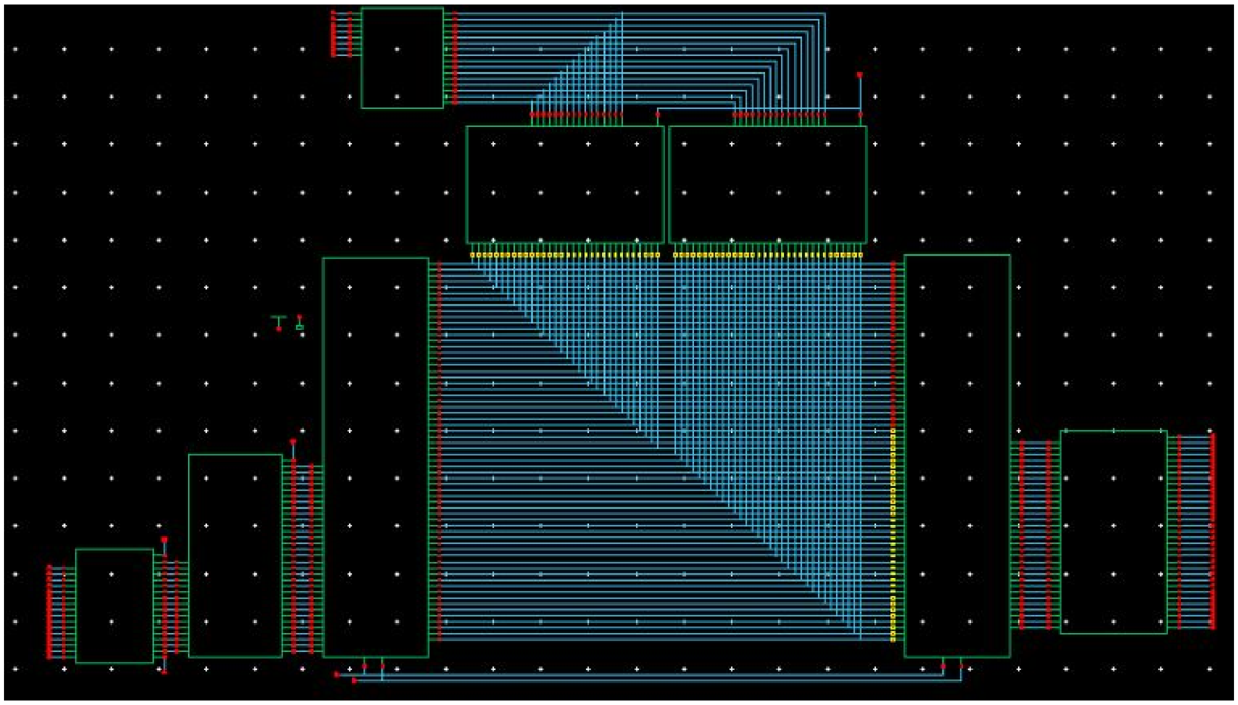


Figure 32. SRAM 512-Bit Schematic

4-to-16 Decoder used in SRAM schematic. See Figures 30 and 31 for Decoder symbol/schematic.

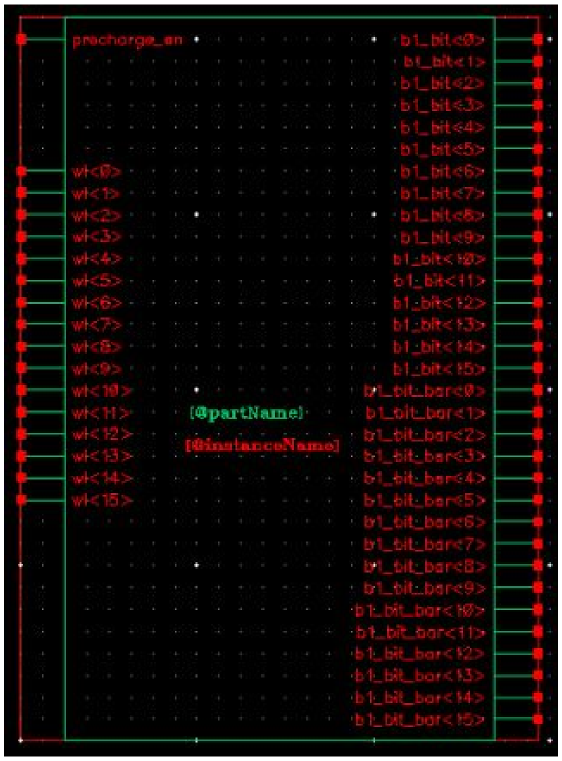


Figure 33. SRAM Bank Symbol

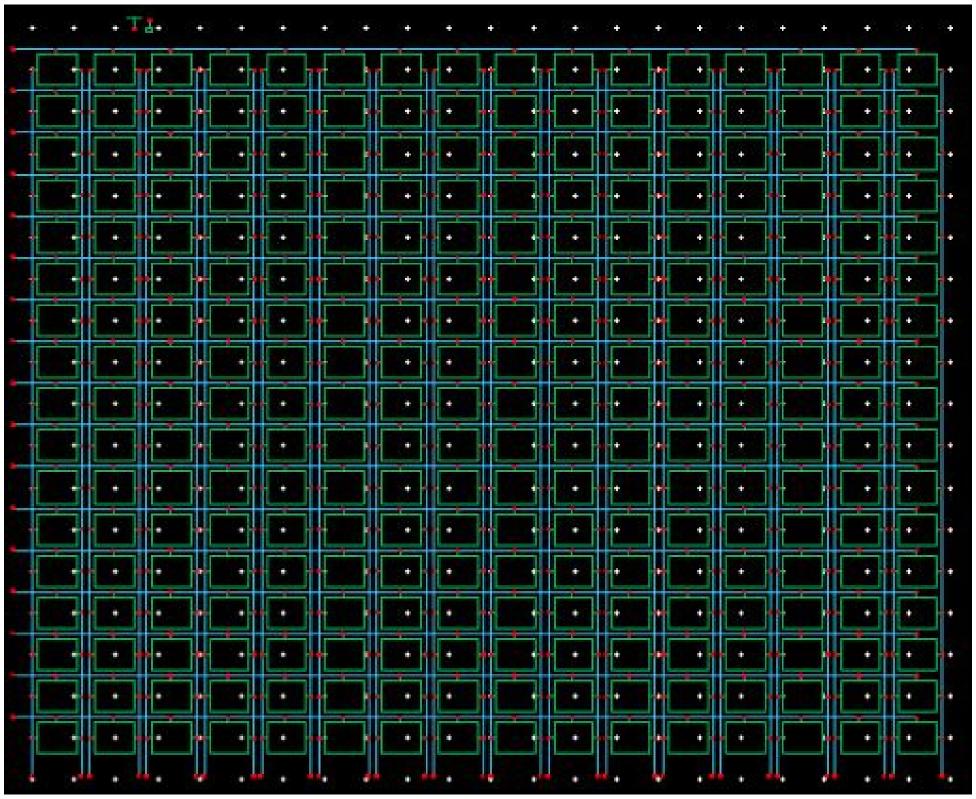


Figure 34. SRAM Bank Schematic

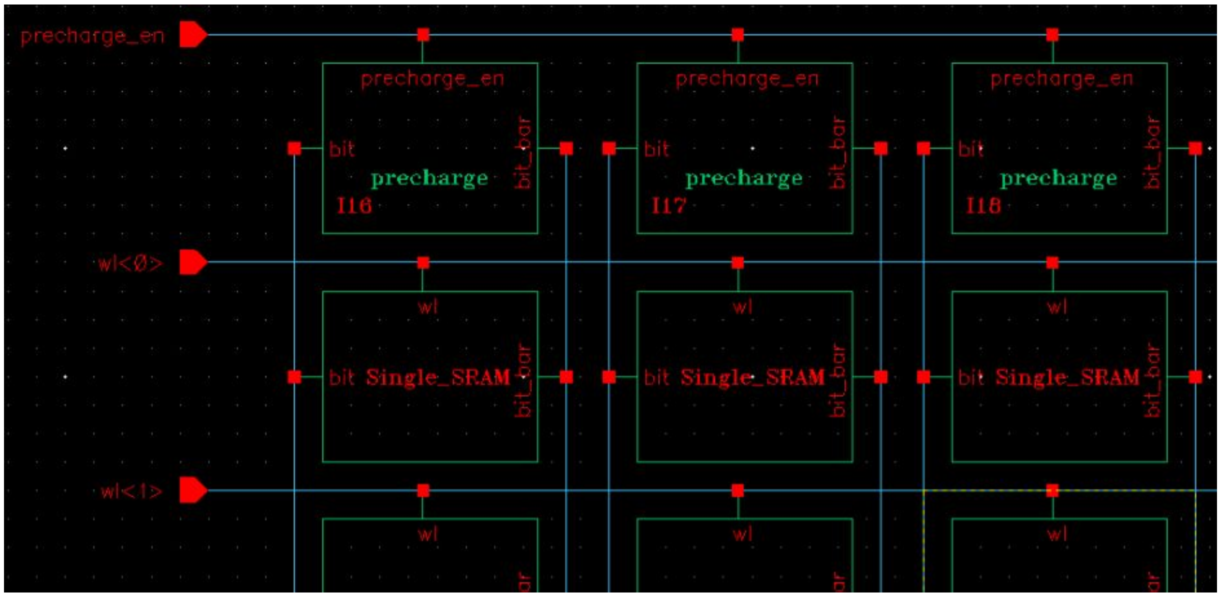


Figure 35. Zoom-In Upper-Left-Hand-Corner of SRAM Bank

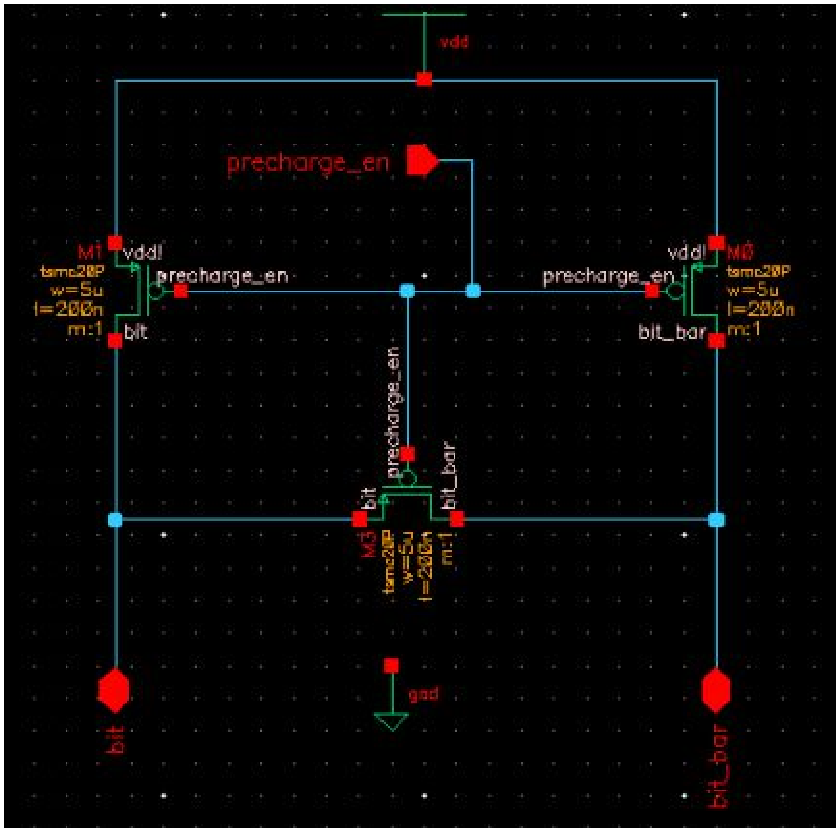


Figure 36. Precharge Schematic

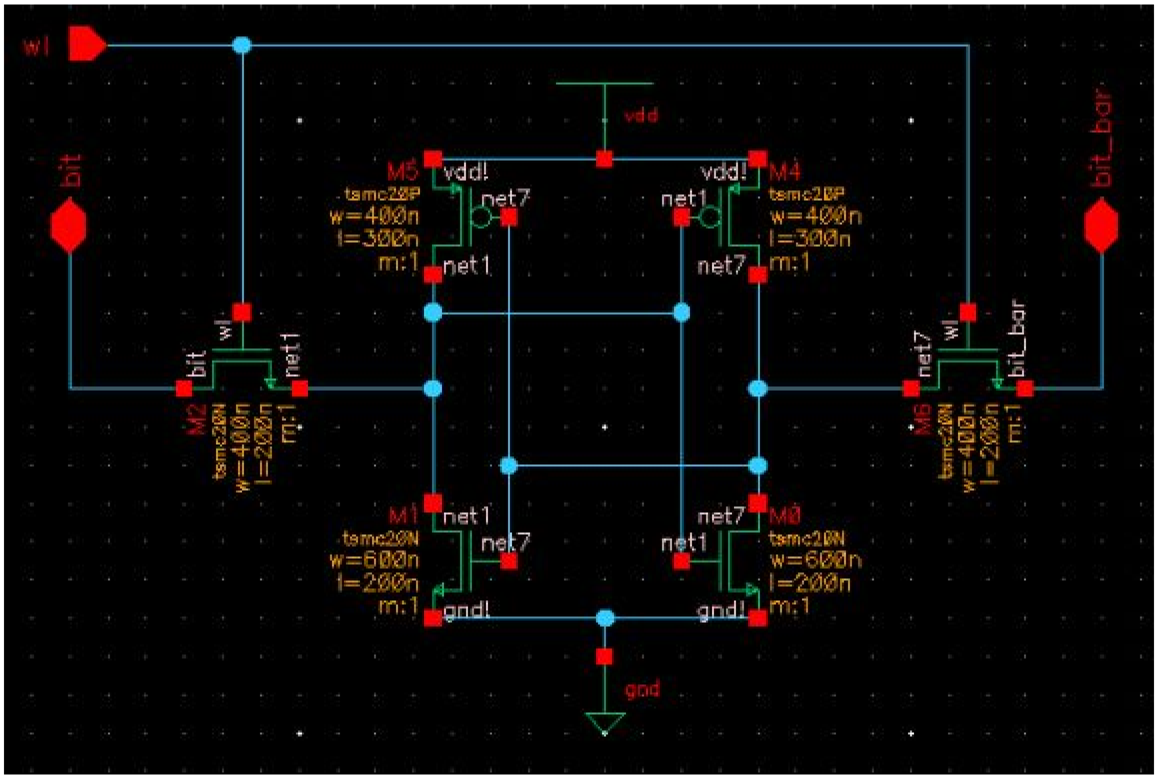


Figure 37. Single SRAM Cell Schematic

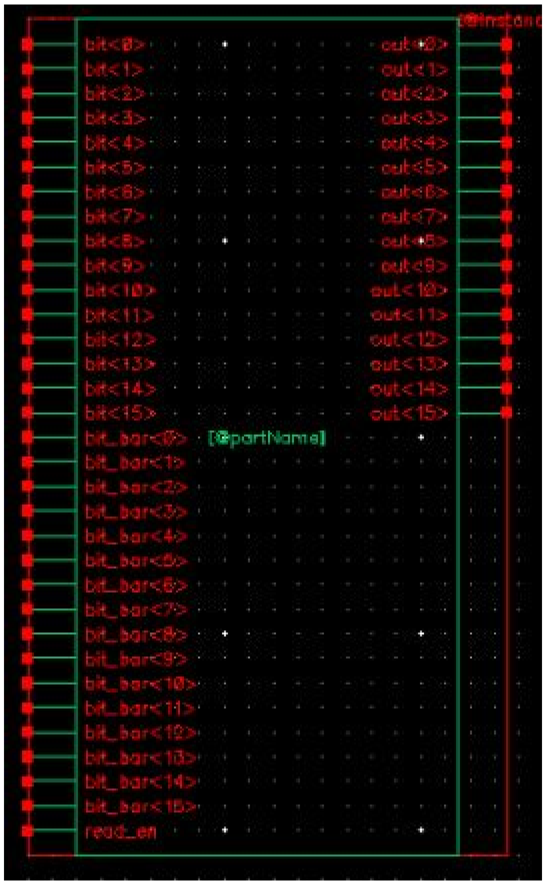


Figure 38. Sense Amp 16-bit Symbol

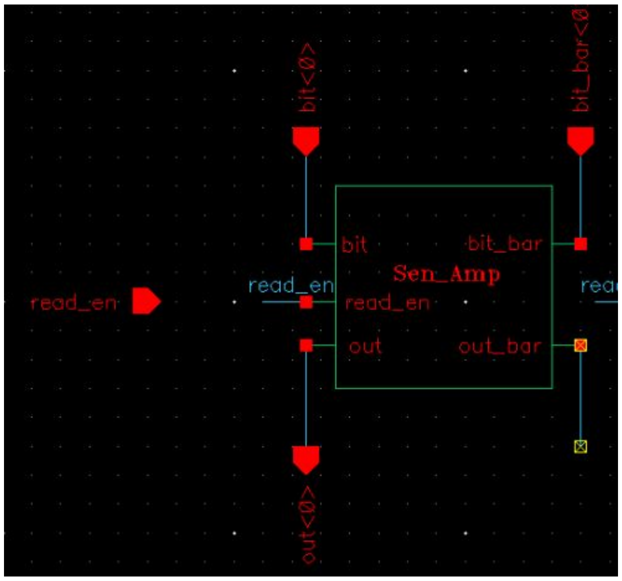


Figure 39. Sense Amp 16-bit (1 bit shown)

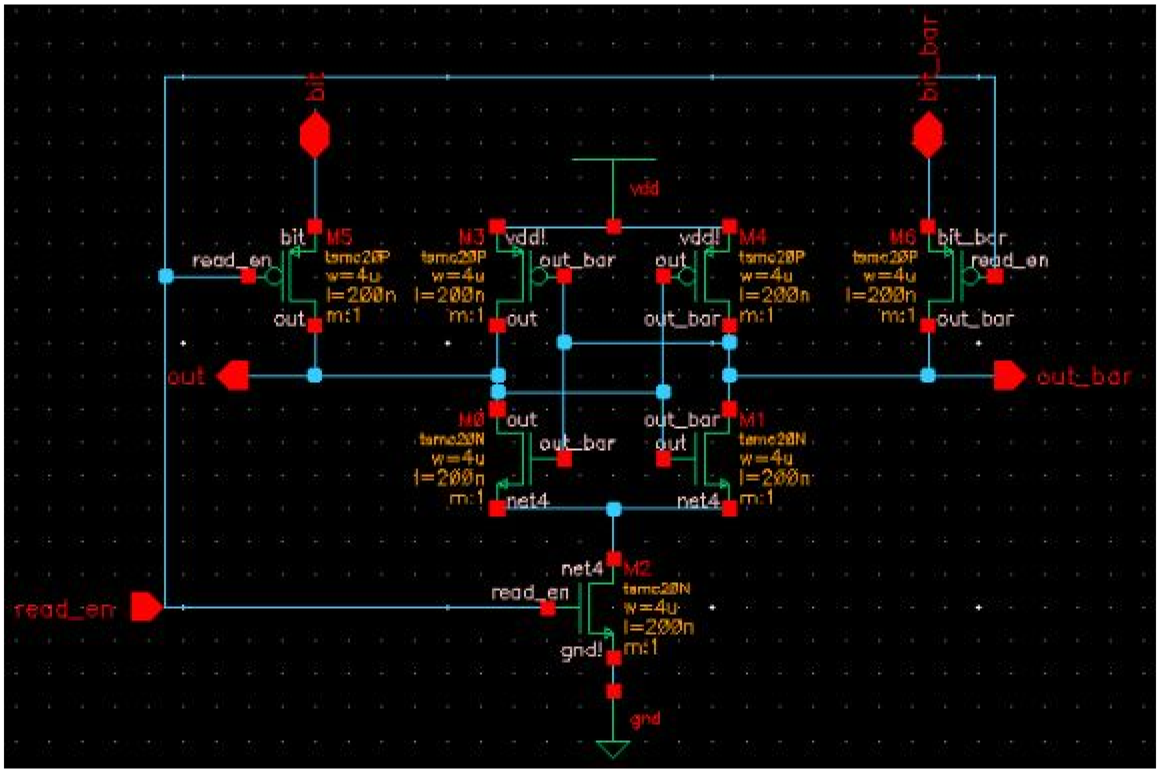


Figure 40. Sense Amp 1-bit Schematic

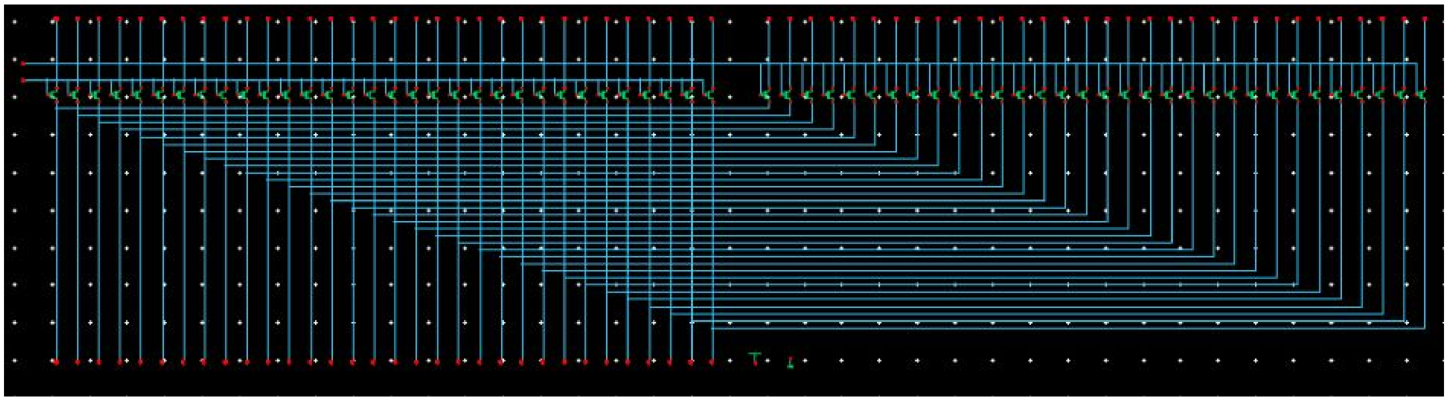


Figure 41. Read Mux Schematic

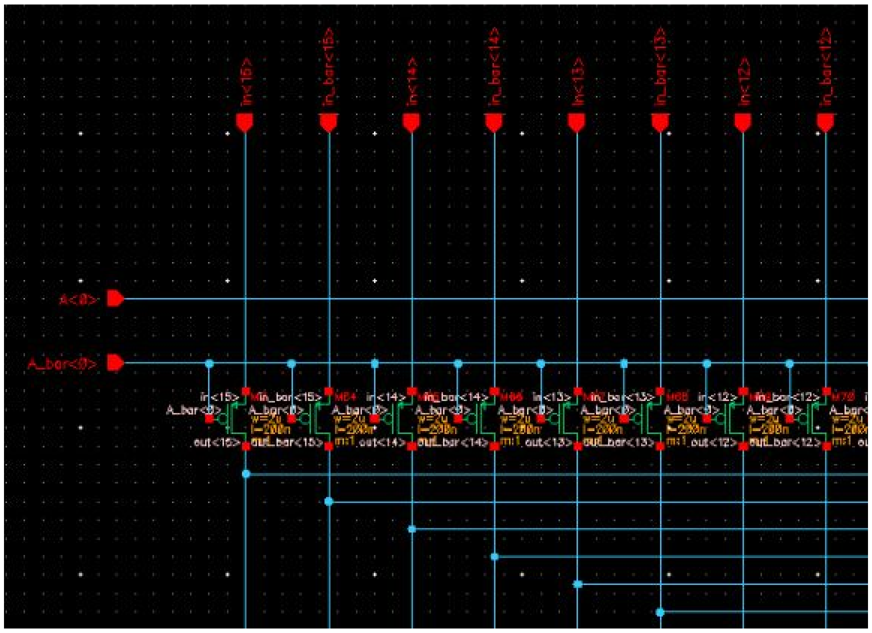


Figure 42. Zoom-In of Read Mux Schematic

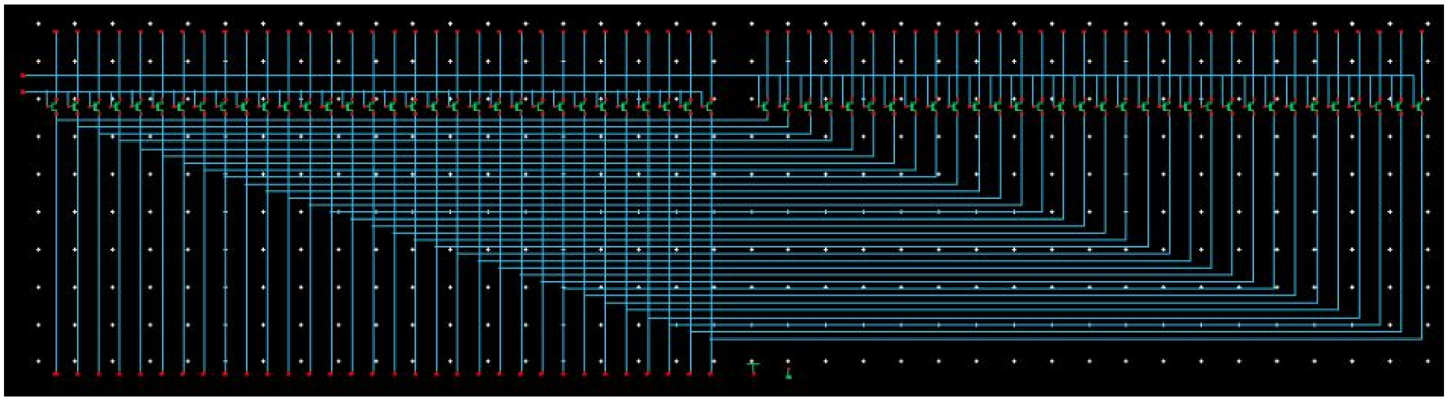


Figure 43. Write Mux Schematic

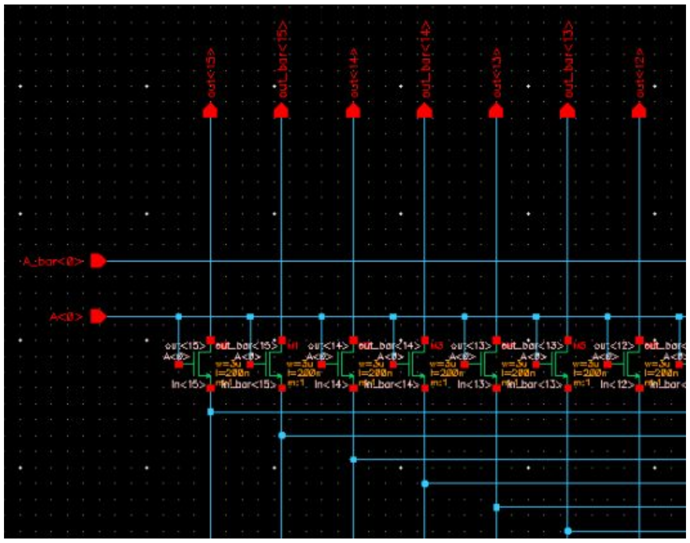


Figure 44. Zoom-In of Write Mux Schematic

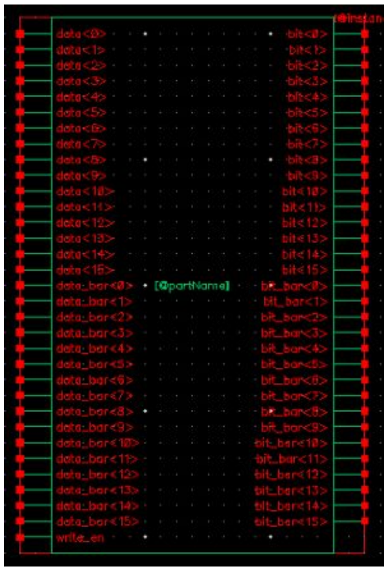


Figure 45. Write Path 16-bit Symbol

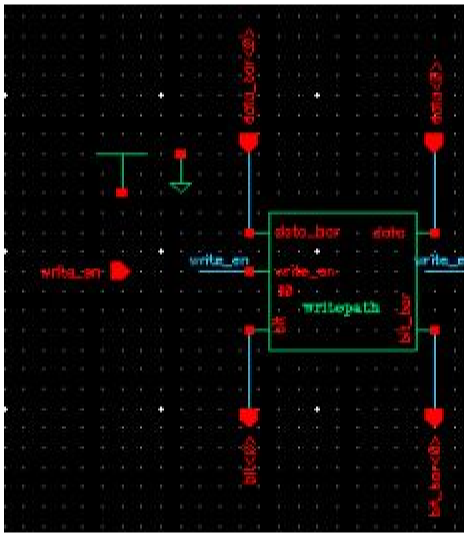


Figure 46. Write Path 16-bit Schematic (showing 1-bit)

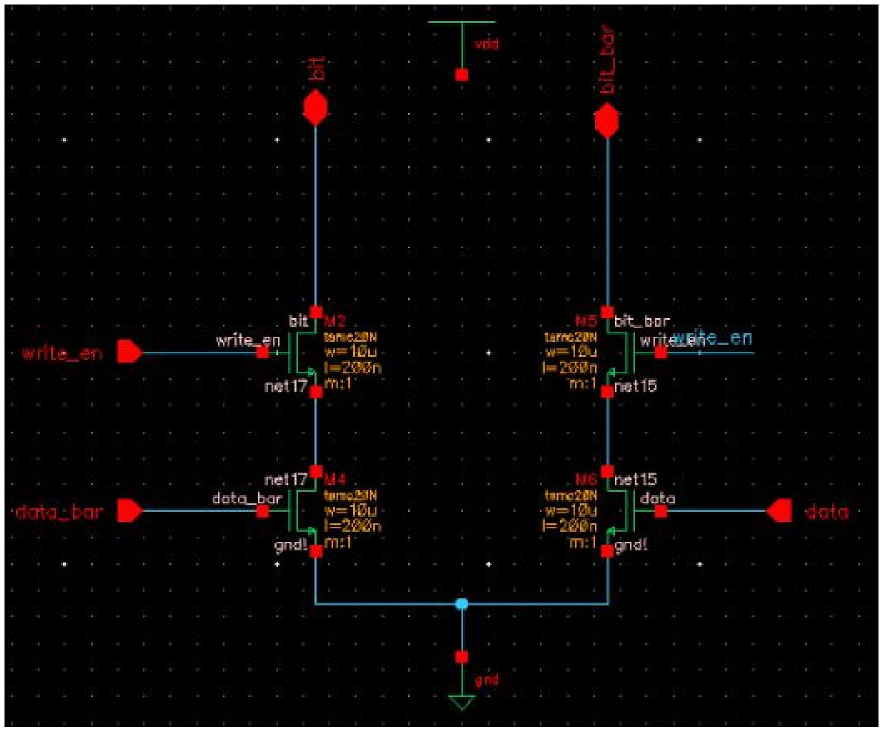


Figure 47. Write Path 1-bit Schematic

See Figures 22 and 23 for D Flip Flop used to register outputs of SRAM.

VI. WB Stage

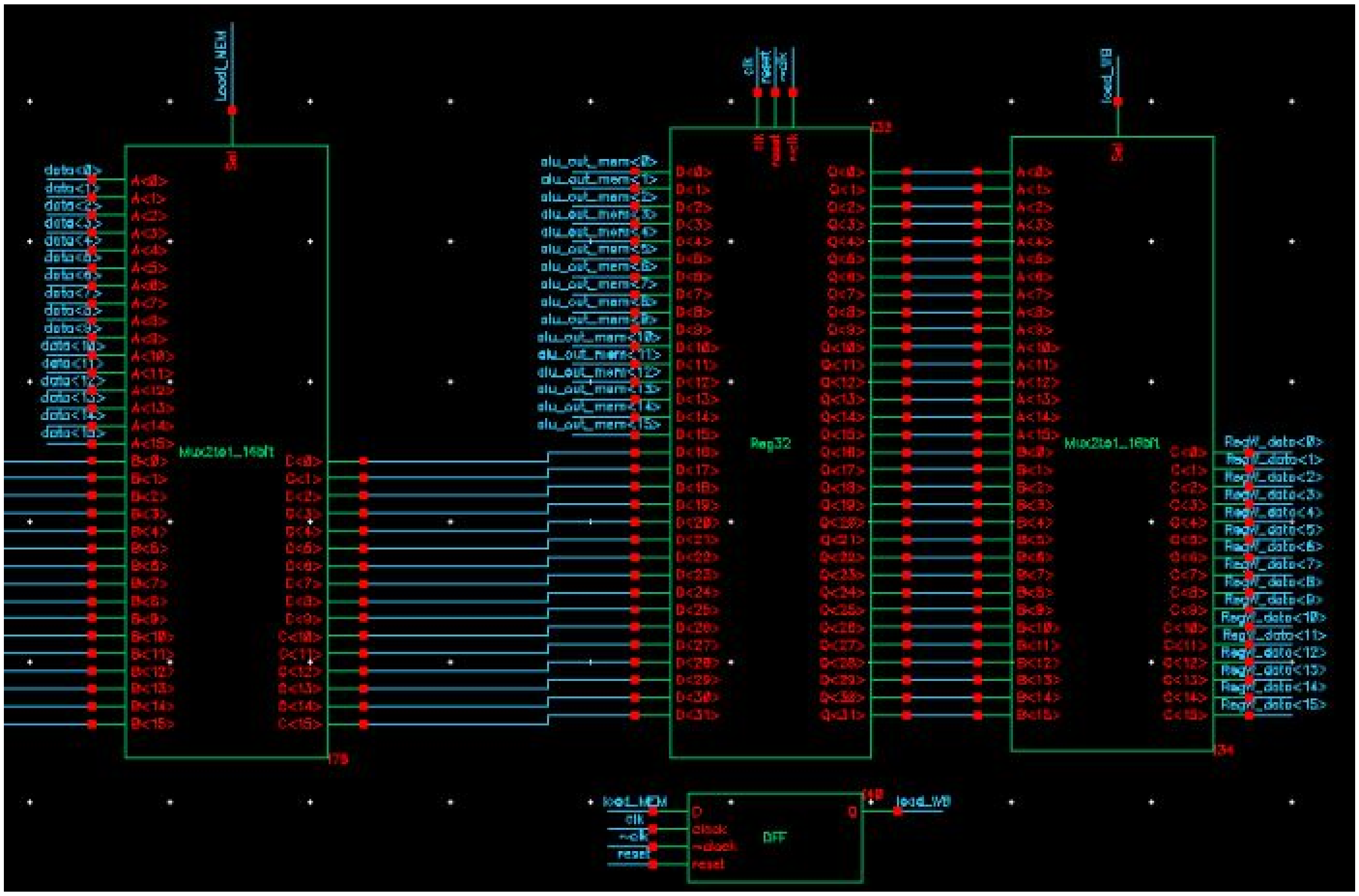


Figure 48. Write-Back Circuitry

Write-Back Circuitry used to write results back into directed 16-bit register in RF stage.

VII. Optimization

a. General Optimization Techniques

The RF is designed with simple latch (a pass transistor and two inverters). It is able to perform write and read in the same register address within signal clock cycle.

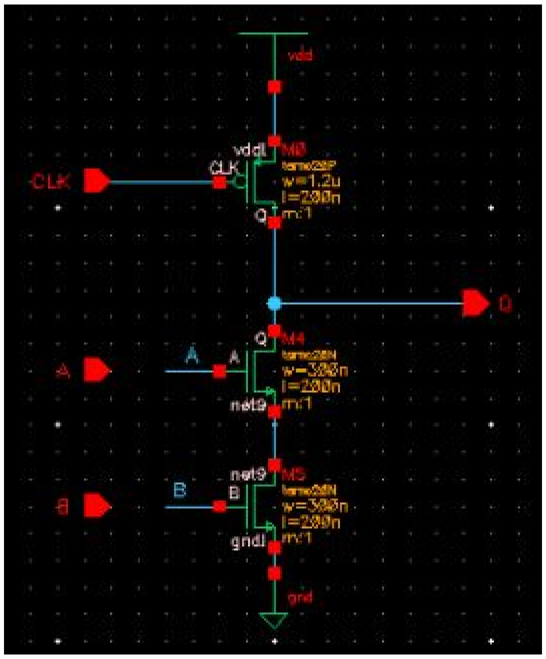
ALU is designed to save OP code bit address, to be power efficient by using decoder to select functions, to save area by combining the functions. ADD and MIN are combined into one block with inverters to invert input B to perform ADD and MIN. The shift register can be reversely connected to switch between SFL/SFR. The Multiplier is pipelined. OR/AND function is also combined and use transmission gate to save space and increase speed.

MEM stage contains the same design as lab2 and modified control signal, to precharge during the first half clock cycle and read or write during the second half clock cycle.

And the WB stage only contains a 2 to 1 16-bit mux which is built from pass transistors.

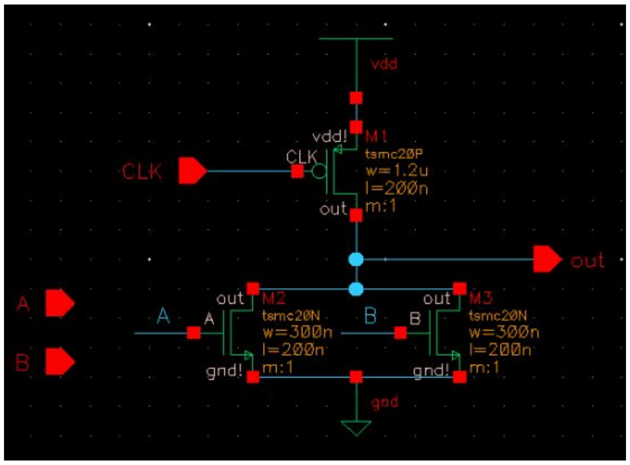
b. Dynamic Logic

Dynamic logic was implemented for several logic gates (see below). When the clock is low, the dynamic logic gate is in precharge phase and output is pulled high (parasitic capacitance charged to Vdd). When clock goes high (evaluation phase), the precharge transistor turns off and the output depends solely on the value of the inputs. If logic is correct, the pull down network (PDN) will pull the output low, otherwise, the output will stay high.



Dynamic Logic NAND2

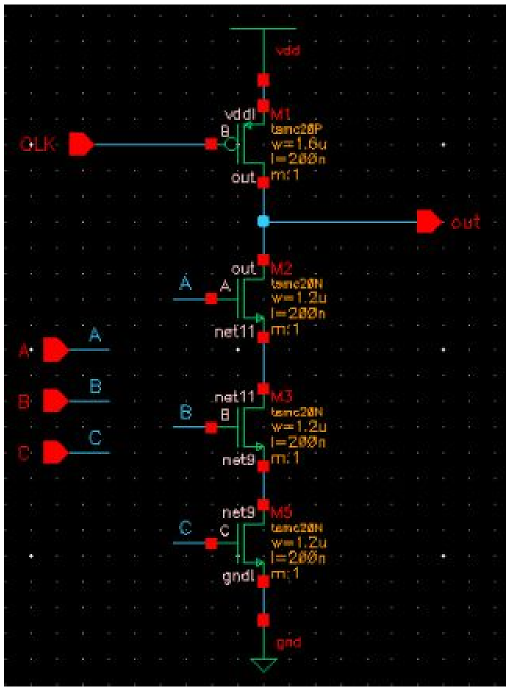
By using dynamic logic (footless) for the NAND2 gate (used in 4-to-16 decoder used in SRAM memory, other places), the gate is both smaller (got rid of one transistor) and faster - there is no PUN to fight with the PDN, gate becomes much faster. With the amount of NAND2 gates used in overall project, this area x delay savings adds up (delay maybe not so much as not all are on critical path).



Dynamic Logic NOR2

By using dynamic logic (footless) for the NOR2 gate (used in 4-to-16 decoder used in SRAM memory, other places), the gate is both smaller (got rid of one transistor) and faster - there is no PUN to fight with the PDN, gate becomes much faster. The NOR2 dynamic logic gate is even faster than the NAND2 logic gate (opposite of static CMOS) because the PDN is a parallel network for NOR2 dynamic gate.

Note: In the 4-to-16 decoder, if we have a dynamic NAND2 followed by a dynamic NOR2, we must put a static inverter in-between the two and then a static inverter at the output to get the right logic. We must use this type of domino logic to ensure that the output is able to get back up to Vdd during the evaluation phase.



Dynamic Logic NAND3

In using dynamic logic for the NAND3 gate, we are able to get rid of 2x PMOS transistors. Again, this gate is faster than its static equivalent due to the absence of a PUN to fight with the PDN.

c. Power Optimization

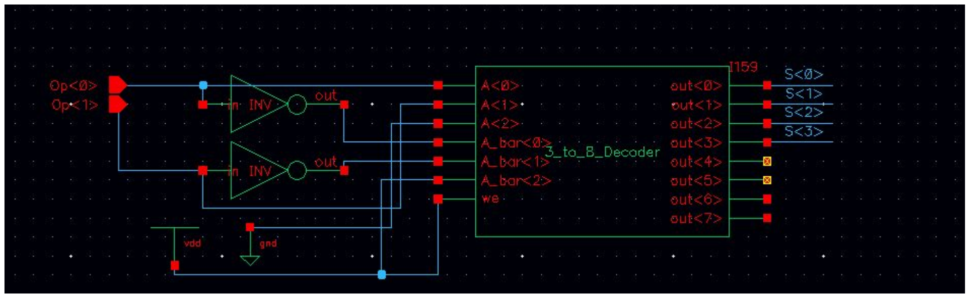
We are mainly concerned with optimizing dynamic power in this project. Leakage optimization does not buy us much in this project as the technology we are using is old will relatively low leakage. Power gating implementation will most likely increase the power x area x delay product.

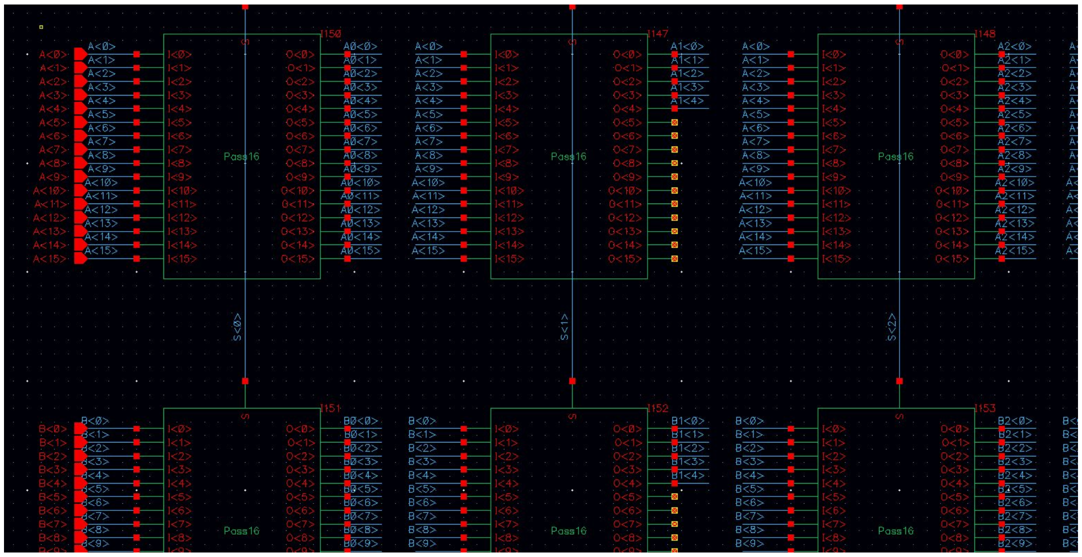
Pipelining: One way in which we optimize for dynamic power in this project is pipelining. The CPU as a whole is pipelined into 5 stages (as specified in project description). In addition, we have also pipelined our MUL block for additional power savings. This pipelining does come at a cost of increased area, but not to the point of offsetting the pros of the power savings from the optimization.

Glitch Reduction: In the layout, we are cognizant of path delays involved in each block/stage (try to ensure signals at each stage arrive at ~same time based on metal/trace lengths in layout) in order to avoid glitching hazards. Glitching hazards give bad data and lead to increased switching activity.

Locality of Reference (LOR): In the layout, we try our best to avoid sending data over long distances. Especially if not needed. Requires thoughtful floor planning in the layout.

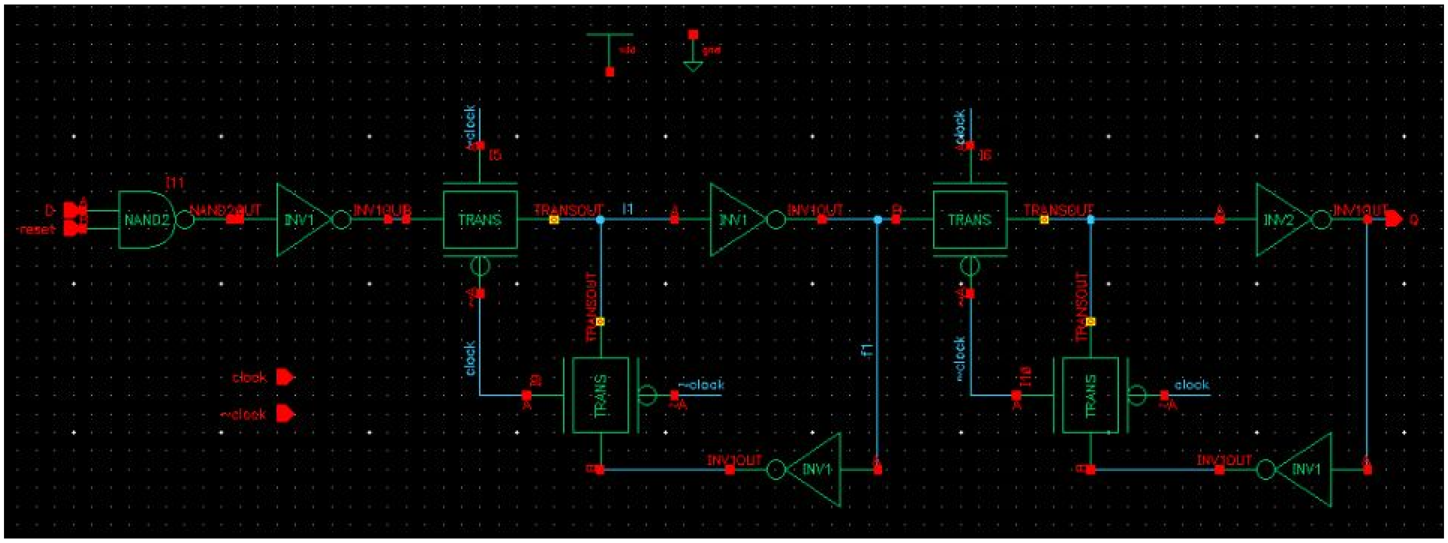
Gated Input in ALU:





Every input signal of each computation module is gated; the pass transistors will only be turned on when selected module is used (add, sub, add, or, multiplier, FL, SFR). The input data will not change when module is not used. Thus, the switching power is saved

d. DFF Optimization

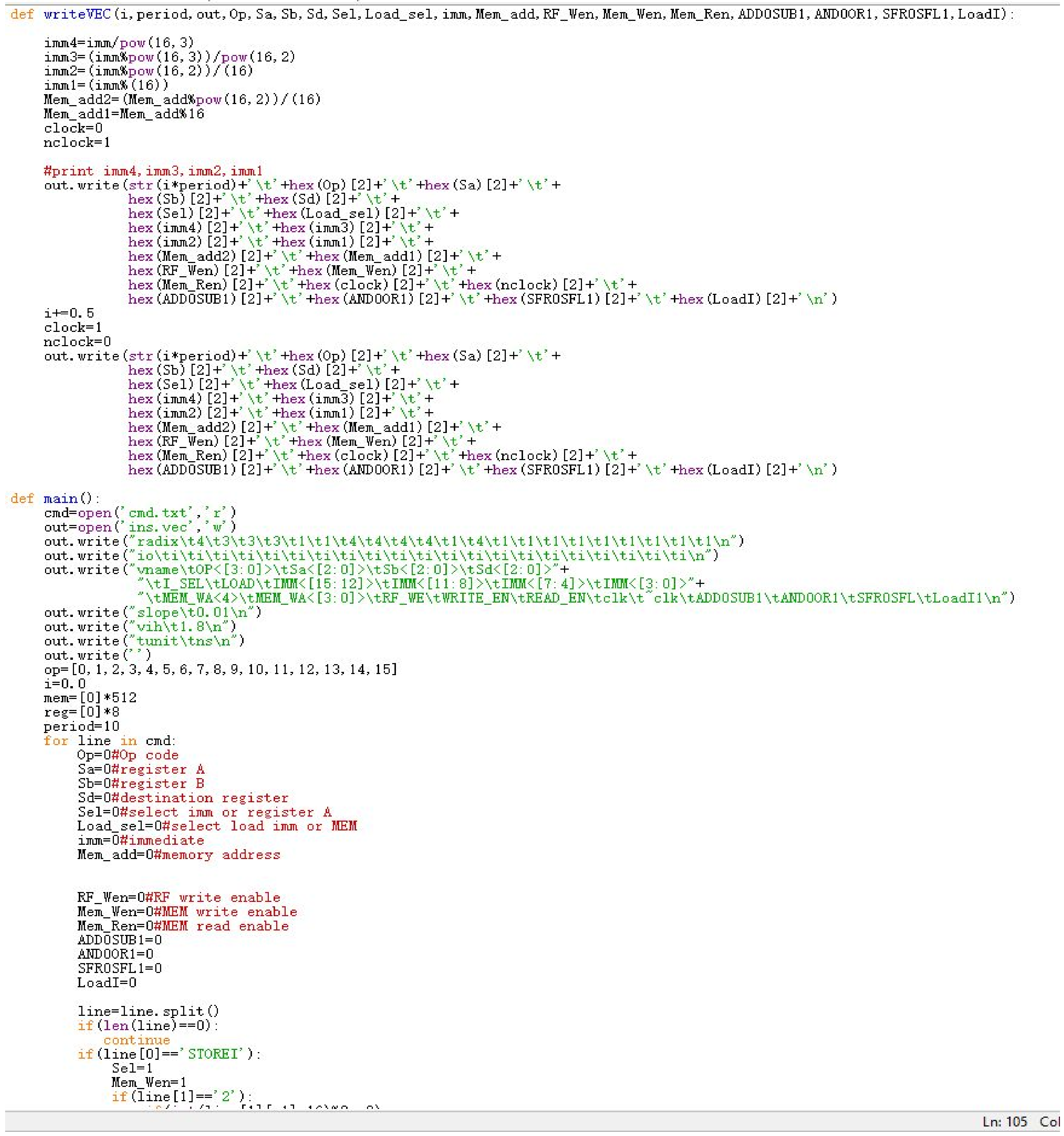


TG-Based DFF 1

Transmission Gate-Based D Flip Flop #1 (from lecture slides) is the DFF chosen for our design. This is a single-edge triggered flip flop. While not as fast as pulsed or dual-edge triggered flip flop, it is simple, robust, compact, and energy efficient. And the output inverter can be easily sized up, which will be easier for timing optimization.

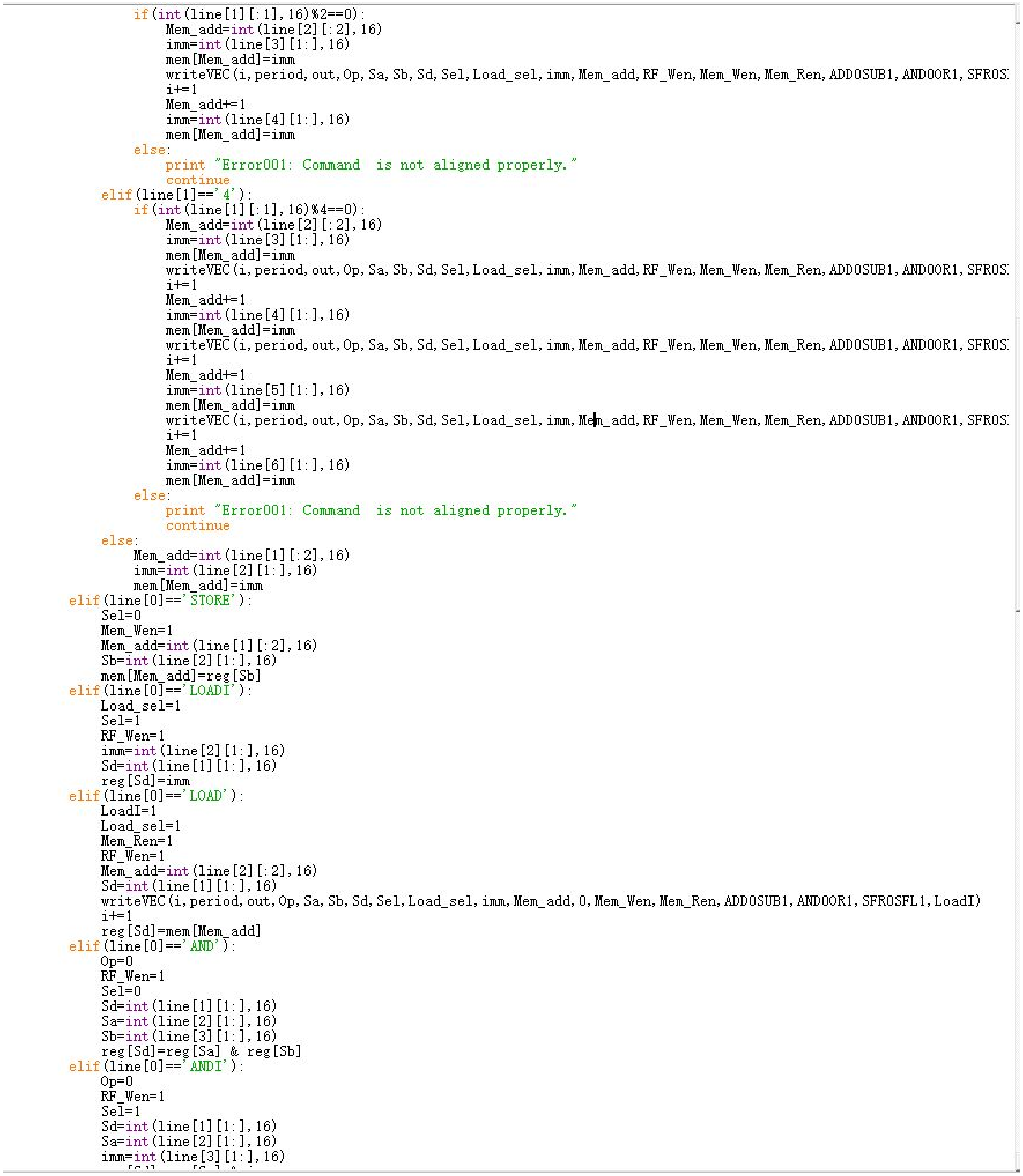
VIII. Python/Vector Files

a. Generation.py

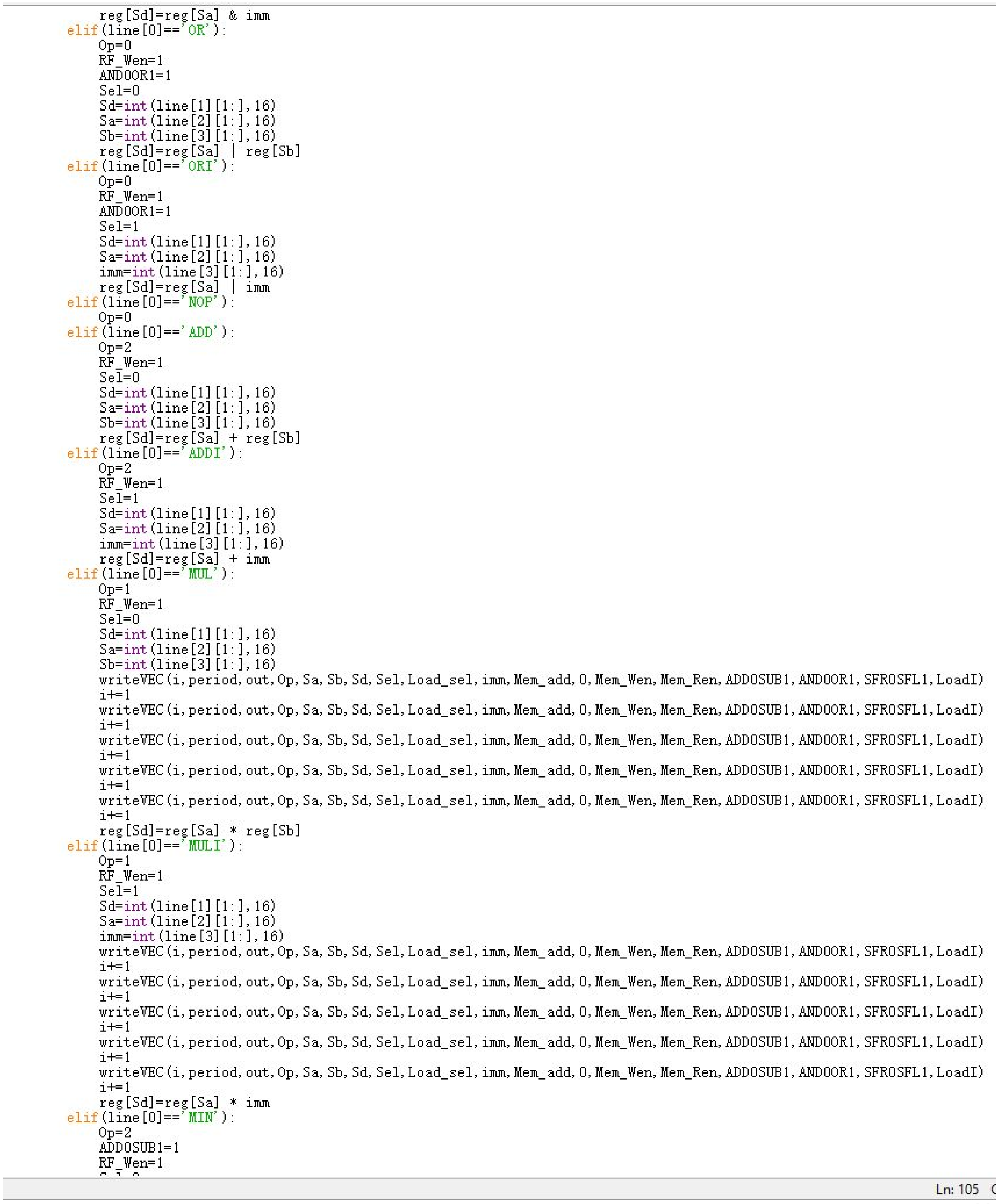


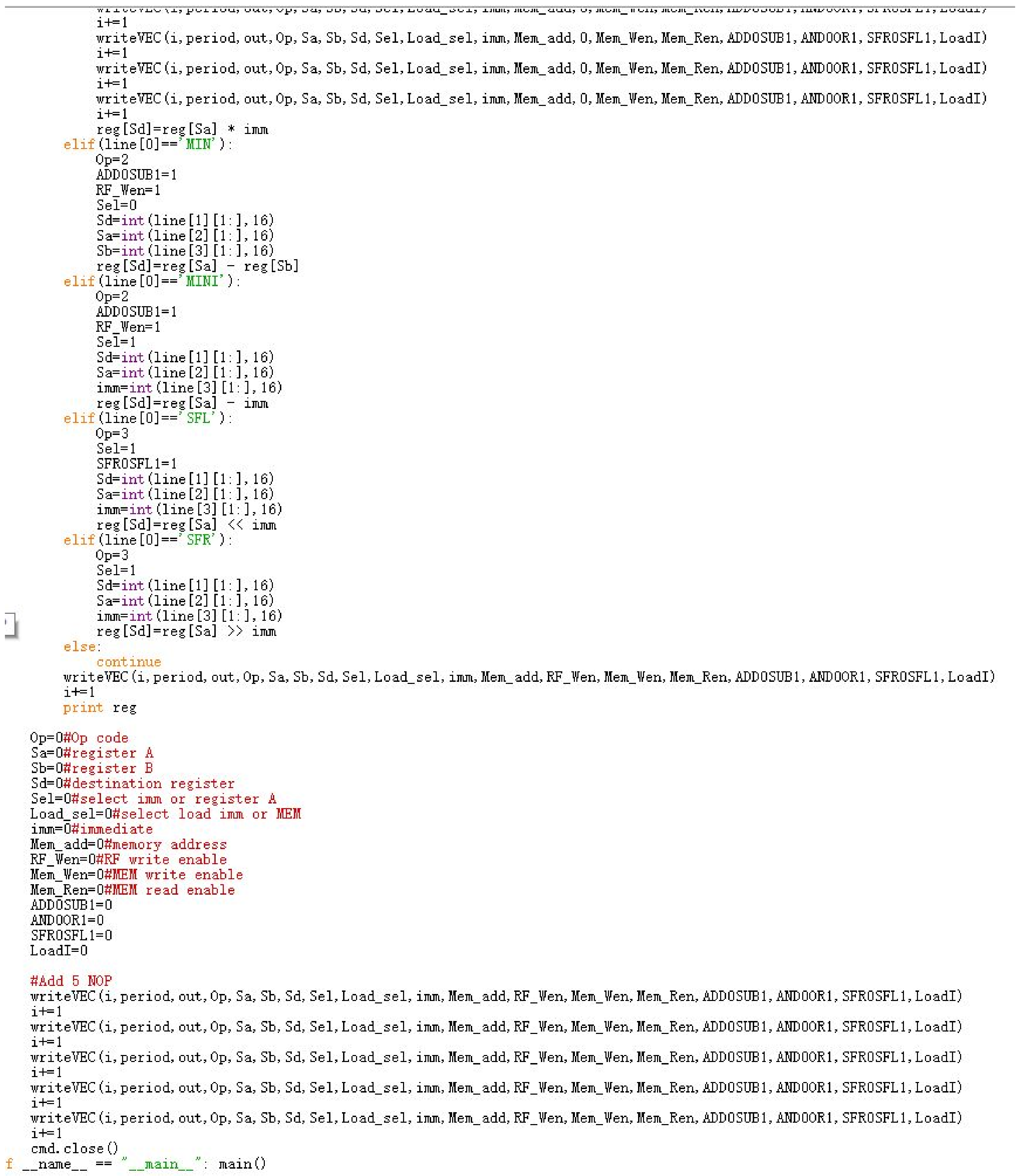
```
def writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADDOSUB1,ANDOOR1,SFROSFL1,LoadI):

    imm4=imm/pow(16,3)
    imm3=(imm%pow(16,3))/pow(16,2)
    imm2=(imm%pow(16,2))/(16)
    imm1=(imm%(16))
    Mem_add2=(Mem_add%pow(16,2))/(16)
    Mem_add1=Mem_add%16
    clock=0
    nclock=1

    #print imm4,imm3,imm2,imm1
    out.write(str(i*period)+'\t'+hex(Op)[2]+'\t'+hex(Sa)[2]+'\t'+
              hex(Sb)[2]+'\t'+hex(Sd)[2]+'\t'+
              hex(Sel)[2]+'\t'+hex(Load_sel)[2]+'\t'+
              hex(imm4)[2]+'\t'+hex(imm3)[2]+'\t'+
              hex(imm2)[2]+'\t'+hex(imm1)[2]+'\t'+
              hex(Mem_add2)[2]+'\t'+hex(Mem_add1)[2]+'\t'+
              hex(RF_Wen)[2]+'\t'+hex(Mem_Wen)[2]+'\t'+
              hex(Mem_Ren)[2]+'\t'+hex(clock)[2]+'\t'+hex(nclock)[2]+'\t'+
              hex(ADDOSUB1)[2]+'\t'+hex(ANDOOR1)[2]+'\t'+hex(SFROSFL1)[2]+'\t'+hex(LoadI)[2]+'\n')
    i+=0.5
    clock=1
    nclock=0
    out.write(str(i*period)+'\t'+hex(Op)[2]+'\t'+hex(Sa)[2]+'\t'+
              hex(Sb)[2]+'\t'+hex(Sd)[2]+'\t'+
              hex(Sel)[2]+'\t'+hex(Load_sel)[2]+'\t'+
              hex(imm4)[2]+'\t'+hex(imm3)[2]+'\t'+
              hex(imm2)[2]+'\t'+hex(imm1)[2]+'\t'+
              hex(Mem_add2)[2]+'\t'+hex(Mem_add1)[2]+'\t'+
              hex(RF_Wen)[2]+'\t'+hex(Mem_Wen)[2]+'\t'+
              hex(Mem_Ren)[2]+'\t'+hex(clock)[2]+'\t'+hex(nclock)[2]+'\t'+
              hex(ADDOSUB1)[2]+'\t'+hex(ANDOOR1)[2]+'\t'+hex(SFROSFL1)[2]+'\t'+hex(LoadI)[2]+'\n')

def main():
    cmd=open('cmd.txt','r')
    out=open('ins.vec','w')
    out.write("radix\t4\t3\t3\t3\t1\t1\t4\t4\t4\t4\t1\t4\t1\t1\t1\t1\t1\t1\t1\t1\t1\n")
    out.write("io\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\ti\n")
    out.write("vname\tOP<[3:0]>\tSa<[2:0]>\tSb<[2:0]>\tSd<[2:0]>"+
              "\tI_SEL\tLOAD\tIMM<[15:12]>\tIMM<[11:8]>\tIMM<[7:4]>\tIMM<[3:0]>"+
              "\tMEM_WA<4>\tMEM_WA<[3:0]>\tRF_WE\tWRITE_EN\tREAD_EN\tclk\t~clk\tADDOSUB1\tANDOOR1\tSFROSFL\tLoadI1\n")
    out.write("slope\t0.01\n")
    out.write("vih\t1.8\n")
    out.write("tunit\tns\n")
    out.write('')
    op=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
    i=0.0
    mem=[0]*512
    reg=[0]*8
    period=10
    for line in cmd:
        Op=0#Op code
        Sa=0#register A
        Sb=0#register B
        Sd=0#destination register
        Sel=0#select imm or register A
        Load_sel=0#select load imm or MEM
        imm=0#immediate
        Mem_add=0#memory address


        RF_Wen=0#RF write enable
        Mem_Wen=0#MEM write enable
        Mem_Ren=0#MEM read enable
        ADDOSUB1=0
        ANDOOR1=0
        SFROSFL1=0
        LoadI=0

        line=line.split()
        if(len(line)==0):
           continue
        if(line[0]=='STOREI'):
            Sel=1
            Mem_Wen=1
            if(line[1]=='2'):
```

Ln: 105 Col



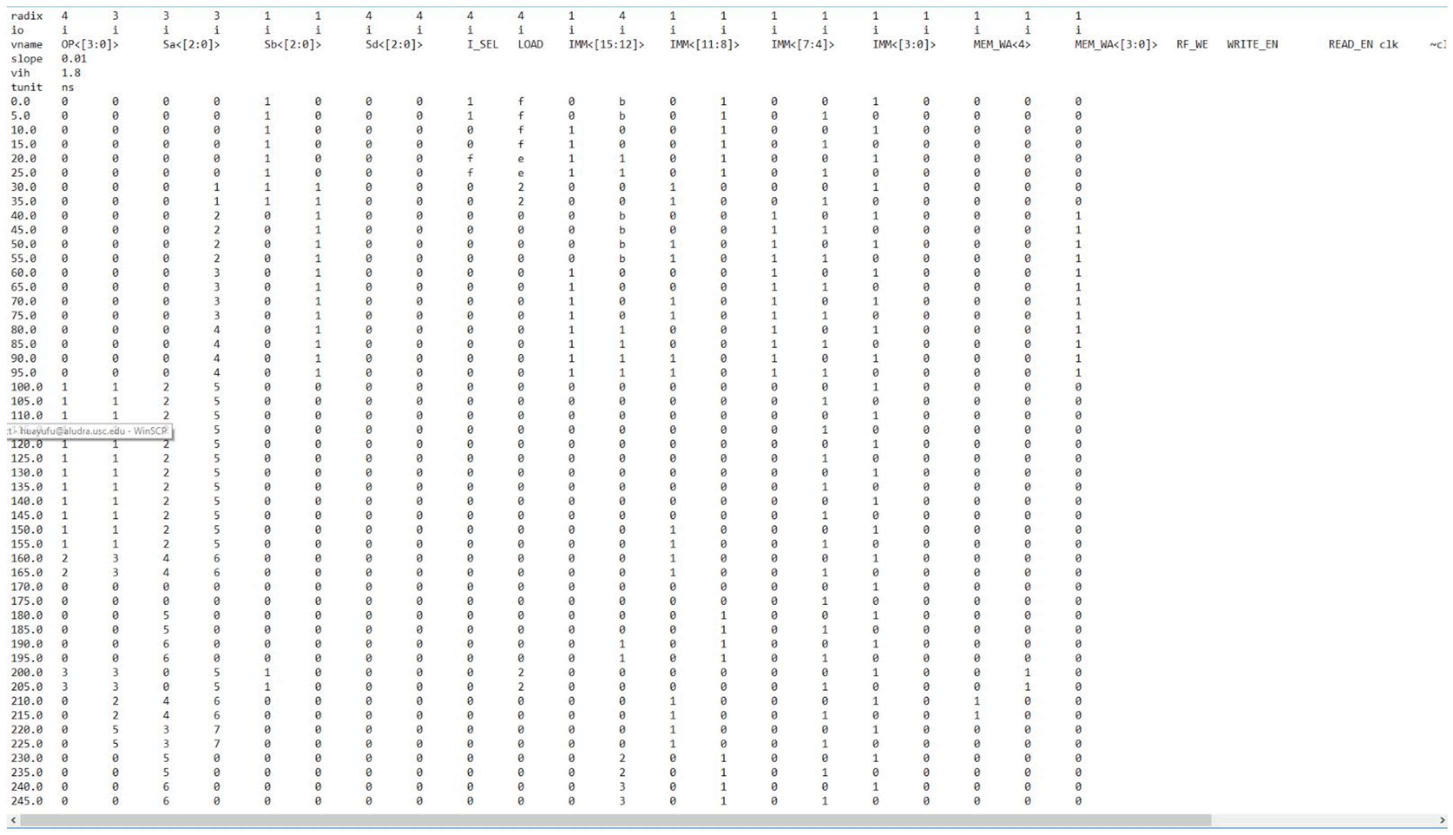
```
            if(int(line[1][:1],16)%2==0):
                Mem_add=int(line[2][:2],16)
                imm=int(line[3][1:],16)
                mem[Mem_add]=imm
                writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFROS
                i+=1
                Mem_add+=1
                imm=int(line[4][1:],16)
                mem[Mem_add]=imm
            else:
                print "Error001: Command  is not aligned properly."
                continue
        elif(line[1]=='4'):
            if(int(line[1][:1],16)%4==0):
                Mem_add=int(line[2][:2],16)
                imm=int(line[3][1:],16)
                mem[Mem_add]=imm
                writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFROS
                i+=1
                Mem_add+=1
                imm=int(line[4][1:],16)
                mem[Mem_add]=imm
                writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFROS
                i+=1
                Mem_add+=1
                imm=int(line[5][1:],16)
                mem[Mem_add]=imm
                writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFROS
                i+=1
                Mem_add+=1
                imm=int(line[6][1:],16)
                mem[Mem_add]=imm
            else:
                print "Error001: Command  is not aligned properly."
                continue
        else:
            Mem_add=int(line[1][:2],16)
            imm=int(line[2][1:],16)
            mem[Mem_add]=imm
    elif(line[0]=='STORE'):
        Sel=0
        Mem_Wen=1
        Mem_add=int(line[1][:2],16)
        Sb=int(line[2][1:],16)
        mem[Mem_add]=reg[Sb]
    elif(line[0]=='LOADI'):
        Load_sel=1
        Sel=1
        RF_Wen=1
        imm=int(line[2][1:],16)
        Sd=int(line[1][1:],16)
        reg[Sd]=imm
    elif(line[0]=='LOAD'):
        LoadI=1
        Load_sel=1
        Mem_Ren=1
        RF_Wen=1
        Mem_add=int(line[2][:2],16)
        Sd=int(line[1][1:],16)
        writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
        i+=1
        reg[Sd]=mem[Mem_add]
    elif(line[0]=='AND'):
        Op=0
        RF_Wen=1
        Sel=0
        Sd=int(line[1][1:],16)
        Sa=int(line[2][1:],16)
        Sb=int(line[3][1:],16)
        reg[Sd]=reg[Sa] & reg[Sb]
    elif(line[0]=='ANDI'):
        Op=0
        RF_Wen=1
        Sel=1
        Sd=int(line[1][1:],16)
        Sa=int(line[2][1:],16)
        imm=int(line[3][1:],16)
```


```
            reg[Sd]=reg[Sa] & imm
        elif(line[0]=='OR'):
            Op=0
            RF_Wen=1
            AND0OR1=1
            Sel=0
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            Sb=int(line[3][1:],16)
            reg[Sd]=reg[Sa] | reg[Sb]
        elif(line[0]=='ORI'):
            Op=0
            RF_Wen=1
            AND0OR1=1
            Sel=1
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            imm=int(line[3][1:],16)
            reg[Sd]=reg[Sa] | imm
        elif(line[0]=='NOP'):
            Op=0
        elif(line[0]=='ADD'):
            Op=2
            RF_Wen=1
            Sel=0
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            Sb=int(line[3][1:],16)
            reg[Sd]=reg[Sa] + reg[Sb]
        elif(line[0]=='ADDI'):
            Op=2
            RF_Wen=1
            Sel=1
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            imm=int(line[3][1:],16)
            reg[Sd]=reg[Sa] + imm
        elif(line[0]=='MUL'):
            Op=1
            RF_Wen=1
            Sel=0
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            Sb=int(line[3][1:],16)
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            reg[Sd]=reg[Sa] * reg[Sb]
        elif(line[0]=='MULI'):
            Op=1
            RF_Wen=1
            Sel=1
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            imm=int(line[3][1:],16)
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            reg[Sd]=reg[Sa] * imm
        elif(line[0]=='MIN'):
            Op=2
            ADD0SUB1=1
            RF_Wen=1
```



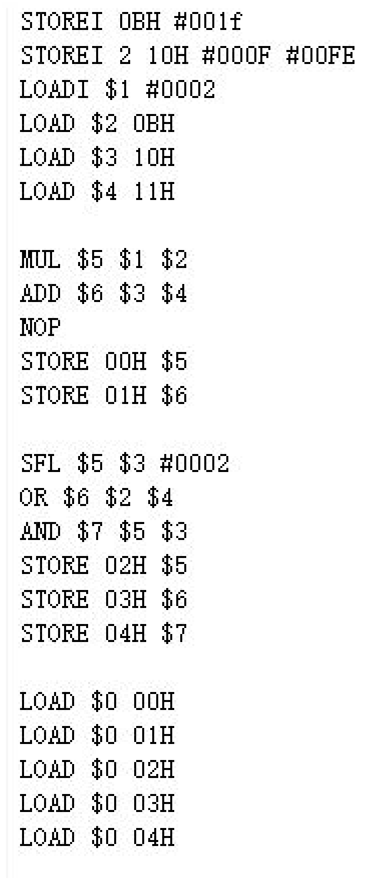
```
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,0,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
            i+=1
            reg[Sd]=reg[Sa] * imm
        elif(line[0]=='MIN'):
            Op=2
            ADD0SUB1=1
            RF_Wen=1
            Sel=0
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            Sb=int(line[3][1:],16)
            reg[Sd]=reg[Sa] - reg[Sb]
        elif(line[0]=='MINI'):
            Op=2
            ADD0SUB1=1
            RF_Wen=1
            Sel=1
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            imm=int(line[3][1:],16)
            reg[Sd]=reg[Sa] - imm
        elif(line[0]=='SFL'):
            Op=3
            Sel=1
            SFR0SFL1=1
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            imm=int(line[3][1:],16)
            reg[Sd]=reg[Sa] << imm
        elif(line[0]=='SFR'):
            Op=3
            Sel=1
            Sd=int(line[1][1:],16)
            Sa=int(line[2][1:],16)
            imm=int(line[3][1:],16)
            reg[Sd]=reg[Sa] >> imm
        else:
            continue
        writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
        i+=1
        print reg

    Op=0#Op code
    Sa=0#register A
    Sb=0#register B
    Sd=0#destination register
    Sel=0#select imm or register A
    Load_sel=0#select load imm or MEM
    imm=0#immediate
    Mem_add=0#memory address
    RF_Wen=0#RF write enable
    Mem_Wen=0#MEM write enable
    Mem_Ren=0#MEM read enable
    ADD0SUB1=0
    AND0OR1=0
    SFR0SFL1=0
    LoadI=0

    #Add 5 NOP
    writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
    i+=1
    writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
    i+=1
    writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
    i+=1
    writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
    i+=1
    writeVEC(i,period,out,Op,Sa,Sb,Sd,Sel,Load_sel,imm,Mem_add,RF_Wen,Mem_Wen,Mem_Ren,ADD0SUB1,AND0OR1,SFR0SFL1,LoadI)
    i+=1
    cmd.close()
f __name__ == "__main__": main()
```

b. Ins.vec



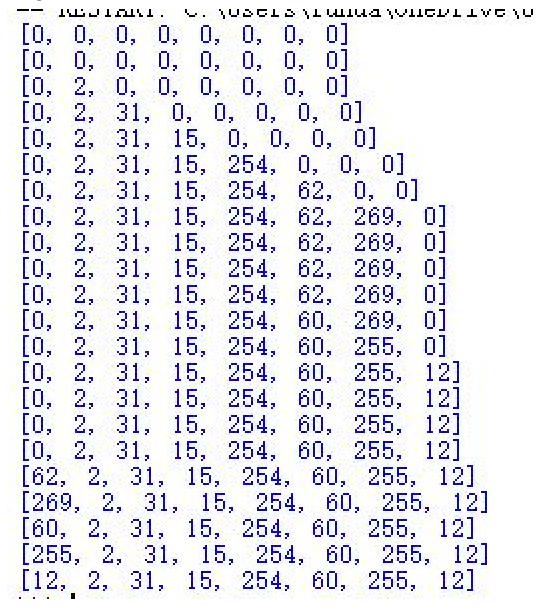
```
radix  4       3       3       3       1       1       4       4       4       4       1       4       1       1       1       1       1       1       1       1       1
io     i       i       i       i       i       i       i       i       i       i       i       i       i       i       i       i       i       i       i       i       i
vname  OP<[3:0]>       Sa<[2:0]>       Sb<[2:0]>       Sd<[2:0]>       I_SEL   LOAD    IMM<[15:12]>    IMM<[11:8]>     IMM<[7:4]>      IMM<[3:0]>      MEM_WA<4>       MEM_WA<[3:0]>   RF_WE   WRITE_EN        READ_EN clk     ~cl
slope  0.01
vih    1.8
tunit  ns
0.0    0       0       0       0       1       0       0       0       1       f       0       b       0       1       0       0       1       0       0       0       0
5.0    0       0       0       0       1       0       0       0       1       f       0       b       0       1       0       1       0       0       0       0       0
10.0   0       0       0       0       1       0       0       0       0       f       1       0       0       1       0       0       1       0       0       0       0
15.0   0       0       0       0       1       0       0       0       0       f       1       0       0       1       0       1       0       0       0       0       0
20.0   0       0       0       0       1       0       0       0       f       e       1       1       0       1       0       0       1       0       0       0       0
25.0   0       0       0       0       1       0       0       0       f       e       1       1       0       1       0       1       0       0       0       0       0
30.0   0       0       0       1       1       1       0       0       0       2       0       0       1       0       0       0       1       0       0       0       0
35.0   0       0       0       1       1       1       0       0       0       2       0       0       1       0       0       1       0       0       0       0       0
40.0   0       0       0       2       0       1       0       0       0       0       0       b       0       0       1       0       1       0       0       0       1
45.0   0       0       0       2       0       1       0       0       0       0       0       b       0       0       1       1       0       0       0       0       1
50.0   0       0       0       2       0       1       0       0       0       0       0       b       1       0       1       0       1       0       0       0       1
55.0   0       0       0       2       0       1       0       0       0       0       0       b       1       0       1       1       0       0       0       0       1
60.0   0       0       0       3       0       1       0       0       0       0       1       0       0       0       1       0       1       0       0       0       1
65.0   0       0       0       3       0       1       0       0       0       0       1       0       0       0       1       1       0       0       0       0       1
70.0   0       0       0       3       0       1       0       0       0       0       1       0       1       0       1       0       1       0       0       0       1
75.0   0       0       0       3       0       1       0       0       0       0       1       0       1       0       1       1       0       0       0       0       1
80.0   0       0       0       4       0       1       0       0       0       0       1       1       0       0       1       0       1       0       0       0       1
85.0   0       0       0       4       0       1       0       0       0       0       1       1       0       0       1       1       0       0       0       0       1
90.0   0       0       0       4       0       1       0       0       0       0       1       1       1       0       1       0       1       0       0       0       1
95.0   0       0       0       4       0       1       0       0       0       0       1       1       1       0       1       1       0       0       0       0       1
100.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0
105.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0       0
110.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0
:t - huayufu@aludra.usc.edu - WinSCP 5 0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0       0
120.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0
125.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0       0
130.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0
135.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0       0
140.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0
145.0  1       1       2       5       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0       0
150.0  1       1       2       5       0       0       0       0       0       0       0       0       1       0       0       0       1       0       0       0       0
155.0  1       1       2       5       0       0       0       0       0       0       0       0       1       0       0       1       0       0       0       0       0
160.0  2       3       4       6       0       0       0       0       0       0       0       0       1       0       0       0       1       0       0       0       0
165.0  2       3       4       6       0       0       0       0       0       0       0       0       1       0       0       1       0       0       0       0       0
170.0  0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0
175.0  0       0       0       0       0       0       0       0       0       0       0       0       0       0       0       1       0       0       0       0       0
180.0  0       0       5       0       0       0       0       0       0       0       0       0       0       1       0       0       1       0       0       0       0
185.0  0       0       5       0       0       0       0       0       0       0       0       0       0       1       0       1       0       0       0       0       0
190.0  0       0       6       0       0       0       0       0       0       0       0       1       0       1       0       0       1       0       0       0       0
195.0  0       0       6       0       0       0       0       0       0       0       0       1       0       1       0       1       0       0       0       0       0
200.0  3       3       0       5       1       0       0       0       0       2       0       0       0       0       0       0       1       0       0       1       0
205.0  3       3       0       5       1       0       0       0       0       2       0       0       0       0       0       1       0       0       0       1       0
210.0  0       2       4       6       0       0       0       0       0       0       0       0       1       0       0       0       1       0       1       0       0
215.0  0       2       4       6       0       0       0       0       0       0       0       0       1       0       0       1       0       0       1       0       0
220.0  0       5       3       7       0       0       0       0       0       0       0       0       1       0       0       0       1       0       0       0       0
225.0  0       5       3       7       0       0       0       0       0       0       0       0       1       0       0       1       0       0       0       0       0
230.0  0       0       5       0       0       0       0       0       0       0       0       2       0       1       0       0       1       0       0       0       0
235.0  0       0       5       0       0       0       0       0       0       0       0       2       0       1       0       1       0       0       0       0       0
240.0  0       0       6       0       0       0       0       0       0       0       0       3       0       1       0       0       1       0       0       0       0
245.0  0       0       6       0       0       0       0       0       0       0       0       3       0       1       0       1       0       0       0       0       0
```

c. Cmd.txt



```
STOREI 0BH #001f
STOREI 2 10H #000F #00FE
LOADI $1 #0002
LOAD $2 0BH
LOAD $3 10H
LOAD $4 11H

MUL $5 $1 $2
ADD $6 $3 $4
NOP
STORE 00H $5
STORE 01H $6

SFL $5 $3 #0002
OR $6 $2 $4
AND $7 $5 $3
STORE 02H $5
STORE 03H $6
STORE 04H $7

LOAD $0 00H
LOAD $0 01H
LOAD $0 02H
LOAD $0 03H
LOAD $0 04H
```

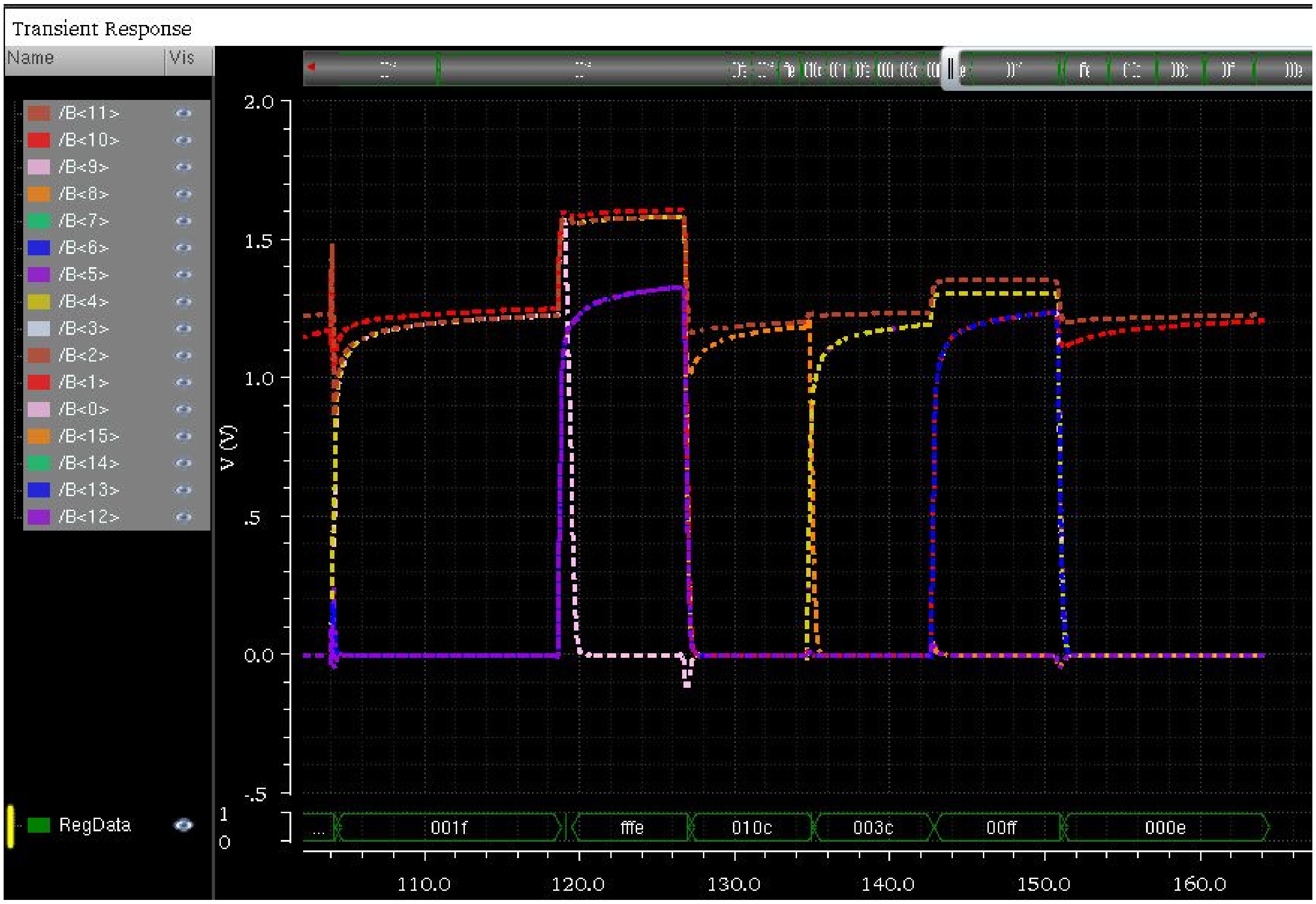
d. golden result



```
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 0, 0, 0, 0, 0, 0]
[0, 2, 31, 0, 0, 0, 0, 0]
[0, 2, 31, 15, 0, 0, 0, 0]
[0, 2, 31, 15, 254, 0, 0, 0]
[0, 2, 31, 15, 254, 62, 0, 0]
[0, 2, 31, 15, 254, 62, 269, 0]
[0, 2, 31, 15, 254, 62, 269, 0]
[0, 2, 31, 15, 254, 62, 269, 0]
[0, 2, 31, 15, 254, 62, 269, 0]
[0, 2, 31, 15, 254, 60, 269, 0]
[0, 2, 31, 15, 254, 60, 255, 0]
[0, 2, 31, 15, 254, 60, 255, 12]
[0, 2, 31, 15, 254, 60, 255, 12]
[0, 2, 31, 15, 254, 60, 255, 12]
[0, 2, 31, 15, 254, 60, 255, 12]
[62, 2, 31, 15, 254, 60, 255, 12]
[269, 2, 31, 15, 254, 60, 255, 12]
[60, 2, 31, 15, 254, 60, 255, 12]
[255, 2, 31, 15, 254, 60, 255, 12]
[12, 2, 31, 15, 254, 60, 255, 12]
```

(golden result for mult and shift have not been corrected)

e. simulation result



Simulation shows the correct result and the clock period is 4ns.