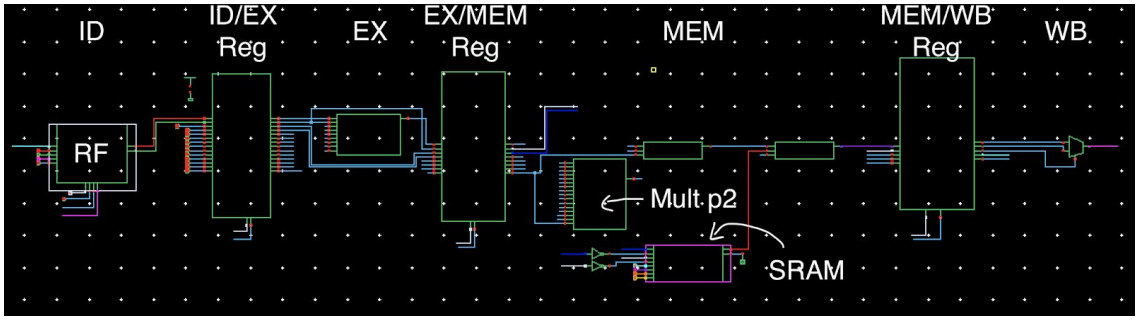
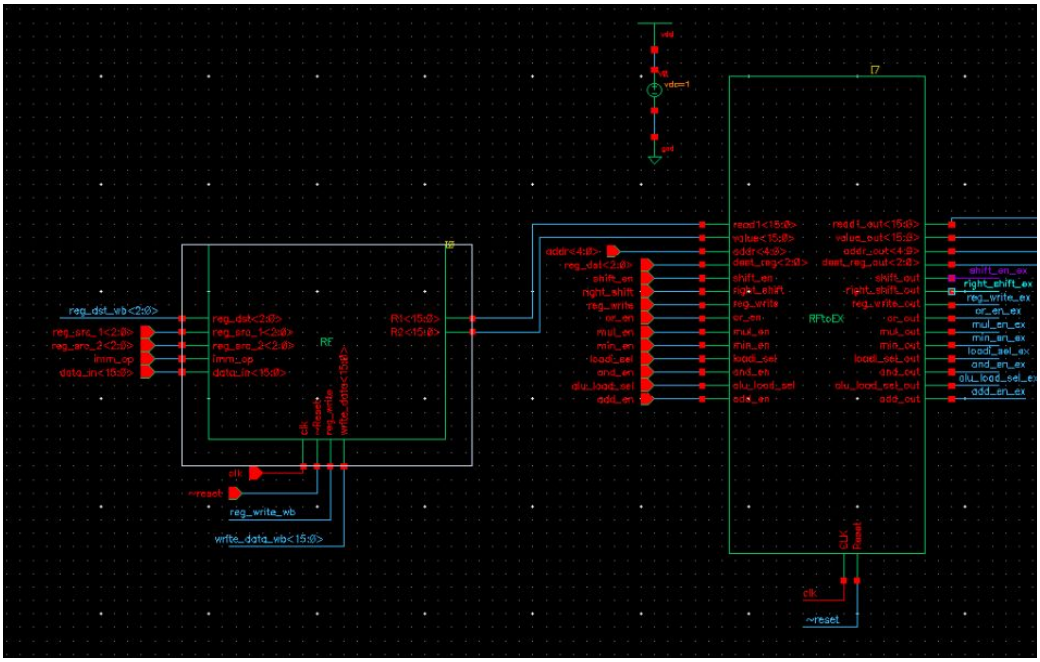
EE577a - Final Project
Design of a General-Purpose Microprocessor Using Software and Hardware Components
Phase 3

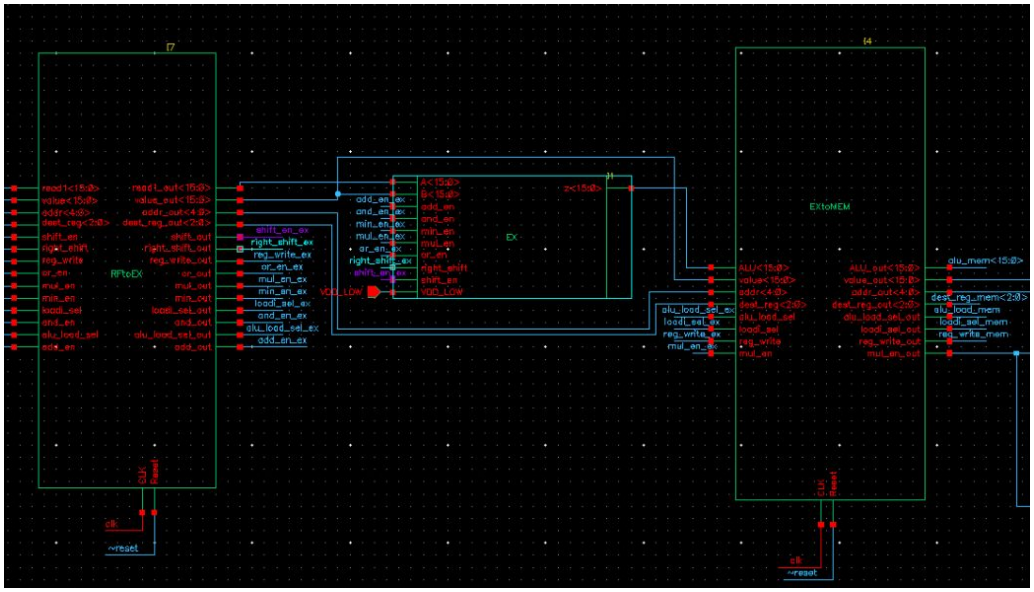
Team 5
Yulong Ding 8456815156 - Front-end / Back-end Python Scripting / Layout
Samuel Bruner 3198542984 - Register File / System Integration / Layout
Angsagan Abdigazy 2509824628 - ALU / Layout

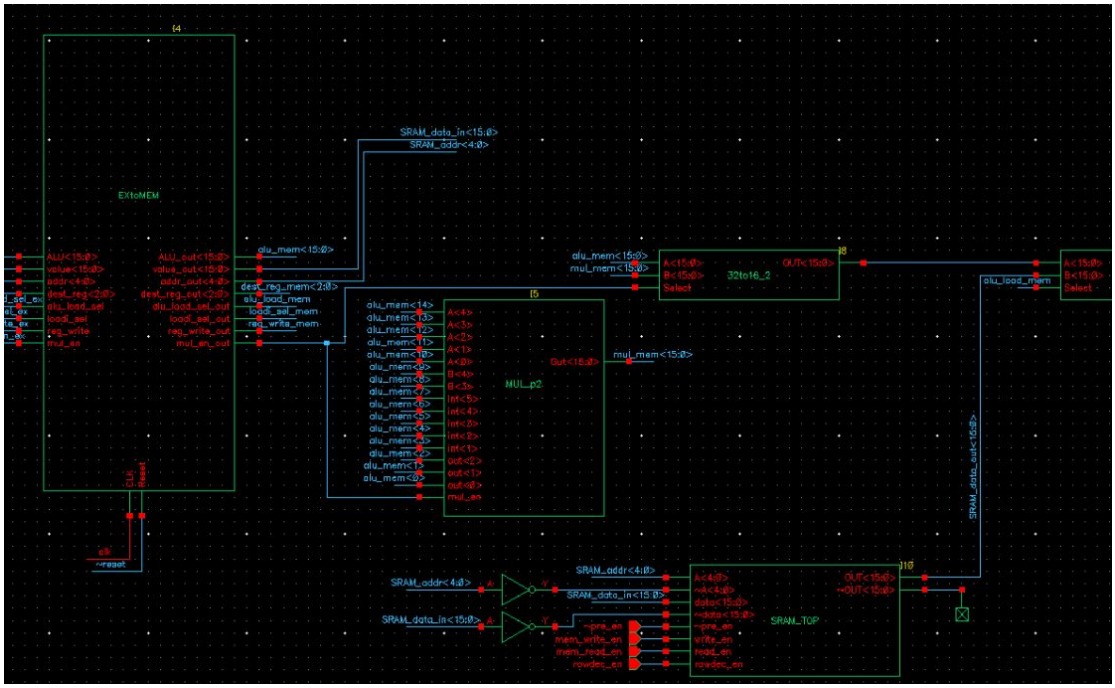
Table of Contents

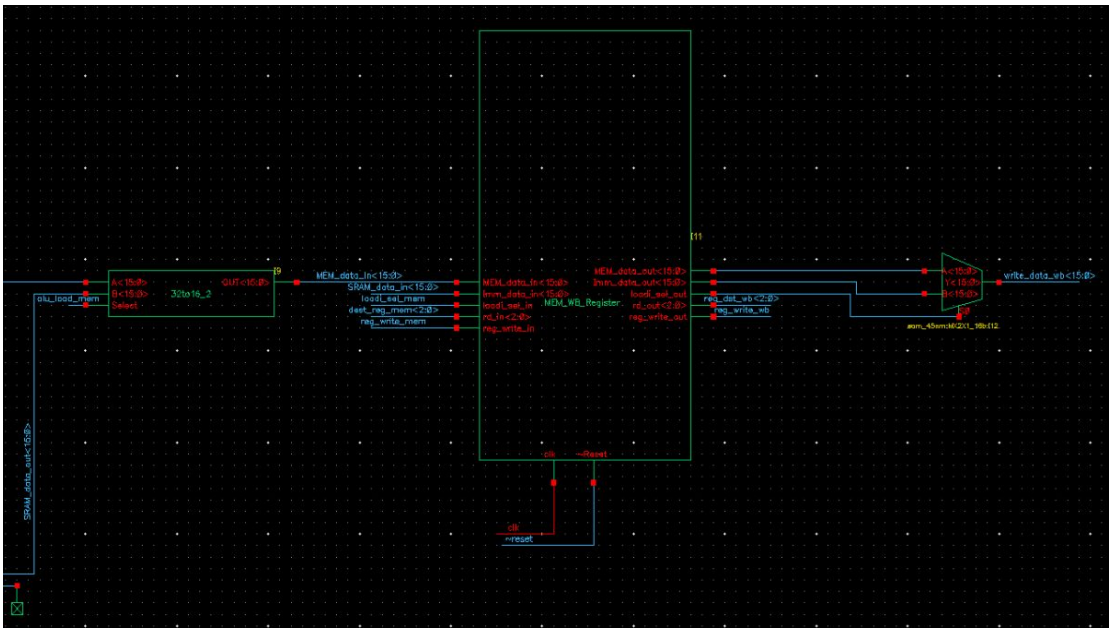
- **Top-Level CPU Schematic**
    - Note: refer to Phase 1 Report for the detailed schematic of each block.





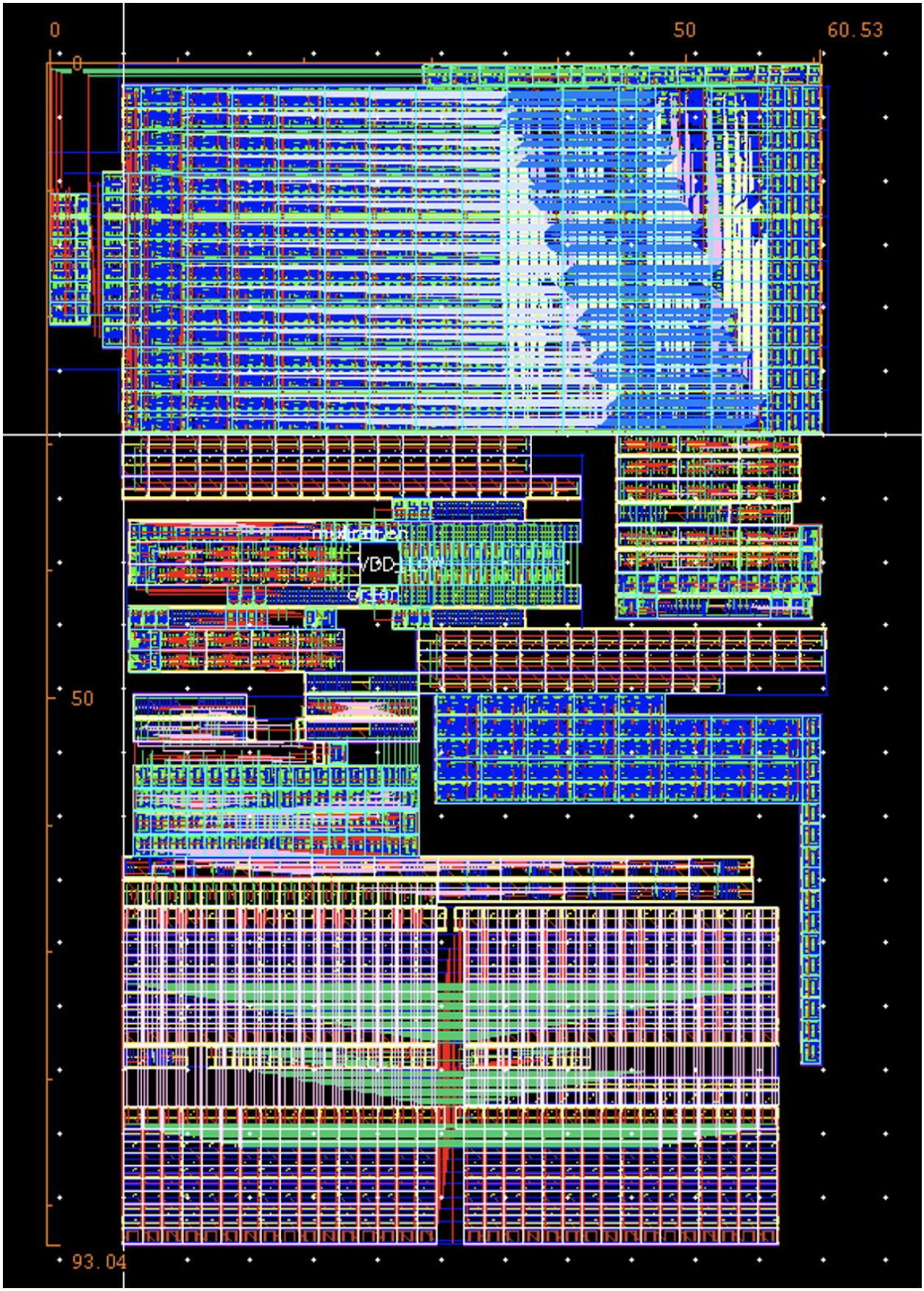

I7
RFtoEX
EX
I1
EXtoMEM
I4
clk
~reset

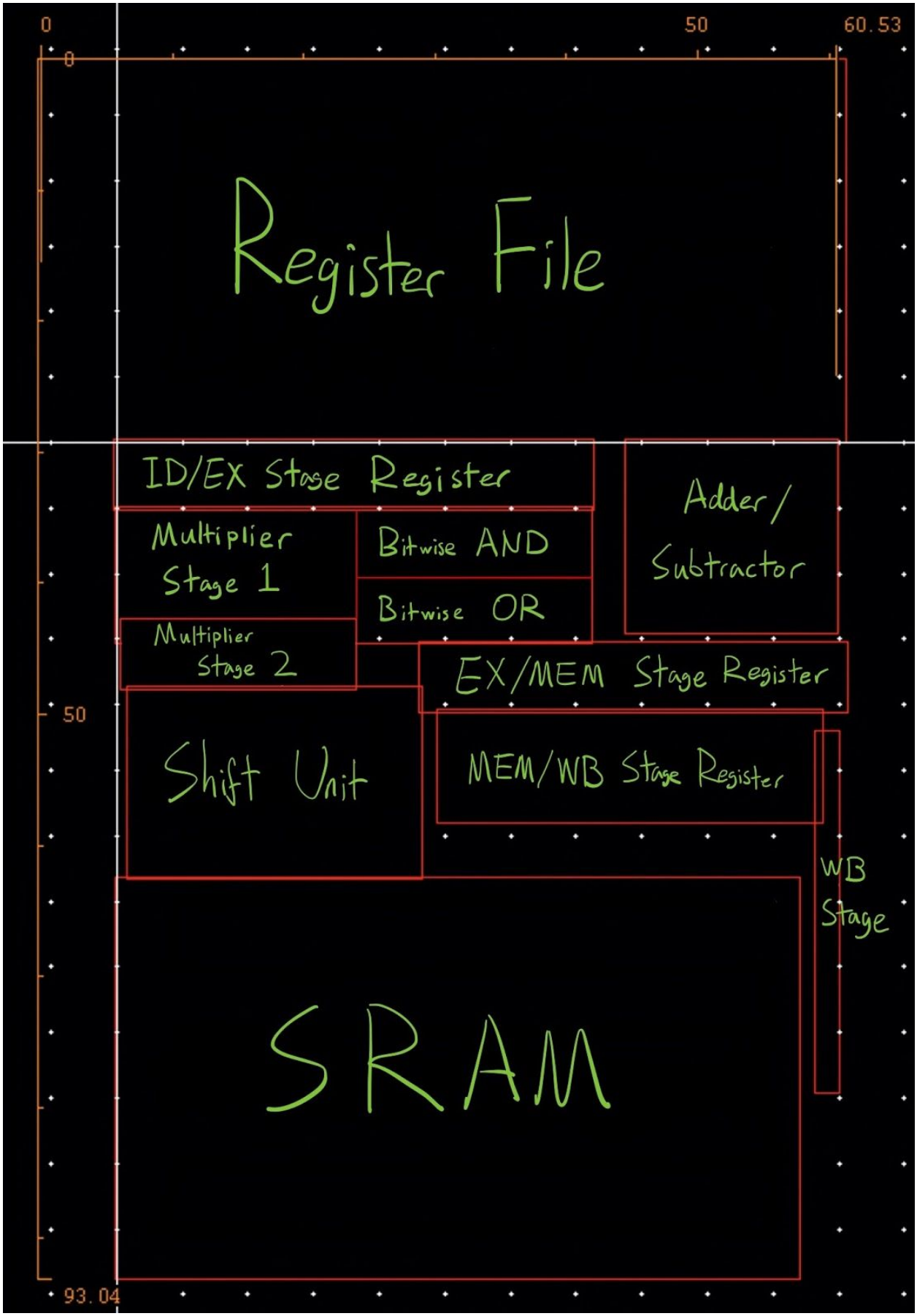

I4
EXtoMEM
SRAM_data_in<15:0>
SRAM_addr<4:0>
I5
MUL_p2
I8
32to16_2
SRAM_TOP
I10
clk
~reset



- **CPU Layout**
    - **Top-Level CPU Floorplan**

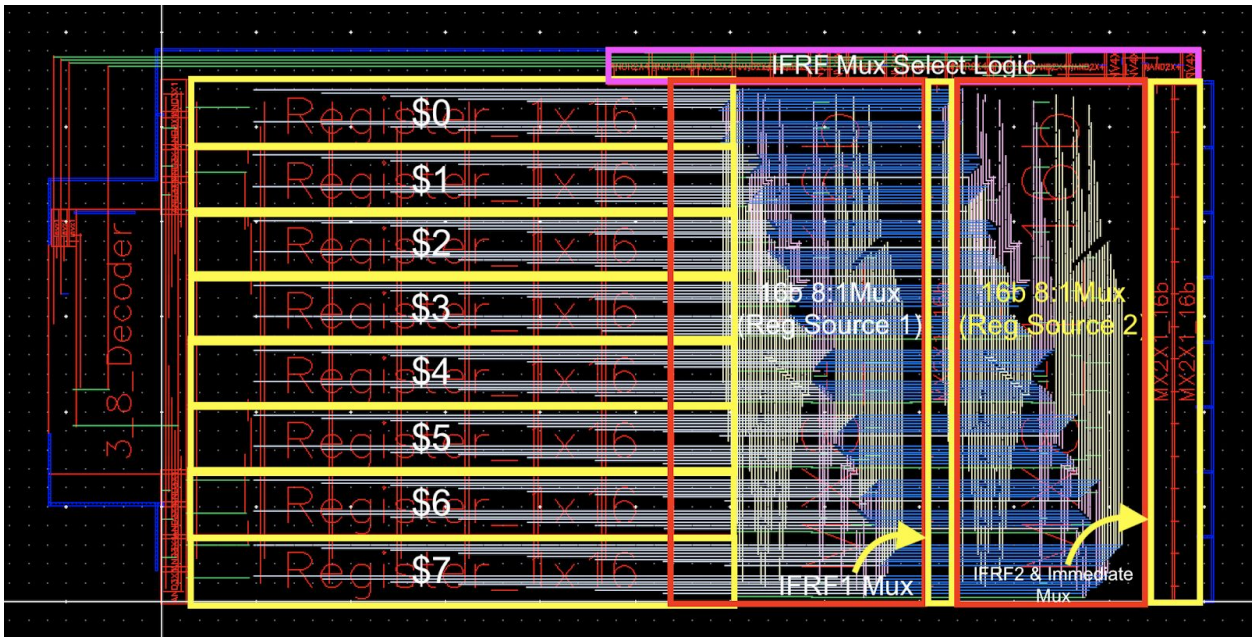
The top-level floorplan of the processor shown below contains layouts of major modules including the register file, all ALU submodules in the EX stage, the SRAM in the MEM stage, all stage registers as well as peripheral logic in the WB stage. However, note that the routing between modules is incomplete given the limited time and thus the top-level layout below does not pass LVS. Nevertheless, this top-level layout contains all modules of the 5-stage pipelined CPU in place and therefore serves as a floorplan reference providing a fairly accurate area estimate for the processor.

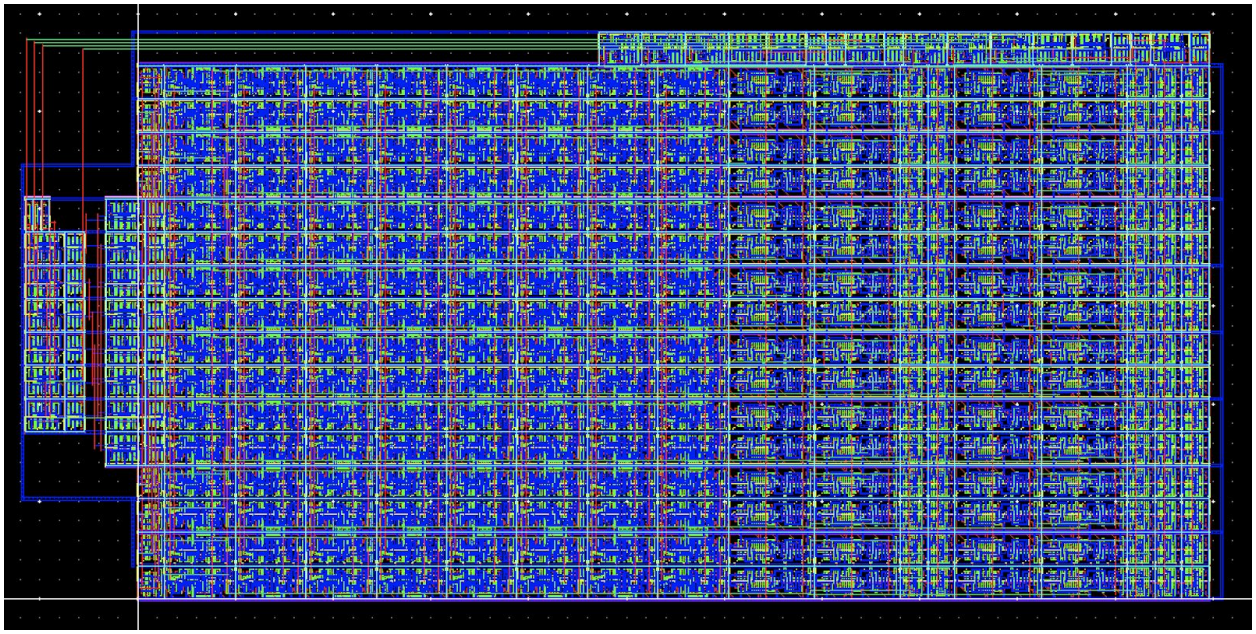

0
50
60.53
0
VDD
50
93.04

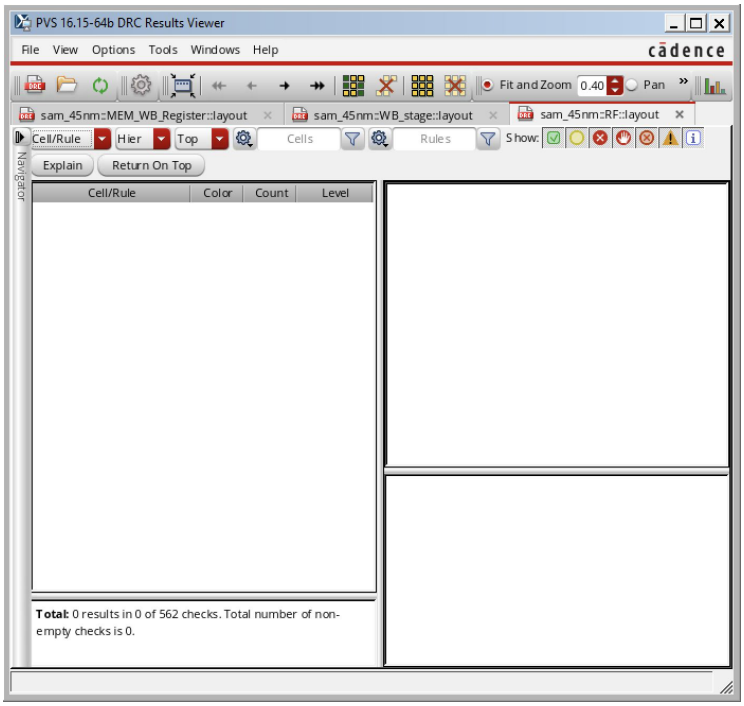

0
50
60.53
0
Register File
ID/EX Stage Register
Adder/
Subtractor
Multiplier
Stage 1
Bitwise AND
Bitwise OR
Multiplier
Stage 2
EX/MEM Stage Register
50
Shift Unit
MEM/WB Stage Register
WB
Stage
SRAM
93.04

- **Register File Layout**

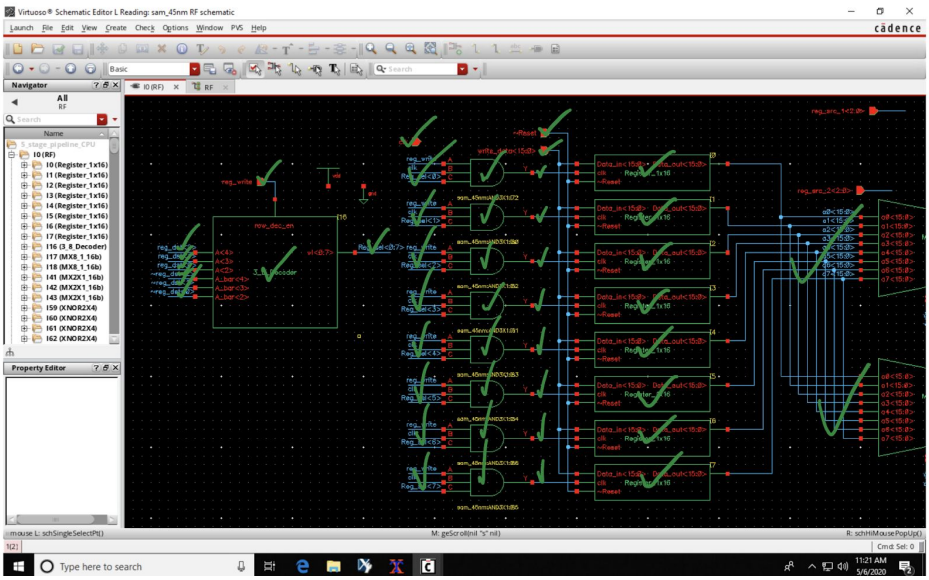
The annotated register file layout can be seen below. The design is currently clear of DRC. However, it is slightly incomplete and unable to pass LVS.

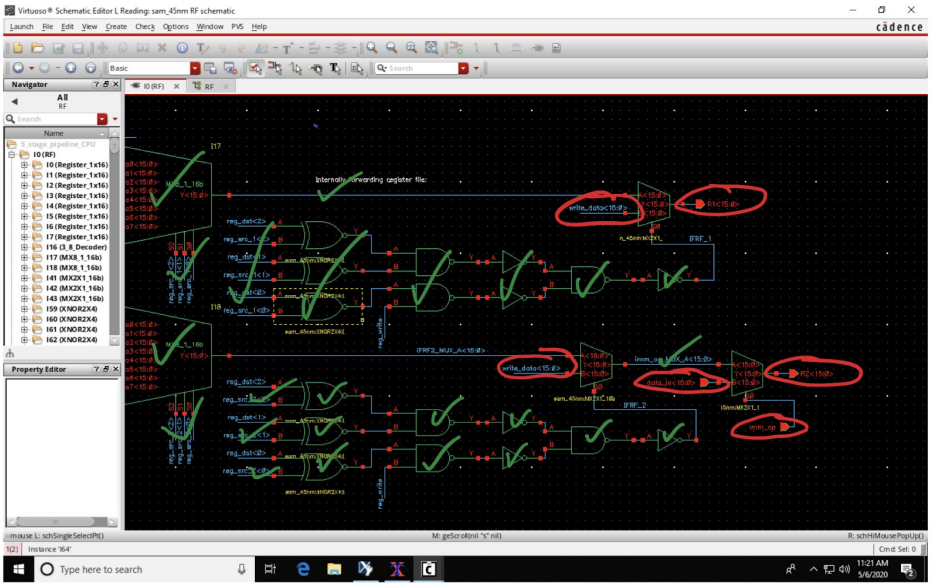




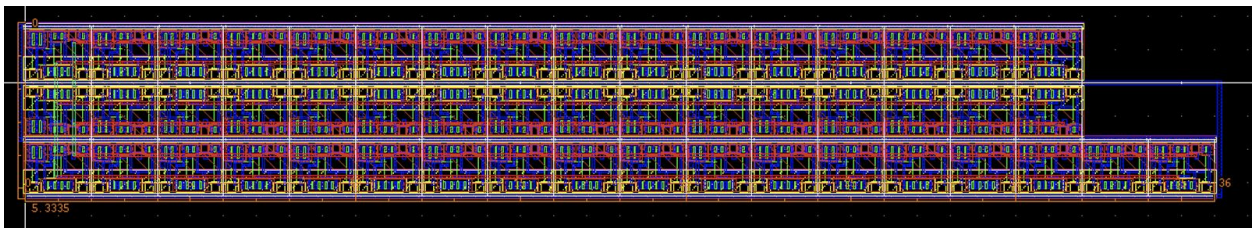


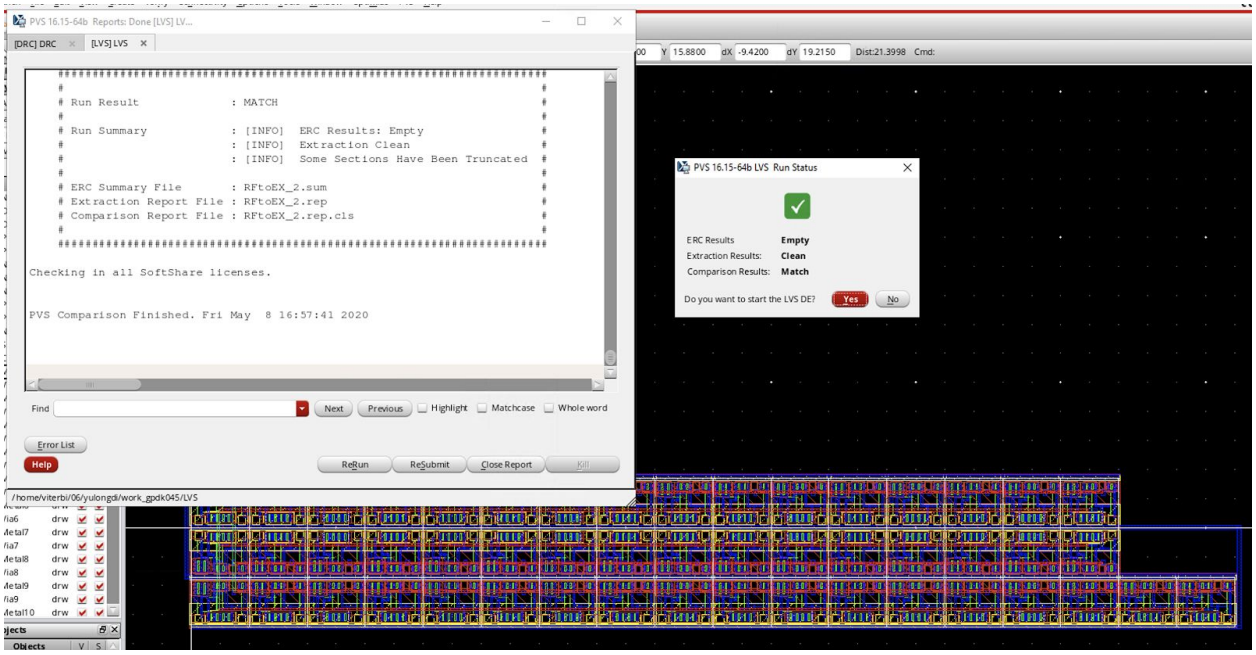
The progress of the RF layout can be seen by the following annotated schematic. The majority of the layout is completed. However, the write data input from WB stage still needs to be routed to both of the 2:1 IFRF Muxes.

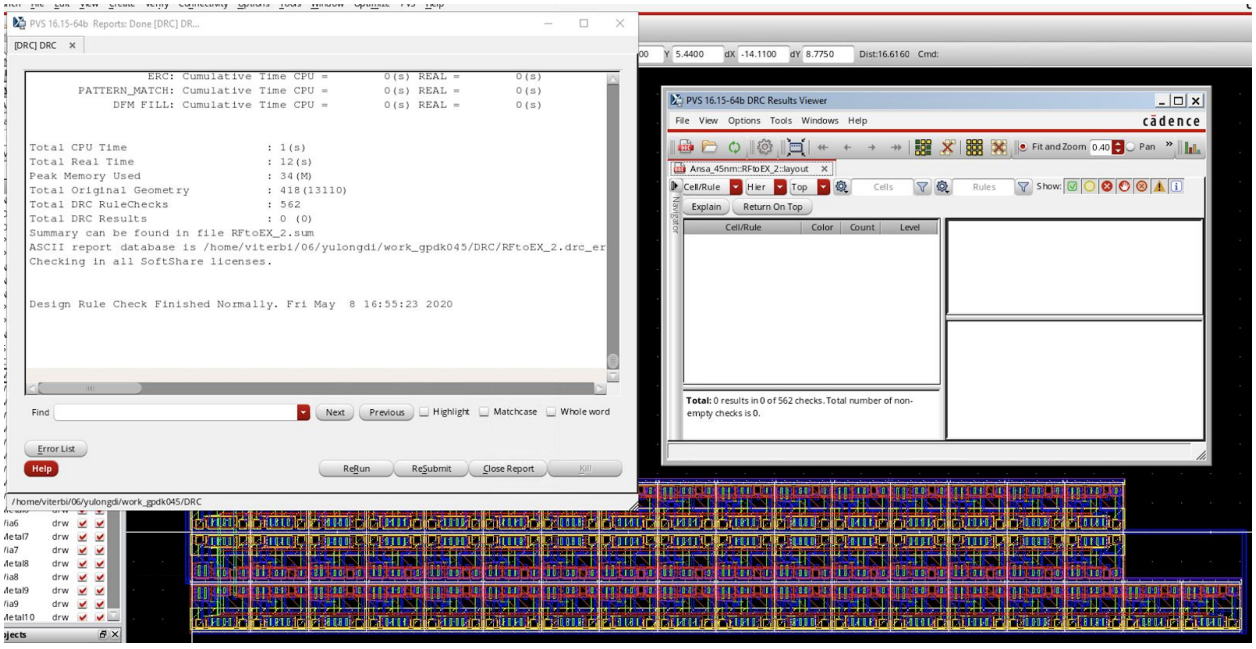




- **ID/EX Pipeline Register Layout**



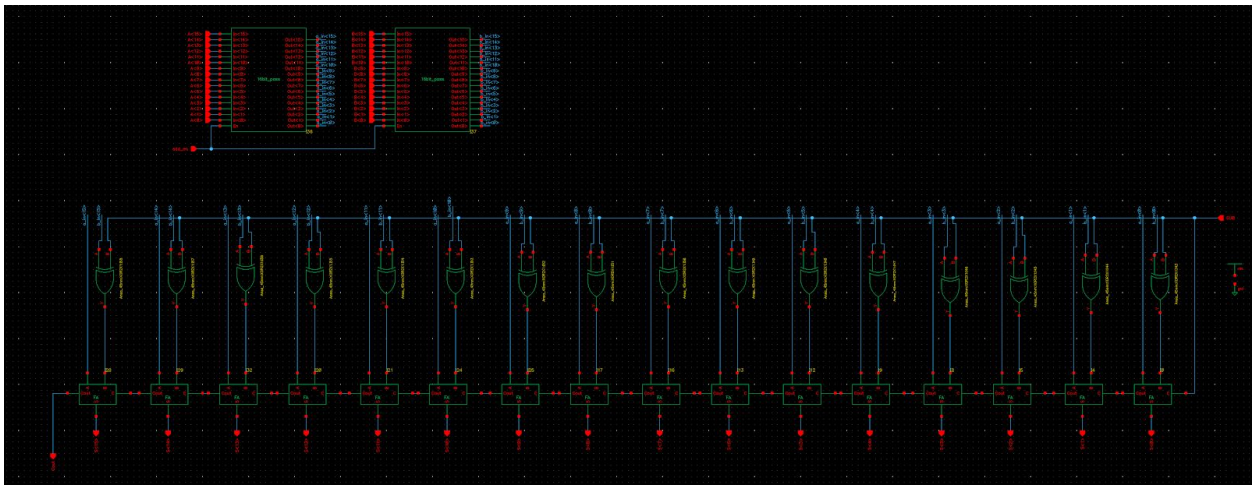


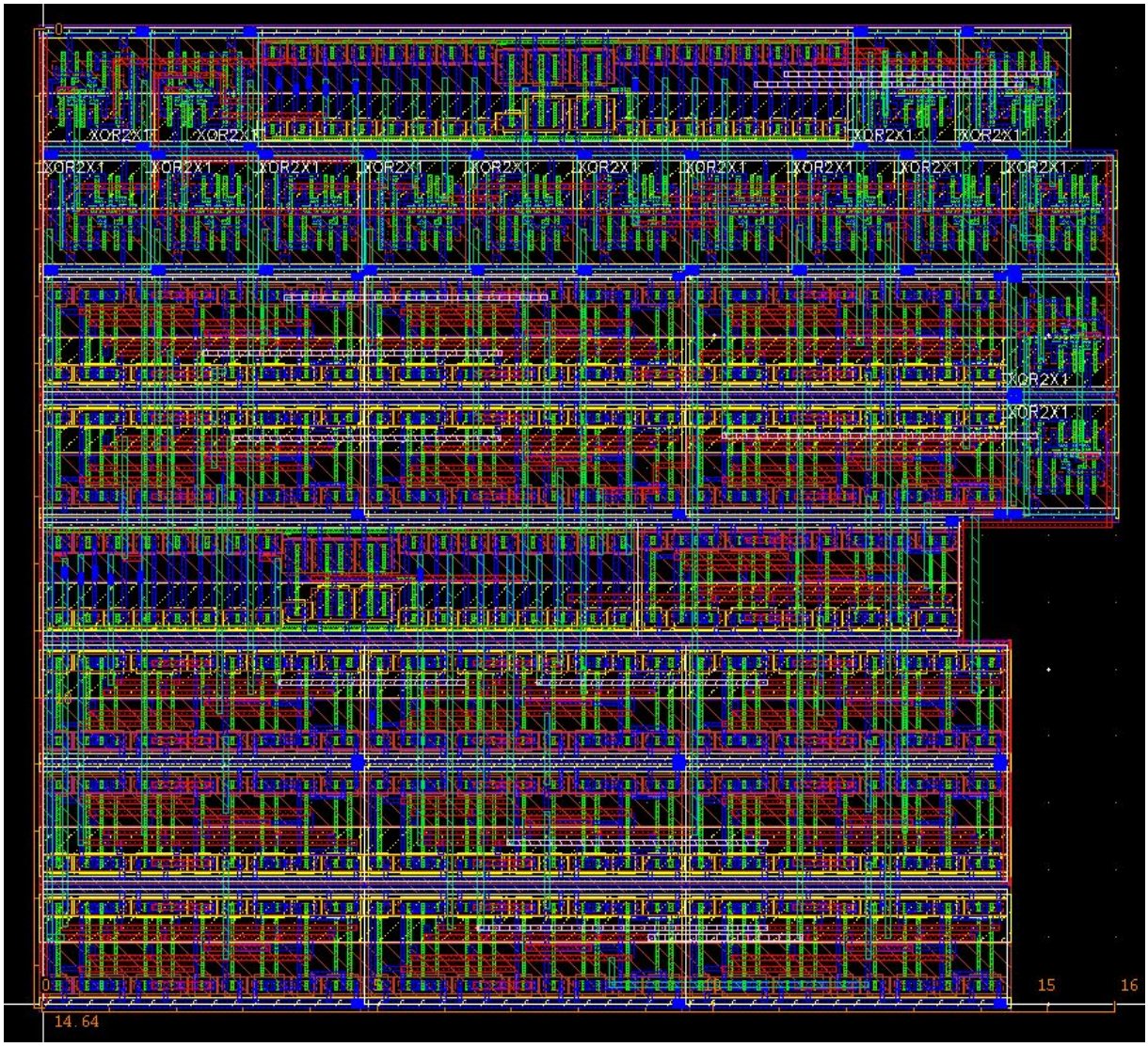


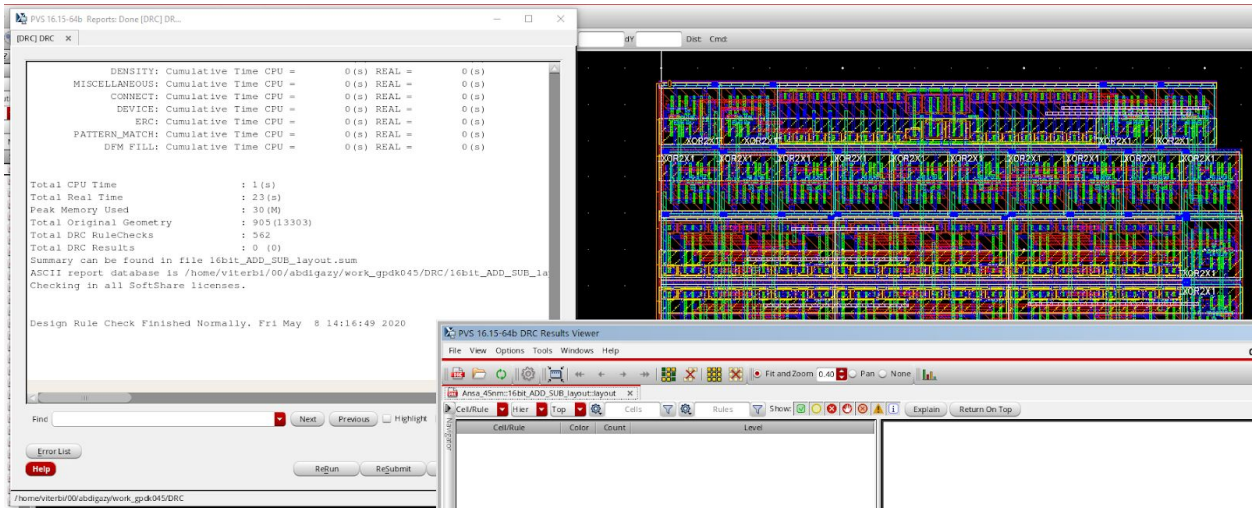
- **EX-Stage (ALU) Layout**

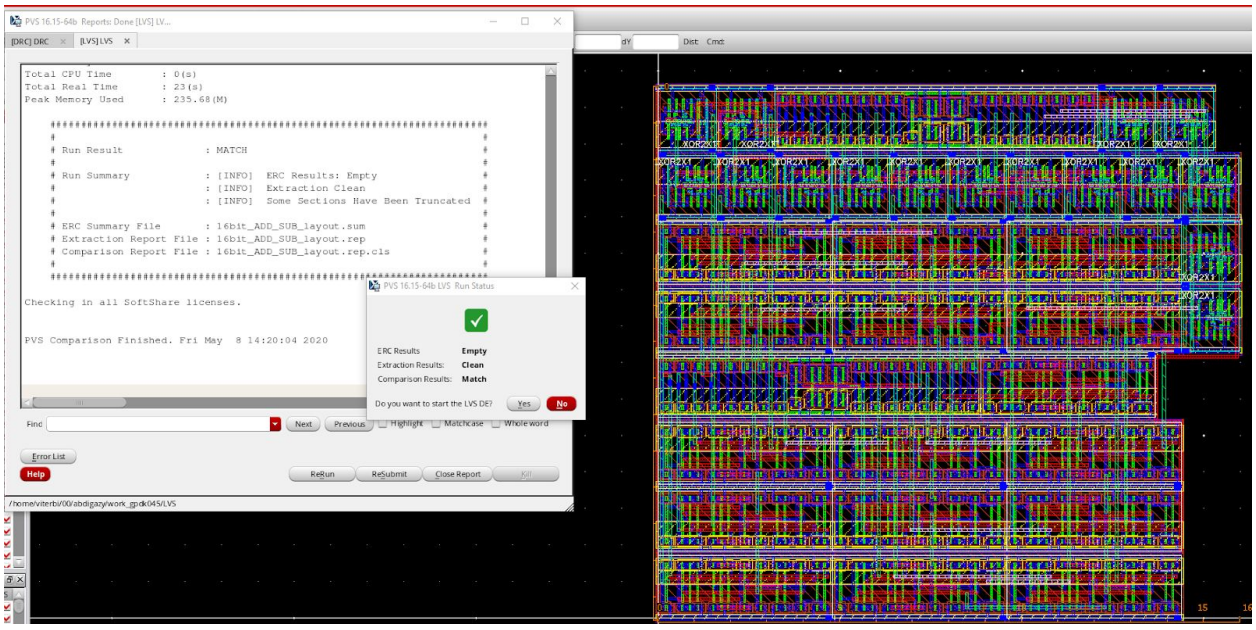
- ADD/MIN - Programmable Carry Ripple Adder/Subtractor

If MIN function is implemented, Cin = 1. The MSB of the final result is used to select the minimum of two inputs. If MSB = 1, then the result is negative and A is smaller than B. If MSB = 0, then the result is positive and B is smaller than A.

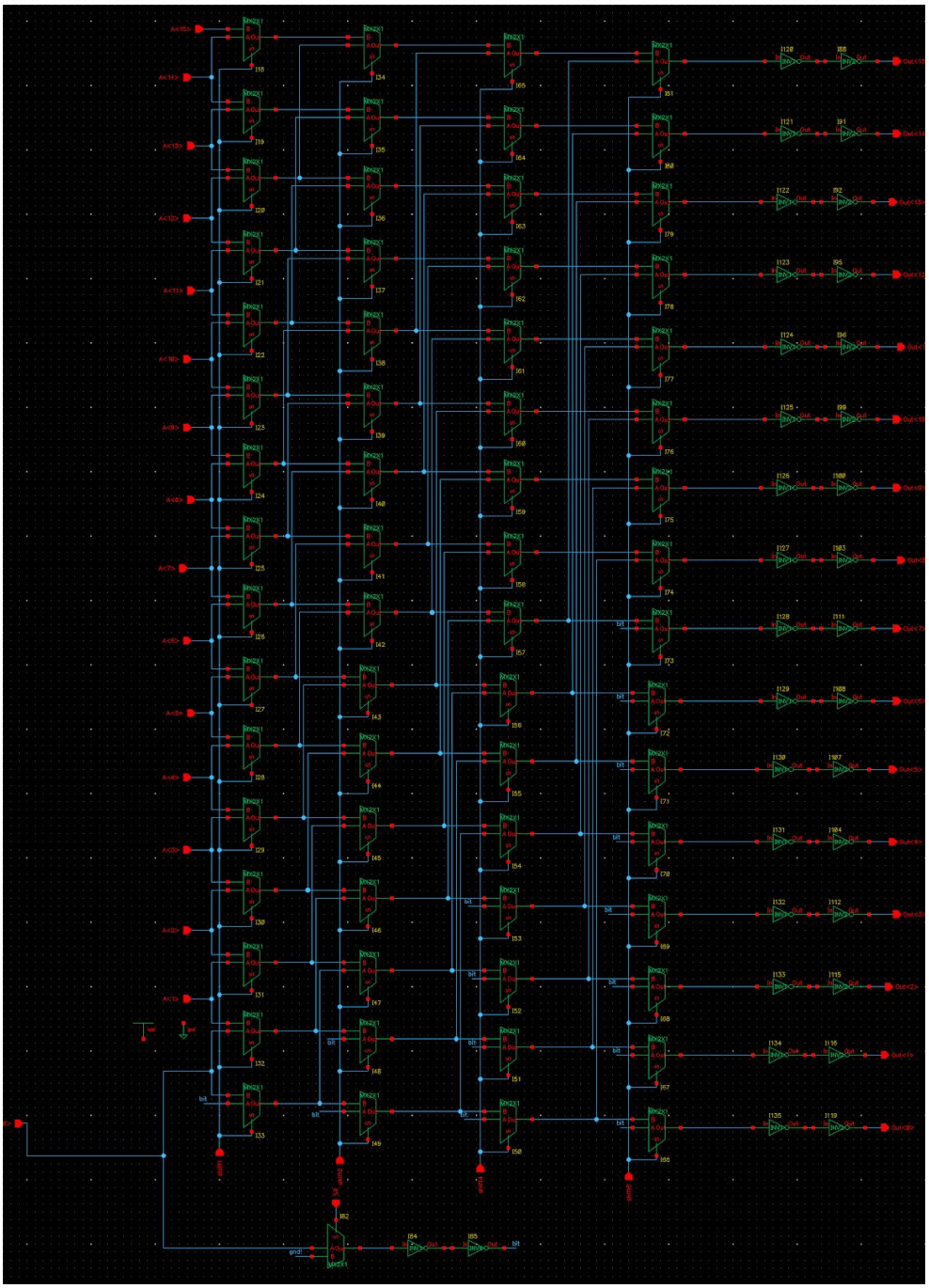


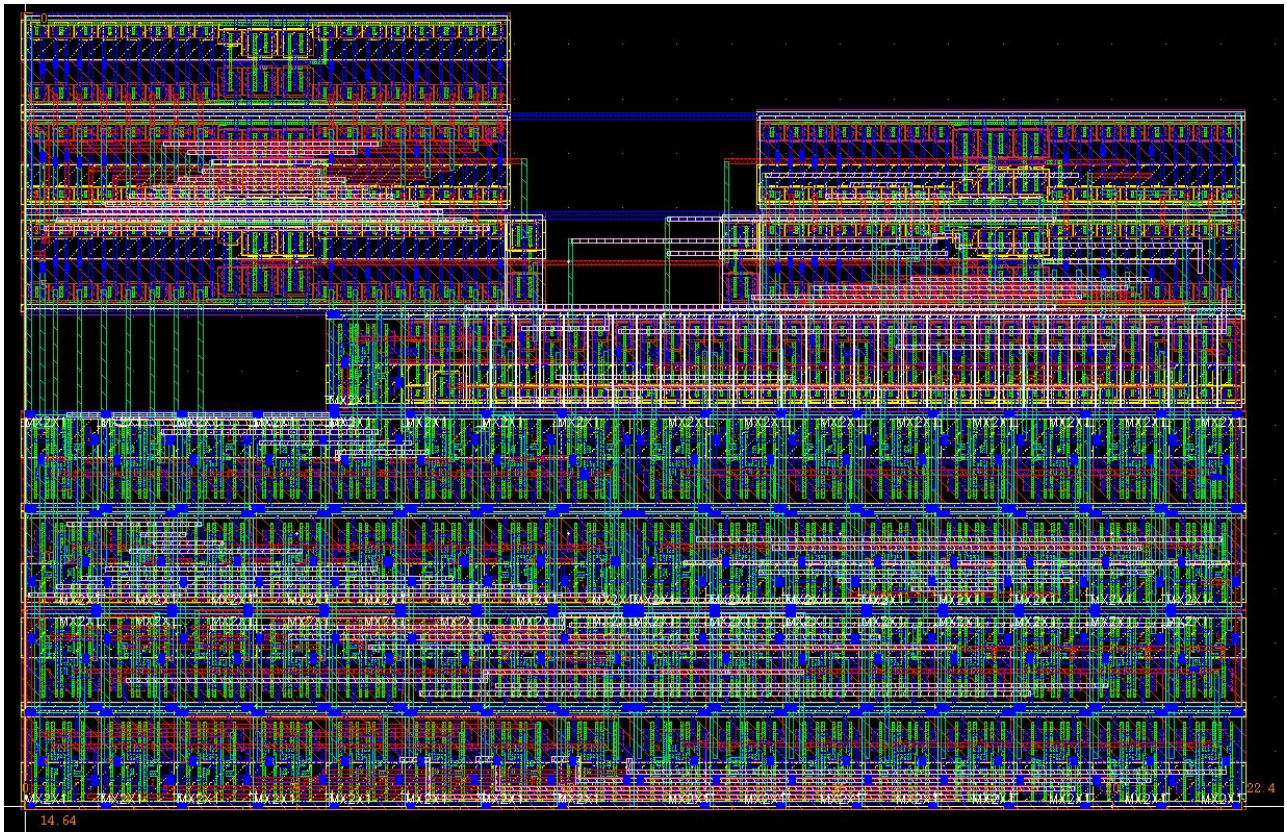

XOR2X1
15
16
14.64

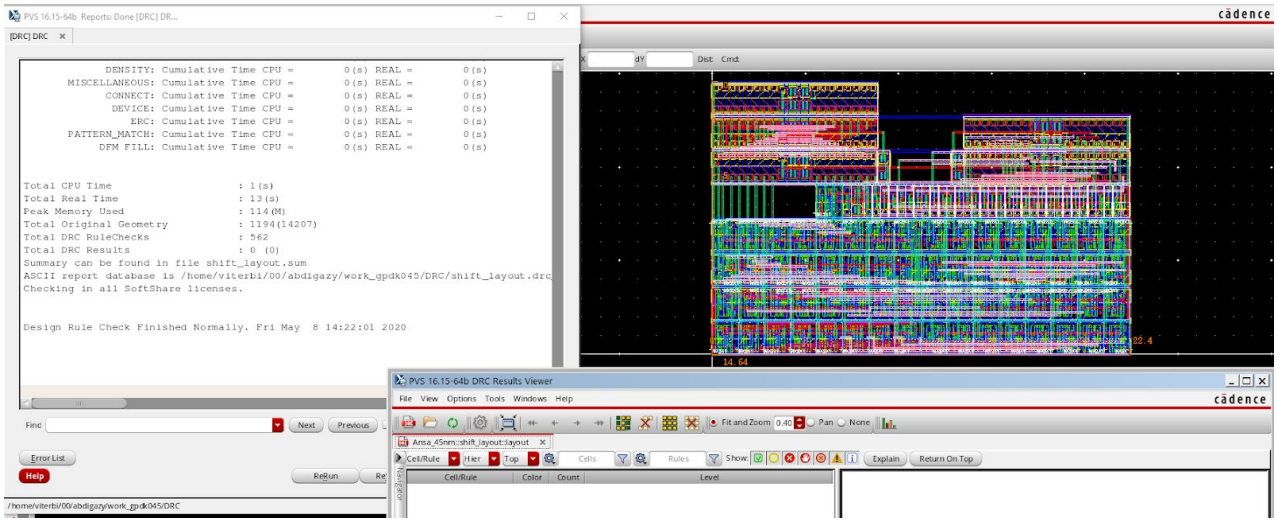

DENSITY: Cumulative Time CPU = 0(s) REAL = 0(s)
MISCELLANEOUS: Cumulative Time CPU = 0(s) REAL = 0(s)
CONNECT: Cumulative Time CPU = 0(s) REAL = 0(s)
DEVICE: Cumulative Time CPU = 0(s) REAL = 0(s)
ERC: Cumulative Time CPU = 0(s) REAL = 0(s)
PATTERN_MATCH: Cumulative Time CPU = 0(s) REAL = 0(s)
DFM FILL: Cumulative Time CPU = 0(s) REAL = 0(s)
Total CPU Time : 1(s)
Total Real Time : 23(s)
Peak Memory Used : 30(M)
Total Original Geometry : 905(13303)
Total DRC RuleChecks : 562
Total DRC Results : 0 (0)
Summary can be found in file 16bit_ADD_SUB_layout.sum
ASCII report database is /home/viterbi/00/abdigazy/work_gpdk045/DRC/16bit_ADD_SUB_la
Checking in all SoftShare licenses.
Design Rule Check Finished Normally. Fri May 8 14:16:49 2020
PVS 16.15-64b DRC Results Viewer
File View Options Tools Windows Help
Error List
Help
ReRun
ReSubmit

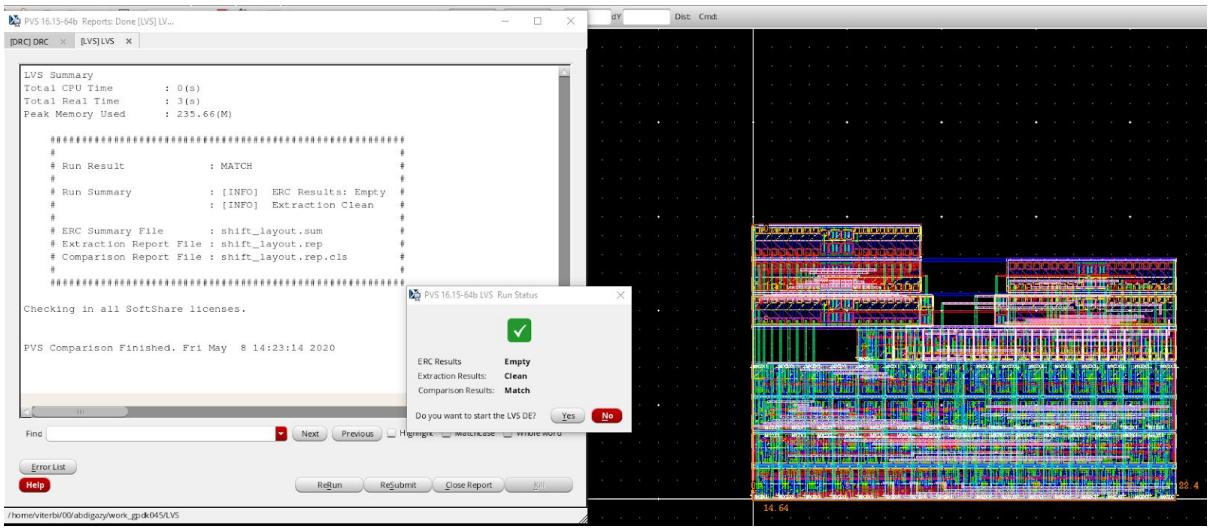


- SFL/SFR
    - The SFL/SFR submodule of the ALU performs arithmetic left/right shift on the 16-bit data input by a specified number of bits.

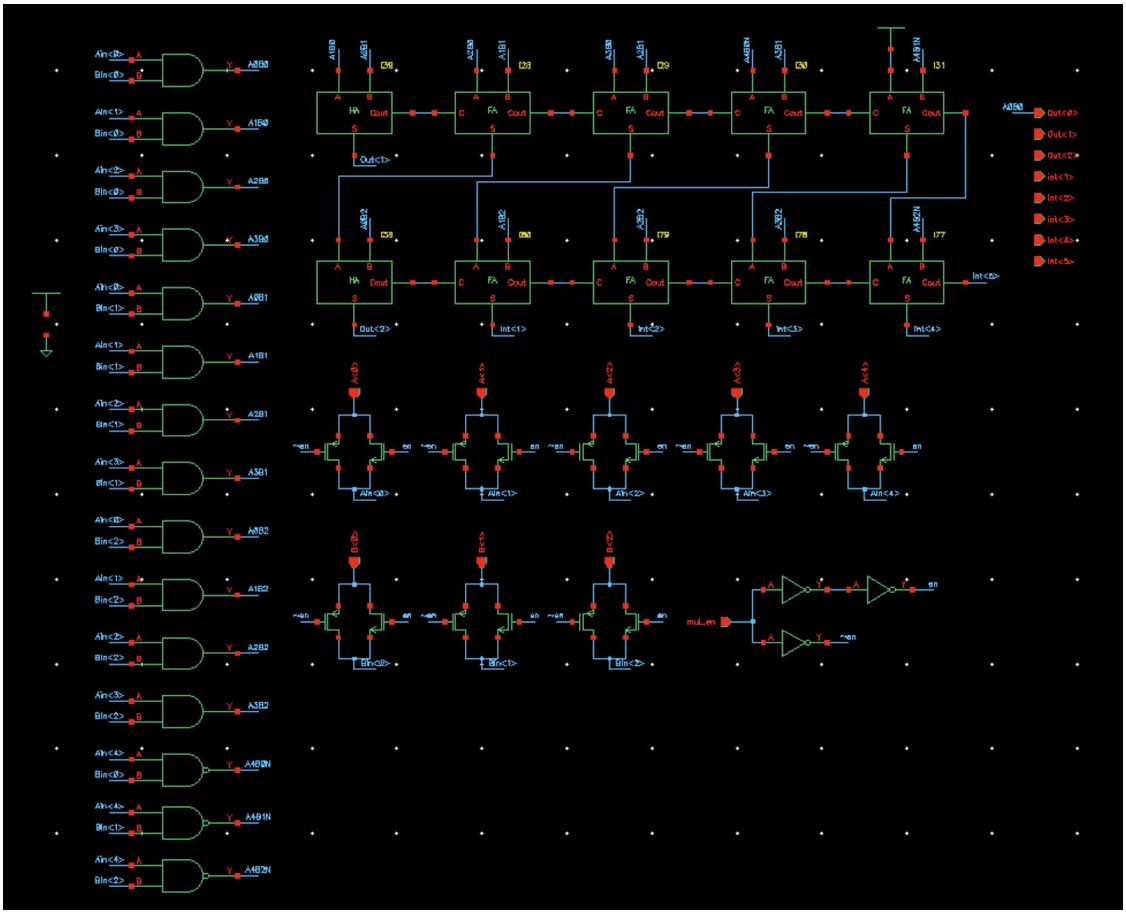


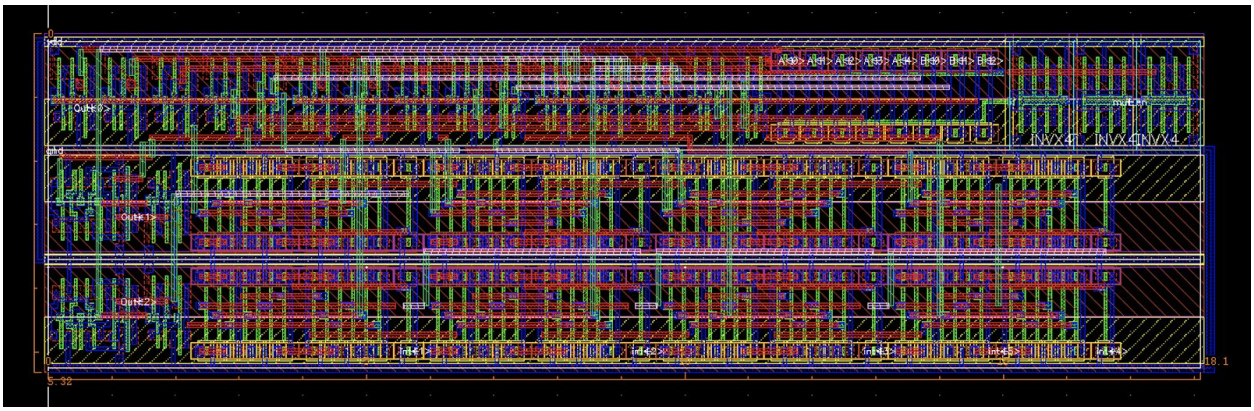


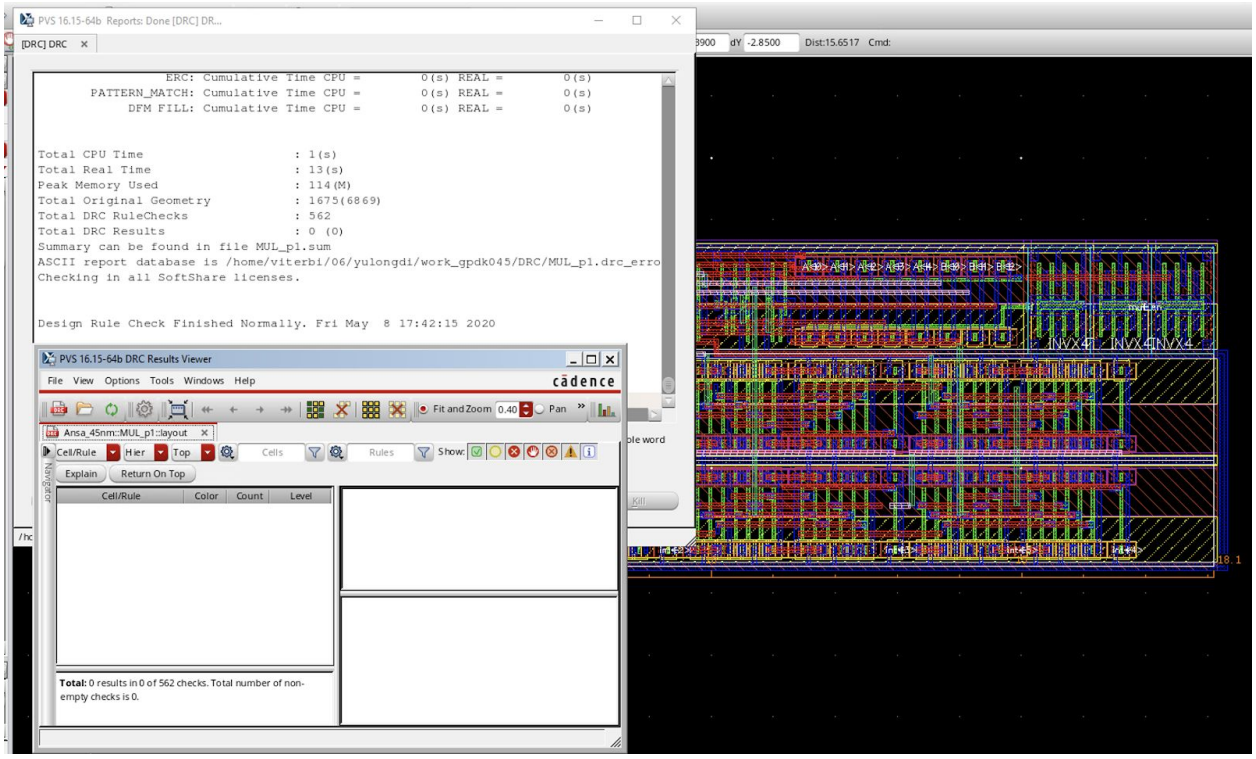

DENSITY: Cumulative Time CPU = 0(s) REAL = 0(s)
MISCELLANEOUS: Cumulative Time CPU = 0(s) REAL = 0(s)
CONNECT: Cumulative Time CPU = 0(s) REAL = 0(s)
DEVICE: Cumulative Time CPU = 0(s) REAL = 0(s)
ERC: Cumulative Time CPU = 0(s) REAL = 0(s)
PATTERN_MATCH: Cumulative Time CPU = 0(s) REAL = 0(s)
DFM FILL: Cumulative Time CPU = 0(s) REAL = 0(s)
Total CPU Time : 1(s)
Total Real Time : 13(s)
Peak Memory Used : 114(M)
Total Original Geometry : 1194(14207)
Total DRC RuleChecks : 562
Total DRC Results : 0 (0)
Summary can be found in file shift_layout.sum
ASCII report database is /home/viterbi/00/abdigazy/work_gpdk045/DRC/shift_layout.drc
Checking in all SoftShare licenses.
Design Rule Check Finished Normally. Fri May 8 14:22:01 2020
PVS 16.15-64b DRC Results Viewer
cadence

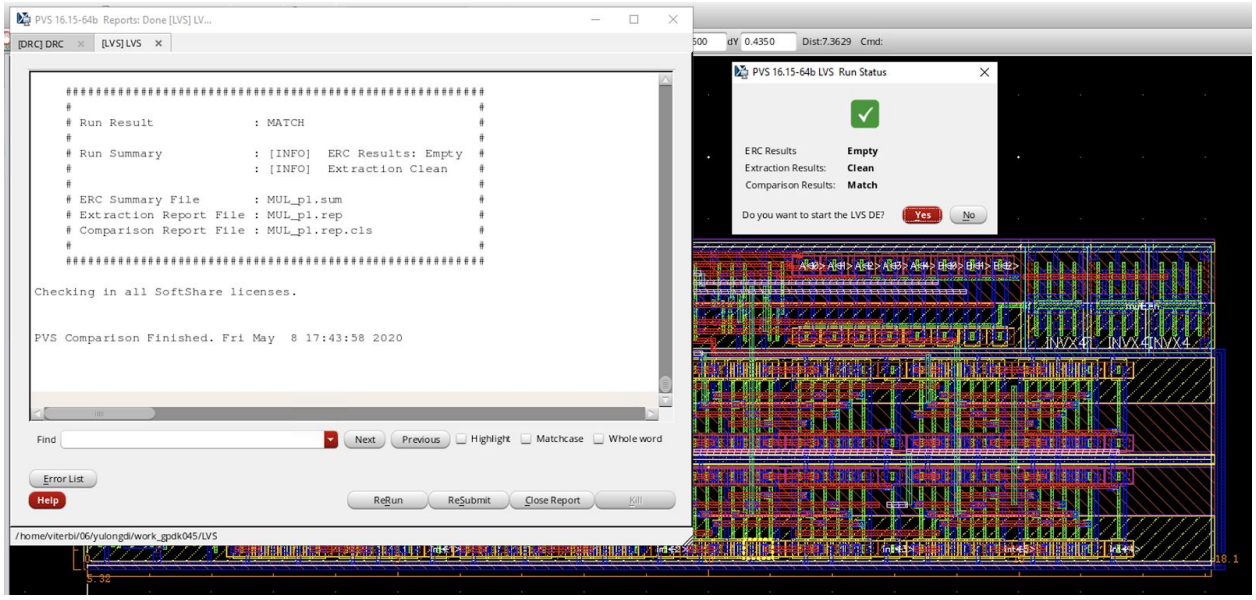


- MUL Stage 1
  - The pipelined 2’s complement 5-bit multiplier contains 2 separate stages to reduce the combinational critical path of the circuit. The first stage resides in the EX stage of the processor along with the rest of the ALU submodules before the EX/MEM stage register.

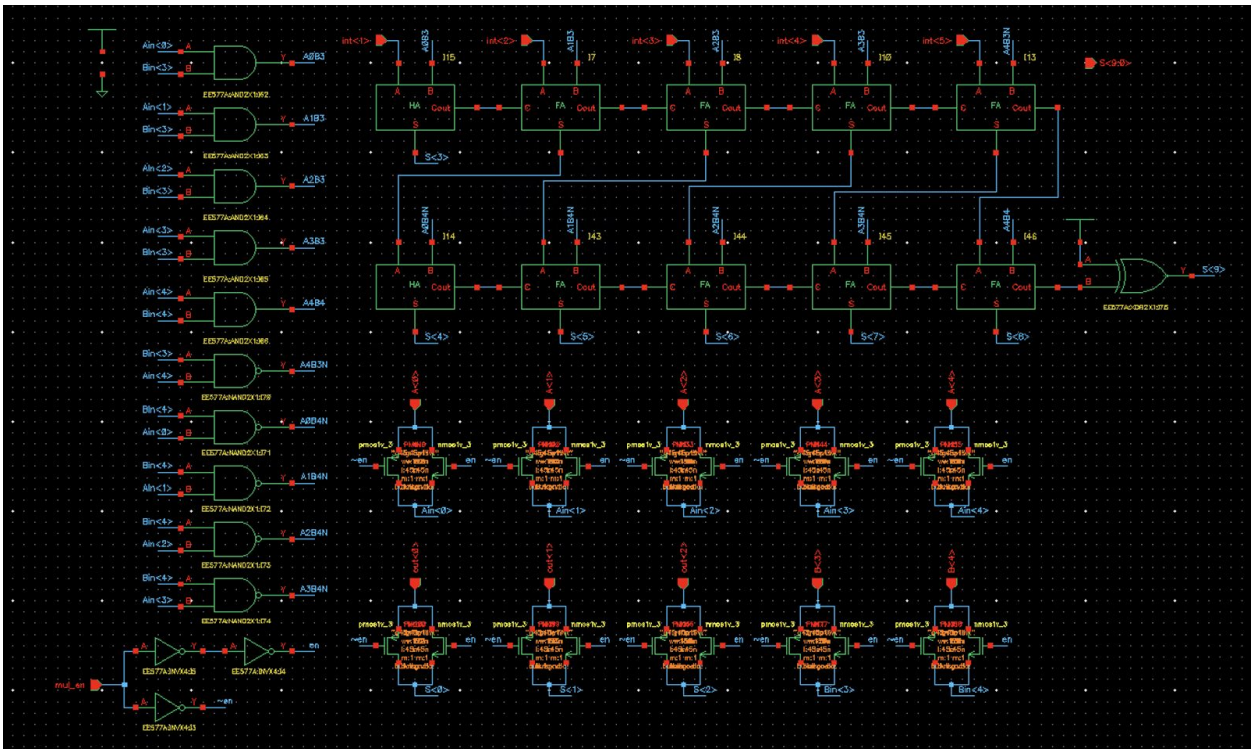


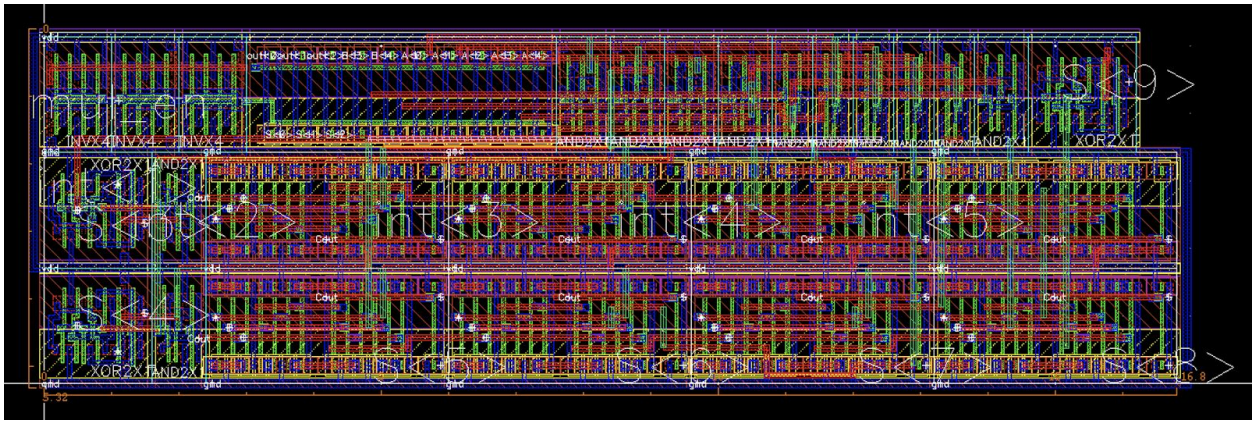


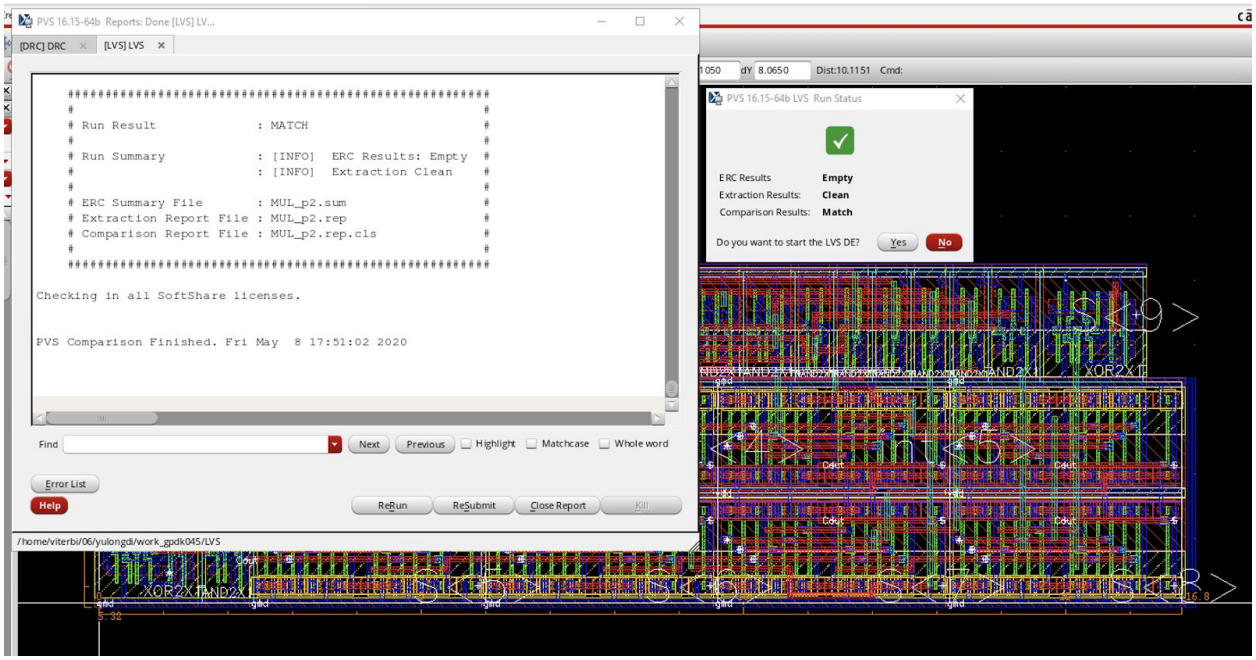

PVS 16.15-64b Reports: Done [DRC] DR...
[DRC] DRC
ERC: Cumulative Time CPU = 0(s) REAL = 0(s)
PATTERN_MATCH: Cumulative Time CPU = 0(s) REAL = 0(s)
DFM FILL: Cumulative Time CPU = 0(s) REAL = 0(s)
Total CPU Time : 1(s)
Total Real Time : 13(s)
Peak Memory Used : 114(M)
Total Original Geometry : 1675(6869)
Total DRC RuleChecks : 562
Total DRC Results : 0 (0)
Summary can be found in file MUL_p1.sum
ASCII report database is /home/viterbi/06/yulongdi/work_gpdk045/DRC/MUL_p1.drc_erro
Checking in all SoftShare licenses.
Design Rule Check Finished Normally. Fri May 8 17:42:15 2020
PVS 16.15-64b DRC Results Viewer
File View Options Tools Windows Help
cadence
Ansa_45nm::MUL_p1::layout
Explain
Return On Top
Cell/Rule
Color
Count
Level
Total: 0 results in 0 of 562 checks. Total number of non-empty checks is 0.
dY -2.8500
Dist:15.6517
Cmd:

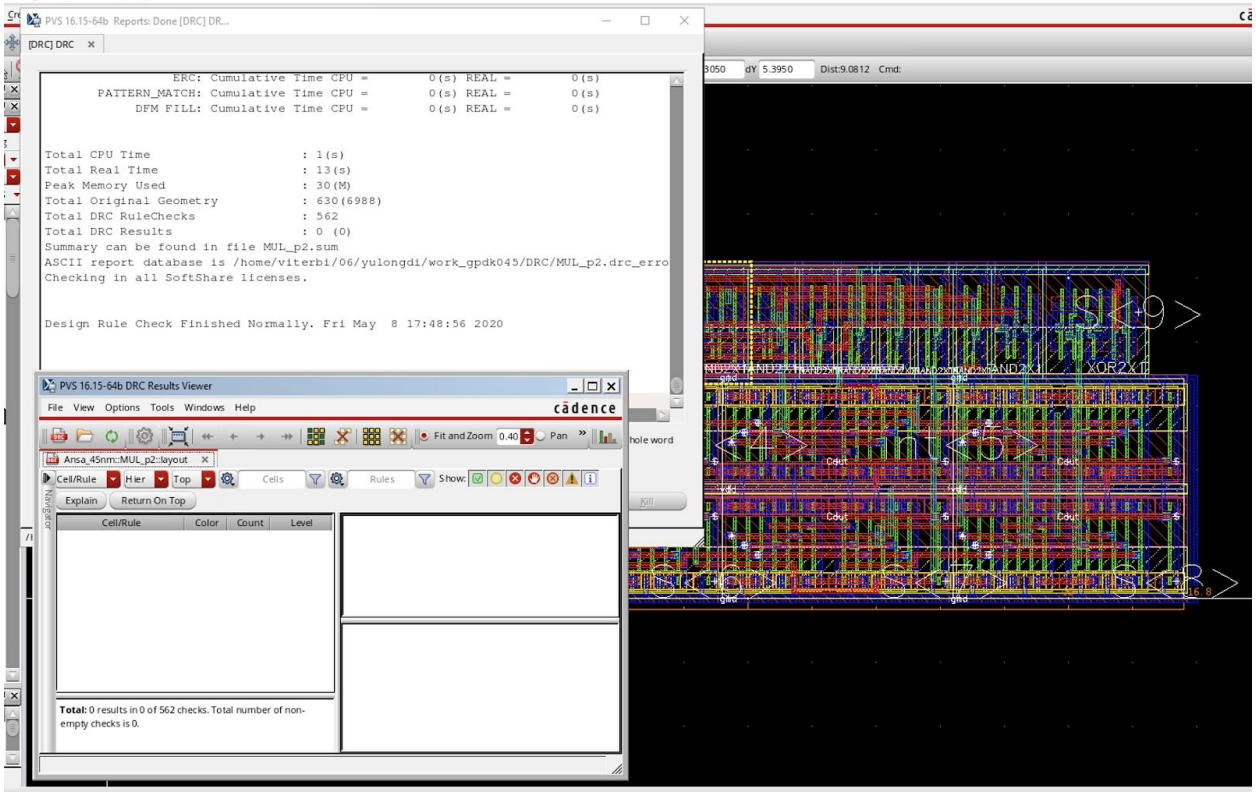

PVS 16.15-64b Reports: Done [LVS] LV...
[DRC] DRC
[LVS] LVS
Run Result : MATCH
Run Summary : [INFO] ERC Results: Empty
: [INFO] Extraction Clean
ERC Summary File : MUL_p1.sum
Extraction Report File : MUL_p1.rep
Comparison Report File : MUL_p1.rep.cls
Checking in all SoftShare licenses.
PVS Comparison Finished. Fri May 8 17:43:58 2020
Find
Next
Previous
Highlight
Matchcase
Whole word
Error List
Help
ReRun
ReSubmit
Close Report
Kill
/home/viterbi/06/yulongdi/work_gpdk045/LVS
PVS 16.15-64b LVS Run Status
ERC Results Empty
Extraction Results: Clean
Comparison Results: Match
Do you want to start the LVS DE?
Yes
No
dY 0.4350
Dist:7.3629
Cmd:

- MUL Stage 2
    - The second stage of the pipelined multiplier resides in the MEM stage of the processor along with the SRAM after the EX/MEM stage register.

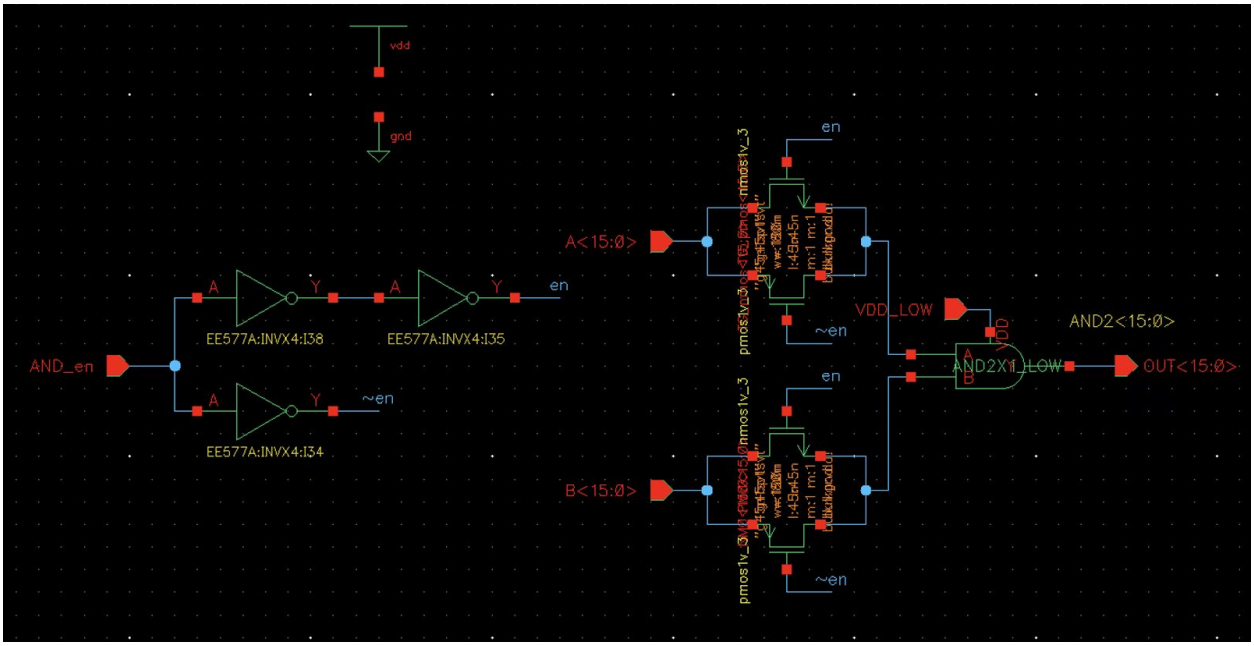


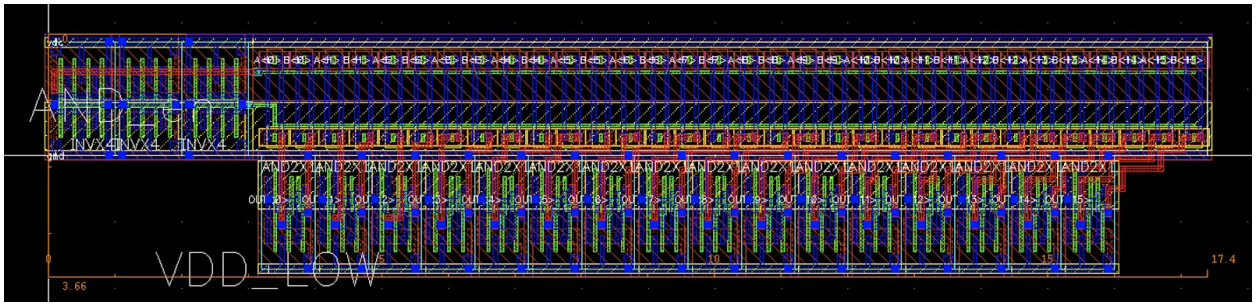


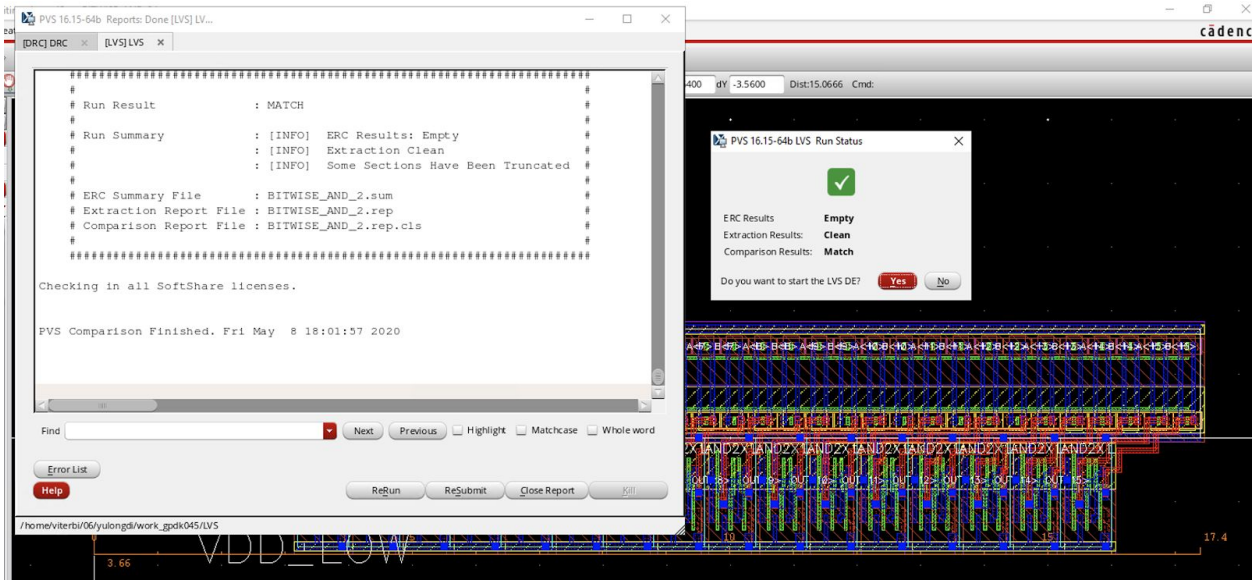

PVS 16.15-64b Reports: Done [LVS] LV...
[DRC] DRC
[LVS] LVS
Run Result : MATCH
Run Summary : [INFO] ERC Results: Empty
: [INFO] Extraction Clean
ERC Summary File : MUL_p2.sum
Extraction Report File : MUL_p2.rep
Comparison Report File : MUL_p2.rep.cls
Checking in all SoftShare licenses.
PVS Comparison Finished. Fri May 8 17:51:02 2020
Find
Next
Previous
Highlight
Matchcase
Whole word
Error List
Help
ReRun
ReSubmit
Close Report
Kill
/home/viterbi/06/yulongdi/work_gpdk045/LVS
PVS 16.15-64b LVS Run Status
ERC Results Empty
Extraction Results: Clean
Comparison Results: Match
Do you want to start the LVS DE?
Yes
No
Dist:10.1151
Cmd:

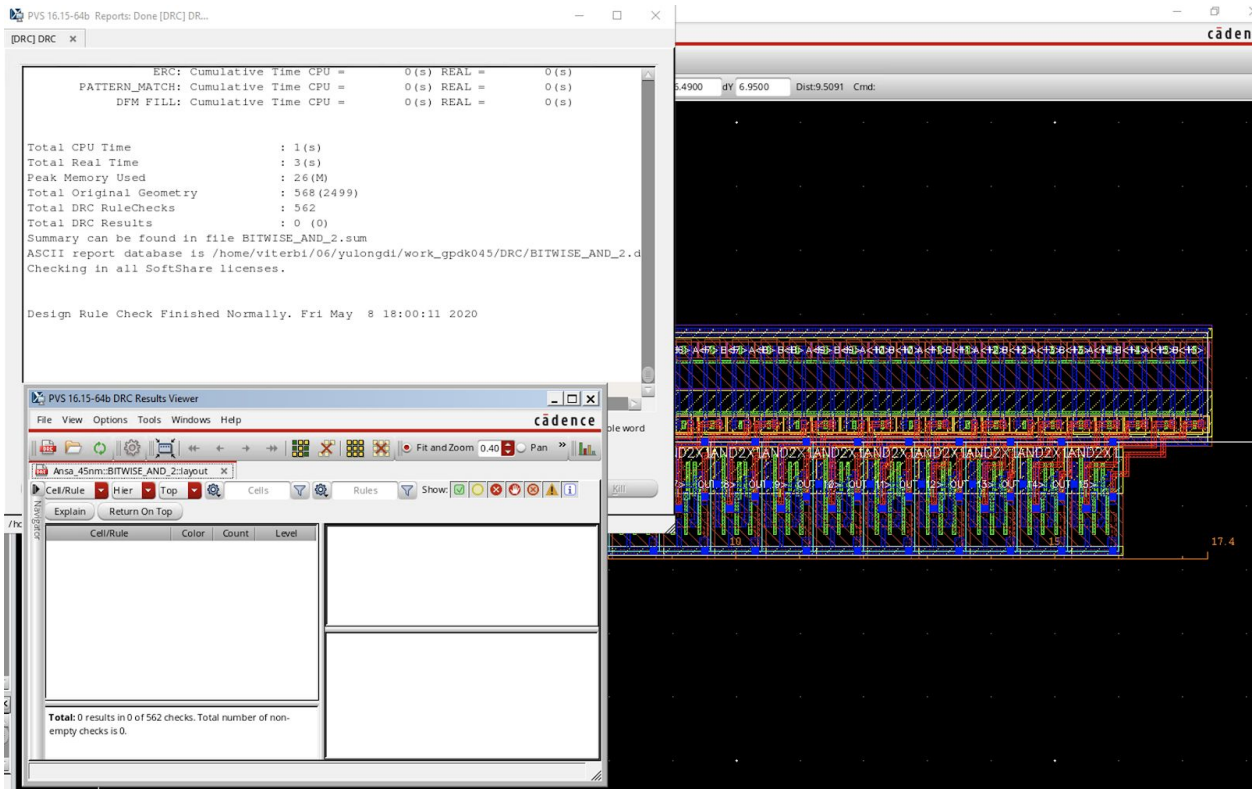

PVS 16.15-64b Reports: Done [DRC] DR...
[DRC] DRC
ERC: Cumulative Time CPU = 0(s) REAL = 0(s)
PATTERN_MATCH: Cumulative Time CPU = 0(s) REAL = 0(s)
DFM FILL: Cumulative Time CPU = 0(s) REAL = 0(s)
Total CPU Time : 1(s)
Total Real Time : 13(s)
Peak Memory Used : 30(M)
Total Original Geometry : 630(6988)
Total DRC RuleChecks : 562
Total DRC Results : 0 (0)
Summary can be found in file MUL_p2.sum
ASCII report database is /home/viterbi/06/yulongdi/work_gpdk045/DRC/MUL_p2.drc_erro
Checking in all SoftShare licenses.
Design Rule Check Finished Normally. Fri May 8 17:48:56 2020
PVS 16.15-64b DRC Results Viewer
File View Options Tools Windows Help
cadence
Fit and Zoom
Pan
Ansa_45nm::MUL_p2::layout
Cell/Rule
Hier
Top
Cells
Rules
Show:
Explain
Return On Top
Cell/Rule
Color
Count
Level
Total: 0 results in 0 of 562 checks. Total number of non-empty checks is 0.
Dist:9.0812
Cmd:

- Bitwise AND

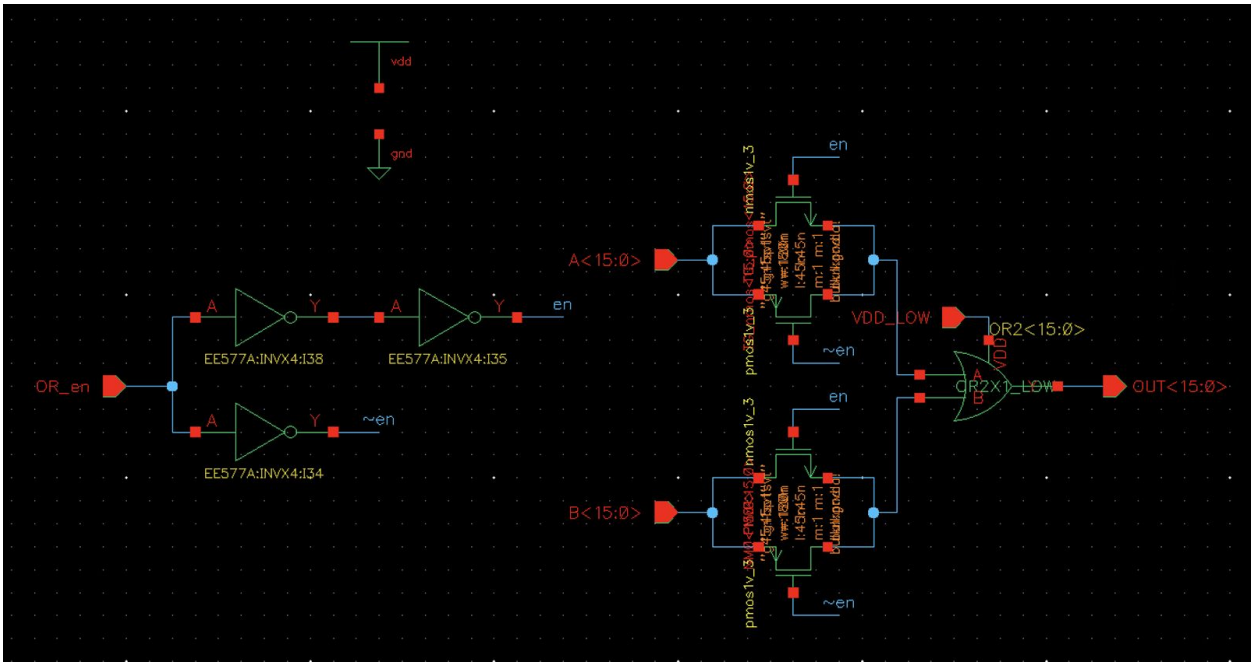

vdd
gnd
A<15:0>
B<15:0>
AND_en
EE577A:INVX4:I38
EE577A:INVX4:I35
EE577A:INVX4:I34
en
~en
VDD_LOW
AND2X1_LOW
AND2<15:0>
OUT<15:0>

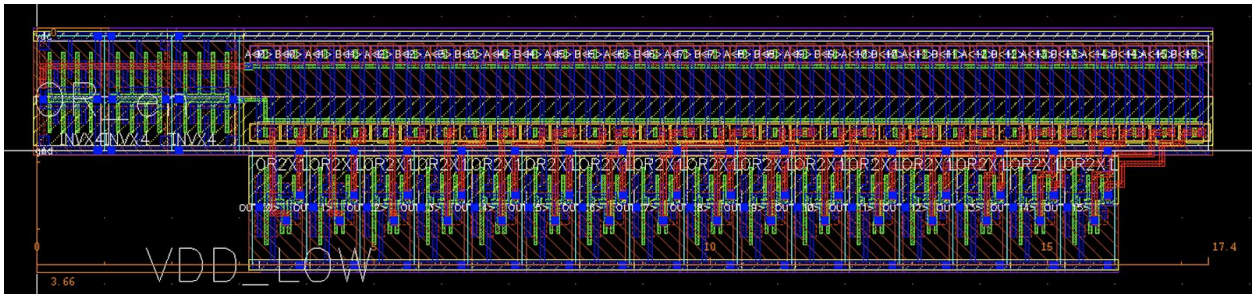

AND_en
INVX4
AND2X1
VDD_LOW
3.66
17.4

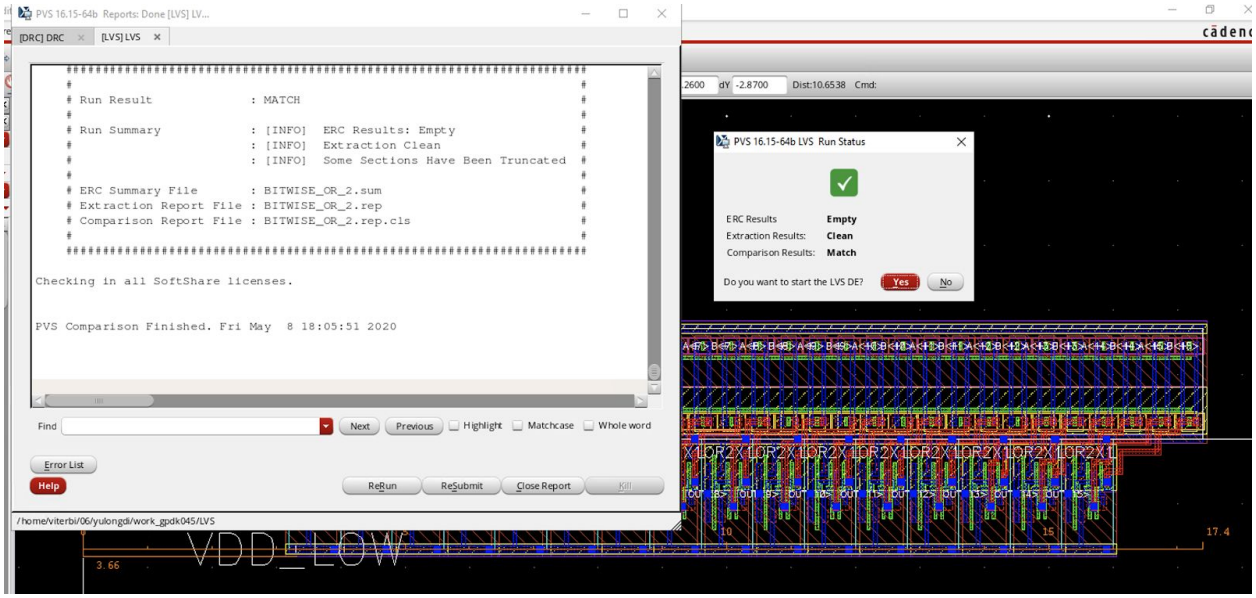

PVS 16.15-64b Reports: Done [LVS] LV...
[DRC] DRC
[LVS] LVS
Run Result : MATCH
Run Summary : [INFO] ERC Results: Empty
: [INFO] Extraction Clean
: [INFO] Some Sections Have Been Truncated
ERC Summary File : BITWISE_AND_2.sum
Extraction Report File : BITWISE_AND_2.rep
Comparison Report File : BITWISE_AND_2.rep.cls
Checking in all SoftShare licenses.
PVS Comparison Finished. Fri May 8 18:01:57 2020
Find
Next
Previous
Highlight
Matchcase
Whole word
Error List
Help
ReRun
ReSubmit
Close Report
Kill
/home/viterbi/06/yulongdi/work_gpdk045/LVS
PVS 16.15-64b LVS Run Status
ERC Results Empty
Extraction Results: Clean
Comparison Results: Match
Do you want to start the LVS DE?
Yes
No
Dist:15.0666
VDD_LOW
3.66
17.4

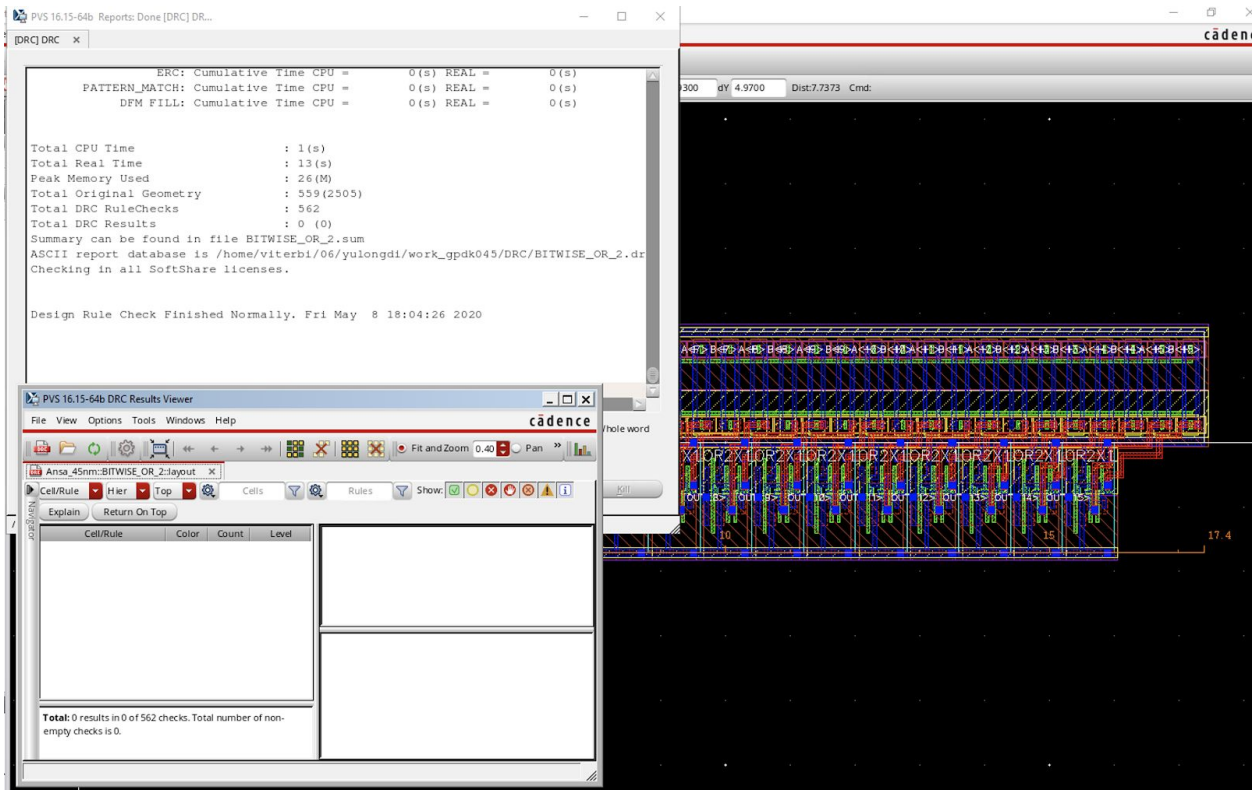

PVS 16.15-64b Reports: Done [DRC] DR...
[DRC] DRC
ERC: Cumulative Time CPU = 0(s) REAL = 0(s)
PATTERN_MATCH: Cumulative Time CPU = 0(s) REAL = 0(s)
DFM FILL: Cumulative Time CPU = 0(s) REAL = 0(s)
Total CPU Time : 1(s)
Total Real Time : 3(s)
Peak Memory Used : 26(M)
Total Original Geometry : 568(2499)
Total DRC RuleChecks : 562
Total DRC Results : 0 (0)
Summary can be found in file BITWISE_AND_2.sum
ASCII report database is /home/viterbi/06/yulongdi/work_gpdk045/DRC/BITWISE_AND_2.d
Checking in all SoftShare licenses.
Design Rule Check Finished Normally. Fri May 8 18:00:11 2020
PVS 16.15-64b DRC Results Viewer
File View Options Tools Windows Help
Fit and Zoom
Pan
Ansa_45nm::BITWISE_AND_2::layout
Cell/Rule
Hier
Top
Cells
Rules
Show:
Explain
Return On Top
Cell/Rule
Color
Count
Level
Total: 0 results in 0 of 562 checks. Total number of non-empty checks is 0.
Dist:9.5091
17.4

- Bitwise OR

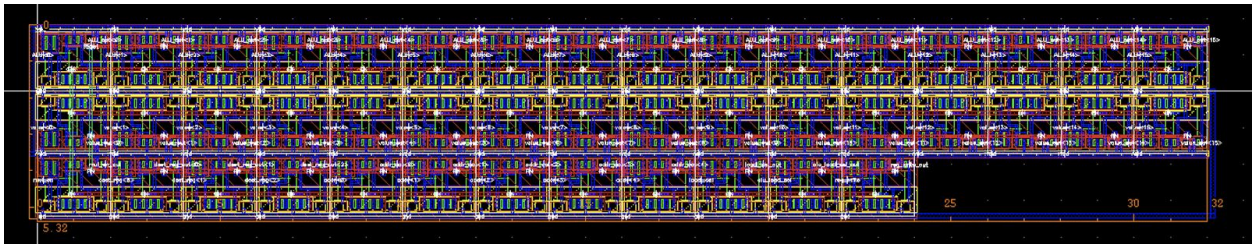

vdd
gnd
A<15:0>
B<15:0>
en
~en
VDD_LOW
OR2<15:0>
OUT<15:0>
OR_en
EE577A:INVX4:I38
EE577A:INVX4:I35
EE577A:INVX4:I34
OR2X1_LOW

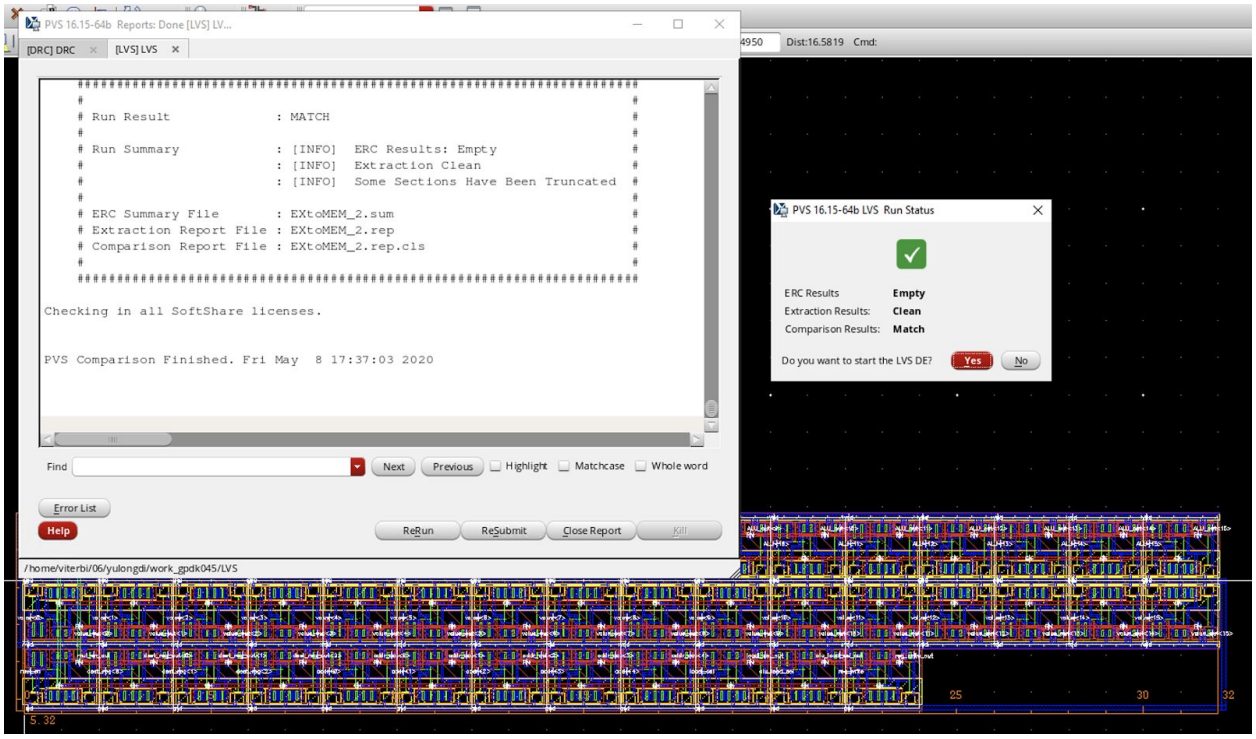

VDD_LOW

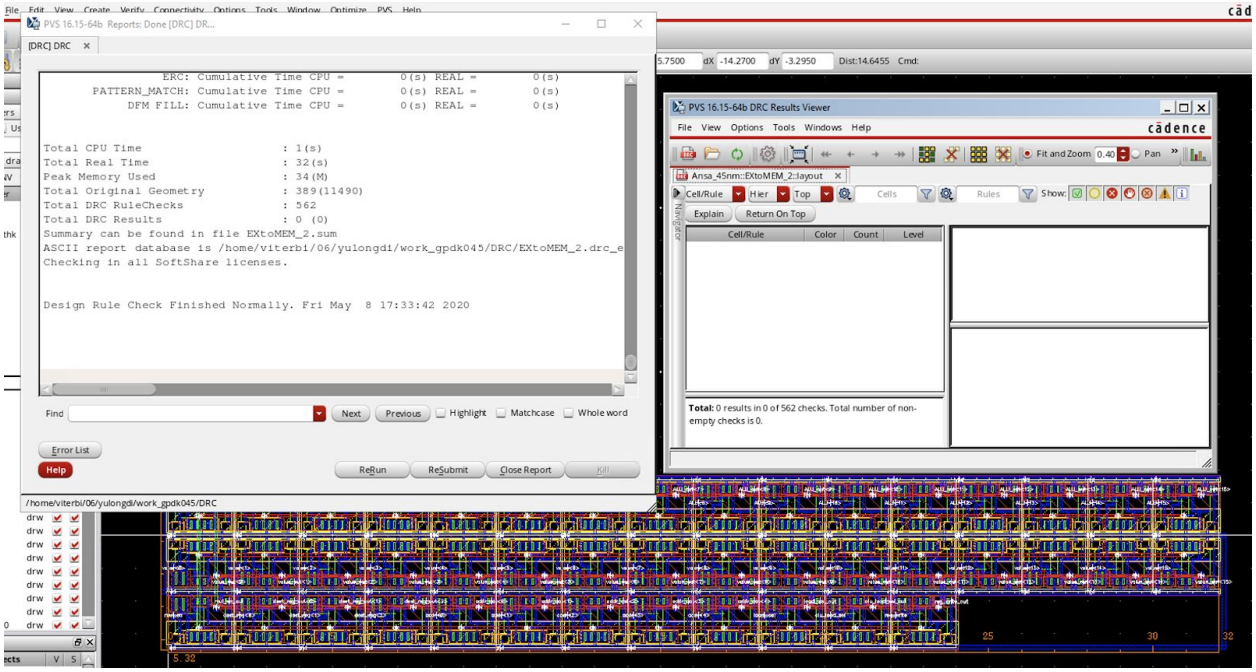




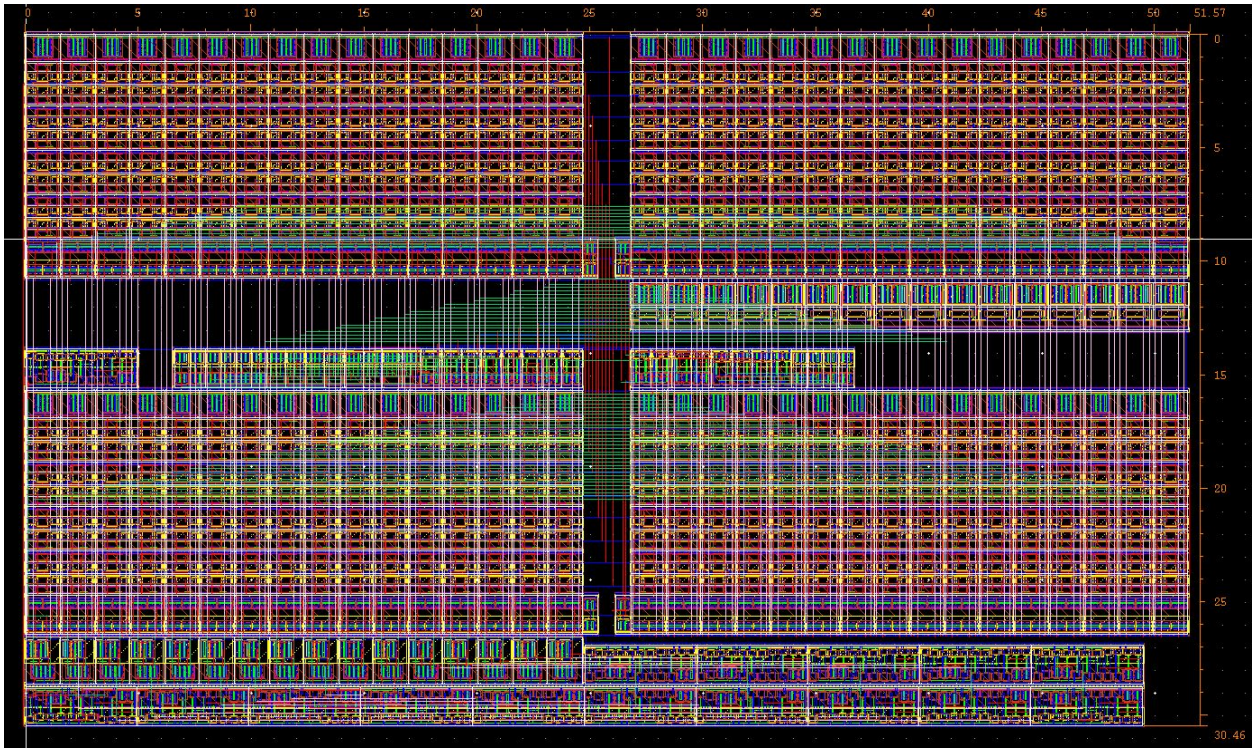
- **EX/MEM Pipeline Register Layout**

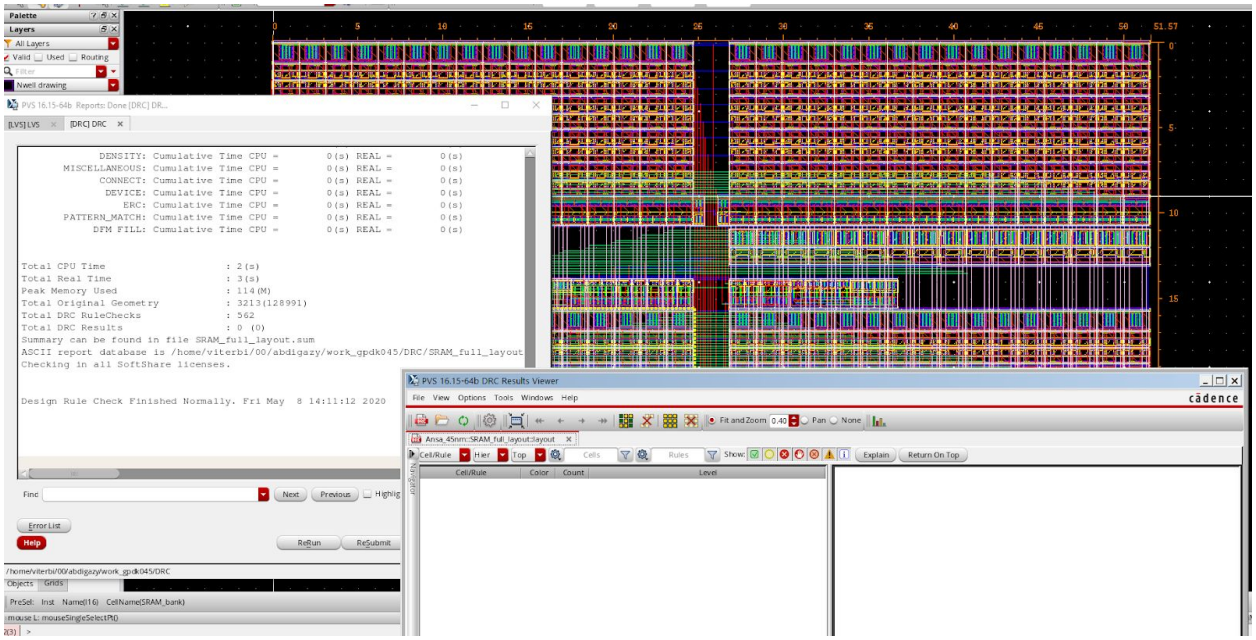


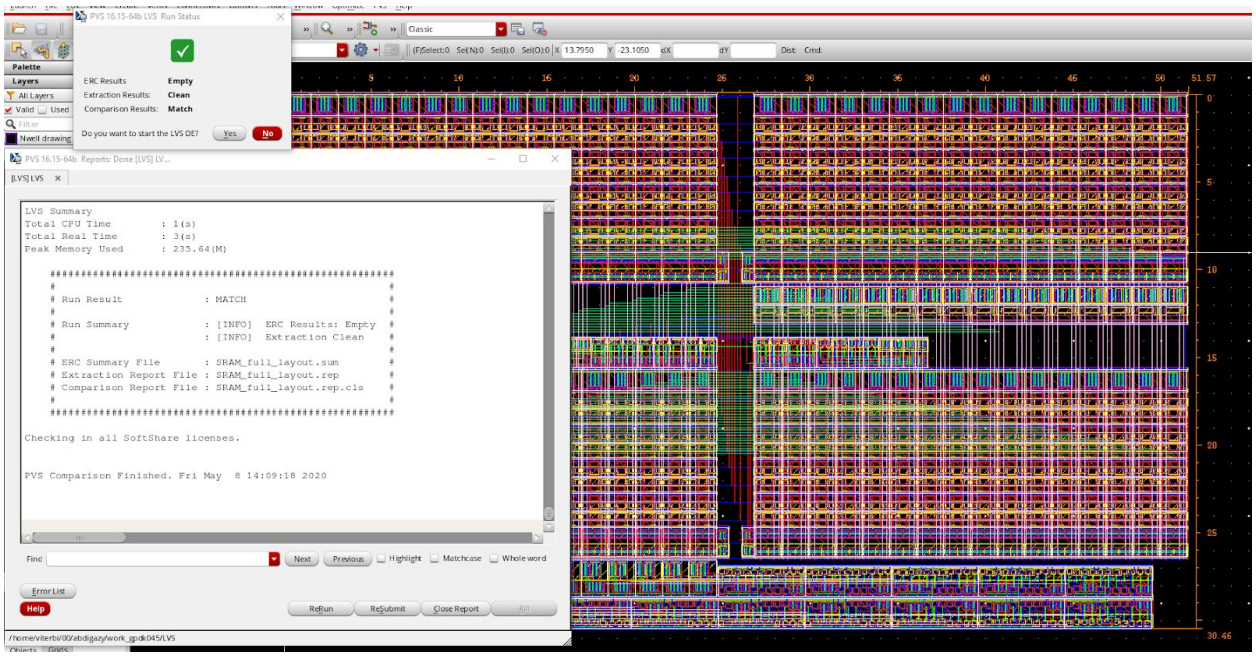

PVS 16.15-64b Reports: Done [LVS] LV...
[DRC] DRC
[LVS] LVS
Run Result : MATCH
Run Summary : [INFO] ERC Results: Empty
: [INFO] Extraction Clean
: [INFO] Some Sections Have Been Truncated
ERC Summary File : EXtoMEM_2.sum
Extraction Report File : EXtoMEM_2.rep
Comparison Report File : EXtoMEM_2.rep.cls
Checking in all SoftShare licenses.
PVS Comparison Finished. Fri May 8 17:37:03 2020
Find
Next
Previous
Highlight
Matchcase
Whole word
Error List
Help
ReRun
ReSubmit
Close Report
Kill
/home/viterbi/06/yulongdi/work_gpdk045/LVS
Dist:16.5819 Cmd:
PVS 16.15-64b LVS Run Status
ERC Results Empty
Extraction Results: Clean
Comparison Results: Match
Do you want to start the LVS DE?
Yes
No



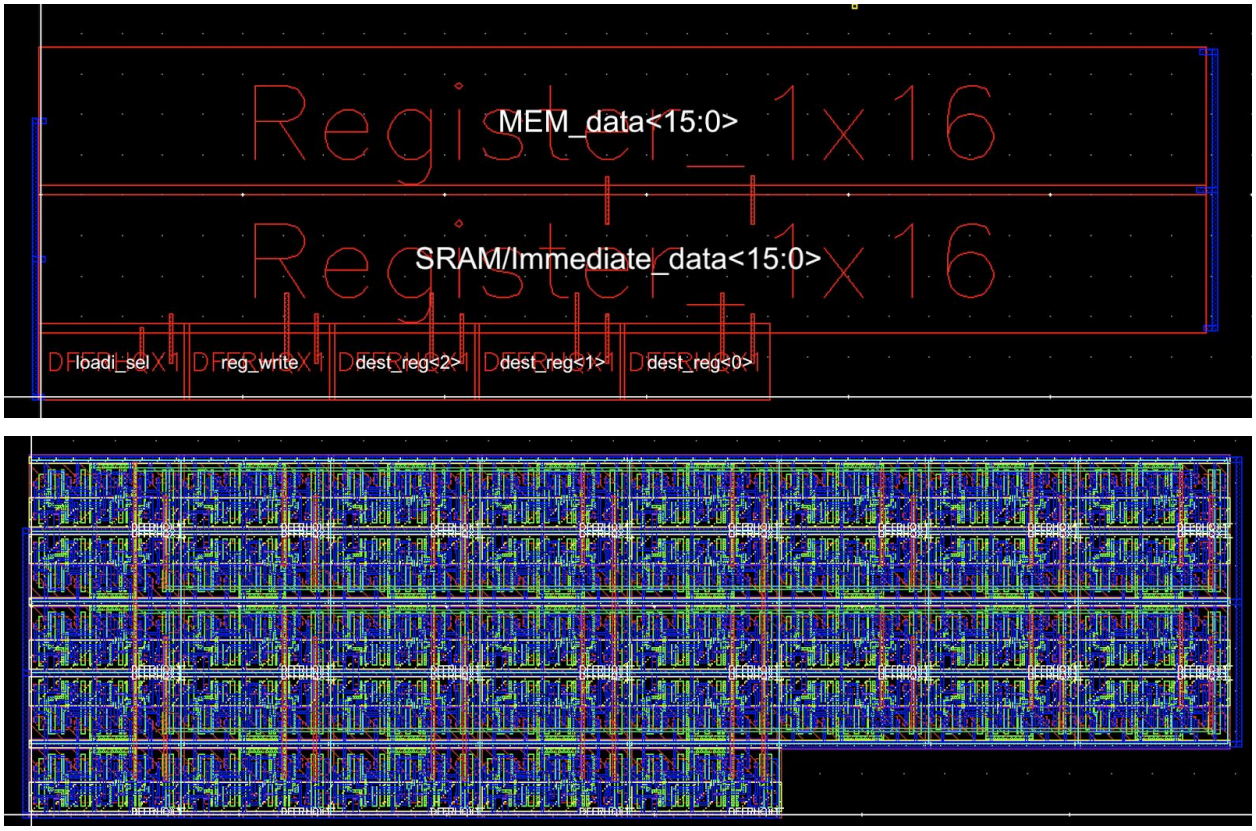
- **MEM-Stage (SRAM) Layout**
  - The SRAM design, including the schematic and the complete top-level layout, is adopted from the verified working design of Lab 2.

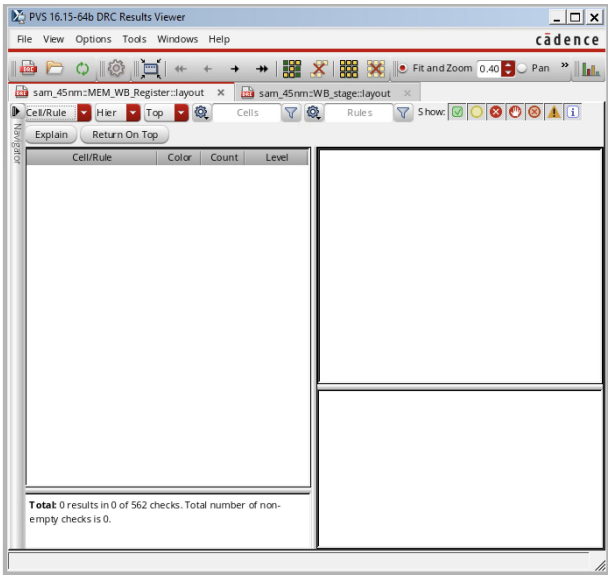


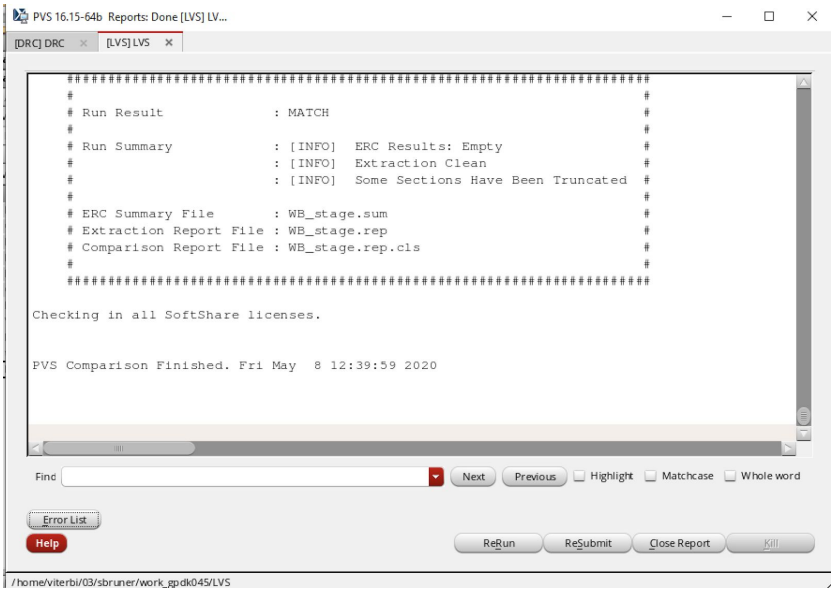

DENSITY: Cumulative Time CPU = 0(s) REAL = 0(s)
MISCELLANEOUS: Cumulative Time CPU = 0(s) REAL = 0(s)
CONNECT: Cumulative Time CPU = 0(s) REAL = 0(s)
DEVICE: Cumulative Time CPU = 0(s) REAL = 0(s)
ERC: Cumulative Time CPU = 0(s) REAL = 0(s)
PATTERN_MATCH: Cumulative Time CPU = 0(s) REAL = 0(s)
DFM FILL: Cumulative Time CPU = 0(s) REAL = 0(s)
Total CPU Time : 2(s)
Total Real Time : 3(s)
Peak Memory Used : 114(M)
Total Original Geometry : 3213(128991)
Total DRC RuleChecks : 562
Total DRC Results : 0 (0)
Summary can be found in file SRAM_full_layout.sum
ASCII report database is /home/viterbi/00/abdigazy/work_gpdk045/DRC/SRAM_full_layout
Checking in all SoftShare licenses.
Design Rule Check Finished Normally. Fri May 8 14:11:12 2020
PVS 16.15-64b DRC Results Viewer
cadence


ERC Results Empty
Extraction Results: Clean
Comparison Results: Match
Do you want to start the LVS DE?
LVS Summary
Total CPU Time : 1(s)
Total Real Time : 3(s)
Peak Memory Used : 235.64(M)
Run Result : MATCH
Run Summary : [INFO] ERC Results: Empty
: [INFO] Extraction Clean
ERC Summary File : SRAM_full_layout.sum
Extraction Report File : SRAM_full_layout.rep
Comparison Report File : SRAM_full_layout.rep.cls
Checking in all SoftShare licenses.
PVS Comparison Finished. Fri May 8 14:09:18 2020

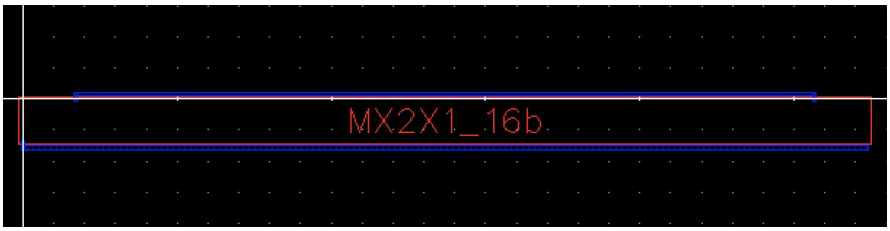
- **MEM/WB Pipeline Register Layout**

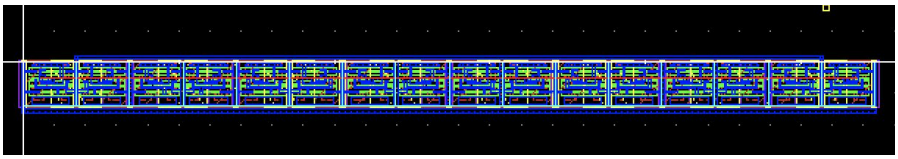


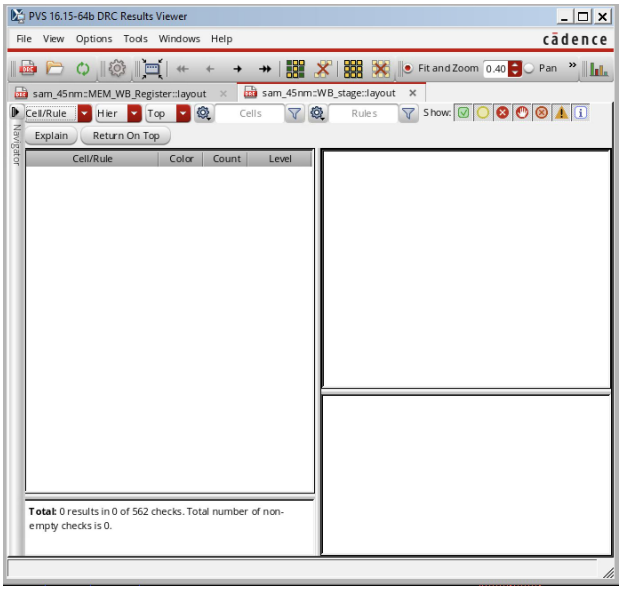


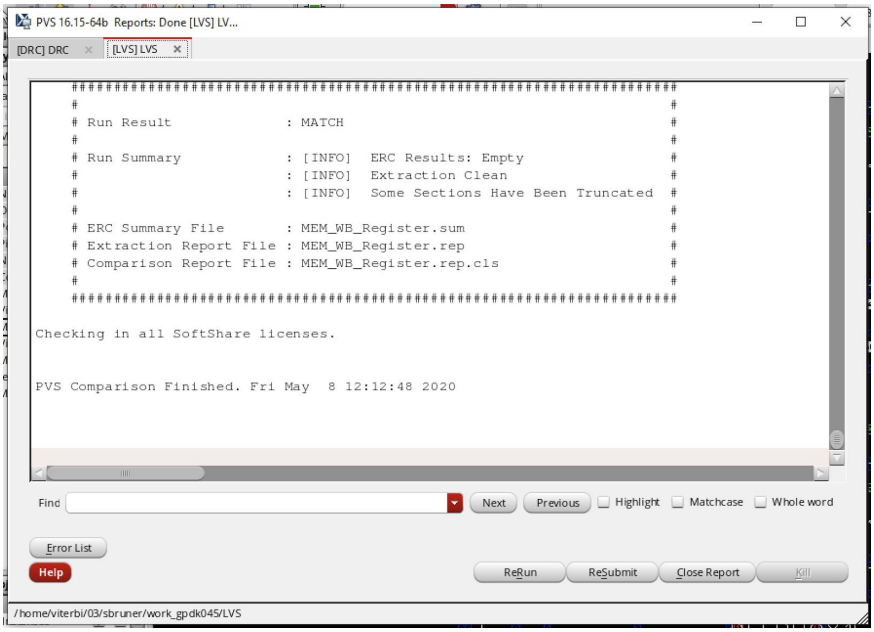


- **WB-Stage Layout**


MX2X1_16b

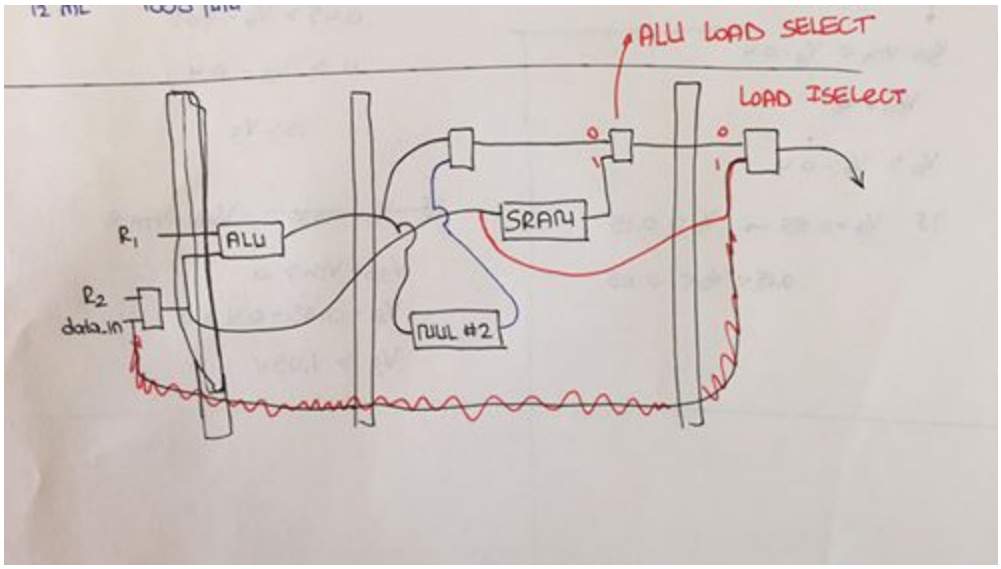



PVS 16.15-64b DRC Results Viewer
File View Options Tools Windows Help
cadence
Fit and Zoom 0.40 Pan
sam_45nm::MEM_WB_Register::layout
sam_45nm::WB_stage::layout
Cell/Rule Hier Top Cells Rules Show:
Explain Return On Top
Cell/Rule Color Count Level
Total: 0 results in 0 of 562 checks. Total number of non-empty checks is 0.


PVS 16.15-64b Reports: Done [LVS] LV...
[DRC] DRC
[LVS] LVS
Run Result : MATCH
Run Summary : [INFO] ERC Results: Empty
: [INFO] Extraction Clean
: [INFO] Some Sections Have Been Truncated
ERC Summary File : MEM_WB_Register.sum
Extraction Report File : MEM_WB_Register.rep
Comparison Report File : MEM_WB_Register.rep.cls
Checking in all SoftShare licenses.
PVS Comparison Finished. Fri May 8 12:12:48 2020
Find Next Previous Highlight Matchcase Whole word
Error List
Help
ReRun ReSubmit Close Report Kill
/home/viterbi/03/sbruner/work_gpdk045/LVS

- **Data Path Optimization**

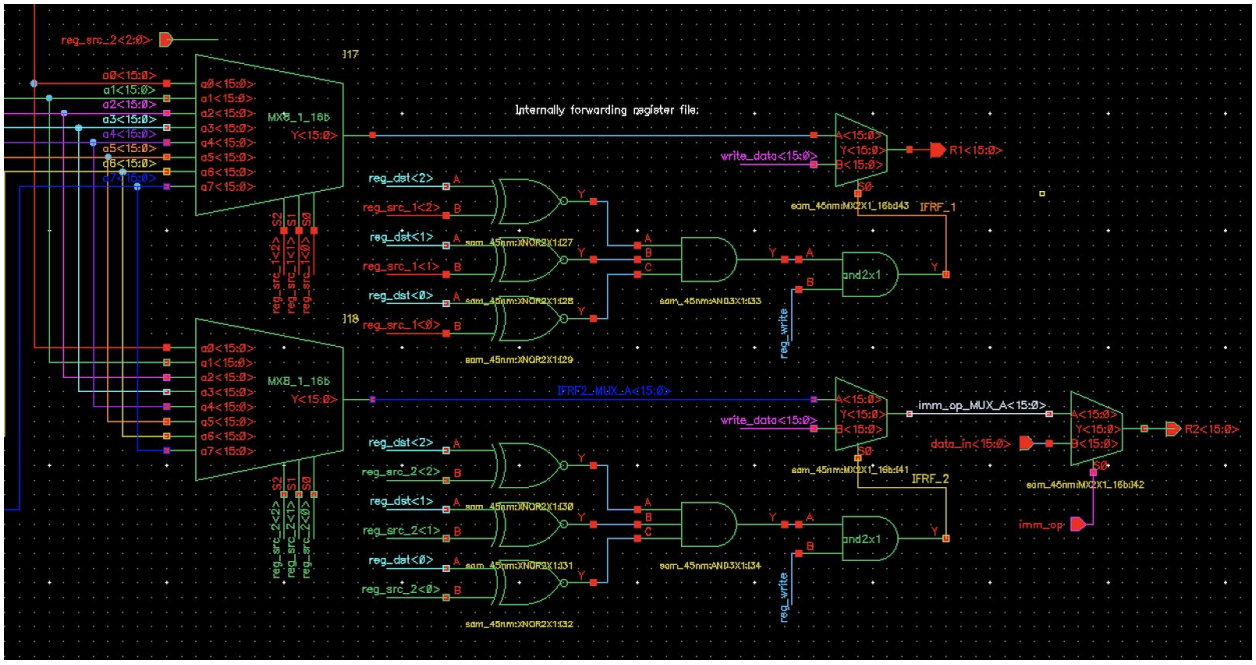
The following modification of the immediate value datapath was made in order to reduce area by 32 DFFs. The original design simply forwarded a dedicated 16b path from ID stage through WB stage. After the modification, a 16b 2 to 1 MUX is inserted at the end of the source register 2 datapath before outputting to the ID/EX stage register. The imm_op control signal is then able to select between the immediate value or source register 2 value. Since the immediate value is only needed at the input of the SRAM, we are able to share the R2 datapath through the ID/EX and EX/MEM stage registers. The immediate value only then is given a dedicated path as it passes through the MEM/WB stage register.



- **Register File Optimization**

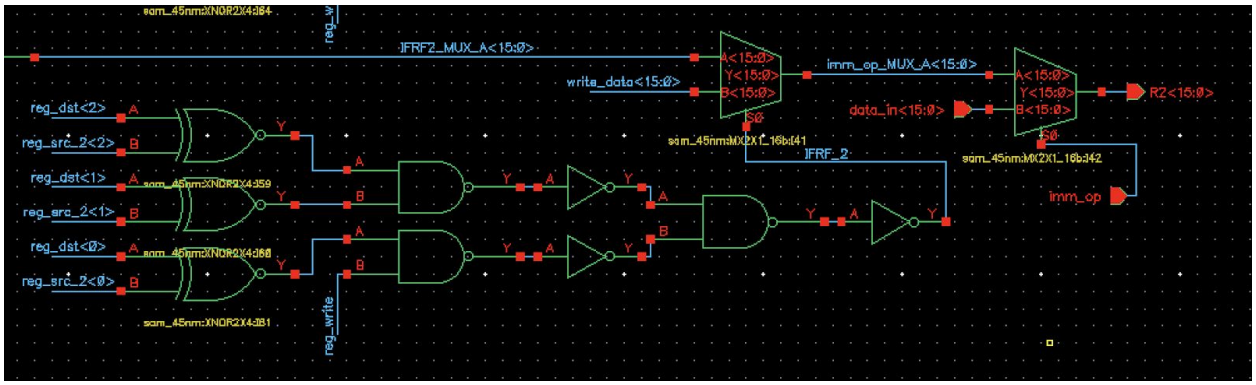
  - Phase 2 Optimizations

In the latter portion of the RF datapath, an internally forwarding register file (IFRF) was implemented in order to reduce the minimum number of NOPs following a RAW data hazard from 3 to 2. In the original datapath design, if an ID/EX RAW dependency occured, the senior instruction would need to completely pass through WB stage before the junior instruction is no longer stalled. After the IFRF implementation, senior instructions stalling junior instructions are able to forward their values from the WB stage to the RF in ID stage before exiting the WB stage. There is a saving of one NOP as a result.



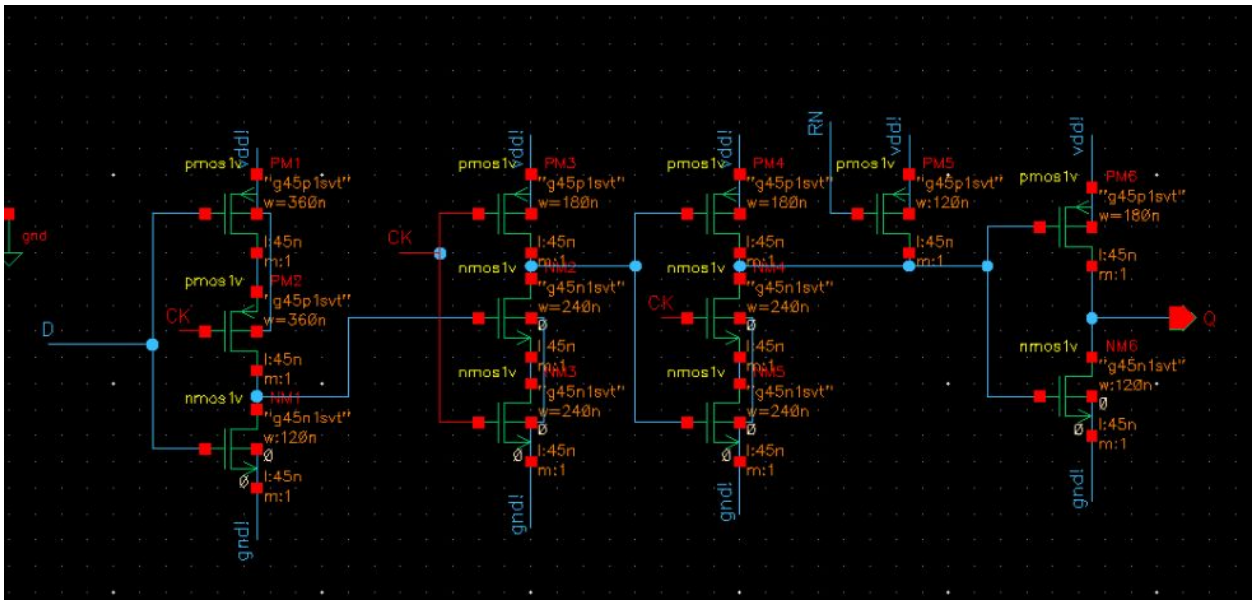
- Phase 3 Optimizations

When decreasing the clock period below 0.7ns, the register file final output originally yielded incorrect data for some registers. These incorrect values correspond to instructions with data dependencies. The root cause was attributed to delay in the IFRF signal propagation as the internally forwarding data could not reach EX stage before the clock edge. The internally forwarded value was reaching the IF/EX stage register 24ps after the clock edge. The following figure shows the implementation of logical effort on the IFRF signal datapath as well as 4X sizing of each gate to decrease propagation delay. After this optimization, the clock period was able to successfully achieve 0.59ns.



- **Power Optimization**

- True Single-Phase Clock (TSPC) Flip-Flop



Given the large number of flip flops required in the pipeline stage registers of the datapath and the control signal transfer path, different flip-flop architectures beyond the standard master-slave latch DFF construction were explored for further power and transistor area reduction without sacrificing performance. The true single-phase clock dynamic structure was ultimately adopted given its compact structure and size. Because there are no transmission gates, the FF uses a single clock signal without needing to invert and thus reduces the effect of clock skew. Compared to the standard master-slave DFF from the gpdk045 library with asynchronous reset, the transistor count is brought down from 28T to 12T per FF and cell area is reduced by approximately 50% per FF.

The most promising aspect of the TSPC FF is its reduced power consumption. With a significant reduction in transistor count, both static leakage power and dynamic switching power is reduced. By adopting TSPC FFs in the construction of the ID/EX and EX/MEM stage registers, a reduction of 150μW in total power from over 900 μW was achieved for the processor overall compared to using standard master-slave latch DFFs. An attempt to further reduce power consumption was made by replacing the DFFs of the register file with these TSPC FFs. However, possibly due to the dynamic logic nature of the TSPC construction, functionality issues were encountered after the replacement and thus the register file kept standard DFFs, while the stage registers (consisting of 94 FFs in total) adopted TSPC FFs.

- Static Voltage Scaling

The reference voltage VDD in AND/OR blocks was reduced to 700mV. This voltage reference was directly generated by the ideal external pin. Since the output of the ALU is followed by stage registers (DFFs), the output of the AND/OR blocks is still properly latched and the correct operation of the ALU is preserved. Even though the reduction of VDD brings more delay into the circuit, the delay from AND/OR blocks is still far from the bottleneck. This technique saved us 10uW of total power consumption at 1GHz of clock operation.

- **Final Python Script**

```python
# Front-end: Instruction Fetch and Decode
fileIn = open('sample_instructions_new.txt', 'r')
fileOutCPU = open('cpu_new.vec', 'w')
fileOutMEM = open('mem_new.vec', 'w')
fileOutCPU.write('radix 1 1 1 3 3 3 ')
fileOutCPU.write('1 1 1 1 1 1 1 ')
fileOutCPU.write('1 1 1 4444 14\n')
fileOutCPU.write('io i i i i i i i i i i i i i i i i i i\n')
fileOutCPU.write('vname clk ~reset reg_write reg_dst<[2:0]> reg_src_1<[2:0]> reg_src_2<[2:0]> ')
fileOutCPU.write('and_en or_en add_en mul_en min_en shift_en right_shift ')
fileOutCPU.write('imm_op alu_load_sel loadi_sel data_in<[15:0]> addr<[4:0]>\n')
fileOutCPU.write('tunit ns\nslope 0.005\nvih 1.0\nvil 0.0\n\n')
fileOutMEM.write('radix 1 1 1 1\n')
fileOutMEM.write('io i i i i\n')
fileOutMEM.write('vname ~pre_en mem_write_en mem_read_en rowdec_en\n')
fileOutMEM.write('tunit ns\nslope 0.005\nvih 1.0\nvil 0.0\n\n')
destination = ['none', 'none']
instructionList = []
commandNum = 0
for line in fileIn:
    line = line.replace('$', '')
    line = line.replace('#', '')
    line = line.upper()
    instruction = line.split(' ')
    if instruction == ['\n']:
        continue
    commandNum += 1
    instruction[-1] = instruction[-1].replace('\n', '')
    if instruction[0] == 'STOREI' or instruction[0] == 'STORE' or instruction[0] == 'LOAD':
        for i in range(1, 3):
            if len(instruction[i]) == 3 and instruction[i][-1] == 'H':
                instruction[i] = instruction[i].replace('H', '')
            elif len(instruction[i]) == 6 and instruction[i][-1] == 'B':
                instruction[i] = '%X' % int(instruction[i].replace('B', ''), 2)
                if len(instruction[i]) == 1:
                    instruction[i] = '0' + instruction[i]
    if instruction[0] == 'STOREI':
        if len(instruction[1]) == 1:
            addrLSB = instruction[2][-1]
            if instruction[1] == '1':
                instruction.pop(1)
                instructionList.append(instruction)
                destination.pop(0)
                destination.append('none')
            elif instruction[1] == '2':
                if (addrLSB != '0' and addrLSB != '2' and addrLSB != '4' and addrLSB != '6'
                    and addrLSB != '8' and addrLSB != 'A' and addrLSB != 'C' and addrLSB != 'E'):
                    print('Error001: Command ' + str(commandNum) + ' is not aligned properly.')
                    continue
                else:
                    instructionList.append(['STOREI', instruction[2], instruction[3]])
                    instructionList.append(['STOREI', instruction[2][0] + chr(ord(addrLSB) + 1), instruction[4]])
                    destination.pop(0)
                    destination.pop(0)
                    destination.append('none')
                    destination.append('none')
            elif instruction[1] == '4':
                if addrLSB != '0' and addrLSB != '4' and addrLSB != '8' and addrLSB != 'C':
                    print('Error001: Command ' + str(commandNum) + ' is not aligned properly.')
                    continue
                else:
                    instructionList.append(['STOREI', instruction[2], instruction[3]])
                    instructionList.append(['STOREI', instruction[2][0] + chr(ord(addrLSB) + 1), instruction[4]])
                    if addrLSB == '8':
                        instructionList.append(['STOREI', instruction[2][0] + 'A', instruction[5]])
                        instructionList.append(['STOREI', instruction[2][0] + 'B', instruction[6]])
```

```
                    else:
                        instructionList.append(['STOREI', instruction[2][0] + chr(ord(addrLSB) + 2), instruction[5]])
                        instructionList.append(['STOREI', instruction[2][0] + chr(ord(addrLSB) + 3), instruction[6]])
                    destination = ['none', 'none']
            else:
                print('Error000: Command ' + str(commandNum) + ' has invalid burst length.')
                continue
        else:
            instructionList.append(instruction)
            destination.pop(0)
            destination.append('none')
    else:
        if instruction[0] != 'LOADI' and instruction[0] != 'LOAD' and instruction[0] != 'NOP':
            while instruction[2] in destination or (instruction[0] != 'STORE' and instruction[3] in destination):
                instructionList.append(['NOP'])
                destination.pop(0)
                destination.append('none')
        instructionList.append(instruction)
        destination.pop(0)
        if instruction[0] != 'STORE' and instruction[0] != 'NOP':
            destination.append(instruction[1])
        else:
            destination.append('none')
clkPeriod = 0.6
clk = '1'
reset_bar = '0'
reg_write = '0'
reg_dst = '0'
reg_src_1 = '0'
reg_src_2 = '0'
and_en = '0'
or_en = '0'
add_en = '0'
mul_en = '0'
min_en = '0'
shift_en = '0'
right_shift = '0'
imm_op = '0'
alu_load_sel = '0'
loadi_sel = '0'
data_in = '0000'
addr = '00'
fileOutCPU.write('0    ')
fileOutCPU.write(clk + ' ' + reset_bar + ' ' + reg_write + ' ' + reg_dst + ' ' + reg_src_1 + ' ' + reg_src_2 + ' ')
fileOutCPU.write(and_en + ' ' + or_en + ' ' + add_en + ' ' + mul_en + ' ' + min_en + ' ' + shift_en + ' ' + right_shift + ' ')
fileOutCPU.write(imm_op + ' ' + alu_load_sel + ' ' + loadi_sel + ' ' + data_in + ' ' + addr + '\n')
fileOutMEM.write('0  1 0 0 0\n')
reset_bar = '1'
for i in range(len(instructionList)):
    op = instructionList[i]
    if op[0] == 'STOREI' or op[0] == 'STORE' or op[0] == 'NOP':
        reg_write = '0'
    else:
        reg_write = '1'
        reg_dst = op[1]
    if op[0] != 'STOREI' and op[0] != 'STORE' and op[0] != 'LOADI' and op[0] != 'LOAD' and op[0] != 'NOP':
        reg_src_1 = op[2]
    if op[0] == 'STORE':
        reg_src_2 = op[2]
    elif op[0] == 'AND' or op[0] == 'OR' or op[0] == 'ADD' or op[0] == 'MUL' or op[0] == 'MIN':
        reg_src_2 = op[3]
    if op[0] == 'ANDI' or op[0] == 'AND':
        and_en = '1'
    else:
        and_en = '0'
    if op[0] == 'ORI' or op[0] == 'OR':
```

```
133         or_en = '1'
134     else:
135         or_en = '0'
136     if op[0] == 'ADDI' or op[0] == 'MINI' or op[0] == 'ADD' or op[0] == 'MIN':
137         add_en = '1'
138     else:
139         add_en = '0'
140     if op[0] == 'MULI' or op[0] == 'MUL':
141         mul_en = '1'
142     else:
143         mul_en = '0'
144     if op[0] == 'MINI' or op[0] == 'MIN':
145         min_en = '1'
146     else:
147         min_en = '0'
148     if op[0] == 'SFL':
149         shift_en = '1'
150         right_shift = '0'
151     elif op[0] == 'SFR':
152         shift_en = '1'
153         right_shift = '1'
154     else:
155         shift_en = '0'
156     if op[0] == 'STORE' or op[0] == 'LOAD' or op[0] == 'AND' or op[0] == 'OR' or op[0] == 'ADD' or op[0] == 'MUL' or op[0] == 'MIN':
157         imm_op = '0'
158     else:
159         imm_op = '1'
160     if op[0] == 'LOAD':
161         alu_load_sel = '1'
162     else:
163         alu_load_sel = '0'
164     if op[0] == 'LOADI':
165         loadi_sel = '1'
166     else:
167         loadi_sel = '0'
168     if op[0] == 'STOREI' or op[0] == 'LOADI':
169         data_in = op[2]
170     elif op[0] == 'ANDI' or op[0] == 'ORI'or op[0] == 'ADDI' or op[0] == 'MINI' or op[0] == 'SFL' or op[0] == 'SFR':
171         data_in = op[3]
172     elif op[0] == 'MULI':
173         data_in = '00' + op[3]
174     if op[0] == 'STOREI' or op[0] == 'STORE':
175         addr = op[1]
176         fileOutMEM.write('\n; ')
177         for argument in op:
178             fileOutMEM.write(argument + ' ')
179         fileOutMEM.write('\n')
180         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod, 3)) + '  ')
181         fileOutMEM.write('0 0 0 0\n')
182         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod + 0.1, 3)) + '  ')
183         fileOutMEM.write('1 1 0 1\n')
184         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod + clkPeriod * 0.8, 3)) + '  ')
185         fileOutMEM.write('1 0 0 0\n')
186     elif op[0] == 'LOAD':
187         addr = op[2]
188         fileOutMEM.write('\n; ')
189         for argument in op:
190             fileOutMEM.write(argument + ' ')
191         fileOutMEM.write('\n')
192         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod, 3)) + '  ')
193         fileOutMEM.write('0 0 0 0\n')
194         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod + 0.1, 3)) + '  ')
195         fileOutMEM.write('1 0 0 1\n')
196         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod + clkPeriod * 0.6, 3)) + '  ')
197         fileOutMEM.write('1 0 1 0\n')
198         fileOutMEM.write(str(round(i * clkPeriod + 2 * clkPeriod + clkPeriod * 0.9, 3)) + '  ')
```

```
            fileOutMEM.write('1 0 0 0\n')
        fileOutCPU.write('\n; ')
        for argument in op:
            fileOutCPU.write(argument + ' ')
        fileOutCPU.write('\n')
        clk = '0'
        fileOutCPU.write(str(round(i * clkPeriod + clkPeriod / 2, 3)) + '  ')
        fileOutCPU.write(clk + ' ' + reset_bar + ' ' + reg_write + ' ' + reg_dst + ' ' + reg_src_1 + ' ' + reg_src_2 + ' ')
        fileOutCPU.write(and_en + ' ' + or_en + ' ' + add_en + ' ' + mul_en + ' ' + min_en + ' ' + shift_en + ' ' + right_shift + ' ')
        fileOutCPU.write(imm_op + ' ' + alu_load_sel + ' ' + loadi_sel + ' ' + data_in + ' ' + addr + '\n')
        clk = '1'
        fileOutCPU.write(str(round(i * clkPeriod + clkPeriod, 3)) + '    ')
        fileOutCPU.write(clk + ' ' + reset_bar + ' ' + reg_write + ' ' + reg_dst + ' ' + reg_src_1 + ' ' + reg_src_2 + ' ')
        fileOutCPU.write(and_en + ' ' + or_en + ' ' + add_en + ' ' + mul_en + ' ' + min_en + ' ' + shift_en + ' ' + right_shift + ' ')
        fileOutCPU.write(imm_op + ' ' + alu_load_sel + ' ' + loadi_sel + ' ' + data_in + ' ' + addr + '\n')
    fileOutCPU.write('\n; END OF INSTRUCTION SEQUENCE')
    for j in range(1, 6):
        clk = '0'
        fileOutCPU.write('\n' + str(round((i + j) * clkPeriod + clkPeriod / 2, 3)) + '  ')
        fileOutCPU.write(clk + ' 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00\n')
        clk = '1'
        fileOutCPU.write(str(round((i + j) * clkPeriod + clkPeriod, 3)) + '    ')
        fileOutCPU.write(clk + ' 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00\n')
    fileIn.close()
    fileOutCPU.close()
    fileOutMEM.close()

    # Back-end: Automated Result Verification
    def signed(value, bits):
        if value & (1 << (bits - 1)):
            value = value - (1 << bits)
        return value
    def unsigned(value, bits):
        if value < 0:
            value = value + (1 << bits)
        return value
    fileOutGolden = open('golden_results_new.txt', 'w')
    SRAM = {}
    RF = {}
    for instruction in instructionList:
        if instruction[0] == 'STOREI':
            SRAM[instruction[1]] = int(instruction[2], 16)
        elif instruction[0] == 'LOADI':
            RF[instruction[1]] = int(instruction[2], 16)
        elif instruction[0] == 'STORE':
            if instruction[2] in RF.keys():
                SRAM[instruction[1]] = RF[instruction[2]]
        elif instruction[0] == 'LOAD':
            if instruction[2] in SRAM.keys():
                RF[instruction[1]] = SRAM[instruction[2]]
        elif instruction[0] == 'AND':
            if instruction[2] in RF.keys() and instruction[3] in RF.keys():
                RF[instruction[1]] = RF[instruction[2]] & RF[instruction[3]]
        elif instruction[0] == 'ANDI':
            if instruction[2] in RF.keys():
                RF[instruction[1]] = RF[instruction[2]] & int(instruction[3], 16)
        elif instruction[0] == 'OR':
            if instruction[2] in RF.keys() and instruction[3] in RF.keys():
                RF[instruction[1]] = RF[instruction[2]] | RF[instruction[3]]
        elif instruction[0] == 'ORI':
            if instruction[2] in RF.keys():
                RF[instruction[1]] = RF[instruction[2]] | int(instruction[3], 16)
        elif instruction[0] == 'ADD':
            if instruction[2] in RF.keys() and instruction[3] in RF.keys():
                result = RF[instruction[2]] + RF[instruction[3]]
                result_hex = hex(result).replace('0x', '')
```

```
            if len(result_hex) > 4:
                result_hex = result_hex[1:]
            result = int(result_hex, 16)
            RF[instruction[1]] = result
        elif instruction[0] == 'ADDI':
            if instruction[2] in RF.keys():
                result = RF[instruction[2]] + int(instruction[3], 16)
                result_hex = hex(result).replace('0x', '')
                if len(result_hex) > 4:
                    result_hex = result_hex[1:]
                result = int(result_hex, 16)
                RF[instruction[1]] = result
        elif instruction[0] == 'MUL':
            if instruction[2] in RF.keys() and instruction[3] in RF.keys():
                op1 = signed(int(bin(RF[instruction[2]]).replace('0b', '')[-5:], 2), 5)
                op2 = signed(int(bin(RF[instruction[3]]).replace('0b', '')[-5:], 2), 5)
                result = op1 * op2
                RF[instruction[1]] = unsigned(result, 16)
        elif instruction[0] == 'MULI':
            if instruction[2] in RF.keys():
                op1 = signed(int(bin(RF[instruction[2]]).replace('0b', '')[-5:], 2), 5)
                op2 = signed(int(instruction[3], 16), 5)
                result = op1 * op2
                RF[instruction[1]] = unsigned(result, 16)
        elif instruction[0] == 'MIN':
            if instruction[2] in RF.keys() and instruction[3] in RF.keys():
                if signed(RF[instruction[2]], 16) < signed(RF[instruction[3]], 16):
                    RF[instruction[1]] = RF[instruction[2]]
                else:
                    RF[instruction[1]] = RF[instruction[3]]
        elif instruction[0] == 'MINI':
            if instruction[2] in RF.keys():
                if signed(RF[instruction[2]], 16) < signed(int(instruction[3], 16), 16):
                    RF[instruction[1]] = RF[instruction[2]]
                else:
                    RF[instruction[1]] = int(instruction[3], 16)
        elif instruction[0] == 'SFL':
            if instruction[2] in RF.keys():
                result = RF[instruction[2]] << int(instruction[3], 16)
                result_hex = hex(result).replace('0x', '')
                if len(result_hex) > 4:
                    result_hex = result_hex[-4:]
                    result = int(result_hex, 16)
                RF[instruction[1]] = result
        elif instruction[0] == 'SFR':
            if instruction[2] in RF.keys():
                RF[instruction[1]] = unsigned(signed(RF[instruction[2]], 16) >> int(instruction[3], 16), 16)
    fileOutGolden.write('Register File - Final Content:\n')
    for key in sorted(RF):
        fileOutGolden.write('$' + key + ': ')
        content_hex = hex(RF[key]).replace('0x', '')
        while len(content_hex) < 4:
            content_hex = '0' + content_hex
        fileOutGolden.write(content_hex.upper() + '\n')
    fileOutGolden.write('\nSRAM - Final Content:\n')
    for key in sorted(SRAM):
        fileOutGolden.write(key + 'H: ')
        content_hex = hex(SRAM[key]).replace('0x', '')
        while len(content_hex) < 4:
            content_hex = '0' + content_hex
        fileOutGolden.write(content_hex.upper() + '\n')
    fileOutGolden.close()
```

- **Vector File and Golden Results**
    - Note that the asynchronous control signals required for precharging, reading from and writing to the SRAM are produced directly by the front-end Python script through a separate vector file named “mem.vec” apart from the main processor input vector file “cpu.vec”.

```
radix 1 1 1 3 3 3 1 1 1 1 1 1 1 1 1 1 4444 14
io i i i i i i i i i i i i i i i i
vname clk ~reset reg_write reg_dst<[2:0]> reg_src_1<[2:0]> reg_src_2<[2:0]> and_en or_en add_en mul_en min_en shift_en right_shift imm_op alu_load_sel loadi_sel data_in<[15:0]> addr<[4:0]>
tunit ns
slope 0.005
vih 1.0
vil 0.0

0    1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00

; STOREI 0A FF0F
0.3  0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 FF0F 0A
0.6    1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 FF0F 0A

; STOREI 0B 0014
0.9  0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0014 0B
1.2    1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0014 0B

; STOREI 18 000B
1.5  0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 000B 18
1.8    1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 000B 18

; STOREI 19 00EE
2.1  0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 00EE 19
2.4    1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 00EE 19

; LOAD 7 0A
2.7  0 1 1 7 0 0 0 0 0 0 0 0 0 0 1 0 00EE 0A
3.0    1 1 1 7 0 0 0 0 0 0 0 0 0 0 1 0 00EE 0A

; LOAD 2 0B
3.3  0 1 1 2 0 0 0 0 0 0 0 0 0 0 1 0 00EE 0B
3.6    1 1 1 2 0 0 0 0 0 0 0 0 0 0 1 0 00EE 0B

; LOAD 3 19
3.9  0 1 1 3 0 0 0 0 0 0 0 0 0 0 1 0 00EE 19
4.2    1 1 1 3 0 0 0 0 0 0 0 0 0 0 1 0 00EE 19

; LOAD 4 18
4.5  0 1 1 4 0 0 0 0 0 0 0 0 0 0 1 0 00EE 18
4.8    1 1 1 4 0 0 0 0 0 0 0 0 0 0 1 0 00EE 18

; MUL 5 7 2
5.1  0 1 1 5 7 2 0 0 0 1 0 0 0 0 0 0 00EE 18
5.4    1 1 1 5 7 2 0 0 0 1 0 0 0 0 0 0 00EE 18

; NOP
5.7  0 1 0 5 7 2 0 0 0 0 0 0 0 0 1 0 0 00EE 18
6.0    1 1 0 5 7 2 0 0 0 0 0 0 0 0 1 0 0 00EE 18

; NOP
6.3  0 1 0 5 7 2 0 0 0 0 0 0 0 0 1 0 0 00EE 18
6.6    1 1 0 5 7 2 0 0 0 0 0 0 0 0 1 0 0 00EE 18

; MUL 6 3 5
6.9  0 1 1 6 3 5 0 0 0 1 0 0 0 0 0 0 00EE 18
7.2    1 1 1 6 3 5 0 0 0 1 0 0 0 0 0 0 00EE 18

; MIN 1 7 2
7.5  0 1 1 1 7 2 0 0 1 0 1 0 0 0 0 0 00EE 18
7.8    1 1 1 1 7 2 0 0 1 0 1 0 0 0 0 0 00EE 18

; STORE 1F 5
8.1  0 1 0 1 7 5 0 0 0 0 0 0 0 0 0 0 00EE 1F
8.4    1 1 0 1 7 5 0 0 0 0 0 0 0 0 0 0 00EE 1F

```

```
; STORE 00 6
8.7  0 1 0 1 7 6 0 0 0 0 0 0 0 0 0 0 00EE 00
9.0    1 1 0 1 7 6 0 0 0 0 0 0 0 0 0 0 00EE 00

; SFL 5 3 0005
9.3  0 1 1 5 3 6 0 0 0 0 0 1 0 1 0 0 0005 00
9.6    1 1 1 5 3 6 0 0 0 0 0 1 0 1 0 0 0005 00

; NOP
9.9  0 1 0 5 3 6 0 0 0 0 0 0 0 1 0 0 0005 00
10.2    1 1 0 5 3 6 0 0 0 0 0 0 0 1 0 0 0005 00

; NOP
10.5  0 1 0 5 3 6 0 0 0 0 0 0 0 1 0 0 0005 00
10.8    1 1 0 5 3 6 0 0 0 0 0 0 0 1 0 0 0005 00

; OR 6 5 4
11.1  0 1 1 6 5 4 0 1 0 0 0 0 0 0 0 0 0005 00
11.4    1 1 1 6 5 4 0 1 0 0 0 0 0 0 0 0 0005 00

; NOP
11.7  0 1 0 6 5 4 0 0 0 0 0 0 0 1 0 0 0005 00
12.0    1 1 0 6 5 4 0 0 0 0 0 0 0 1 0 0 0005 00

; NOP
12.3  0 1 0 6 5 4 0 0 0 0 0 0 0 1 0 0 0005 00
12.6    1 1 0 6 5 4 0 0 0 0 0 0 0 1 0 0 0005 00

; ANDI 6 6 00CC
12.9  0 1 1 6 6 4 1 0 0 0 0 0 0 1 0 0 00CC 00
13.2    1 1 1 6 6 4 1 0 0 0 0 0 0 1 0 0 00CC 00

; NOP
13.5  0 1 0 6 6 4 0 0 0 0 0 0 0 1 0 0 00CC 00
13.8    1 1 0 6 6 4 0 0 0 0 0 0 0 1 0 0 00CC 00

; NOP
14.1  0 1 0 6 6 4 0 0 0 0 0 0 0 1 0 0 00CC 00
14.4    1 1 0 6 6 4 0 0 0 0 0 0 0 1 0 0 00CC 00

; ADD 6 6 4
14.7  0 1 1 6 6 4 0 0 1 0 0 0 0 0 0 0 00CC 00
15.0    1 1 1 6 6 4 0 0 1 0 0 0 0 0 0 0 00CC 00

; STORE 09 5
15.3  0 1 0 6 6 5 0 0 0 0 0 0 0 0 0 0 00CC 09
15.6    1 1 0 6 6 5 0 0 0 0 0 0 0 0 0 0 00CC 09

; NOP
15.9  0 1 0 6 6 5 0 0 0 0 0 0 0 1 0 0 00CC 09
16.2    1 1 0 6 6 5 0 0 0 0 0 0 0 1 0 0 00CC 09

; STORE 10 6
16.5  0 1 0 6 6 6 0 0 0 0 0 0 0 0 0 0 00CC 10
16.8    1 1 0 6 6 6 0 0 0 0 0 0 0 0 0 0 00CC 10

; LOAD 1 00
17.1  0 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 00
17.4    1 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 00

; LOAD 1 1F
17.7  0 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 1F
18.0    1 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 1F

; LOAD 1 09
18.3  0 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 09
18.6    1 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 09

; LOAD 1 10
18.9  0 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 10
19.2    1 1 1 1 6 6 0 0 0 0 0 0 0 0 1 0 00CC 10

; END OF INSTRUCTION SEQUENCE
19.5  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00
19.8    1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00

20.1  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00
20.4    1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00

20.7  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00
21.0    1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00

21.3  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00
21.6    1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00

21.9  0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00
22.2    1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0000 00
```

```
radix 1 1 1 1
io i i i i
vname ~pre_en mem_write_en mem_read_en rowdec_en
tunit ns
slope 0.005
vih 1.0
vil 0.0

0  1 0 0 0

; STOREI 0A FF0F
1.2  0 0 0 0
1.3  1 1 0 1
1.68  1 0 0 0

; STOREI 0B 0014
1.8  0 0 0 0
1.9  1 1 0 1
2.28  1 0 0 0

; STOREI 18 000B
2.4  0 0 0 0
2.5  1 1 0 1
2.88  1 0 0 0

; STOREI 19 00EE
3.0  0 0 0 0
3.1  1 1 0 1
3.48  1 0 0 0

; LOAD 7 0A
3.6  0 0 0 0
3.7  1 0 0 1
3.96  1 0 1 0
4.14  1 0 0 0

; LOAD 2 0B
4.2  0 0 0 0
4.3  1 0 0 1
4.56  1 0 1 0
4.74  1 0 0 0

; LOAD 3 19
4.8  0 0 0 0
4.9  1 0 0 1
5.16  1 0 1 0
5.34  1 0 0 0

; LOAD 4 18
5.4  0 0 0 0
5.5  1 0 0 1
5.76  1 0 1 0
5.94  1 0 0 0

; STORE 1F 5
9.0  0 0 0 0
9.1  1 1 0 1
9.48  1 0 0 0

; STORE 00 6
9.6  0 0 0 0
9.7  1 1 0 1
10.08  1 0 0 0
```

```
; STORE 09 5
16.2  0 0 0 0
16.3  1 1 0 1
16.68  1 0 0 0

; STORE 10 6
17.4  0 0 0 0
17.5  1 1 0 1
17.88  1 0 0 0

; LOAD 1 00
18.0  0 0 0 0
18.1  1 0 0 1
18.36  1 0 1 0
18.54  1 0 0 0

; LOAD 1 1F
18.6  0 0 0 0
18.7  1 0 0 1
18.96  1 0 1 0
19.14  1 0 0 0

; LOAD 1 09
19.2  0 0 0 0
19.3  1 0 0 1
19.56  1 0 1 0
19.74  1 0 0 0

; LOAD 1 10
19.8  0 0 0 0
19.9  1 0 0 1
20.16  1 0 1 0
20.34  1 0 0 0
```

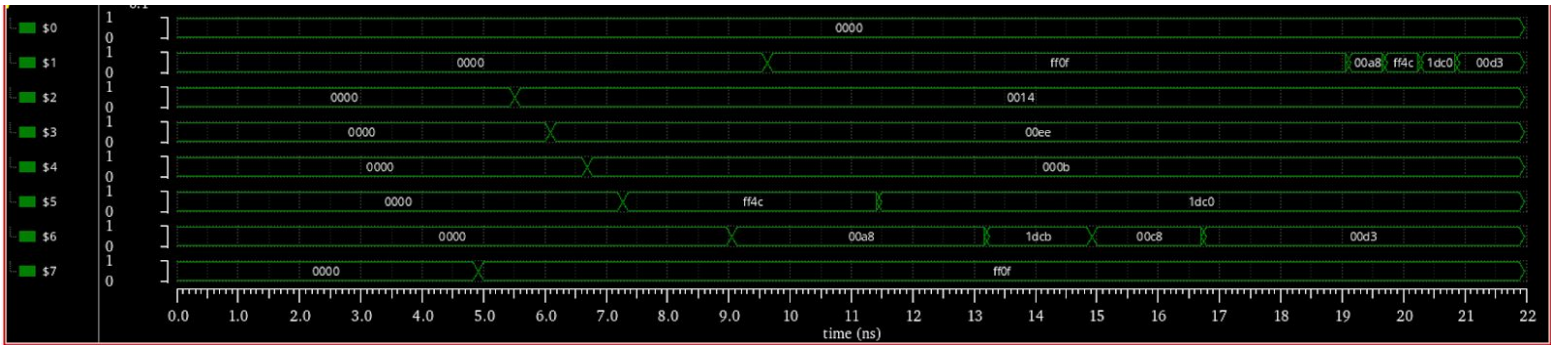
- Demo Instructions and Golden Results:

```
STOREI 0AH #ff0f
STOREI 0BH #0014
STOREI 2 18H #000B #00EE
LOAD $7 0AH
LOAD $2 0BH
LOAD $3 19H
LOAD $4 18H
MUL $5 $7 $2
MUL $6 $3 $5
MIN $1 $7 $2
STORE 1FH $5
STORE 00H $6
SFL $5 $3 #0005
OR $6 $5 $4
ANDI $6 $6 #00CC
ADD $6 $6 $4
STORE 09H $5
STORE 10H $6
LOAD $1 00H
LOAD $1 1FH
LOAD $1 09H
LOAD $1 10H
```

```
Register File - Final Content:
$1: 00D3
$2: 0014
$3: 00EE
$4: 000B
$5: 1DC0
$6: 00D3
$7: FF0F

SRAM - Final Content:
00H: 00A8
09H: 1DC0
0AH: FF0F
0BH: 0014
10H: 00D3
18H: 000B
19H: 00EE
1FH: FF4C
```

- **Simulation Results**
  - The following waveforms show the register file contents from $0 to $7 throughout the execution of the demo instruction sequence.



- **Performance Summary**

Clock Period [ns]	Total Time [ns]	Power [µW]	Estimated Area [µm^2]
0.59	**20.85**	**794**	**5632 (60.53µm x 93.04µm)**