ECE 6346 VLSI DESIGN PROJECT

Design of a Full Adder and a Multiplexer to realize simple Arithmetic-Logic-Unit (ALU)

SUBMITTED BY

GOKUL BALAGOPAL 1590661

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING CULLEN COLLEGE OF ENGINEERING UNIVERSITY OF HOUSTON

Contents

1 OBJECTIVE

The objective of this project is to realize simple Arithmetic-Logic-Unit (ALU) using design a Full Adder and a Multiplexer. Cadence IC Design Tool with Ami06 technology was utilized to simulate the behaviour of the optimized schematic and layout for the counter design.

2 INTRODUCTION

2.1 ARITHMETIC-LOGIC-UNIT (ALU)

An **Arithmetic Logic Unit** (**ALU**) is a combinational digital circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code indicating the operation to be performed; the ALU's output is the result of the performed operation. In many designs, the ALU also has status inputs or outputs, or both, which convey information about a previous operation or the current operation, respectively, between the ALU and external status registers.

2.2 ALU OPERATION

An ALU is a combinational logic circuit, meaning that its outputs will change asynchronously in response to input changes. In normal operation, stable signals are applied to all of the ALU inputs and, when enough time (known as the "propagation delay") has passed for the signals to propagate through the ALU circuitry, the result of the ALU operation appears at the ALU outputs. The external circuitry connected to the ALU is responsible for ensuring the stability of ALU input signals throughout the operation. In general, external circuitry controls an ALU by applying signals to its inputs. Typically, the external circuitry employs sequential logic to control the ALU operation, which is paced by a clock signal of a sufficiently low frequency to ensure enough time for the ALU outputs to settle under worst-case conditions.

2.3 ALU FUNCTIONS

Several basic arithmetic and bitwise logic functions are commonly supported by ALUs.

2.3.1 Arithmetic operations

Add: A and B are summed, and the sum appears at Y and carry-out.
Subtract: B is subtracted from A (or vice versa) and the difference appears at Y and carry-out.
Two's complement (negate): A (or B) is subtracted from zero and the difference appears at Y.
Increment: A (or B) is increased by one and the resulting value appears at Y.
Decrement: A (or B) is decreased by one and the resulting value appears at Y.
Pass through: all bits of A (or B) appear unmodified at Y. This operation is typically used to determine the parity of the operand or whether it is zero or negative, or to load the operand into a processor register.

2.3.2 Bitwise logical operations

AND: the bitwise AND of A and B appears at Y.

OR: the bitwise OR of A and B appears at Y.
Exclusive OR: the bitwise XOR of A and B appears at Y
One's Complement: all bits of A (or B) are inverted and appear at Y.

2.3.3 Bit shift operations

ALU shift operations cause operand A (or B) to shift left or right (depending on the opcode) and the shifted operand appears at Y. Simple ALUs typically can shift the operand by only one bit position, whereas more complex ALUs employ barrel shifters that allow them to shift the operand by an arbitrary number of bits in one operation.

Arithmetic Shift: The operand is treated as a two's complement integer, meaning that the most significant bit is a "sign" bit and is preserved.

Logical Shift : A logic zero is shifted into the operand. This is used to shift unsigned integers.

Rotate : The operand is treated as a circular buffer of bits so its least and most significant bits are effectively adjacent.

Rotate through Carry: Carry bit and operand are collectively treated as a circular buffer of bits.

3. DESIGN PROCEDURE:

The ALU as the name suggests provides for arithmetic and logical operations on the operands. The Basic Arithmetic of addition, subtraction, multiplication et.al and Logical functions like AND, OR, XOR are some of the functions of the ALU.

3.1 Bottom – Up Approach

In the Bottom Up approach of solving a problem, one identifies the least significant units (i.e. functions which cannot be broken down further functionally) and uses them as building blocks for the complex system.

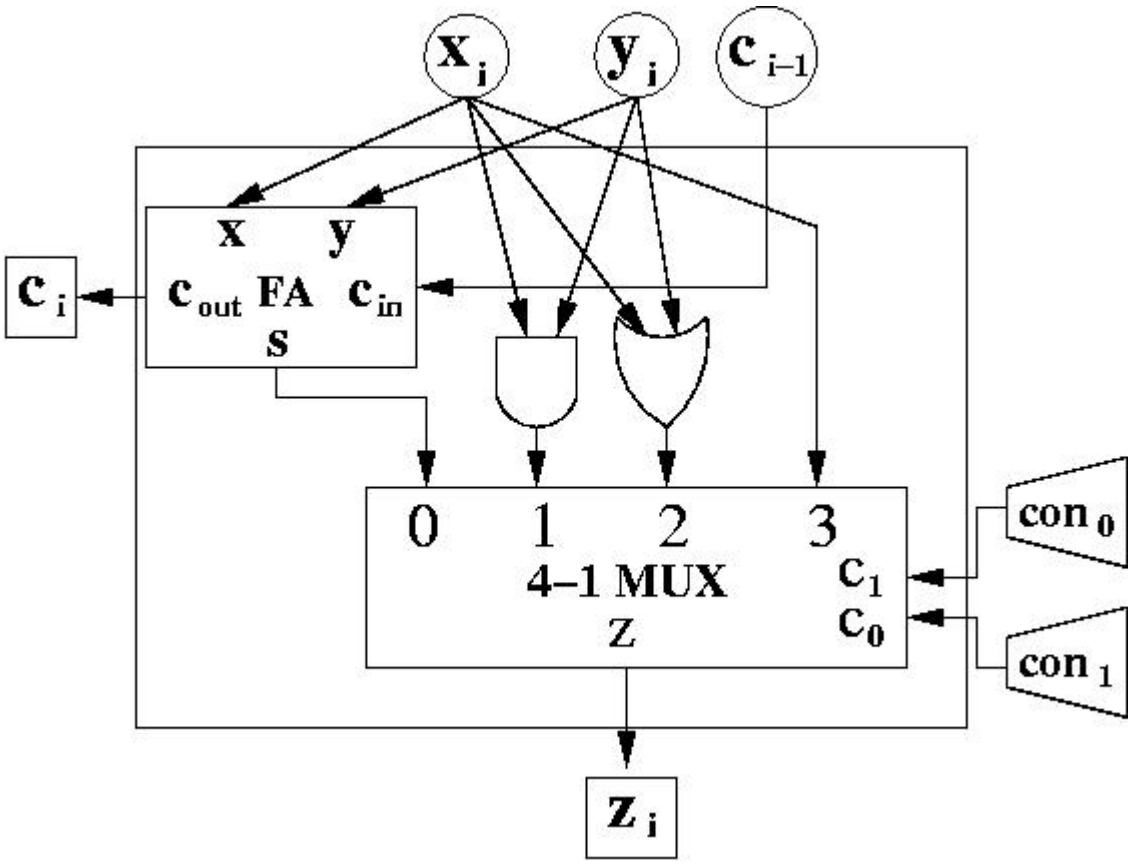
In our project we design an ALU, which performs the 1 bit unsigned addition, subtraction using 2's complement method are the fundamental arithmetic operations using which one can do most of the arithmetic functions and logical operations like AND and OR. The addition is performed by a full adder, subtraction is performed by a Full subtractor and Logical operations are performed by AND and OR gates. Multiplexers are combinational control elements that are used to control the flow of data in the circuit

3.2 Design Tips

1. It is good practice not to interconnect wires to form junctions even if we are sure that only one lead into the junction or node drives the node at any given time. Use logic gates for such inter-connects.
2. Use De-Morgans theorem to reduce NOR operations to their equivalent NAND operations for PMOS transistors in series in NOR make the gate area larger than the NMOS transistors in series in NAND.
3. Try simplifying circuits using K-Maps for simplified circuits save a lot of active components and leads to efficient chip real-estate.

3.3 Circuit Diagram:

Here's the circuit diagram of a one-bit ALU



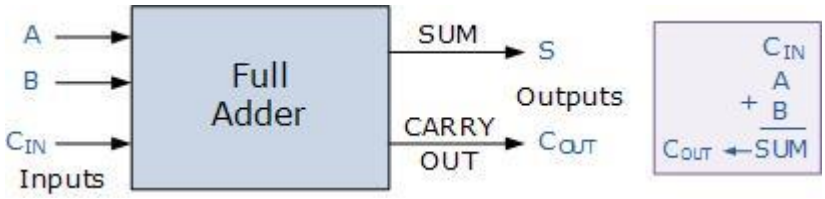
4. INTERNAL MODULES &
OPERATIONS 4.1 FULL ADDER

An **adder** is a digital circuit that performs addition of numbers. In many computers and other kinds of processors adders are used in the arithmetic logic units or ALU. They are also utilized in other parts of the processor, where they are used to calculate addresses, table indices, increment and decrement operators, and similar operations.

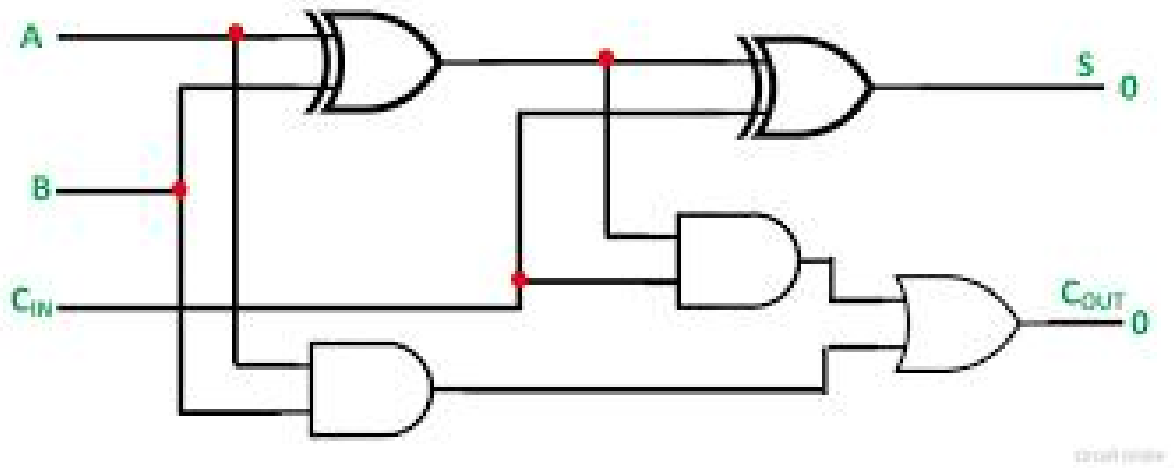
A **full adder** adds binary numbers and accounts for values carried in as well as out. A one-bit full-adder adds three one-bit numbers, often written as *A*, *B*, and C_{in}; *A* and *B* are the operands, and C_{in} is a bit carried in from the previous less-significant stage.The full adder is usually a component in a cascade of adders, which add 8, 16, 32, etc. bit binary numbers. The circuit produces outputs which are carry and sum typically represented by the signals C_{out} and *S*

A full adder can be implemented in many different ways such as with a custom transistor-level circuit or composed of other gates.

Symbol



Schematic



Schematic of Full adder

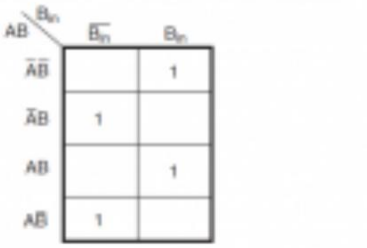
Truth Table

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

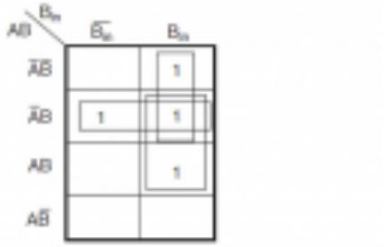
4.2 FULL SUBTRACTOR

The full subtractor is a combinational circuit which is used to perform subtraction of three input bits. In full subtractor '1' is borrowed by the previous adjacent lower minuend bit. Hence there are three bits are considered at the input of a full subtractor. There are two outputs, that are DIFFERENCE output D and BORROW output Bo. The K-maps for the two outputs are shown in figure. If we compare DIFFERENCE output D and BORROW output Bo with full adder the DIFFERENCE output D is the same as that for the SUM output. Further, the BORROW output Bo is similar to CARRY-OUT.

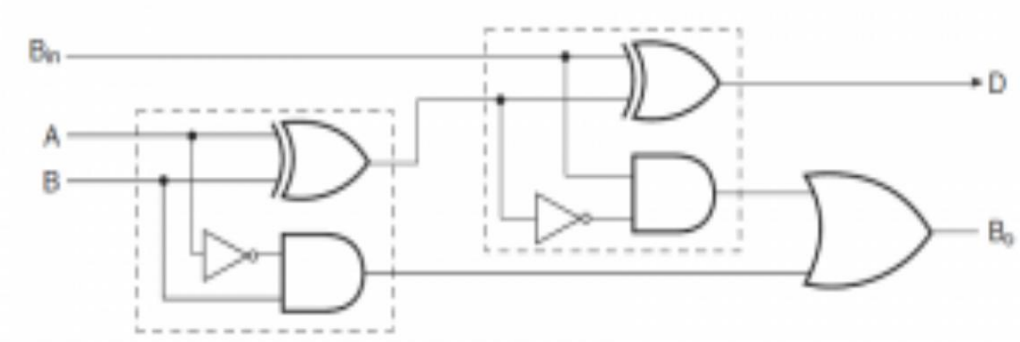
Minuend (A)	Subtrahend (B)	Borrow In (B_{in})	Difference (D)	Borrow Out (B_0)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



$$D = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + A.\overline{B}.\overline{B}_{in} + A.B.B_{in}$$



$$B_o = \overline{A}.\overline{B}.B_{in} + \overline{A}.B.\overline{B}_{in} + \overline{A}.B.B_{in} + A.B.B_{in}$$

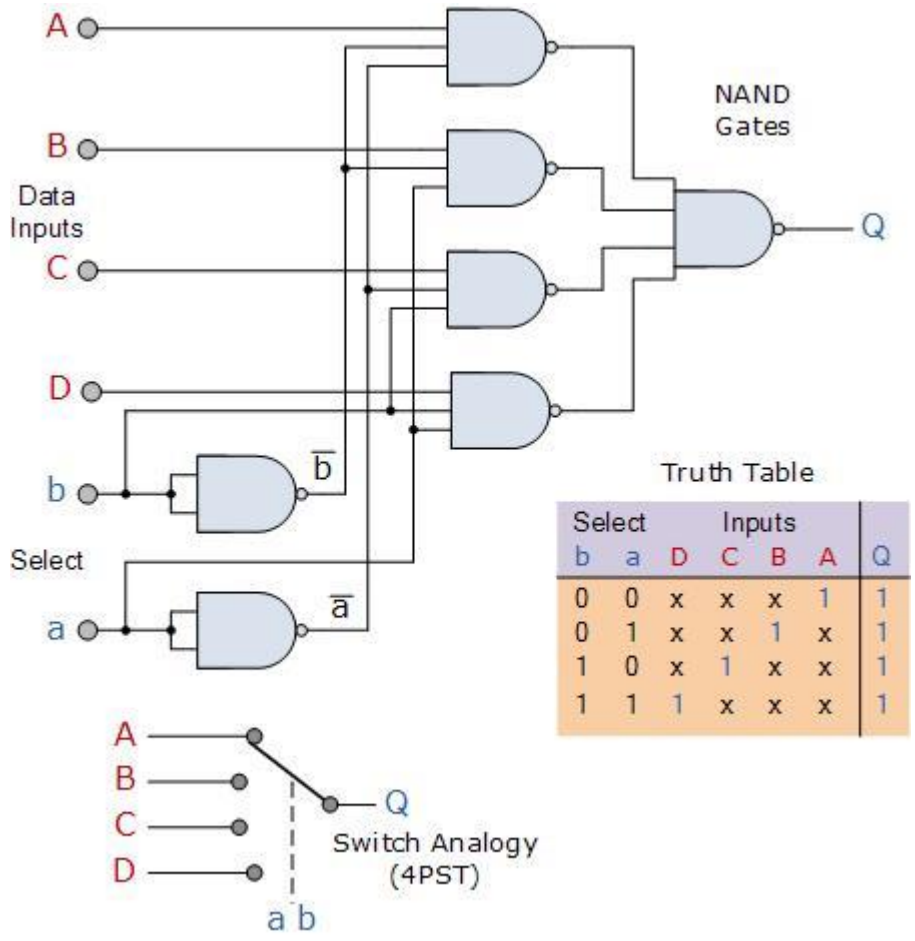


4.3 MULTIPLEXER

A **multiplexer** (or **mux**) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line.[1] A multiplexer of 2^n inputs has n select lines, which are used to select which input line to send to the output.[2] Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth.[1] A multiplexer is also called a **data selector**. Multiplexers can also be used to implement Boolean functions of multiple variables.

The Boolean expression for this 4-to-1 **Multiplexer** above with inputs A to D and data select lines a, b is given as:

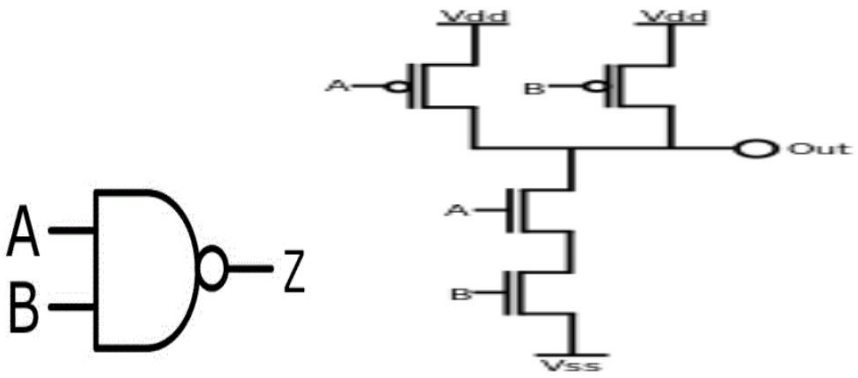
$$Q = abA + abB + abC + abD$$



4.4 BASIC GATES

4.4.1 NAND

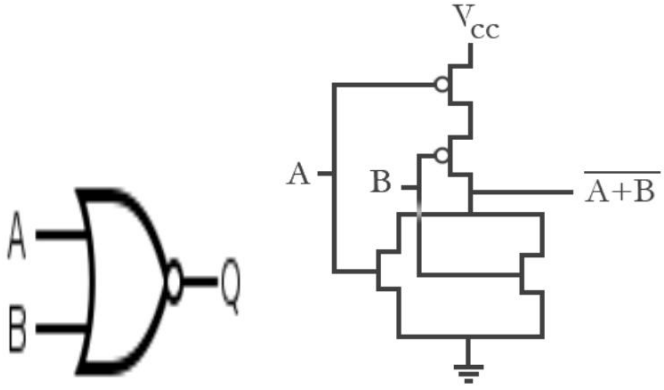
A **NAND gate** (**negative-AND**) is a logic gate which produces an output which is false only if all its inputs are true; thus its output is complement to that of the AND gate. A LOW (0) output results only if both the inputs to the gate are HIGH (1); if one or both inputs are LOW (0), a HIGH (1) output results. The NAND gate is significant because any boolean function can be implemented by using a combination of NAND gates. This property is called functional completeness



INPUT		OUTPUT
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

4.4.2 NOR

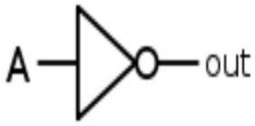
The **NOR gate** is a digital logic gate that implements logical NOR - it behaves according to the truth table to the right. A HIGH output (1) results if both the inputs to the gate are LOW (0); if one or both input is HIGH (1), a LOW output (0) results. NOR is the result of the negation of the OR operator. It can also be seen as an AND gate with all the inputs inverted. NOR is a functionally complete operation—NOR gates can be combined to generate any other logical function.

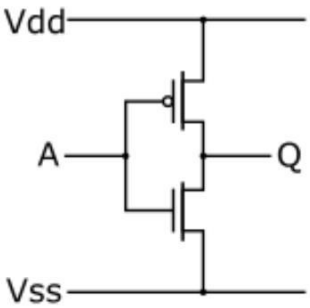


INPUT		OUTPUT
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

4.4.3 NOT

In digital logic, an **inverter** or **NOT gate** is a logic gate which implements logical negation. An inverter circuit outputs a voltage representing the opposite logic-level to its input. Its main function is to invert the input signal applied. If the applied input is low then the output becomes high and vice versa. Inverters can be constructed using a single NMOS transistor or a single PMOS transistor coupled with a resistor. Since this 'resistive-drain' approach uses only a single type of transistor, it can be fabricated at low cost. However, because current flows through the resistor in one of the two states, the resistive-drain configuration is disadvantaged for power consumption and processing speed. Alternatively, inverters can be constructed using two complementary transistors in a CMOS configuration. This configuration greatly reduces power consumption since one of the transistors is always off in both logic states.

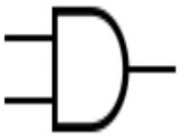


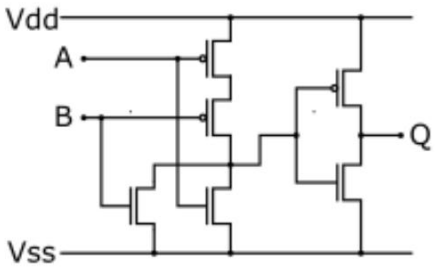


NPUT	OUTPUT
A	NOT A
0	1
1	0

4.4.4 AND

The **AND gate** is a basic digital logic gate that implements logical conjunction - it behaves according to the truth table to the right. A HIGH output (1) results only if both the inputs to the AND gate are HIGH (1). If neither or only one input to the AND gate is HIGH, a LOW output results.

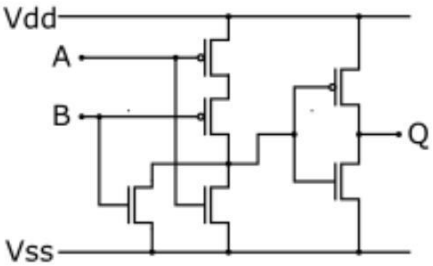




INPUT		OUTPUT
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

4.4.5 OR

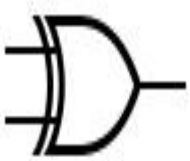
The **OR gate** is a digital logic gate that implements logical disjunction – it behaves according to the truth table to the right. A HIGH output (1) results if one or both the inputs to the gate are HIGH (1). If neither input is high, a LOW output (0) results. In another sense, the function of OR effectively finds the *maximum* between two binary digits, just as the complementary AND function finds the *minimum*.

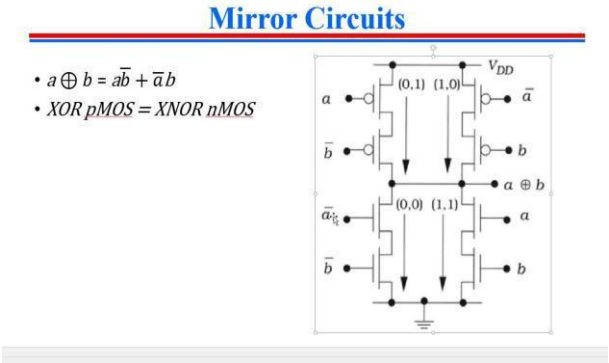


INPUT		OUTPUT
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

4.4.6 XOR

The **XOR gate** (sometimes **EOR gate**, or **EXOR gate** and pronounced as **Exclusive OR gate**) is a digital logic gate that gives a true (1/HIGH) output when the number of true inputs is odd. An XOR gate implements an exclusive or; that is, a true output results if one, and only one, of the inputs to the gate is true. If both inputs are false (0/LOW) or both are true, a false output results. XOR represents the inequality function, i.e., the output is true if the inputs are not alike otherwise the output is false. A way to remember XOR is "one or the other but not both".

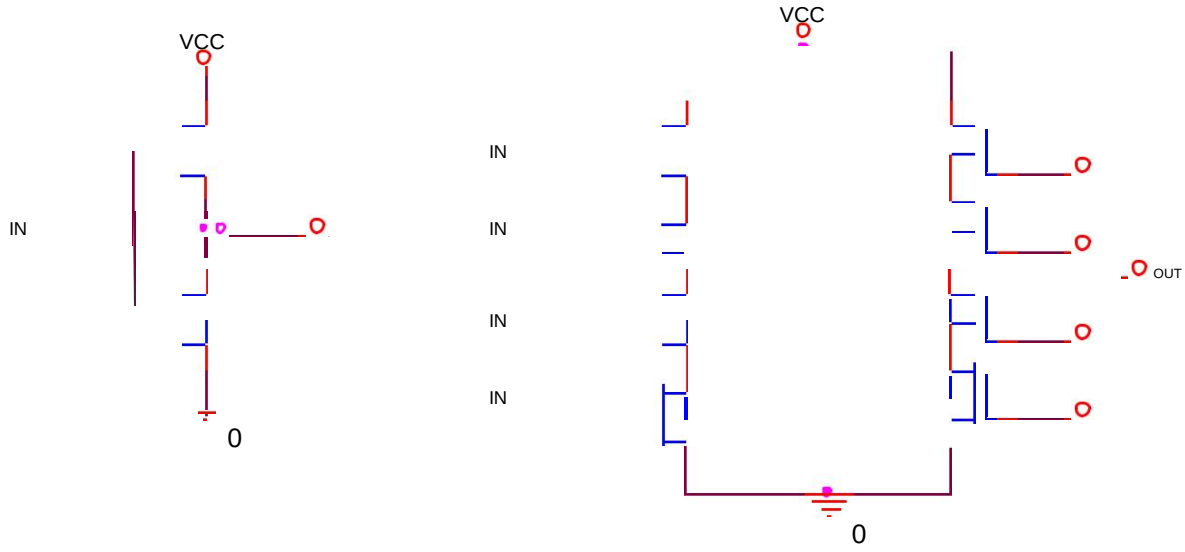




INPUT		OUTPUT
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

4.5 **MIRROR CIRCUITS**

Mirror circuits are based on series-parallel configurations of MOSFETs. A mirror circuit has the same transistor topology for the nFETs and the pFETs. NAND2, NOR2, XOR2,XNOR2 logic gates can be constructed using the same mirror circuit structure The different functionalities are implemented by varying the inputs at each gate Only one general layout is necessary.This simplifies the layout process. The mirror circuits can be used to minimize the number of transistors.

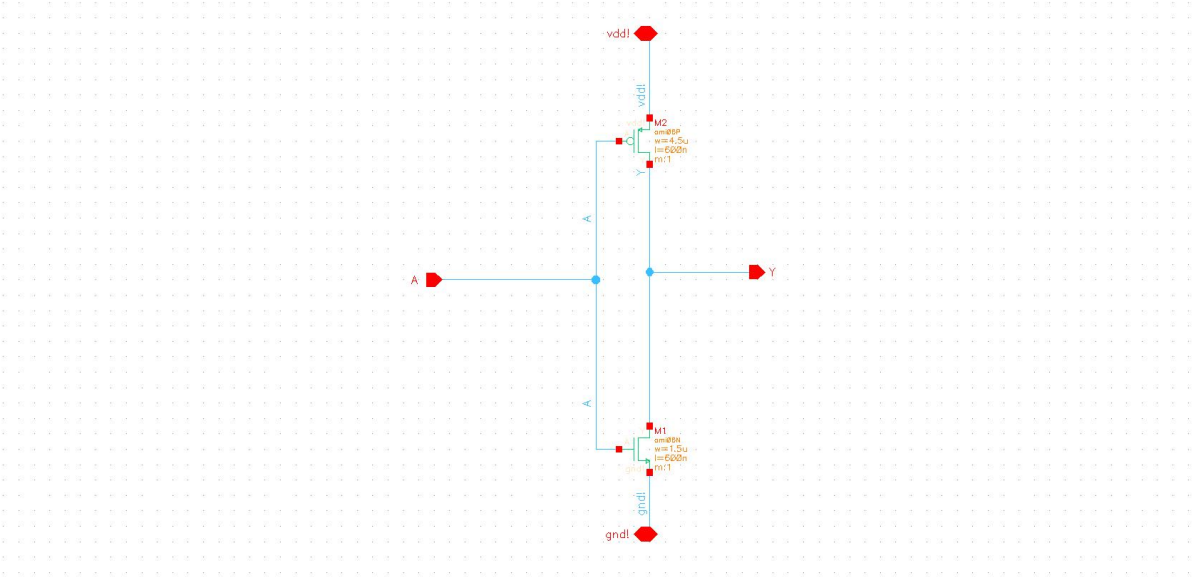


5.CADENCE IMPLEMENTATION

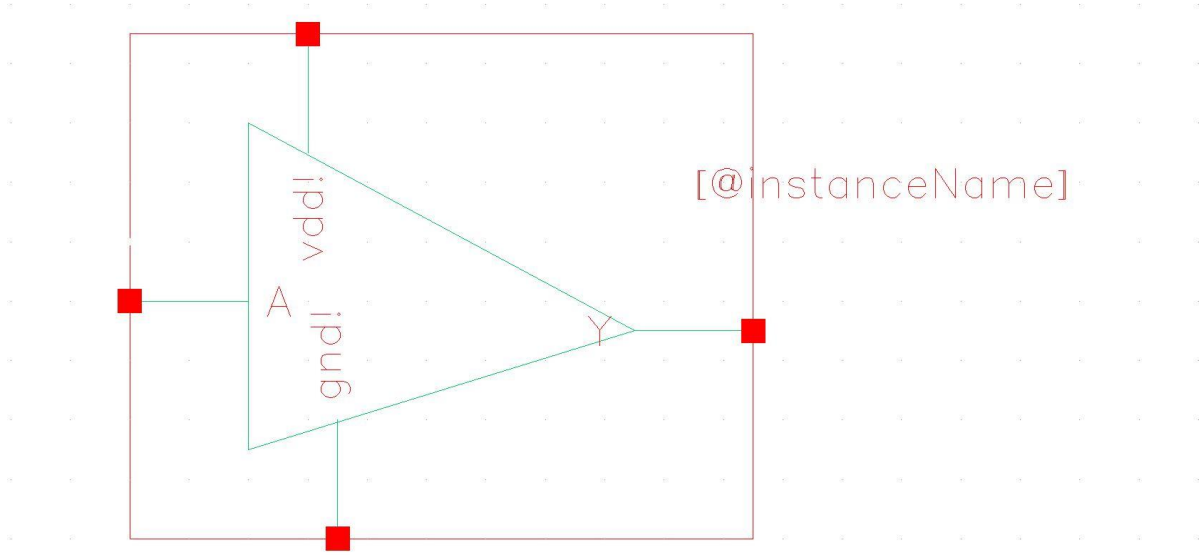
5.1 NOT GATE

The schematic, symbol, test bench, layout, and simulations of 2 input NAND gate are shown below.

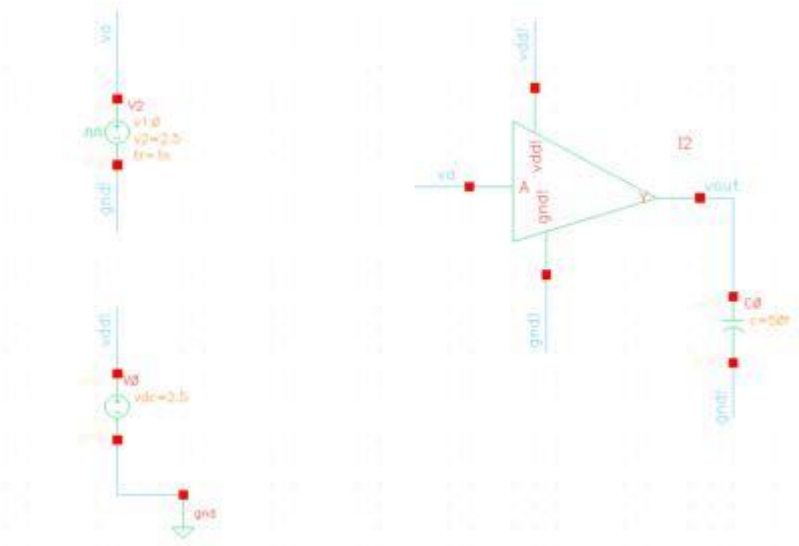
Schematic



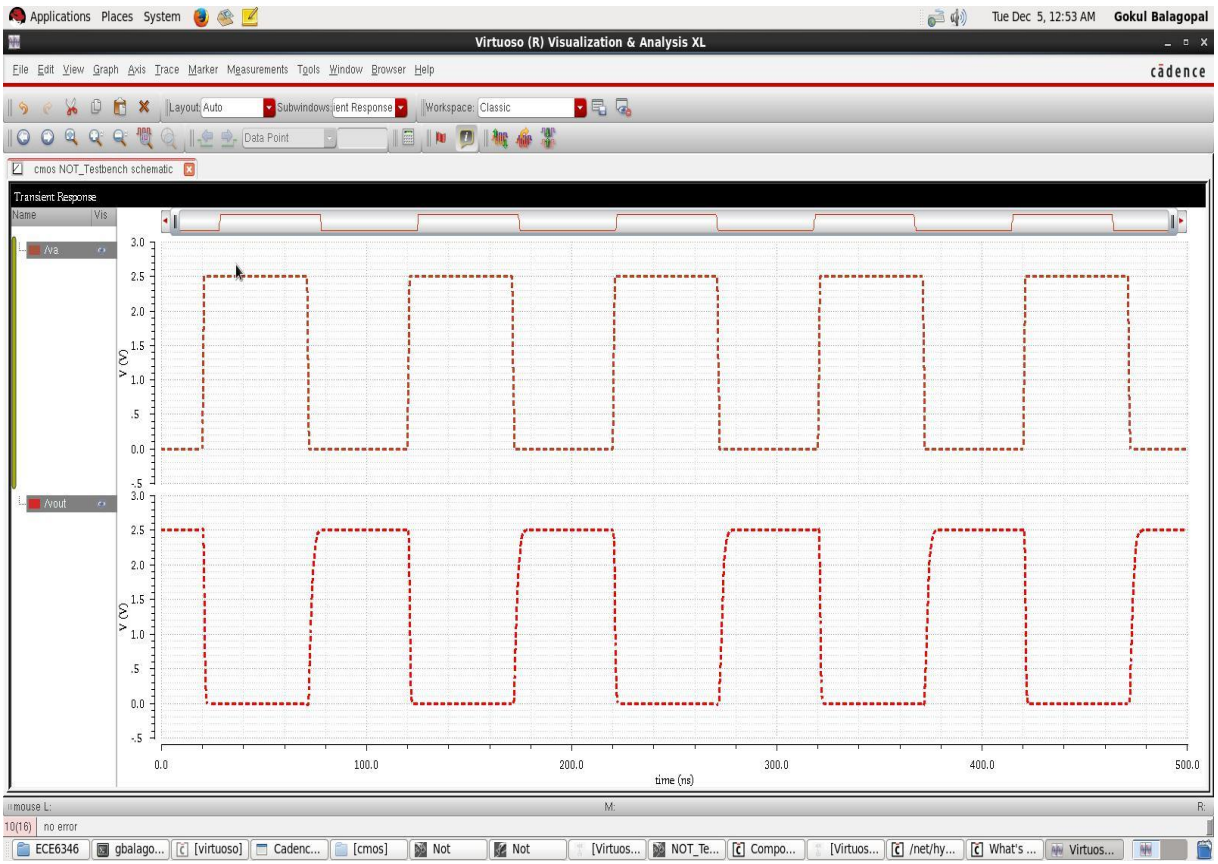
Symbol



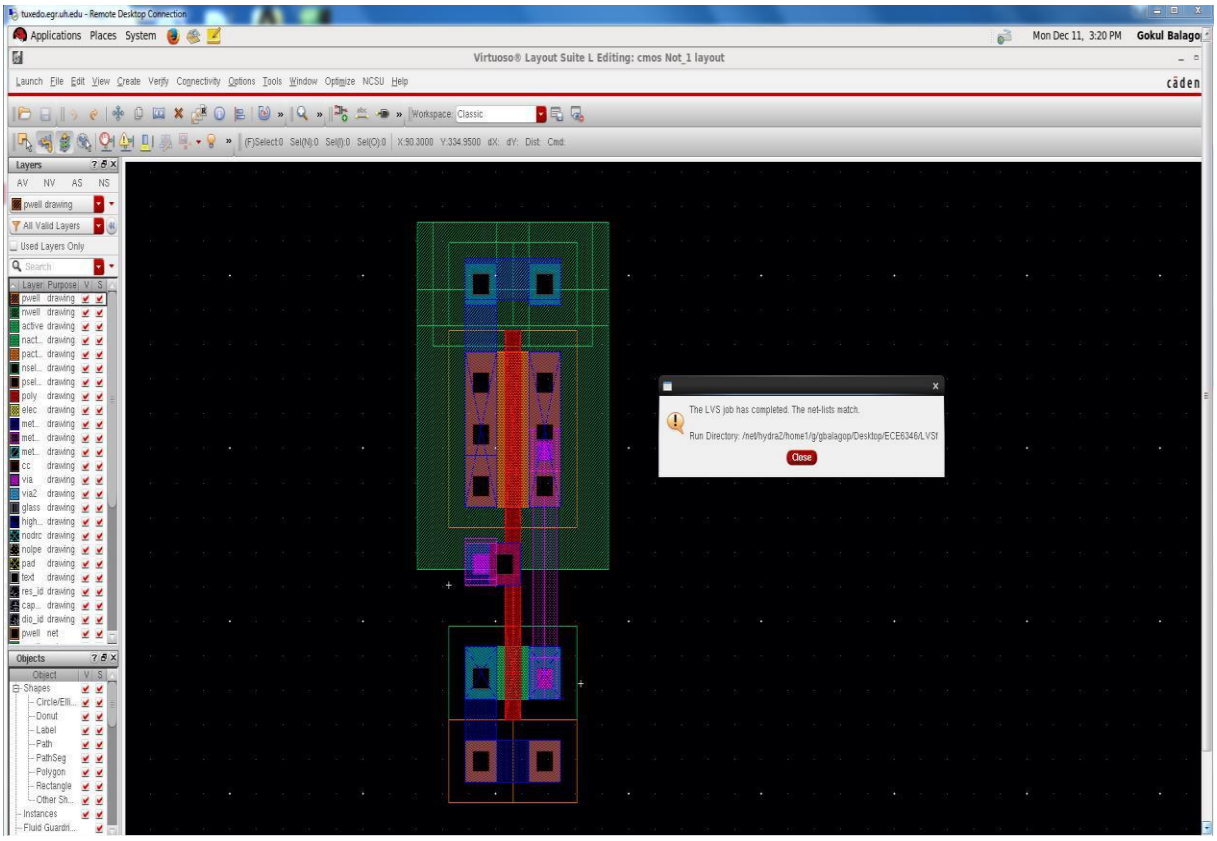
Testbench



Testbench Output wave form

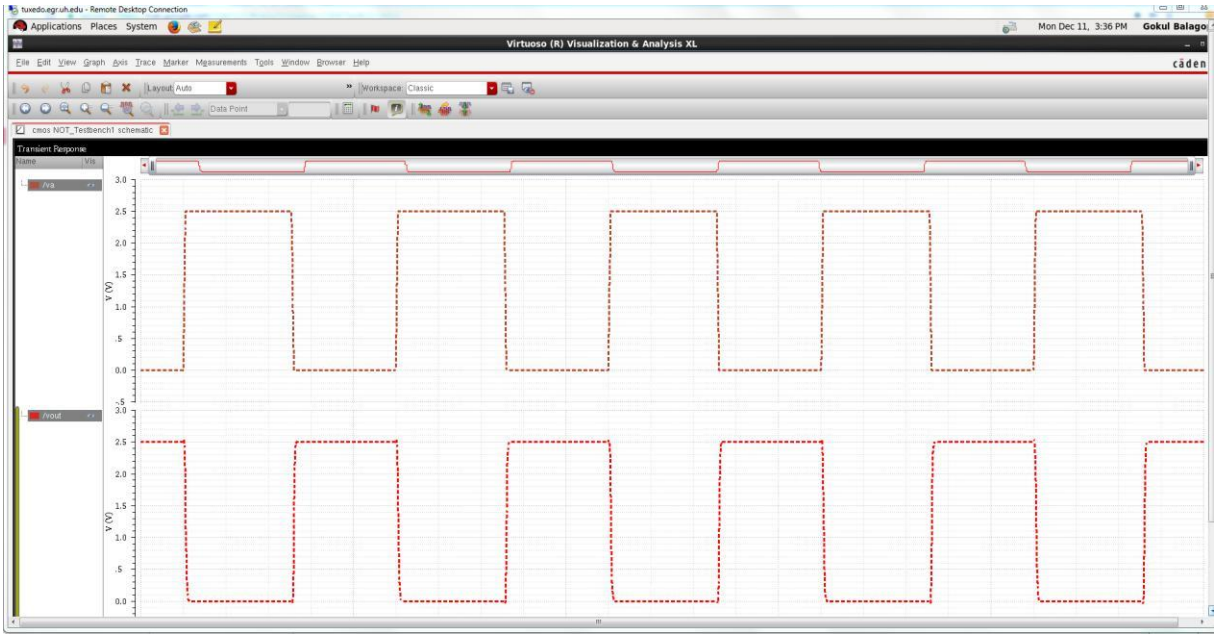


Layout & LVS



Parasitic Extracted

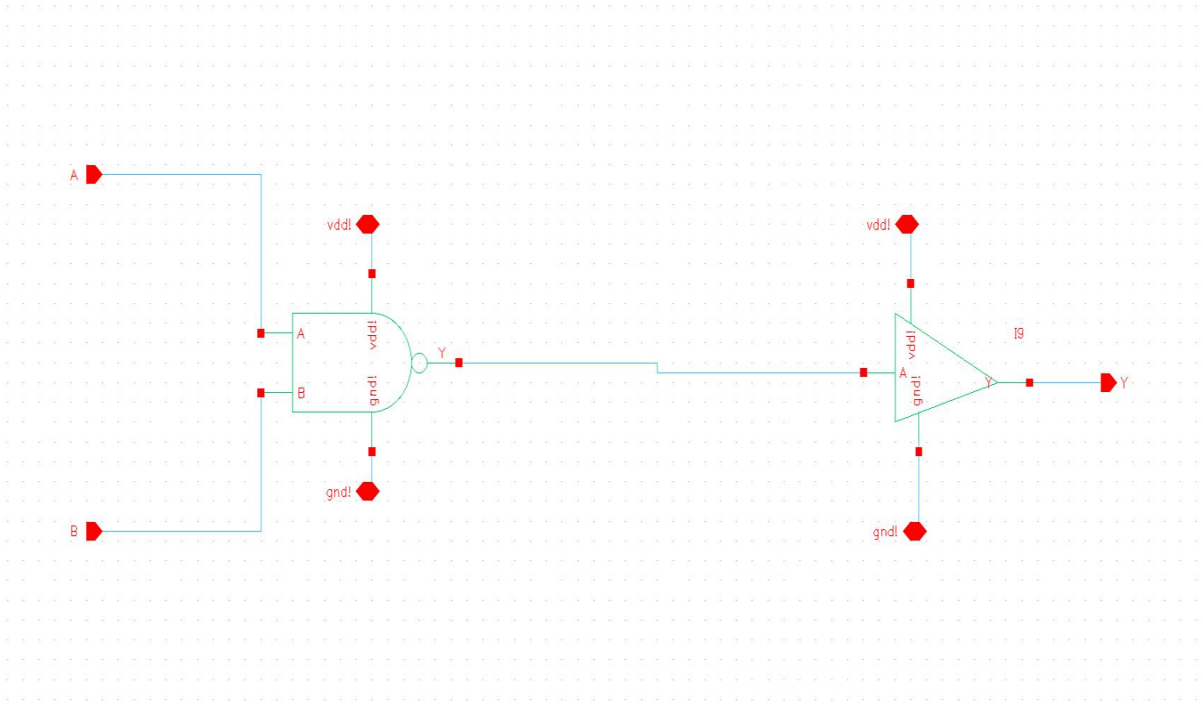
Final Output



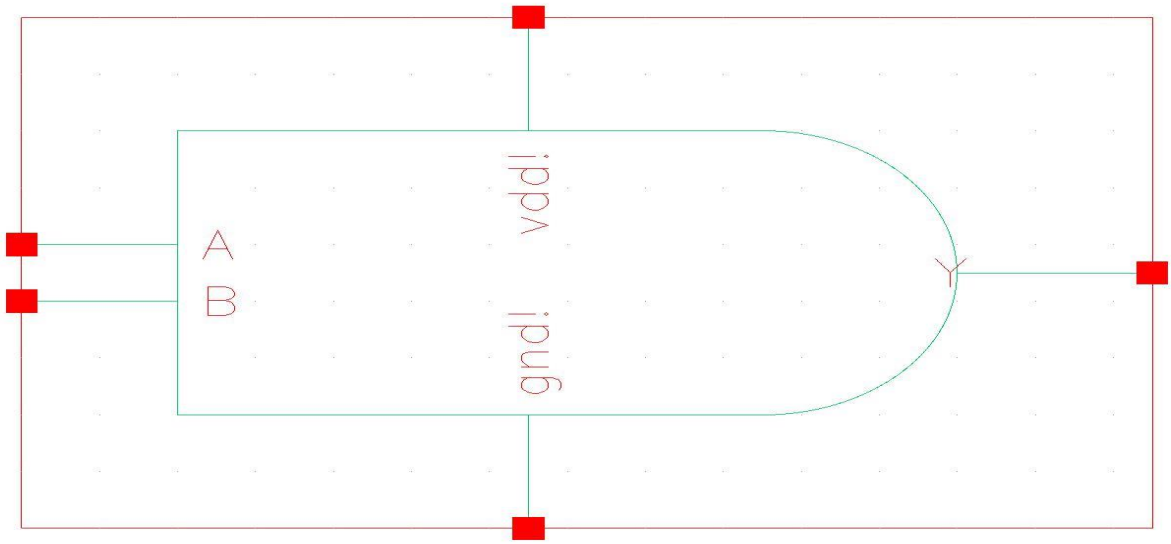
5.2 AND

The schematic, symbol, test bench, layout, and simulations of 2 input NAND gate are shown below

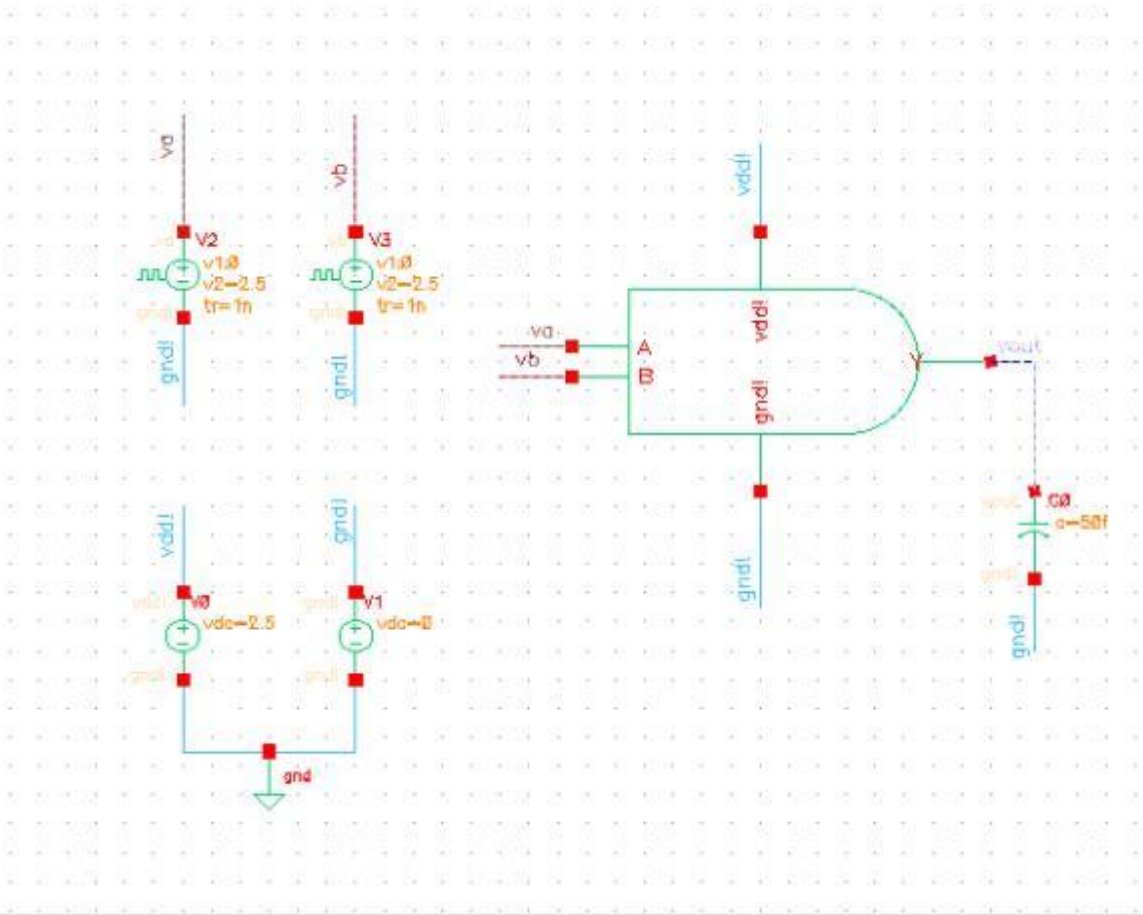
Schematic



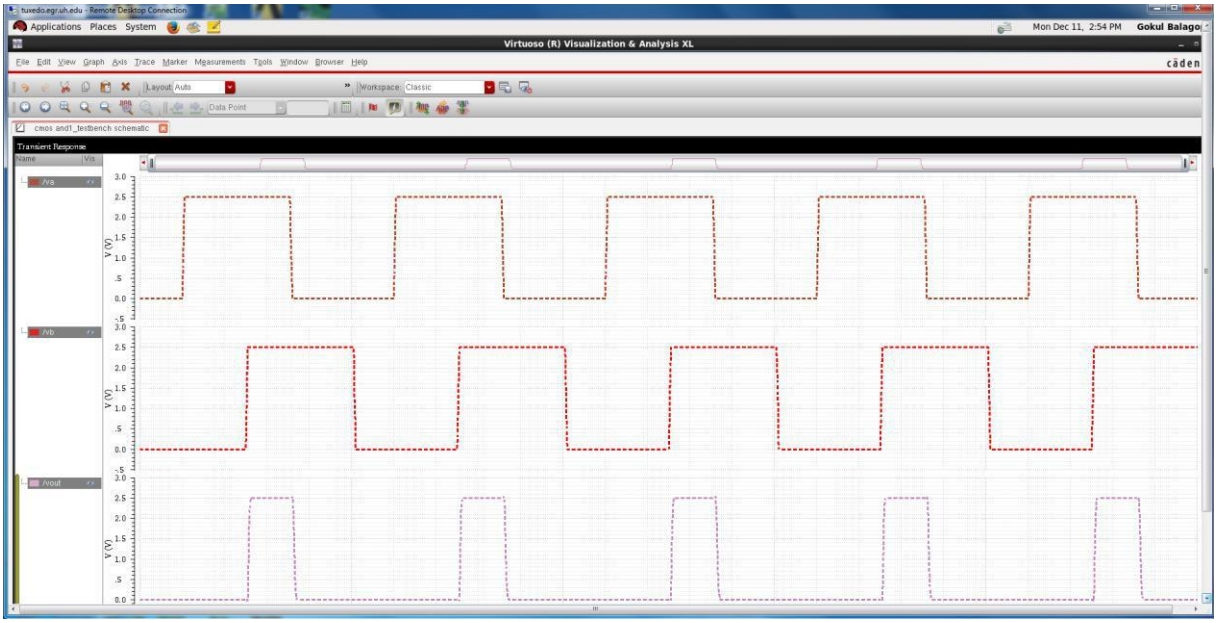
Symbol



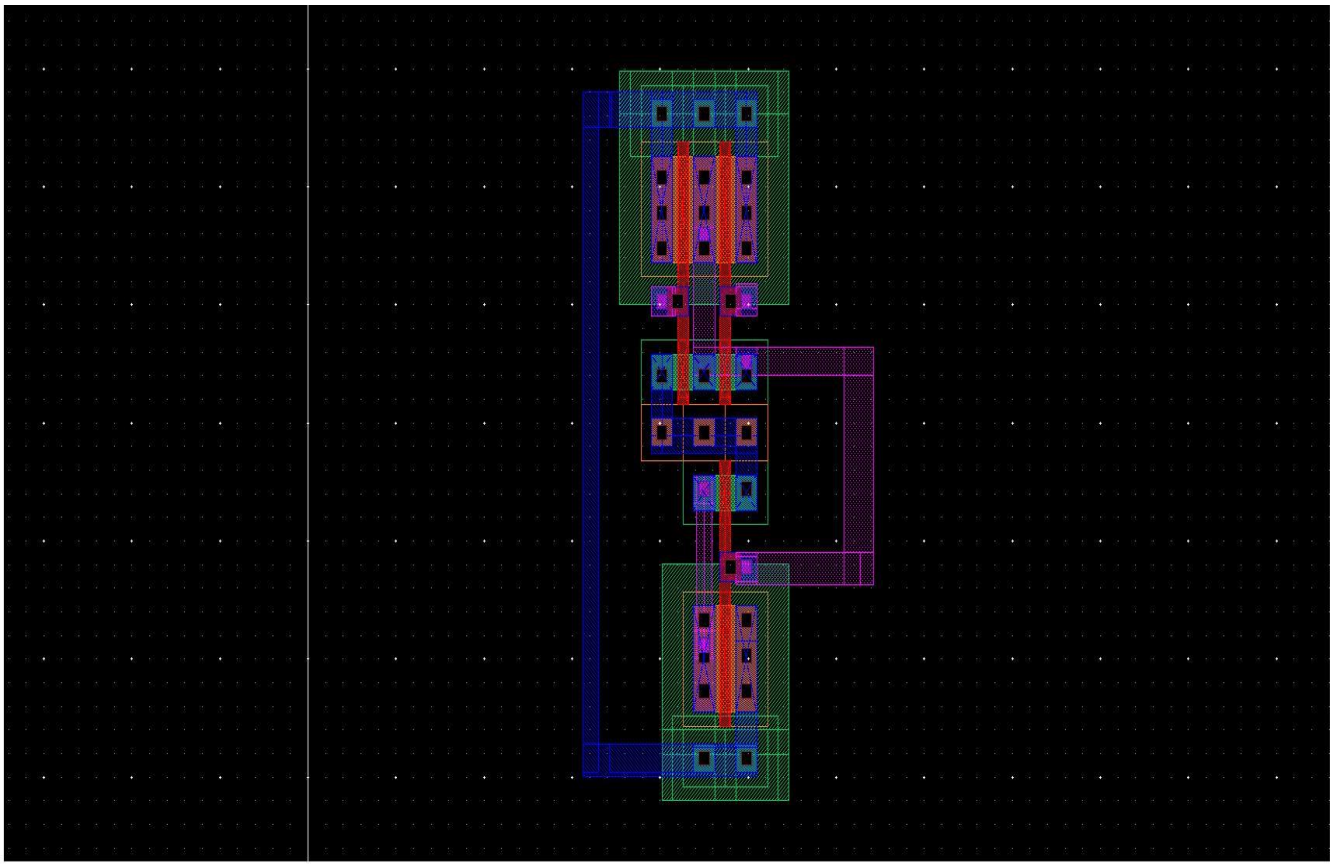
Testbench



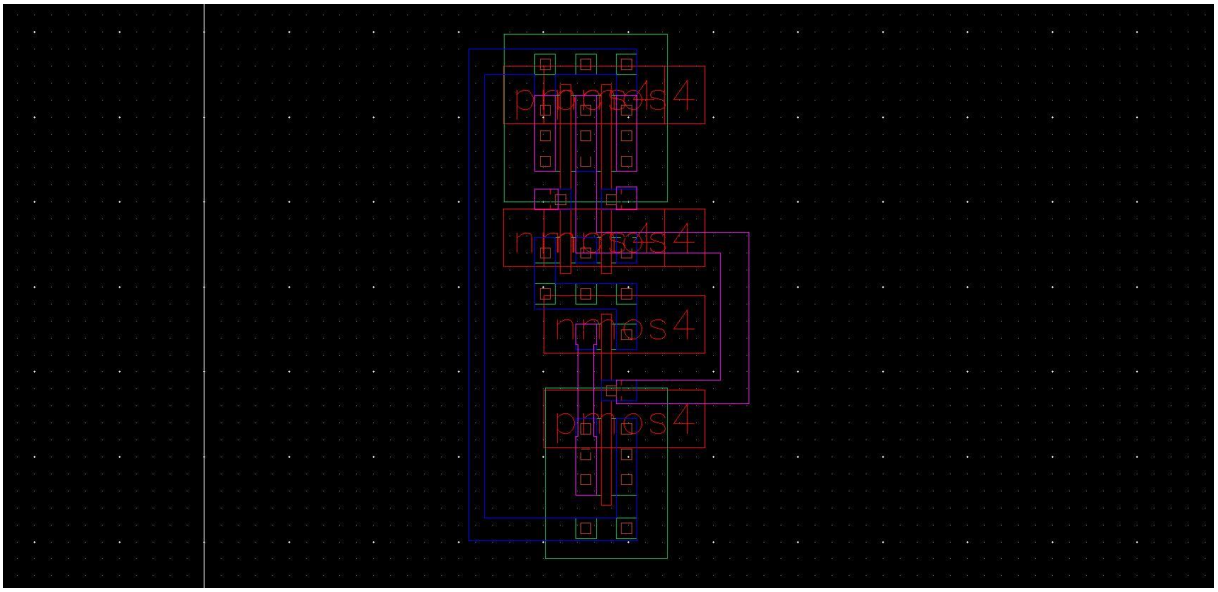
Testbench Output wave form



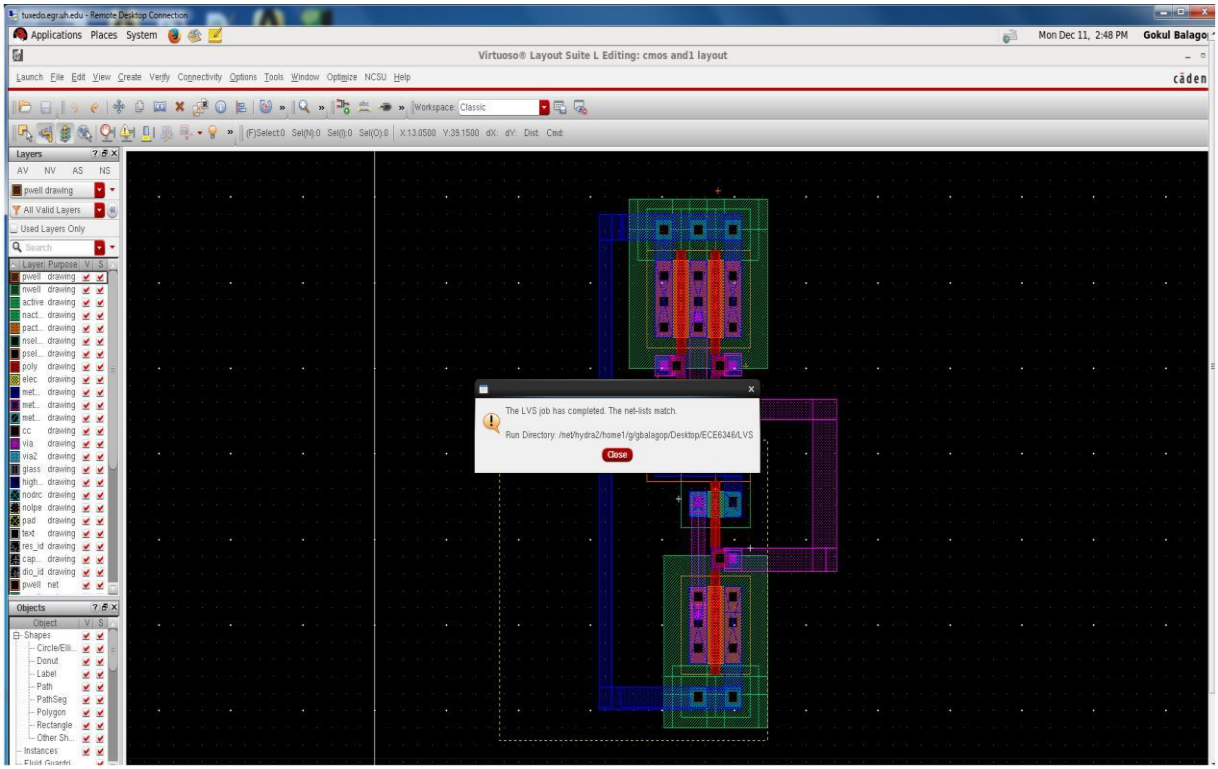
Layout



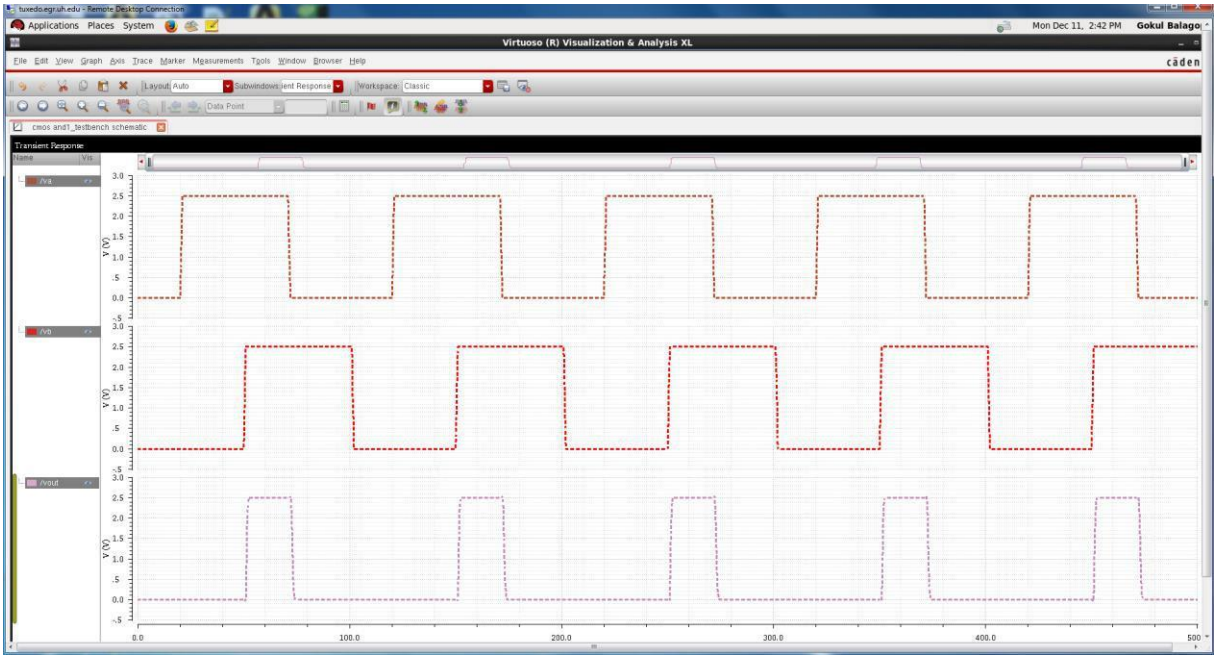
Parasitic Extracted



LVS



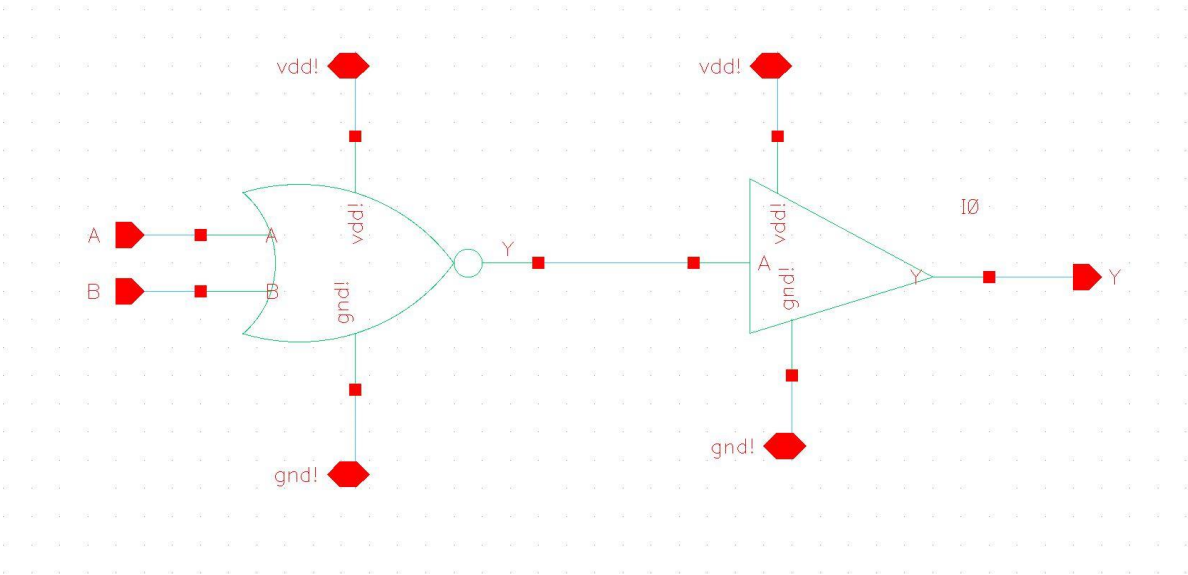
Final Output



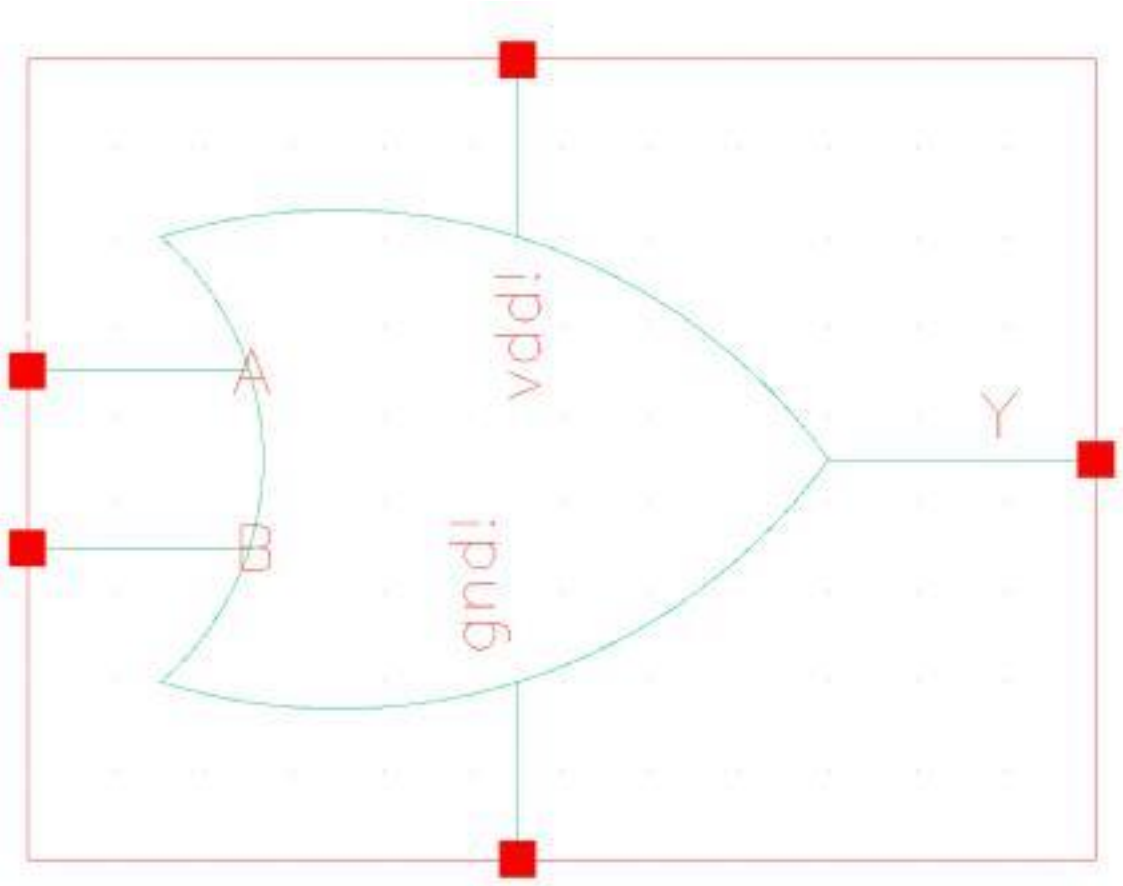
5.3 OR

Schematic

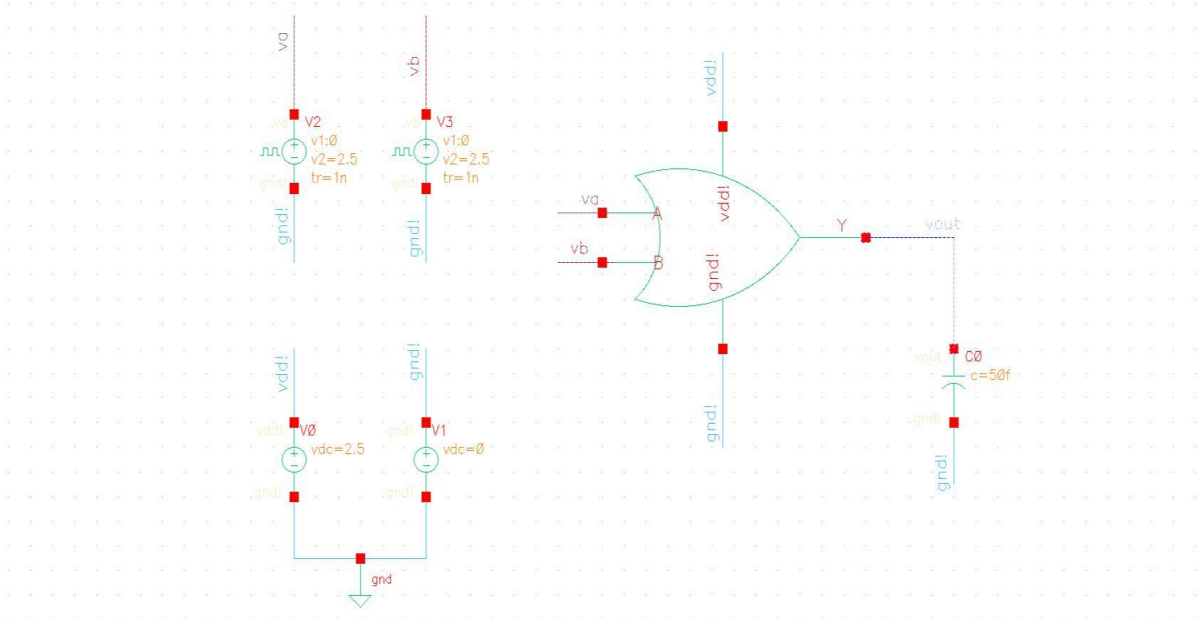
The schematic, symbol, test bench, layout, and simulations of 2 input NAND gate are shown below.



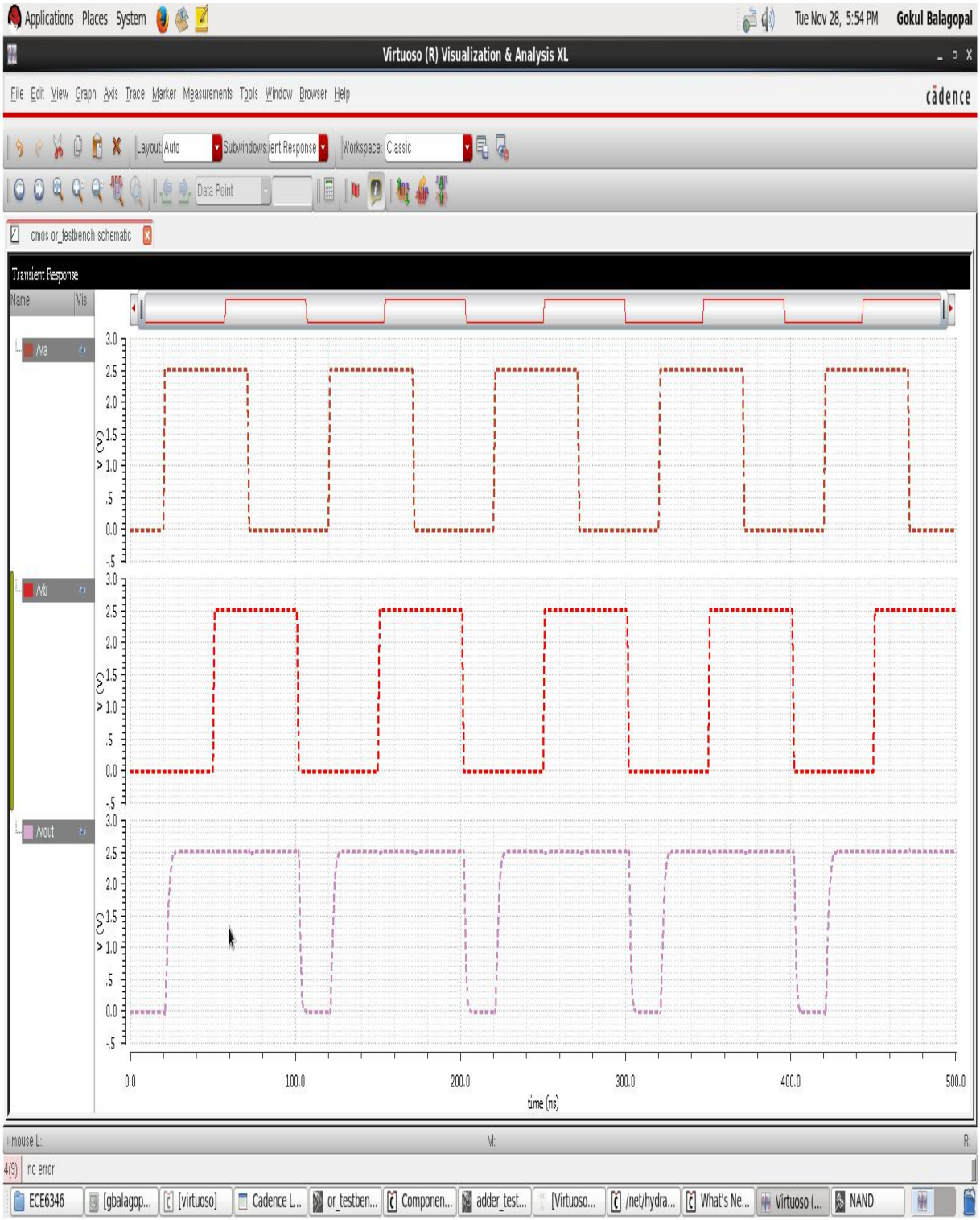
Symbol



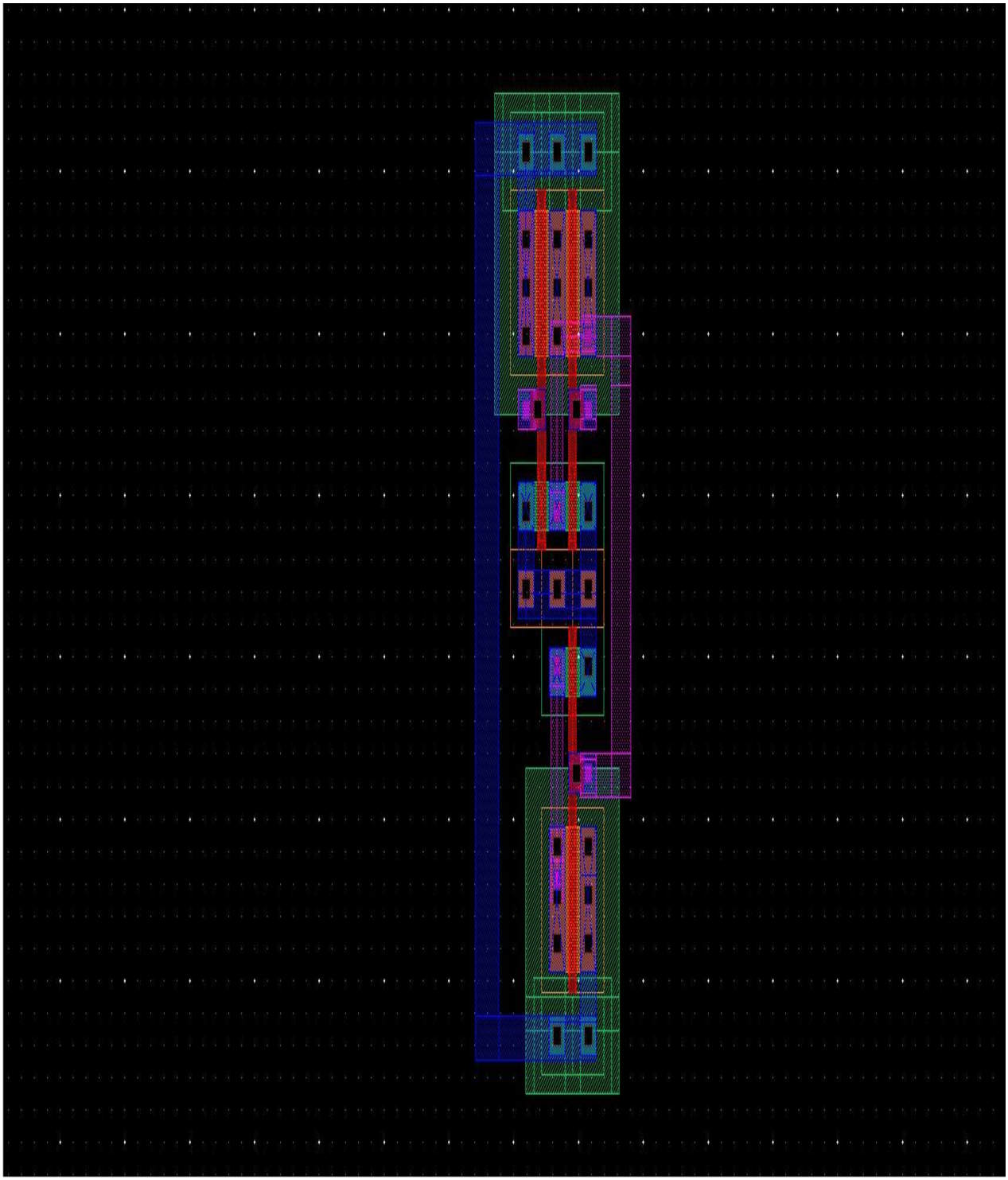
Testbench



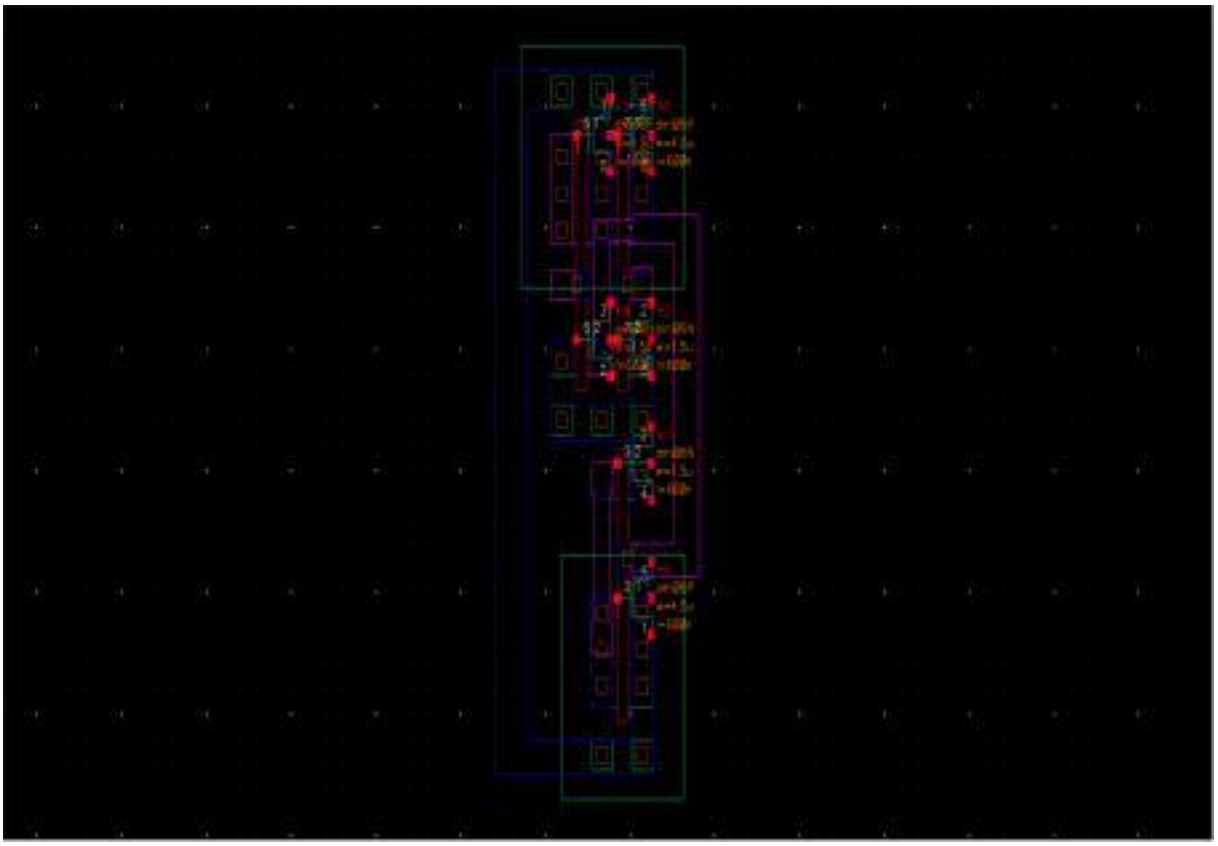
Testbench Output wave form



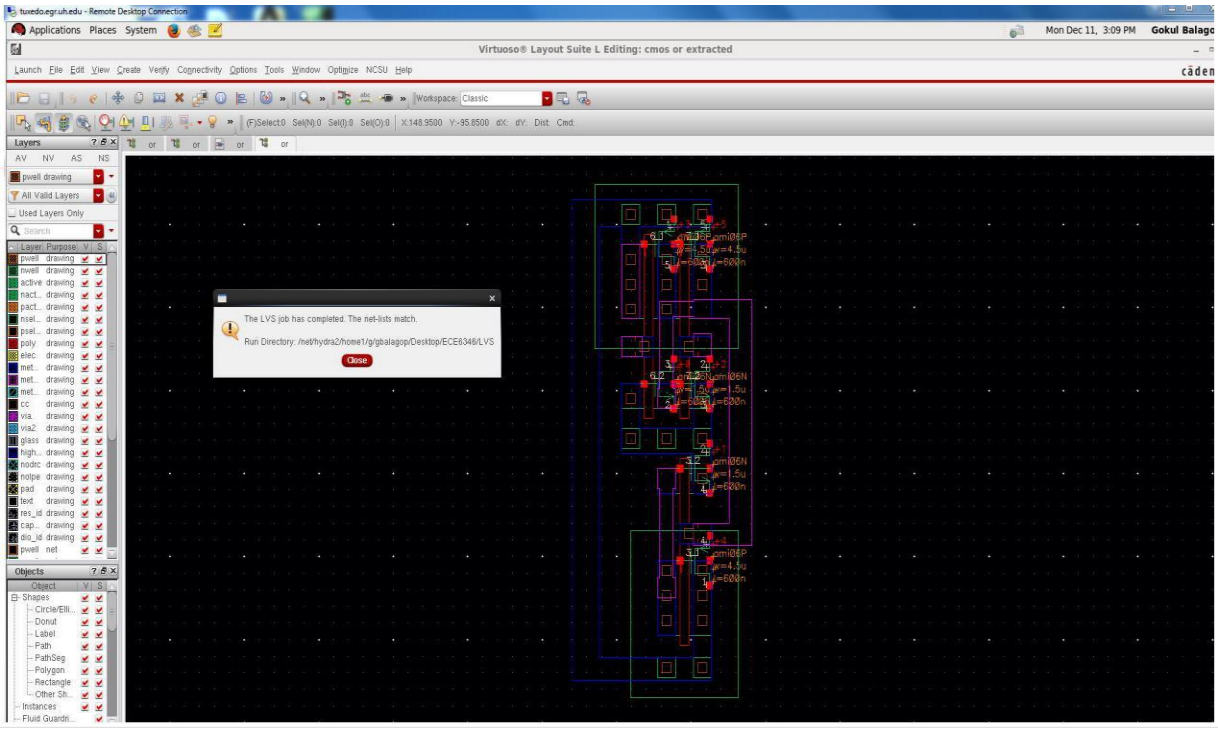
Layout



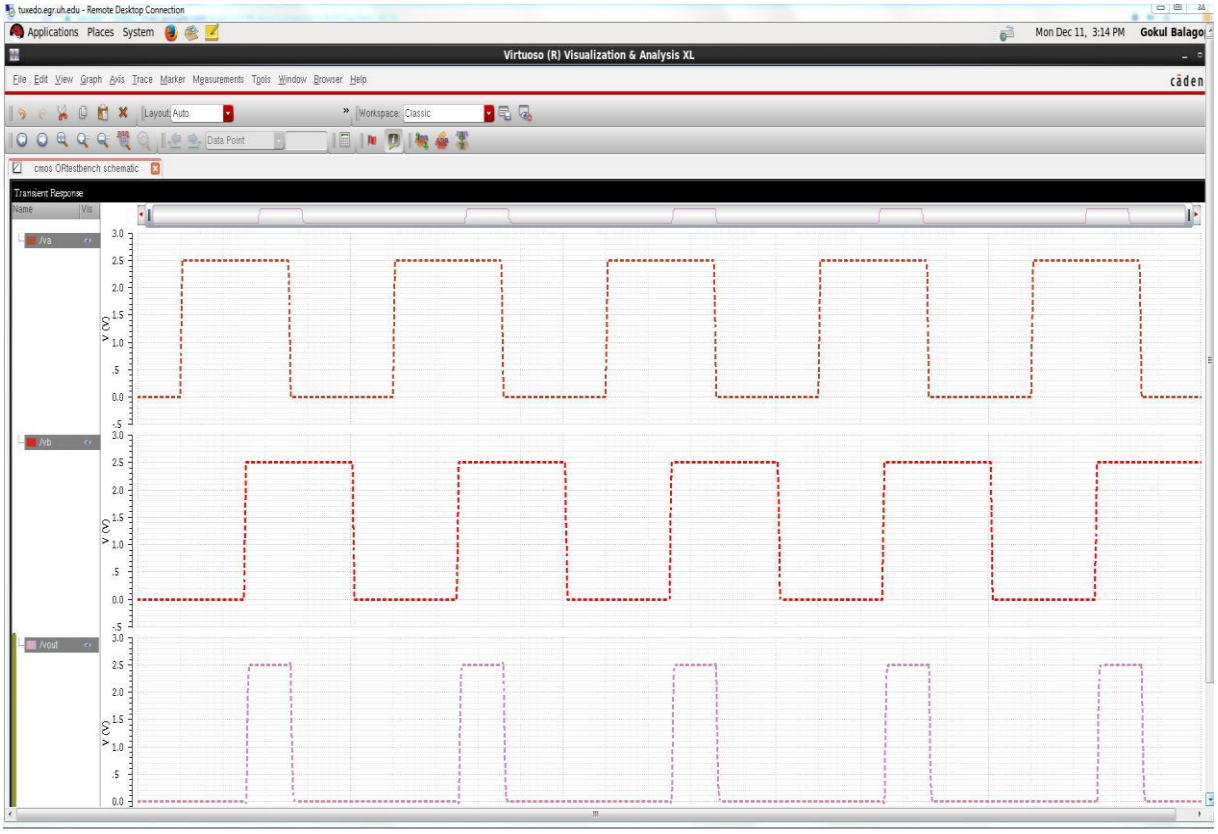
Parasitic Extracted



LVS



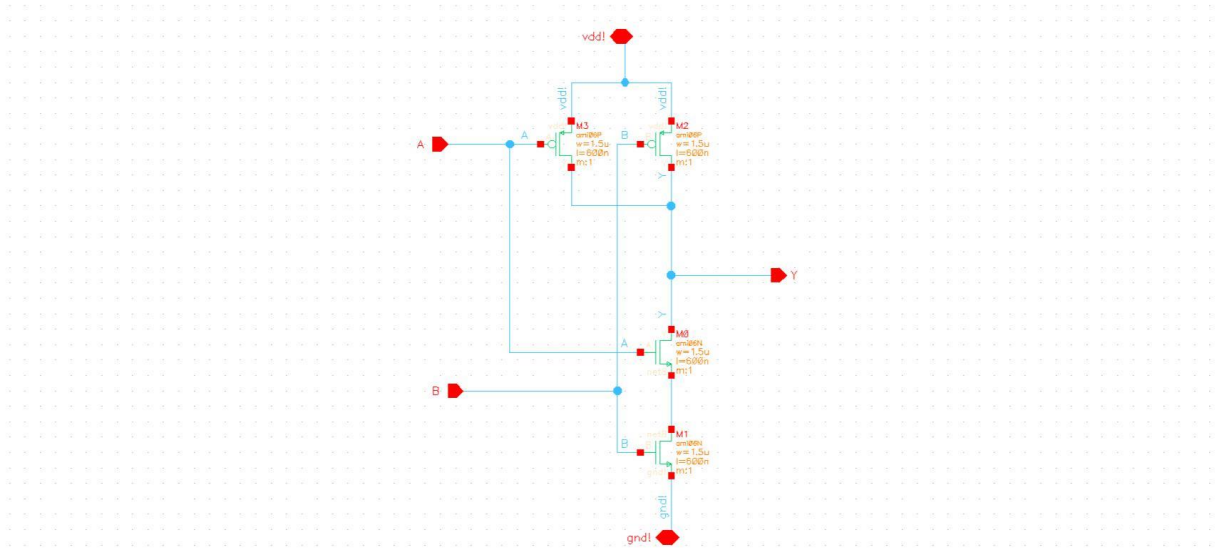
Final Output



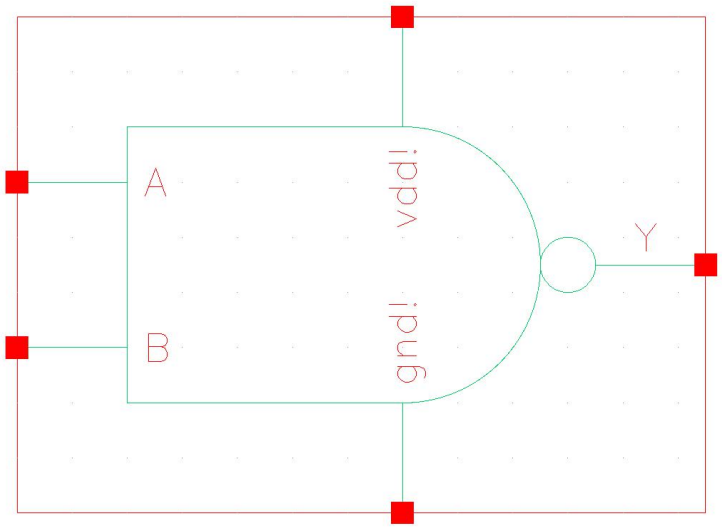
5.4 NAND GATE

2-input NAND

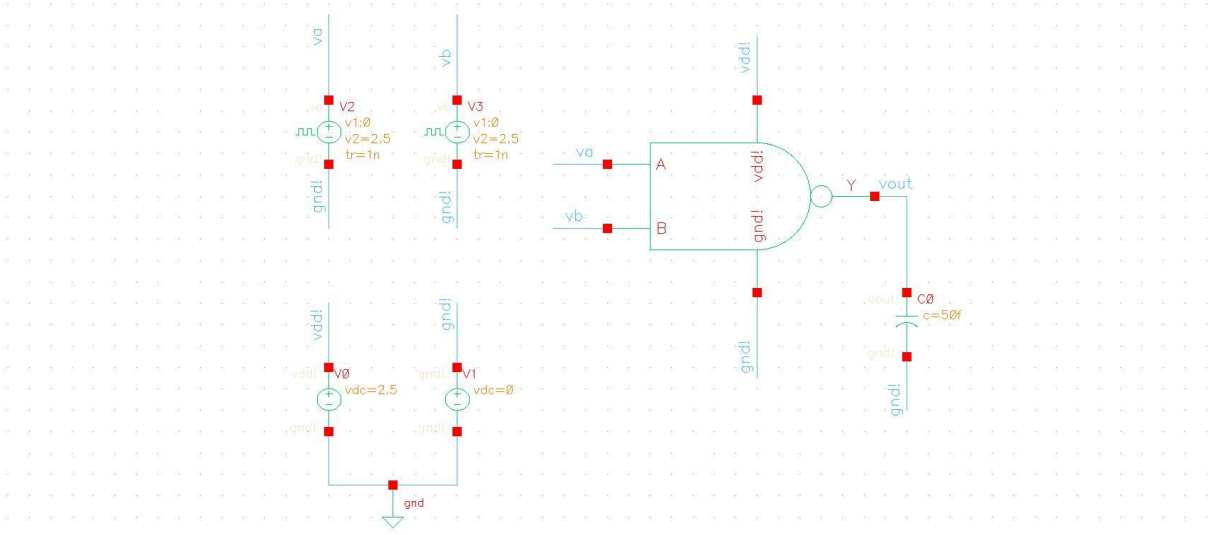
Schematic



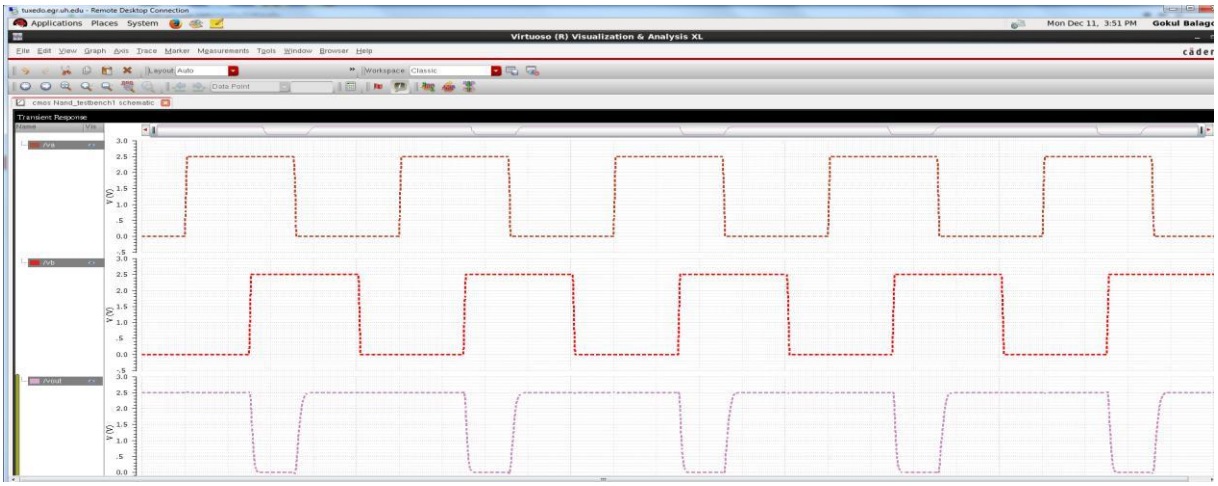
Symbol



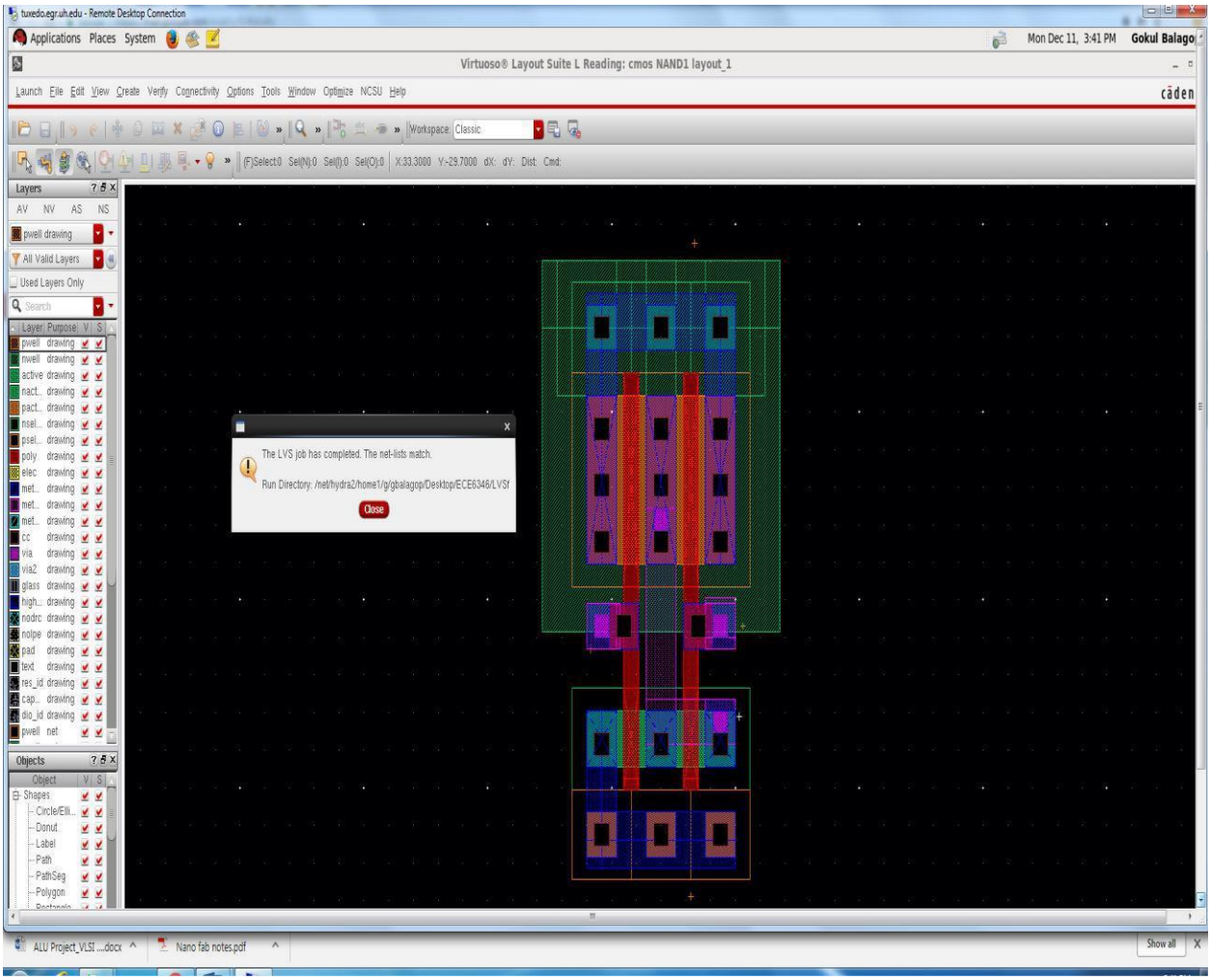
Testbench



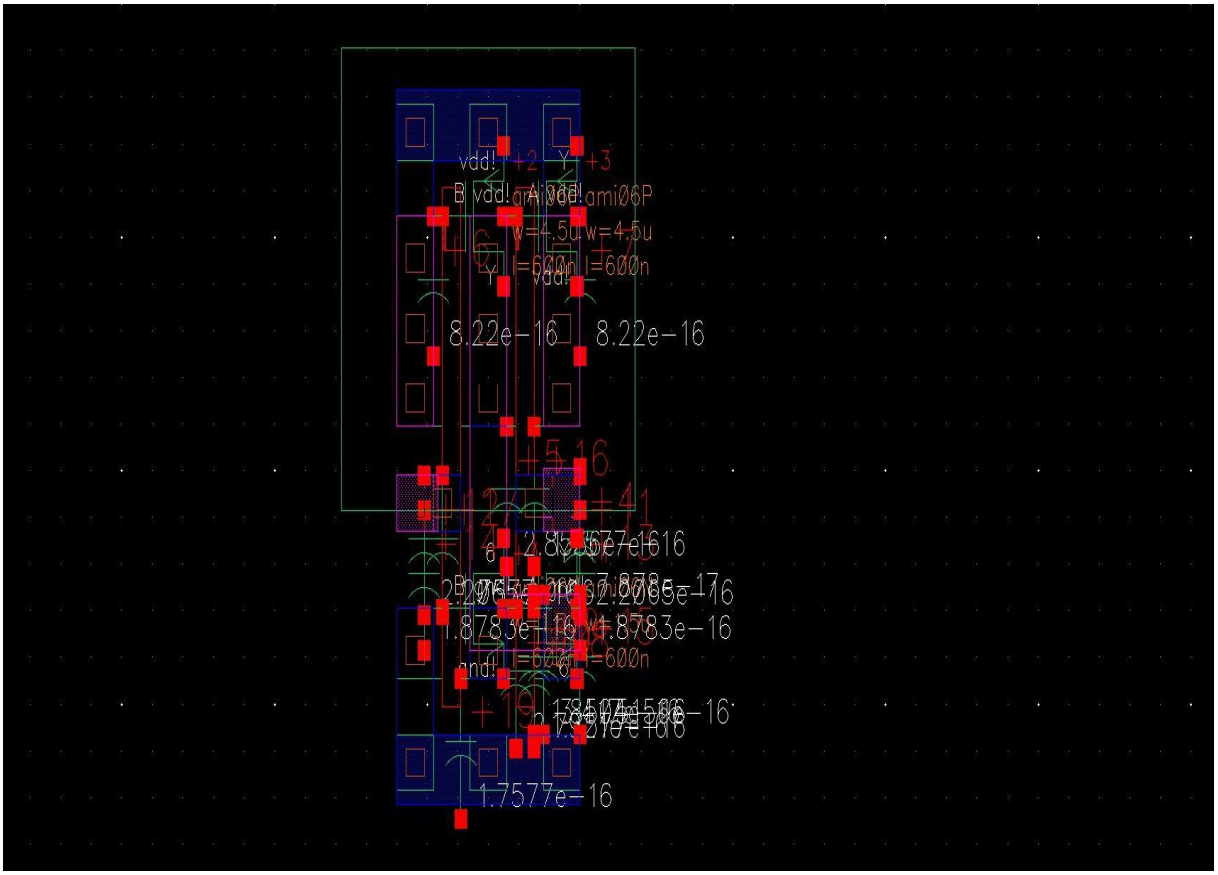
Testbench Output Wave form



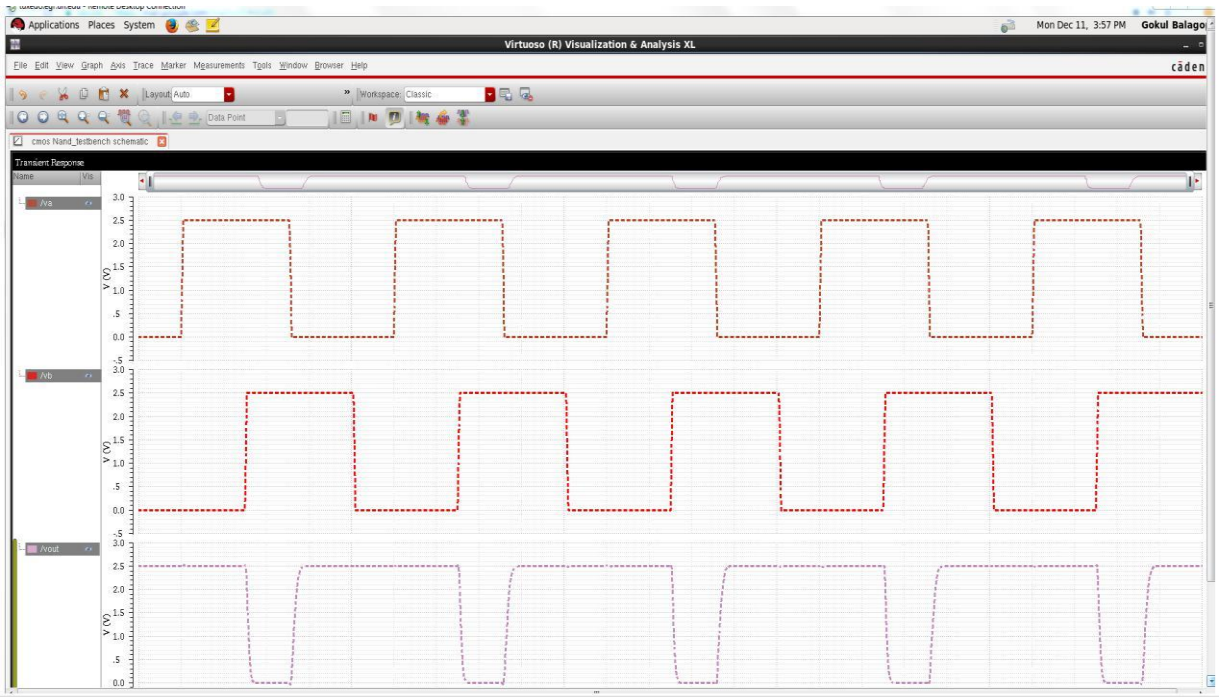
Layout & LVS



Parasitic Extracted

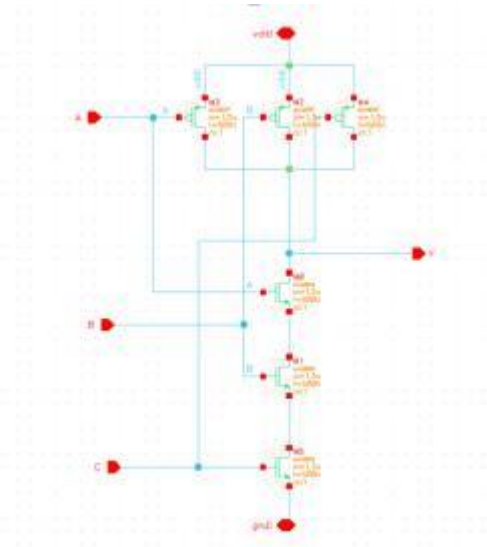


Final Output

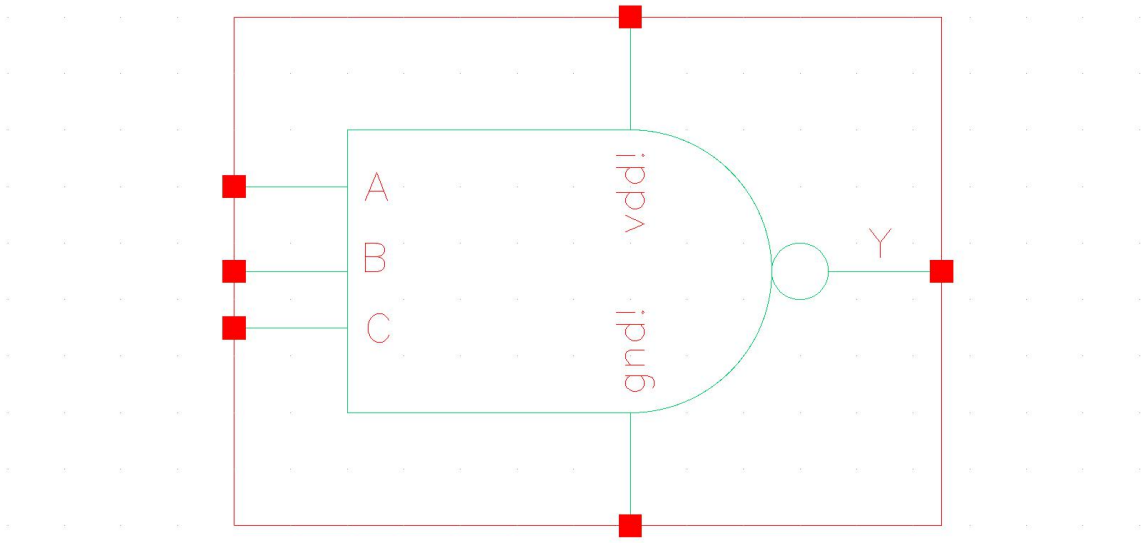


3-input NAND GATE

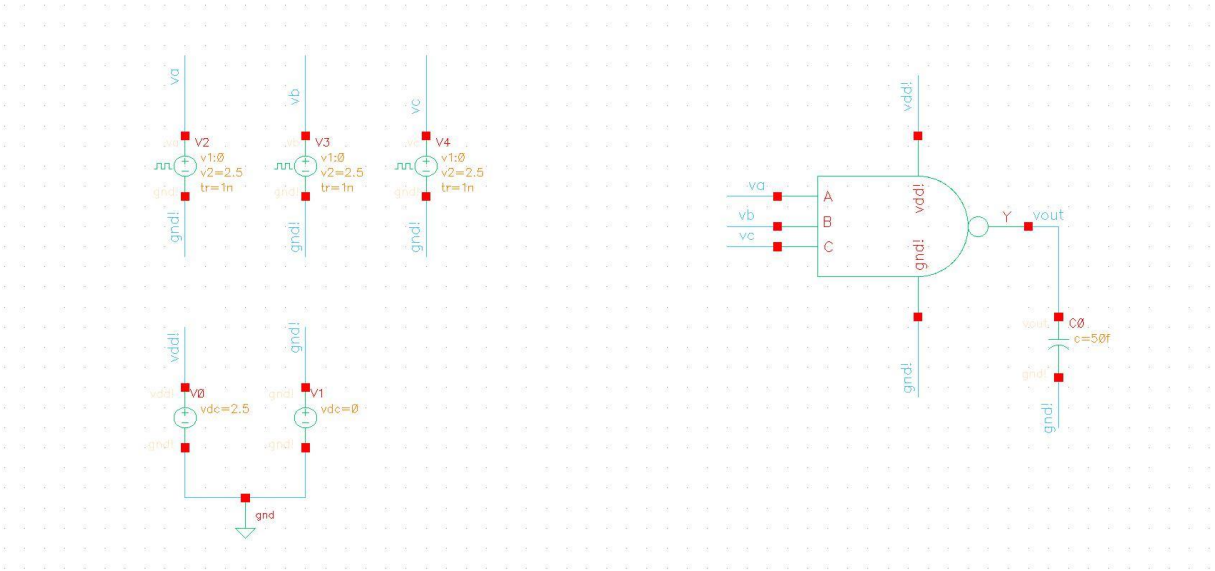
Schematic



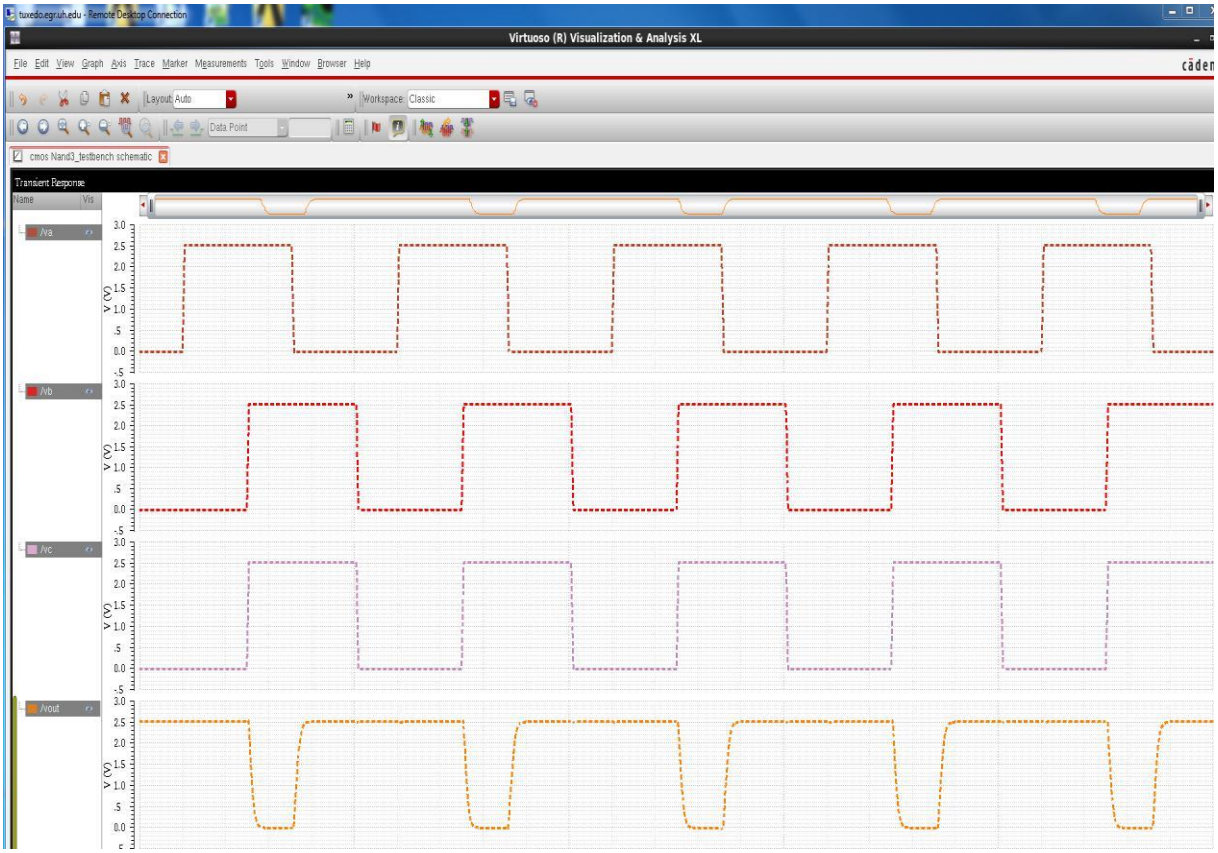
Symbol



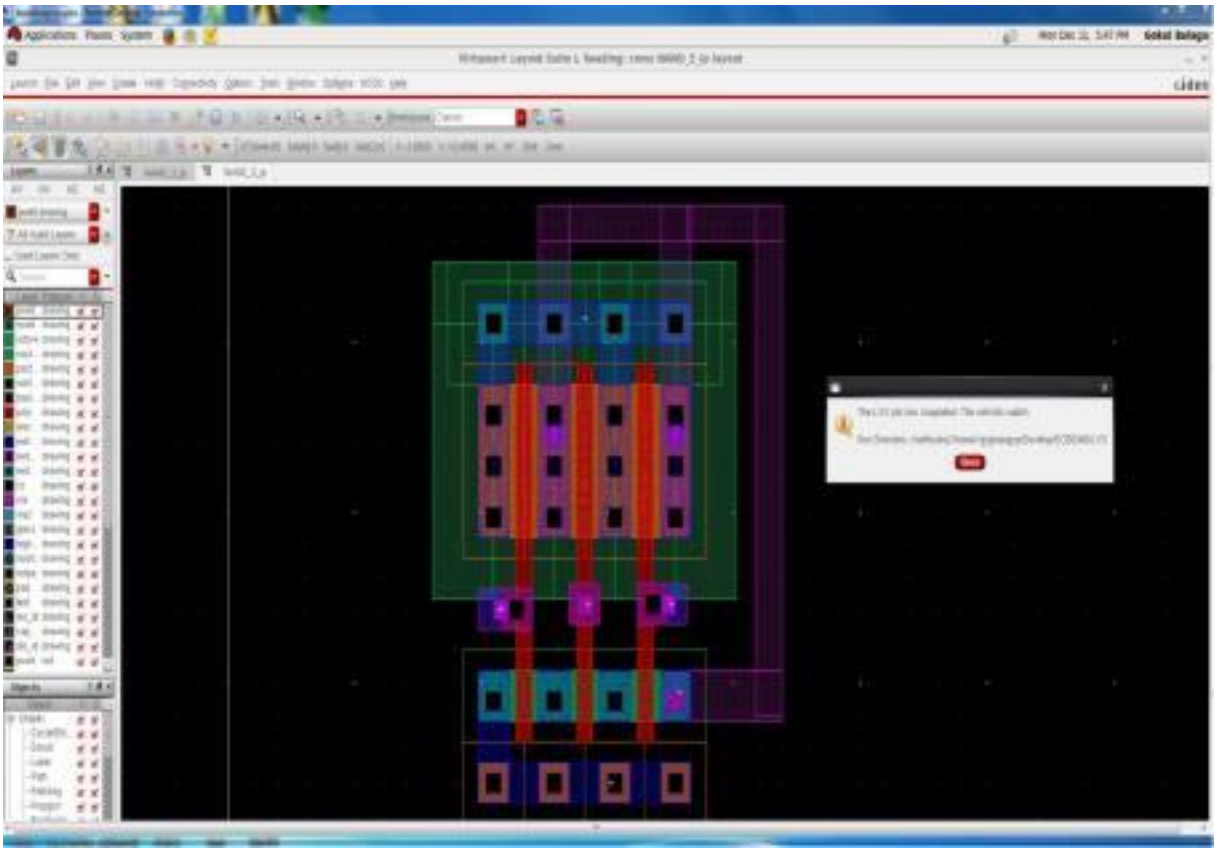
Testbench



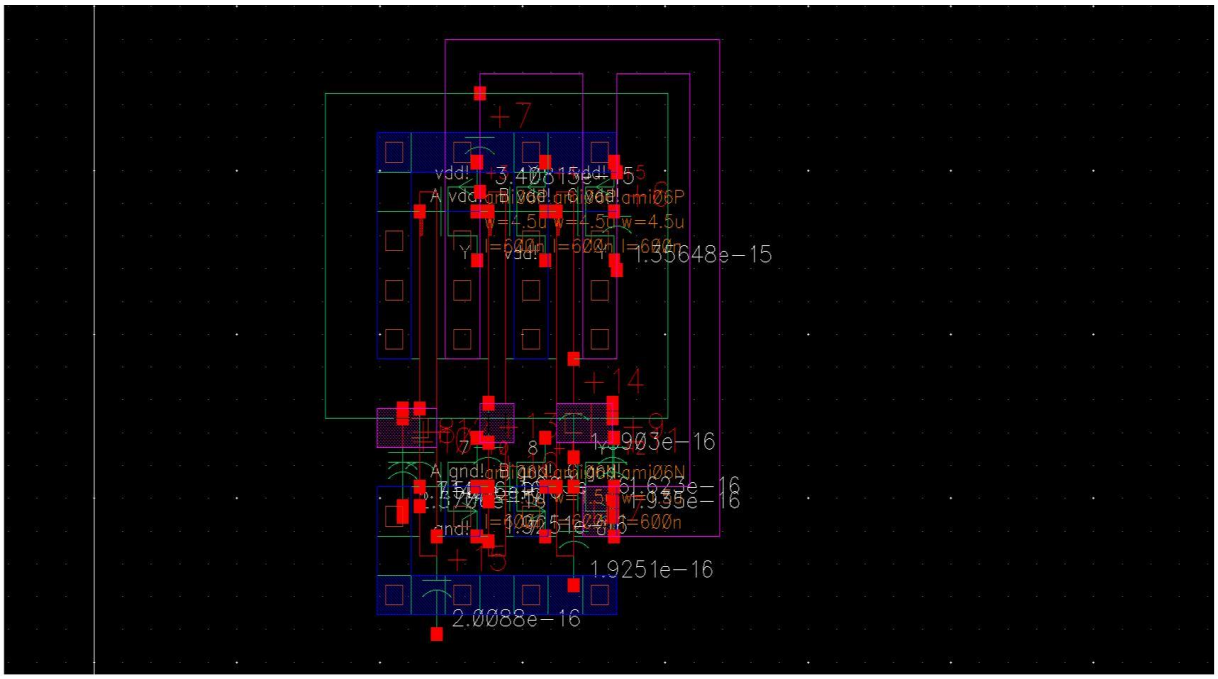
Testbench Output waveform



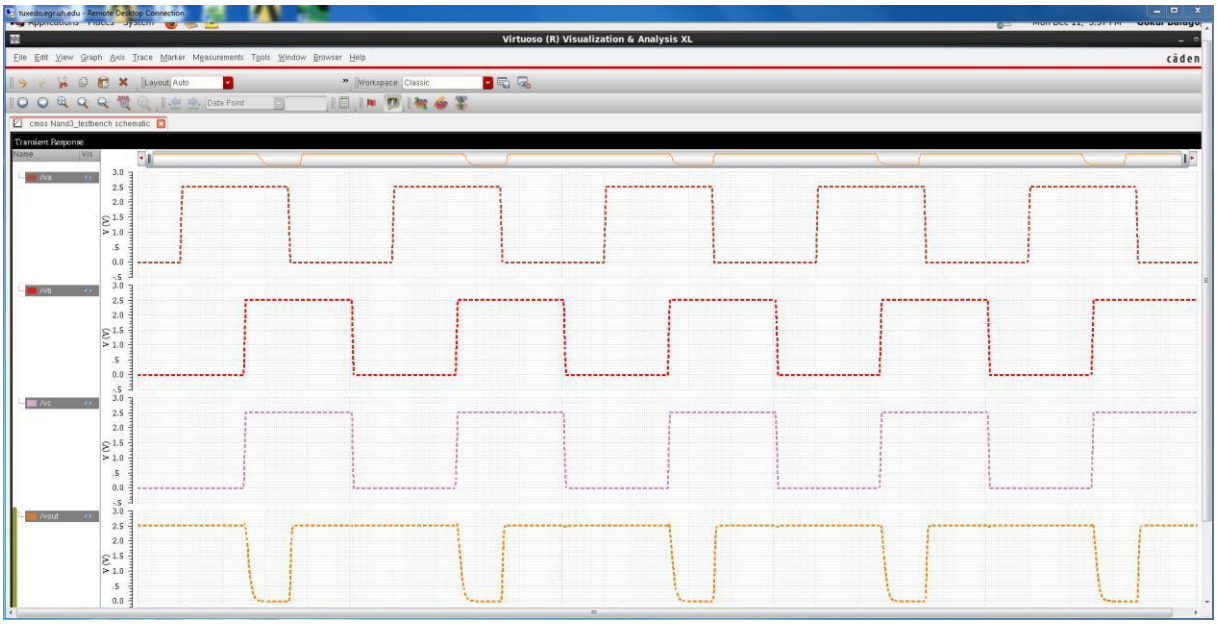
Layout & LVS



Parasitic Extracted

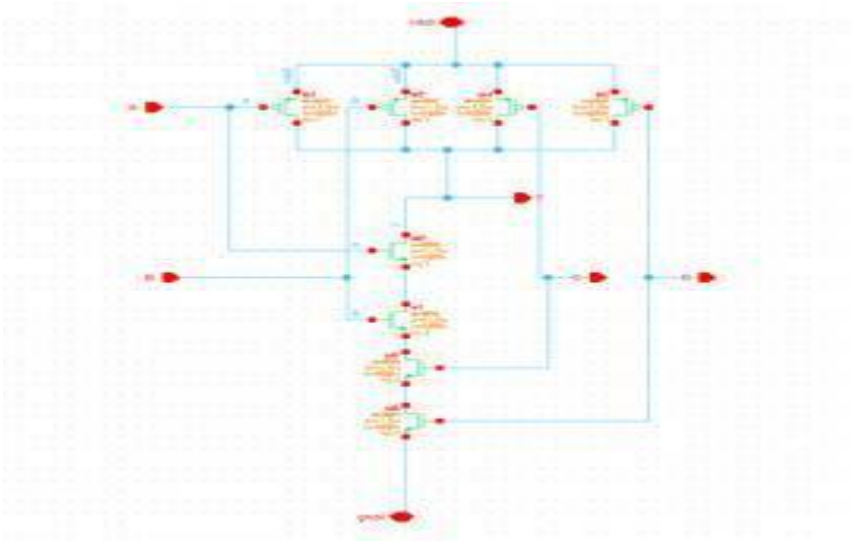


Final Output Waveform

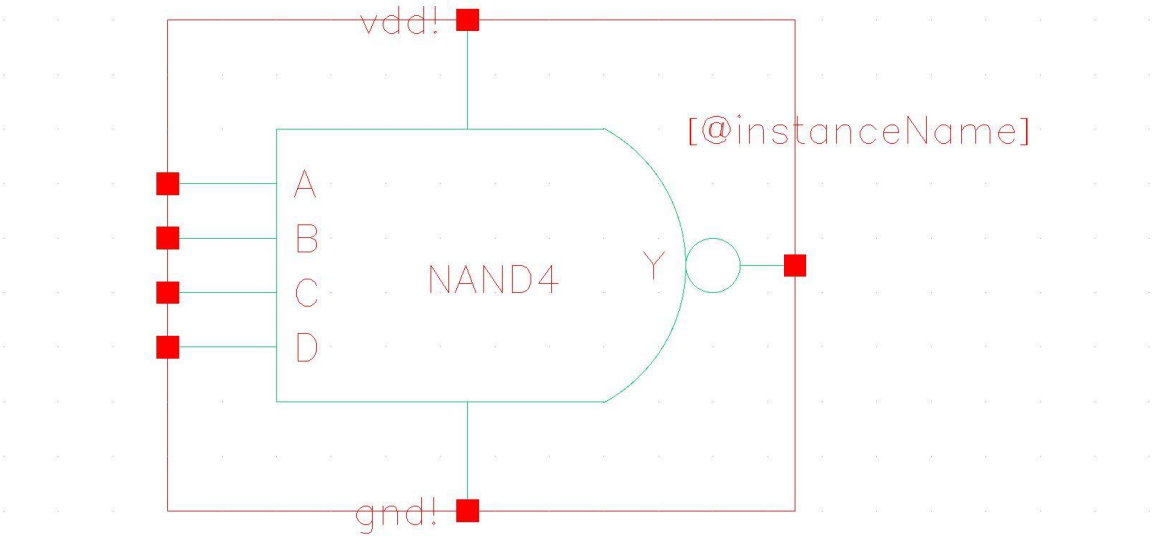


4-input NAND

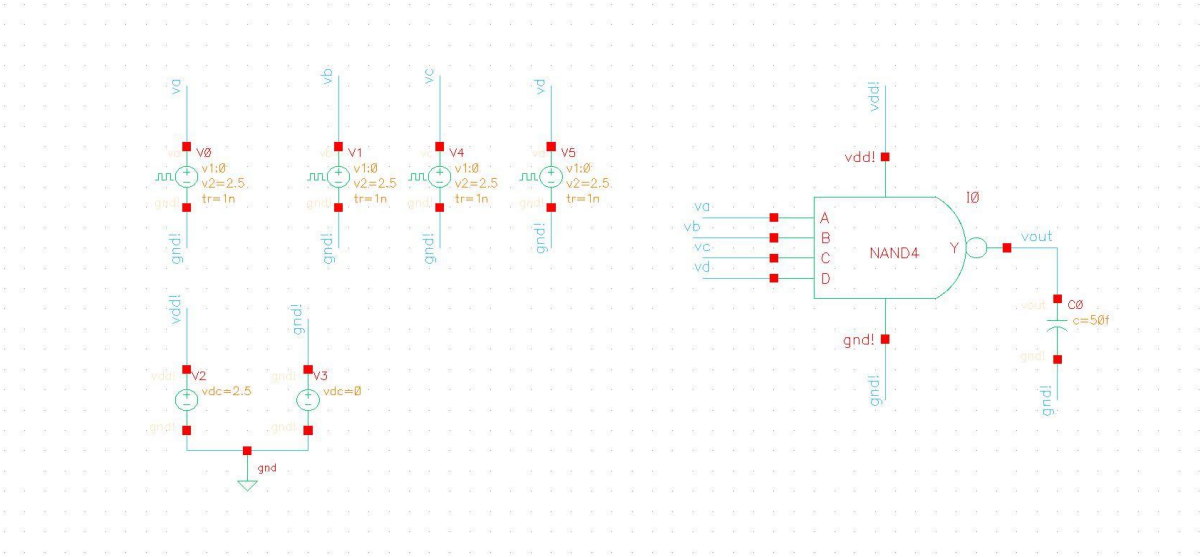
Schematic



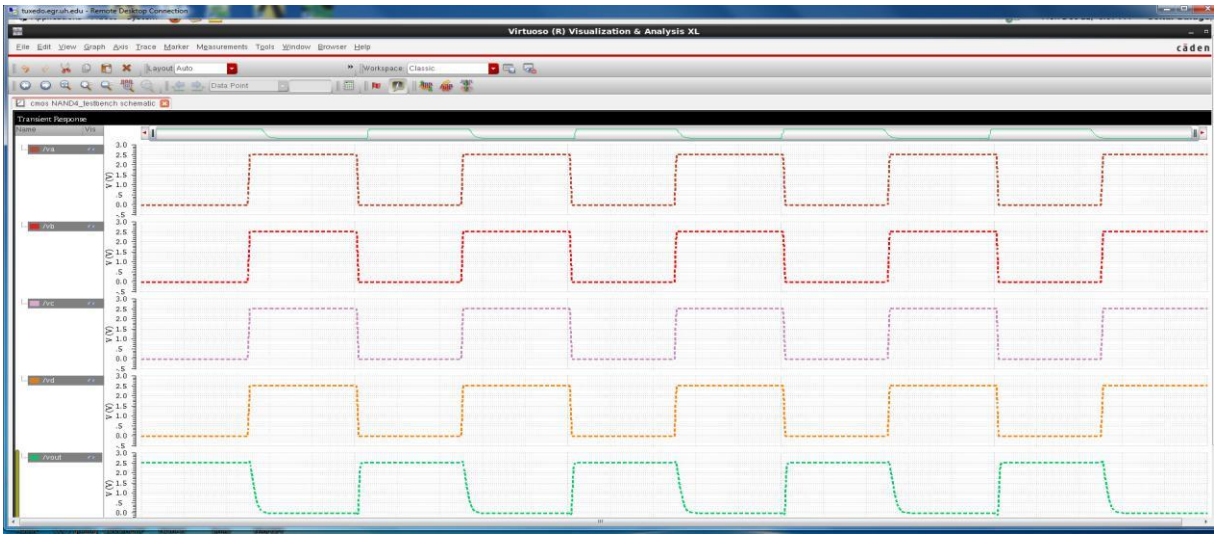
Symbol



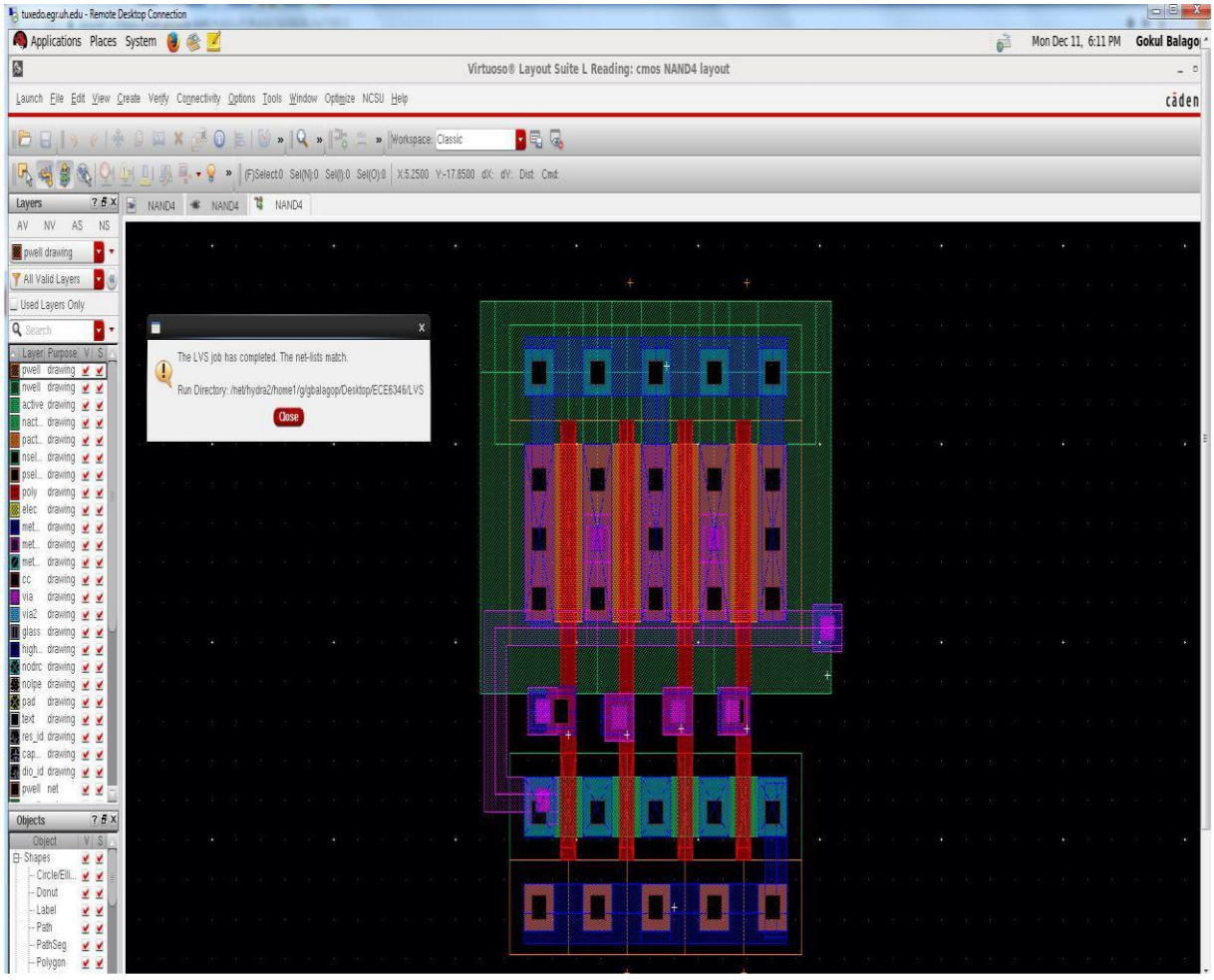
Testbench



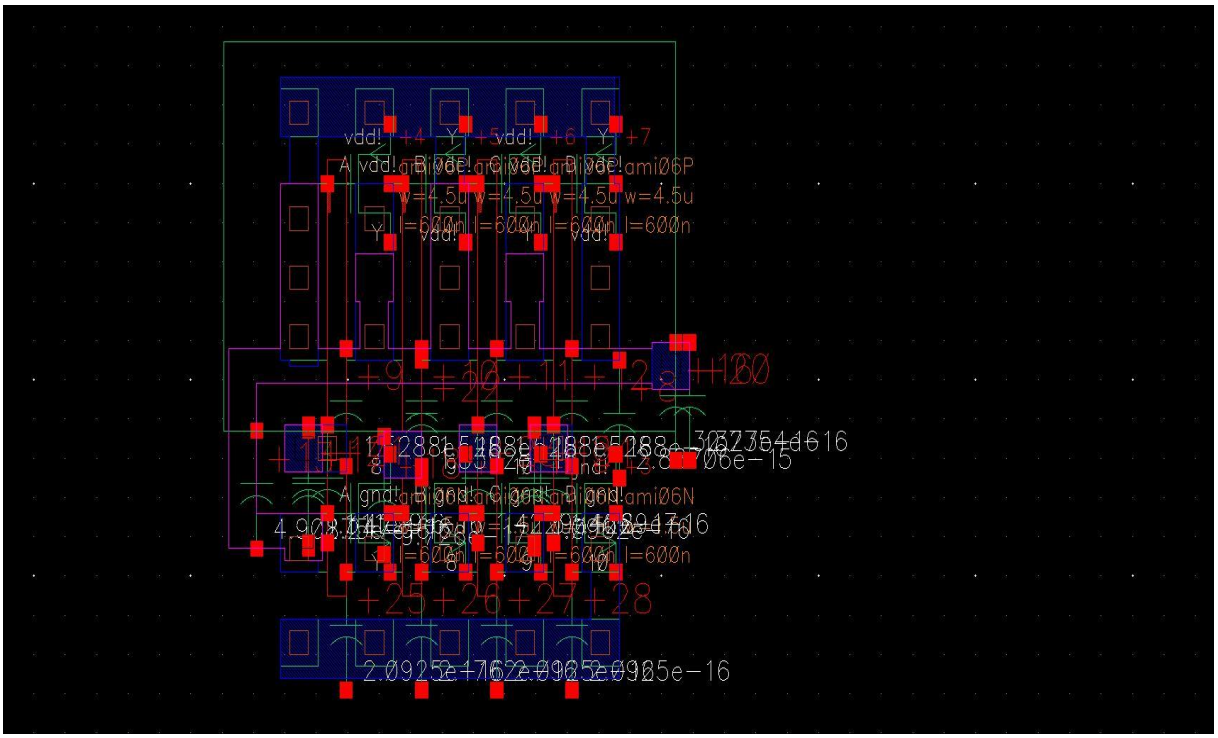
Testbench Output Waveform



Layout & LVS



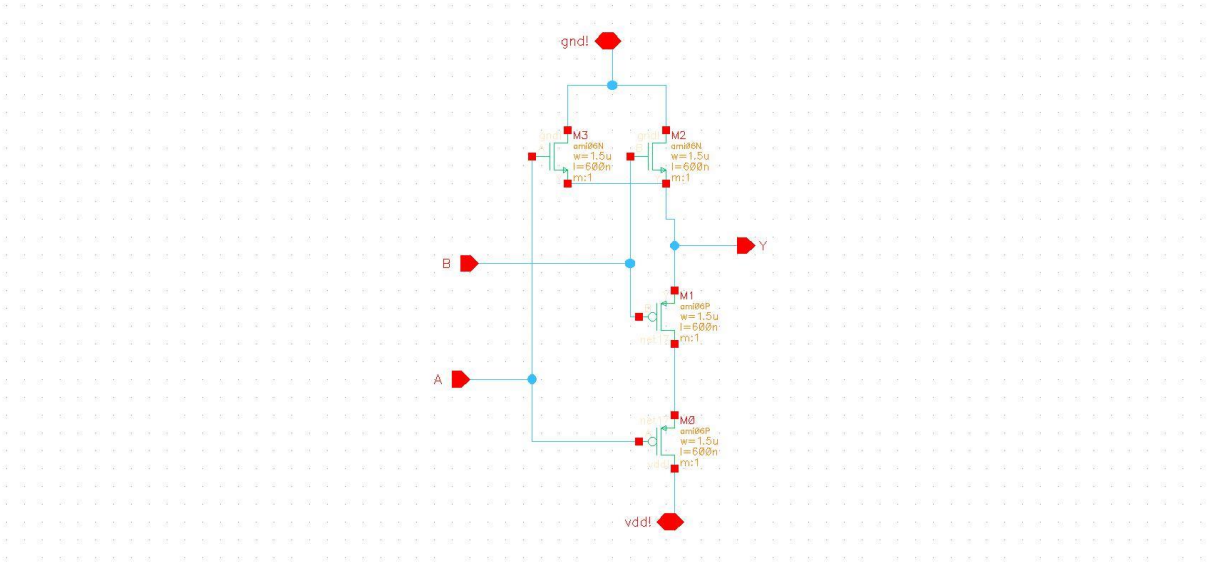
Parasitic Extracted



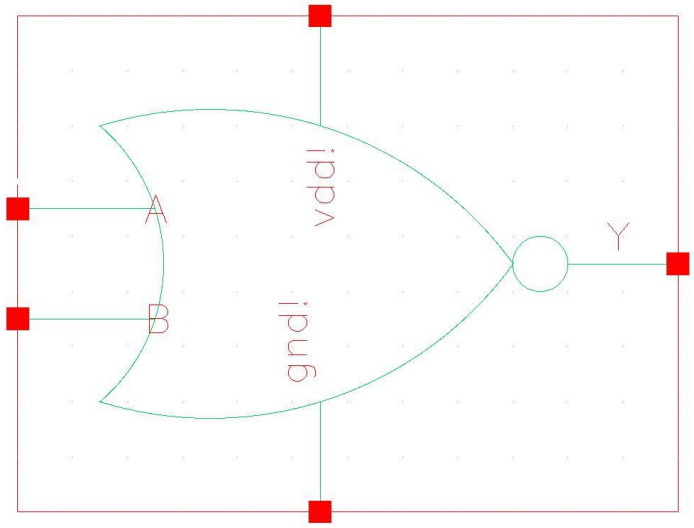
Final Output Waveform

5.5 NOR

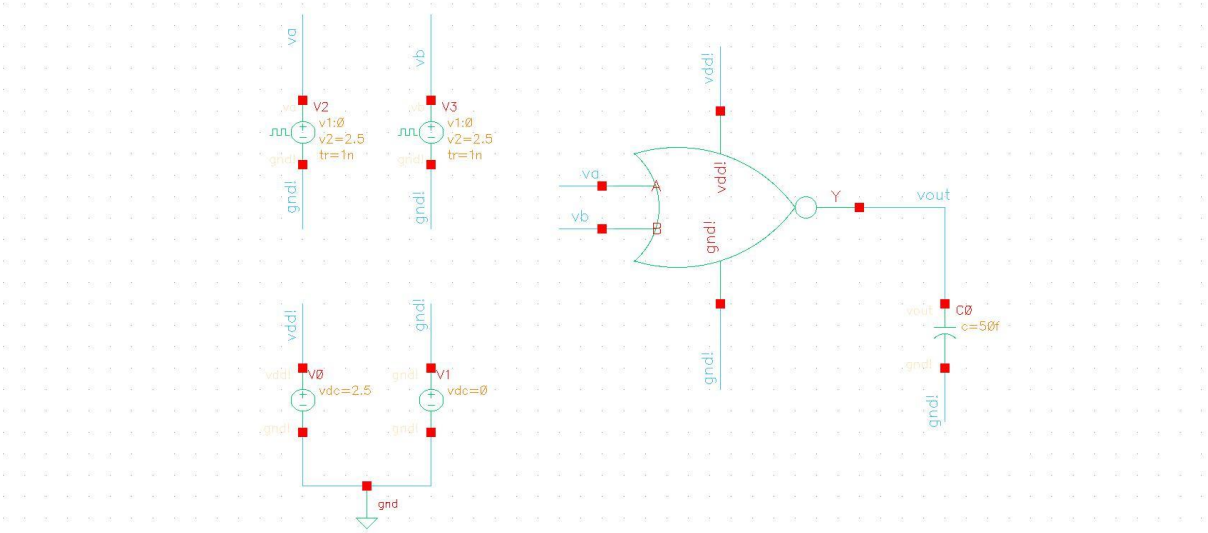
Schematic



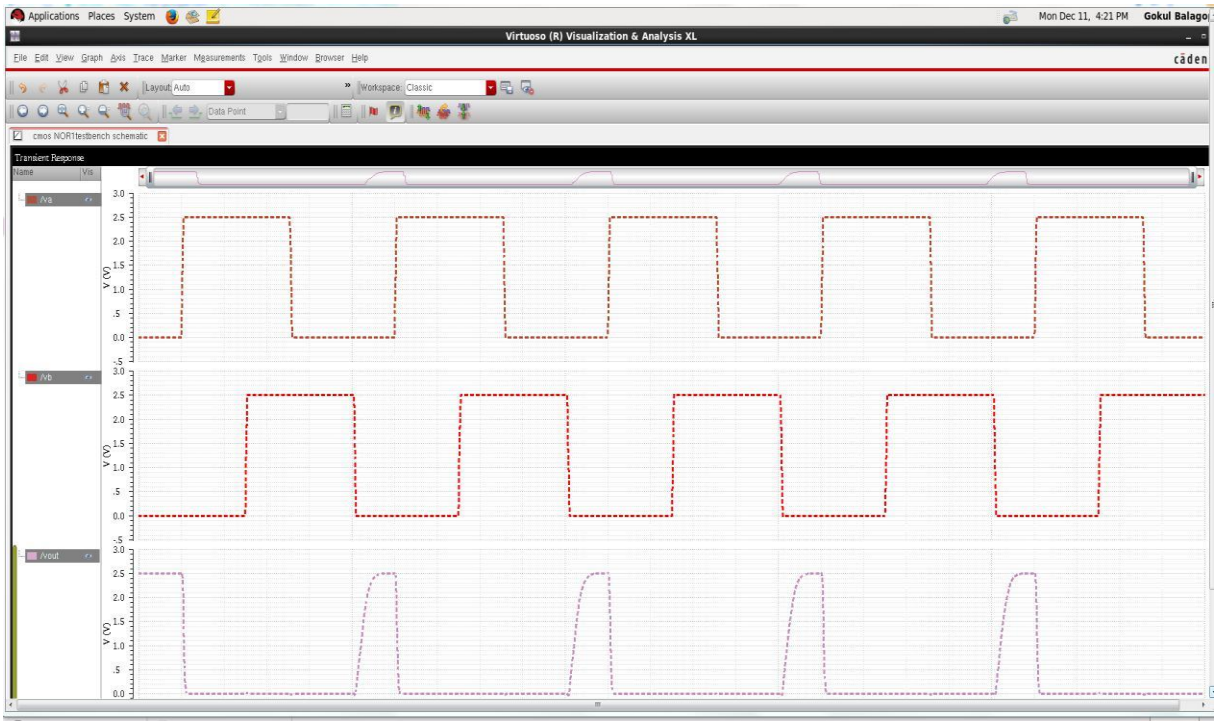
Symbol



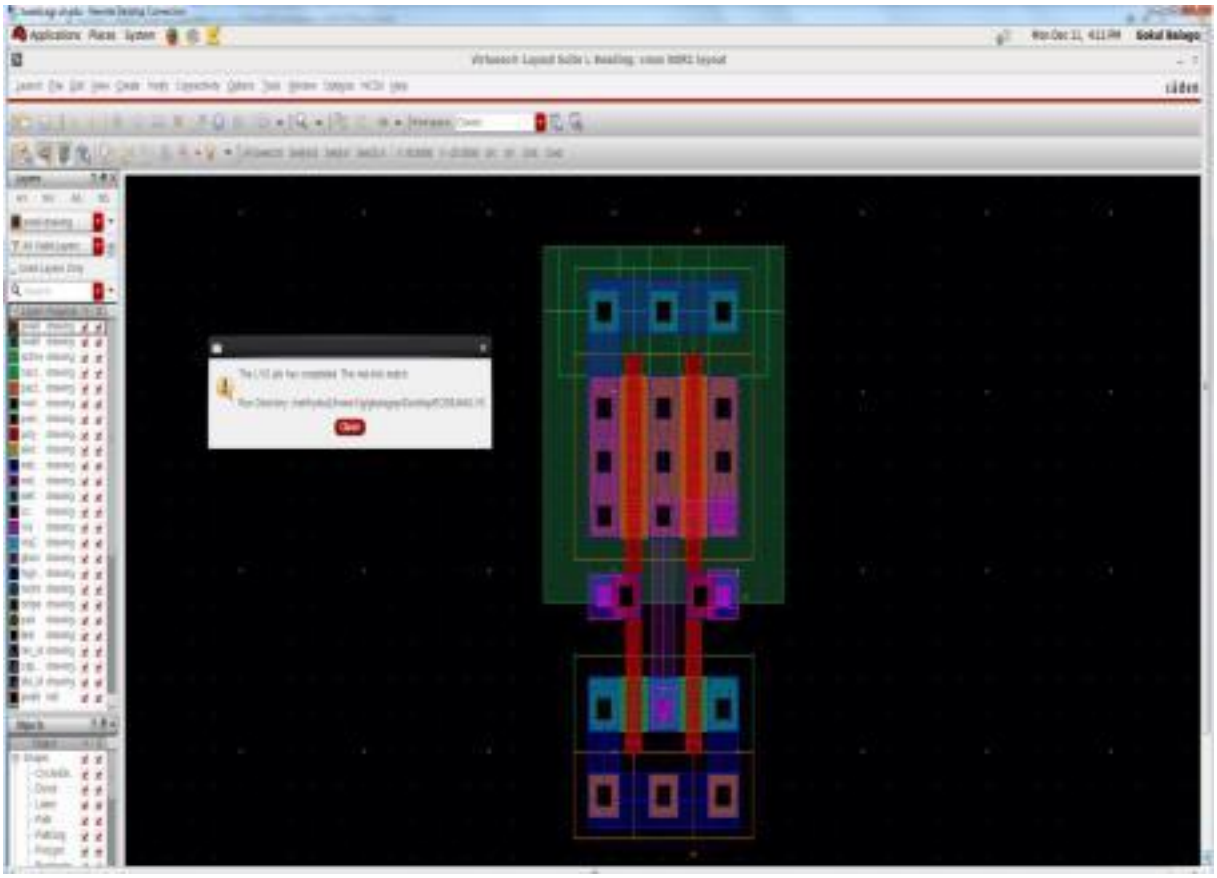
Testbench



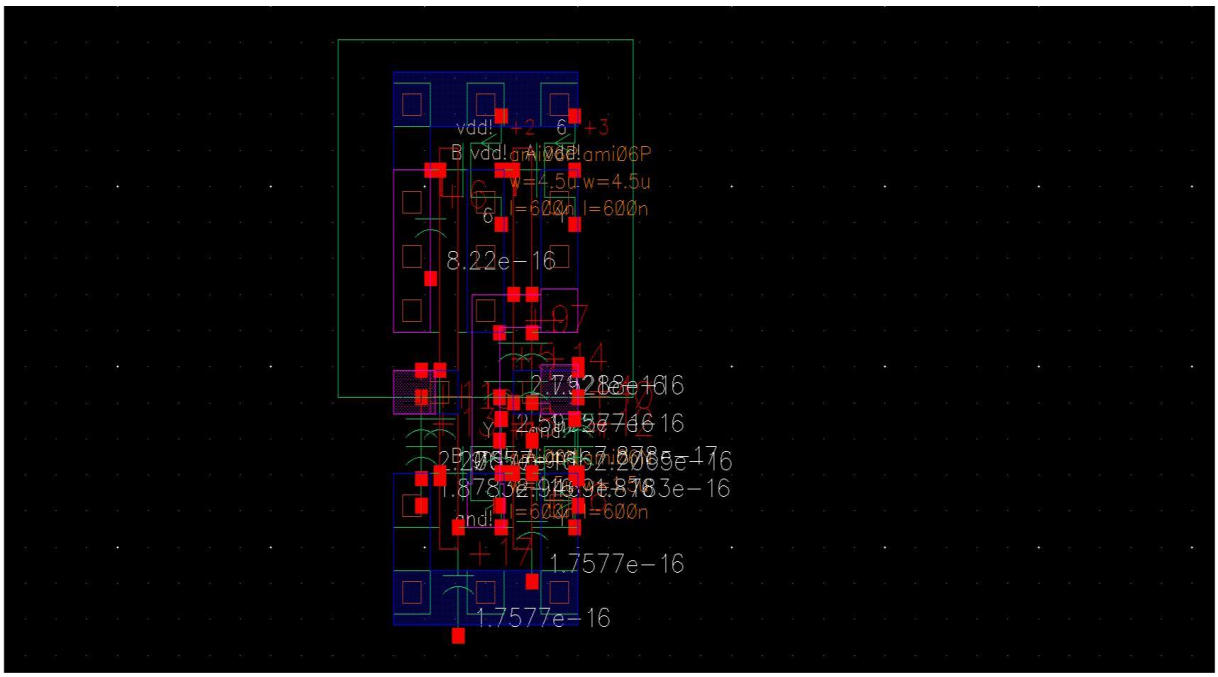
Testbench output wave form



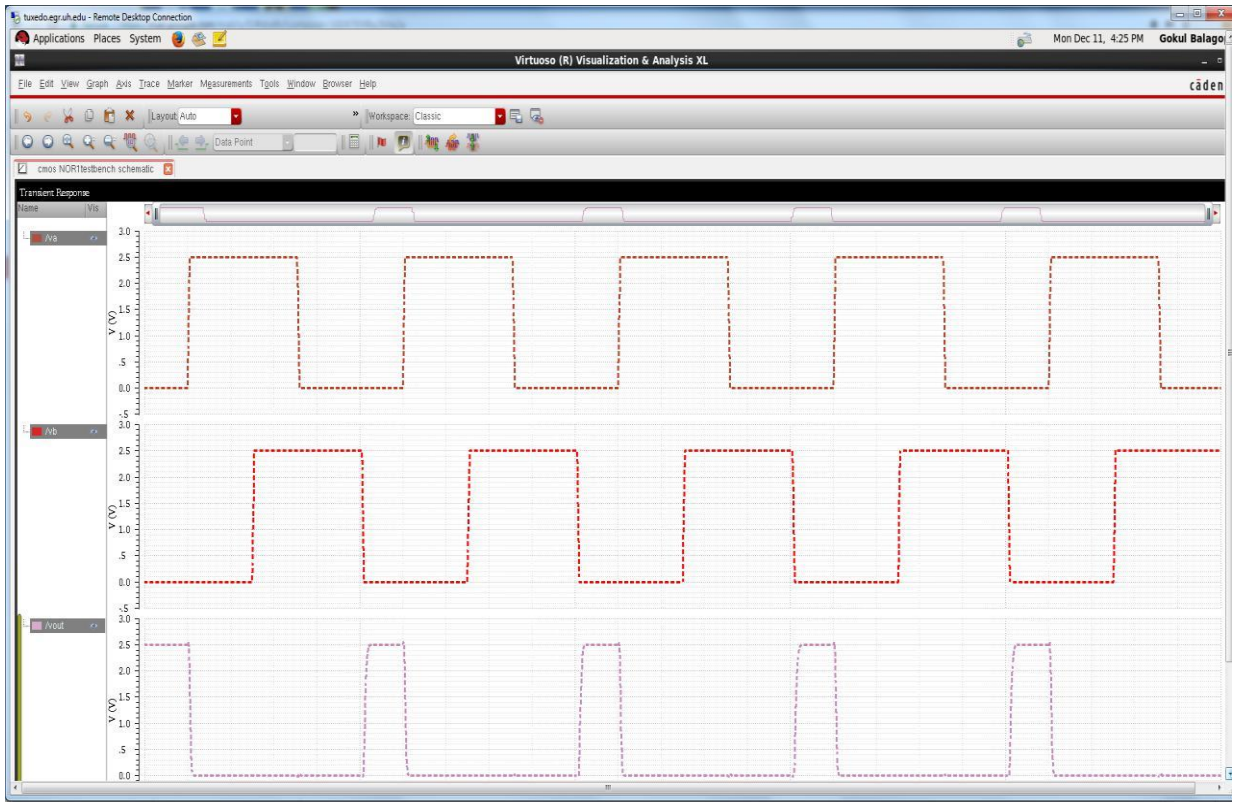
Layout & LVS



Parasitic Extracted

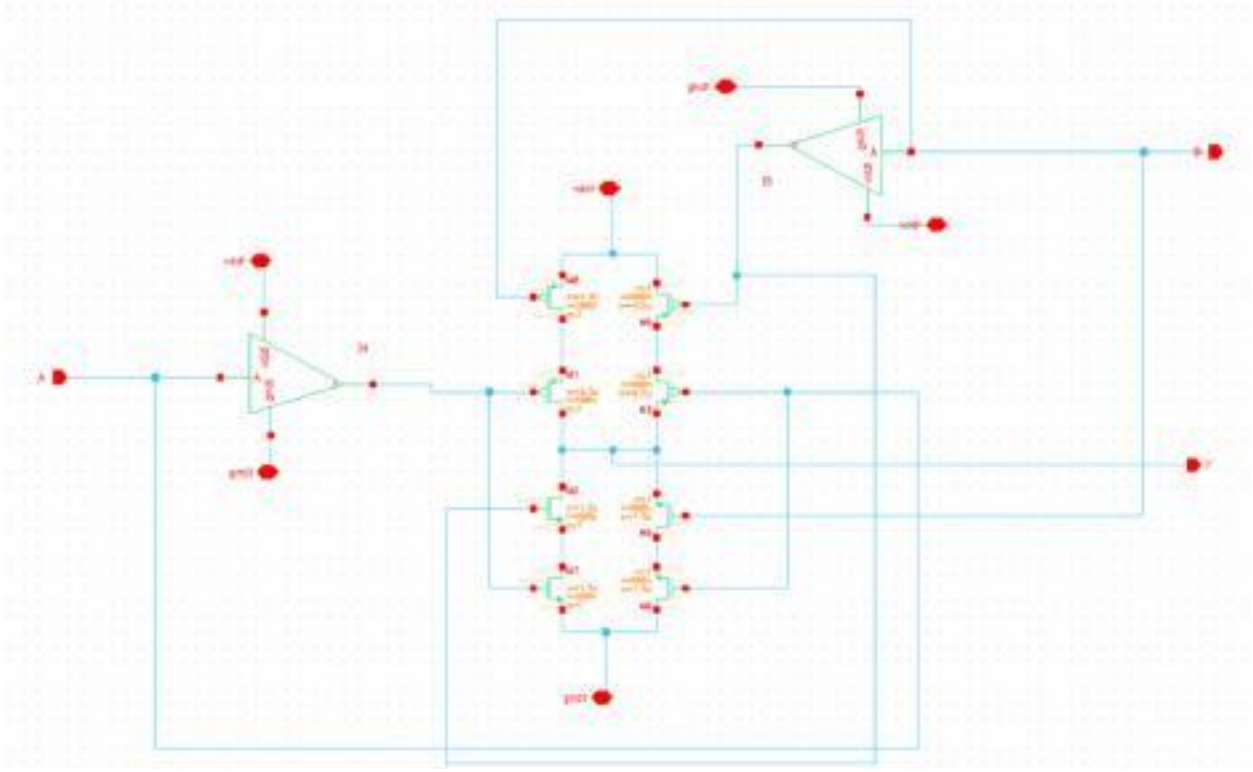


Final output waveform

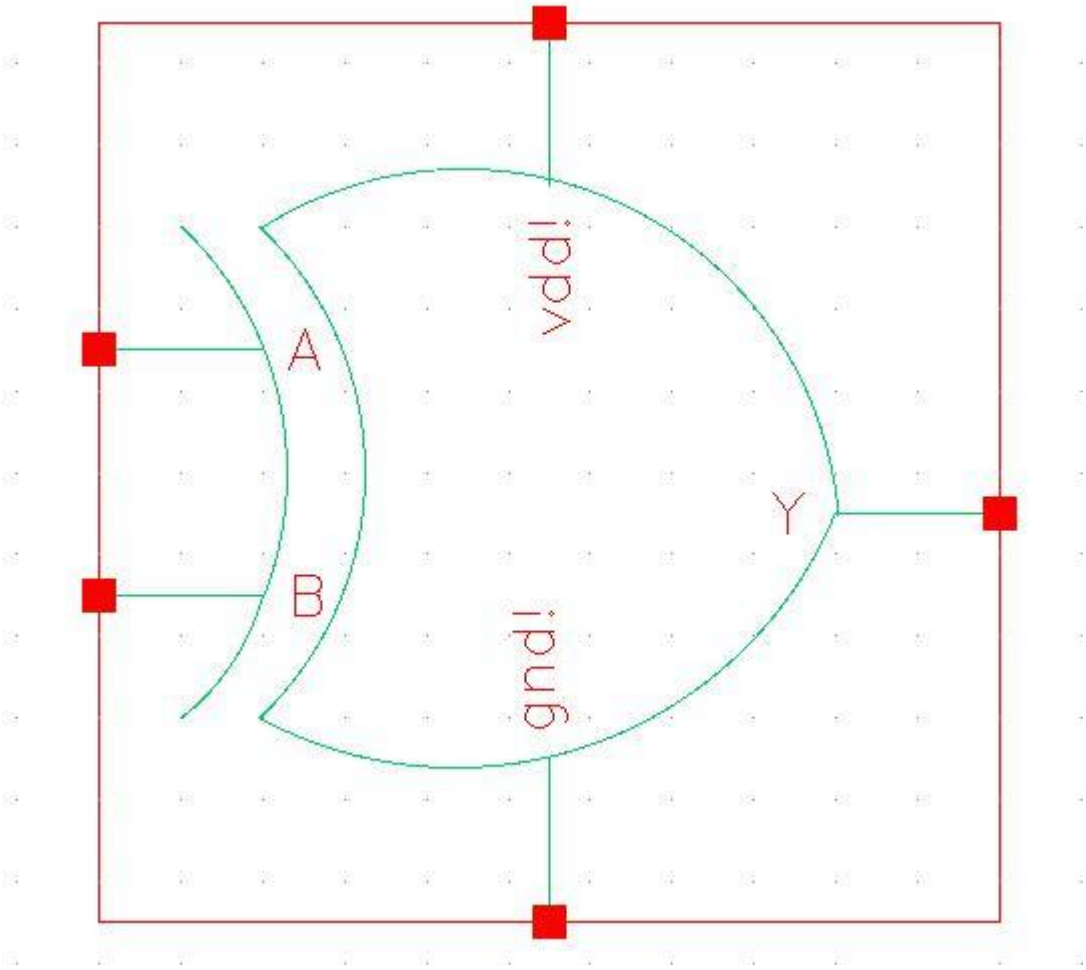


5.7 XOR

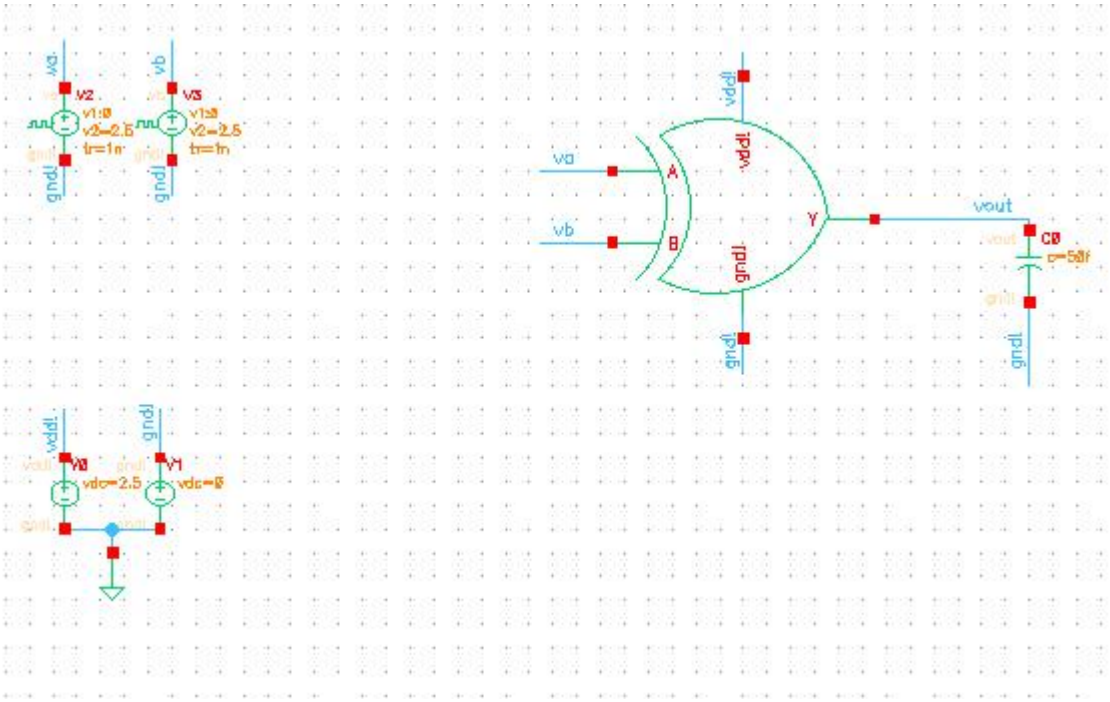
Schematic



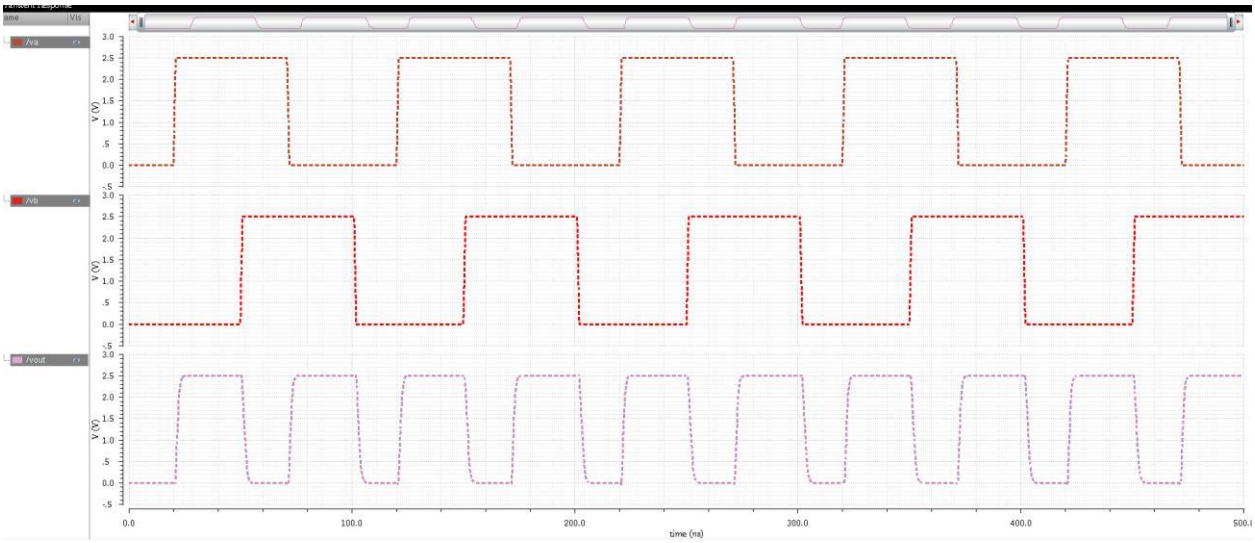
Symbol



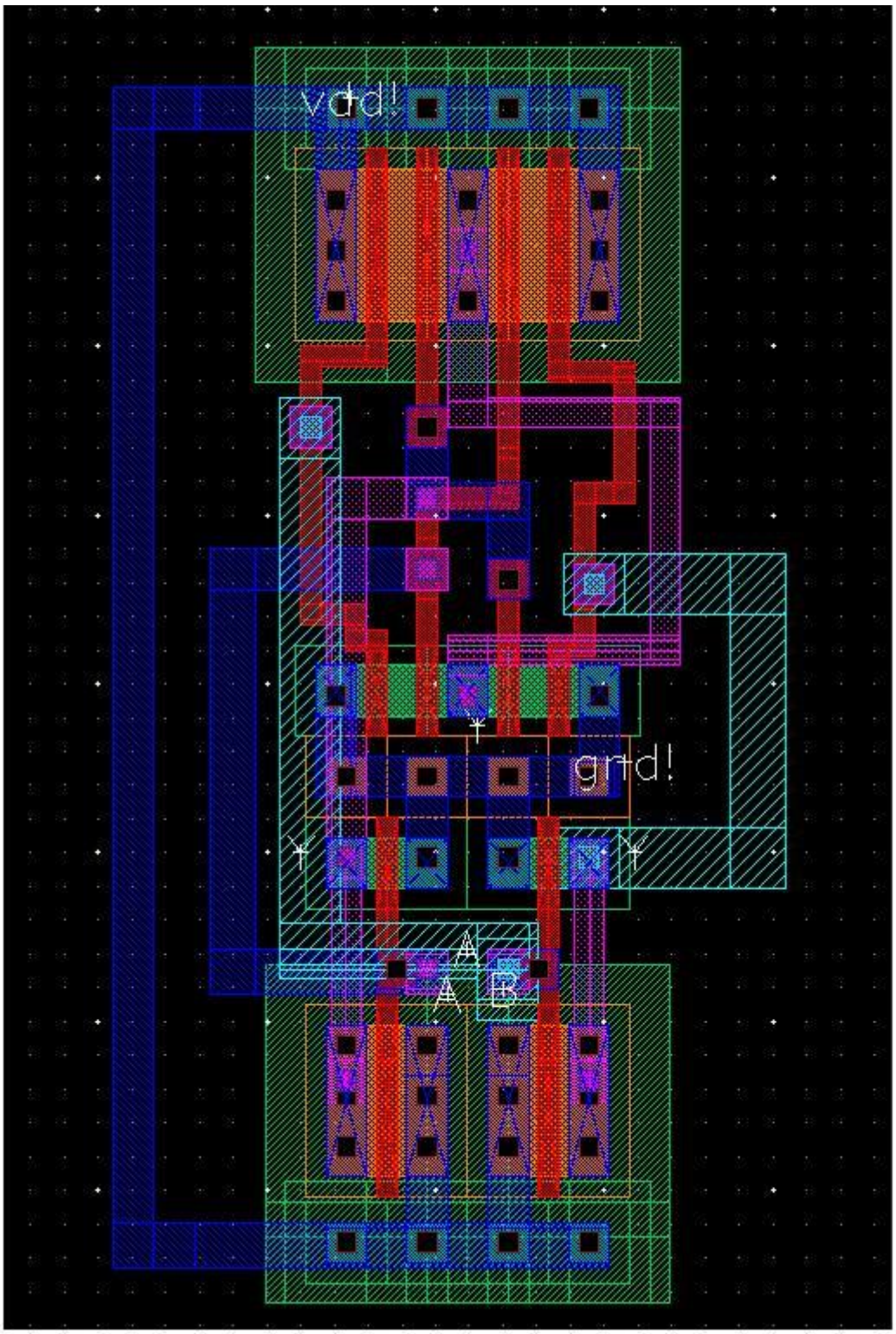
Testbench



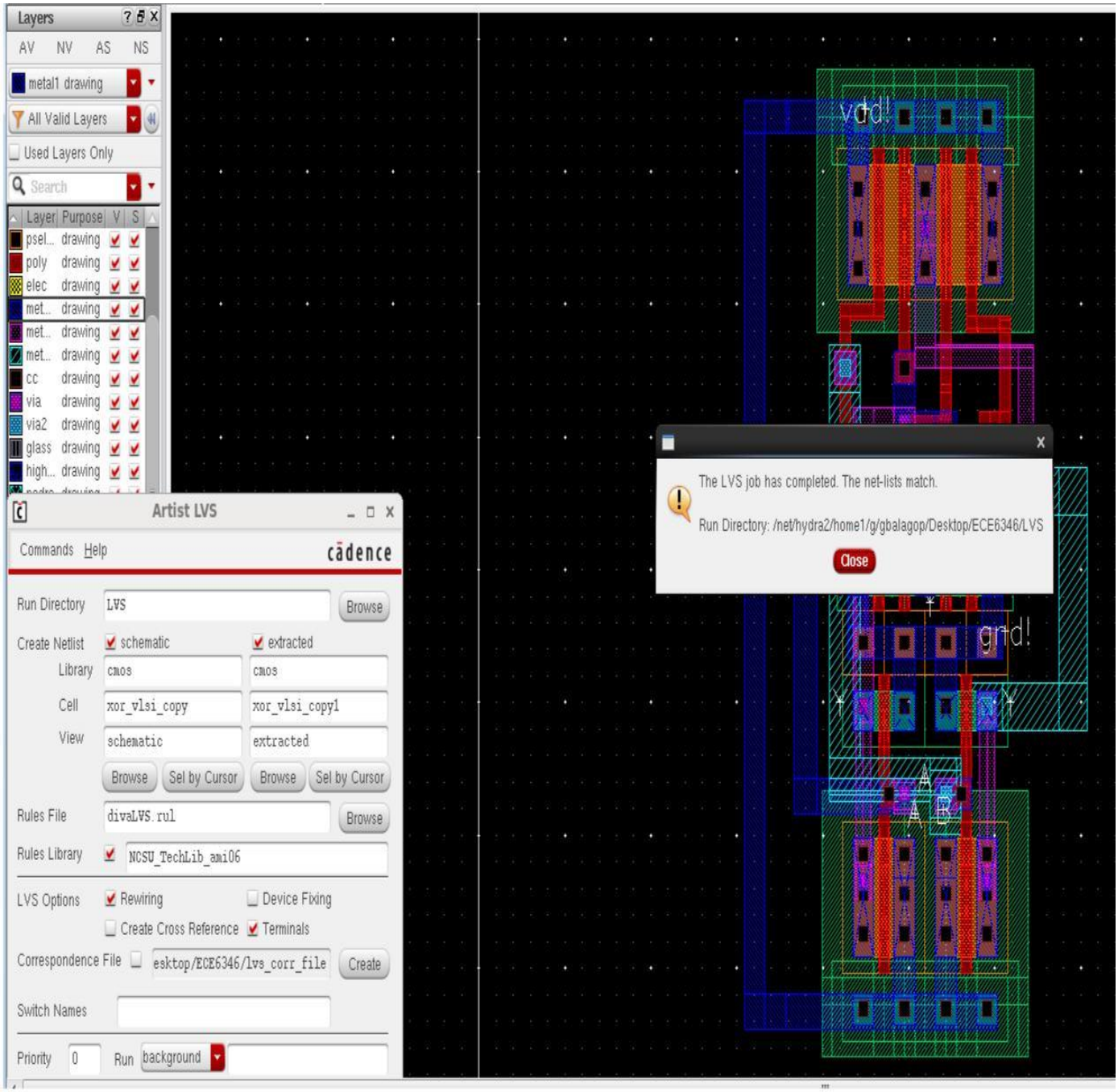
Testbench output wave form



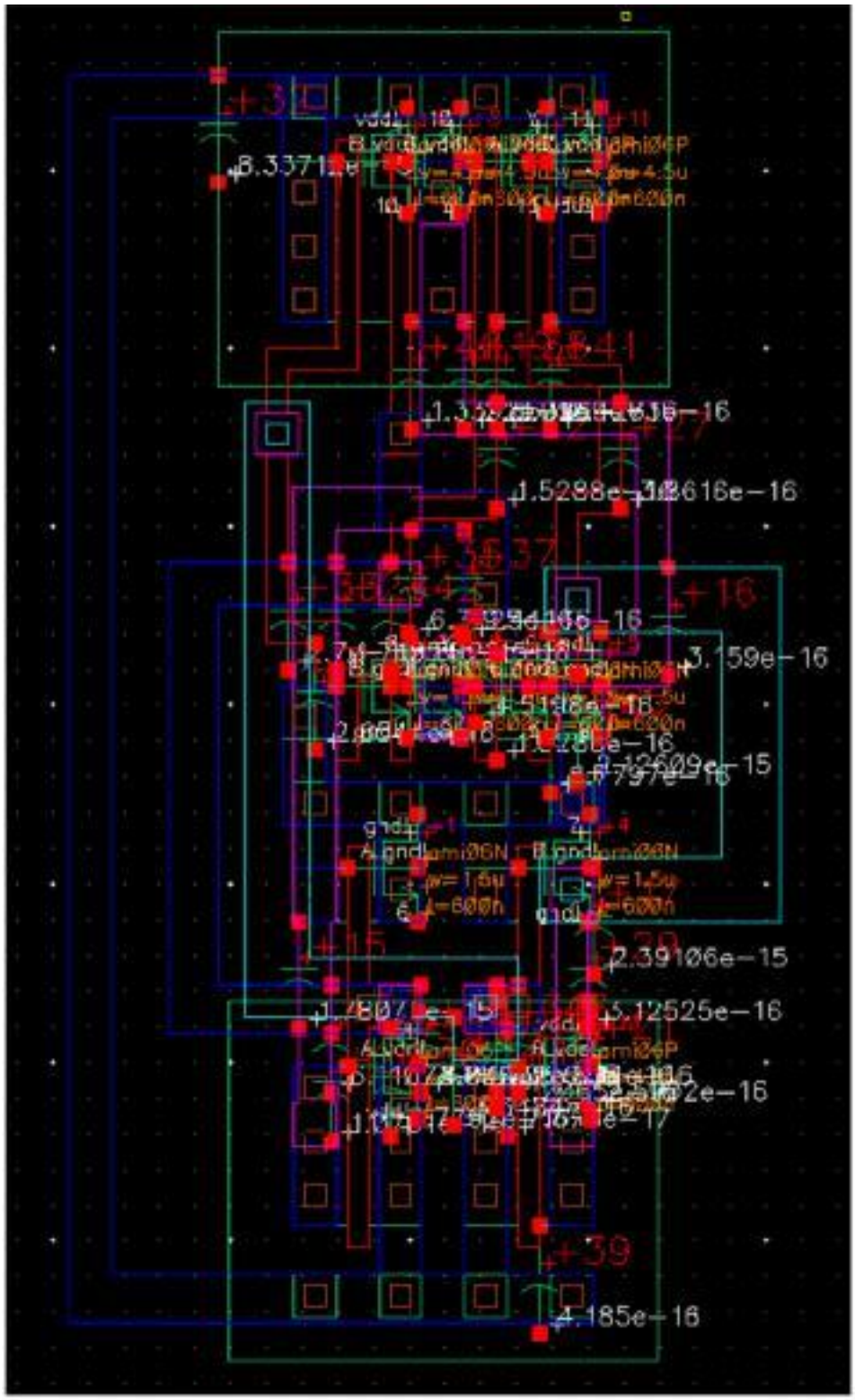
Layout



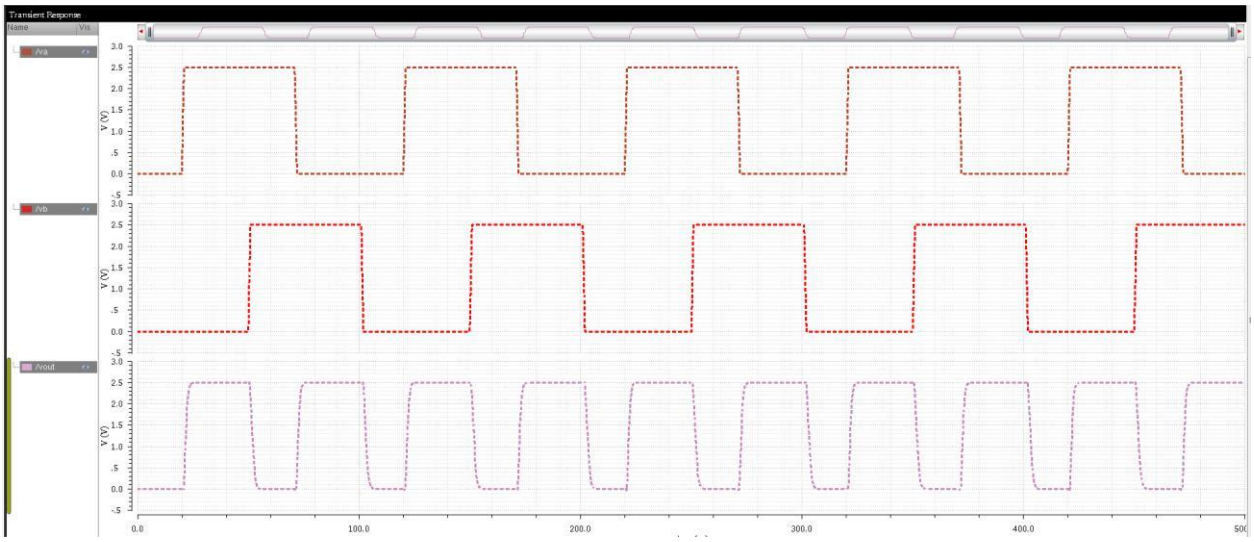
Layout & LVS verified



Parasitic Extracted

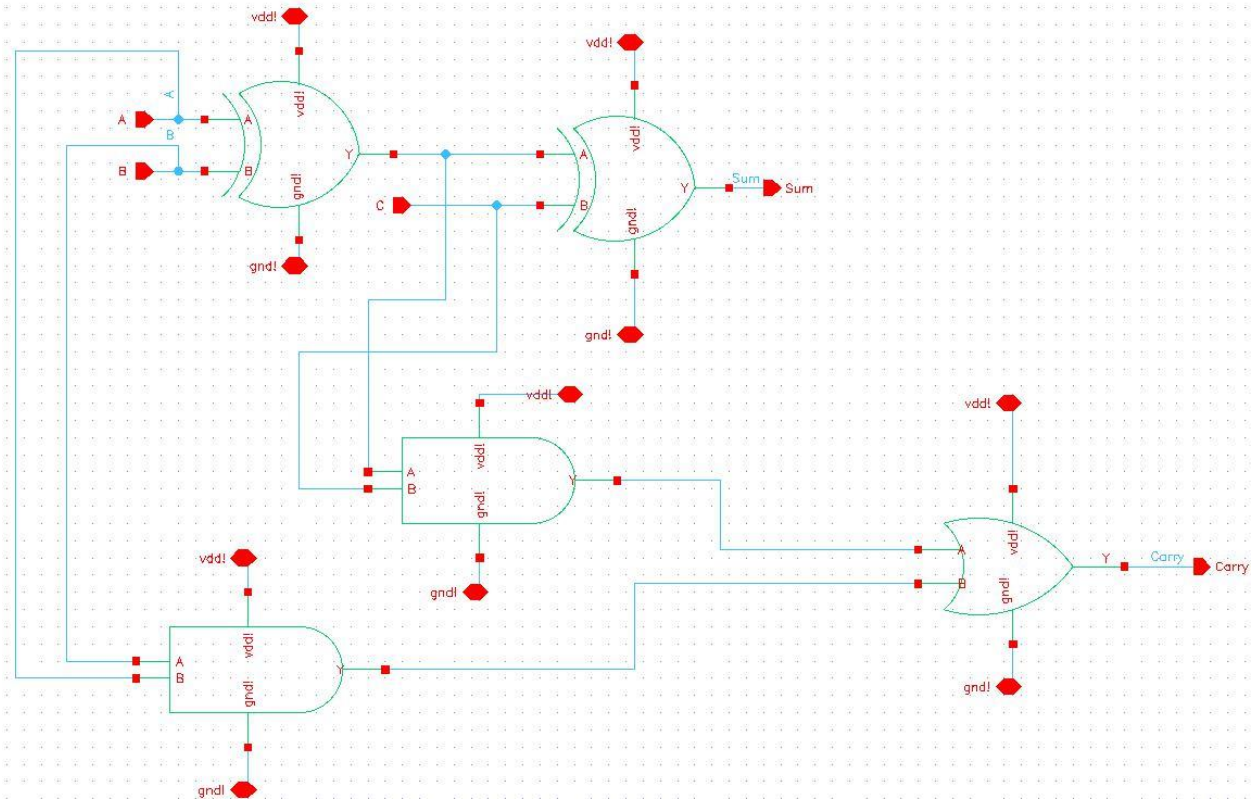


Final output waveform

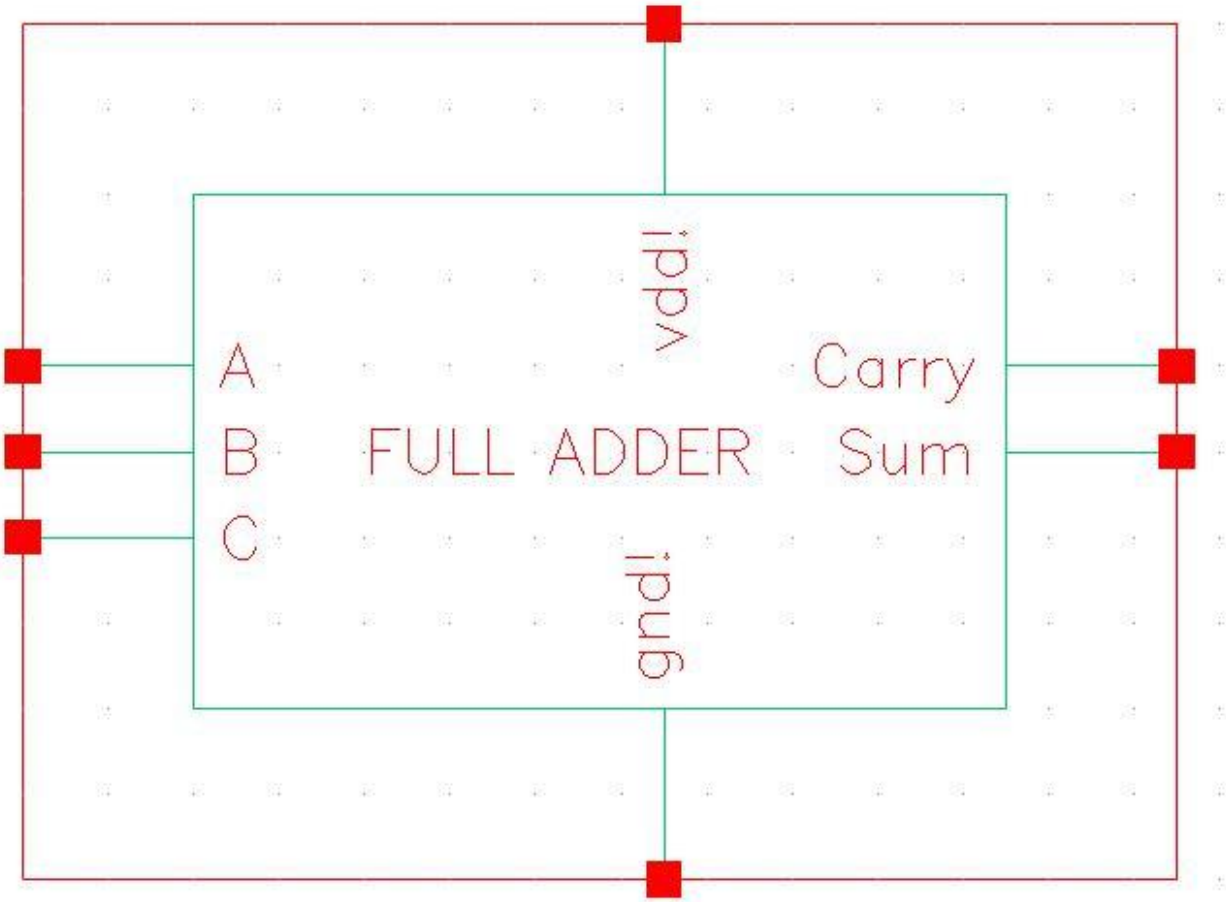


5.8 Adder

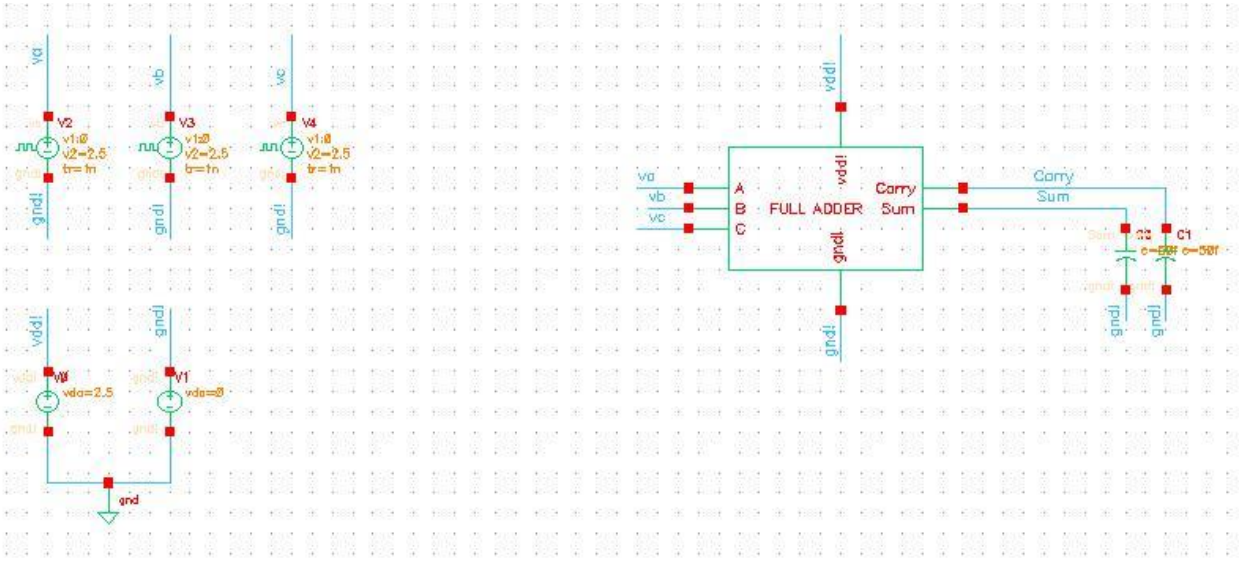
Schematic



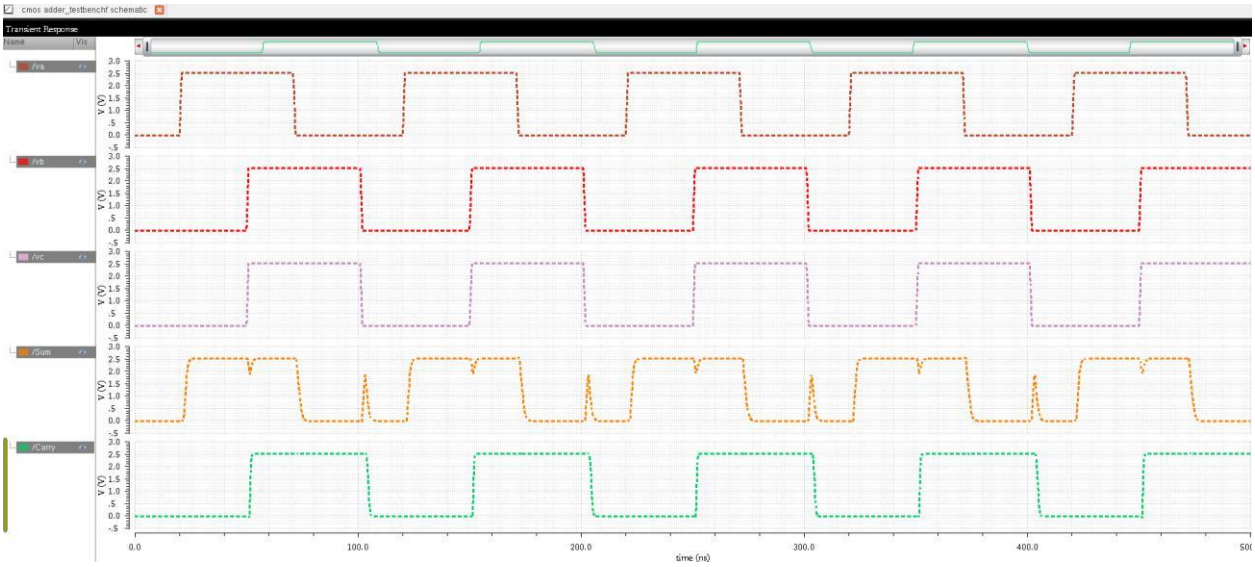
Symbol



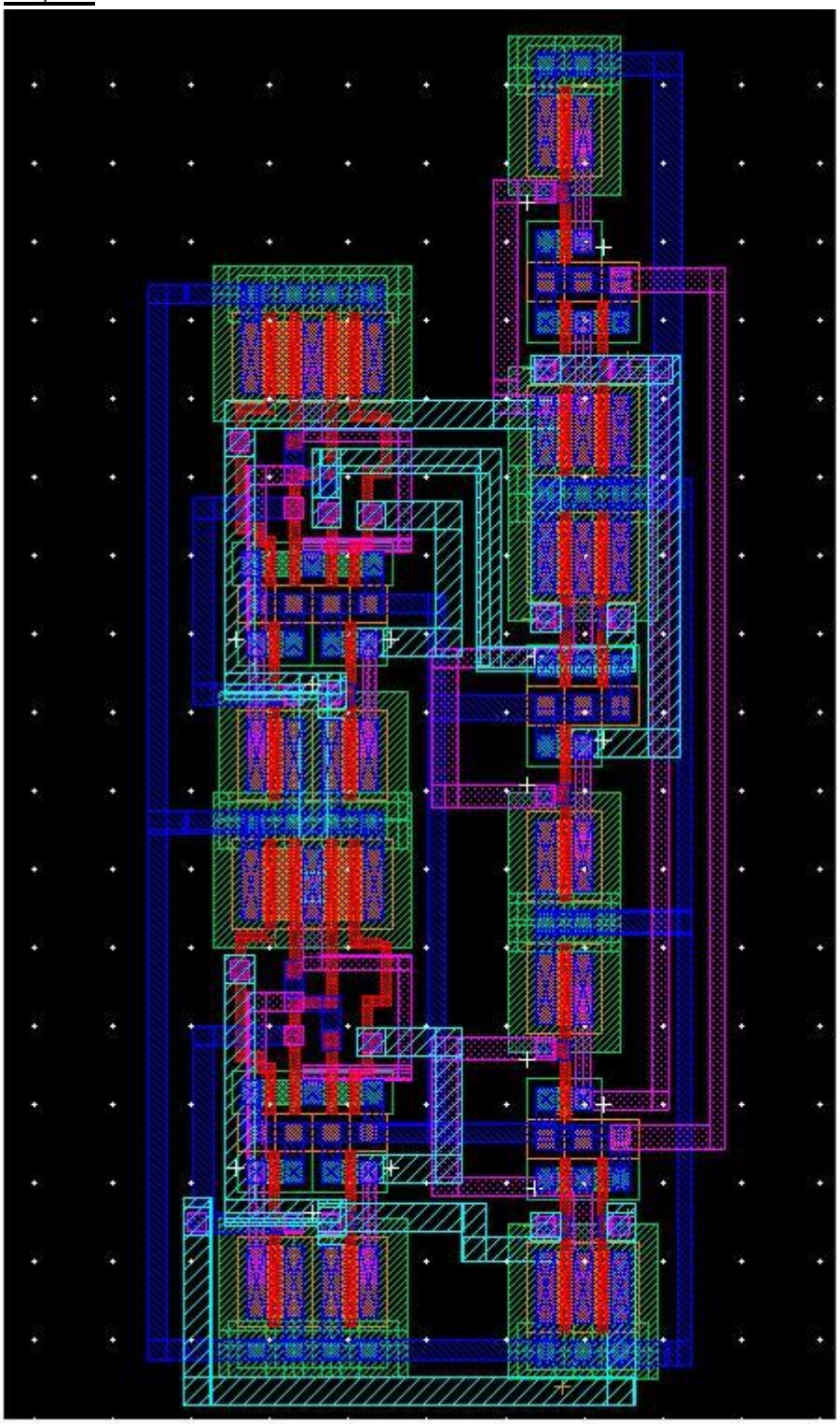
Testbench



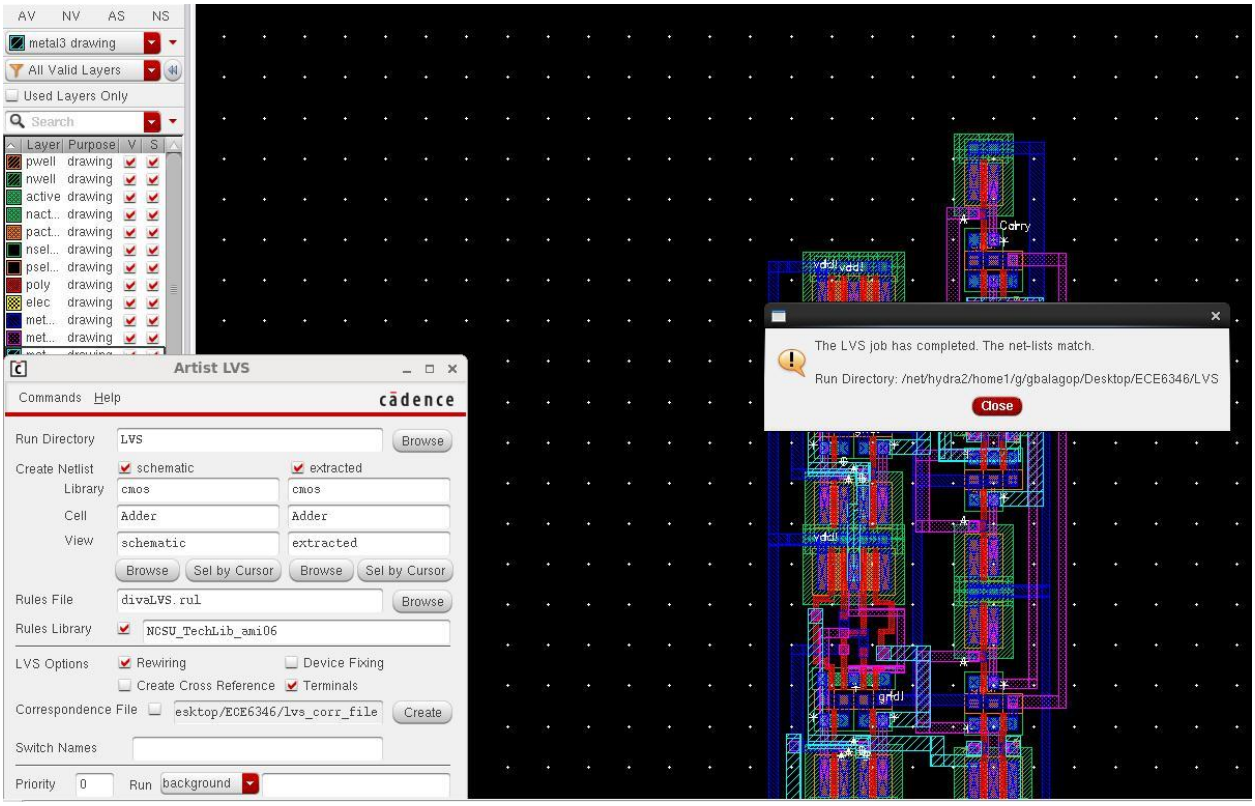
Testbench output wave form



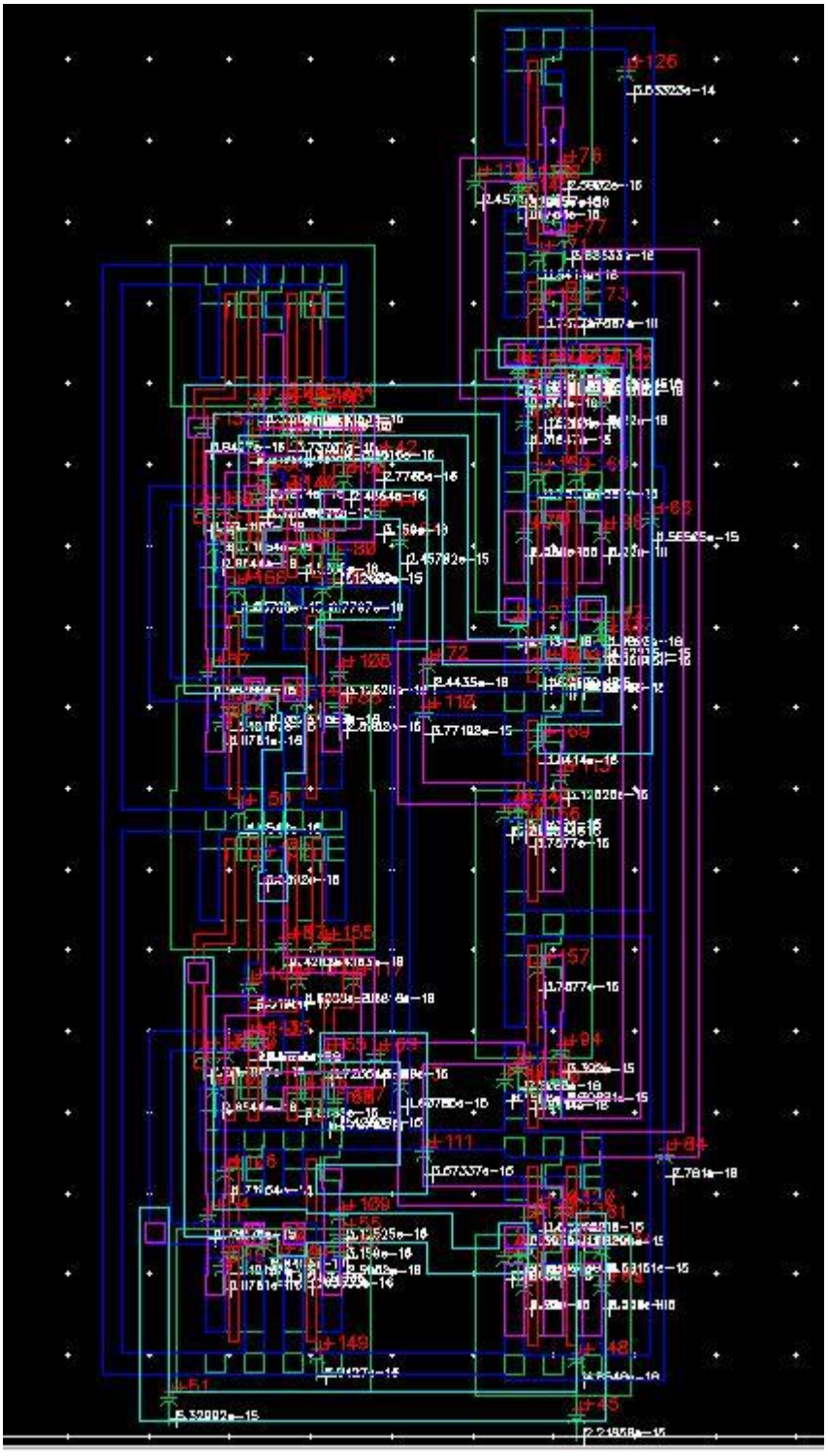
Layout



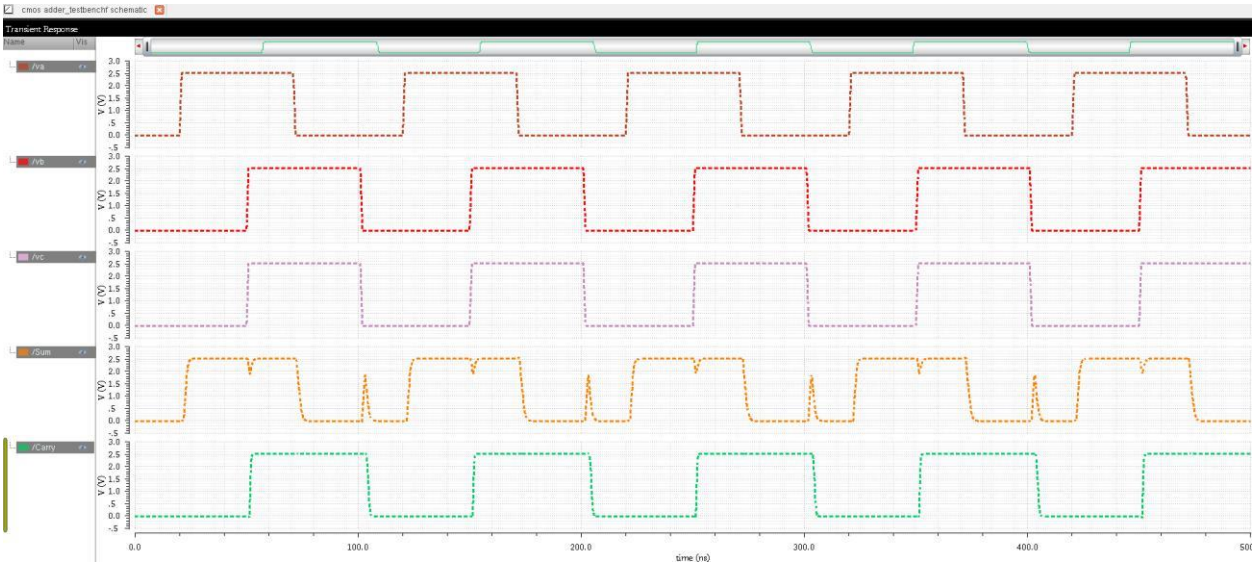
Layout and LVS Verified



Parasitic Extracted

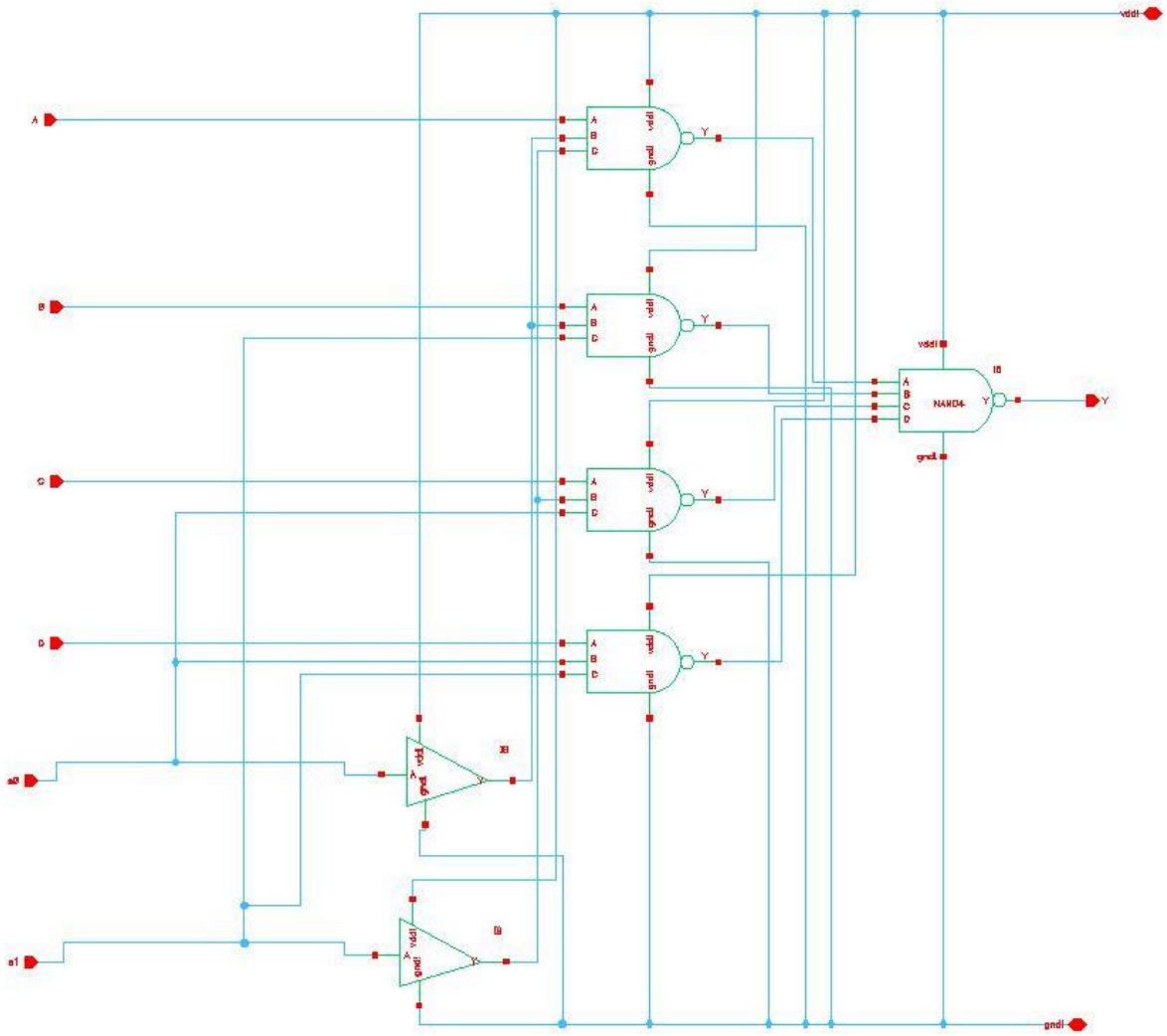


Final output waveform

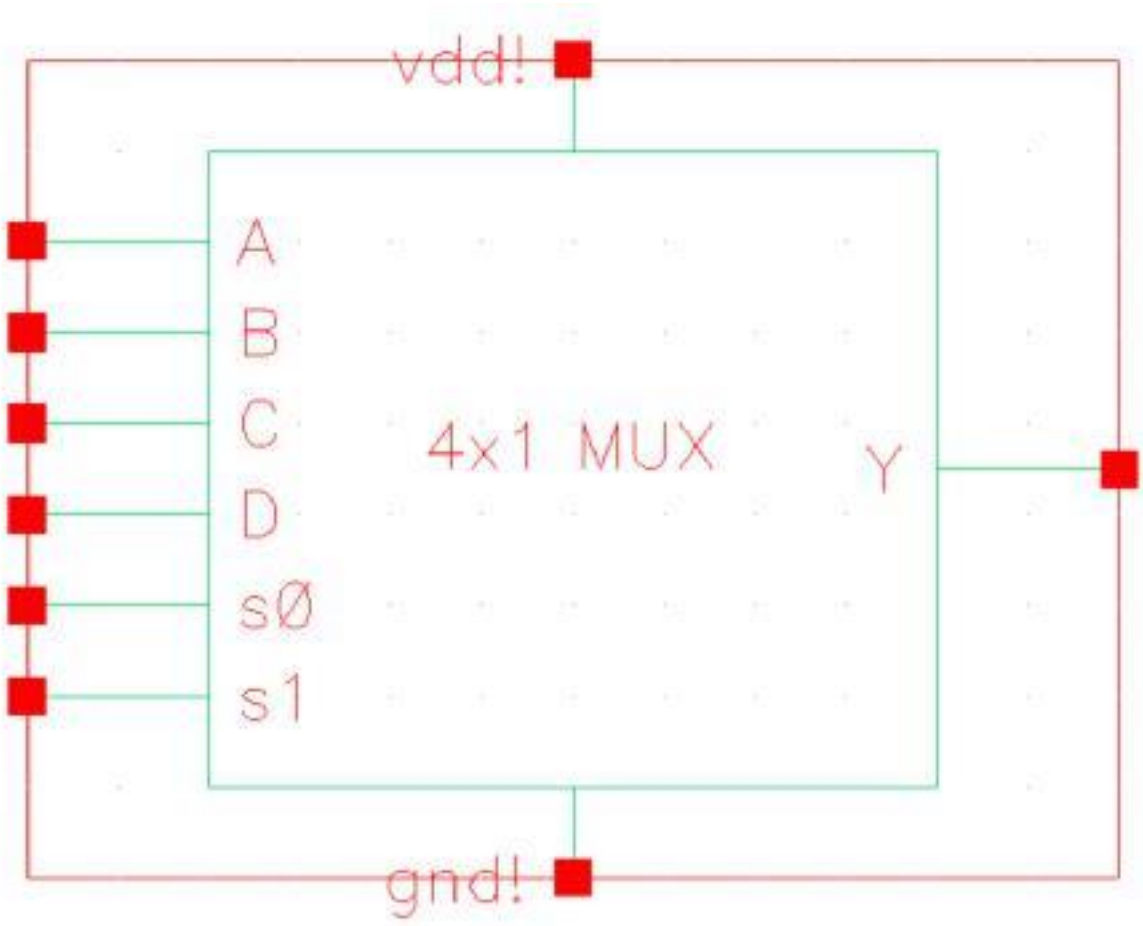


5.9 Mux(4x1)

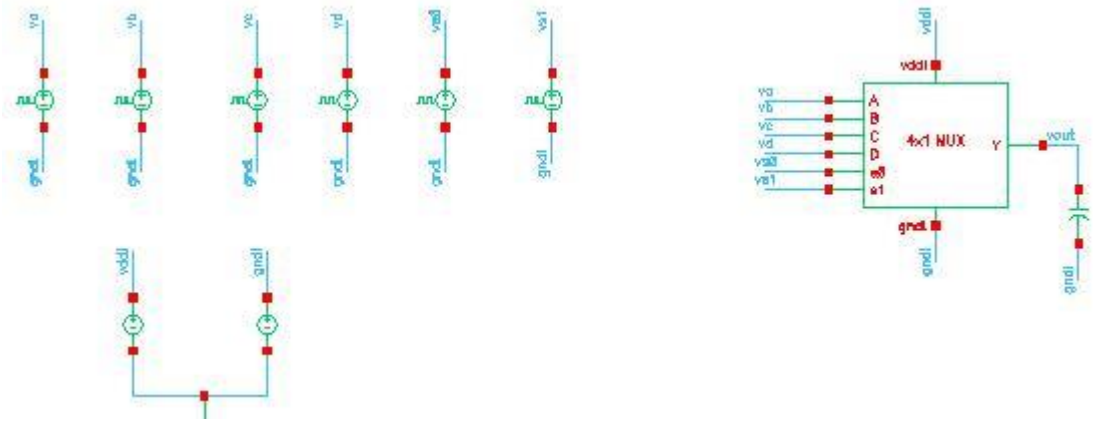
Schematic



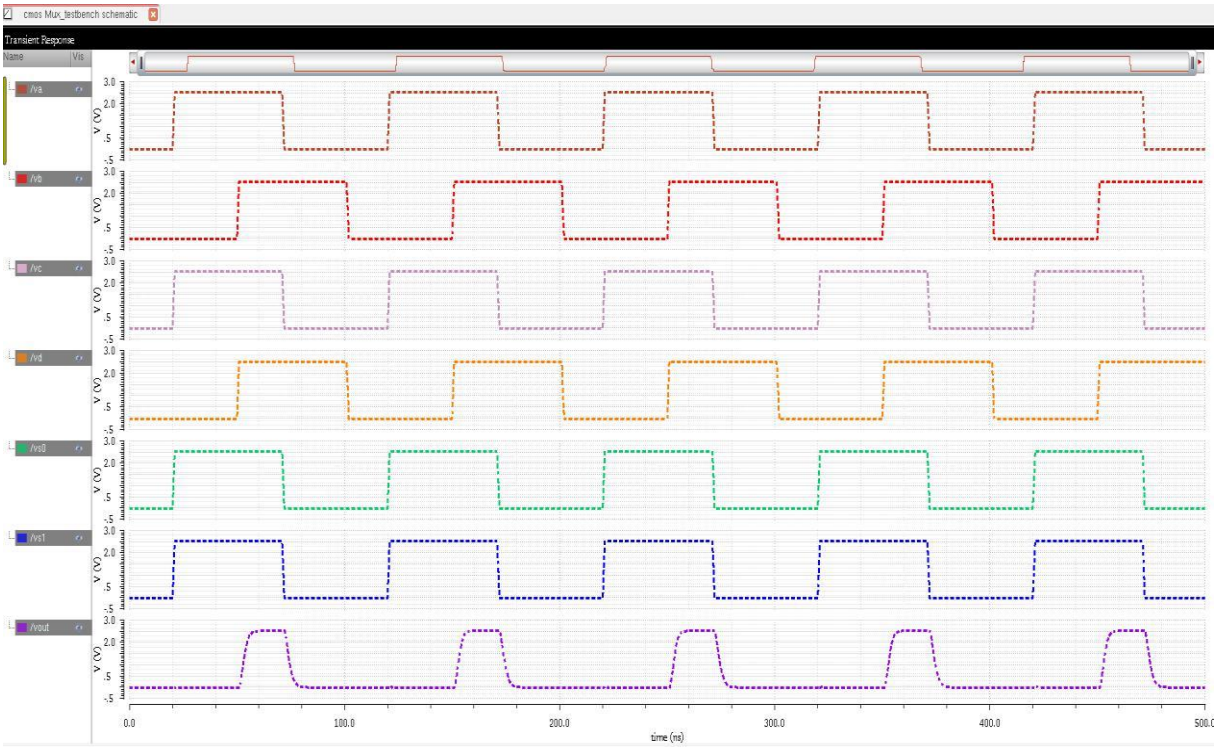
Symbol



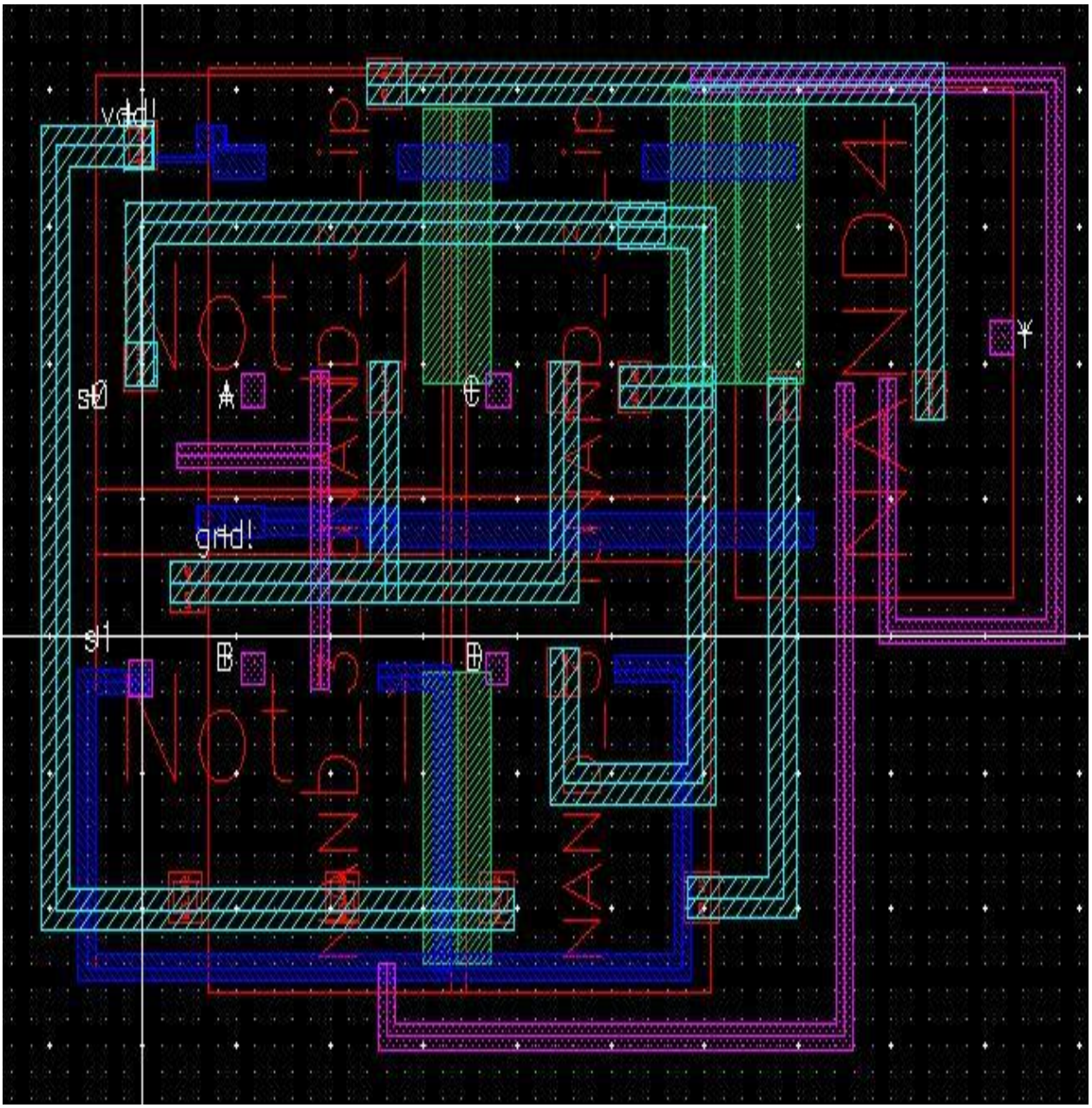
Testbench



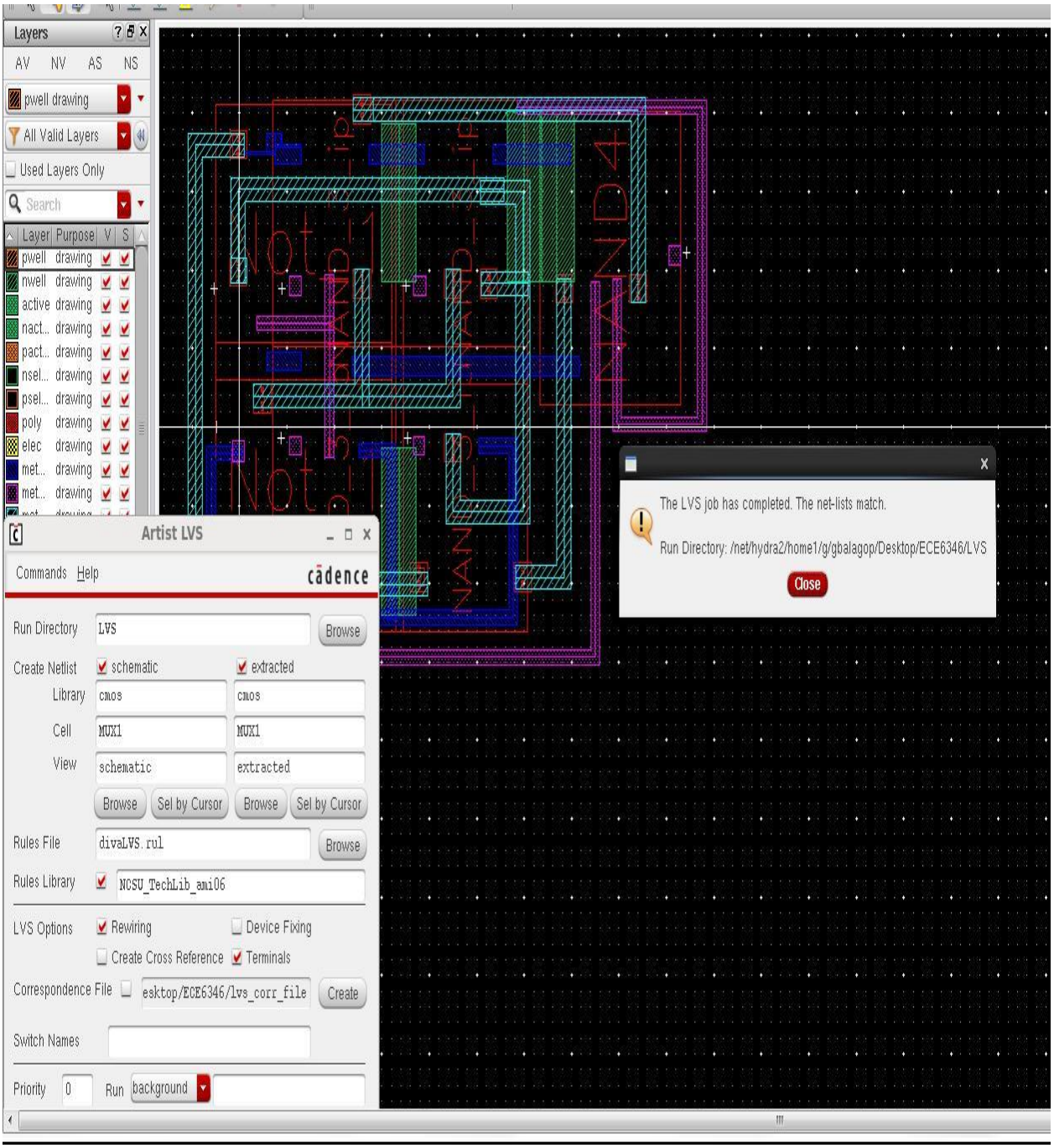
Testbench waveform



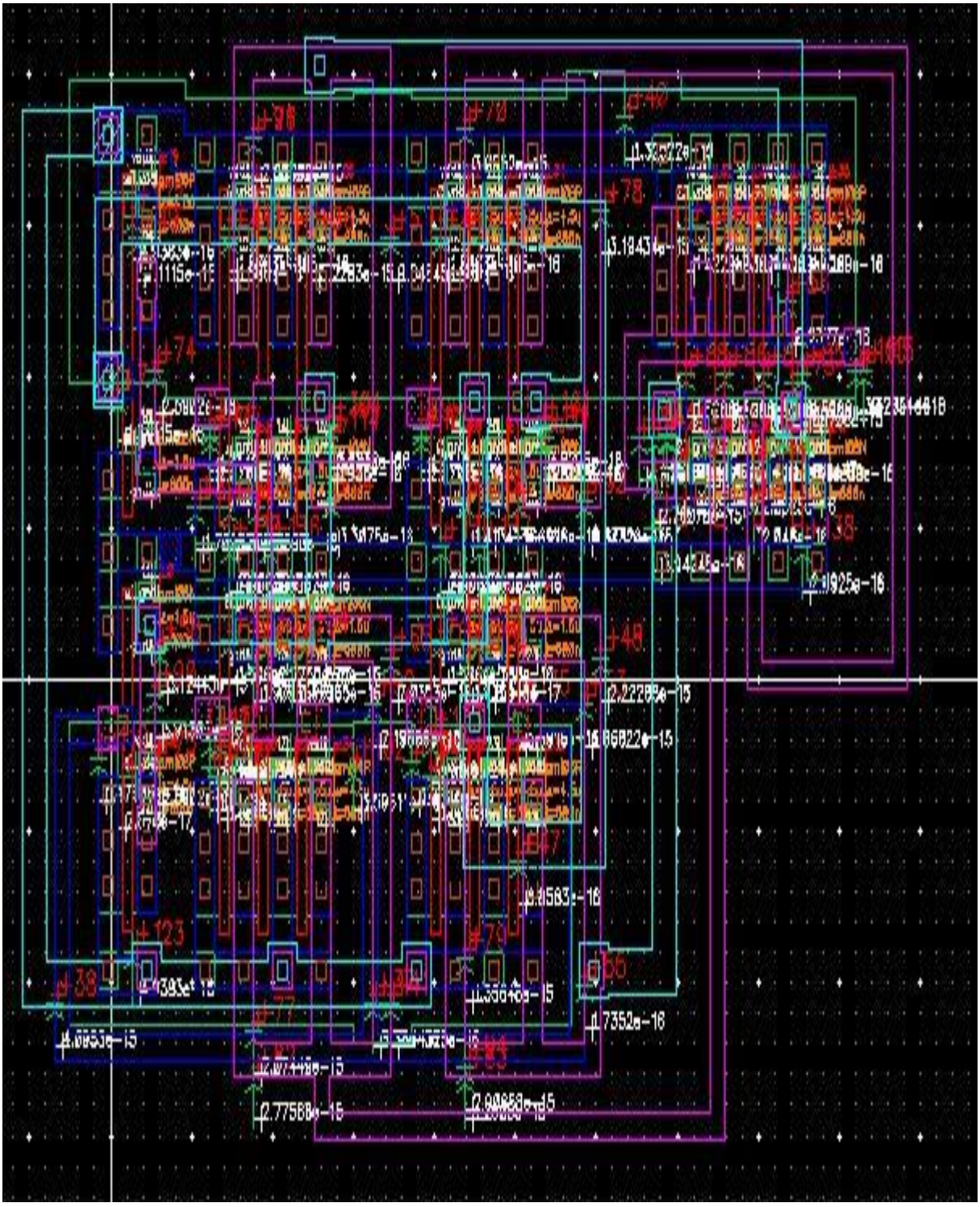
Layout



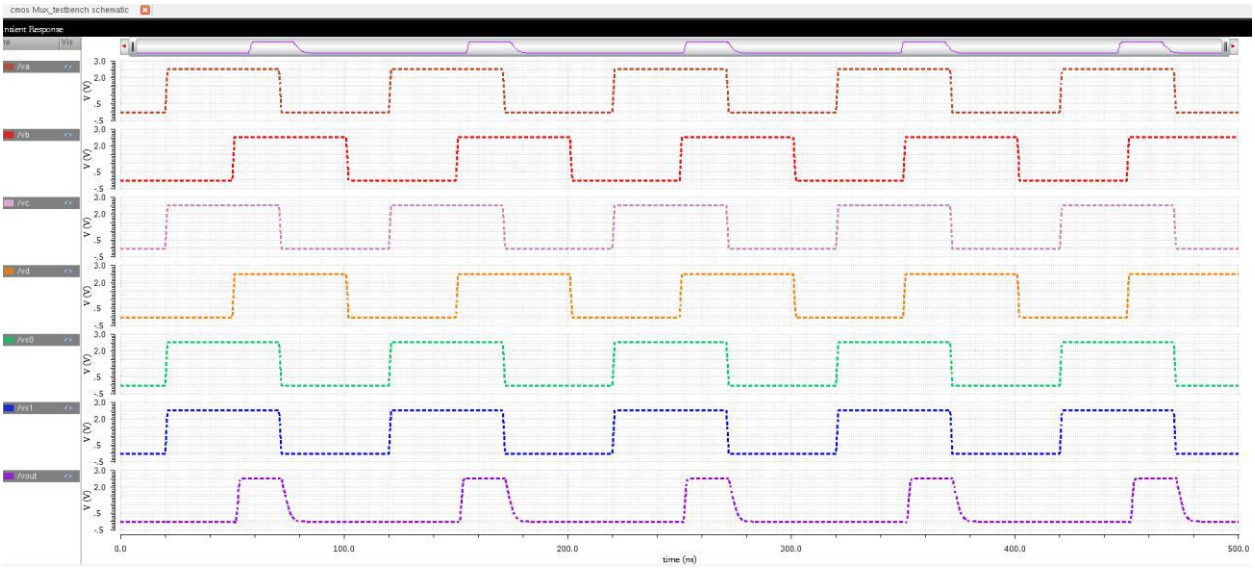
Layout and LVS verification



Parasitic Extracted

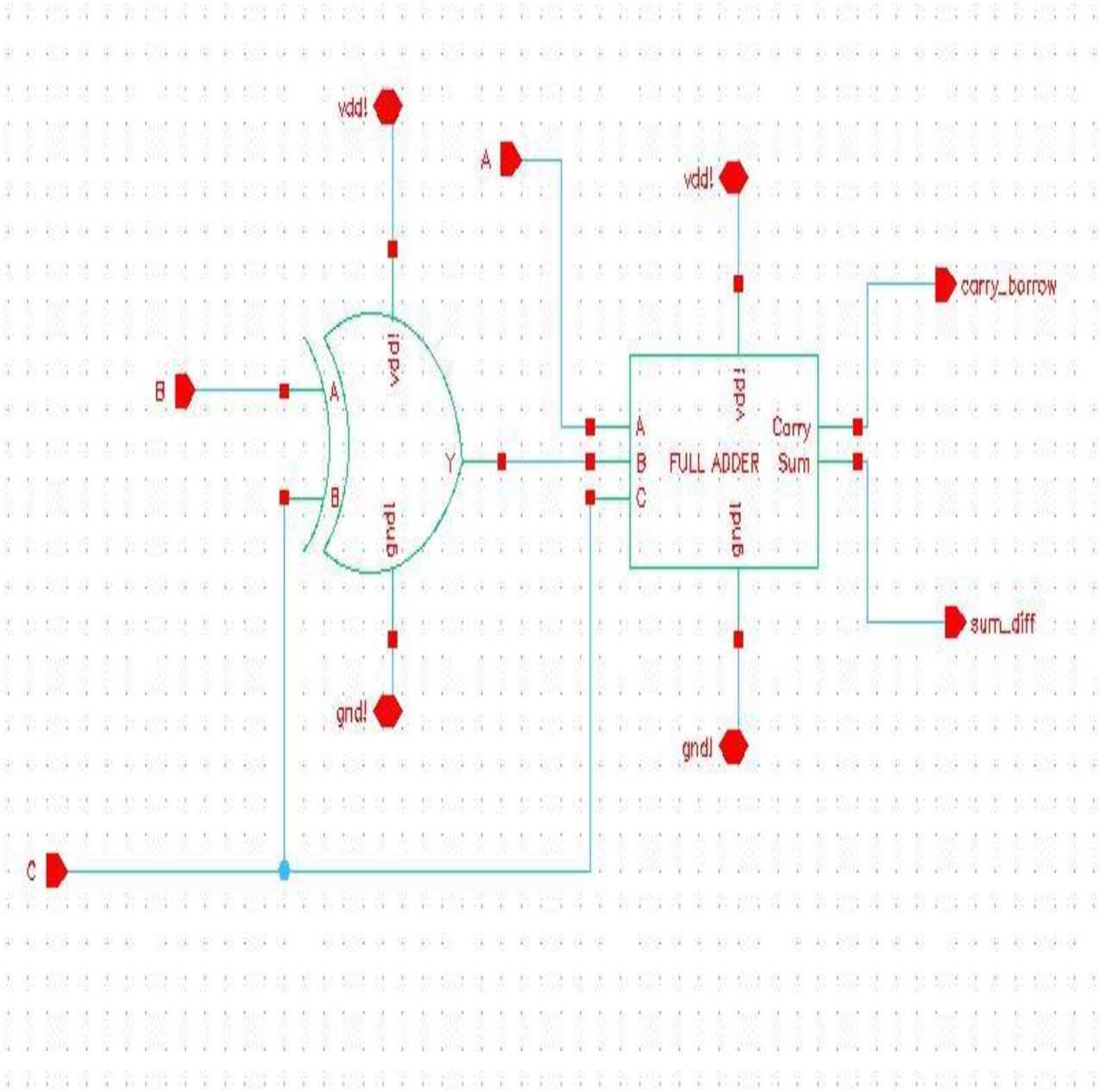


Final output waveform

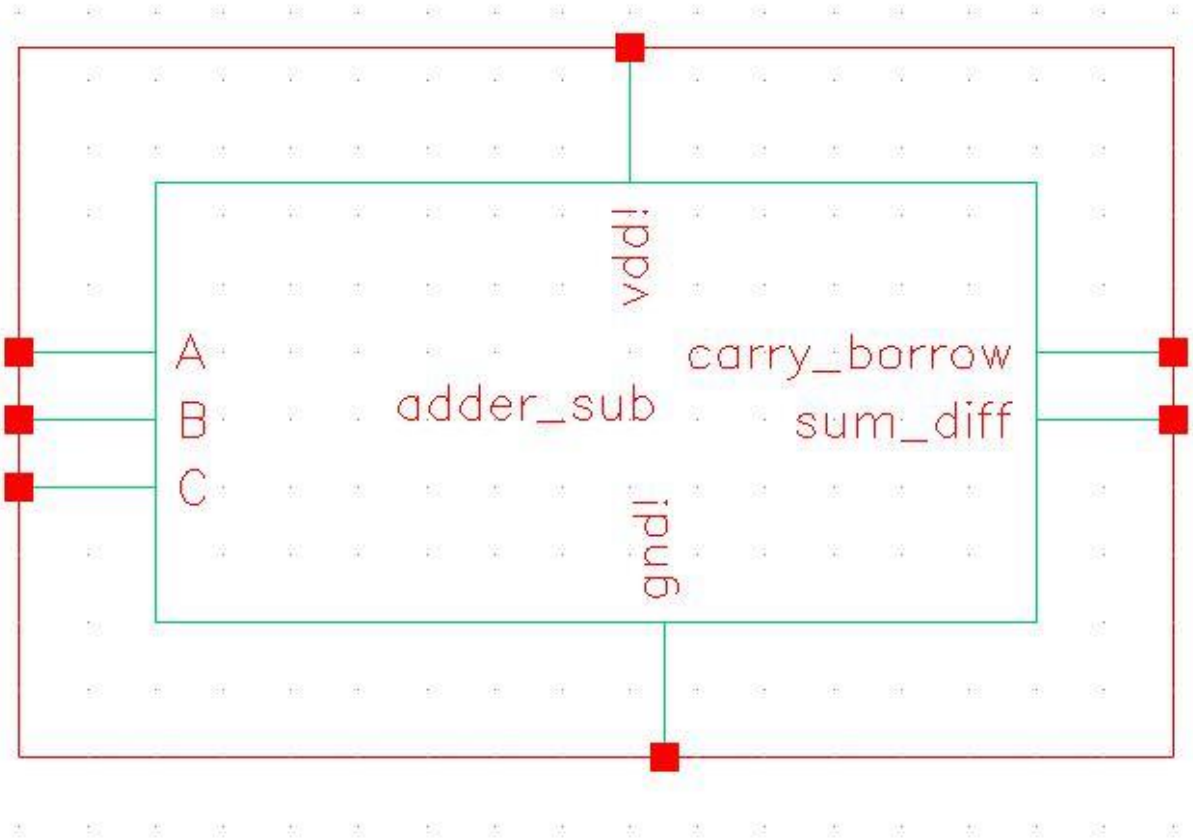


5.9 Adder/Subtractor

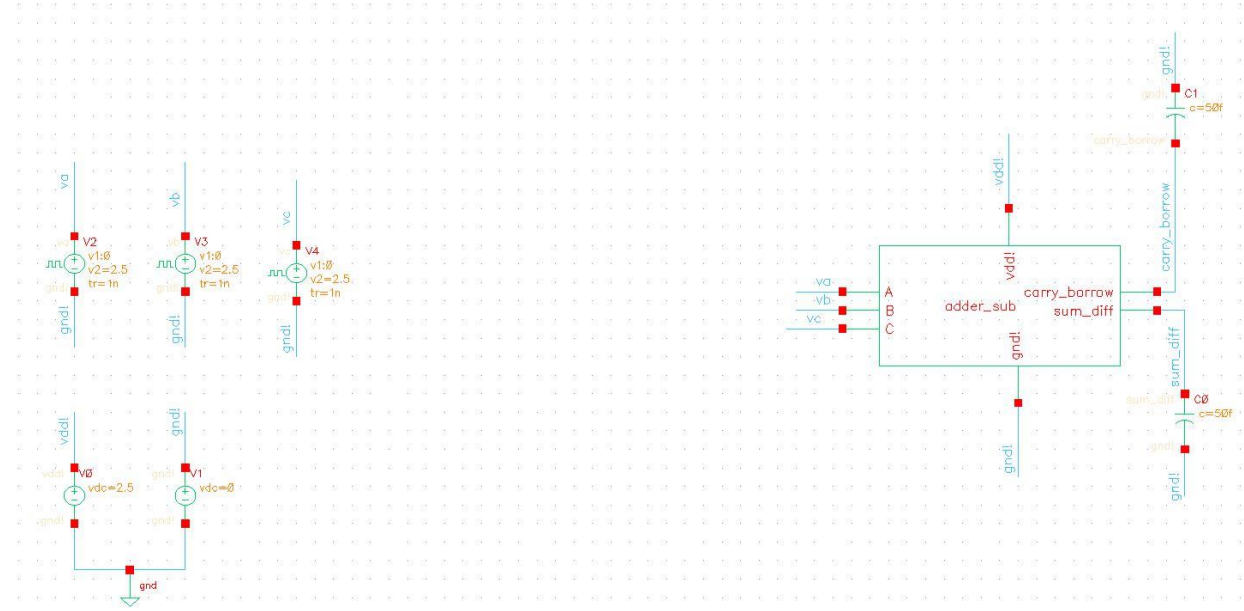
Schematic



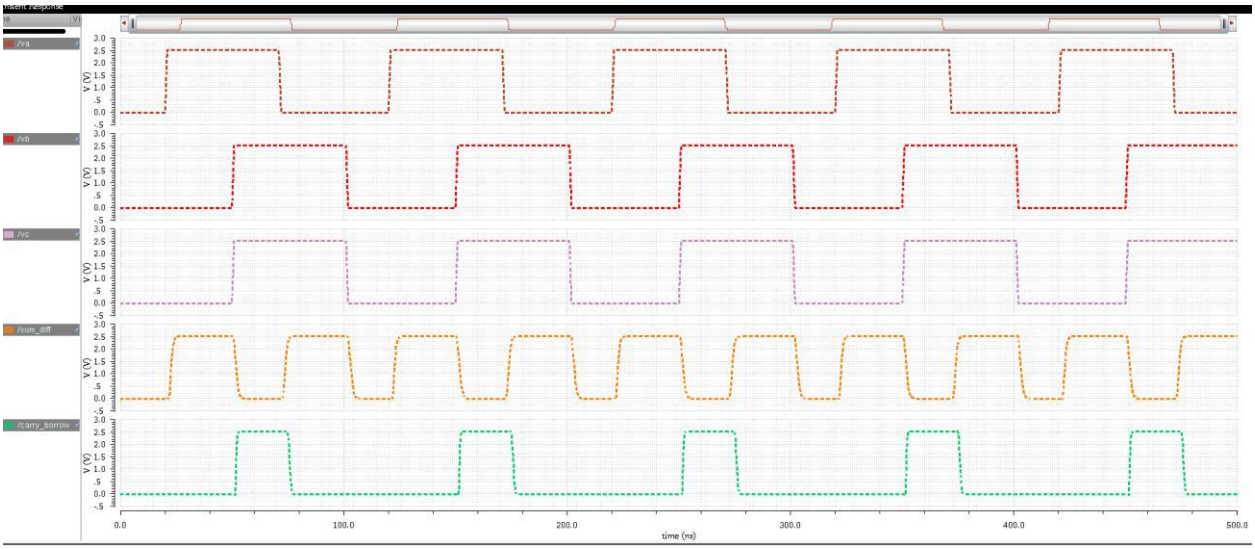
Symbol



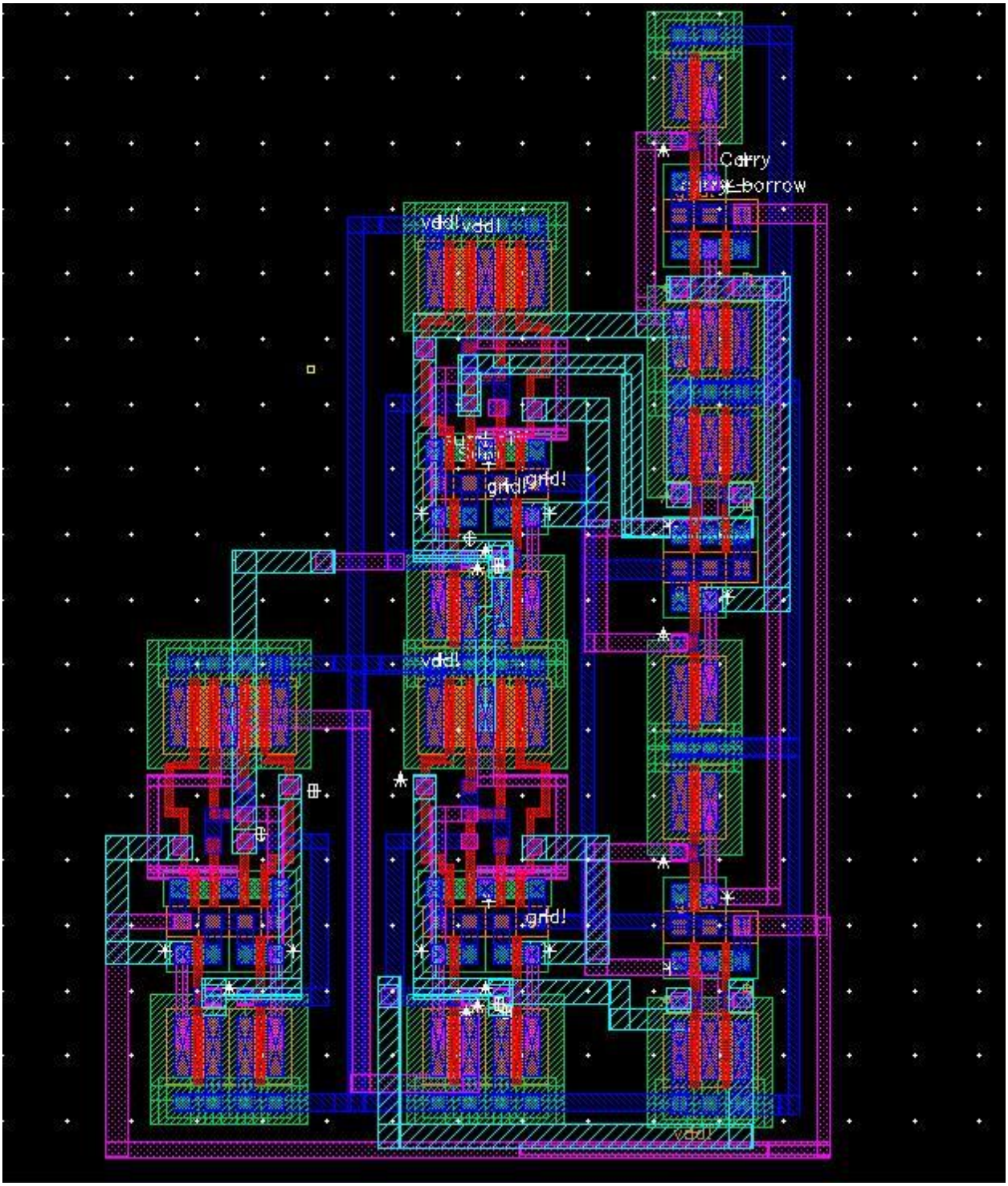
Testbench



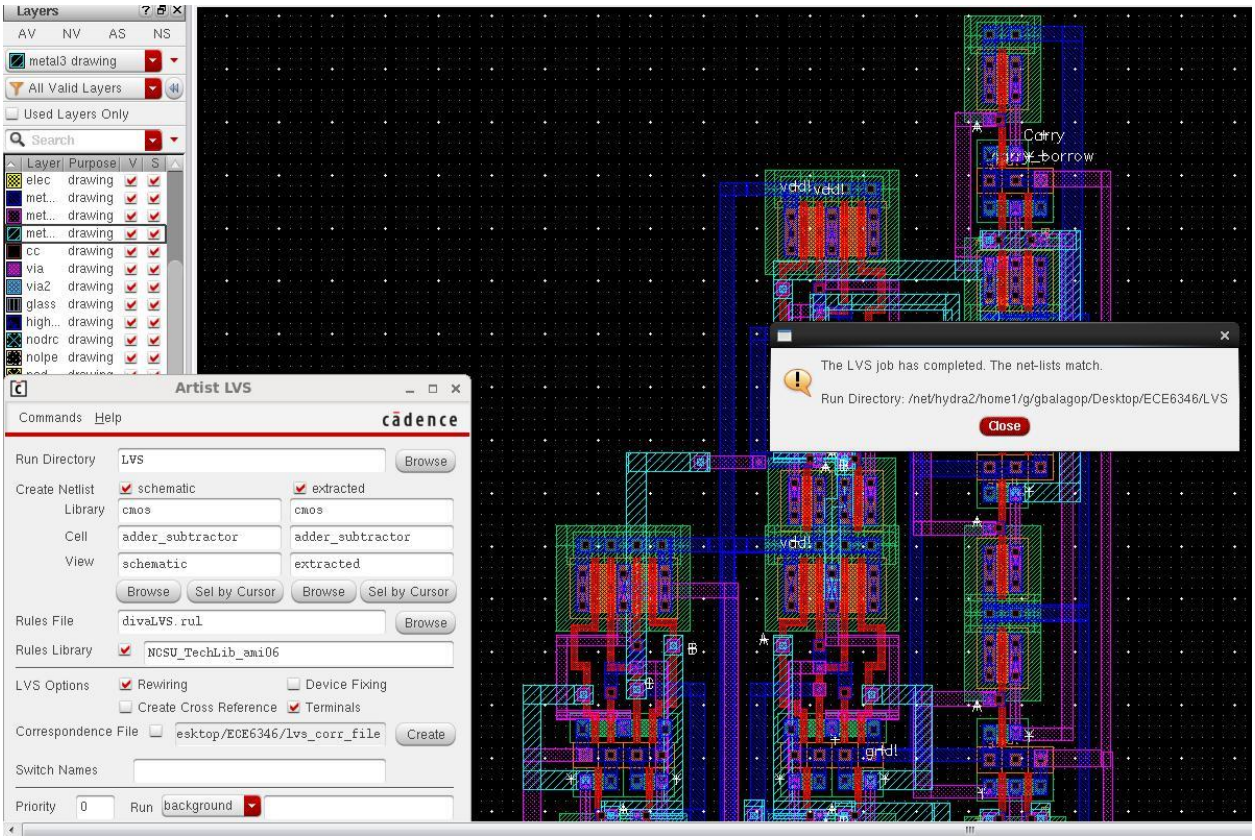
Testbench output waveform



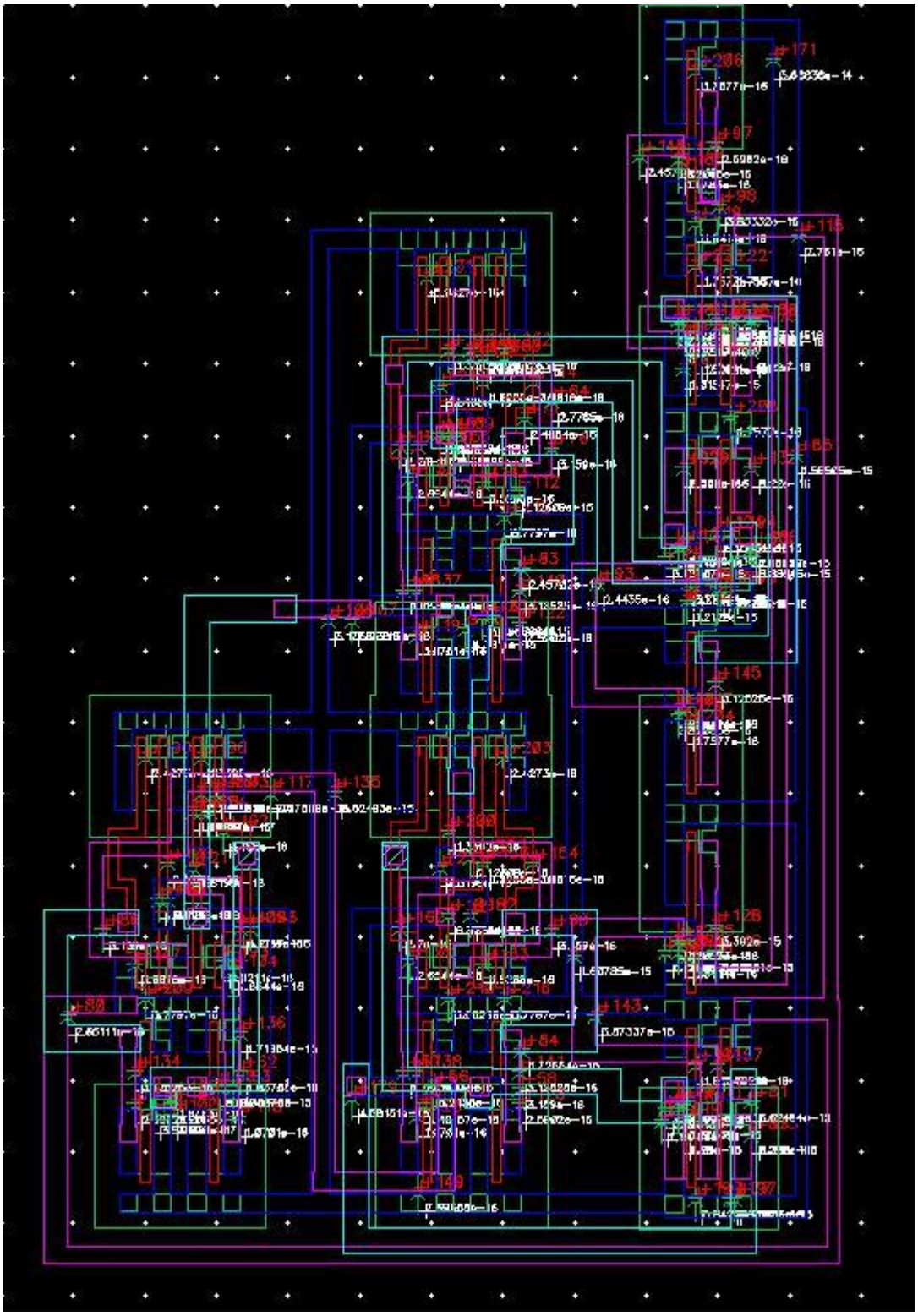
Layout



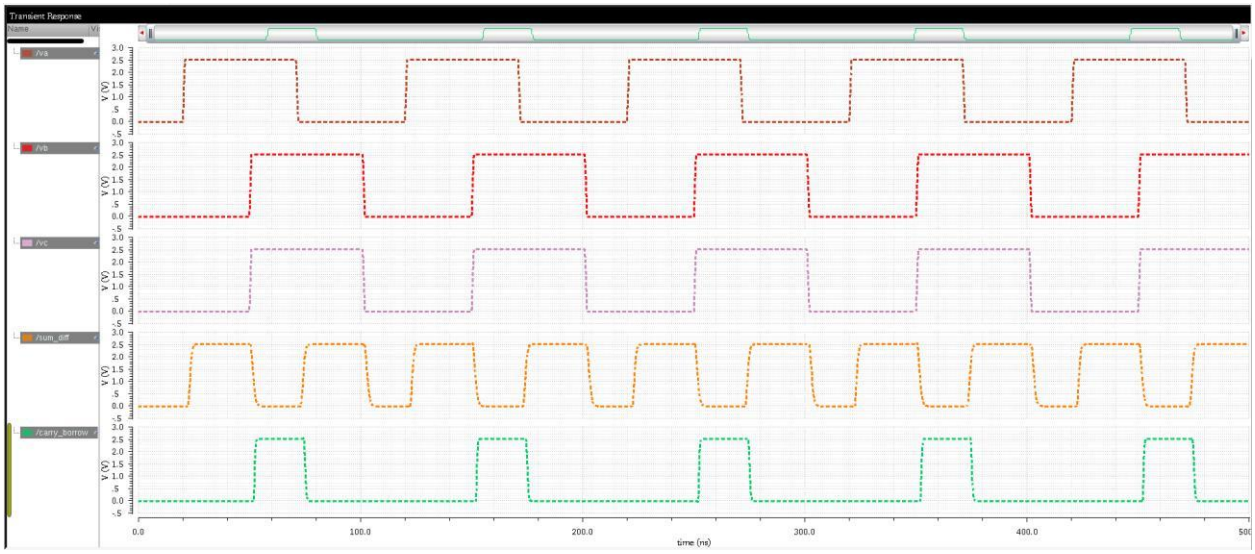
Layout and LVS verification



Parasitic Extracted

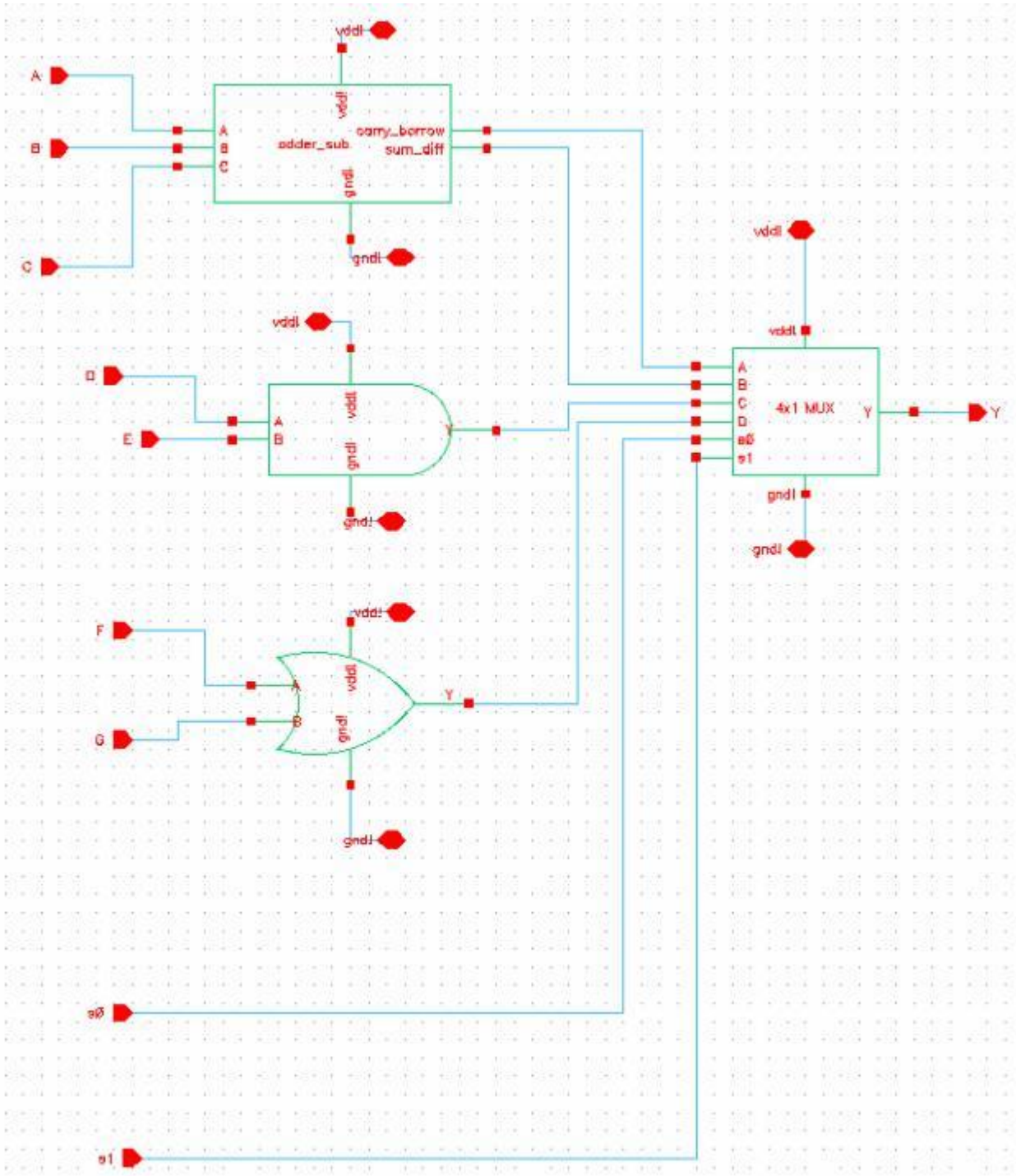


Final Output waveform

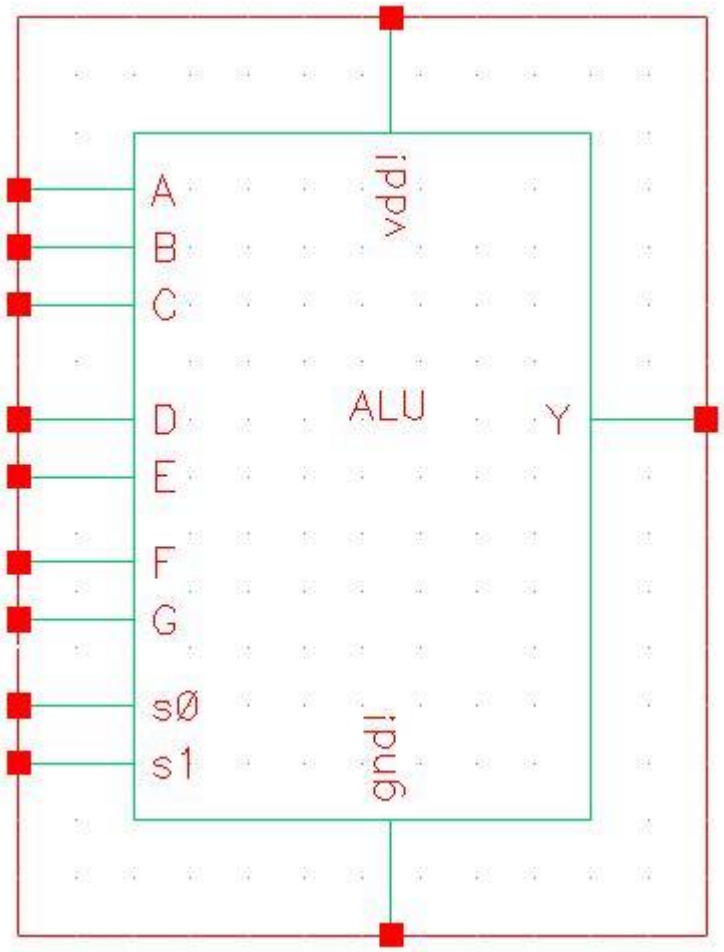


5.10 ALU

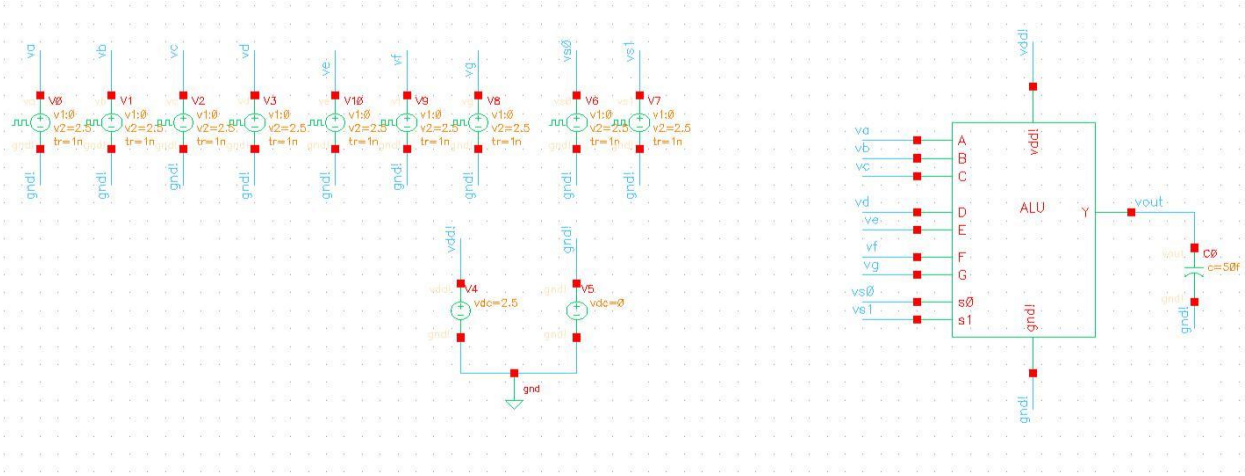
Schematic



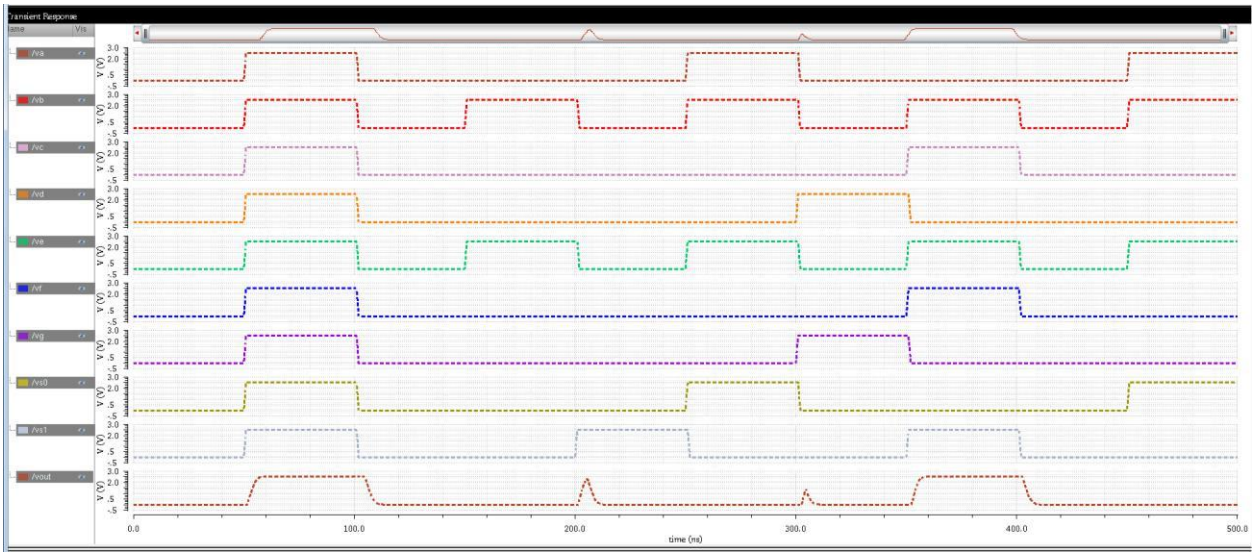
Symbol



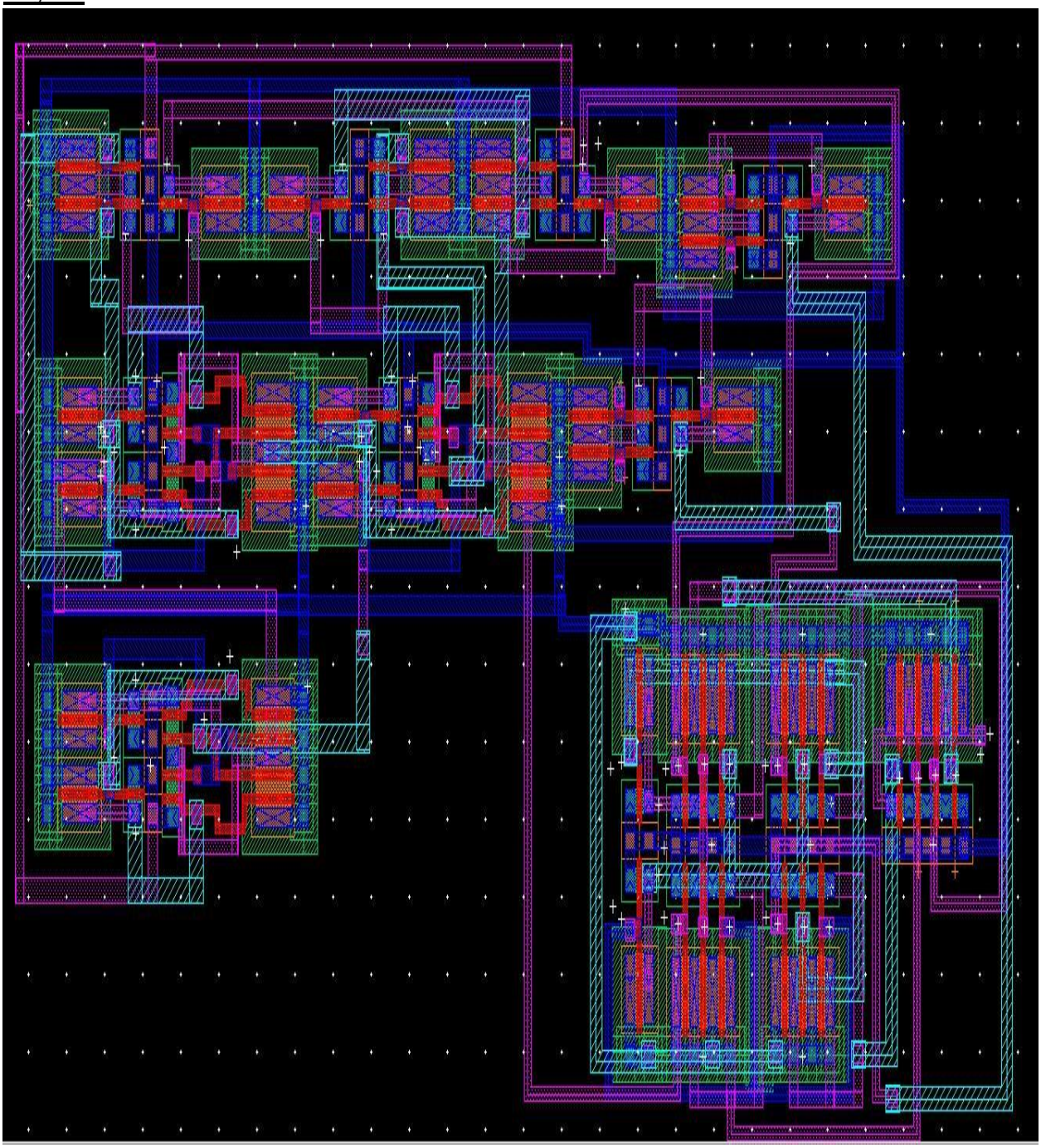
Testbench



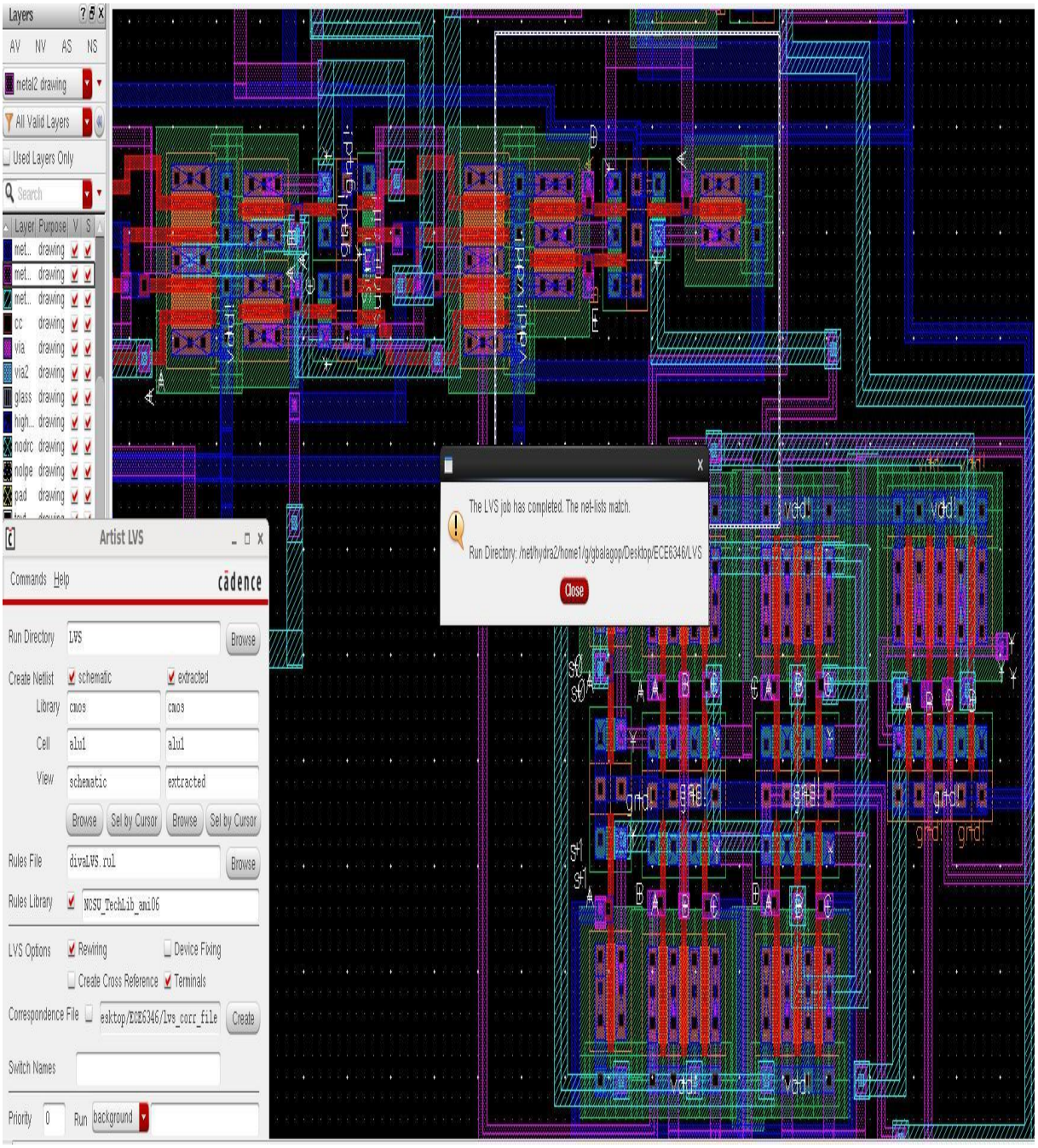
Testbench Output waveform



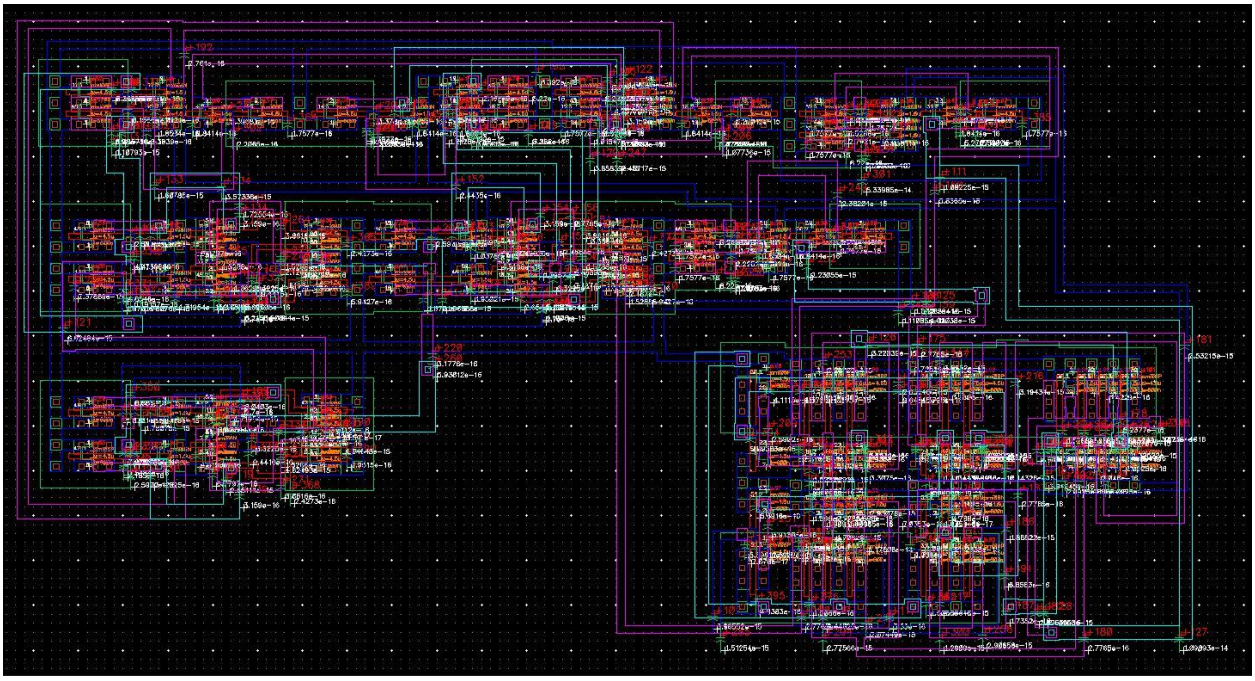
Layout



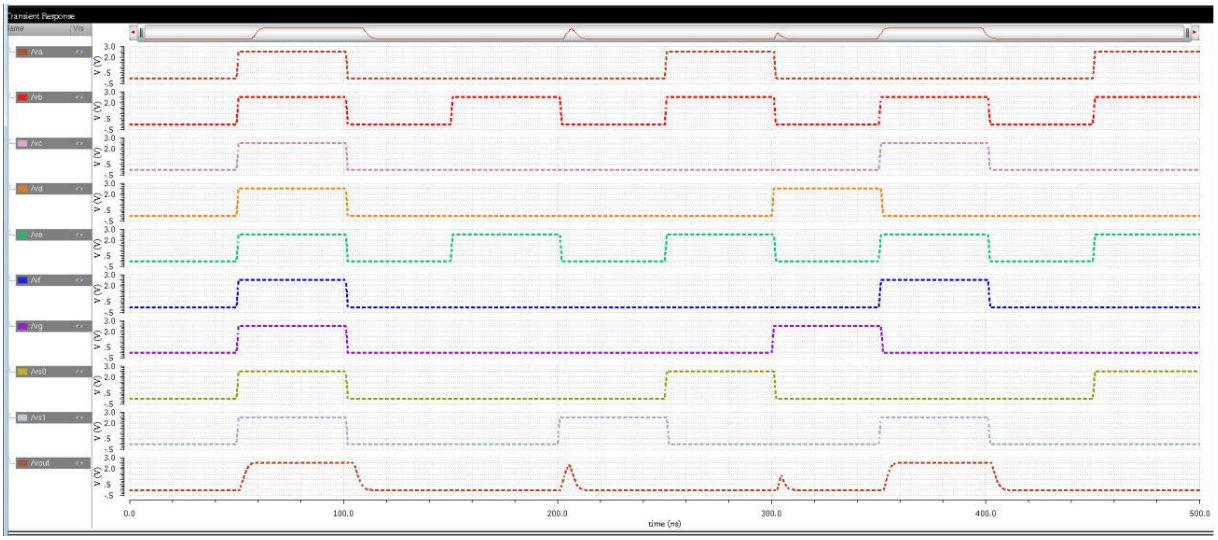
Layout and LVS verified



Parasitics Extracted



Final Output waveform



6. CONCLUSION

Thus the 1 bit ALU with basic arithmetic and logical operations was implemented successfully using Cadence IC Design Tool. All the design rules for implementing the design are satisfied. The design is optimized following stick diagram following Euler graph and designed using the minimum possible number of transistors in order to reduce the power consumption and to increase the performance.

7. ACKNOWLEDGEMENT

This acknowledgment is to thank Prof. Wanda Wosik for constant guidance and support throughout this course. We would like to thank the T.A Mr. Uday Kiran for guiding us through this project. We thank Uh Cullen College of Engineering and Management for providing us the infrastructure required to successfully complete this project