CMPE640 CUSTOM VLSI DESIGN

PROJECT

ON

CACHE DESIGN

Submitted by

VEMULAPLLI SRI SAMADARSINI

(JV29859)

jv29859@umbc.edu

FALL 2023

INDEX

LIST OF FIGURES

INTRODUCTION

VHDL stands for VHSIC Hardware Description Language which is used to describe and simulate the behaviour of digital circuits. In VHDL, design code describes the behaviour of the circuit and test bench code is to test its behaviour. After creating both design file and the test bench file, we need to compile and simulate the circuit. There come 3 steps named Compiling, Elaborating, and Simulating. Also, the tools allow you to view the waveforms for signals at any level of hierarchy. This can be done by running VHDL code in GUI mode (SimVision).

In this project, we designed and simulated the cache block in VHDL. The cache block designed here is a direct mapped cache which is byte addressable data. The cache has 4 blocks, with four bytes per block. Therefore, you need 2 bits to select one of the 4 blocks and 2 bits to select the correct byte from the block. The remaining 2 bits in the address are used as a tag.

1.1 CACHE

The CPU address is a 6-bit address will be provided by the CPU on the rising edge of clock along with the start signal. The CPU data is 8-bit input/output data bus. Used as input for write requests and output for read requests. Data will be provided on the rising edge of clock for write requests along with the start signal. The start signal indicates a read/write request from the CPU, goes high on a positive edge. Busy signal indicates the CPU that your chip is processing the previous request. Read_Write (RD/WR') is high CPU is requesting a read operation, if low a write operation. A high on reset should invalidate all the entries in the cache and reset your state machine to its reset state. Clock is input to the chip. The CPU inputs/outputs and the memory inputs/outputs will be synchronized with this clock signal also. Memory Address (MA) is addressing the memory when there is a read miss. You should always specify the address for the final word in the block, with the final two bits of the address being 00. The memory controller will give you the data for the entire block (4 bytes) and automatically increment the address four times. Memory Data (MD)- Data taken from the memory. After receiving the enable signal, the memory needs eight clock cycles to supply the first byte of data. It will progressively supply the four bytes of information needed for the entire block. After asserting enable, the first data byte becomes valid on the eighth negative edge and remains steady for two clock cycles. After asserting enable, the subsequent byte will be supplied on the tenth negative edge and remain stable for two clock cycles. The final two data bytes will be delivered on the 12th and 14th negative edges, with timings that are comparable. Enable signal is an output that instructs the memory to begin a read operation at the address specified on the Memory Address. After asserting this signal, the memory returns four bytes as previously indicated, beginning at the eighth clock cycle.

The cache performs the four types of operations. They are Read Hit, Write Hit, Write Miss, Read Miss. For all four operations the CPU turns start high, provides the address and turns the read_write signal on the positive edge of clock. On the negative edge all signals are latched and turn on the busy signal.

1.1.1 Read Hit

The CPU performs read operations by starting high, providing the address, and turning the read_write signal high on the positive edge of the clock. On the negative edge, it latches all required signals and turns on the busy signal. The CPU removes the inputs on the positive edge

once it receives a busy signal. The data being referenced is checked by comparing the tag bits of the address with the tag bits stored in the block. The block is also checked for validity. The correct byte is read from the cache. If a tag and valid checking operation signals a hit, output enable goes high on the next negative edge, data is latched in the output register, and the busy signal is turned off. The CPU reads the data off the data bus on the positive edge.

1.1.2 Write Hit

The CPU signals are timed in accordance with the read, but the data to be written is supplied and read_write is set to low. It is necessary to latch the necessary inputs on the negative edge, enable busy, and execute tag/valid compare. If the outcome is a hit, the data should be written to the appropriate byte in the chosen block on the subsequent negative edge, and busy should remain high. Following the receipt of the write request, the second negative edge should go low, indicating that the write process has finished. On every successive clock cycle, the CPU will supply a fresh input.

1.1.3 Write Miss

A write miss does not require a block of memory when a write through is performed without write allocation. Although the actions and timing are identical to the write hit scenario, there is no need to run a cache operation. After two clocks, the busy should become low, enabling the CPU to process another request.

1.1.4 Read Miss

The CPU provides signals in the read hit procedure, maintaining high activity on the second negative edge of the clock. When a block is not in the cache, the CPU activates the enable signal to the memory, asking for a block read and providing the byte address. The memory latches the address when enable is high on a positive edge. After one clock cycle, enable should be switched low. The memory provides data for the final byte on the negative edge after 8 clock cycles, with stability lasting for two clock cycles. After setting up the proper write address, the cache is updated with the new data byte and the valid bit is set to on and updated during the first write operation. The output register is read, latches the right byte requested by the CPU, and output is enabled on a negative edge.

COMPONENTS USED IN THE DESIGN

The cache design is implemented by dividing the whole design into individual blocks. They are single bit Cache cell, single byte Cache, Cache block, Cache, Hit Miss Checker, Address register, Data register, State machine, and finally chip. The state machine here is designed to perform the cache operations of Read miss, Read hit, Write miss and Write Hit. To implement these above blocks, the basic components used are 2-input AND gate, 3-input AND gate, 2-input XOR gate, Inverter, D flipflop, D latch, 2X4 Decoder, Transmission gate, Multiplexers, 2-input NAND, 2-input NOR logics are designed.

2.1 2-input AND gate

The 2-input AND gate is implemented in VHDL as shown in file “and2ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.1. The inputs for 2-input AND gate are a, b and the output is c.

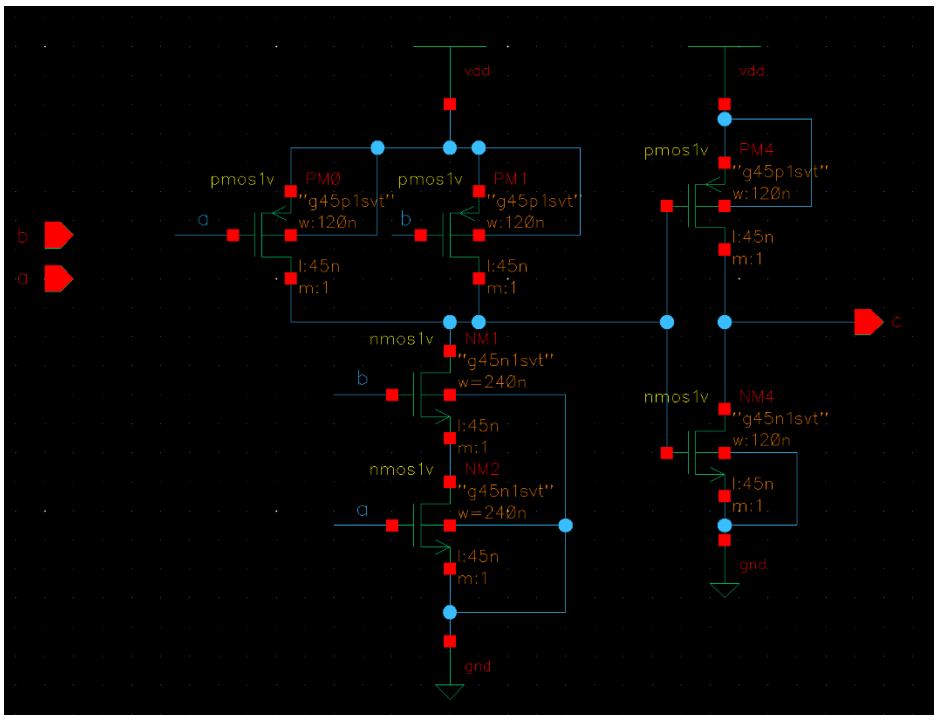

Fig 2.1: Schematic of 2-input AND gate

For the above schematic the layout is implemented as shown in below Fig 2.2. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

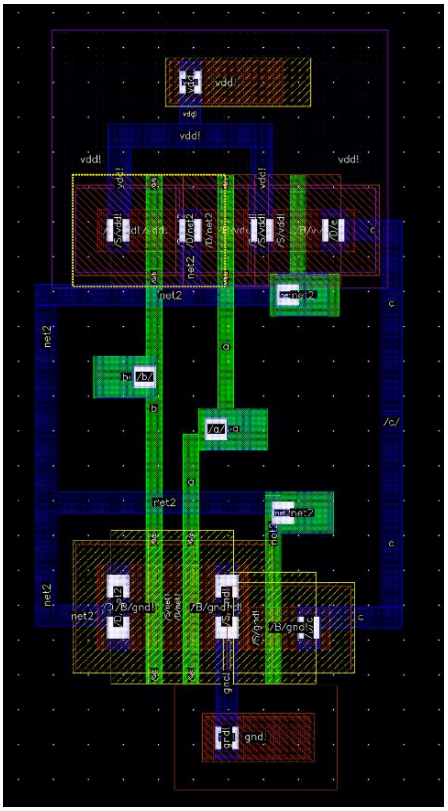

Fig 2.2: Layout of 2-input AND gate

2.2 3-input AND gate

The 3-input AND gate is implemented in VHDL as shown in file “and3ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.3. The inputs for 3-input AND gate are a, b, c and the output is d.

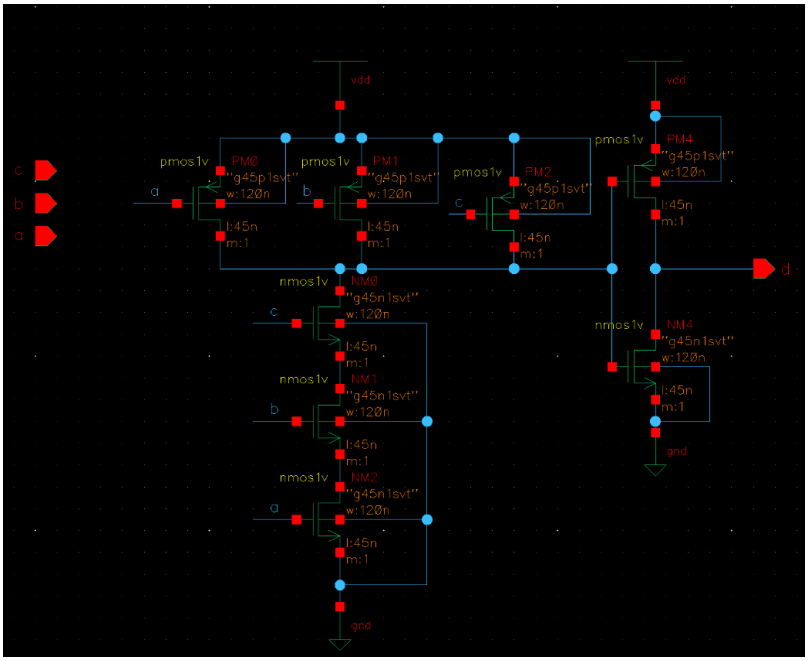

Fig 2.3: Schematic of 3-input AND gate

For the above schematic the layout is implemented as shown in below Fig 2.4. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

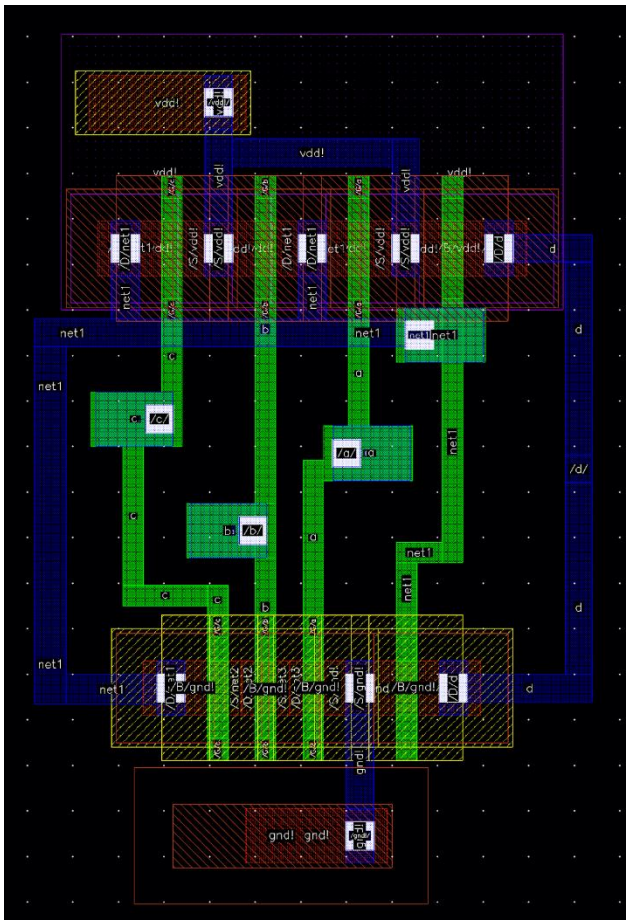

Fig 2.4: Layout of 3-input AND gate

2.3 2- input OR gate

The 2-input OR gate is implemented in VHDL as shown in file “or2ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.5. The inputs for 2-input OR gate are a, b and the output is c.

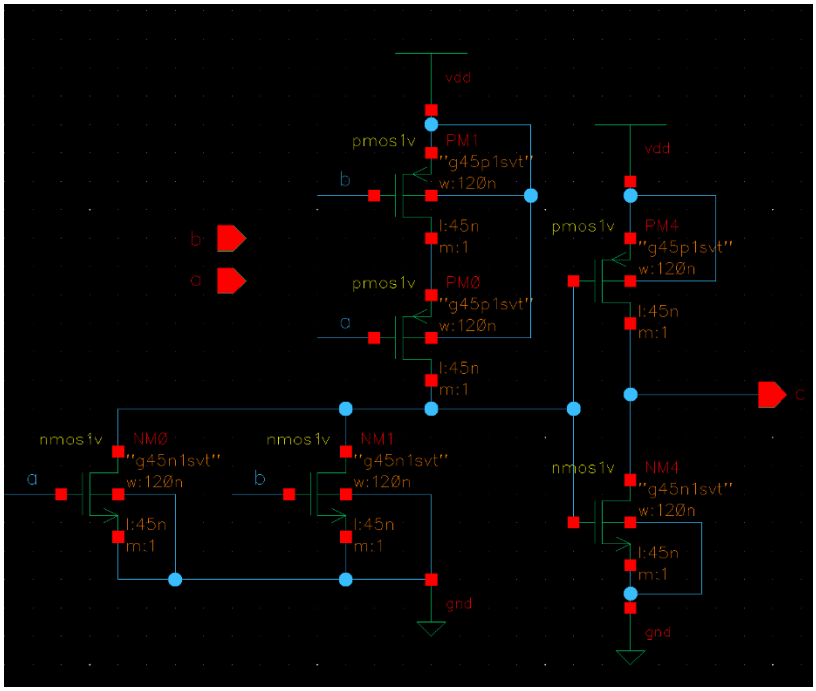

Fig 2.5: Schematic of 2-input OR gate

For the above schematic the layout is implemented as shown in below Fig 2.6. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

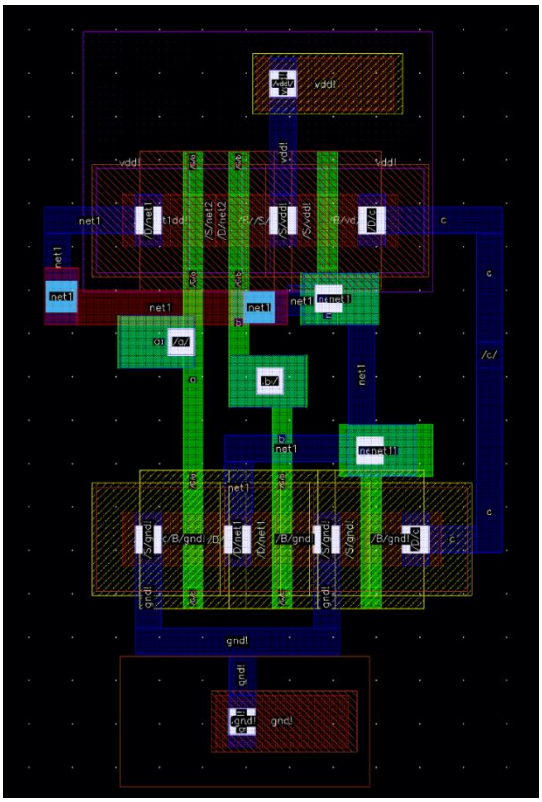

Fig 2.6: Layout of 2-input OR gate

2.4 3- input OR gate

The 3-input OR gate is implemented in VHDL as shown in file “or3ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.7. The inputs for 3-input OR gate are a, b, c and the output is d.

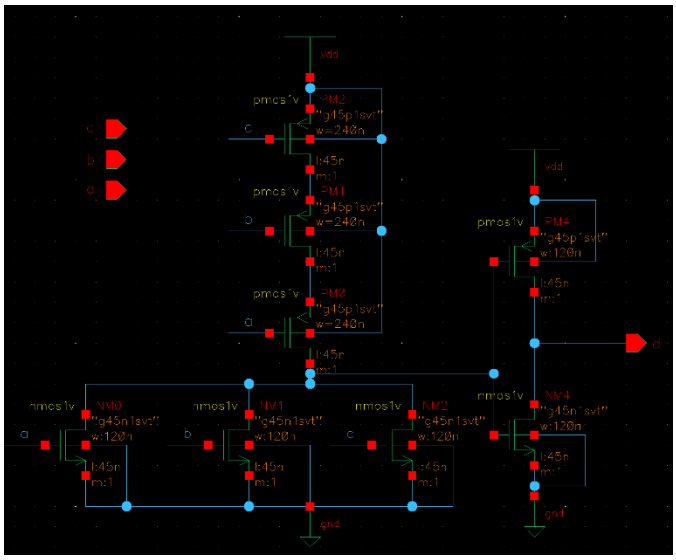

Fig 2.7 Schematic of 3-input OR gate

For the above schematic the layout is implemented as shown in below Fig 2.8. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

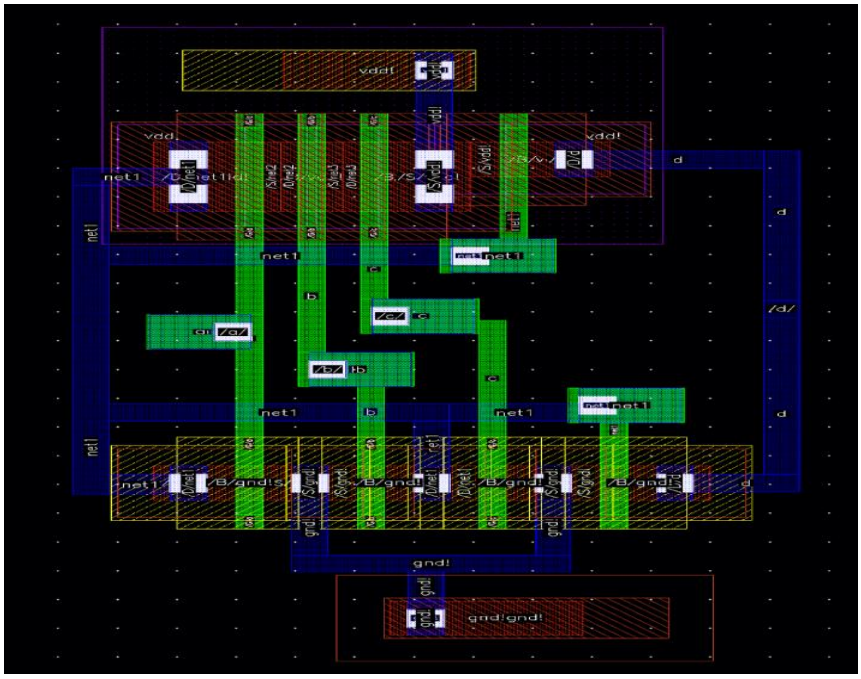

Fig 2.8: Layout of 3-input OR gate

2.5 4- input OR gate

The 4-input OR gate is implemented in VHDL as shown in file “or4ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.9. The inputs for 4-input OR gate are a, b, c, d and the output is e.

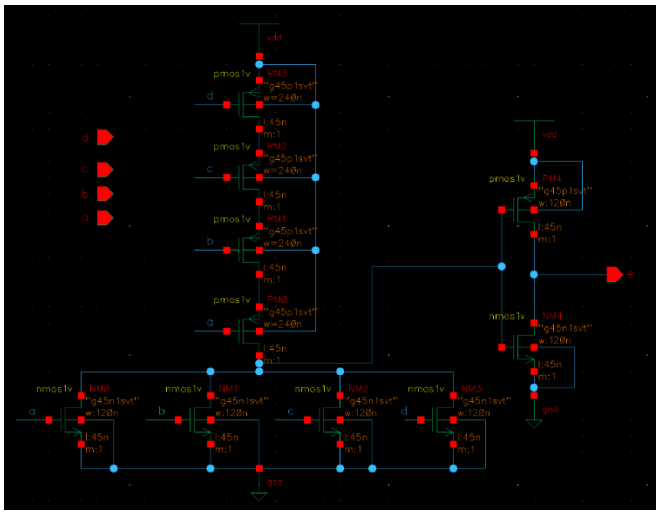

Fig 2.9: Schematic of 4-input OR gate

For the above schematic the layout is implemented as shown in below Fig 2.10. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

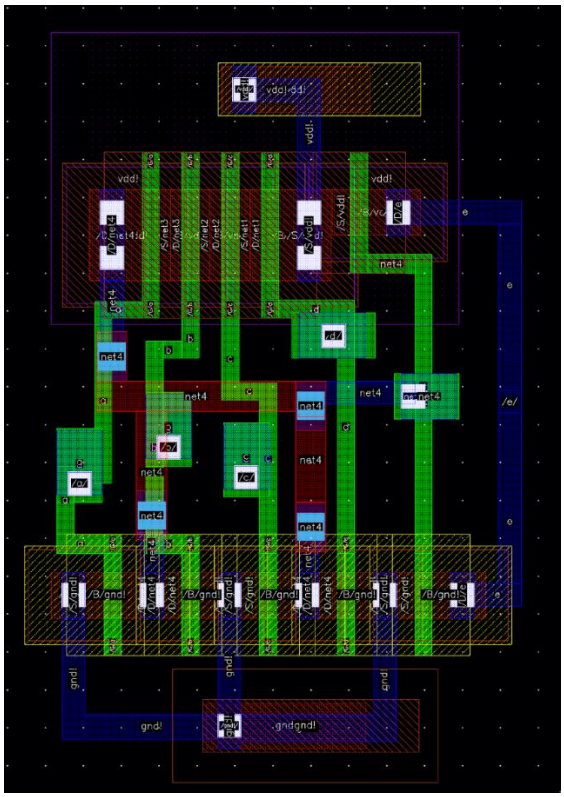

Fig 2.10: Layout of 4-input OR gate

2.6 2- input XOR gate

The 2-input XOR gate is implemented in VHDL as shown in file “xor4ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.11. The inputs for 2-input XOR gate are a, b and the output is c.

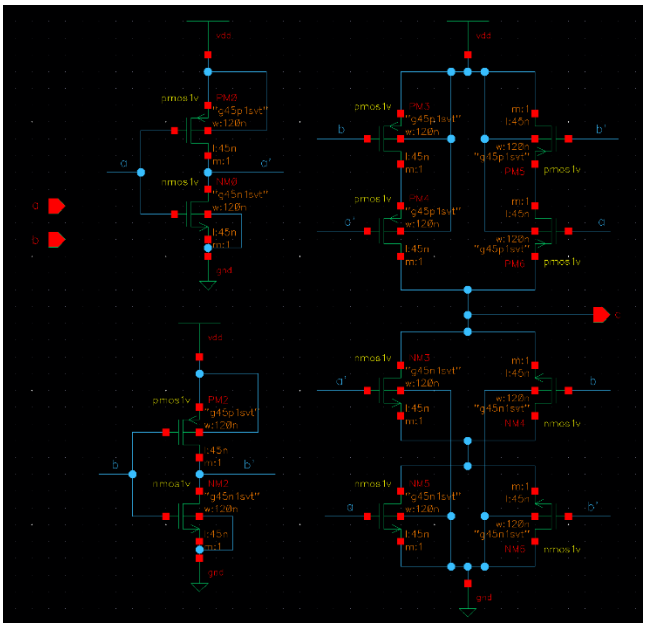

Fig 2.11: Schematic of 2-input XOR gate

For the above schematic the layout is implemented as shown in below Fig 2.12. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

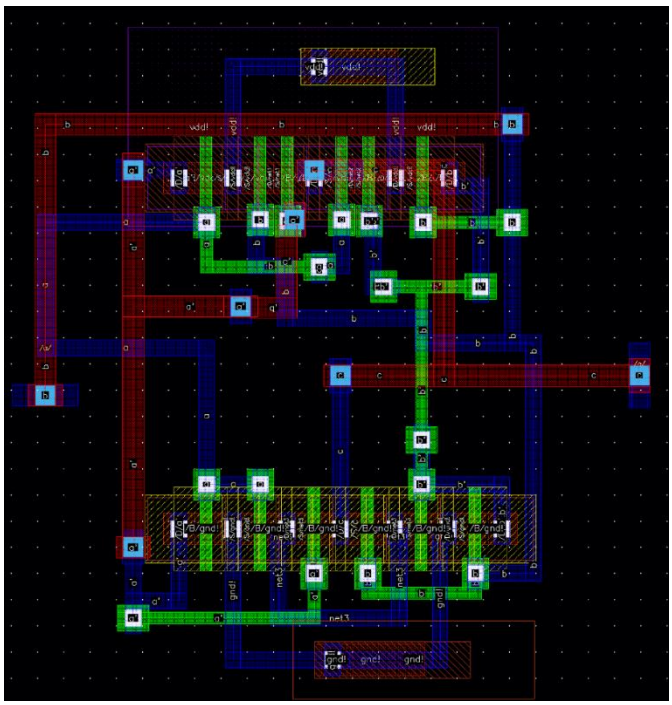

Fig 2.12: Layout of 2-input XOR gate

2.7 Inverter

The Inverter logic is implemented in VHDL as shown in file “inverter.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.13. The inputs for inverter is a and the output is b.

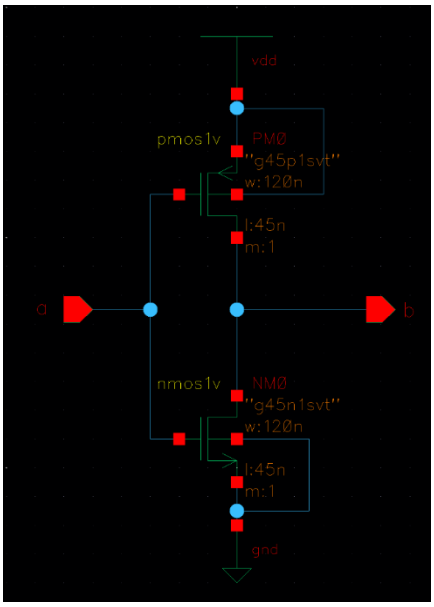

Fig 2.13: Schematic of Inverter

For the above schematic the layout is implemented as shown in below Fig 2.14. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

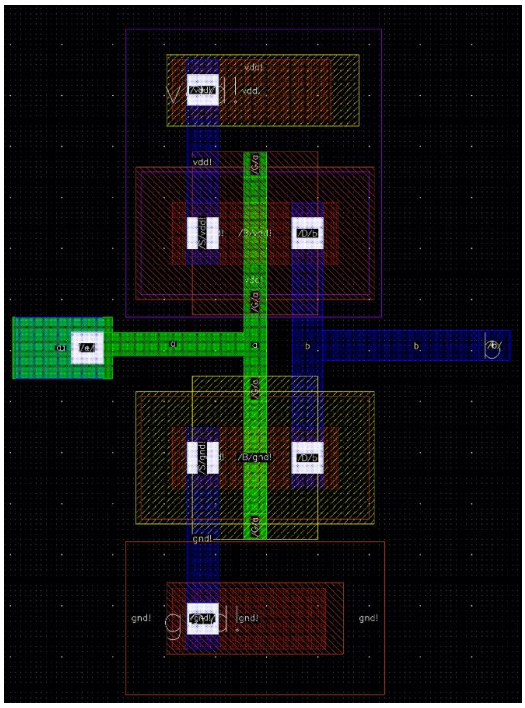

Fig 2.14: Layout of Inverter

2.8 2- input NAND gate

The 2-input NAND gate is implemented in VHDL as shown in file “nand2ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.15. The inputs for 2-input NAND gate are a, b and the output is c.

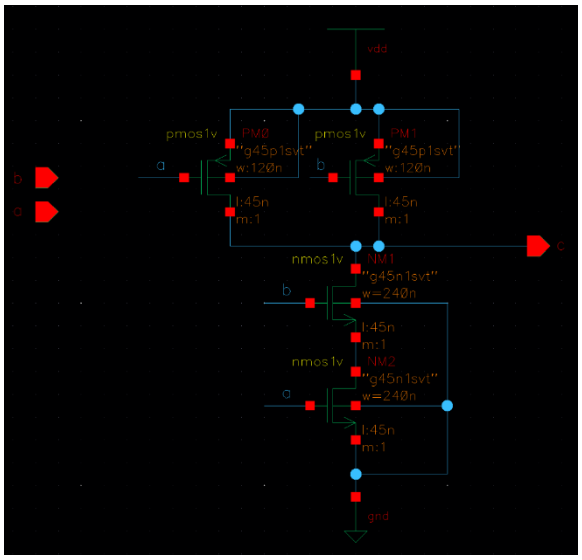

Fig 2.15: Schematic of 2-input NAND gate

For the above schematic the layout is implemented as shown in below Fig 2.16. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

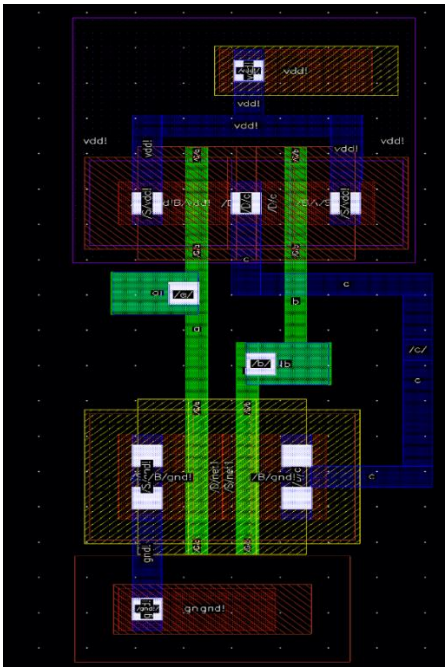

Fig 2.16: Layout of 2-input NAND gate

2.9 3-input NOR gate

The 3-input NOR gate is implemented in VHDL as shown in file “nor3ip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is designed for the respective VHDL functionality as shown in Fig 2.17. The inputs for 3-input NOR gate are a, b, c and the output is d.

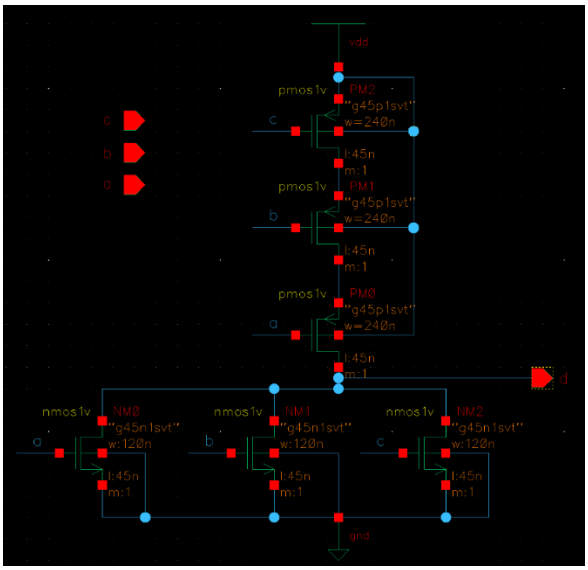

Fig 2.17: Schematic of 3-input NOR gate

For the above schematic the layout is implemented as shown in below Fig 2.18. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

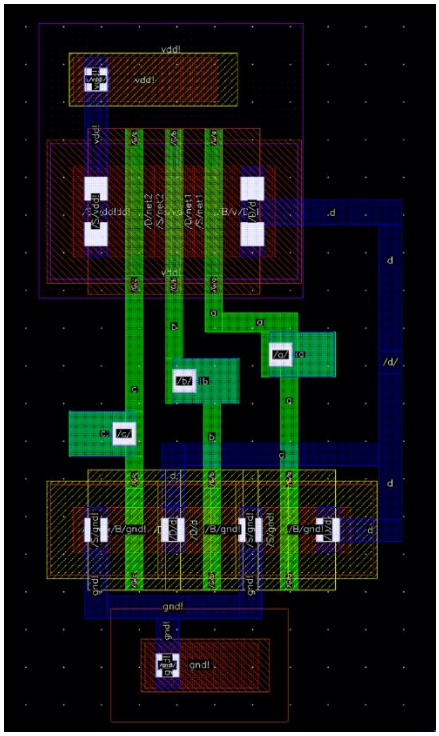

Fig 2.18: Layout of 3-input NOR gate

2.10 Decoder2x4

The 2x4 Decoder is implemented in VHDL as shown in file “decoder2x4.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.19. The inputs for decoder2x4 are 2 bit a, 2 bit abar, chip enable and the output is 4bit output.

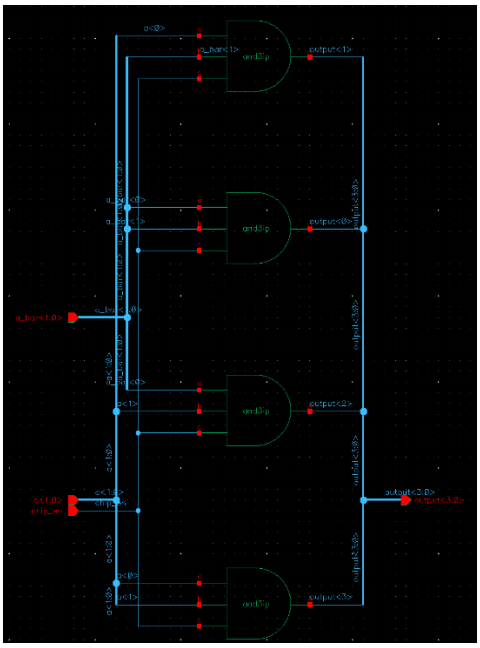

Fig 2.19: Schematic of decoder 2x4

For the above schematic the layout is implemented as shown in below Fig 2.20. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

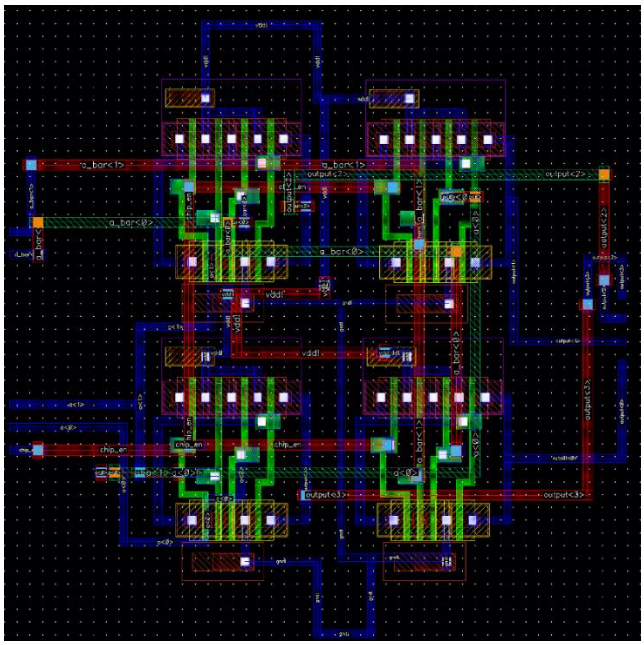

Fig 2.20: Layout of 2x4 decoder

2.11 D flip-flop

The D flip flop is implemented in VHDL as shown in file “dff.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.21. The inputs for D flip flop are data signal and clock and the output is q and qbar.

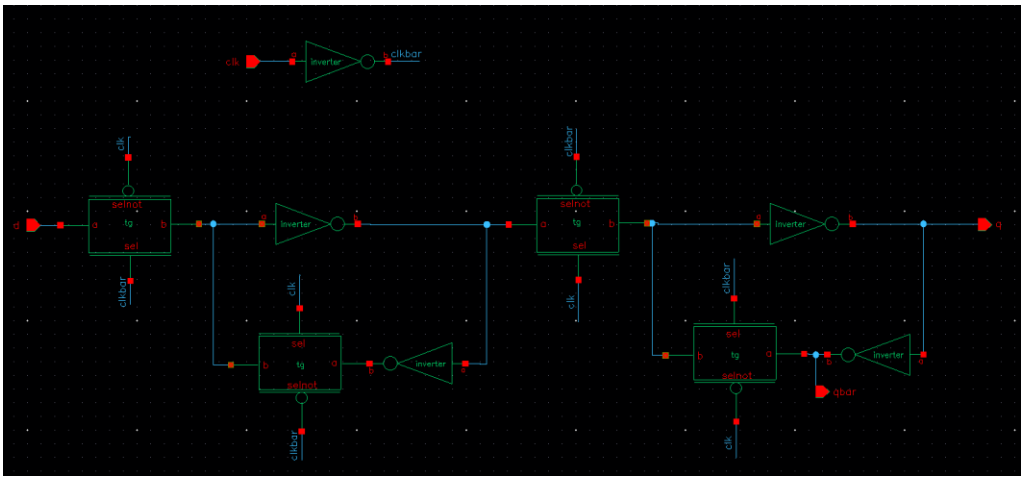

Fig 2.21: Schematic of D flipflop

For the above schematic the layout is implemented as shown in below Fig 2.21. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

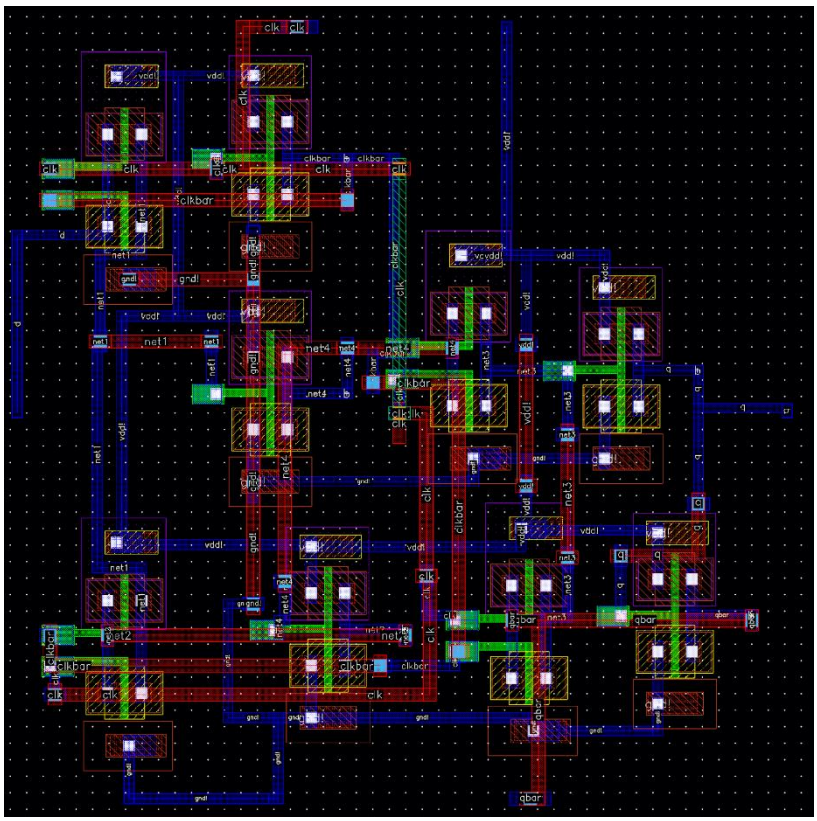

Fig 2.22: Layout of D flipflop

2.12 D latch

The D latch is implemented in VHDL as shown below. The input is d and the clock signal is given as clk. The outputs are q and qbar. In this project, we used Dlatch as positive edge Dlatch. The input goes to output whenever the clock provided is high. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.23.

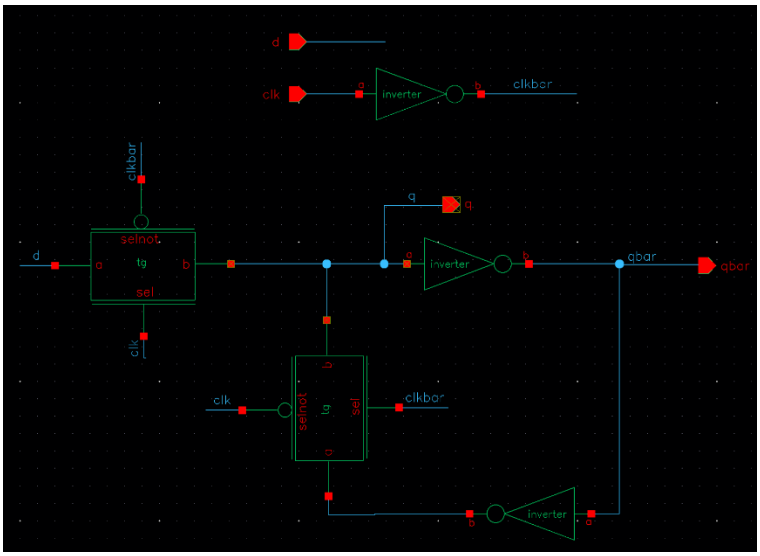

Fig 2.23: Schematic of D latch

For the above schematic the layout is implemented as shown in below Fig 2.24. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

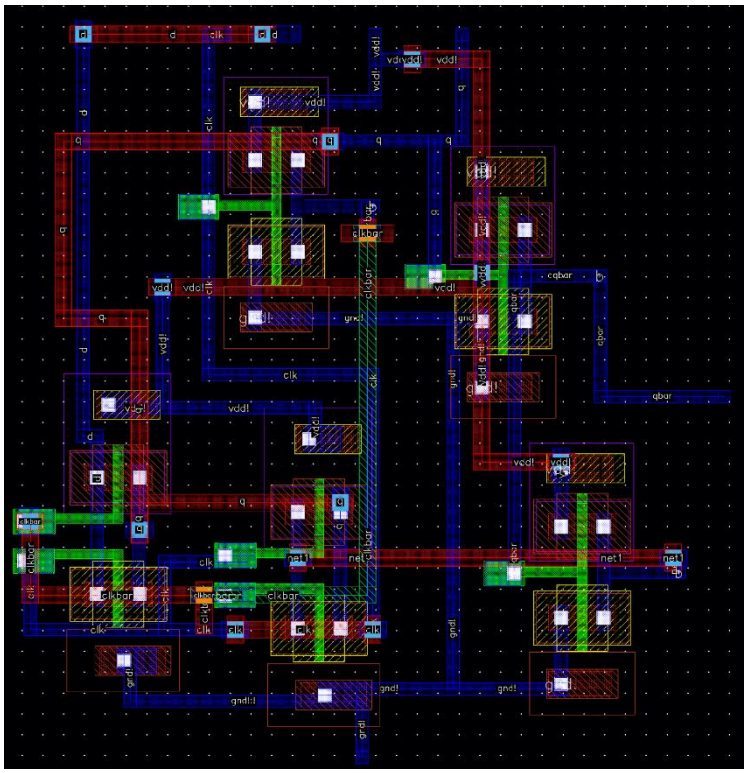

Fig 2.24: Layout of Dlatch

2.13 2 to 1 Multiplexer (1 bit)

The 1 bit 2 to 1 multiplexer is implemented in VHDL as shown in file “mux2to1_1bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.25. The inputs for multiplexer are 2 bit data and a select line and the output is output.

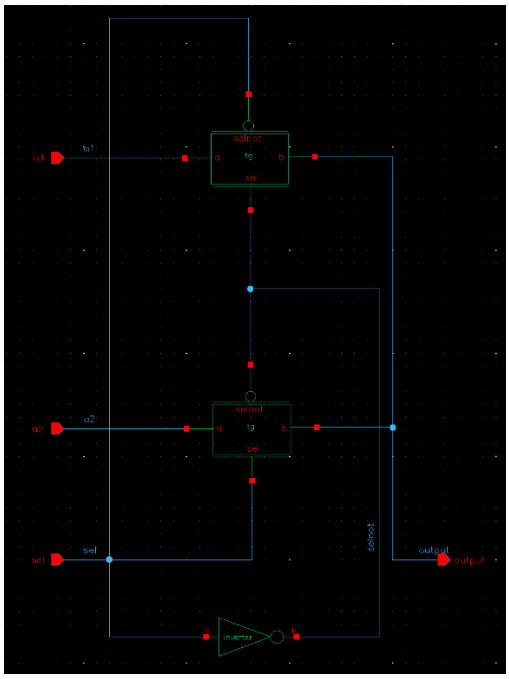

Fig 2.25: Schematic of 1 bit 2 to 1 multiplexer

For the above schematic the layout is implemented as shown in below Fig 2.26. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

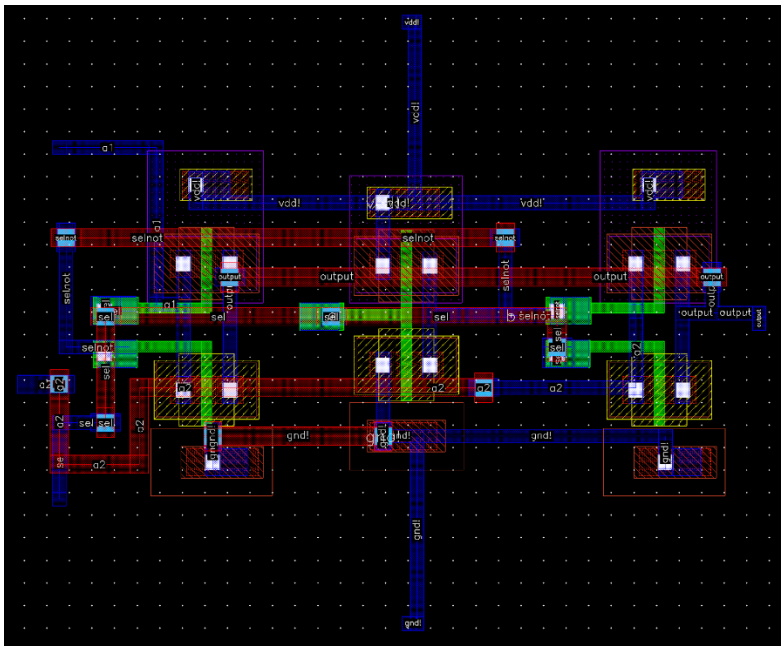

Fig 2.26: Layout of 1 bit 2 to 1 multiplexer

2.14 2 to 1 Multiplexer (4 bit)

The 4 bit 2 to 1 multiplexer is implemented in VHDL as shown in file “mux2to1_4bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.27. The inputs for multiplexer are 4 bit data and a select line and the output is output

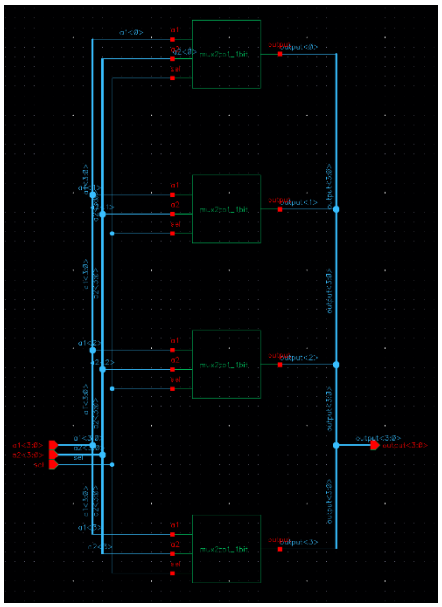

Fig 2.27: Schematic of 4 bit 2 to 1 multiplexer

For the above schematic the layout is implemented as shown in below Fig 2.28. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

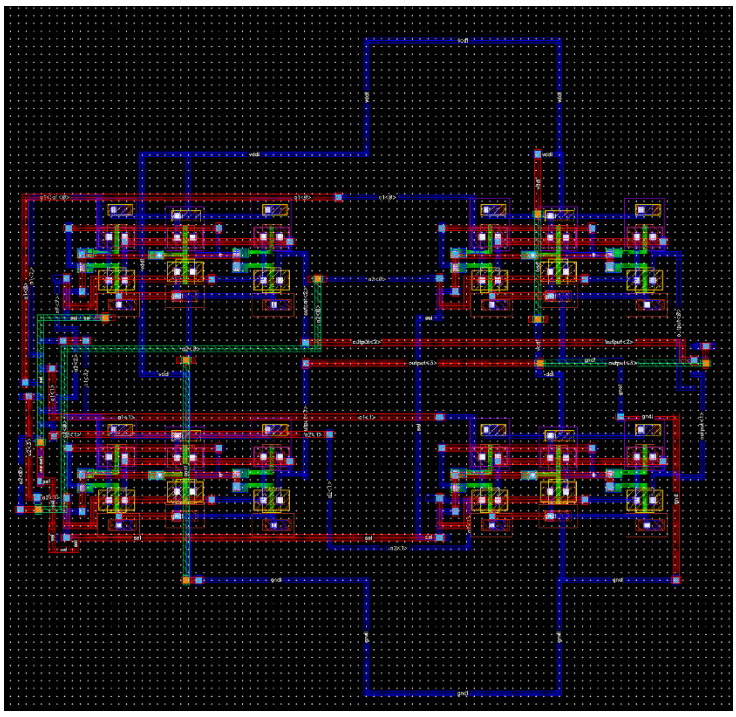

Fig 2.28: Layout of 4 bit 2 to 1 multiplexer

2.15 2 to 1 Multiplexer (8 bit)

The 8 bit 2 to 1 multiplexer is implemented in VHDL as shown in file “mux2to1_8bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.29. The inputs for multiplexer are 8 bit data and a select line and the output is output.

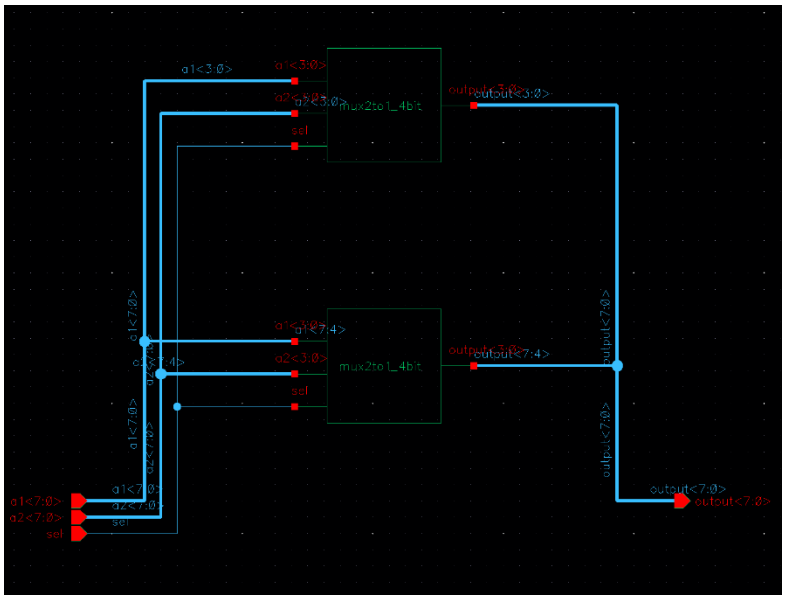

Fig 2.29: Schematic of 8 bit 2 to 1 multiplexer

For the above schematic the layout is implemented as shown in below Fig 2.30. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

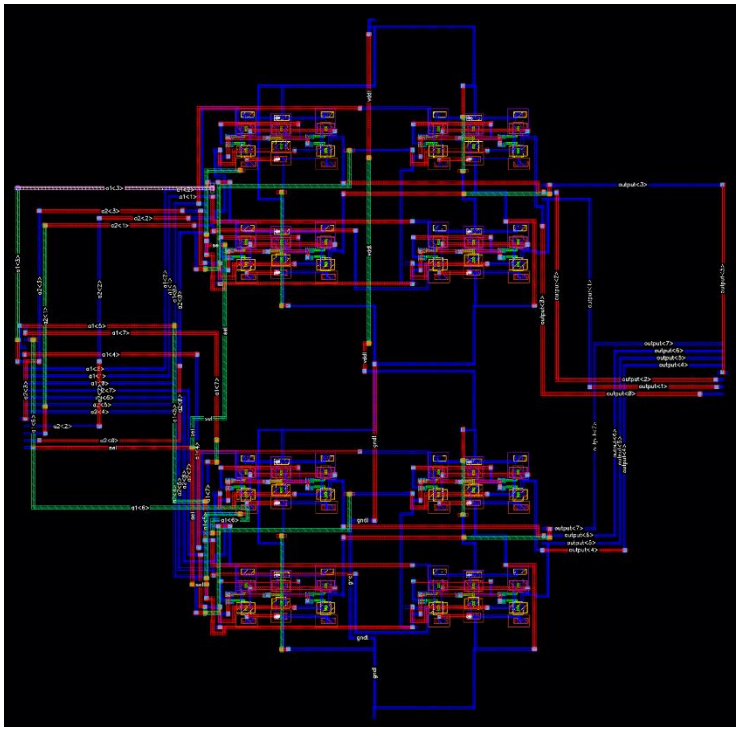

Fig 2.30: Layout of 8 bit 2 to 1 multiplexer

2.16 4 to 1 Multiplexer (1 bit)

The 1 bit 4 to 1 multiplexer is implemented in VHDL as shown in file “mux4to1_1bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.31. The inputs for multiplexer are 4 bit data and two select lines and the output is output. Here three 2 to 1 multiplexers are used to build the 1 bit 4 to 1 multiplexer.

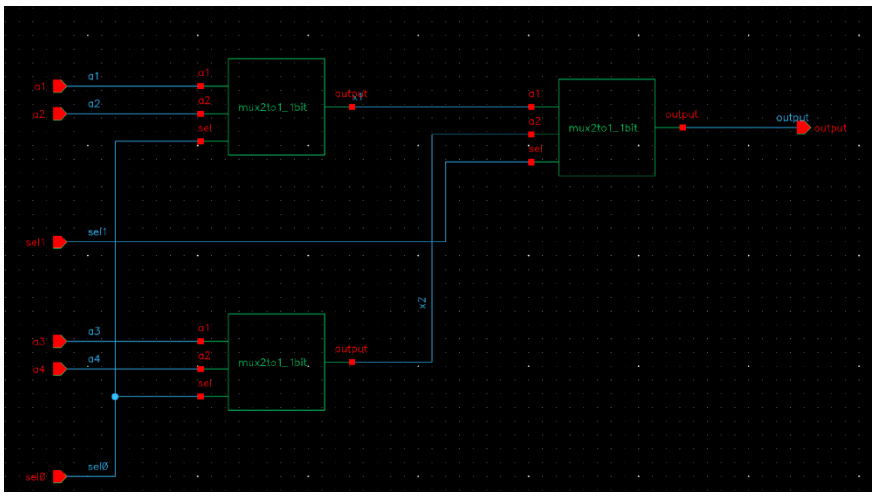

Fig 2.31: Schematic of 1bit 4 to 1 multiplexer

For the above schematic the layout is implemented as shown in below Fig 2.32. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

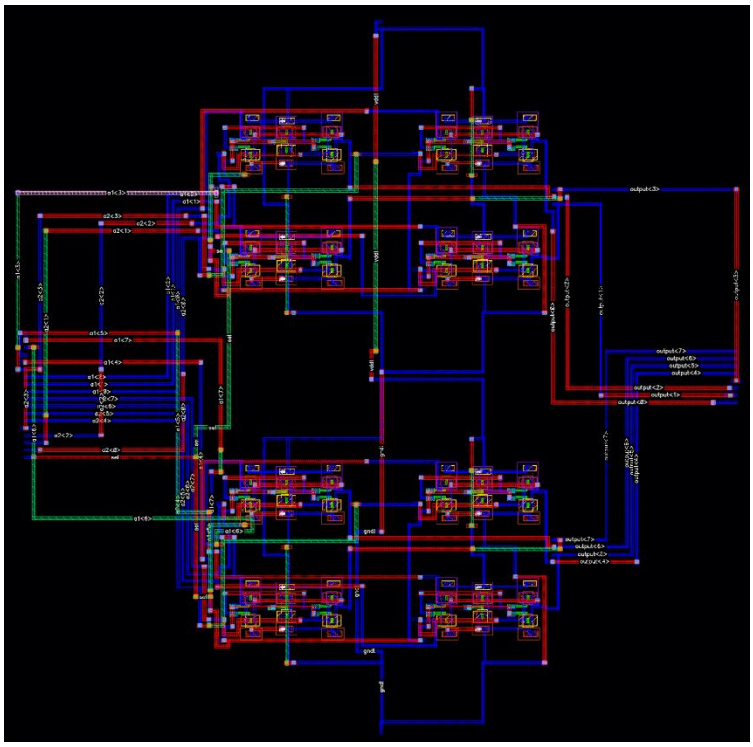

Fig 2.32: Layout of 1 bit 4 to 1 multiplexer

2.17 4 to 1 Multiplexer (2 bit)

The 2 bit 4 to 1 multiplexer is implemented in VHDL as shown in file “mux4to1_2bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.33. The inputs for multiplexer are 2 bit 4 data lines and two select lines and the output is output. Here two 4 to 1 multiplexers are used to build the 2 bit 4 to 1 multiplexer.

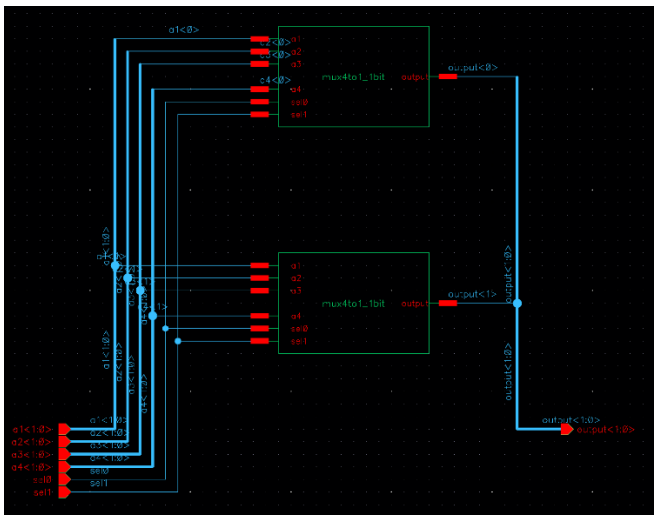

Fig 2.33: Schematic of 2 bit 4 to 1 multiplexer

For the above schematic the layout is implemented as shown in below Fig 2.34. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

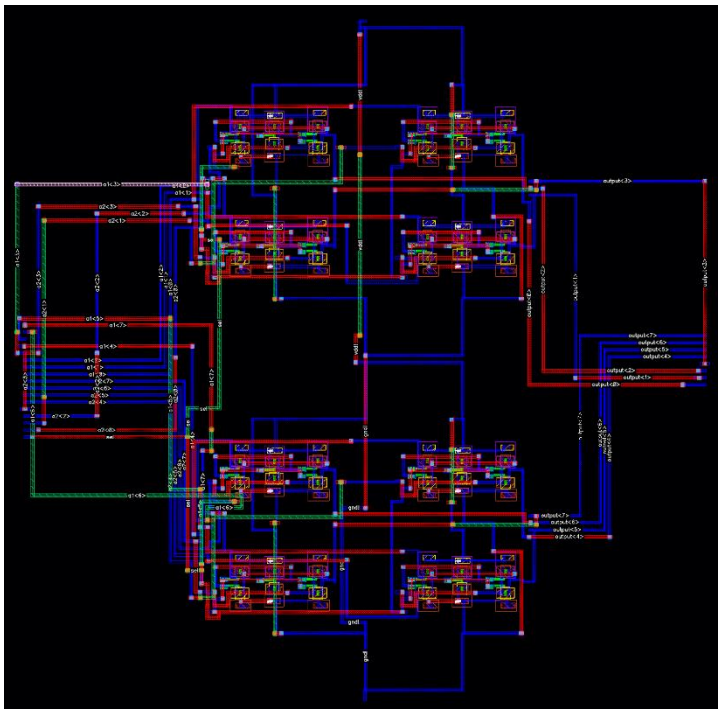

Fig 2.34: Layout of 2 bit 4 to 1 multiplexer

2.18 Transmission Gate

The Transmission gate is implemented in VHDL as shown in file “tg.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 2.35. The input is a, and the clock signal is given as clk and clkb to nmos and pmos respectively. The output is b. In transmission gate whenever the clk is high, the input goes to output.

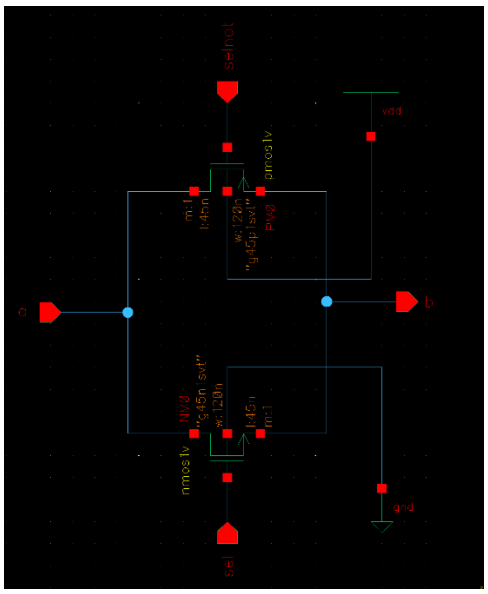

Fig 2.35: Schematic of Transmission gate

For the above schematic the layout is implemented as shown in below Fig 2.36. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

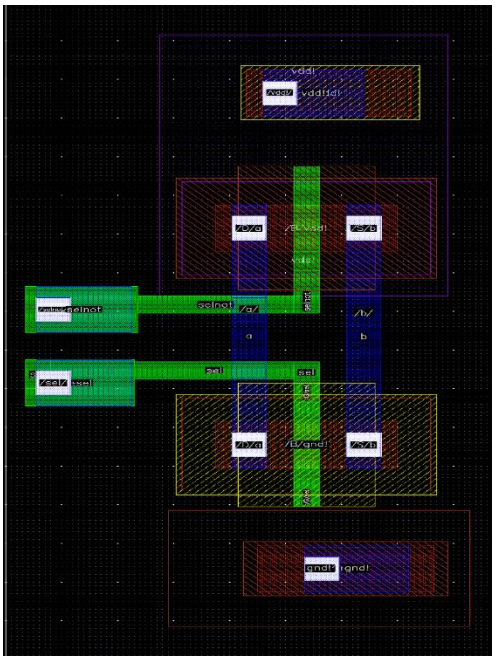

Fig 2.36: Layout of Transmission gate

IMPLEMENTATION OF CACHE

The cache is implemented by designing the individual blocks as mentioned in previous section. First, single bit cache cell is designed and then one byte cache. Then cache block is designed consists of four bytes and then main cache is designed. For performing the four-cache operations state machine is designed using shift register cells, multiplexers, D flip flops, and some basic logic gates.

3.1 Cache cell Decoder

The Cache cell decoder is implemented in VHDL as shown in file “cachecell_decoder.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.1. The input signals are read write signal and chip enable signal. Here the output is read enable and write enable signals.

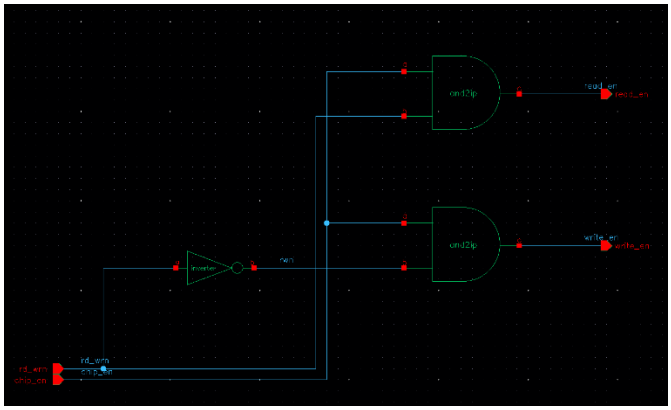

Fig 3.1: Schematic of Cache cell decoder

For the above schematic the layout is implemented as shown in below Fig 3.2. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

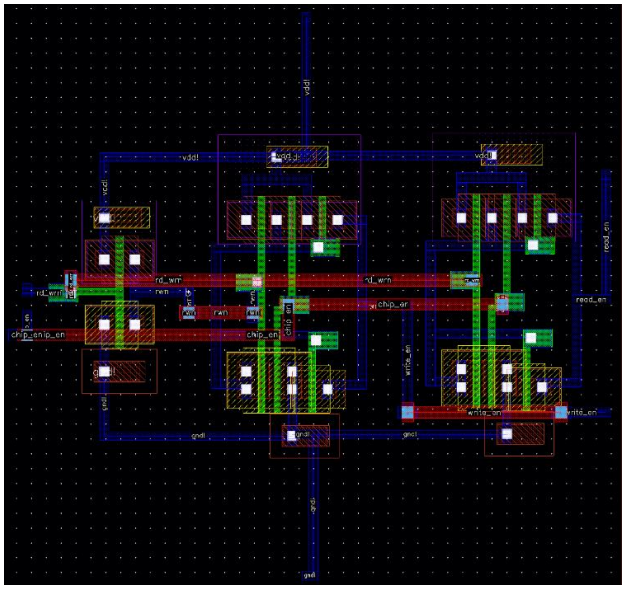

Fig 3.2: Layout of Cache cell decoder

3.2 Single bit Cache Cell

Cache cell is designed using d-latch and Transmission gate and cache cell decoder. Here the Read_Write operation is performed by providing one input signal as RD/WR'. The functionality is implemented in VHDL in the file "cachecell.vhd".The below logic circuit represents the cachecell in below Fig 3.3. Coming to its operation, the inputs are enable and RD/WR'. When both the inputs are high, read operation is done. While the R/W' is low, the write operation is done.

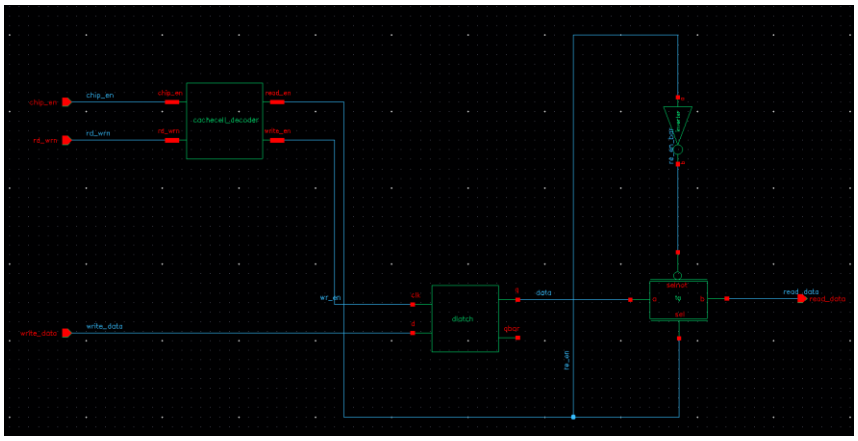

Fig 3.3: Logic diagram of Cache Cell

For the above schematic the layout is implemented as shown in below Fig 3.4. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

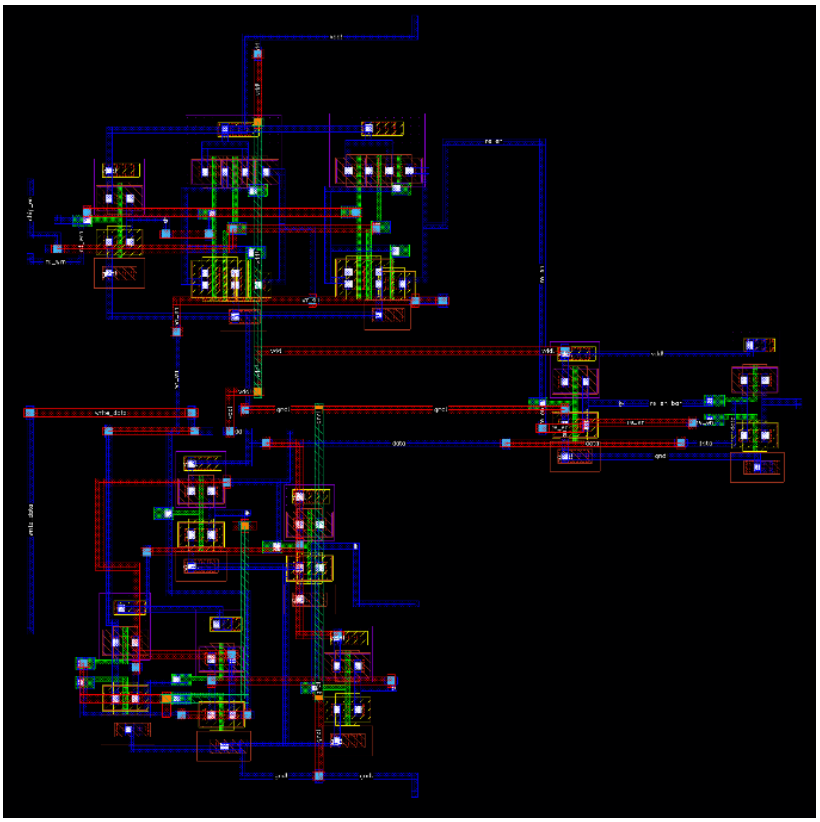

Fig 3.4: Layout of Cache Cell

3.3 Single byte Cache

The one-byte cache represents the 8bits of the cache cell. Here, the one- byte cache is designed by combining the 8 single bit cache cells. The single byte cache VHDL code is written in the file named “cachebyte.vhd” in the attached zip file. Here one byte cache is designed as shown below in Fig 3.5.

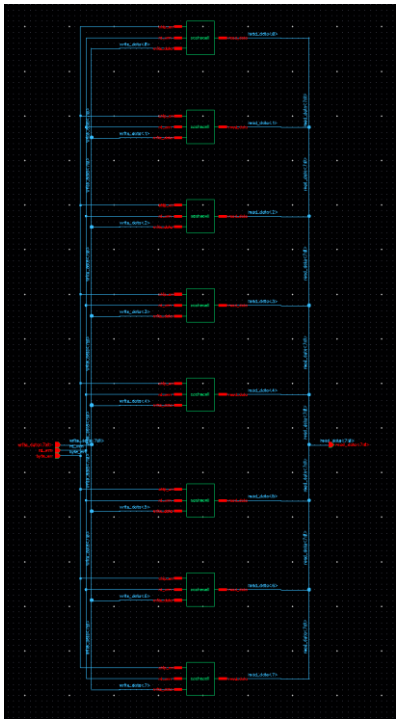

Fig 3.5: Schematic of Cache byte

For the above schematic the layout is implemented as shown in below Fig 3.6. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

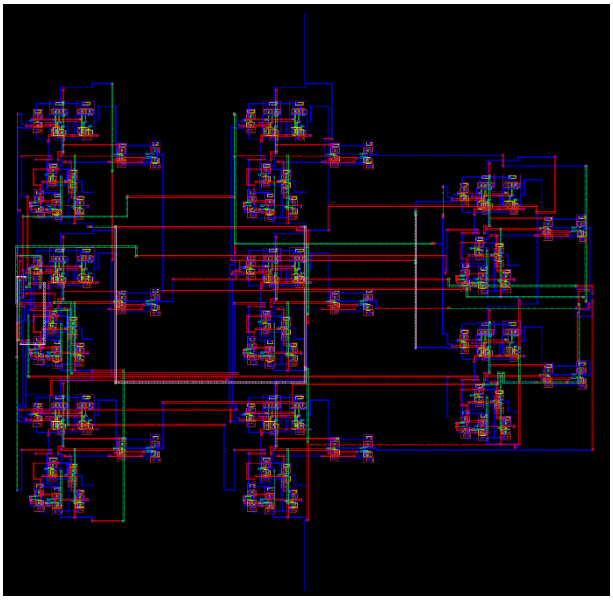

Fig 3.6: Layout of Cache byte

3.4 Register Cell

The Register Cell is implemented in VHDL as shown in file “registercell.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.7. The register cell is implemented using the multiplexers, and D flip flops.

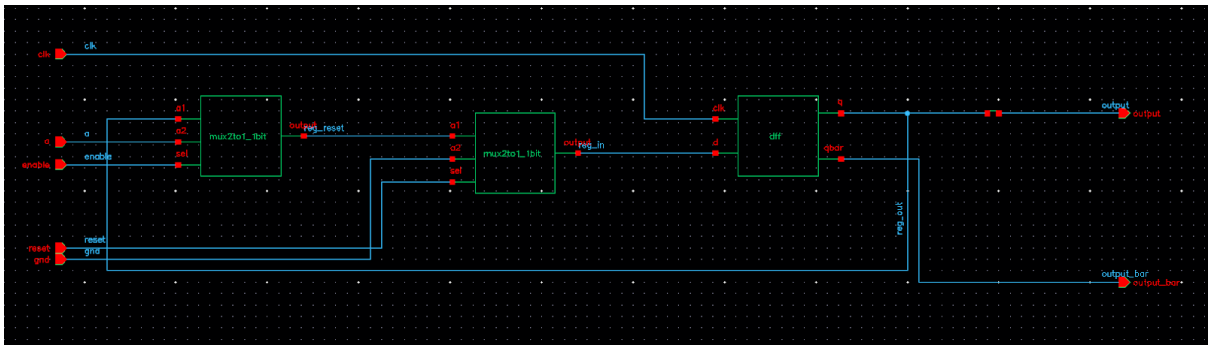

Fig 3.7: Schematic of Register cell

For the above schematic the layout is implemented as shown in below Fig 3.8. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

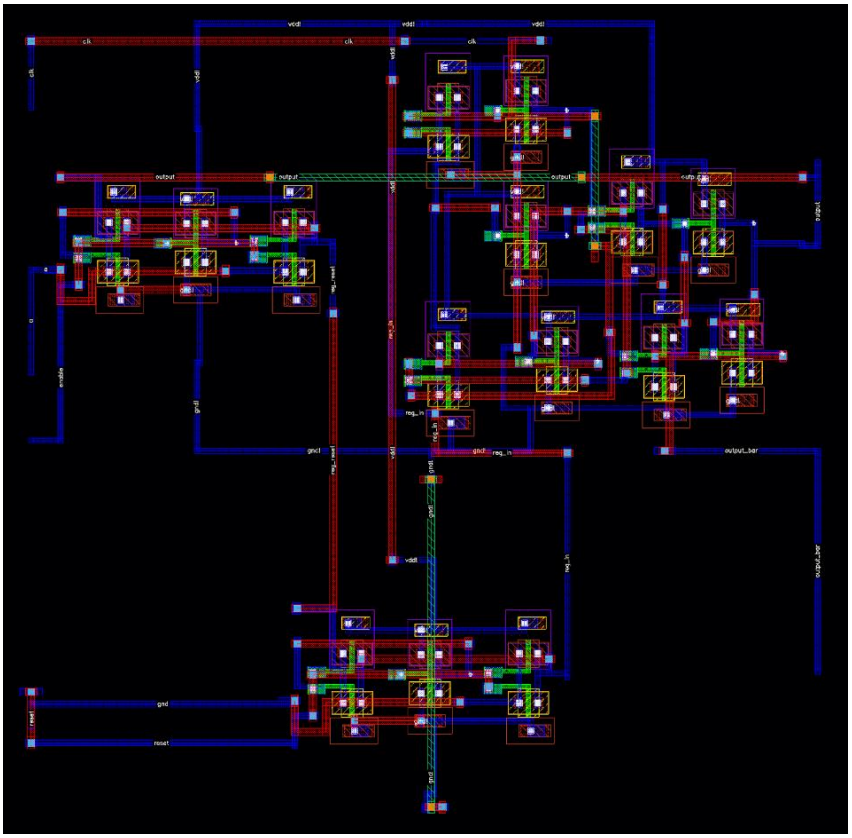

Fig 3.8: Layout of Register cell

3.5 Tag Cell

The Tag Cell is implemented in VHDL as shown in file “tagcell.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.9. The tag cell is implemented using the register cells.

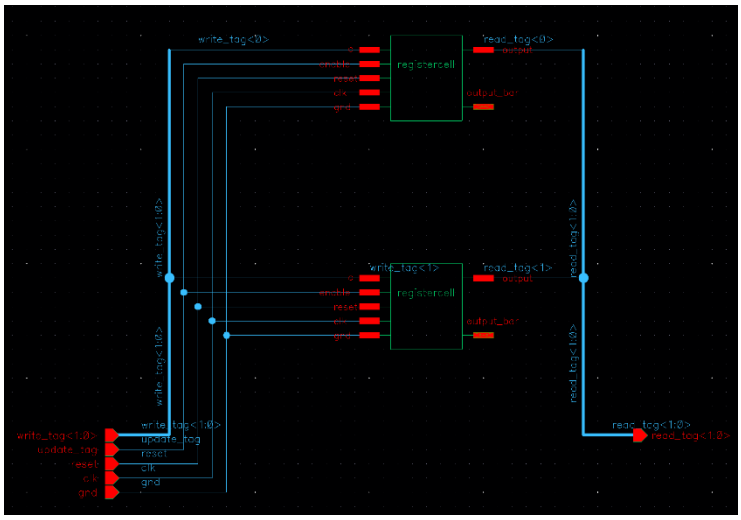

Fig 3.9: Schematic of Tag cell

For the above schematic the layout is implemented as shown in below Fig 3.10. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

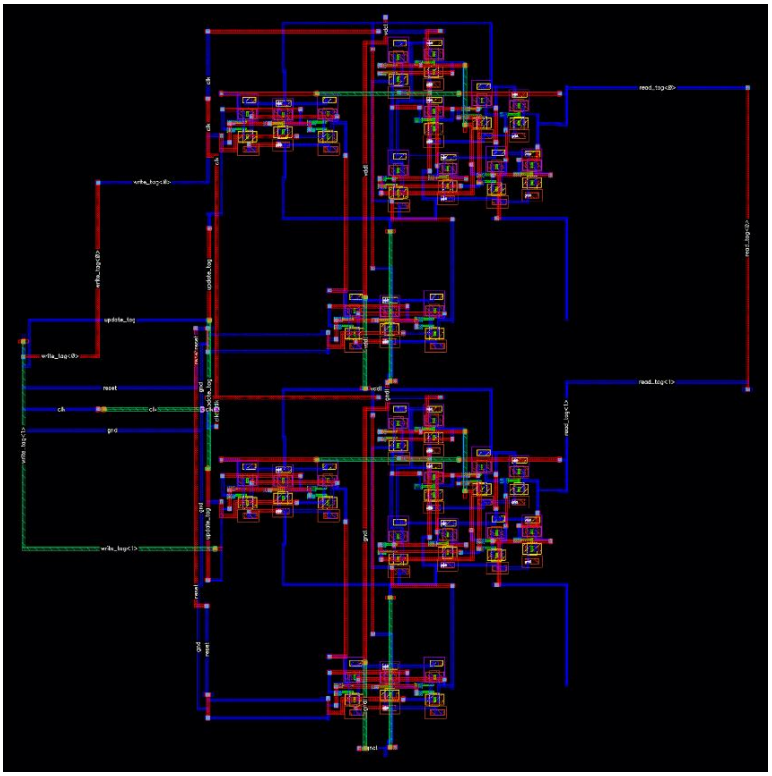

Fig 3.10: Layout of Tag cell

3.6 Cache block

Cache block represents the four bytes of cache. Previously designed single byte cache is used to implement the four bytes by combining the single byte cache. The VHDL code for cache block is written in the file named “cacheblock.vhd” in the attached zip file. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.11. The Cache block is implemented using the cache bytes and 2input and gates.

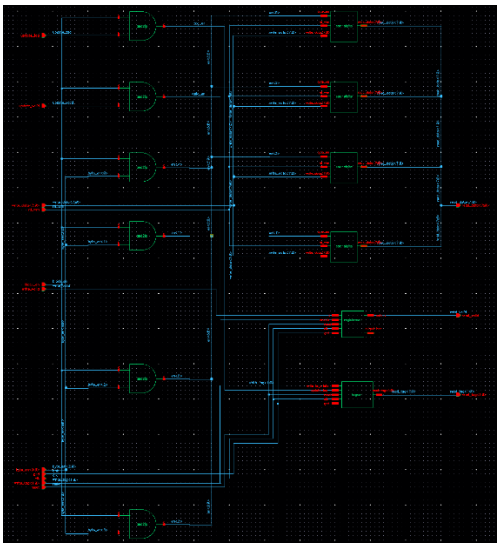

Fig 3.11: Schematic of Cache block

For the above schematic the layout is implemented as shown in below Fig 3.12. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

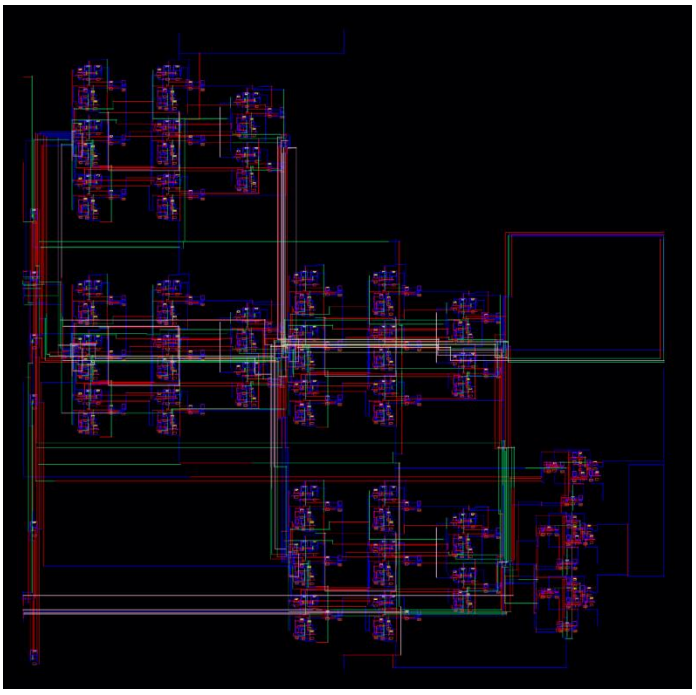

Fig 3.12: Layout of Cache block

3.7 16-byte Cache

The 16-byte Cache is implemented by using four cache blocks. The VHDL code for 16-byte cache is shown below and written in file named “cache.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.14. The cache is implemented using four cache blocks.

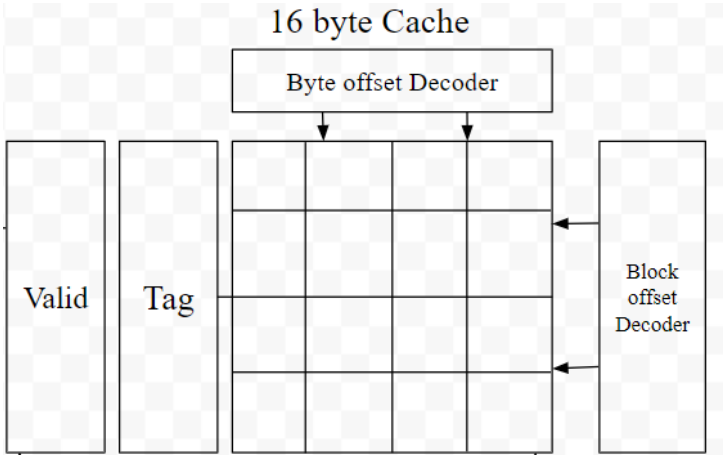

Fig 3.13: Representation 16-byte cache

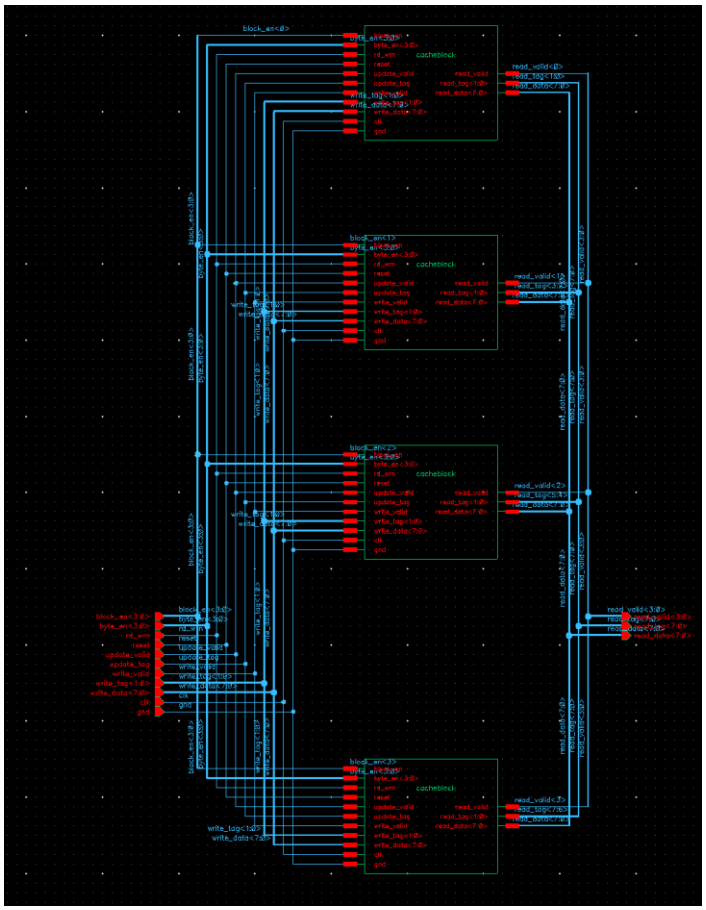

Fig 3.14: Schematic of Cache

For the above schematic the layout is implemented as shown in below Fig 3.15. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.16.

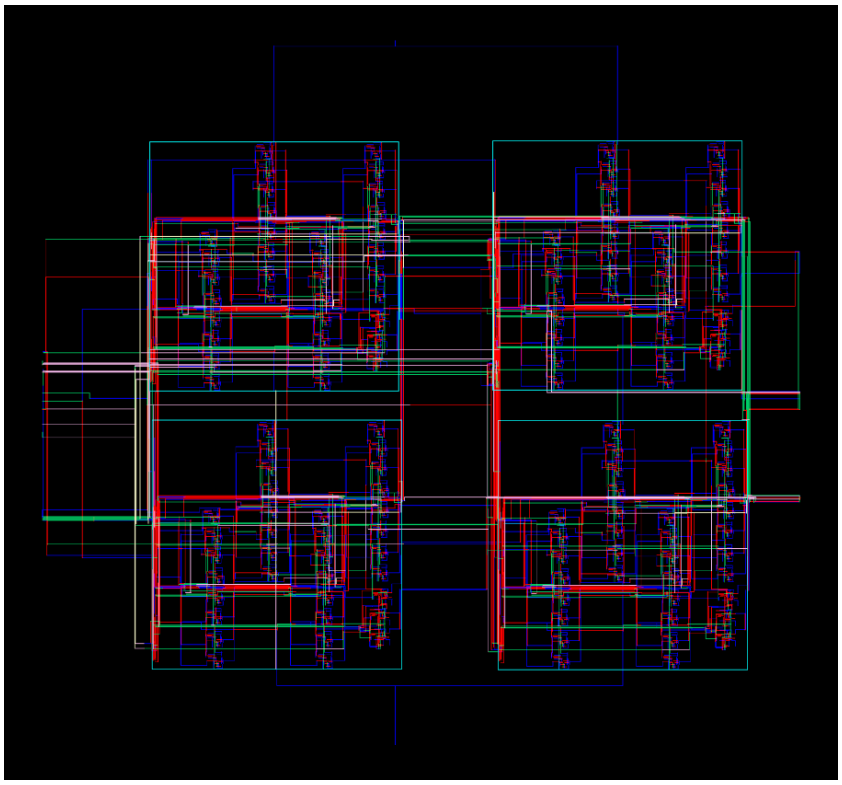

Fig 3.15: Layout of Cache

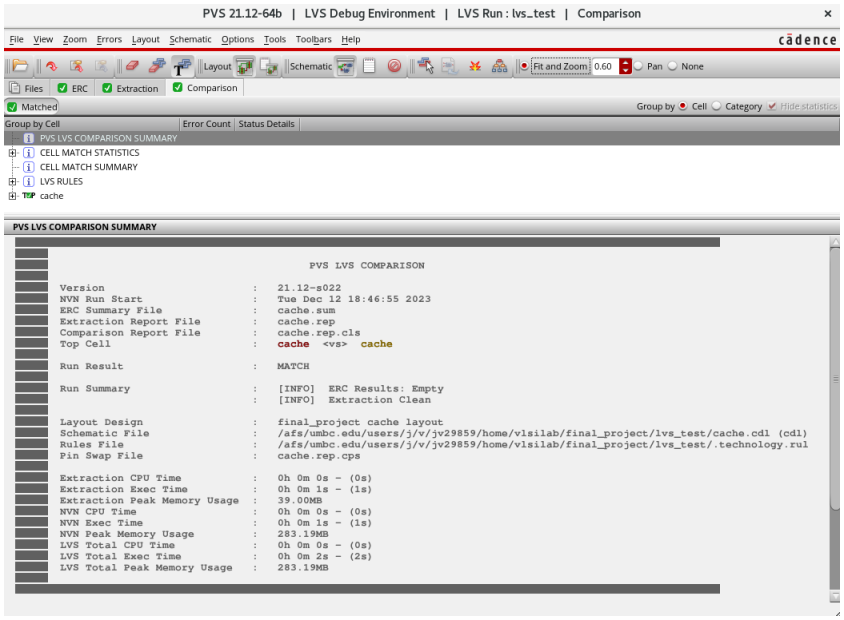

Fig 3.16: LVS report of Cache

3.8 Cache Decoder

The Cache decoder VHDL code is written in file named “cache_decoder.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.17. The cache decoder is implemented using 2x4 decoders and multiplexers.

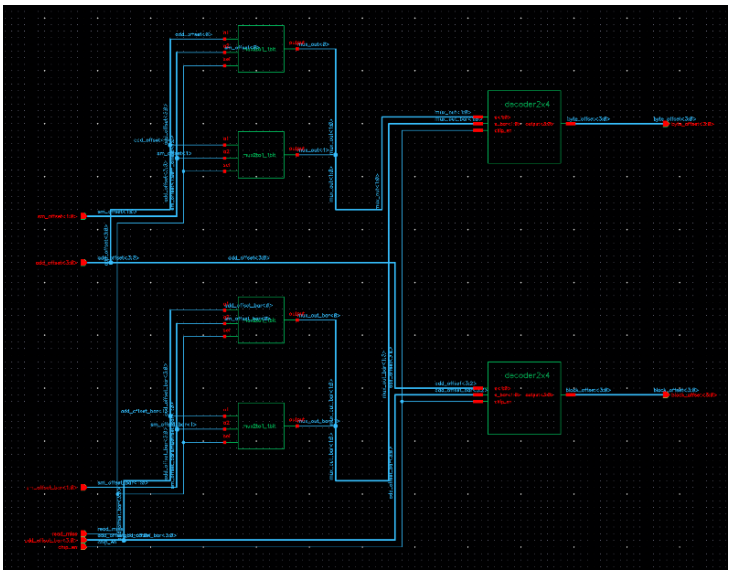

Fig 3.17: Schematic of Cache decoder

For the above schematic the layout is implemented as shown in below Fig 3.18. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.19.

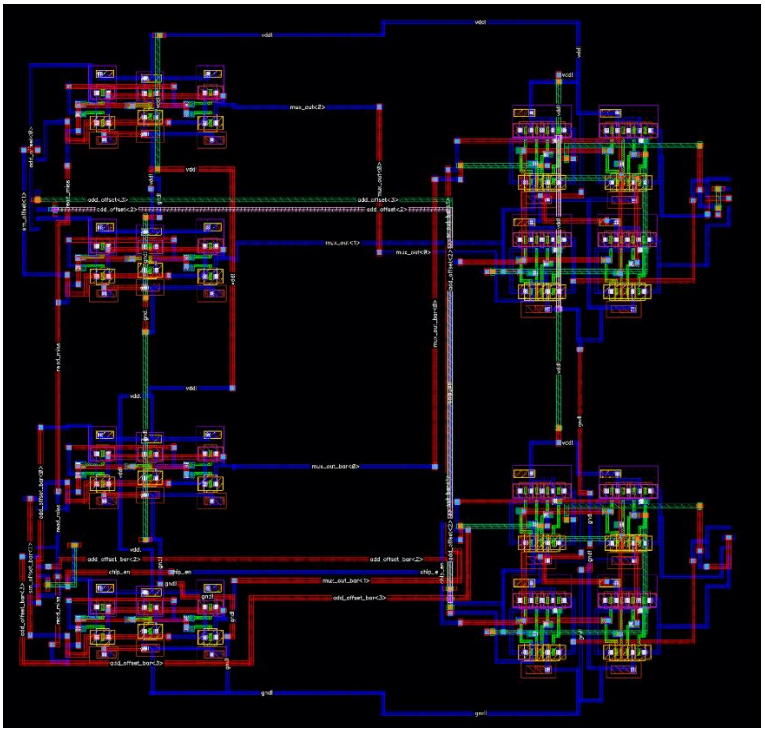

Fig 3.18: Layout of the Cache decoder

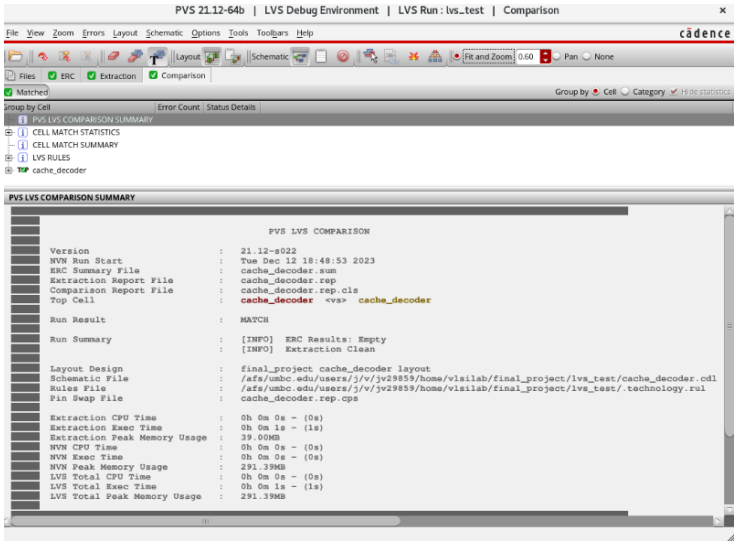

Fig 3.19: LVS report of the Cache decoder

3.9 Address Register

The Address register VHDL code is written in file named “add_reg.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.20. The address register is implemented using register cells.

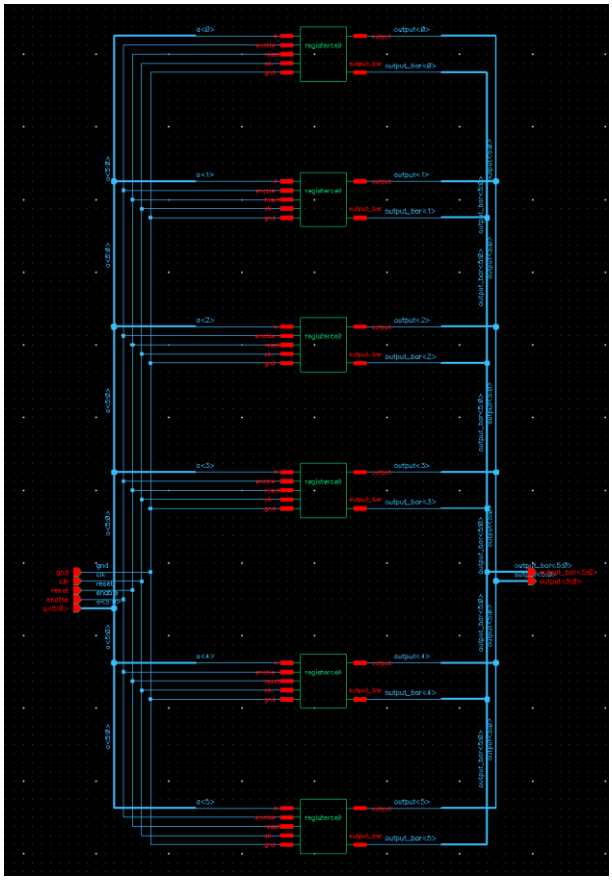

Fig 3.20: Schematic of the Address register

For the above schematic the layout is implemented as shown in below Fig 3.21. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.22.

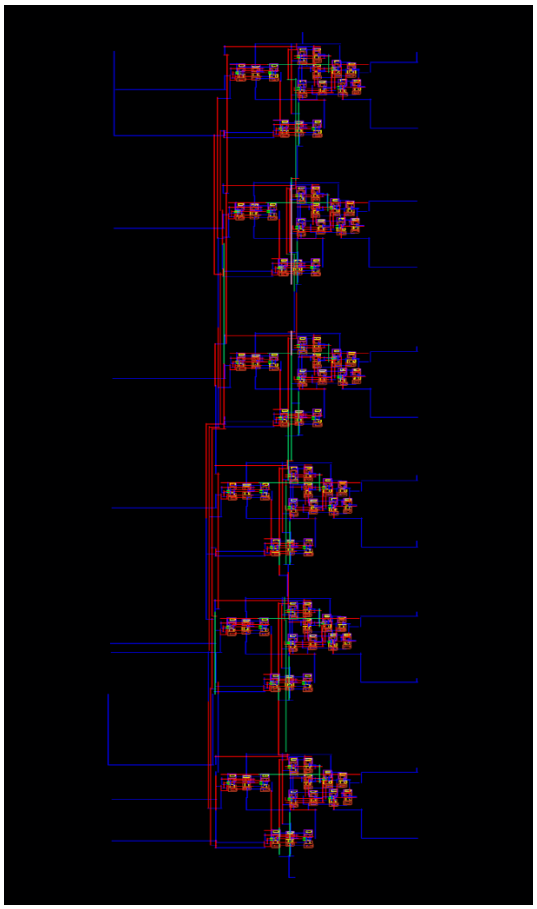

Fig 3.21: Layout of the Address register

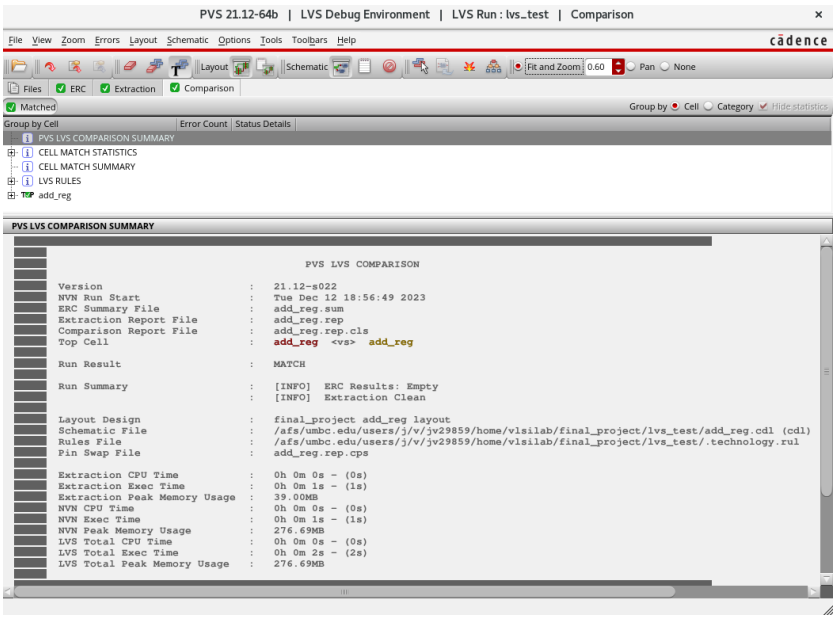

Fig 3.22: LVS report of the Address register

3.10 Data Register

The Data register VHDL code is written in file named “data_reg.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.23. The data register is implemented using register cells.

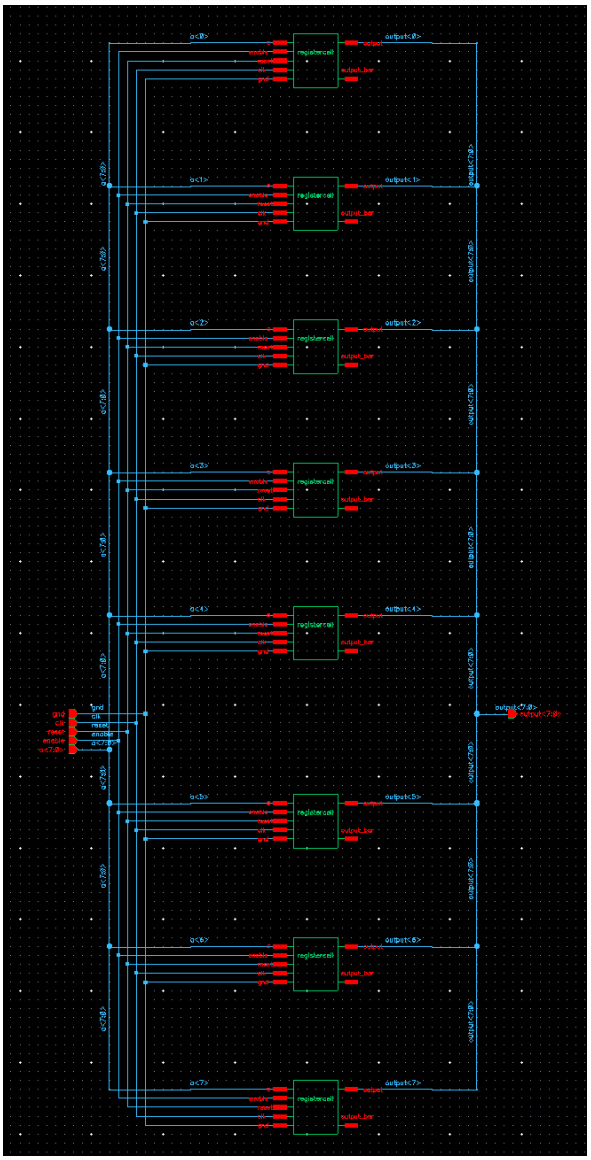

Fig 3.23: Schematic of the Data register

For the above schematic the layout is implemented as shown in below Fig 3.24. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.25.

Fig 3.24: Layout of the Data register

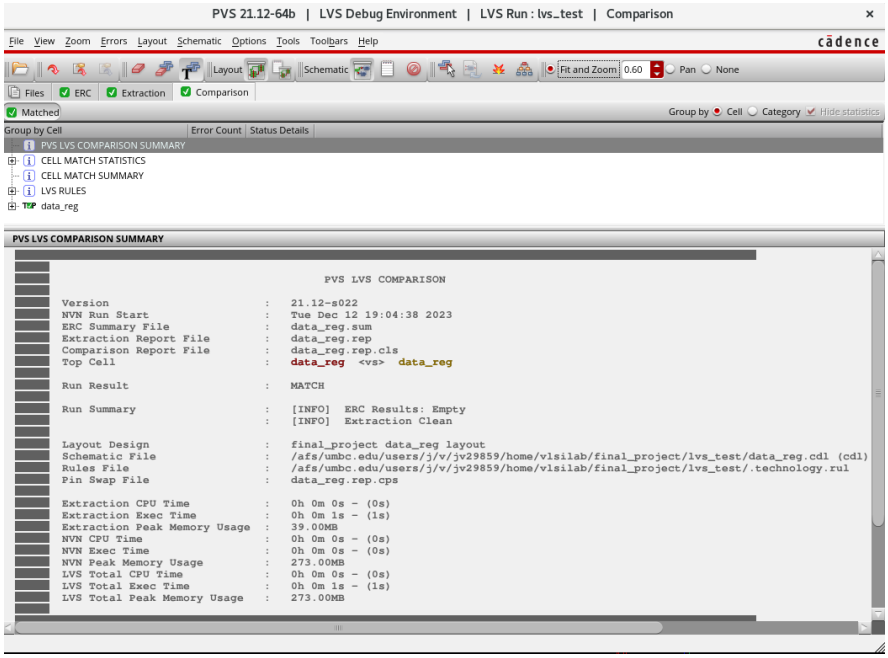

Fig 3.25: LVS report of the Data register

3.11 Hit/Miss Checker

The Hit miss checker is basically does the comparator operation and produce the output as hit or miss. The VHDL code is written in file named “hitmiss_checker.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.26. The Hit miss checker is implemented using multiplexers, xor gates and nor gate.

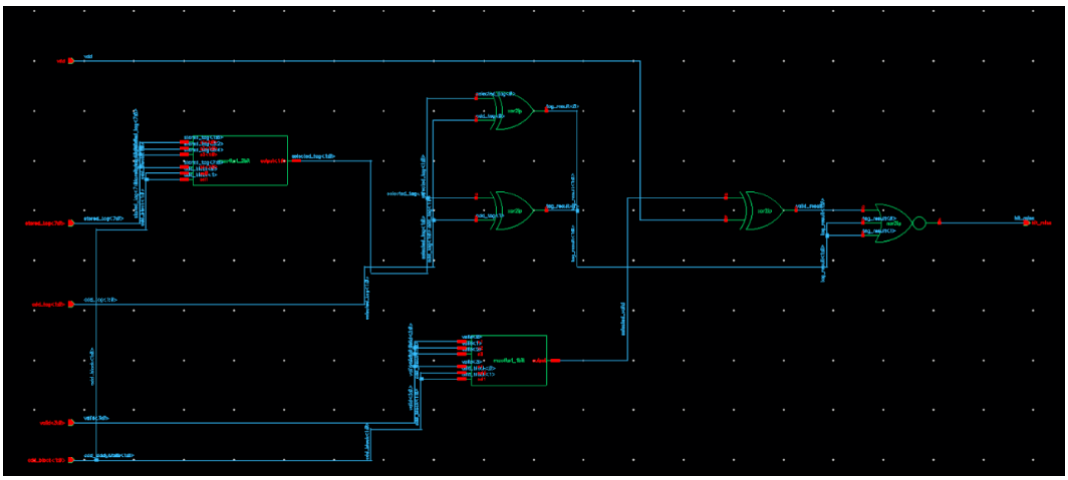

Fig 3.26: Schematic of the Hit/miss checker

For the above schematic the layout is implemented as shown in below Fig 3.27. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.28.

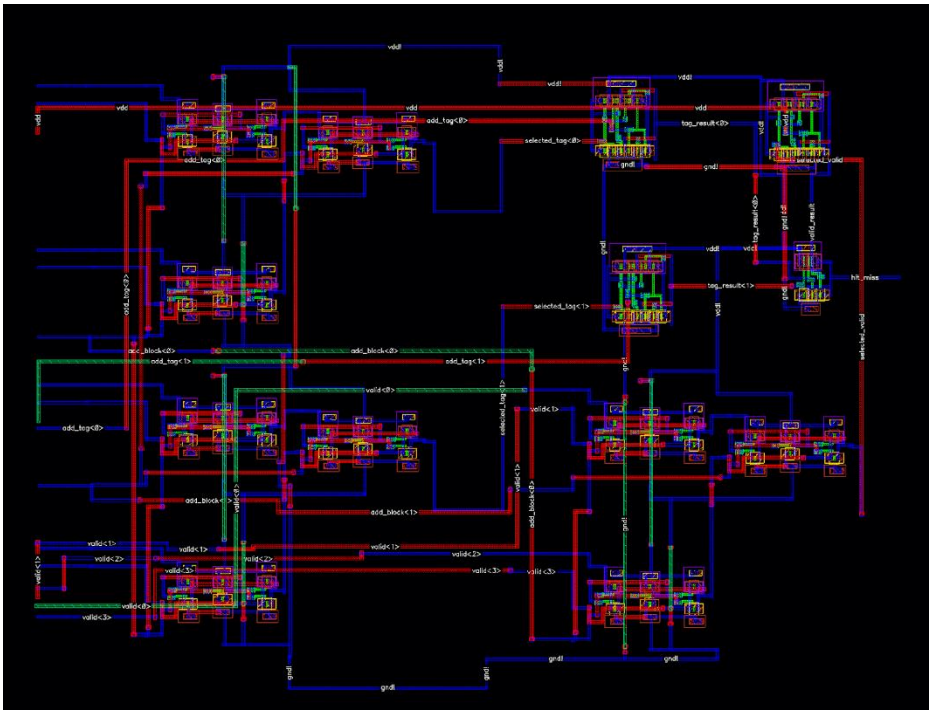

Fig 3.27: Layout of the Hit/miss checker

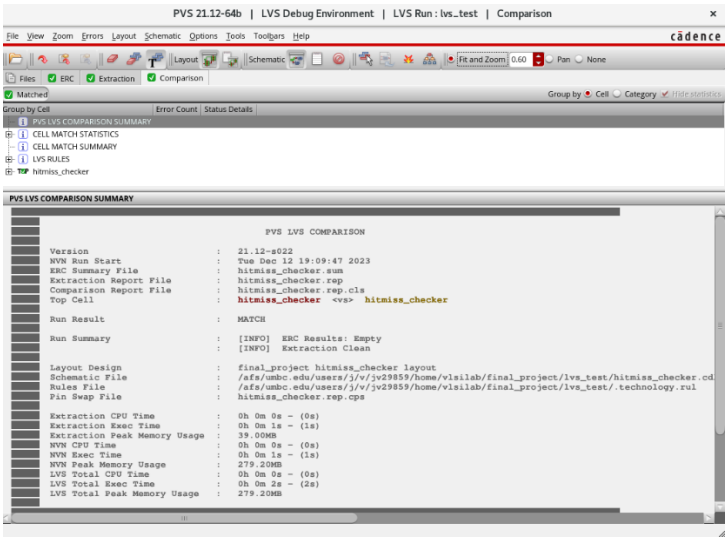

Fig 3.27: LVS report of the Hit/miss checker

3.12 Output Enable (6-bit)

The 6 bit output enable VHDL code is written in file named “outputenable_6bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.28. The 6bit output enable is implemented using transmission gates.

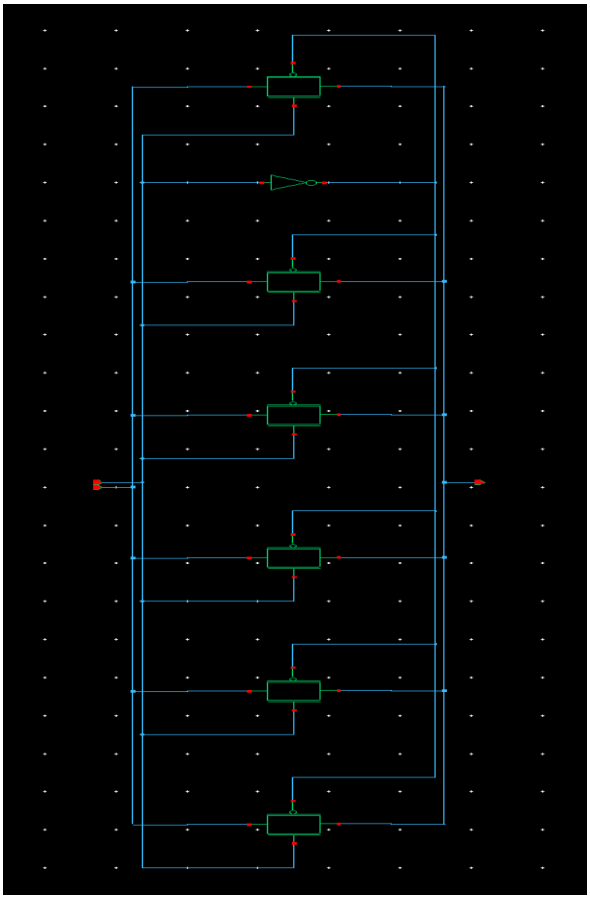

Fig 3.28: Schematic of the 6-bit output enable

For the above schematic the layout is implemented as shown in below Fig 3.29. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.30.

Fig 3.29: Layout of the 6-bit output enable

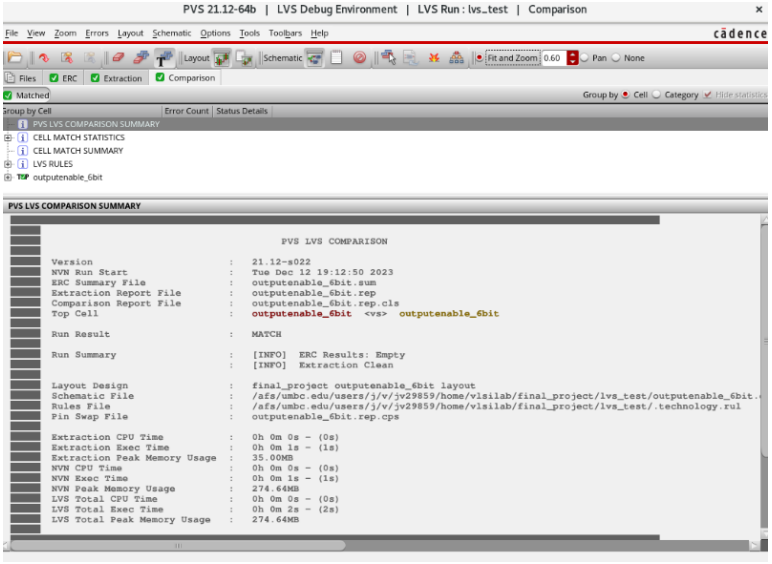

Fig 3.30: LVS of the 6-bit output enable

3.13 Output Enable (8-bit)

The 8-bit output enable VHDL code is written in file named “outputenable_8bit.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.31. The 8-bit output enable is implemented using transmission gates.

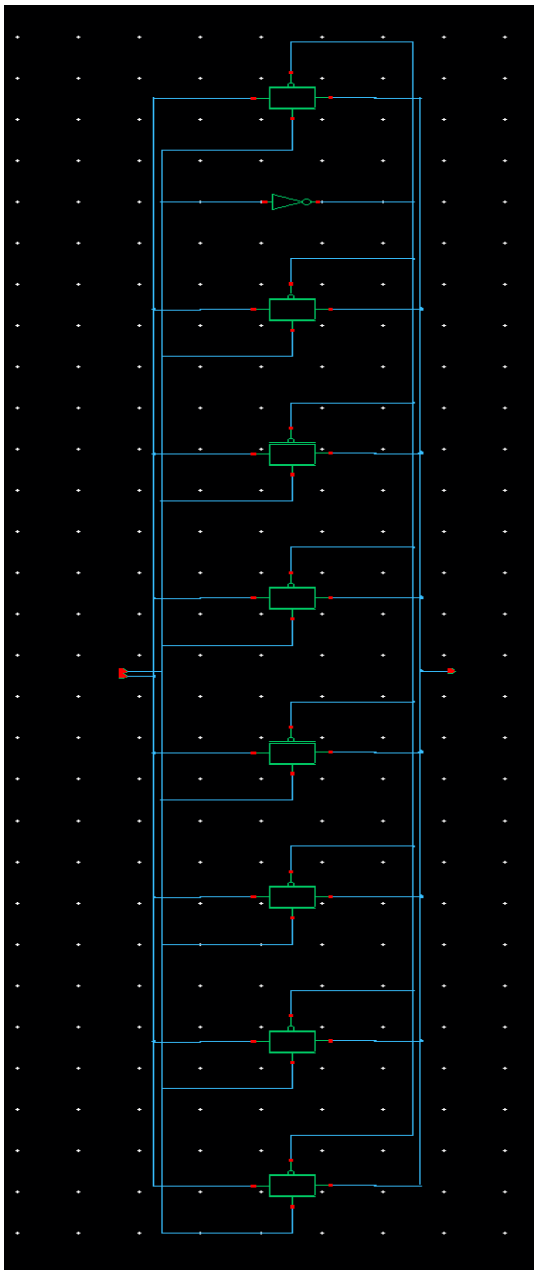

Fig 3.31: Schematic of the 8-bit output enable

For the above schematic the layout is implemented as shown in below Fig 3.32. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.33.

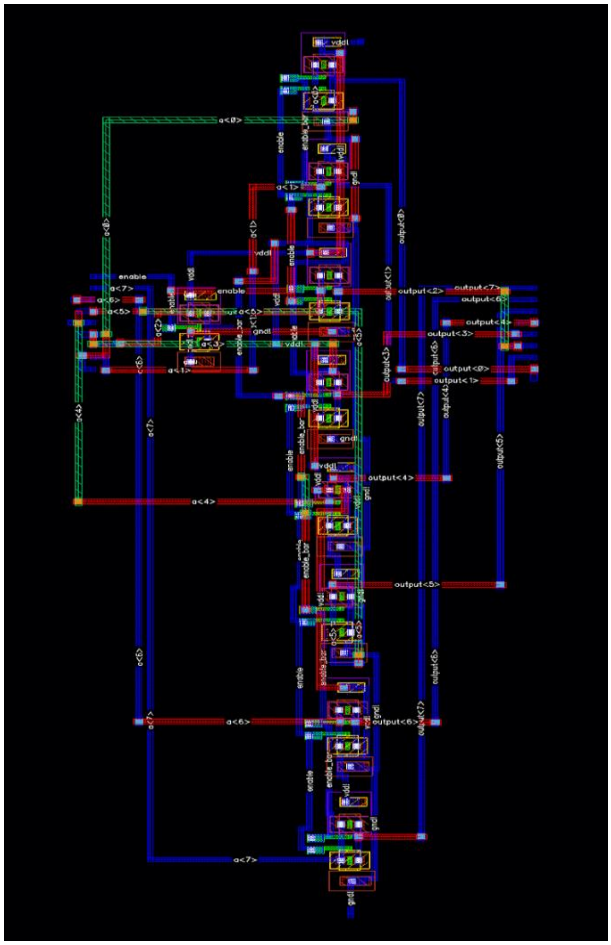

Fig 3.32: Layout of the 8-bit output enable

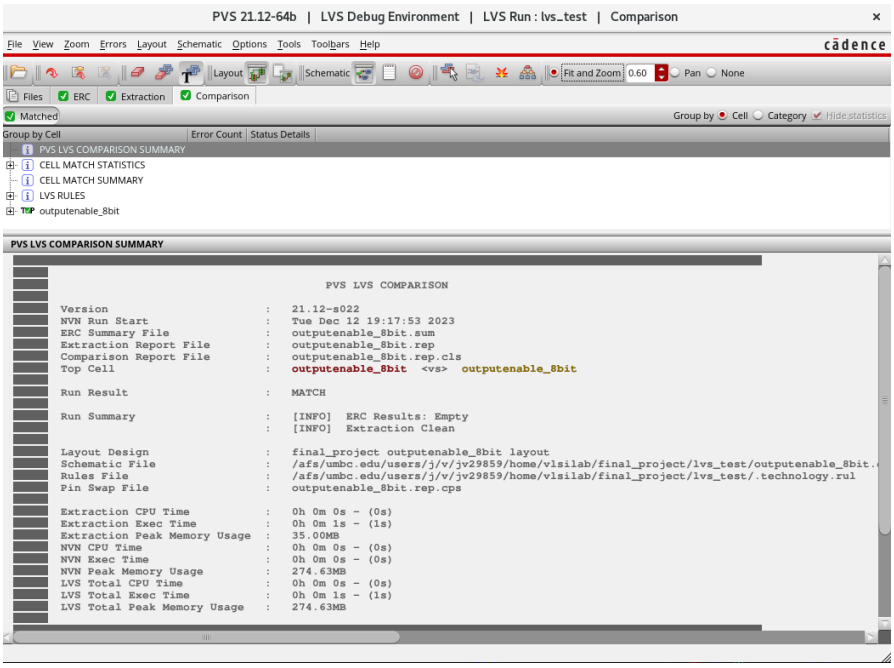

Fig 3.33: LVS report of the 8-bit output enable

3.14 Shift Register cell

The shift register VHDL code is written in file “shiftreg_cell.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.34. The shift register cell is implemented using D flip flop and multiplexer.

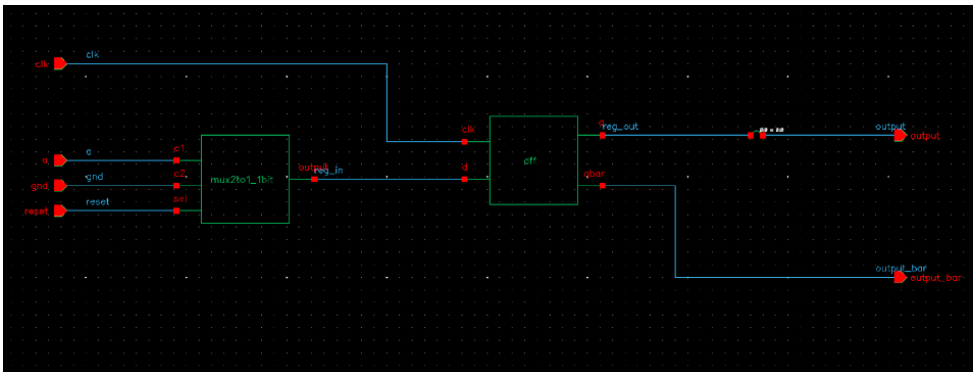

Fig 3.34: Schematic of the Shift register cell

For the above schematic the layout is implemented as shown in below Fig 3.35. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result.

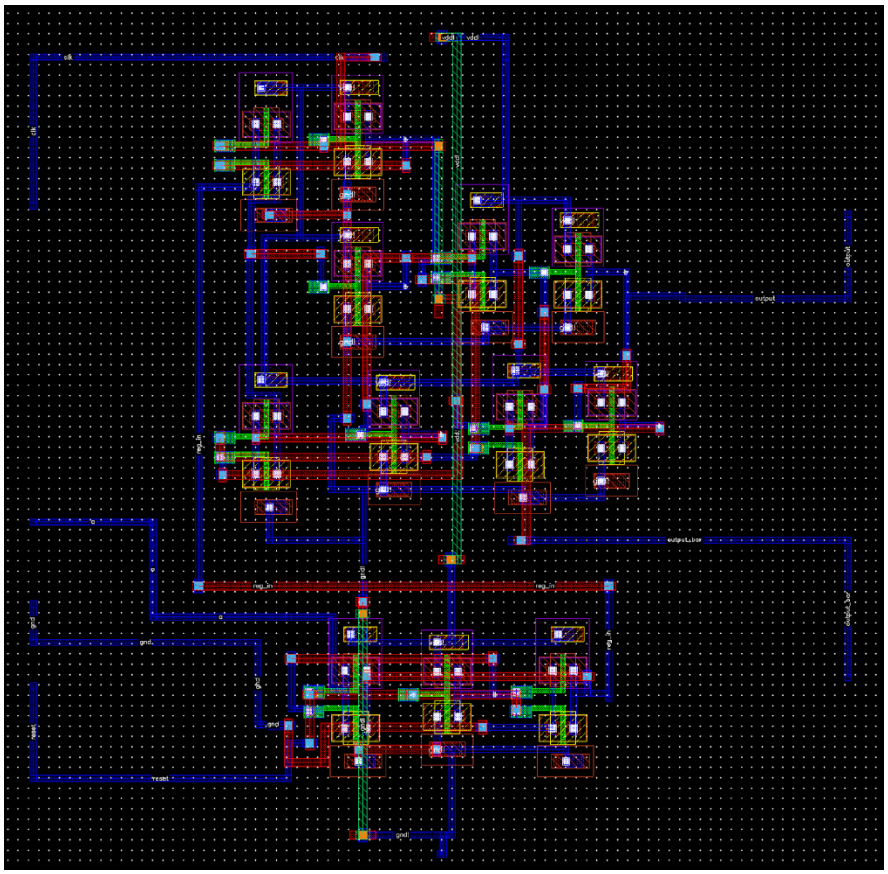

Fig 3.35: Layout of the Shift register cell

3.15 State Machine

The state machine designed performs the Read miss, Read hit, Write miss, and Write hit operations. Based on the input R/W' signal and the Hit/miss checker output i.e., hit/miss, the state machine starts working. Here, in the implementation of cache the read miss operation takes 19 clock cycles, write hit operation need 3 clock cycles, read hit operation takes 2 clock cycles, and the write miss operation takes 3 clock cycles. The VHDL code for state machine is written in the file "state_machine.vhd". The state machine is implemented using shift register cells, multiplexers, D flip flops, and some basic logic gates to perform the required operations. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.36.

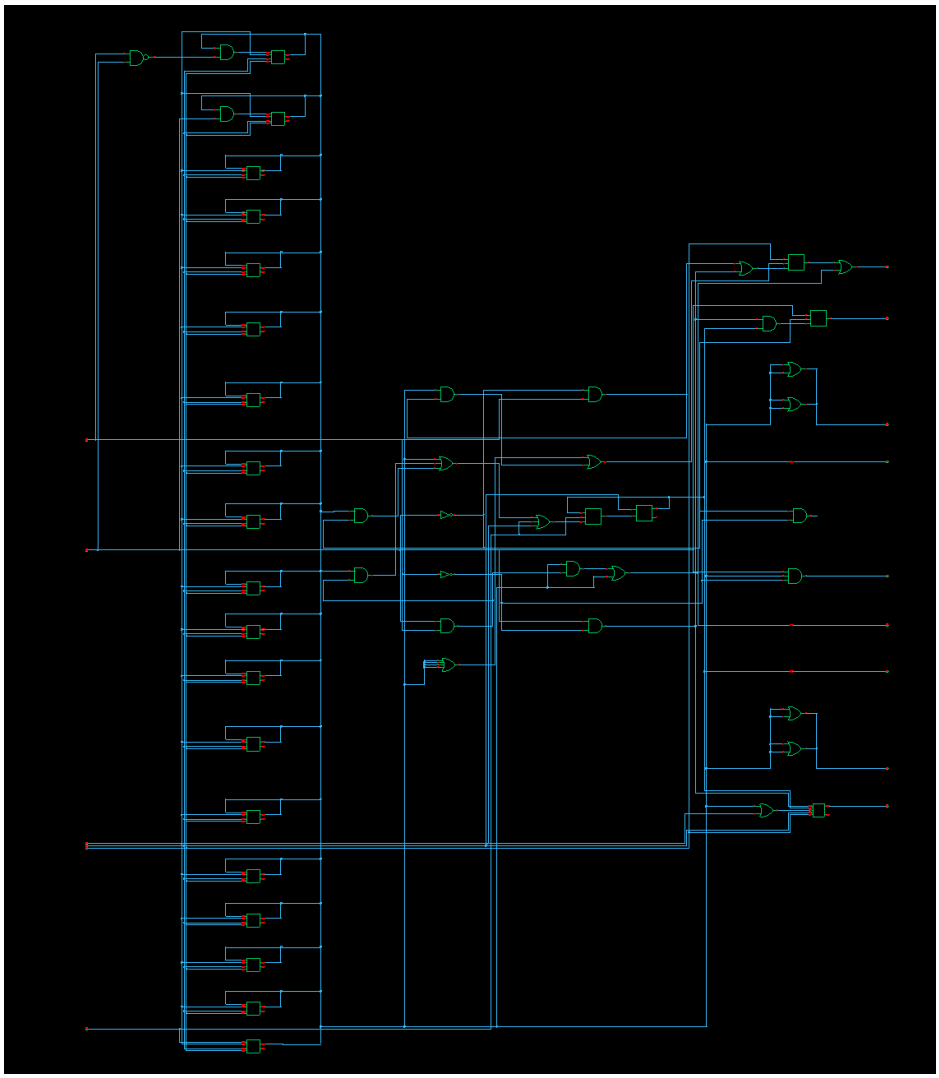

Fig 3.36: Schematic of the State machine

For the above schematic the layout is implemented as shown in below Fig 3.37. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.38.

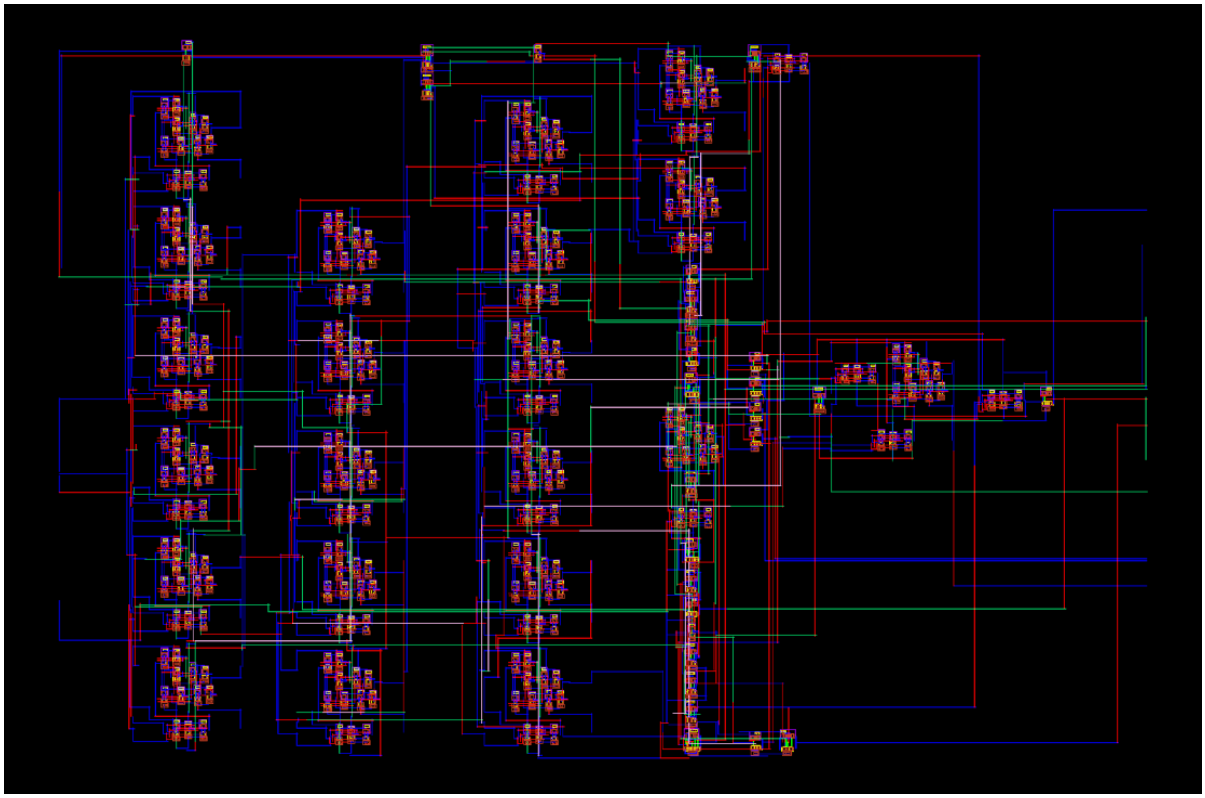

Fig 3.37: Layout of the State machine

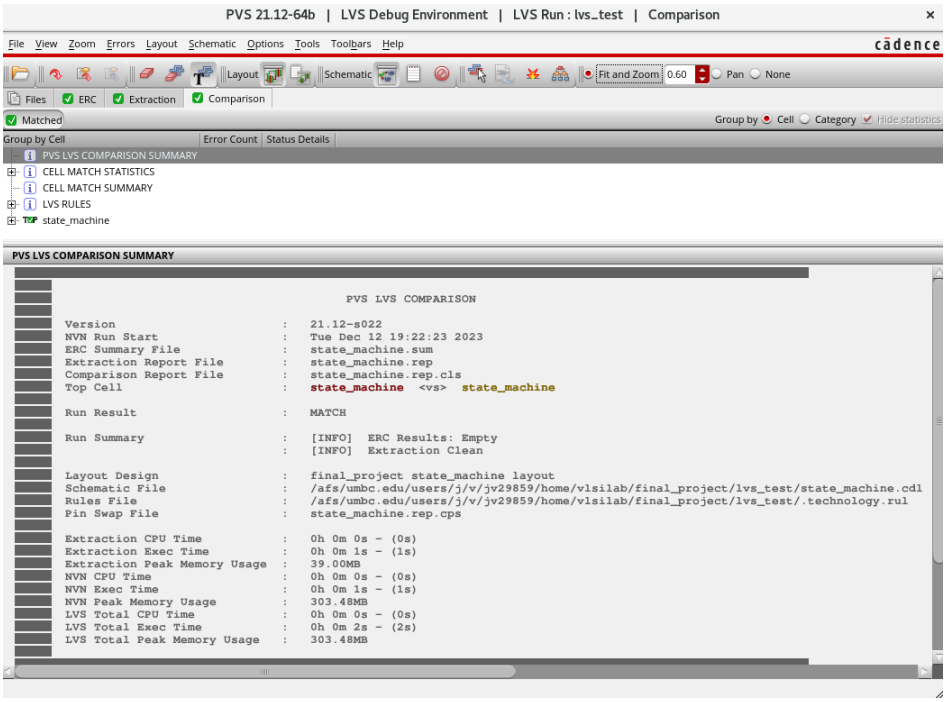

Fig 3.38: LVS report of the State machine

3.16 Chip

The components designed above are now integrating as one chip and written a VHDL code for overall cache design operations. This source code is written in the file “chip.vhd”. The chip is implemented using the already designed components Address register, Data register, statemachine, register cell, cache, cache decoder, hit/miss checker, output enable, and a multiplexer. This code file is tested with the given test bench i.e., chip_test.vhd and with the inputs in input file “chip_in.txt”. After the simulation, we obtained the waveform for all the cache operations as shown in Fig 3.39.

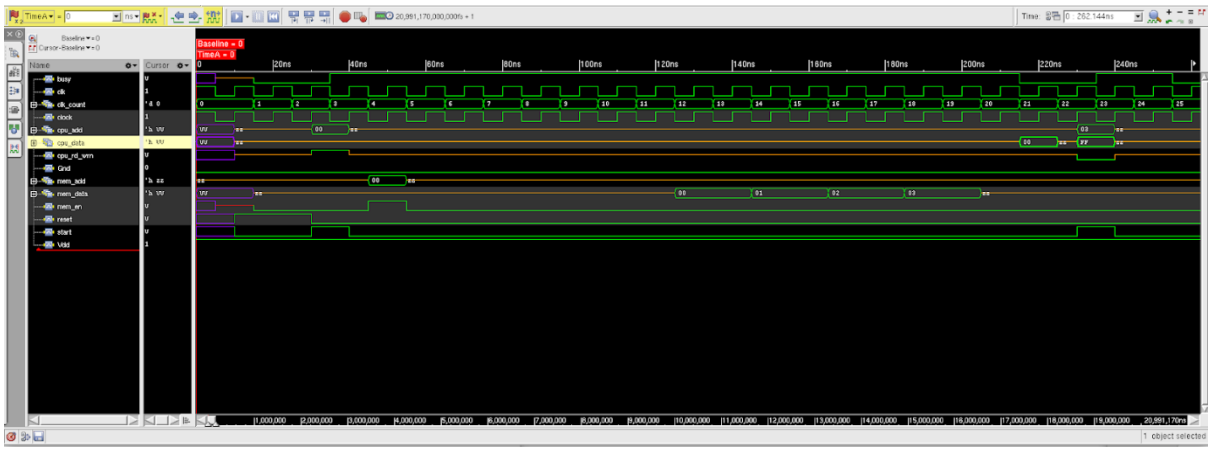

Fig 3.39: Waveform represents the four cache operations

In the above waveform, the read miss operation represents from third clock cycle to 21st clock cycle i.e., 19 clock cycles for read miss operation. When there is a read miss, the data is written into the memory data after 8 clock cycles. The write hit operation represents from 23rd clock cycle to 25th clock cycle. Here, the write operation is done at specified address. The read hit operation represents from 27th clock cycle to 28th clock cycle. Here, the data will read from the previously data written address. The write miss operation represents from 30th clock cycle to 32nd clock cycle. Here, no data will be written in the memory data as it is a miss case.

The Chip VHDL code is written in file “chip.vhd”. The VHDL code is imported into a library named final_project in cadence. Now the schematic is extracted for the respective VHDL functionality as shown in Fig 3.40.

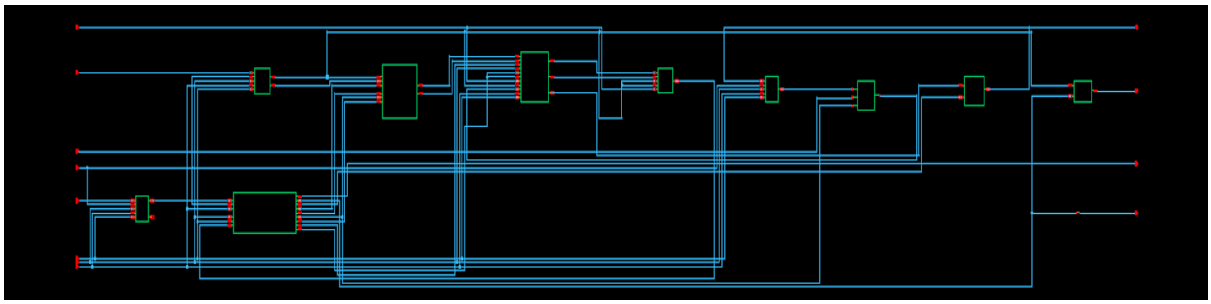

Fig 3.40: Schematic of the chip

For the above schematic the layout is implemented as shown in below Fig 3.41. The layout is implemented and performed the DRC and LVS. The LVS report is obtained with the matched result as shown in Fig 3.42.

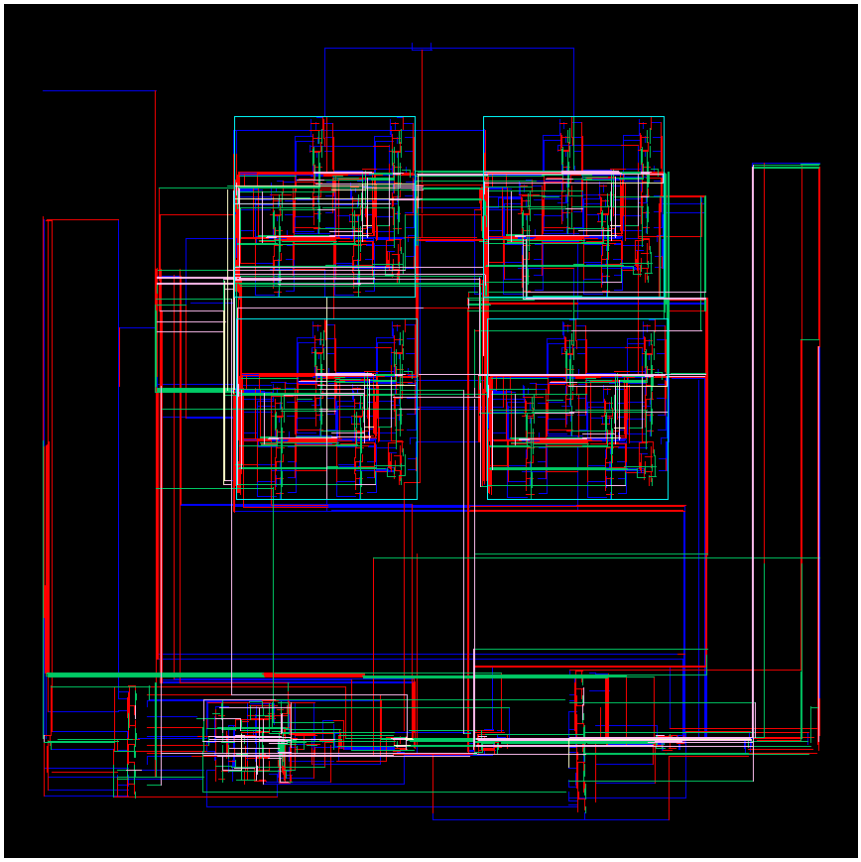

Fig 3.41: Layout of the chip

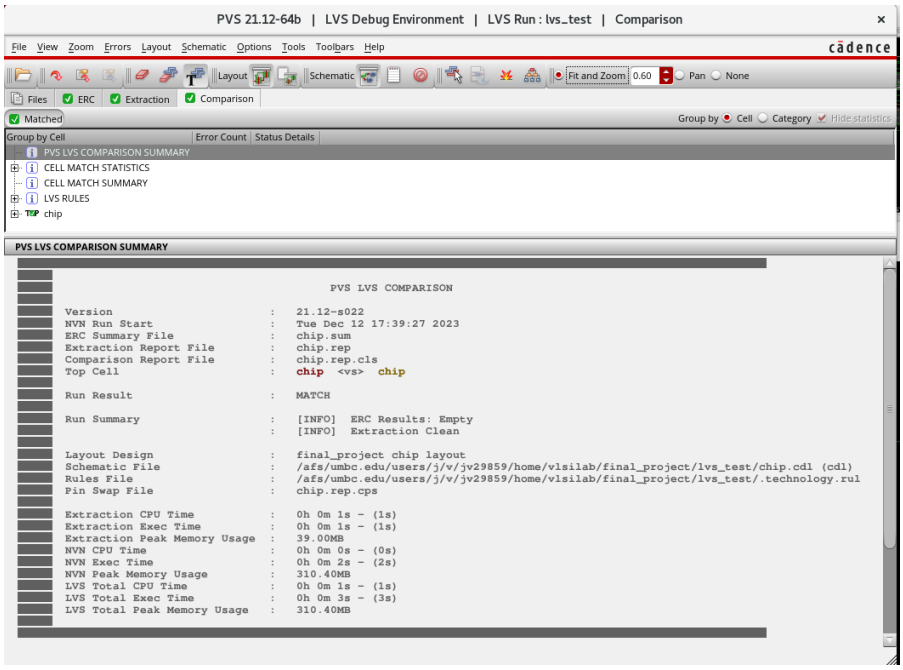

Fig 3.42: LVS report of the chip

The designed chip area is calculated by measuring the length and width of the chip using ruler. The length is measured as 492.025 µm, and the width is measured as 492.175 µm.

The area is calculated by multiplying length and width of the chip and obtained as 0.242nm^2 (242µm^2).

ARCHITECTURE OF CACHE

The designed cache architecture is shown in below Fig 4.1. It consists of Register cells for address registers and data regsiters, 16 byte cache, cache decoders, Hit/ Miss checker, state machine. All combined and connected as follows gives us the final chip which performs the four operations of cache.

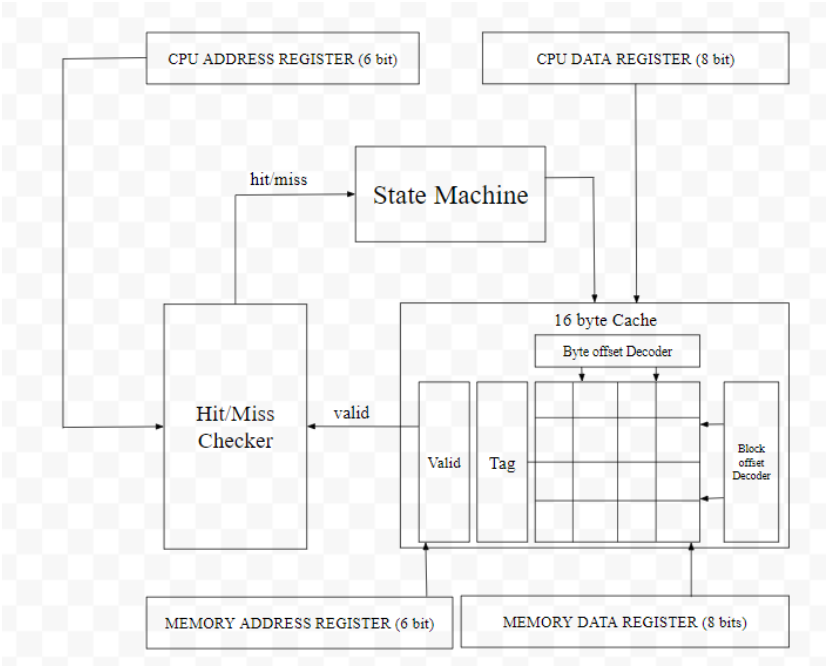

Fig 4.1: 16- byte Cache Architecture

CONCLUSION

In the project, the direct mapped cache is designed by implementing its operations of Read Miss, Write Hit, Read Hit, and Write miss in VHDL code. The written code files is simulated and obtained the waveform which represent the four cache operations. The VHDL codes are imported into the cadence tool into the library “final_project”. From the written codes, the schematics are implemented for basic logic gates, and exported for the high-level circuits. For all the schematic cells, the layouts are designed and attached in the zip file. For all the layouts the DRC and LVS check is performed and observed the reports. Finally, the chip of cache design is implemented in VHDL, as a schematic, and as a layout successfully which performs all the four cache operations.