Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 5

Inverter, Ring Oscillator and Clock Driver Design

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016

- **Group no.:** 35 (Monday)
- **Date of Submission:** 16-03-2025

Aim

1. Design a CMOS inverter with equal rise time and fall time in 180nm CMOS technology, ensuring the logic threshold at $V_{dd}/2$.
2. Simulate and observe the transition time and delay (with a fanout of 4) of the designed inverter.
3. Construct a 7-stage ring oscillator using the inverter and observe the oscillation period of the circuit.
4. Design a clock driver for a fanout of 64.

Theory

- CMOS, short for Complementary Metal-Oxide-Semiconductor, is the type of silicon chip electronics technology that has been used in many devices, which handle signal passing in their circuits.
- There are 4 types of CMOS inverter namely conventional CMOs inverted, static CMOS inverter, dynamic CMOS inverter and pseudo-NMOS inverter.
- Working of a CMOS inverter:
  - **Input High (Logic 1)**: An NMOS transistor is turned on by input of high voltage (logic 1) while a PMOS transistor is turned off there. When these two things happen, the output voltage (logic 0) is lowered through reduced resistance path between an output terminal and ground.
  - **Input Low (Logic 0)**: In contrast, when a low voltage (logic 0) is provided to the input terminal, the NMOS transistor switches off and the PMOS transistor conducts. The output voltage (logic 1) rises as a result of the low resistance path that exists between the output terminal and the positive power supply voltage (VDD).
- Dynamic Characteristics of a CMOS Inverter:
  - **Rise Time or tR**: the time for a signal to go from 10% to 90% of its final value.
  - **Peak Time or tP**: Time taken for a response to reach maximum value.
  - **Fall Time or tF**: the time taken for 90%-10% drop in the signal depending on its value.

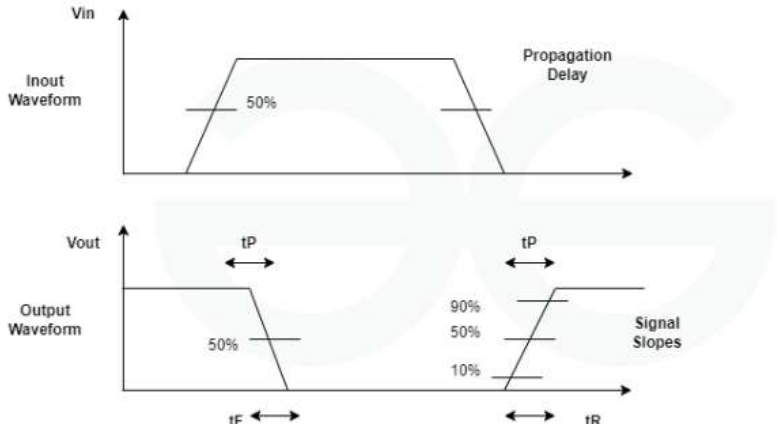


Figure 1: Dynamic Characteristics of CMOS Inverter

- The ring oscillator is one of the most simple, effective, and reliable oscillatory topologies, and it is widely implemented in the industry.The first principle of the ring oscillator is logical instability. By looking at figure below, we can see that the signal is being inverted by the first NOT port, so the output of the second is equal to . The process repeats in the second and third inverter, resulting in an overall output signal equal to . By feeding back this signal to the input, it becomes a combinatory contradiction, as , so the circuit is logically unstable.

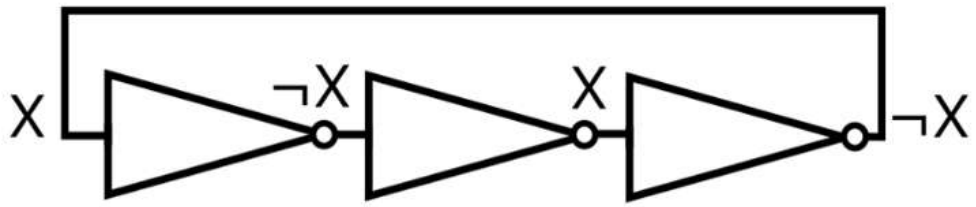


Figure 4: Logical Instability

Figure 2: Instability in Ring Oscillator

- Suppose the dealy of each of the inverters is t_p. Then the time period of the ring oscillator will be:

$$T_p = 2 * n * t_p$$

  where n denotes the number of stages in the ring oscillator.

- It is to be noted that to maintain instability the number of inverters to be used in the ring oscillator should be odd.

- **Fanout** refers to the maximum number of digital inputs that a single output can drive without degrading the signal's quality. It indicates the driving capability of a logic gate.

Part 1: CMOS Inverted Design

Design

We chose the following design parameter:

- $(W/L)_p/(W/L)_n = 1.6u/0.42u$
- $V_{dd} = 1.8V$
- $V_{pulse} = 1.8V$ with $TimePeriod = 100ns$ $PulseWidth = 50ns$ $RiseTime = 10ns$ $FallTime = 10ns$ $DutyCycle = 50\%$
- Load Capacitance of $C_L = 100fF$

Schematic

The following is the schematic of the CMOS inverter:

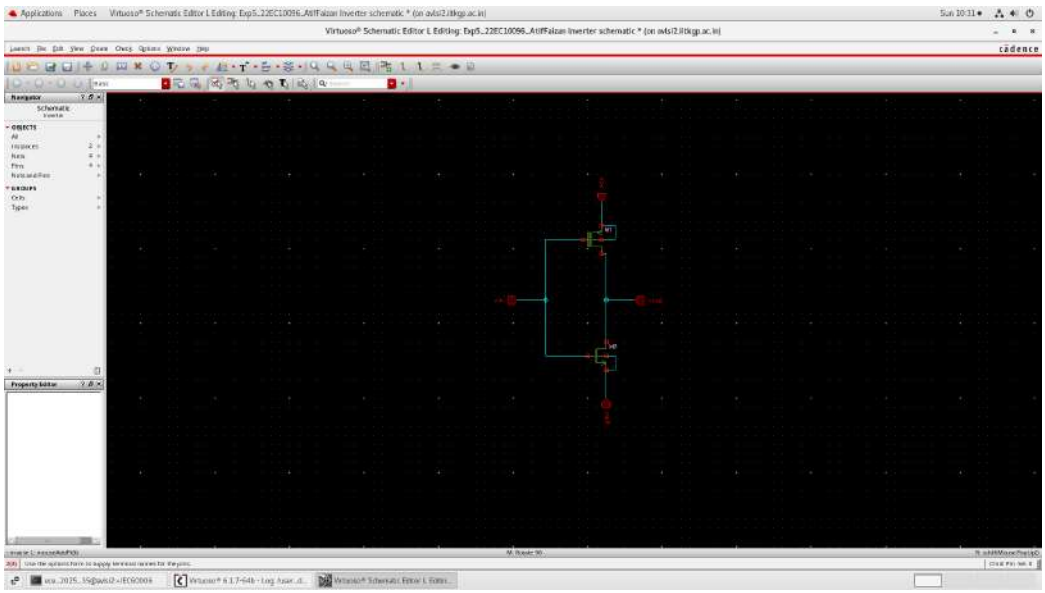


Figure 3: CMOS Inverter Schematic

The following is the symbol creation of the CMOS inverter:

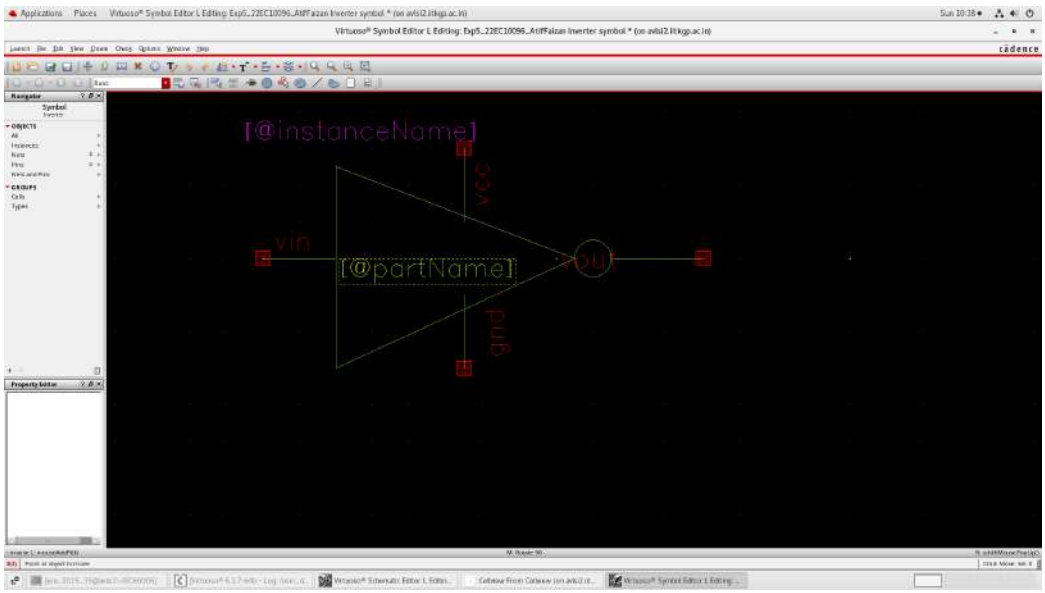


Figure 4: CMOS Symbol

The following is the testbench:

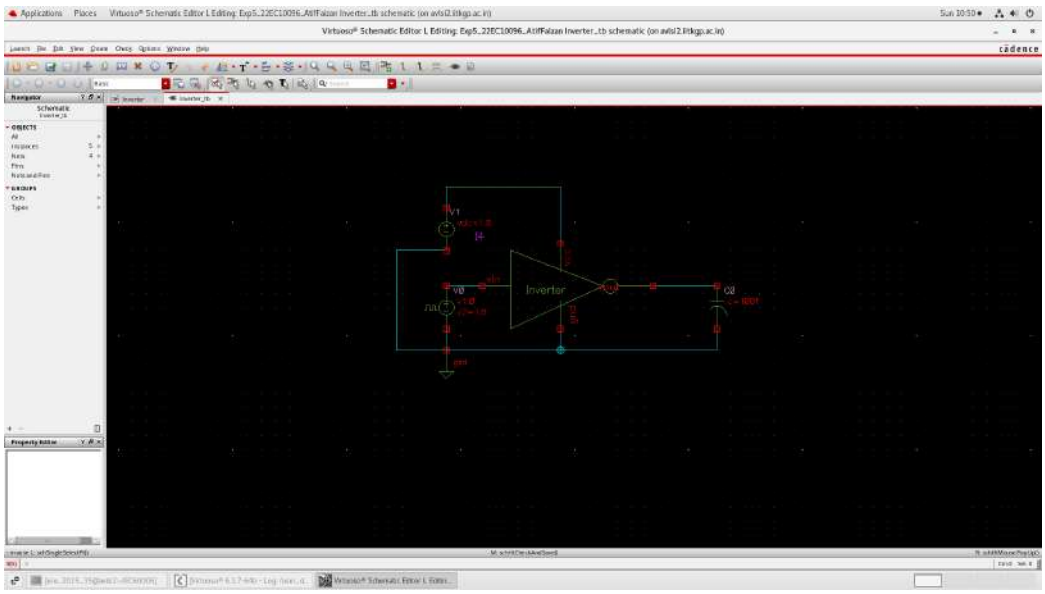


Figure 5: Test Bench

Simulation Results

The following is the simulation result of the testbench:

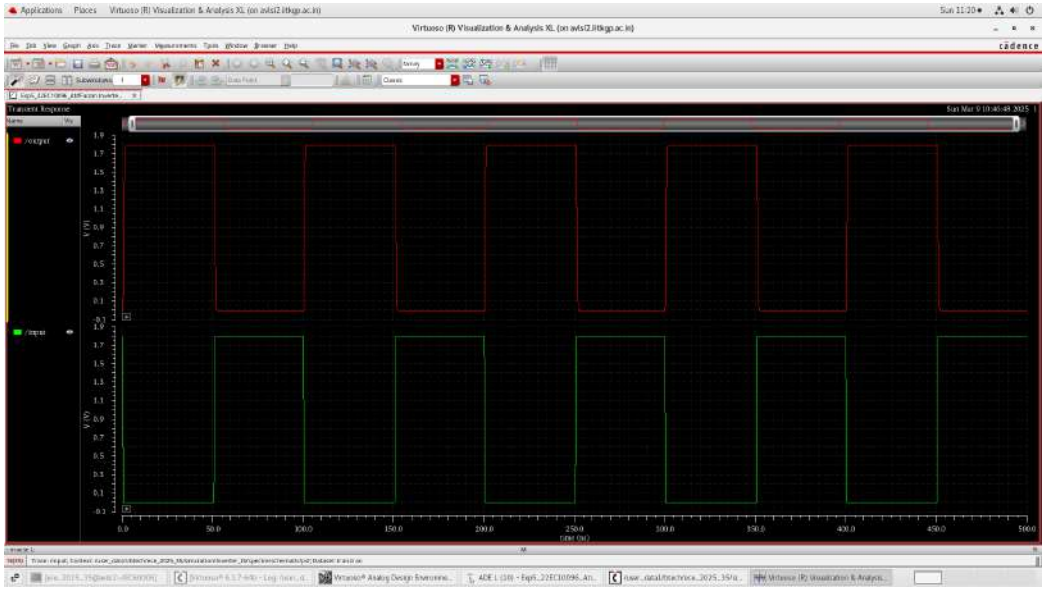


Figure 6: Simulation Results

The following is the Rise Time Observed:

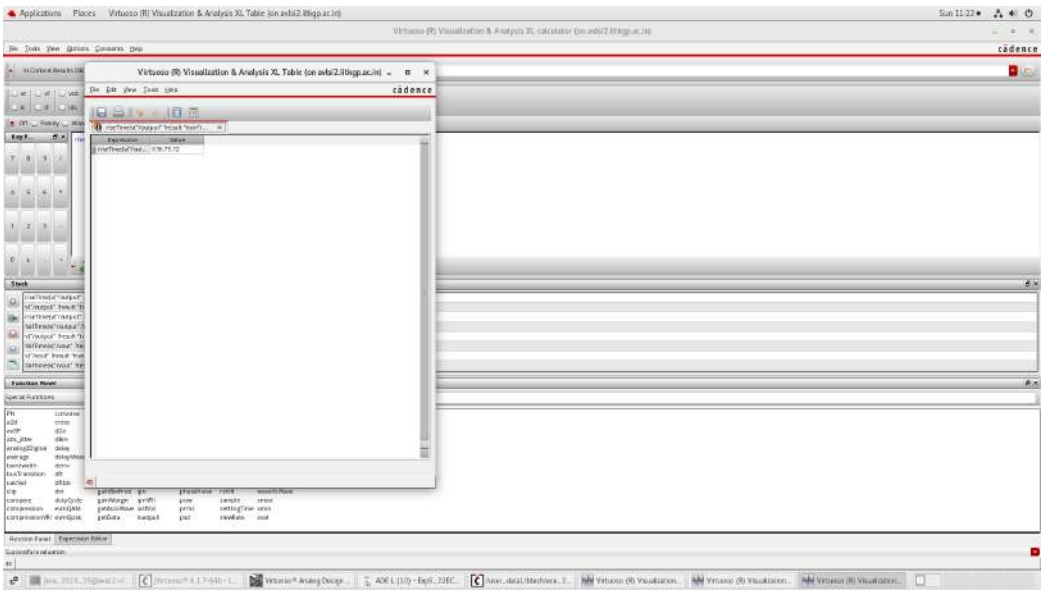


Figure 7: Rise Time

The following is the Fall Time Observed:

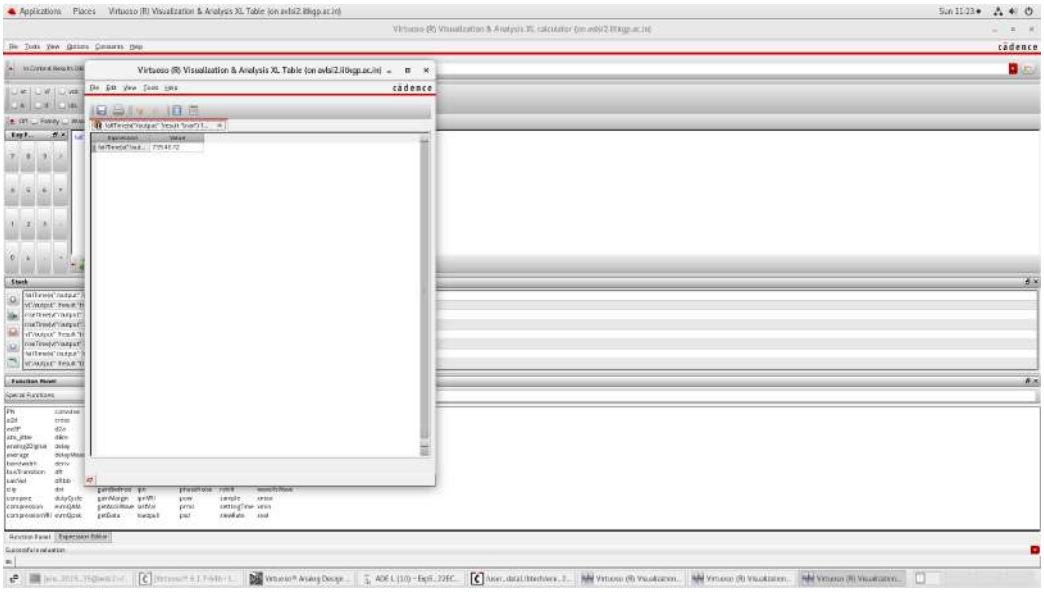


Figure 8: Fall Time

The following is the VTC observed:

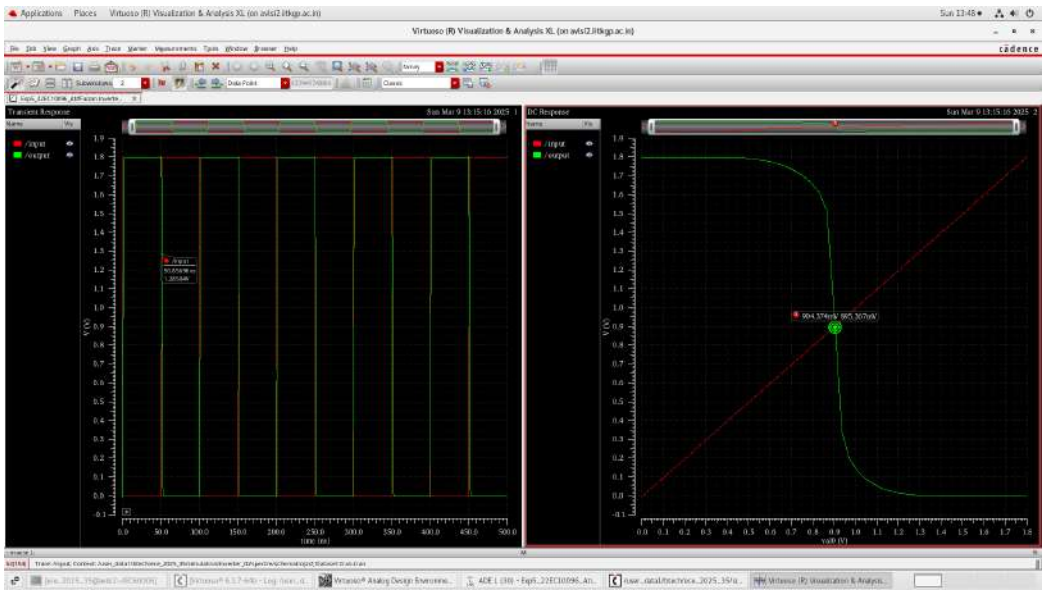


Figure 9: VTC of Inverted

The following are the properties set for the NMOS:

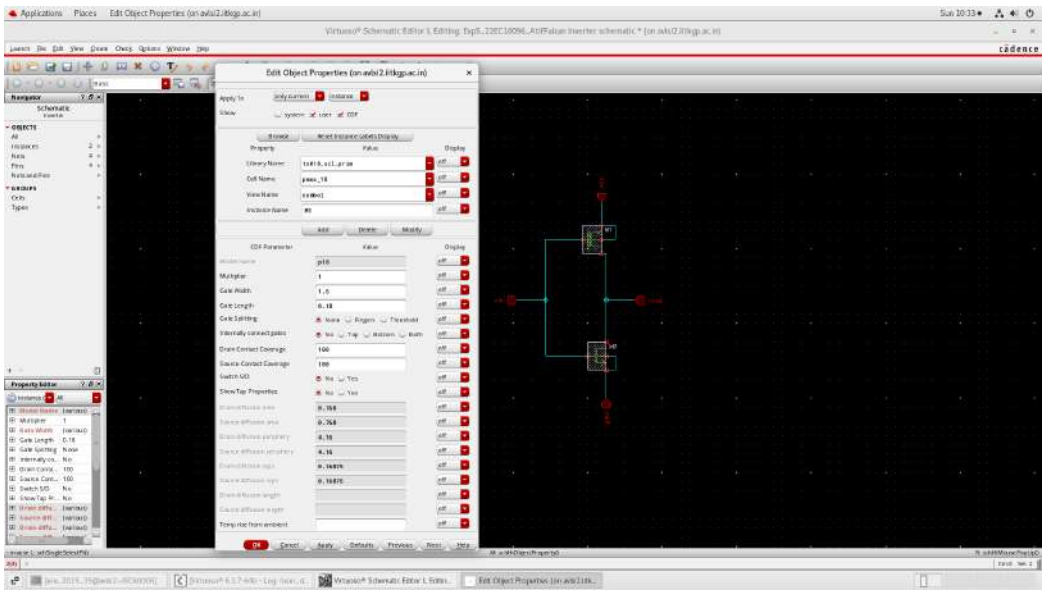


Figure 10: NMOS Properties

Observations

We observed the following things:

- Rise Time = 636.7 ps
- Fall Time = 739.4 ps
- Logic Threshold Voltage = 940.374 mV

We can clearly see that the rise time and fall time are approximately same and the logic threshold voltage is approximately $V_{dd}/2$

Part 2 Inverter with Fanout 4

Design

We used $C_L = 100fF$

Schematic

The schematic is as follows:

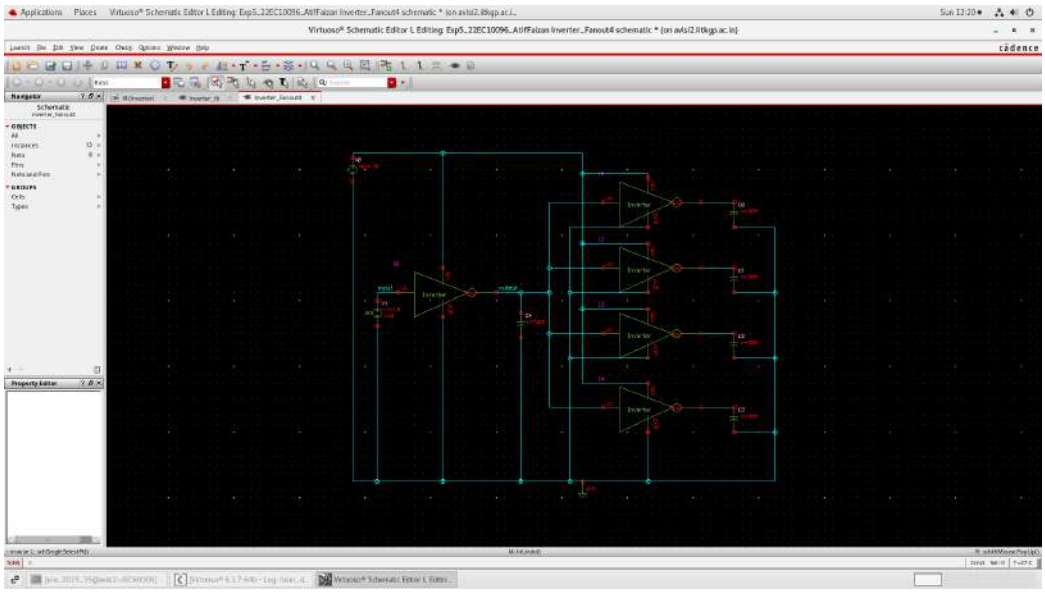


Figure 11: Inverter with Fanout 4 Schematic

0.1 Simulation Results

This is the result of the simulation:

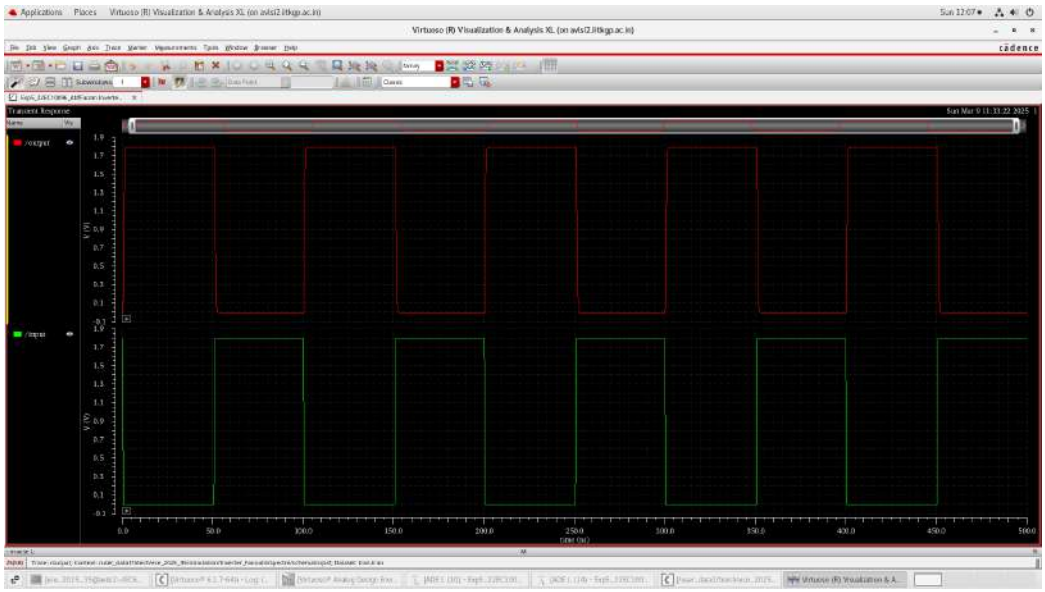


Figure 12: Simulation Result

This is the Rise Time:

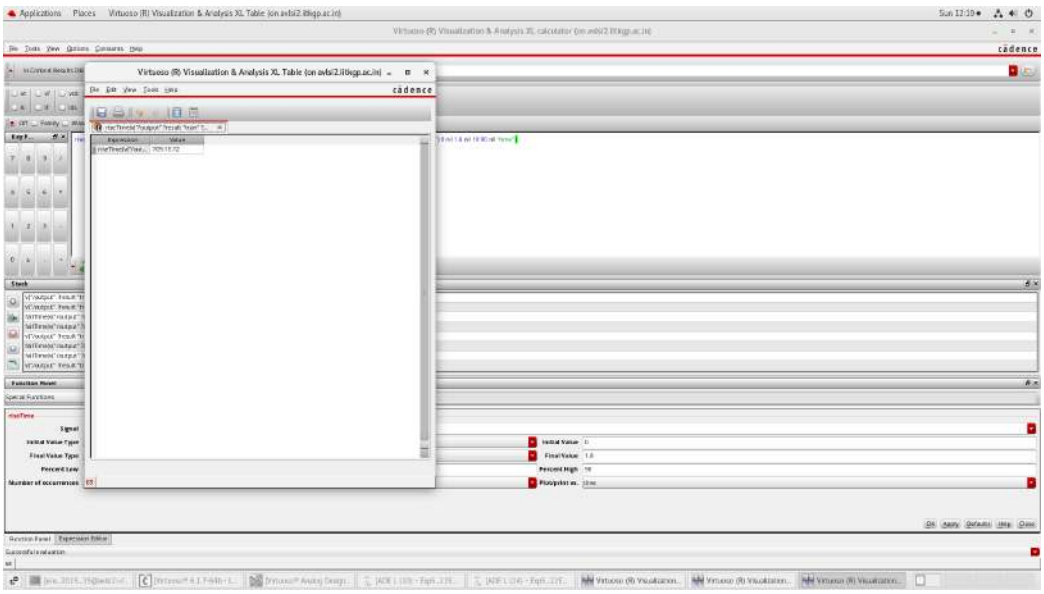


Figure 13: Rise Time

This is the Fall Time:

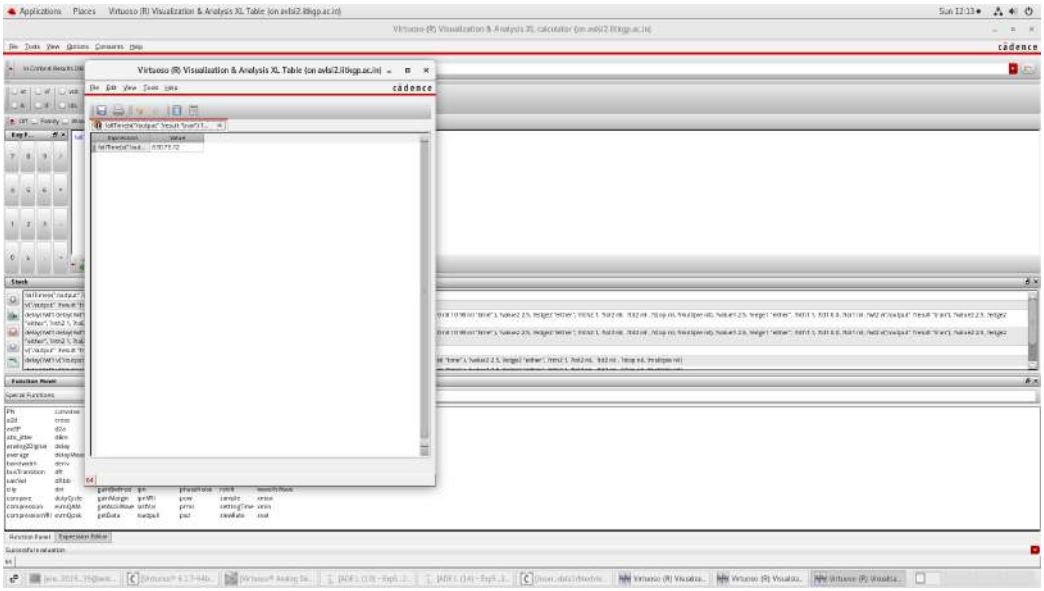


Figure 14: Fall Time

This is the Delay:

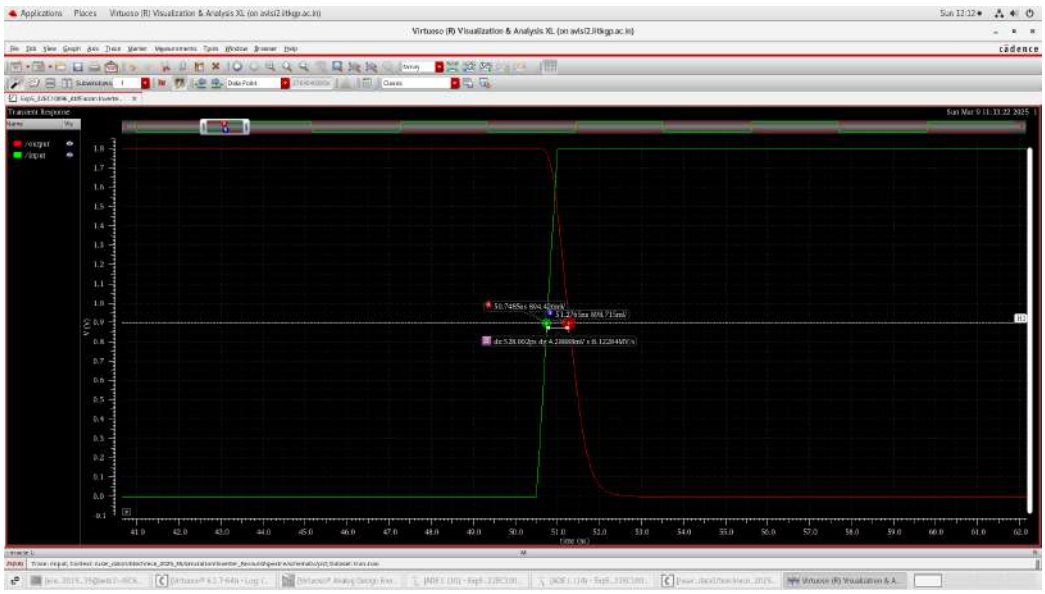


Figure 15: Delay from High to Low

Observation

We observed the following things:

- Rise Time = 709.1ps
- Fall Time = 830.7ps
- $tp_{HL} = 0.528ns$
- $tp_{LH} = 0.76ns$
- Propagation Delay :

$$tp = \frac{tp_{HL} + tp_{LH}}{2} = \frac{0.528 + 0.76}{2} = 0.644$$

Hence Propagation delay = 0.644ns

Part 3: Designing Ring Oscillator

Design

We used $C_L = 100fF$

Schematic

The schematic is as follows:

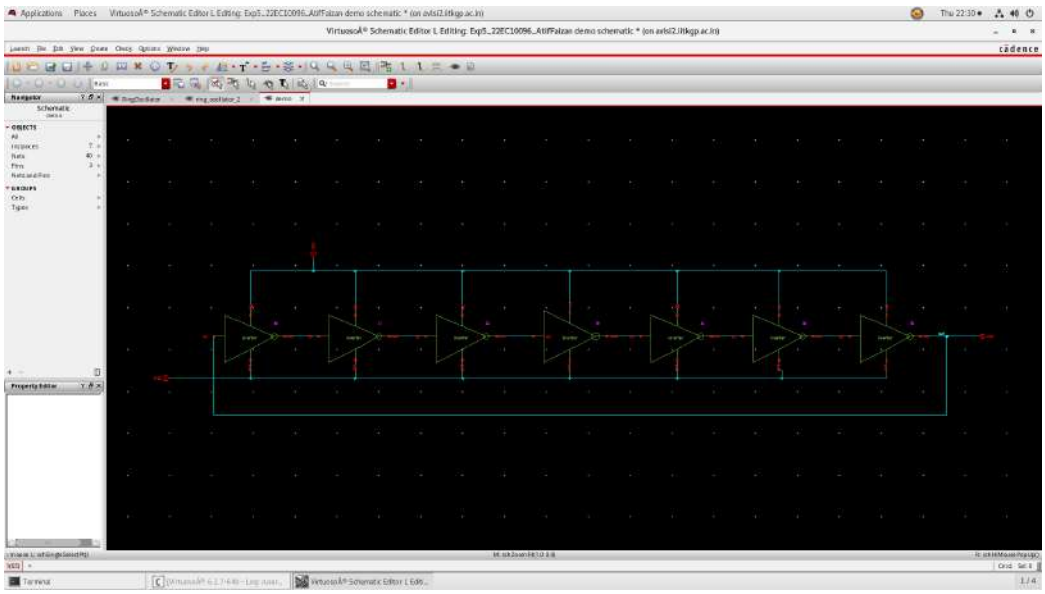


Figure 16: Ring Oscillator Schematic without Load Capacitance

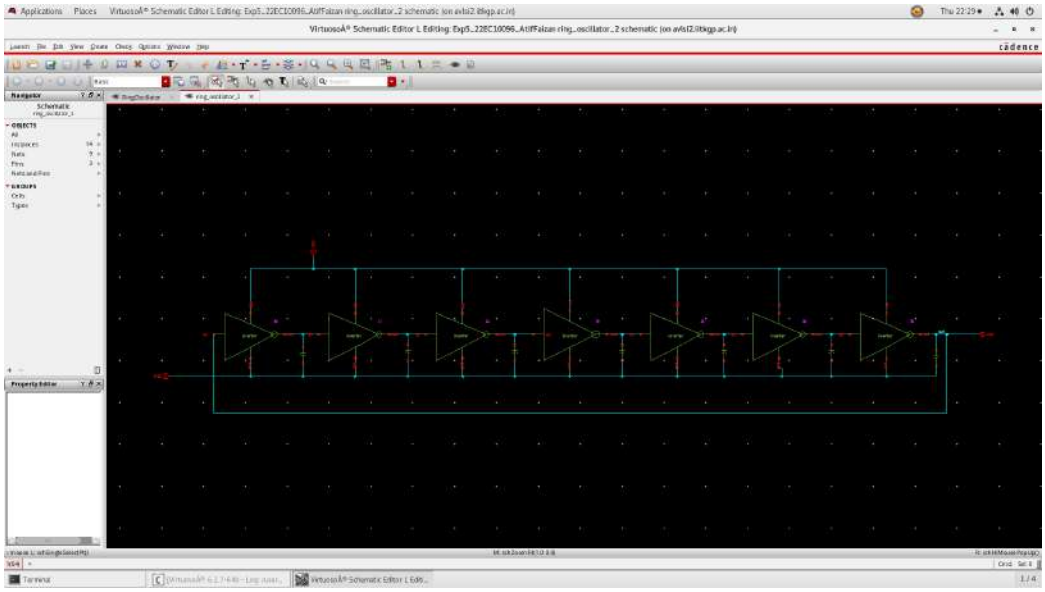


Figure 17: Ring Oscillator Schematic with Load Capacitance of 100fF

0.2 Simulation Results

The rise time is as follows:

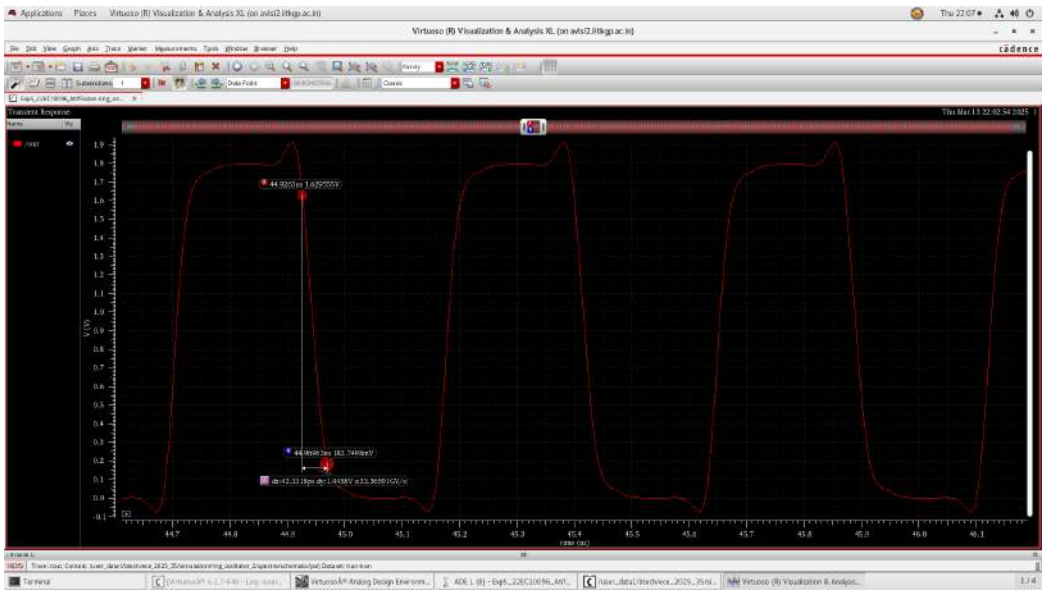


Figure 18: Rise Time without Load Capacitance

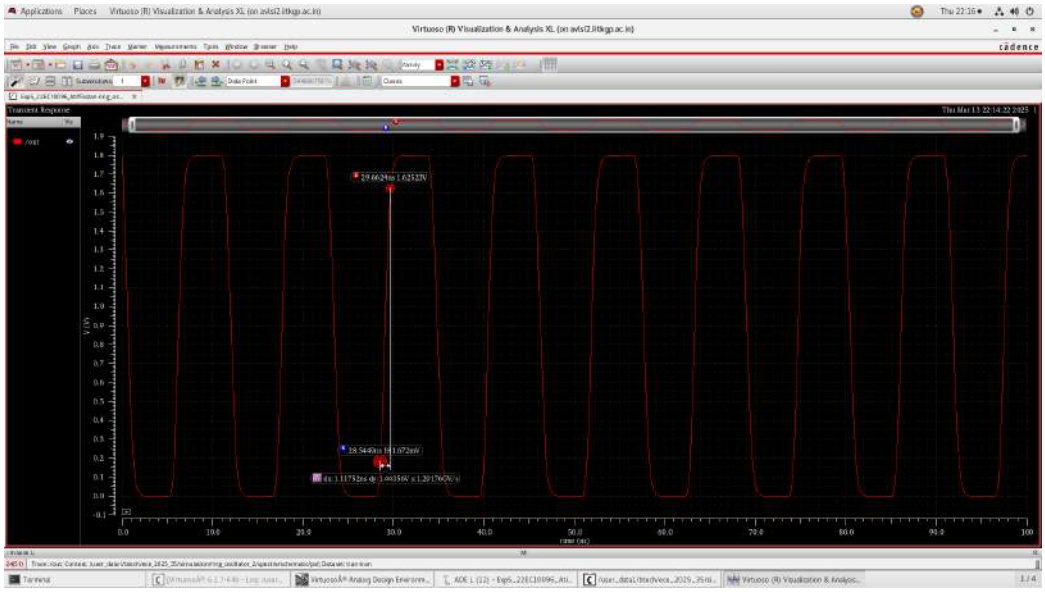


Figure 19: Rise Time with Load Capacitance of 100fF

The fall time is as follows:

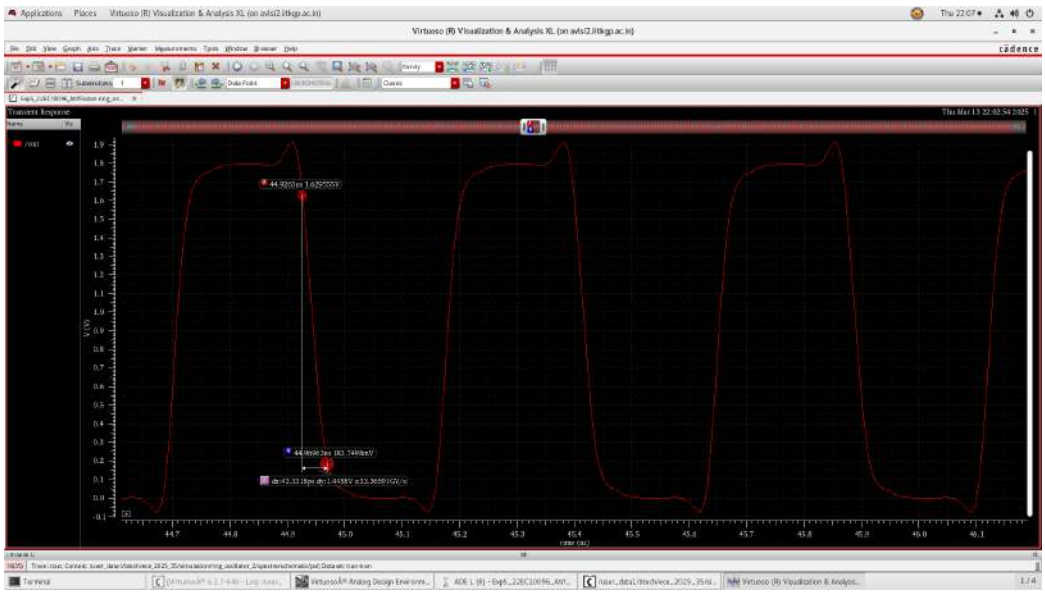


Figure 20: Fall Time without Load Capacitance

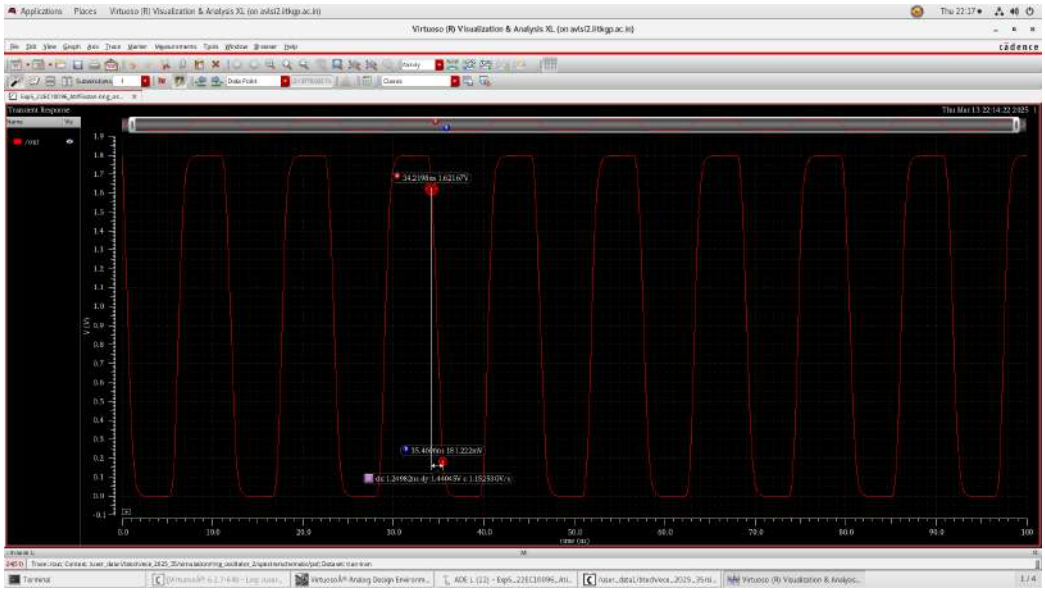


Figure 21: Fall Time with Load Capacitance of 100fF

The period of the circuit is as follows

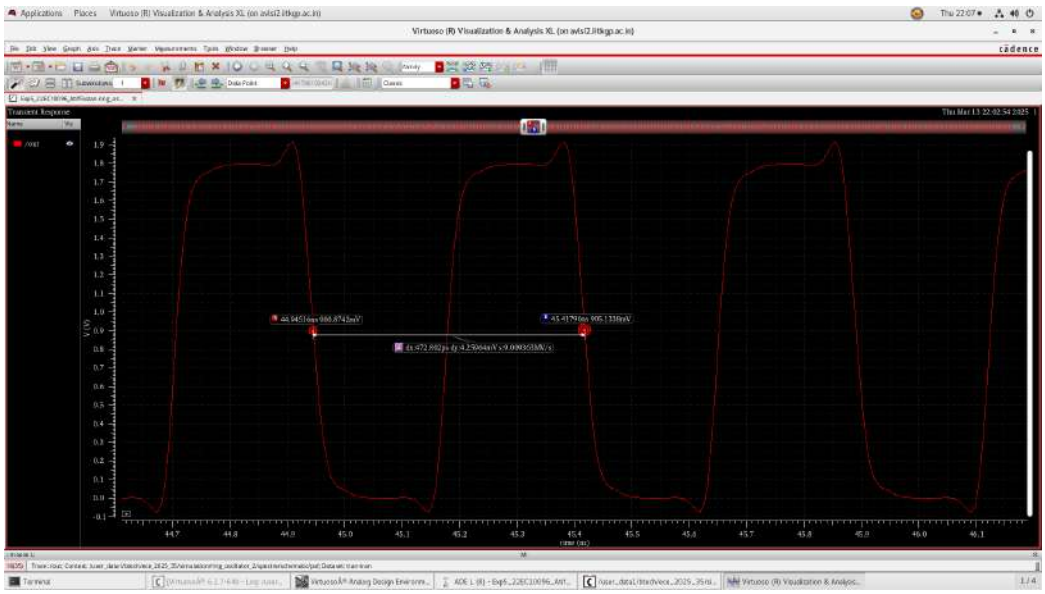


Figure 22: Period of the Ring Oscillator without Load Capacitance

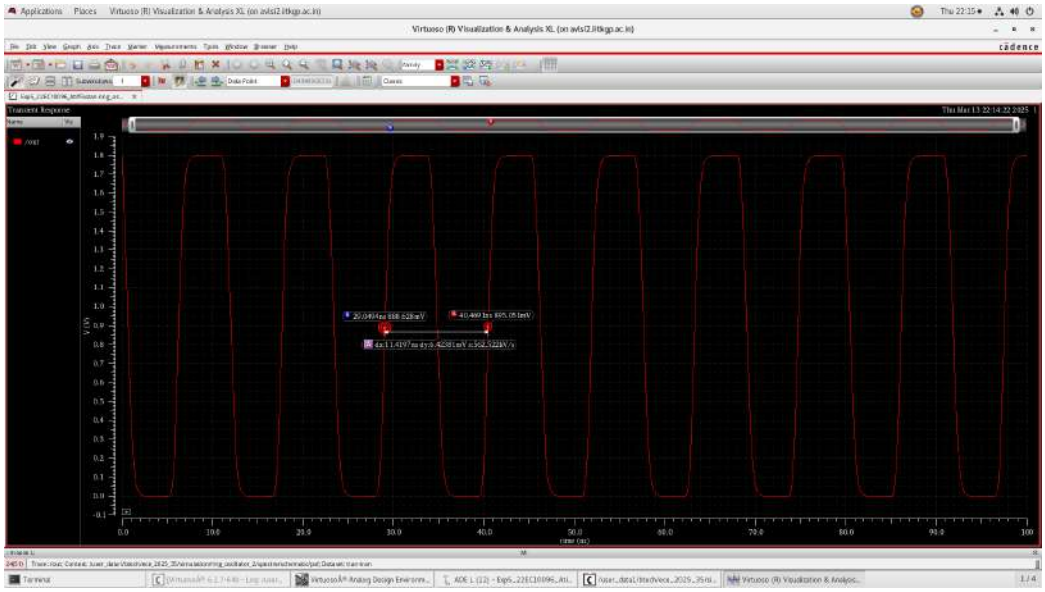


Figure 23: Period of the Ring Oscillator with Load Capacitance of 100fF

Observation

We observed the following things with respect to the Ring Oscillator with Load Capacitance of 100fF:

- Rise Time = 1.11752ns
- Fall Time = 1.24982ns
- Time Period = 11.4197ns

Verification of Time Period of Ring Oscillator

We know that the propagation delay of the inverter designed by us is:

$$tp = 0.644ns$$

We also know that for a ring oscillator time period id:

$$T_p = 2 * n * tp$$

where n denotes the number of inverter used. Hence

$$T_p = 2 * 7 * 0.644 = 9.016ns$$

. The time period observed for the oscillator is 11.4197ns which is approximately same as time period calculated.

Part 4: Design of Clock Driver for Fanout of 64

Design

Fanout of 64 implies that the Clock Driver should be able to drive load of 64 times the capacitance of the driver itself with negligible delay. If the Clock Driver is directly connected to the load capacitance 64C the total delay:

$$tp = tp_o * (1 + \frac{64C}{C}) = 65 * tp_o$$

. This delay is very large and to reduce the delay we progressively increase the load capacitance rather than a sharp increase from C to 64C. We added two stages of Capacitance 4C and 16C after the input to reduce the delay. The new structure is as follows:

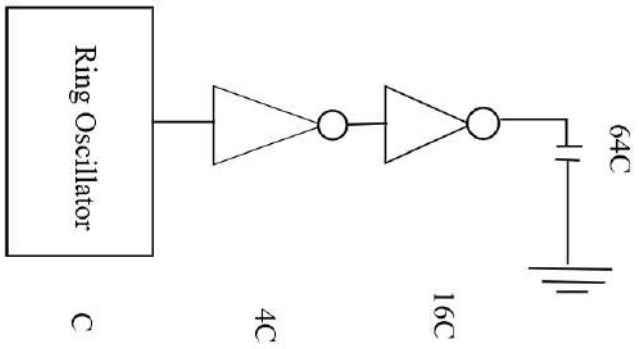


Figure 24: Sizing of Inverter Chain

Now the delay will be $tp = tp_1 + tp_2 + tp_3$ where $tp_1 = tp_o * (1+4/1)$, $tp_2 = tp_o * (1+16/4)$ and $tp_3 = tp_o * (1+64/16)$

$$tp = 5tp_0 + 5tp_o + 5tp_o = 15tp_o$$

Schematic

The schematic is as follows:

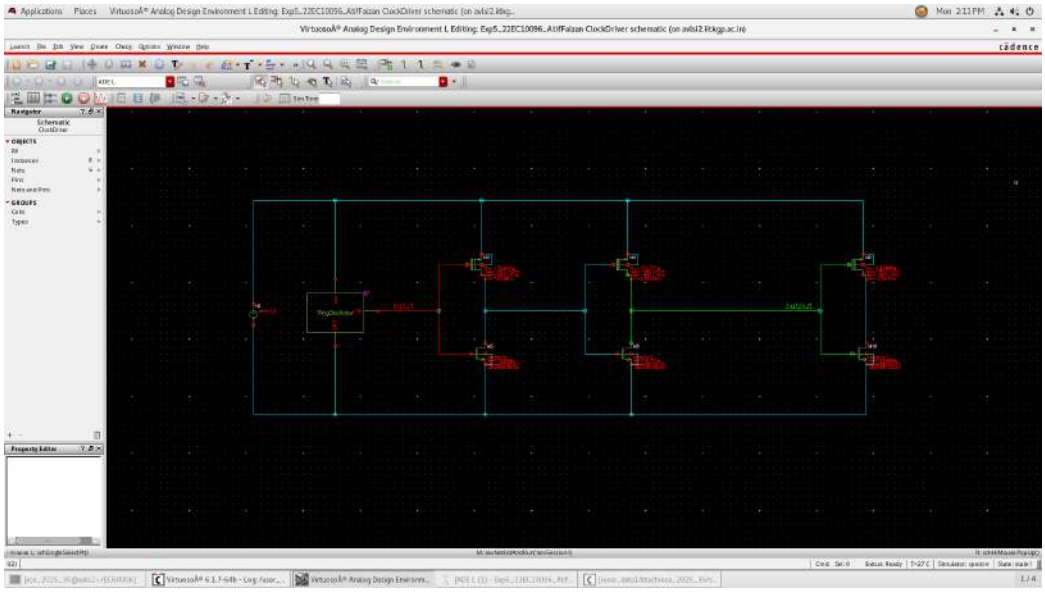


Figure 25: Clock Driver Schematic

0.3 Simulation Results

The input is as follows:

Figure 26: Input to the Clock Driver

The output is as follows:

Figure 27: Output to the Clock Driver

The simulation of the circuit is as follows:

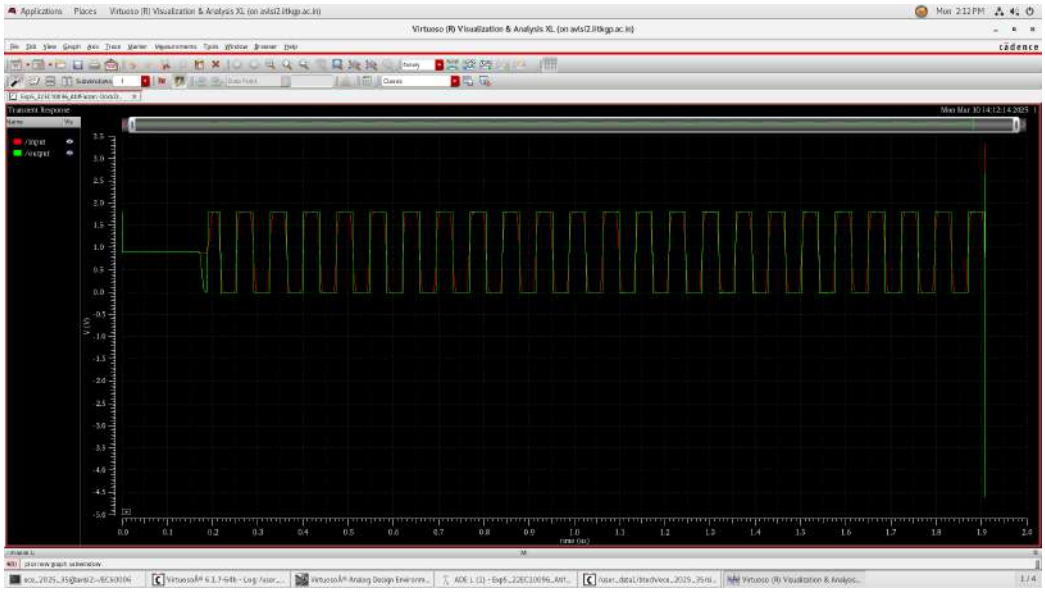


Figure 28: Clock Driver Simulation

Discussion Atif Faizan 22EC10096

- In this experiment we used Cadence virtuoso to design a CMOS inverter, a ring oscillator and a clock driver. The CMOS inverter has been tested against a fanout of 4 and the clock driver has been tested against a fanout of 64. Fanout of x refers to the number of similar devices a device can drive without significant distortion.
- There are many topologies of inverters like resistive load inverters, enhancement load inverters, depletion mode inverters, and cmos inverters. Out of all these inverters CMOS has been widely used because of no static power dissipation (ignoring the leakage current), high noise margins and symmetric rise time and fall time (we can adjust it by adjusting the ration of NMOS and PMOS in the CMOS)
- It should be noted that symmetric rise and fall time is obtained in the CMOS technology when

$$\frac{(W/L)_p}{(W/L)_n} = \frac{\mu_n}{\mu_p}$$

And minimum delay is obtained when

$$\frac{(W/L)_p}{(W/L)_n} = \sqrt{\frac{\mu_n}{\mu_p}}$$

- Symmetric delay is more important to use and hence we sweeped the parameters to get the symmetric rise and fall time.
- We obtained the logic threshold to be $V_{dd}/2$ by appropriately sizing the NMOS and PMOS. Increasing $(W/L)_p$ increases the V_T while increasing the $(W/L)_p$ decreases the V_T.
- The rise time and fall time of the inverter operating against a fanout of 4 was larger and delay was

$$tp = tp_0 * (1 + 4) = 5tp_0$$

.

- Ring Oscillator can be designed with any odd numbers of inverters theoretically. Practically it is not possible to design a ring oscillator with number of gates as low as 3. We designed a 7 stage ring oscillator for the experiment.
- The total delay of the ring oscillator

$$T_p = 2 * tp * n$$

where tp is the delay of the individual inverters and n is the number of inverters used. However we observed a slightly longer delay in simulation. The calculated delay was 9.016ns but the observed delay was 11.4197ns.

- Fanout of 64 implies that the Clock Driver should be able to drive load of 64 times the capacitance of the driver itself with negligible delay.If the Clock Driver is directly connected to the load capacitance 64C the total delay:

$$tp = tp_o * (1 + \frac{64C}{C}) = 65 * tp_o$$

. This delay is very large and to reduce the delay we progressively increase the load capacitance rather than a sharp increase from C to 64C. We added two stages of Capacitance 4C and 16C after the input to reduce the delay. Now the delay will be $tp = tp_1 + tp_2 + tp_3$ where $tp_1 = tp_o * (1 + 4/1)$, $tp_2 = tp_o * (1 + 16/4)$ and $tp_3 = tp_o * (1 + 64/16)$

$$tp = 5tp_0 + 5tp_o + 5tp_o = 15tp_o$$

- A delay of $15tp_o$ is significantly less than the delay of $65tp_o$
- Apart from insights about the experiment we learned about using Cadence tool, the various functionalities like transient analysis, dc sweep, symbol creation, schematic creation analog library and many more. We got a hands on experience of using industry level tools for simulation of our circuits.

Discussion Chiradip Biswas 22EC30016

- In this experiment, in 1st part we designed a CMS inverter, and sized the transistors present in it. The main goal was to make the logic threshold to be VDD/2. Equating the current of pmos and nmos in the saturation region and considering channel length modulation, we obtain the following equation:

$$\frac{1}{2}\mu_n C_{ox}(\frac{W}{L})_n(V_{in} - V_{th})^2(1 + \lambda_n V_{out}) = \frac{1}{2}\mu_p C_{ox}(\frac{W}{L})_p(VDD - V_{in} - |V_{tp}|)^2(1 + \lambda_p(VDD - V_{out})) \quad (1)$$

solving it we get:

$$\frac{(\frac{W}{L})_n}{(\frac{W}{L})_p} = \left(\frac{(VDD/2 - |V_{TP}|)(1 + \lambda_p VDD/2)}{(VDD/2 - V_{Th})(1 + \lambda_n VDD/2)}\right)^2 \frac{\mu_p}{\mu_n} \quad (2)$$

This is the actual formula of width sizing, to have the logic threshold in the region where both are in saturation. Considering the region to be thin enough, the very small error can be tolerated. But for precision, we perform parameter sweep (the gate width of pmos keeping the nmos width to be constant).

Here we have considered the gate capacitances of each of p and nmos to be same. Further simplification can be done by neglecting the channel length modulation and considering the threshold voltage of both nmos and pmos to be same. Then the formula becomes:

$$\frac{(W/L)_p}{(W/L)_n} = \frac{\mu_n}{\mu_p} \tag{3}$$

as the μ_n is approximately 2 times the μ_p, the width ration becomes 2:1.

- The minimum propagation delay by the inverter is obtained by taking this ratio to be $\sqrt{2} : 1$. But in this case the rising and falling time becomes significantly different. This sort of asymmetry is not good for sequential circuits. so we prioritize symmetric rise and fall time over the minimum propagation delay. Increasing the W/L)p shifts logic threshold to higher value [as more charge is stored in load capacitor and fast charging]. Increasing the W/L)P decreases logic threshold (fast discharging).

- When we increase the fan-out of the gate, the load capacitor value increases and hence the rising and falling time increases. Hence the delay increases as well. When the inverter was driving a fan-out of 4, the propagation delay can be written as:

$$tp = tp_0(1 + 4) = 5tp_0 \tag{4}$$

  Here tp_0 is he intrinsic delay of the inverter, that arises due to parasitic capacitors and the resistances observed in the mosfets.

- While building the ring oscillator, we should use odd number of inverter stages, and also the number of stages should be ¿2. When using 2 stages it acts as latch, as positive feedback, hence amplifying the input and saturating to rail voltages. When using 3 stages, the negative feedback voltage comes at 180 degree out of phase, hence having a 360 degree phase shift.

- The oscillator, having n stages and each stage having propagation delay of tp_0 has a time period(T) of $2ntp_0$=9ns (approx). However the period we observed (11ns (approx)) was slightly longer than this calculated value. This is due to the fact that the inverters have a finite rise time and fall time, thus contributing to the observed time period.

- When no load capacitor was used at the output of the oscillator was was observed to be having an upward spike just before falling from VDD to 0, and a downward spike, just before rising from 0 to VDD. This is due to the parasitic gate drain capacitor that couples the input voltage to output voltage. When the voltage reaches significantly greater than the threshold voltage, the charging/discharging pumps out/in the extra charge, thus restoring normalcy.When a larger capacitor was applied at the load,this overshoot value decreased, due to voltage division across the series caps.

- While making the clock driver, to run a 64C capacitance fanout, we did not directly connect the 64C stage (having an input capacitance of 64 times that of the cmos inverter). This increases delay of the system. So we apply some tapered buffer in between the oscillator and the 64C load, with the gate widths being raised by a common ratio of 4 $[(64)^{1/3}]$. This is obtained from minimum propagation delay calculation of an inverter chain. Now the delay becomes: $tp = tp_1 + tp_2 + tp_3$ where $tp_1 = tp_o * (1 + 4/1)$, $tp_2 = tp_o * (1 + 16/4)$ and $tp_3 = tp_o * (1 + 64/16)$

$$tp = 5tp_0 + 5tp_o + 5tp_o = 15tp_o$$

Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 7

Design of 4-bit Adder and 4-bit PIPO Register

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016

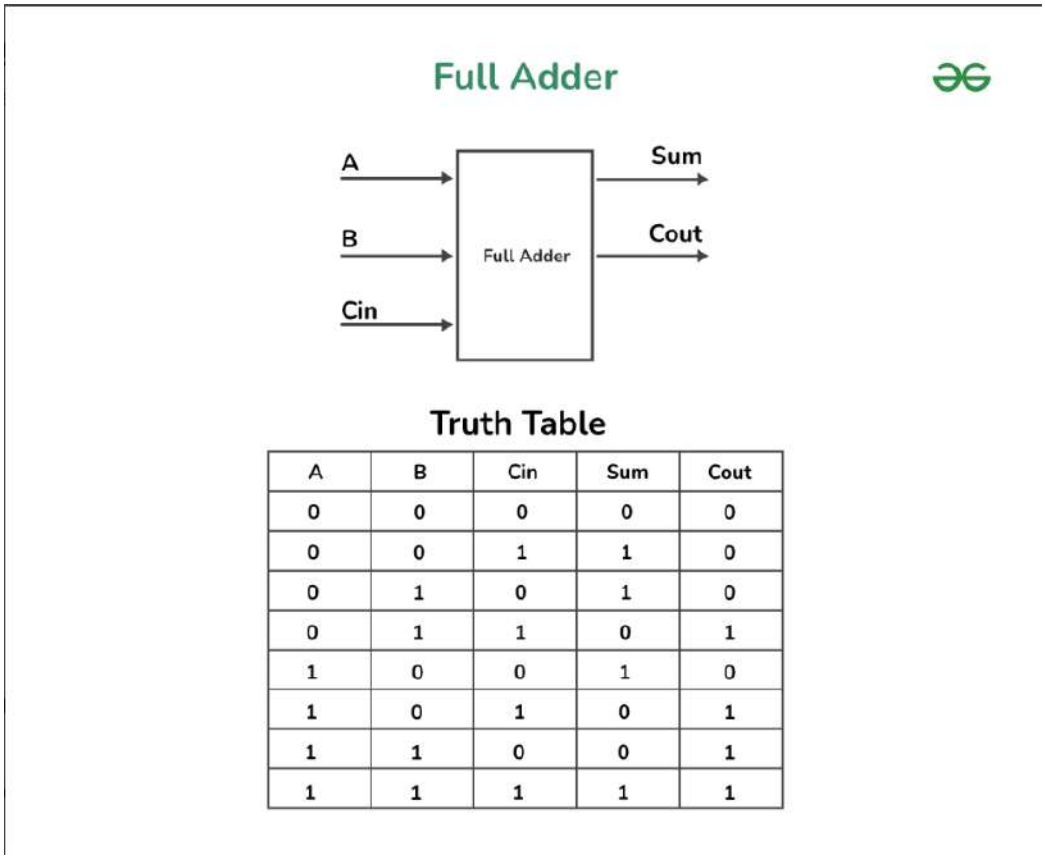
- **Group no.:** 35 (Monday)
- **Date of Submission:** 30-03-2025

Aim

1. Design and simulate 1 bit full adder circuit. Verify its functionality and observe its critical path delays (both worst case and best case) with load capacitance equal to input capacitance of four unit size inverters together.
2. Design and simulate a 4-bit Adder using the designed 1-bit full adder in part (a) of this experiment.
3. Construct a parallel-in-parallel 4-bit register using the D-FF obtained in experiment 6.
4. In simulation demonstrate maximum speed of operation of a pipeline unit containing the 4-bit adder as combinational circuit in-between two register layers (one layer is for feeding the inputs and the other one is for storing the output).

Theory

- Full Adder is a Digital Logic Circuits that can add three inputs and give two outputs i.e. three inputs such as A, B, and input carry as Cin and gives a sum output and carry output i.e. two outputs.
- The following is the Block Diagram and Truth Table of the One bit Full Adder



Truth Table

A	B	Cin	Sum	Cout
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

Figure 1: Block Diagram and Truth Table for One Bit Full Adder

- Logical Expression of One Bit Full Adder from the truth table:

$$Sum = A \oplus B \oplus C_{in}$$

$$C_{\text{out}} = AB + AC_{\text{in}} + BC_{\text{in}}$$

- Implementation of Full Adder using NAND Gates is realization of Full Adder by using minimum nine NAND Gates during which we will have 2 outputs at the end namely Cout and Sum. Given Below is the Circuit For Implementation of Full Adder using NAND Gates.

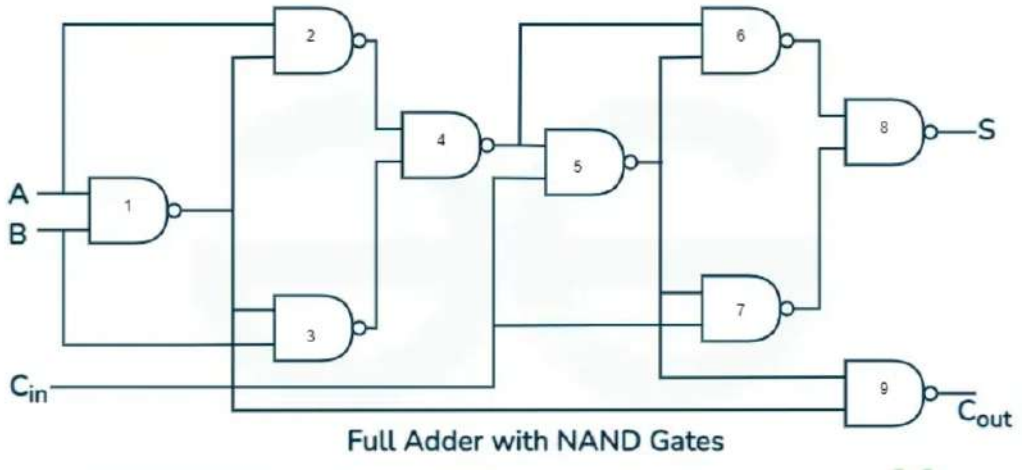


Figure 2: Implementation of Full Adder using NAND Gates

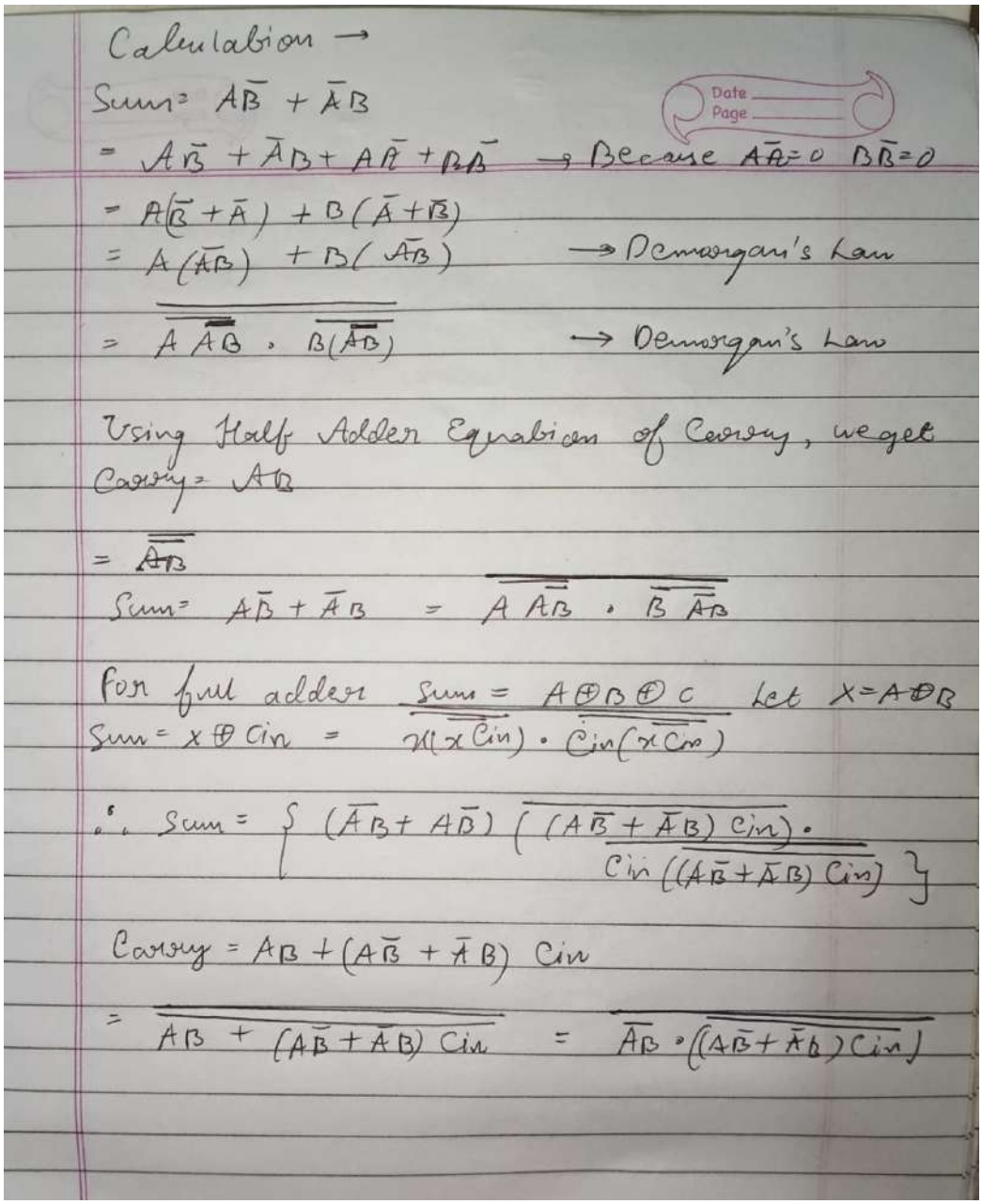


Figure 3: Calculation of Implementation

- We can cascade as many 1 bit full adder as we want to produce a n bit full adder.
- A PIPO shift register is a collection of flip-flops arranged in a series, with each flip-flop capable of storing one bit of data. The primary characteristic that distinguishes a PIPO shift register is its ability to load data and output it in parallel.

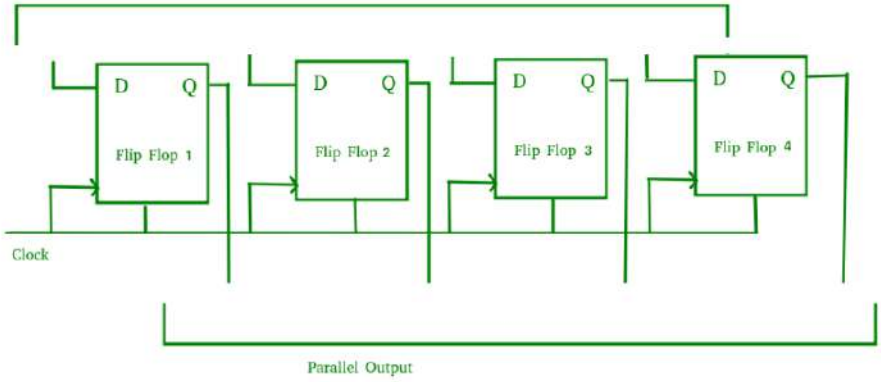


Figure 4: PIPO Shift Register using D flip flops

- Each flip-flop's contents are passed sequentially to the following flip-flop to shift the data out. This can be accomplished by providing a clock signal that triggers the shifting operation. When the clock signal is activated, the data in each flip-flop is transferred to the adjacent flip-flop, allowing new data to be loaded into the first flip-flop.

Part 1: 1-bit Full Adder

Design

We used the NAND gates developed in the previous experiment to design the 1 bit full adder.

Schematic

The following is the schematic of the 1 bit full adder:

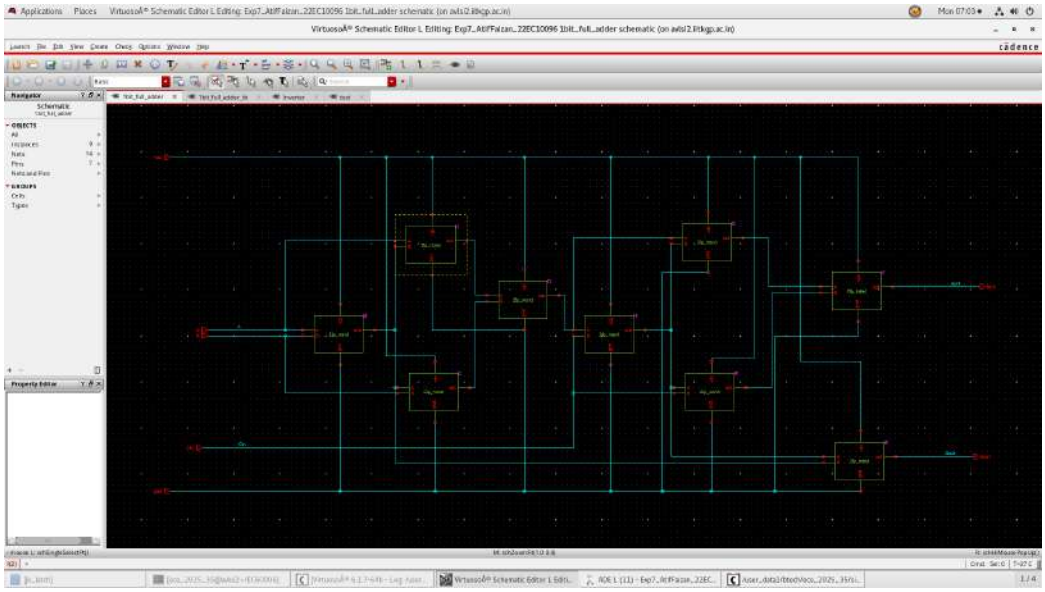


Figure 5: 1-bit Full Adder Schematic

The following is the testbench:

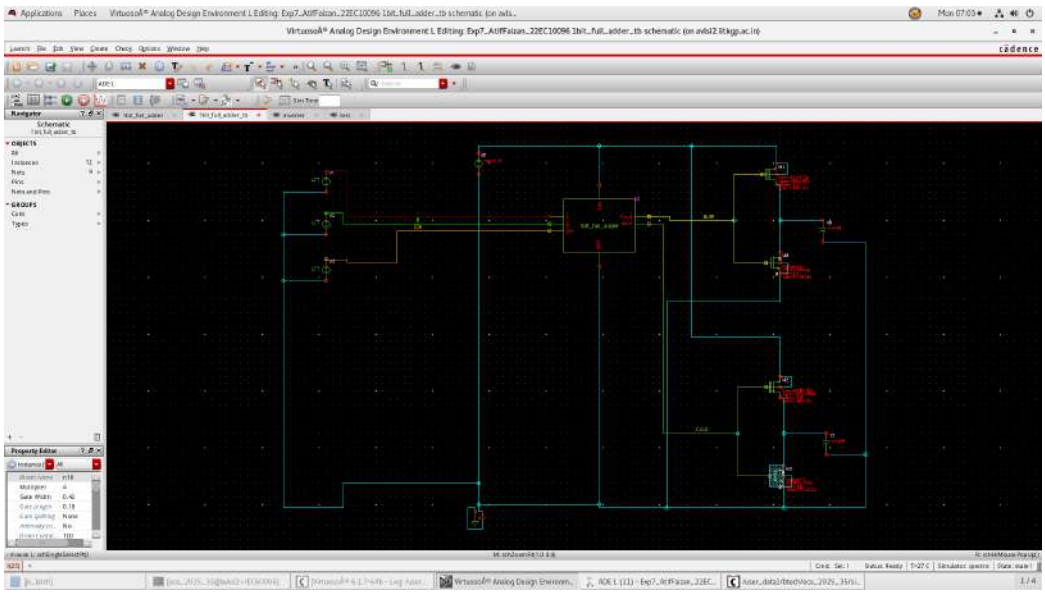


Figure 6: Testbench of 1-bit Full Adder

Simulation Results

The following is the simulation result of the testbench:

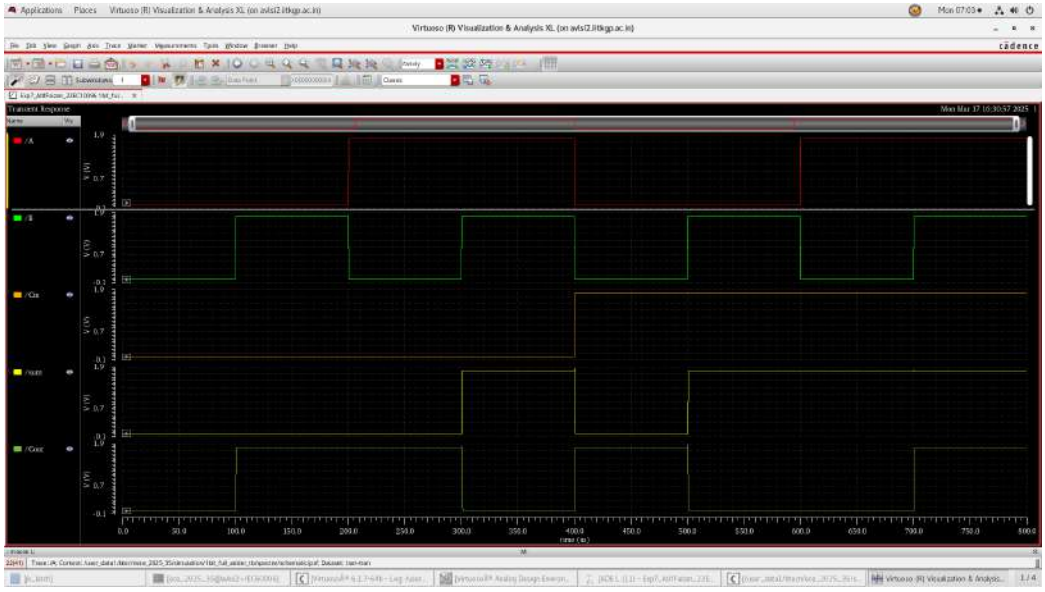


Figure 7: Simulation Results of 1-bit Full Adder

The following is the best case delay:

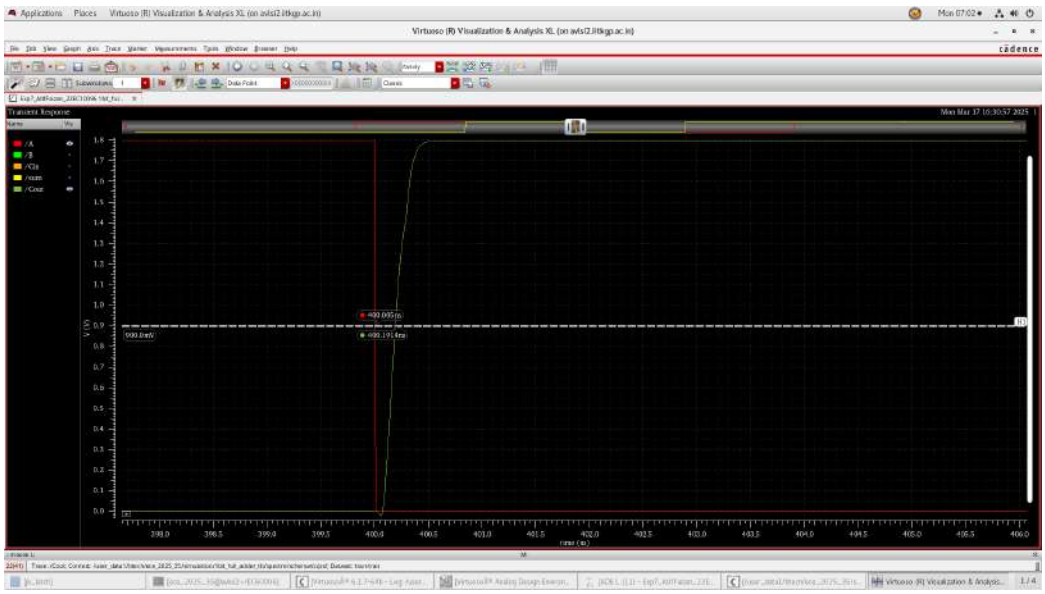


Figure 8: Best Case Delay of 1-bit Full Adder

The following is the worst case delay:

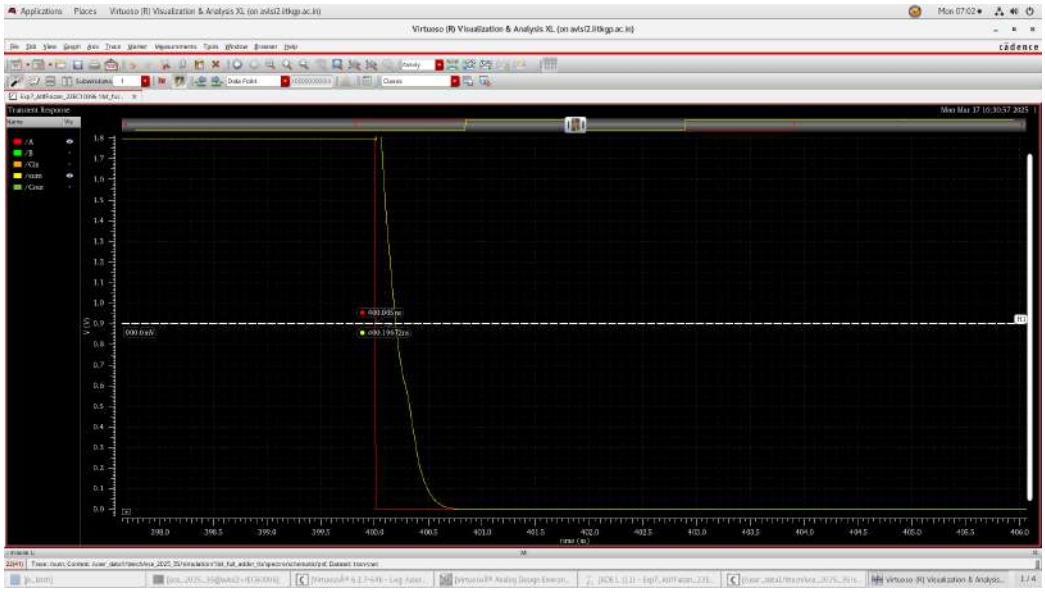


Figure 9: Worst Case Delay of 1-bit Full Adder

Observations

We observed the following things:

- Worst Case Delay = 191.72ps
- Best Case Delay = 186.4ps

4-bit Full Adder

Design

We used 4 1-bit full adder in cascade to design the 4 bit full adder.

Schematic

The following is the schematic of a 4-bit full adder:

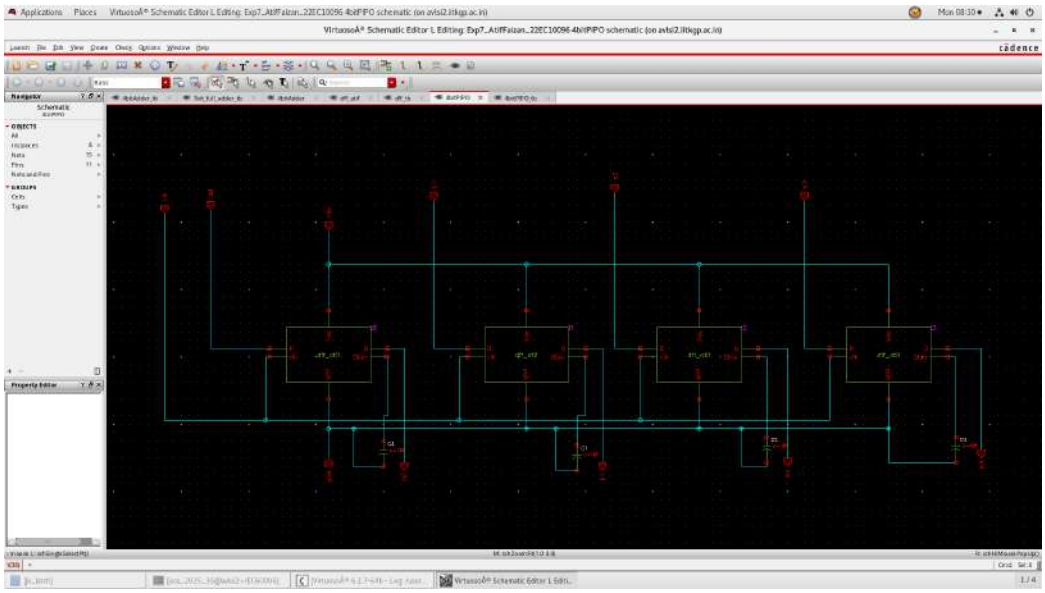


Figure 10: Schematic of 4-bit PIPO Register Using D-Flip Flops

The following is the testbench of the 4-bit Full Adder Simulation:

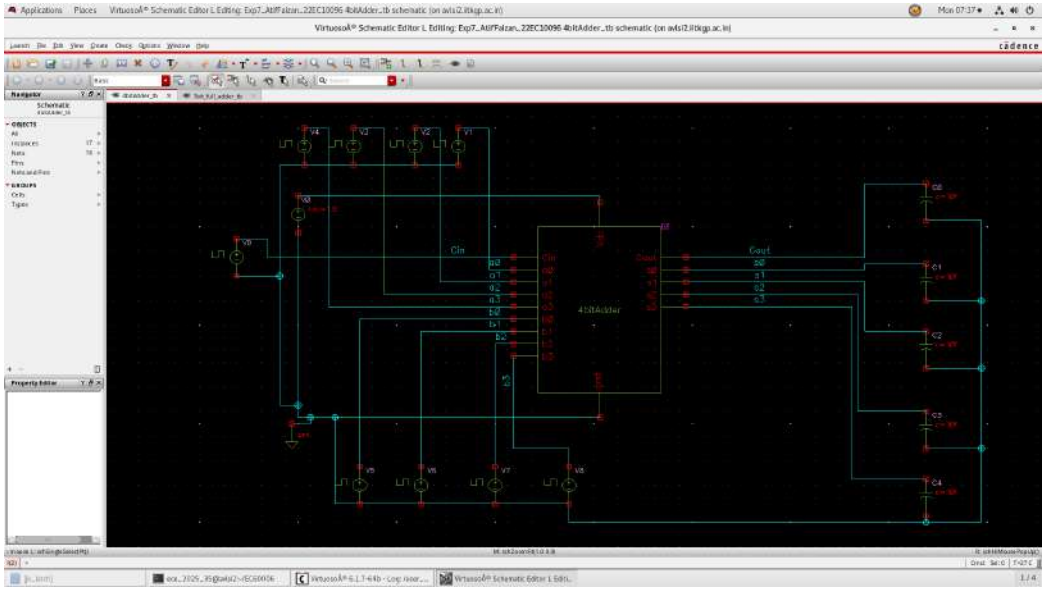


Figure 11: Testbench of 4-bit Full Adder

Simulation Results

The following is the result of the simulation:

Figure 12: Simulation Result of 4-bit Full Adder

Part 3: Designing 4-bit PIPO Register

Design

We used the following d flip flop for designing the PIPO Register:

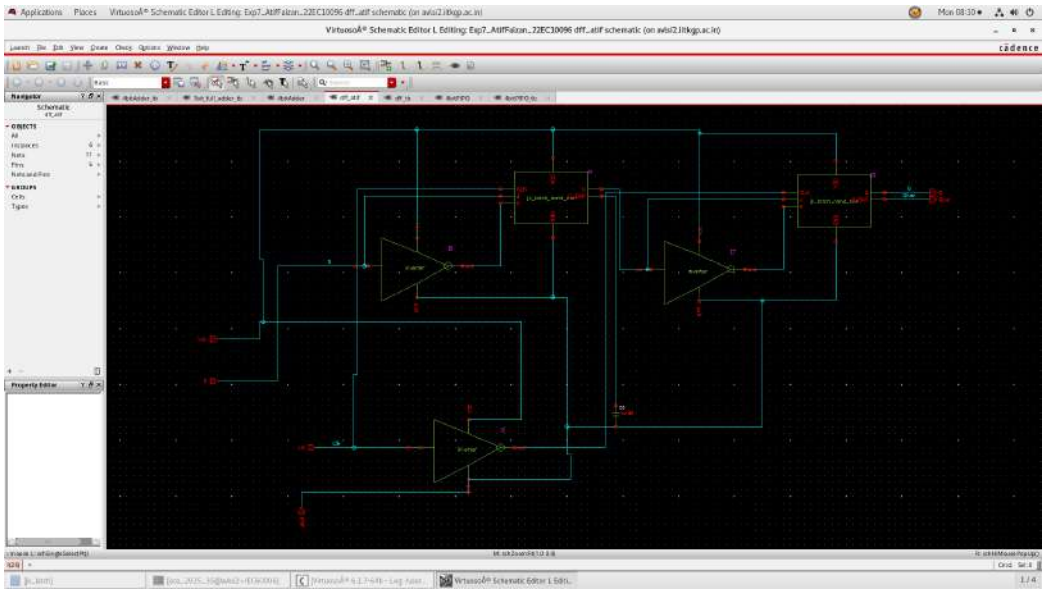


Figure 13: D flip flop

Schematic

The following is the schematic of the 4-bit PIPO Register:

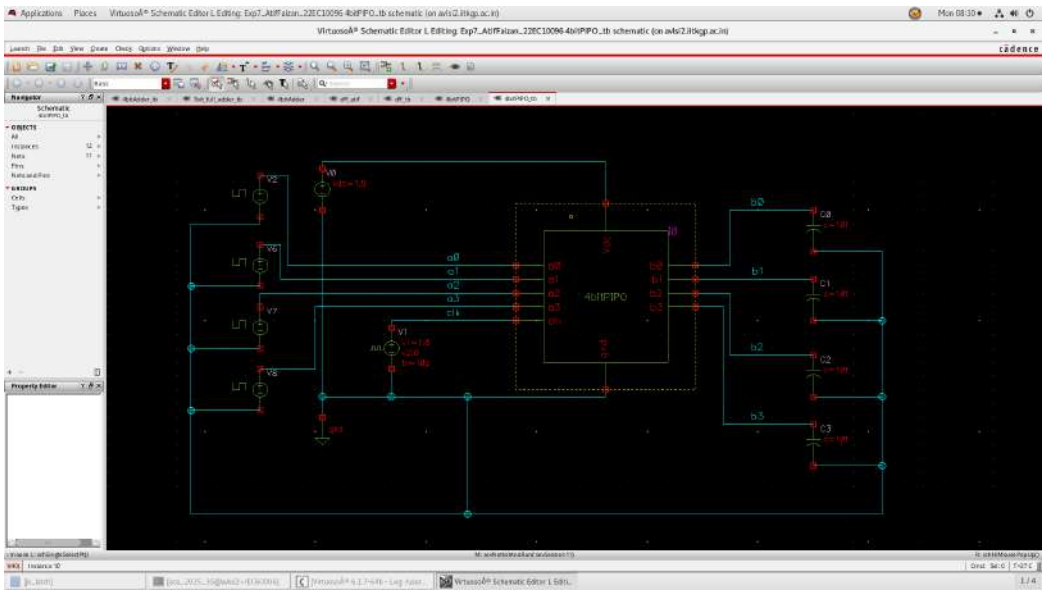


Figure 14: 4-bit PIPO Register Testbench

Simulation Results

The following is the simulation result of the 4-bit PIPO Register:

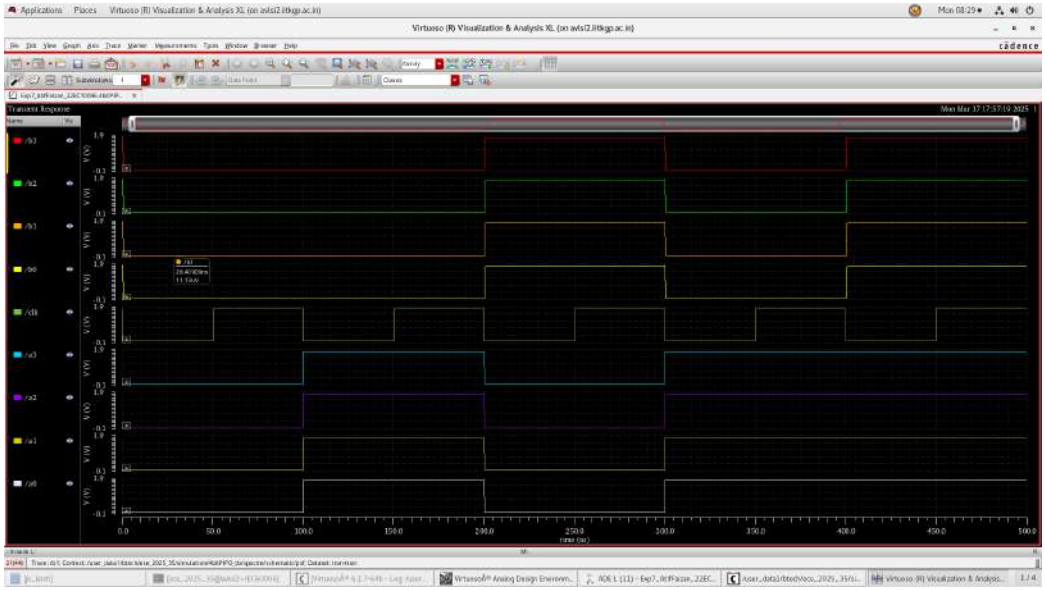


Figure 15: Simulation Result of the PIPO Register

Part 4: Designing of Pipeline Unit

Design

We used 3 4-bit PIPO Register (2 for giving the input and 1 for storing the output) and 1 4-bit full adder for designing the pipeline unit.

Schematic

The following is the schematic of the Pipeline Unit:

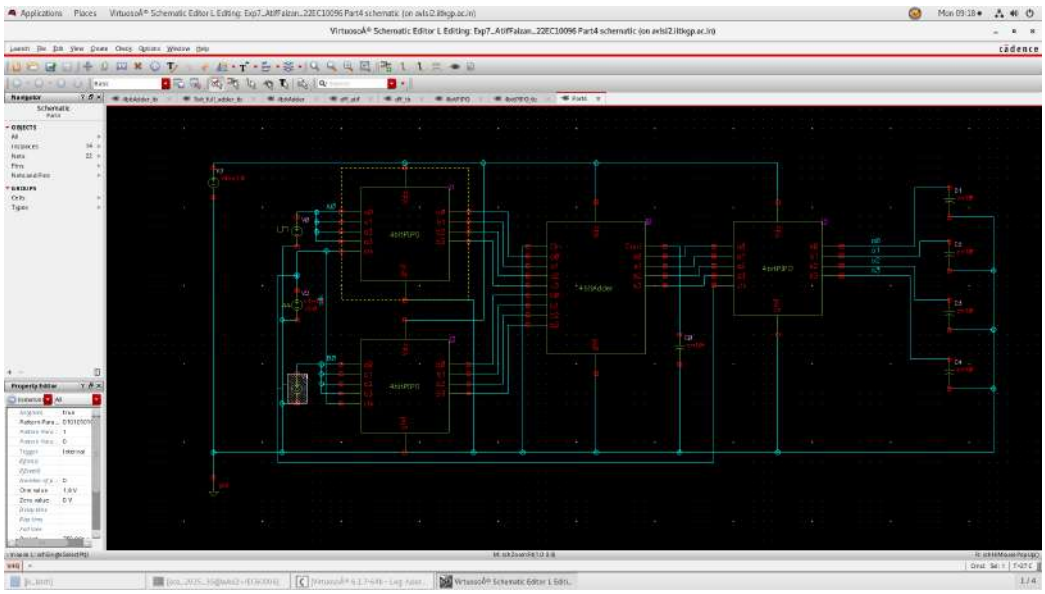


Figure 16: Schematic of Pipeline Unit

Simulation Results

The following are the simulation result of the Pipeline Unit for different time period of the clock:

Figure 17: Simulation Result of Pipeline Unit for Clock Period of 100ns

Figure 18: Simulation Result of Pipeline Unit for Clock Period of 1ns

Figure 19: Simulation Result of Pipeline Unit for Clock Period of 0.75ns

Figure 20: Simulation Result of Pipeline Unit for Clock Period of 0.5ns

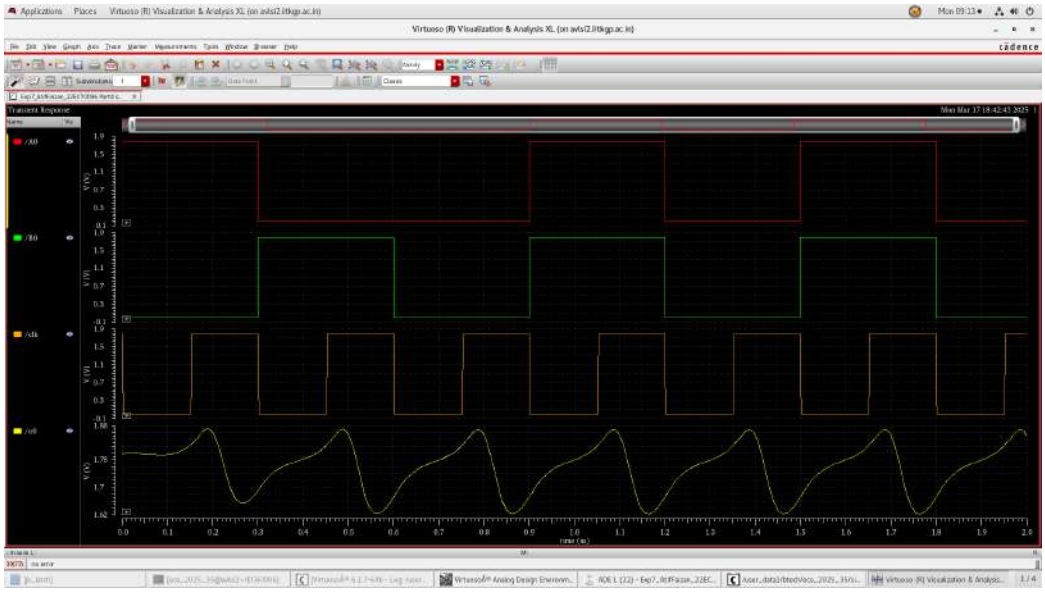


Figure 21: Simulation Result of Pipeline Unit for Clock Period of 0.3ns

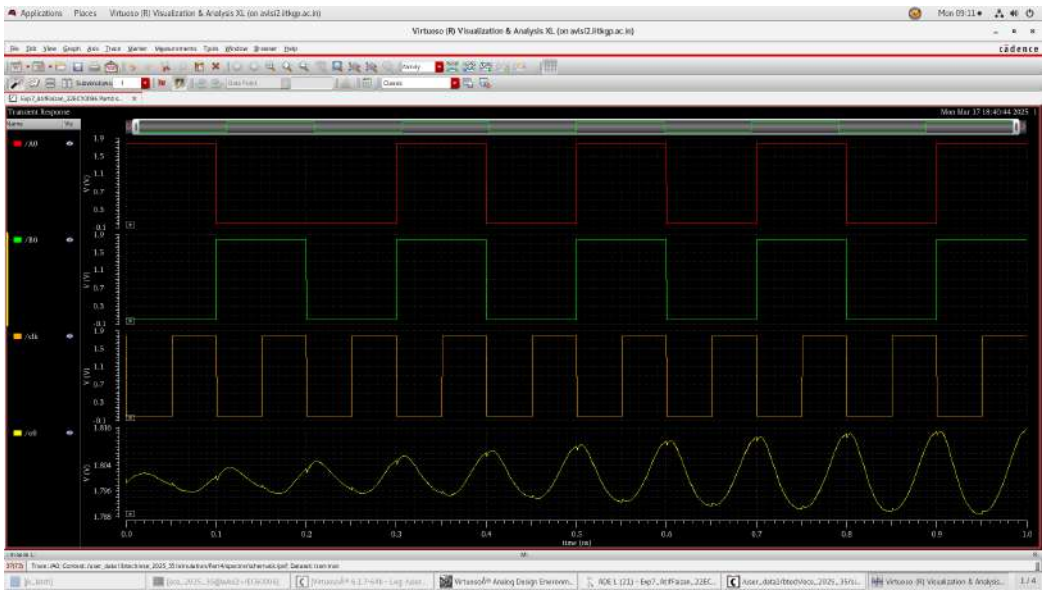


Figure 22: Simulation Result of Pipeline Unit for Clock Period of 0.1ns

Observation: Demonstration of Maximum Speed of Operation of the Pipeline Unit

As it is evident from the Simulation Result that the output get distorted if we use time period of the clock to be less than 0.75ns. Hence the maximum speed of operation is : 1.33GHz.

Verification of Maximum Speed of Operation

There is one more way of verifying the maximum speed of operation. We cannot operate with a clock period lower than the input output delay in case of low frequency of operation. The following is the delay we obtained for a low frequency (high time period of 100ns):

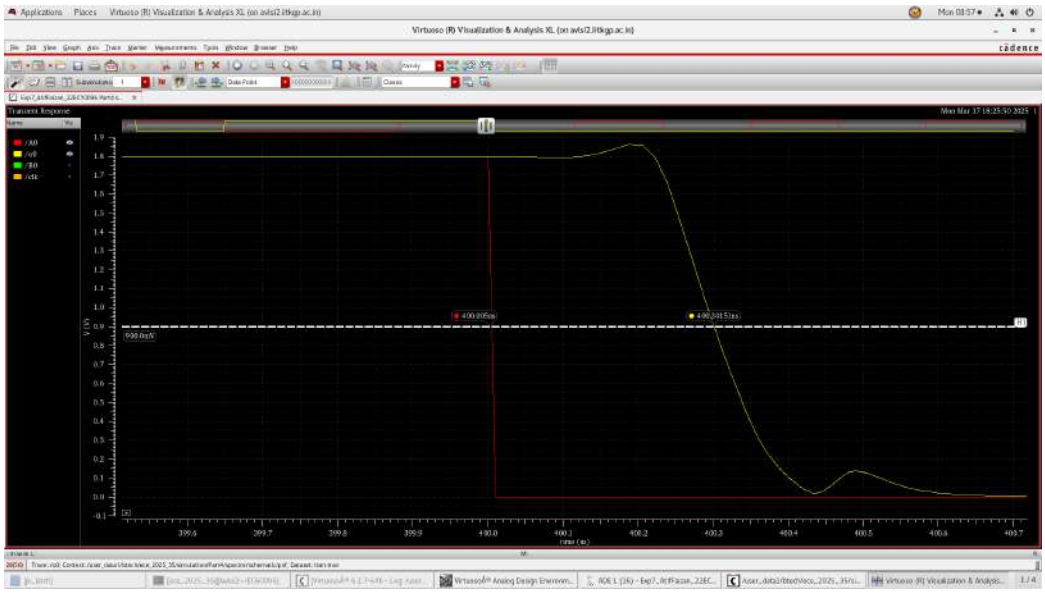


Figure 23: Delay of the Pipeline for a Clock Period of 100ns

It can easily be seen that $Delay = 0.29651ns$ and hence the upperbound on the maximum operating frequency $= 3.37GHz$. We got the maximum operating frequency of 1.33 GHz which is much lower than 3.37GHz because of other factors which leads to distortion of output due to high frequency.

Discussion Atif Faizan 22EC10096

- In this experiment we designed 1 bit Adder, 4 bit Adder, 4 bit PIPO Register and a Pipeline Unit for Addition.
- We used only NAND gates for designing the 1 bit adder and hence we required 9 NAND Gates for the design. The complete design logic for the implementation of 1 bit adder using NAND gates is given in the Logic and Design section. We utilized various laws (like Demorgan's Laws) for simplification of boolean expression for obtaining the simplified version of implementation of adder using NAND gates.
- If we have designed a CMOS based implementation we would have required lesser number of gates.
- For obtaining worst case and best case delay we tried for every possible combinations of A,B and C_{in}. We obtained worst case delay to be 191.72ps and best case delay to be 186.4ps. We tested our adder against a load capacitance of 4 times the capacitance of the unit size inverters that we made in the experiment 5.
- For making the 4 bit adder we cascaded together 4 1-bit adder with the C_{out} of the previous adder fed into the C_{in} of the next adder and A and B for each of the adder were fed in parallel.
- There are various kinds of register namely Serial in Serial Out (SISO), Serial In Parallel Out (SIPO), Parallel In Serial Out (PISO) and Parallel In Parallel Out (PIPO). There are several advantages and disadvantages of all the types. For serial type implementation, hardware requirement is less but delay is more. While for parallel implementation hardware requirement is more but delay is as low as one clock cycle for feeding in and taking out the output.
- For e.g. in a n bit SISO register, we would require n cycles for feeding in the input and n-1 cycles for taking out the output. On the other hand for n bit PIPO register we require only one clock cycle to feed in the input and 0 clock cycle to take the output.
- In this experiment we made a 4 bit PIPO register. For making a 4 bit PIPO Register we used the D flip flop made in the previous experiment using JK latch.
- In the last part of the experiment we were supposed to make a pipeline unit with 4 bit PIPO Register for giving the input and storing the output and performing addition using the 4 bit adder made in part 2.
- For determining the highest frequency of operation for the pipeline unit we first took a very high clock period of 100ns and calculated the delay between the input and the output. The delay came around to be 0.29651ns. Hence we were sure that the minimum time period (maximum operating frequency) of the clock will be somewhere around this period. We tried for several values and obtained 0.75ns as the minimum time period of the clock. Hence the maximum operating frequency of the clock was 1.33GHz. This frequency is the bottle neck in the computational power and it must be increased to perform computation faster.
- The pipeline unit is important in the sense that in big circuits each part of the circuit has different delay. Hence the addition must be interfaced with latches on both the sides as to ensure reliable computation.
- For this experiment out adder was tested against a load capacitance of 4 unit size inverts designed in experiment 5 of AVLSI lab. If the load capacitance connected to the adder is too high then it can impact its delay. Appropriate invert chain with appropriate sizing must be used to interface the load with the adder so as to reduce the delay.
- Since we used PIPO register we obtained the output just in 2 clock cycles (1 clock cycle for loading the input and other for loading the output in the result register).

Discussion Chiradip Biswas 22EC30016

- In this experiment we have designed 1 bit full adder (using the NAND logic gates designed in CMOS logic in the prior experiments).Then used this 1 bit full adder to make a 4bit full adder. Thereafter we made a PIPO register using the d-flip flop designed in the previous experiment. This 4bit adder unit was used along with the input and output registers to make a pipeline unit.
- 1 bit full adder also considers the carry in and gives the carry out bit accordingly. We made the 1 bit full adder using 9 NAND gates (each made using CMOS technology, so 36 transistors). If we had used CMOS technique directly to make the adder it would have costed lesser number of transistors.
- We drove the load capacitor,equivalent to 4 unit sized inverters,designed in previous experiment, in parallel, using this 1 bit full adder. The worst delay came from the path containing maximum number of gates, so we varied the inputs accordingly so that the input and output change occurs through that path and changes the output. Such a path was to change input number's bits (A or B).
- Adding the output capacitors induced more delay and degrade in the rise and fall time as expected.
- The 4 bit full adder was made by cascading 4 of this full 1 bit adder,connected in a manner, one's carry output giving input to the carry of next adder unit. This is called Ripple Carry Adder.
- For ripple carry adder the delay to obtain final output is the sum of delays in carry production of each of the stages and the delay in carry production of the last stage. So it is a slow adder. Better implementation is Carry Look Ahead Adder, which produces all the carries all together.
- The 4 bit register is made of an array of flip-flops. There are several types of registers available:
  - **PIPO:** Parallel Input Parallel Output. Least delay to obtain the input at output. Just a delay of 1 clock cycle. All the data (multiple bits are loaded all in 1 go).
  - **SIPO:** Serial Input Parallel Output. The data at input are loaded serially 1 by 1. But the loaded data (bits) is available as soon as loaded. For proper bit-place positioning $n \cdot T_{clk}$ time is required.
  - **PISO:** Parallel Input Serial Output. The n bits data is loaded all at once, but the data can only be accessed 1 bit at a time, serially.
  - **SISO:** Serial Input Serial Output. Bot input and output are serial. So takes the highest amount of time.
- The pipeline unit was made by inserting register before the input and after the output of the adder unit. In pipeline architectures, each stage is padded in input and output in this manner. This is done to isolate each stage. The latches ensure to hold the output of previous stage till the next active edge of clock signal is encountered, thus preventing the previous stage output interrupting the next stage output calculation process (by accepting the input directly automatically).
- That's why the whole pipeline system can operate at the delay of the slowest stage. The same was observed while finding the maximum operating frequency of the pipeline stage.

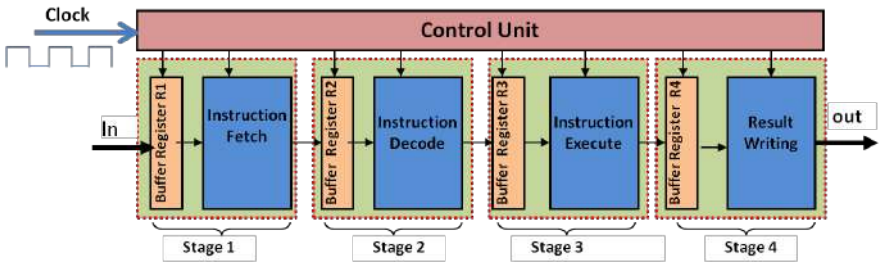


Figure 24: Typical Pipeline Architecture

Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 2

4 Bit Counter Design

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016

- **Group no.:** 35 (Tuesday)
- **Date of Submission:** 4-02-2025

Aim

1. Designing a 4 bit asynchronous down-counter using D flip-flop.

2. Designing a 4 bit synchronous up-counter using T-flip flop.

Theory

- A **Flip-flop** is a circuit used in digital electronics, that is used to store 1 bit of data. The inputs of a flip-flop are the data , clock signal and reset pin and output is Q and $\overline{Q}$. These store the input data at a particular edge of the clock signal (rising or falling edge) fed to these. The data stored can be reset to '0' by giving a high input to pin reset. There are 2 types of flip flops classified based on the active edge of clock signal: 1) Rising Edge triggered, 2) Falling Edge triggered

- **D-flip flop** is a special class of flip-flops in which the output is the input data itself,when we give a particular edge of clock signal. The state transition equation of this flip-flop can be written as:

$$Q_{n+1} = D_n$$

- **T-flip flop** is a class of flip flops in which the input pins are data, clock and when data is high the output state toggles at the active edge of the clock. When the data input is low the output state remains same.

$$Q_{n+1} = \begin{cases} Q_n, & T = 0 \\ \overline{Q_n}, & T = 1 \end{cases} \tag{1}$$

- Counters are a class fo circuits used in digital electronics,which are used to count number of pulses. These circuits only have a clock signal as input and the outputs are individual bits of the counter. Counters are made using a number of flip-flops, each representing 1 bit. There are 2 classes of counters based on the excitation of each flip flop:

  - **Asynchronous:** The clock signals of each flip-flop are different, not common. Each flip-flop triggers the next flip-flop sequentially, causing a ripple effect. So this class is also called **ripple counter**. The maximum operating frequency of flip-flop decreases by a factor of 0.5 for each bit index increase in flip-flop.
  - **Synchronous:** All flip-flops receive a common clock and hence are triggered at the same time. A logical circuit must be present which takes the current state of each flip-flop and generates the input for the next state and hence the output. All flip-flops operate at the same frequency.

  Another class division can be obtained on the basis of the counting

  - **Up-counter:** On each active edge of the clock signal, the output bits of the counter are such that it represent a larger integer than the previous state [incrementing by +1].
  - **Down-counter:** On each active edge of the clock signal, the output bits of the counter are such that it represent a smaller integer than the previous state [incrementing by -11].

  After reaching a certain value 'N-1', the output of the counter again resets back to '0000' [mod(N) counter]

Results/Observations

Part 1: 4 bit Asynchronous Down-counter

Structural Approach

- We utilized the behavioural model of D-flip flops and made appropriate connections between them (using the output of 1 flip-flop as the input of the other one) to make the counter.

```
module dff(
input wire data,
input wire clk,
input wire globalclk,
input wire rst,
output reg out
);

    always @(posedge globalclk)begin
        if(rst)begin
            out<=0;
        end
    end

    always @(posedge clk)begin
        if(rst)begin
            out<=0;
        end else begin
            out<=data;
        end
    end
endmodule
```

Figure 1: Code for the behavioural implementation of D-flip flop

```
module downcounter(
    input wire clk,
    input wire rst,
    output wire[3:0] out
    );

    wire [3:0]q;

    dff dff1(.data(~q[0]),.clk(clk),.globalclk(clk),.rst(rst),.out(q[0]));
    dff dff2(.data(~q[1]),.clk(q[0]),.globalclk(clk),.rst(rst),.out(q[1]));
    dff dff3(.data(~q[2]),.clk(q[1]),.globalclk(clk),.rst(rst),.out(q[2]));
    dff dff4(.data(~q[3]),.clk(q[2]),.globalclk(clk),.rst(rst),.out(q[3]));

    assign out = q;
    //{q[0],q[1],q[2],q[3]};
endmodule
```

Figure 2: Code for structural implementation of 4 bit asynchronous down-counter using the D-flipflops

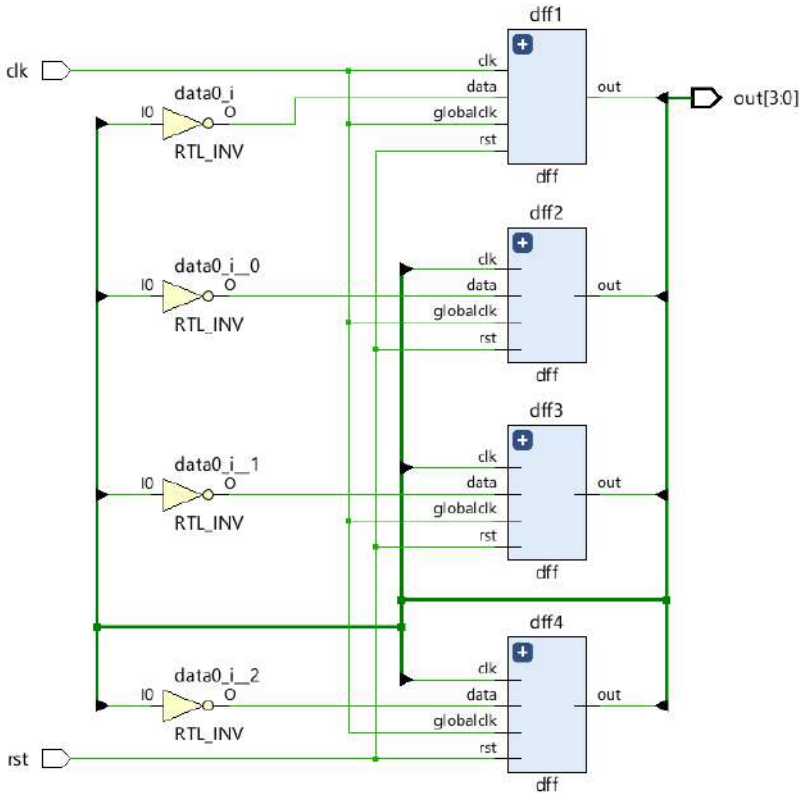


Figure 3: 4 bit down-counter Schematic

```
module downcountertb;
    reg clk;
    reg rst;
    wire [3:0] out;

    downcounter uut (.clk(clk),.rst(rst),.out(out));

    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end

    initial begin
        $monitor("Time: %0d | Reset: %b | Counter Output: %b", $time, rst, out);

        rst = 1;
        #20;
        rst = 0;
        #200; //running the clock for a while;
        // End simulation
        $stop;
    end

endmodule
```

Figure 4: Testbench code for this counter [timescale: unit 1ns, precision: 1ps]

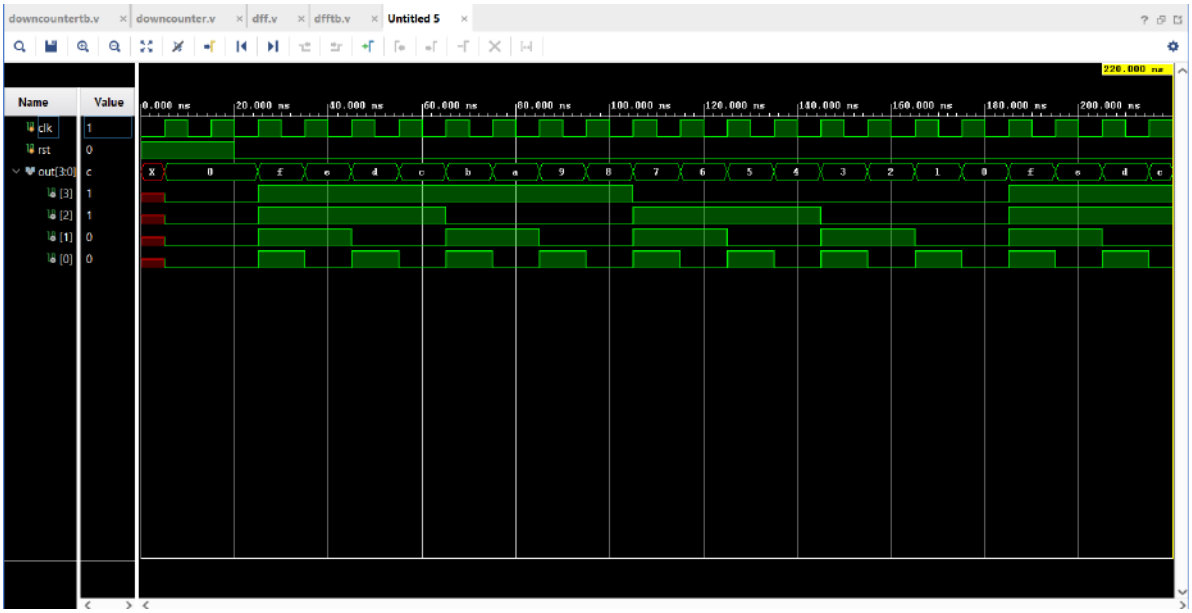


Figure 5: The waveform obtained for this counter

```
Time: 0 | Reset: 1 | Counter Output: xxxx
Time: 5 | Reset: 1 | Counter Output: 0000
Time: 20 | Reset: 0 | Counter Output: 0000
Time: 25 | Reset: 0 | Counter Output: 1111
Time: 35 | Reset: 0 | Counter Output: 1110
Time: 45 | Reset: 0 | Counter Output: 1101
Time: 55 | Reset: 0 | Counter Output: 1100
Time: 65 | Reset: 0 | Counter Output: 1011
Time: 75 | Reset: 0 | Counter Output: 1010
Time: 85 | Reset: 0 | Counter Output: 1001
Time: 95 | Reset: 0 | Counter Output: 1000
Time: 105 | Reset: 0 | Counter Output: 0111
Time: 115 | Reset: 0 | Counter Output: 0110
Time: 125 | Reset: 0 | Counter Output: 0101
Time: 135 | Reset: 0 | Counter Output: 0100
Time: 145 | Reset: 0 | Counter Output: 0011
Time: 155 | Reset: 0 | Counter Output: 0010
Time: 165 | Reset: 0 | Counter Output: 0001
Time: 175 | Reset: 0 | Counter Output: 0000
Time: 185 | Reset: 0 | Counter Output: 1111
Time: 195 | Reset: 0 | Counter Output: 1110
Time: 205 | Reset: 0 | Counter Output: 1101
Time: 215 | Reset: 0 | Counter Output: 1100
$stop called at time : 220 ns : File "E:/Vivado Project/Exp2_part1/Exp2_part1.srcs/sim_1/new/downcountertb.v" Line 43
```

Figure 6: Simulation Console output for the asynchronous down counter

Part 2: 4 bit Synchronous Up-counter

Here using the T-flip flop behavioural modules and the logic circuit to obtain the input of each flip-flop we make the synchronous up-counter.

- **T-FLIP FLOP:**

```
21
22
23  module tff(
24      input wire clk,
25      input wire T,
26      input wire rst,
27      output reg Q
28      );
29
30      initial begin
31          Q=0;
32      end
33
34      always@(posedge clk) begin
35          if(rst) begin
36              Q<=0;
37          end else begin
38              if(T) begin
39                  Q<=~Q;
40              end
41          end
42      end
43
44  endmodule
45
```

Figure 7: T-flip flop behavioural implementation

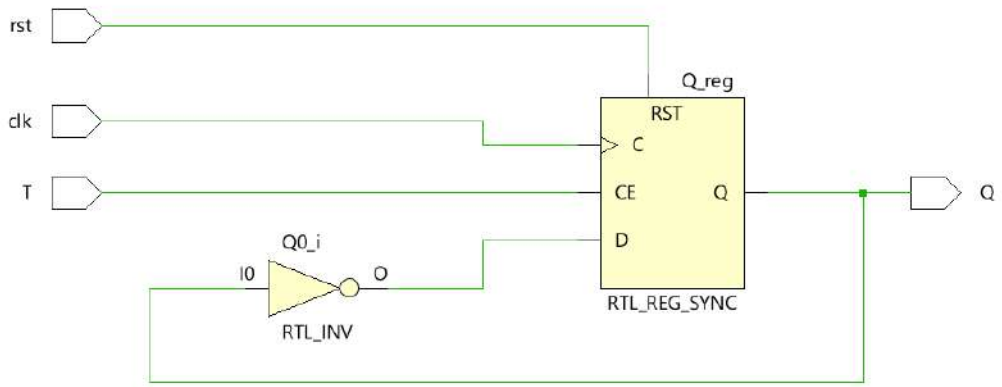


Figure 8: T-flipflop Behavioral Schematic

- **Testing and output of the T-flip flop:**

```
22
23  module tfftb;
24      reg clk,rst,T;
25      wire Q;
26
27      //instantiate the uut
28      tff flop(.T(T),.clk(clk),.rst(rst),.Q(Q));
29
30      //time period:10ns
31      initial
32      begin
33          clk=0;
34          forever #5 clk=~clk;
35      end
36
37      initial begin
38          $monitor("time: %0d  rst=%b T=%b Q=%b",$time,rst,T,Q);
39          rst=1;T=0;
40          #10 rst=0;T=0;
41          #10 T=1;
42          #10 T=1;
43          #10 T=1;
44          $stop;
45
46      end
47
```

Figure 9: testbench for T-flip flop

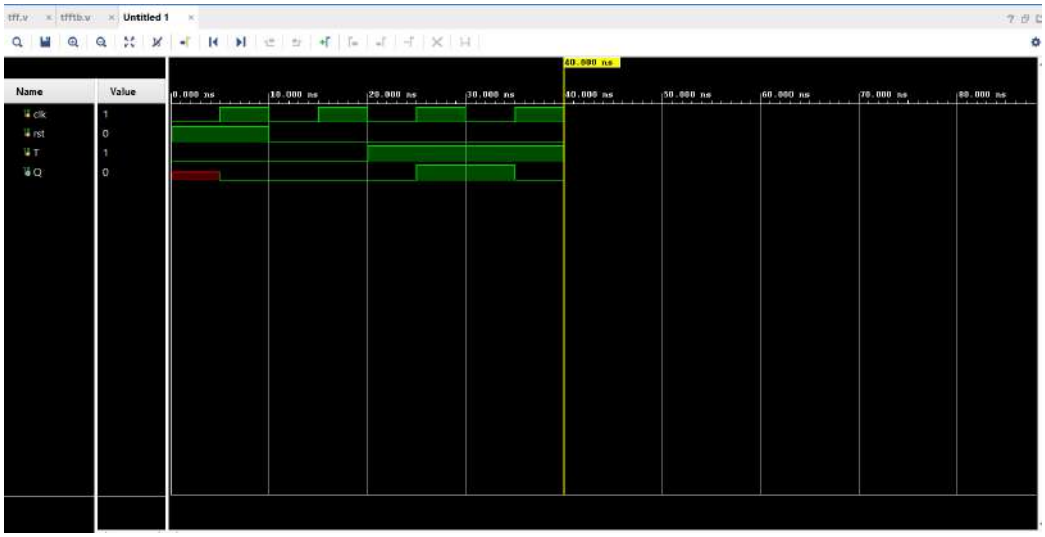


Figure 10: Output waveform of the T-flip flop for the testbench

```
time: 0  rst=1 T=0 Q=x
time: 5  rst=1 T=0 Q=0
time: 10  rst=0 T=0 Q=0
time: 20  rst=0 T=1 Q=0
time: 25  rst=0 T=1 Q=1
time: 35  rst=0 T=1 Q=0
$stop called at time : 40 ns : File "E:/Vivado_Project/Exp2_part2/Exp2_part2.srcs/sim_1/new/tfftb.v" Line 44
```

Figure 11: Output of simulation console for T-flip flop on the testbench

- **4 bit Up-counter**

```
22
23      module upcounter(
24          input wire clk,
25          input wire rst,
26          output wire[3:0] out
27          );
28          wire q[3:0];
29          tff tff1(.clk(clk),.T(1),.rst(rst),.Q(q[0]));
30          tff tff2(.clk(clk),.T(q[0]),.rst(rst),.Q(q[1]));
31          tff tff3(.clk(clk),.T(q[1]&q[0]),.rst(rst),.Q(q[2]));
32          tff tff4(.clk(clk),.T(q[2]&q[1]&q[0]),.rst(rst),.Q(q[3]));
33          assign out = {q[3],q[2],q[1],q[0]};
34      endmodule
35
```

Figure 12: Verilog code for the up-counter

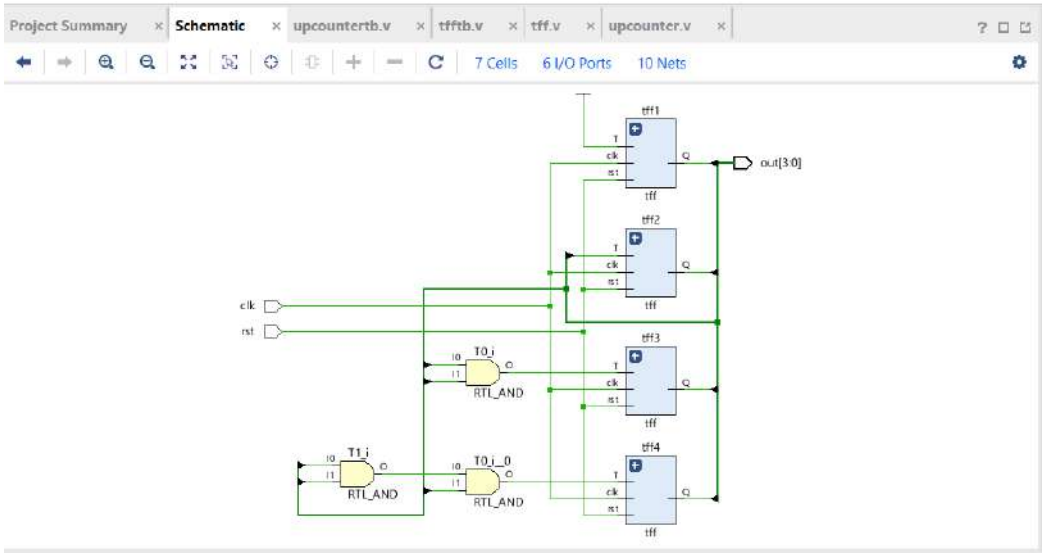


Figure 13: Schematic of the up-counter

- **Testing and output of the counter:**

```
23  module upcountertb;
24      reg clk;
25      reg rst;
26      wire [3:0] out;
27
28      upcounter uut (.clk(clk),.rst(rst),.out(out));
29
30      initial begin
31          clk = 0;
32          forever #5 clk = ~clk;
33      end
34
35      initial begin
36          $monitor("Time: %0d | Reset: %b | Counter Output: %b", $time, rst, out);
37
38          rst = 1;
39          #20;
40          rst = 0;
41          #200; //running the clock for a while;
42          // End simulation
43          $stop;
44      end
45
46  endmodule
47
```

Figure 14: up-counter testbench code

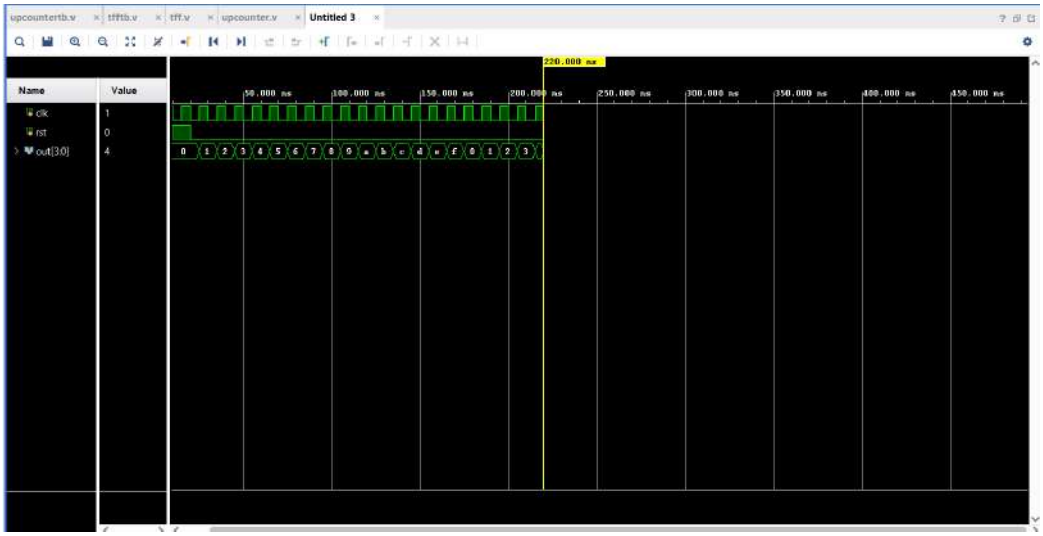


Figure 15: up-counter simulation result waveform

```
Time: 0 | Reset: 1 | Counter Output: 0000
Time: 20 | Reset: 0 | Counter Output: 0000
Time: 25 | Reset: 0 | Counter Output: 0001
Time: 35 | Reset: 0 | Counter Output: 0010
Time: 45 | Reset: 0 | Counter Output: 0011
Time: 55 | Reset: 0 | Counter Output: 0100
Time: 65 | Reset: 0 | Counter Output: 0101
Time: 75 | Reset: 0 | Counter Output: 0110
Time: 85 | Reset: 0 | Counter Output: 0111
Time: 95 | Reset: 0 | Counter Output: 1000
Time: 105 | Reset: 0 | Counter Output: 1001
Time: 115 | Reset: 0 | Counter Output: 1010
Time: 125 | Reset: 0 | Counter Output: 1011
Time: 135 | Reset: 0 | Counter Output: 1100
Time: 145 | Reset: 0 | Counter Output: 1101
Time: 155 | Reset: 0 | Counter Output: 1110
Time: 165 | Reset: 0 | Counter Output: 1111
Time: 175 | Reset: 0 | Counter Output: 0000
Time: 185 | Reset: 0 | Counter Output: 0001
Time: 195 | Reset: 0 | Counter Output: 0010
Time: 205 | Reset: 0 | Counter Output: 0011
Time: 215 | Reset: 0 | Counter Output: 0100
$stop called at time : 220 ns : File "E:/Vivado Project/Exp2 part2/Exp2 part2.srcs/sim 1/new/upcountertb.v" Line 43
```

Figure 16: up-counter simulation console result

Discussion Chiradip Biswas 22EC30016

Part-1:

- In the 1st part of the experiment, the asynchronous down-counter was made by feeding the flip-flops' clock signal with output pin of the previous stage flip-flop. It was observed in the flip-flop output stages that higher indices were operating with quite less frequency compared to the lower stages. $f_n = \frac{f_0}{2^n}$ where the f_n is the operating frequency of the nth bit and the f_0 is that of the LSB.

- The output of flip-flops change from 1 to 0 or 0 to 1, so the next cycle input (D_{n+1}) to each flip-flop was set to be $\overline{Q_n}$. Structurally if we make the counter by feeding the clocks of each flip-flop in the aforementioned manner and the data input of each is given like this, we will make an asynchronous up-counter. So we have to take the complement of each flip-flop output to obtain the down counter, starting from state '1111'.

- But as we need to reset each of the flip-flop to initialize all flip-flops output to 0, we needed a clock pulse to be sent to each of the flip-flop for that for initialization. Also it set the next input to each of the flip-flop as '1'. So the output bits of the counter were acting as the downcounter by default even without taking the counter output as the negation of each bit, keeping the 1st counter's bit as LSB.

- If the flip-flops are not reset at the initial stage the outputs of each one will continue to remain undefined.

- For practical asynchronous counters there will be a clock to Q delay in each flip flop. As we feed the clock of each flip flop from the output of previous flip flop, the net clock to Q delay adds up as the stage increases [for 1 flip-flop if delay is t_{cq}, for the nth bit, the delay will be $n \cdot t_{cq}$]. For proper working of the flip flop, this delay should be less than the clock period.

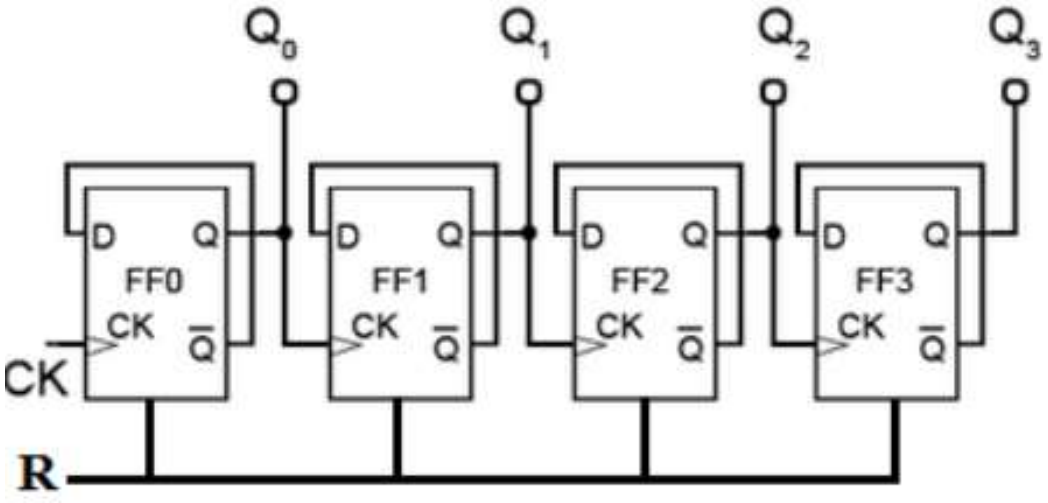


Figure 17: 4 bit asynchronous down counter

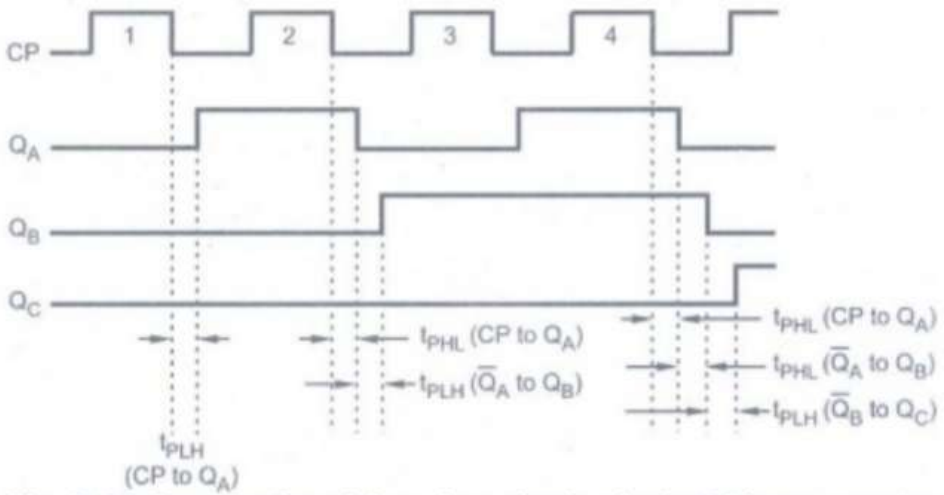


Fig. 5.2.3 Propagation delays in a ripple clocked binary counter

Figure 18: The clock to Q delay build up in asynchronous counter

Part-2:

- In the 2nd part of the experiment, the synchronous up-counter was made using T-flip flop. The T-flip flop can be derive from J K flip-flop by setting both J and K pins to '1', so that the output state toggles at each active edge of the clock.
- In this counter also initially all the flip-flops needed to be reset, to start the counting. The clock signal of each of them is common. In synchronous counter circuits, the next state of any particular state should be set specifically, using logical circuits, where inputs are the current state bits.
- The input data to each flip-flop is obtained by taking AND of the previous 2 stages' flip-flops' outputs.

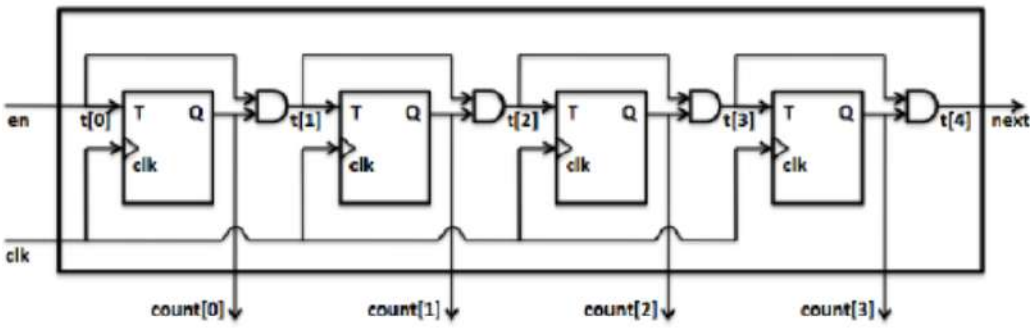


Figure 19: 4 bit up-counter using T flip-flop

Discussion Atif Faizan 22EC10096

- A synchronous counter is a type of counter where all the flip flops are controlled by a single common clock signal. All the flip flops in the counter change their states simultaneously, leading to synchronized counting.
- Advantage of using synchronous counter is that it works faster because all the flip flops change the state at the same time. There are fewer timing errors since everything is synchronized. There is no propagation delay and hence it is best for high speed applications.
- Disadvantage of using synchronous counter is that it consumes high power since all the flip flops change at the same time and the circuit logic is complex.
- Asynchronous counter is also called ripple counter. It is a type of counter where each flip flop is triggered by the output of the previous one, not by a common clock. This leads to delay as each flip-flop changes state in sequence, creating a "ripple" effect.
- Advantage of using asynchronous counter is that it consumes less power and its circuit implementation is easier.
- Ripple Effect is the main disadvantage of using asynchronous flip flop. The delay from flipflop to flipflop can cause timing errors, especially if the counter is used at high speeds.
- In the experiment before starting the clock all the flip flops were to be initialized with 0 otherwise the output will be undetermined.
- In the first part of the experiment we made a D flip flop following the behavioral approach and using these d flip flops as structures and appropriate combinatorial logic we developed the 4 bit asynchronous down counter.
- Similarly, in the 2nd part of the experiment we made a T flip flop following the behavioral approach and using these flip flops as structures and appropriate combinatorial logic we developed the 4 bit synchronous up counter.

Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 3

SEQUENCE DETECTOR USING FSM

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016
- **Group no.:** 35 (Tuesday)
- **Date of Submission:** 11-02-2025

Aim

1. Design a Mealy Sequence Detector, which detects the sequence 100110. Demonstrate thorough the test bench waveform.

2. Design a Moore Sequence Detector which detects the sequence 110010. Demonstrate through test bench waveforms.

Theory

- For combinational designs, the output value completely depends on the present value of the inputs and for sequential designs, output value not only depends on present input but also depends on its previously stored value i.e. past behavior of the design. These sequential designs are formally knowns as finite-state machines that have a fixed number of states. In sequential designs or FSM, a clock signal serves the purpose to control FSM operation. There are two ways to design FSMs, namely Mealy Machine and Moore Machine.

- Both Mealy and Moore machines can be used to design sequence detector logic. Further, these machines are classified as :

  - **Overlapping sequence detector:** Final bits of the sequence can be the start of another sequence. Thus, it allows overlap.
  - **Non-overlapping sequence detector:** Once sequence detection is completed, another sequence detection can be started without any overlap.

- **Mealy Machine:**

  - In this machine, the output produced by the state machine depends on the input events fed to the state machine AND presents an active state of the state machine.
  - The output is not produced inside the state.
  - Outputs depend on the current state and the inputs.
  - Outputs can change asynchronously during the state.
  - The output is associated with the transitions between states.
  - Usually one less state is needed than the number of states needed in Moore Model

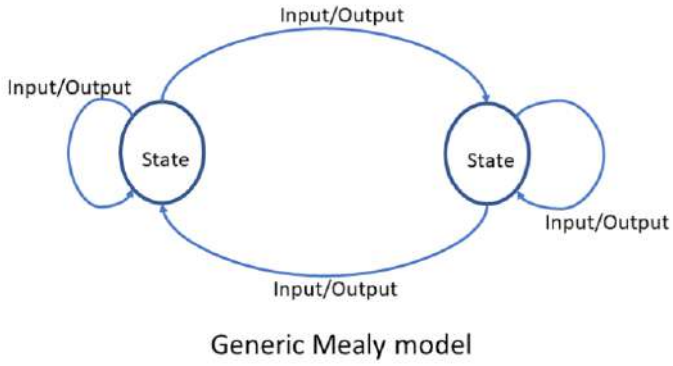


Figure 1: Generic Meanly Model

- **Moore Model:**

  - In this state machine, the output is determined only by the present active state of the state machine and not by any input events.
  - No output during state transition.
  - Outputs depend only on the current state.
  - Outputs change synchronously at the beginning of each state.

– The output is associated with the states.

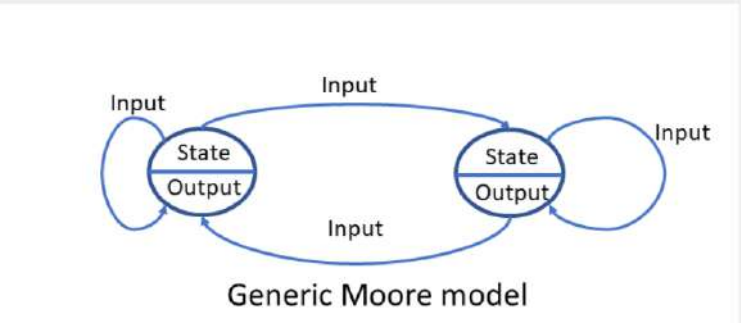


Figure 2: Generic Moore Model

- Following are the differences between Meanly and Moore Model:

Mealy Machine	Moore Machine
Output depends on the present state and current input	Output only depends on the present state. It is independent of current input
Requires fewer states to design	Requires more states to design
Reacts faster to the input and requires less hardware implementation	More logic is required to decode the output.
Difficult to design	Easy to design
Asynchronous output generation even though the states changes in synchronous to the clock.	Synchronous output and state generation w.r.t. clock

Figure 3: Mealy Model vs Moore Model

Logic and Design

- **Part1:** In this part of the experiment we were required to make an overlapping Mealy sequence detector to detect the sequence 100110. We first designed the state transition diagram for the overlapping case which is as follows:

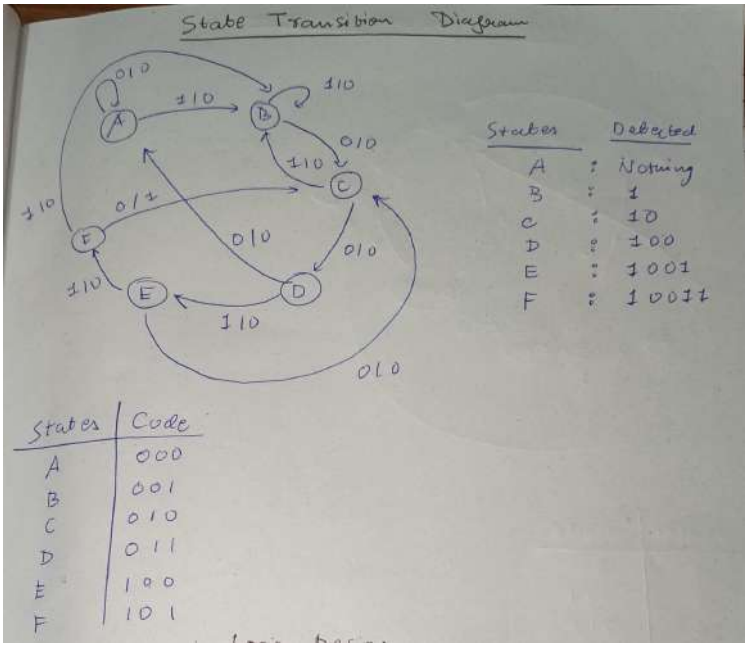


Figure 4: State Transition Diagram Mealy Model (100110 Detection)

As we can clearly see that 6 states namely A,B,C,D,E and F are needed. 3 D flip flops are enough to realize these 6 states and hence we designed the logic using three flip flops as follows:

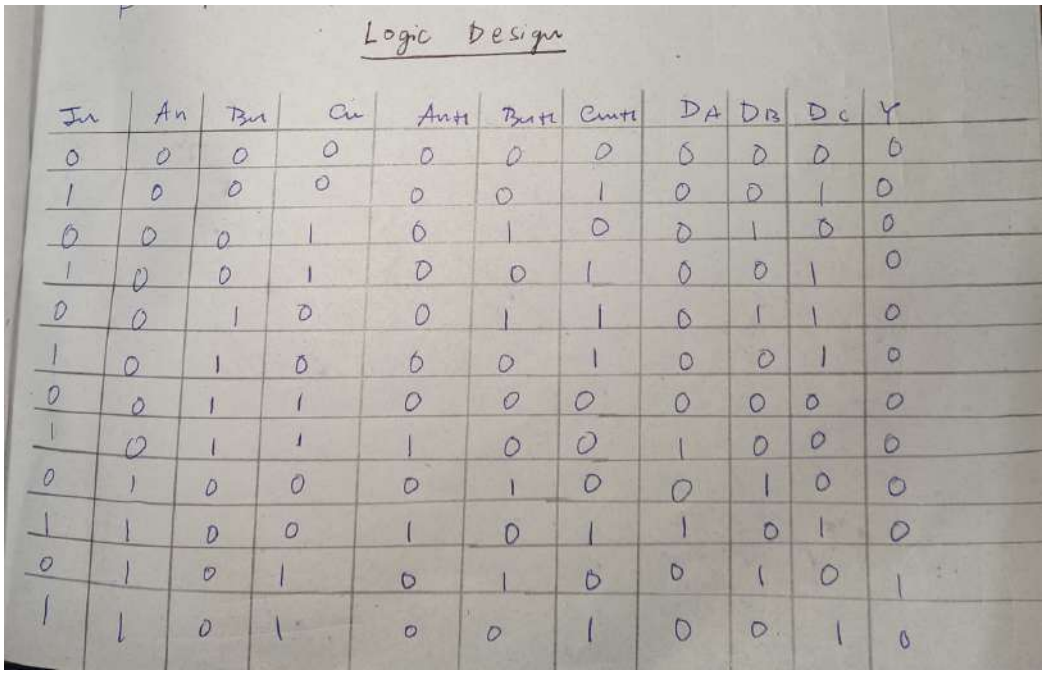


Figure 5: Logic Design

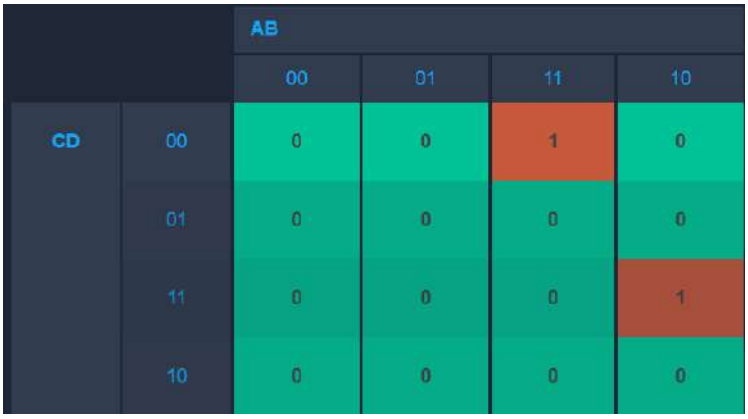


Figure 6: Kmap for D_A

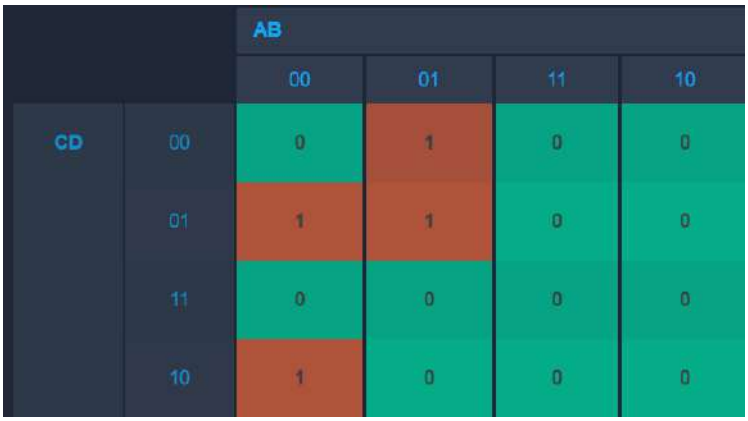


Figure 7: Kmap for D_B

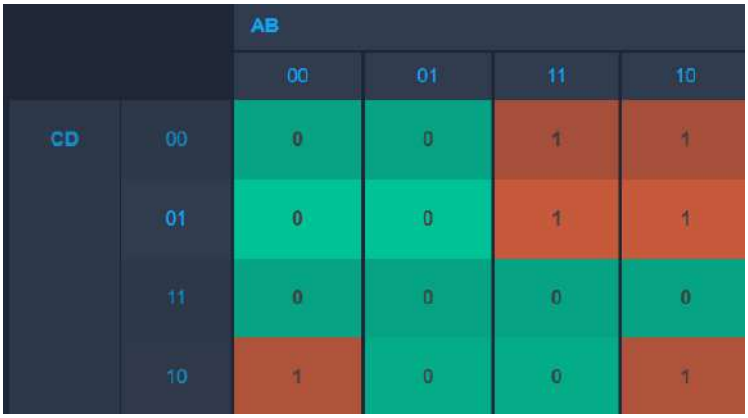


Figure 8: Kmap for D_C

		AB			
		00	01	11	10
CD	00	0	0	0	0
	01	0	1	0	0
	11	0	0	0	0
	10	0	0	0	0

Figure 9: Kmap for Y

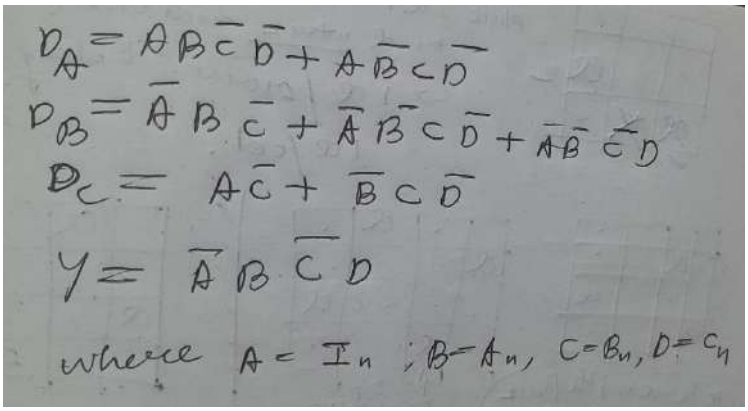


Figure 10: Final Logic

- **Part 2:** In this part of the experiment we were required to make an overlapping Moore sequence detector to detect the sequence 110010. We first designed the state transition diagram for the overlapping case which is as follows:

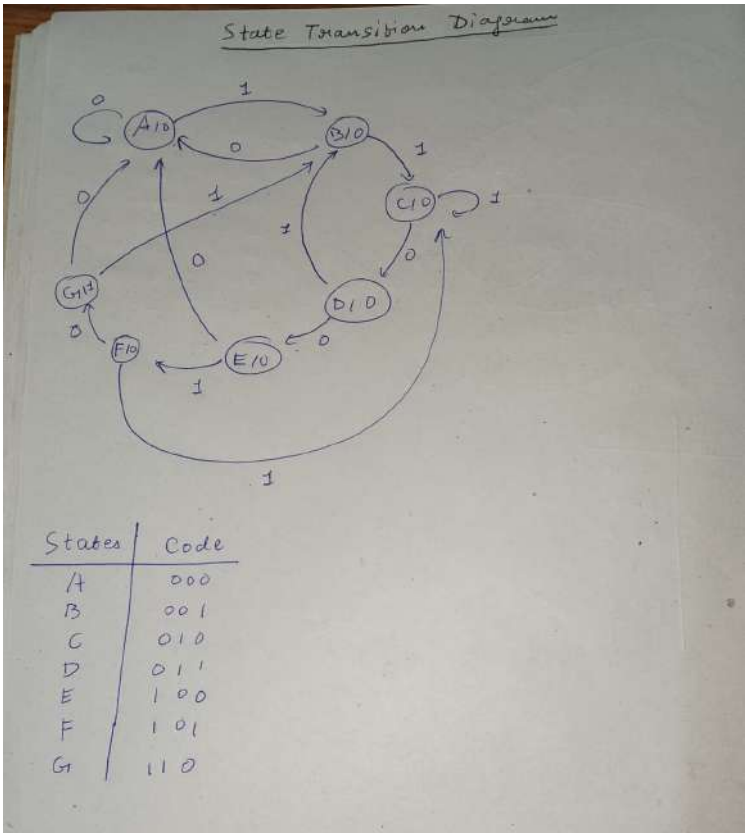


Figure 11: State Transition Diagram Moore Model (110010 Detection)

As we can clearly see that 6 states namely A,B,C,D,E and F are needed. 3 D flip flops are enough to realize these 6 states and hence we designed the logic using three flip flops as follows:

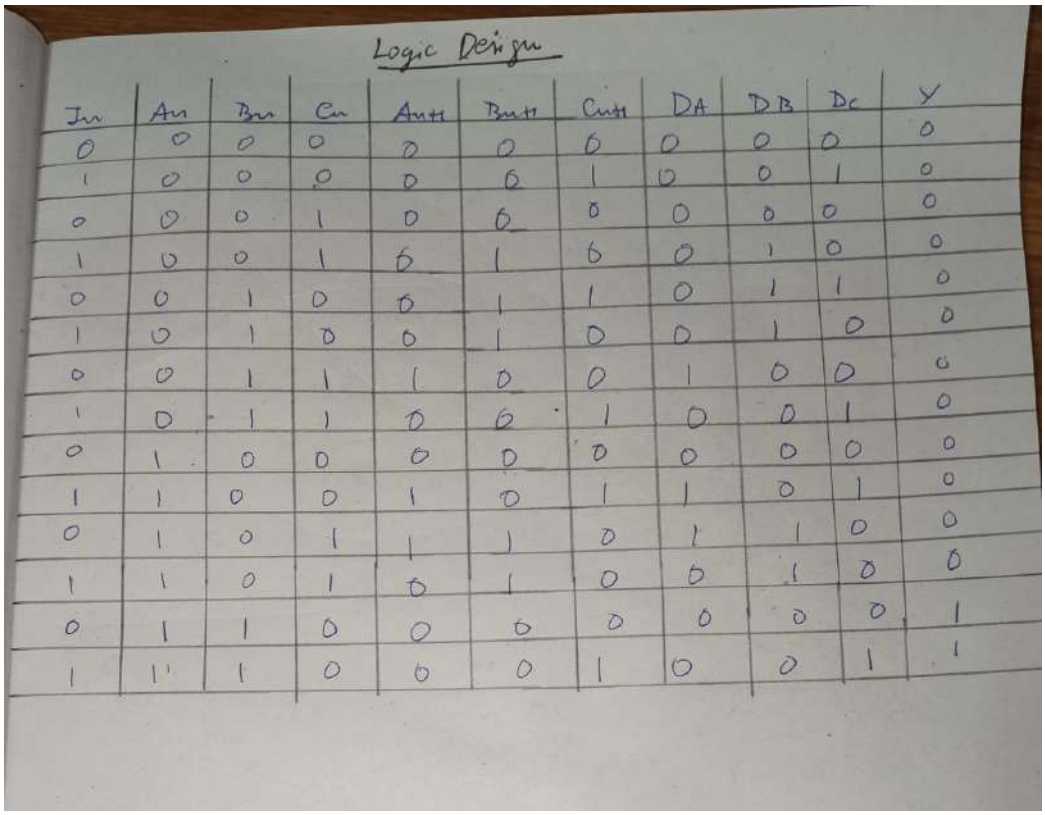


Figure 12: Logic Design

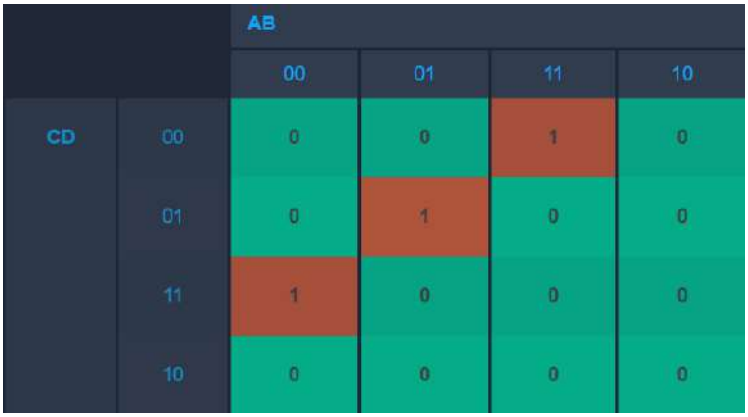


Figure 13: Kmap for D_A

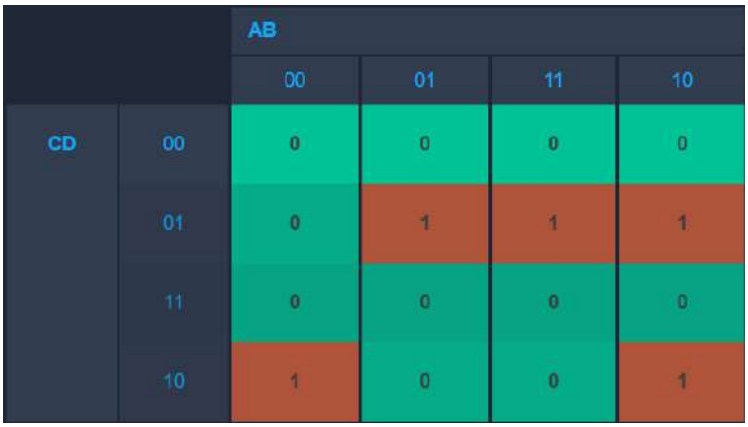


Figure 14: Kmap for D_B

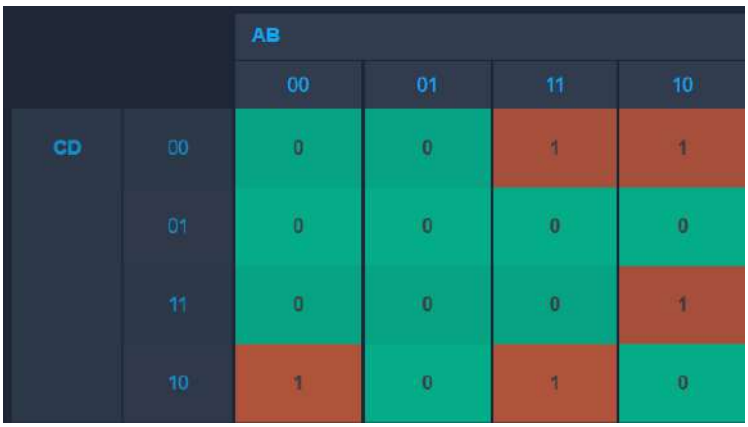


Figure 15: Kmap for D_C

		AB			
		00	01	11	10
CD	00	0	0	0	0
	01	0	0	0	0
	11	0	0	0	0
	10	0	1	1	0

Figure 16: Kmap for Y

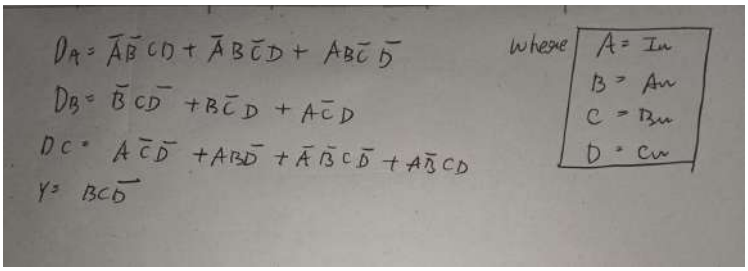


Figure 17: Final Logic

Results/Observations

Part A: Mealy Sequence Detector (100110 Detection)

- We used the following code for making the Mealy Sequence Detector using d flip flops as structural units:

```
module state_machine(
input wire data,
input wire clk,
output wire out
    );

    wire an,bn,cn;
    wire an_bar,bn_bar,cn_bar;

    not(bn_bar,bn);
    not(cn_bar,cn);
    not(an_bar,an);

    wire da_1=data & an & bn_bar & cn_bar;
    wire da_2=data & an_bar & bn & cn;
    wire da= da_1 | da_2;

    wire db_1=(~data) & an & bn_bar;
    wire db_2=(~data) & an_bar & (bn ^ cn);
    wire db=db_1 | db_2;

    wire dc = (data & bn_bar) | (an_bar & bn & cn_bar);


    dff Fa(.data(da),.clk(clk),.rst(0),.out(an));
    dff Fb(.data(db),.clk(clk),.rst(0),.out(bn));
    dff Fc(.data(dc),.clk(clk),.rst(0),.out(cn));

    assign out=(~data) & an & bn_bar & cn;
endmodule
```

Figure 18: Code

- We used the following test bench for testing the Sequence Detector developed:

```
module statemachine_tb;
    reg clk;
    reg data;
    wire out;
    // Instantiate the Sequence Detector module
    state_machine uut (
        .clk(clk),
        .data(data),
        .out(out)
    );
    // Generate Clock Signal (50% Duty Cycle)
    always #5 clk = ~clk;  // 10ns period
    initial begin
        // Initialize signals
        $monitor("Time = %0t | Data = %b | Output = %b", $time, data, out);
        clk = 0;
        data = 0;
        // Apply test sequence: 100110 should produce output = 1
        #10 data = 1;
        #10 data = 0;
        #10 data = 0;
        #10 data = 1;
        #10 data = 0;
        #10 data = 0;
        #10 data = 1;
        #10 data = 1;
        #10 data = 0;
        #10 data = 1;
        #10 data = 1;
        #10 data = 0;  // Another match
        #10 data = 0; // Random extra bits
        #10 data = 0;
```

Figure 19: Test Bench

- The following is the Simulation Result:

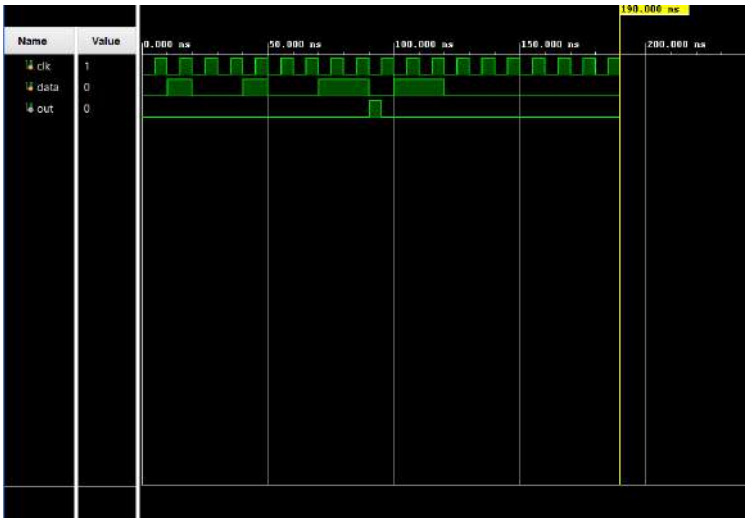


Figure 20: Simulation Result

- The following is the output of the console:

```
# run 1000ns
Time = 0 | Data = 0 | Output = 0
Time = 10000 | Data = 1 | Output = 0
Time = 20000 | Data = 0 | Output = 0
Time = 40000 | Data = 1 | Output = 0
Time = 60000 | Data = 0 | Output = 1
Time = 65000 | Data = 0 | Output = 0
Time = 80000 | Data = 1 | Output = 0
Time = 100000 | Data = 0 | Output = 1
Time = 105000 | Data = 0 | Output = 0
$stop called at time : 190 ns : File "C:/Users/Chiradip Biswas/exp3_part1/exp3_part1.srcs/sim_1/new/statemachine_tb.v" Line 55
```

Figure 21: Console Output

- The following is the schematic of the Model:

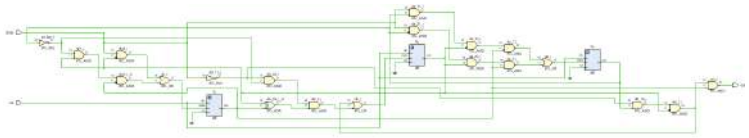


Figure 22: Schematic

Part B: Moore Sequence Detector (110010 Detection)

- We used the following code for making the Moore Sequence Detector using d flip flops as structural units:

```
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module state_machine(
input wire data,
input wire clk,
output wire out
    );

    wire an,bn,cn;
    wire an_bar,bn_bar,cn_bar;

    not(bn_bar,bn);
    not(cn_bar,cn);
    not(an_bar,an);

    wire da = (data & an & bn_bar & cn_bar) | (~data & an & bn_bar & cn) | (~data & an_bar & bn & cn);
    wire db = (an_bar & bn & cn_bar) | ((data | an) & (bn_bar & cn));
    wire dc = (data & bn_bar & cn_bar) | (data & an & cn_bar) | (data & an_bar & bn & cn) | (~data & an_bar & bn & cn_bar);


    dff Fa(.data(da),.clk(clk),.rst(0),.out(an));
    dff Fb(.data(db),.clk(clk),.rst(0),.out(bn));
    dff Fc(.data(dc),.clk(clk),.rst(0),.out(cn));

    assign out= an & bn & cn_bar;
endmodule
```

Figure 23: Code

- We used the following test bench for testing the Sequence Detector developed:

```
module state_machinetb;
    reg clk;
    reg data;
    wire out;
    // Instantiate the Sequence Detector module
    state_machine uut (
        .clk(clk),
        .data(data),
        .out(out)
    );
    // Generate Clock Signal (50% Duty Cycle)
    always #5 clk = ~clk;  // 10ns period
    initial begin
        // Initialize signals
        $monitor("Time = %0t | Data = %b | Output = %b", $time, data, out);
        clk = 0;
        data = 0;
        // Apply test sequence: 100110 should produce output = 1
        #10 data = 1;
        #10 data = 1;
        #10 data = 0;
        #10 data = 0;
        #10 data = 1;
        #10 data = 1;
        #10 data = 1;
        #10 data = 0;
        #10 data = 0;
        #10 data = 1;
        #10 data = 0;
        #10 data = 0;  // Another match
        #10 data = 0; // Random extra bits
```

Figure 24: Test Bench

- The following is the Simulation Result:

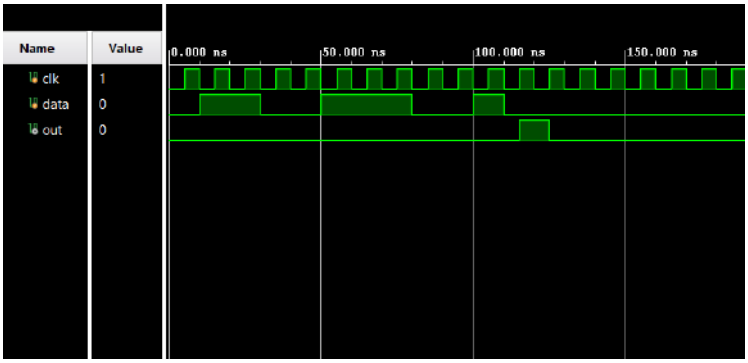


Figure 25: Simulation Result

- The following is the output of the console:

```
# run 1000ns
Time = 0 | Data = 0 | Output = 0
Time = 10000 | Data = 1 | Output = 0
Time = 20000 | Data = 0 | Output = 0
Time = 50000 | Data = 1 | Output = 0
Time = 80000 | Data = 0 | Output = 0
Time = 100000 | Data = 1 | Output = 0
Time = 110000 | Data = 0 | Output = 0
Time = 115000 | Data = 0 | Output = 1
Time = 125000 | Data = 0 | Output = 0
$stop called at time : 190 ns : File "C:/Users/Chiradip Biswas/exp3 part2/exp3 part2.srcs/sim 1/new/state machinetb.v" Line 55
```

Figure 26: Console Output

- The following is the schematic of the Model:

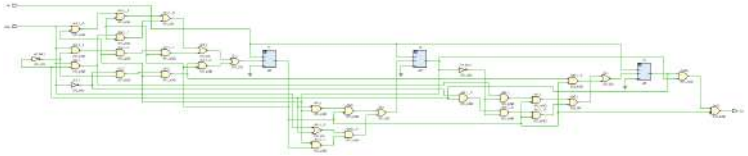


Figure 27: Schematic

Discussion Atif Faizan 22EC10096

- In this experiment we designed Mealy and Moore Sequence Detector to detect 100110 and 110010 respectively considering overlapping sequence.
- We utilized d flip flops as structural units and wrote the code of d flip flop with behavioral approach.
- We adopted the general algorithm to design the sequence detector. We first decided the sates and designed the state transition diagram. Then we constructed the state table for 3 dff. We determined and minimized the connections to be made to D_A, D_B, D_C and Y using Kmaps and hence developed a detector.
- In Meanly Model the output was non dependent on the state but on the transition but in Moore Model the output was dependent on the state.
- Because of this in Moore Model there was one cycle delay in the indication of the presence of sequence than where the actual sequence occurred while in Mealy Model the indication of the presence of sequence was in perfect sync with the occurrence of the sequence.
- In Mealy model the output was dependent on the input while in Moore Model was output was independent of the input.
- Since in Mealy Model the output depends on input, there are higher chances of error because input value is available simultaneously to the output but other values on which output depend may be available after certain gate delays leading to the error. No such error exits in Moore Model.

- Number of states in Moore Model is generally one more than the number of states that would have been required if we realize with mealy model.
- Hence there are trade off in number of states, delay and error. While Mealy Model gives least number of states and least delay, it is prone to error. On the other hand although Moore Model utilizes one extra state and gives output delayed by one clock cycle, it is resistant to error.

Discussion Chiradip Biswas 22EC30016

- In this experiment we made sequence detectors by implementing Finite State Machine and using the number of characters detected in the sequence as states of it. The state transitions occur based upon the input data bit, it receives in each clock cycle.
- For representing the states we use D flip-flops and their outputs to represent the states. In the 1st part, to make a Mealy state machine for detecting a sequence "100110" we had to use 6 states, as there were 6 bits in the sequence, 1 state should be present to represent number of detected bits so far, which ranges from 0 to 6, and so 6 transitions will be present. In Mealy model, the output comes as soon as the input is applied, and as there are 6 transitions, it validates that we will require 6 states. So number of flip-flops used $ceil(log_2(6)) = 3$.
- For the 2nd part we made a Moore model, for which the output solely depends on the state the model is in. The sequence to be detected was "110010". There are 6 bits in the sequence, and the Moore state machine should give the output =1 only when all the bits are detected (Not like Mealy model, giving output along with the input). So there has to be 7 states in this machine.
- In the Moore model, as a final state is required where all the bits of the sequence are detected, to give the output=1, it takes 1 extra state compared to that taken by Mealy model.
- Due to this reason, there is a 1 clock cycle delay observed in Moore state machine compared to Mealy state machine when solving the same problem. The Mealy model gives output as soon as the input arrives, and the Moore model gives the output after it goes to the final state.
- On the other hand, there is a probability of error in Mealy state machine, as the output directly depends on the input, the logic to obtain output is implemented using logic gates, and there are finite delays in these logic gates. So there can be a chance that the required proper output for the input is not produced within the clock cycle. This can cause error.
- In Moore model, as the output just depends on the state, there is less chance of error as the time-bound now is 2 times the clock cycle.
- For real-life realization of the state machines and flip-flops, the setup time and hold time constraints also come into picture and so the input should come at least t_{setup} before the active clock edge and remain constant for t_{hold} after the active edge.

Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 4

DIVISIBILITY BY 5 DETECTOR USING FSM

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016
- **Group no.:** 35 (Tuesday)
- **Date of Submission:** 17-02-2025

Aim

Task 1: Design an FSM in Verilog, which outputs 1 for an aggregate serial binary string if it is divisible by 5; otherwise, it outputs 0.

- Structural Mealy model realization.
- Behavioral/FSM encoding-based model to implement the Moore model.

Task 2: Design an FSM which outputs 1 for an aggregate serial binary string that detects if the last 3 bits are alternating (101 or 010). Implement Moore model realization.

Theory

- For combinational designs, the output value completely depends on the present value of the inputs and for sequential designs, output value not only depends on present input but also depends on its previously stored value i.e. past behavior of the design. These sequential designs are formally knowns as finite-state machines that have a fixed number of states. In sequential designs or FSM, a clock signal serves the purpose to control FSM operation. There are two ways to design FSMs, namely Mealy Machine and Moore Machine.
- Both Mealy and Moore machines can be used to design sequence detector logic. Further, these machines are classified as :
  - **Overlapping sequence detector:** Final bits of the sequence can be the start of another sequence. Thus, it allows overlap.
  - **Non-overlapping sequence detector:** Once sequence detection is completed, another sequence detection can be started without any overlap.
- **Mealy Machine:**
  - In this machine, the output produced by the state machine depends on the input events fed to the state machine AND presents an active state of the state machine.
  - The output is not produced inside the state.
  - Outputs depend on the current state and the inputs.
  - Outputs can change asynchronously during the state.
  - The output is associated with the transitions between states.
  - Usually one less state is needed than the number of states needed in Moore Model

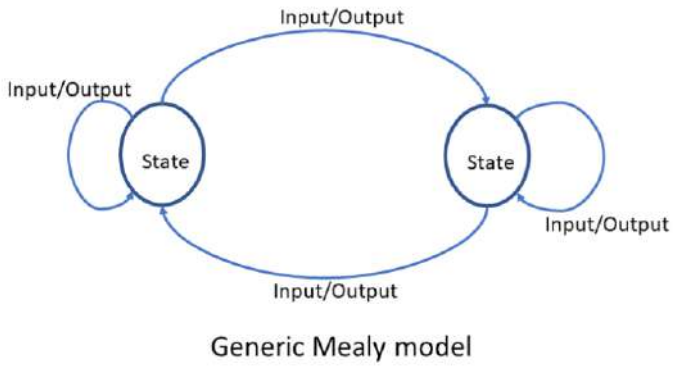


Figure 1: Generic Meanly Model

- **Moore Model:**
  - In this state machine, the output is determined only by the present active state of the state machine and not by any input events.
  - No output during state transition.
  - Outputs depend only on the current state.
  - Outputs change synchronously at the beginning of each state.
  - The output is associated with the states.

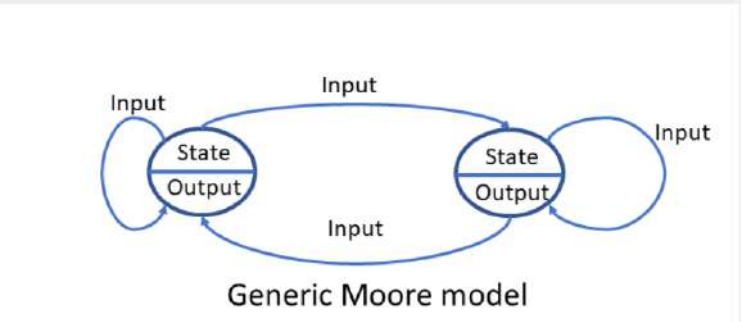


Figure 2: Generic Moore Model

- Following are the differences between Meanly and Moore Model:

Mealy Machine	Moore Machine
Output depends on the present state and current input	Output only depends on the present state. It is independent of current input
Requires fewer states to design	Requires more states to design
Reacts faster to the input and requires less hardware implementation	More logic is required to decode the output.
Difficult to design	Easy to design
Asynchronous output generation even though the states changes in synchronous to the clock.	Synchronous output and state generation w.r.t. clock

Figure 3: Mealy Model vs Moore Model

Logic and Design

- **Part1: Mealy Model** In this part of the experiment we were required to make an FSM that gives 1 if a serial binary bitstream is divisible by 5.This was done by making a Mealy state machine. We first designed the state transition diagram for the overlapping case which is as follows:

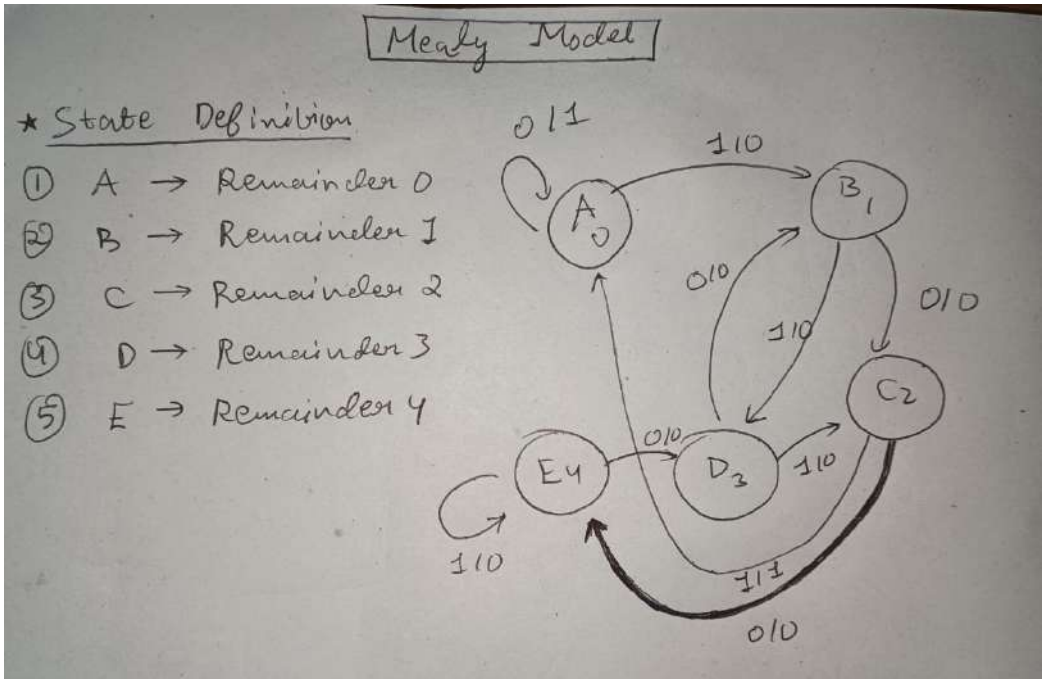


Figure 4: State Transition Diagram Mealy Model

As we can clearly see that 5 states namely A,B,C,D,E are needed. 3 D flip flops are enough to realize these 5 states and hence we designed the logic using three flip flops. But in this experiment instead of using the structural approach of using d flip flops we used the behavioral approach to code the Finite State Machine by using if else statements to determine the next state given the current state and data input as shown in the state table below:

Current State	Input	Next State	Output
A	0	A	1
A	1	B	0
B	0	C	0
B	1	D	0
C	0	E	0
C	1	A	1
D	0	B	0
D	1	C	0
E	0	D	0
E	1	E	0

State Table

Figure 5: State Table Mealy Model

- **Part 1: Moore Model** In this part of the experiment we were required to make an FSM that gives 1 if a serial binary bitstream is divisible by 5.This was done by making a Moore state machine. We first designed the state transition diagram for the overlapping case which is as follows:

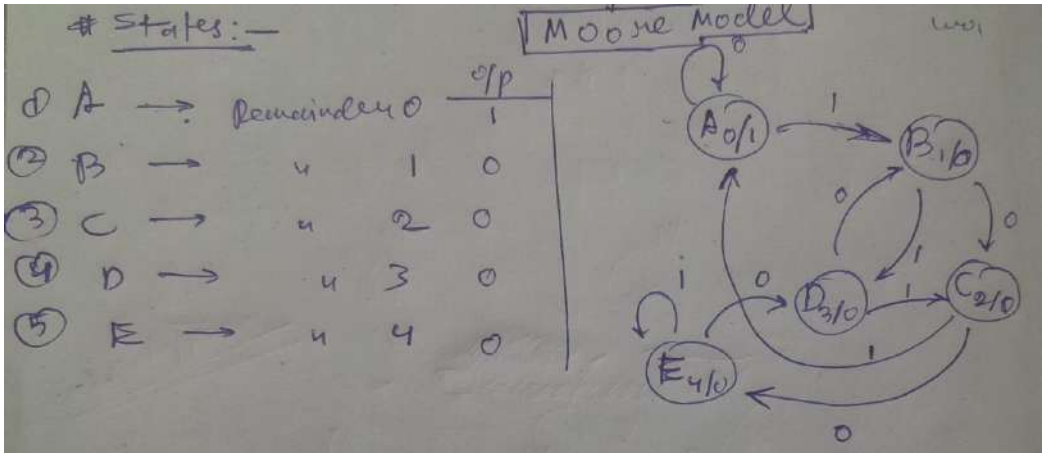


Figure 6: State Transition Diagram Moore Model

As we can clearly see that 5 states namely A,B,C,D,E are needed. 3 D flip flops are enough to realize these 5 states and hence we designed the logic using three flip flops. But in this experiment instead of using the structural approach of using d flip flops we used the behavioral approach to code the Finite State Machine by using if else statements to determine the next state given the current state and data input as shown in the state table below:

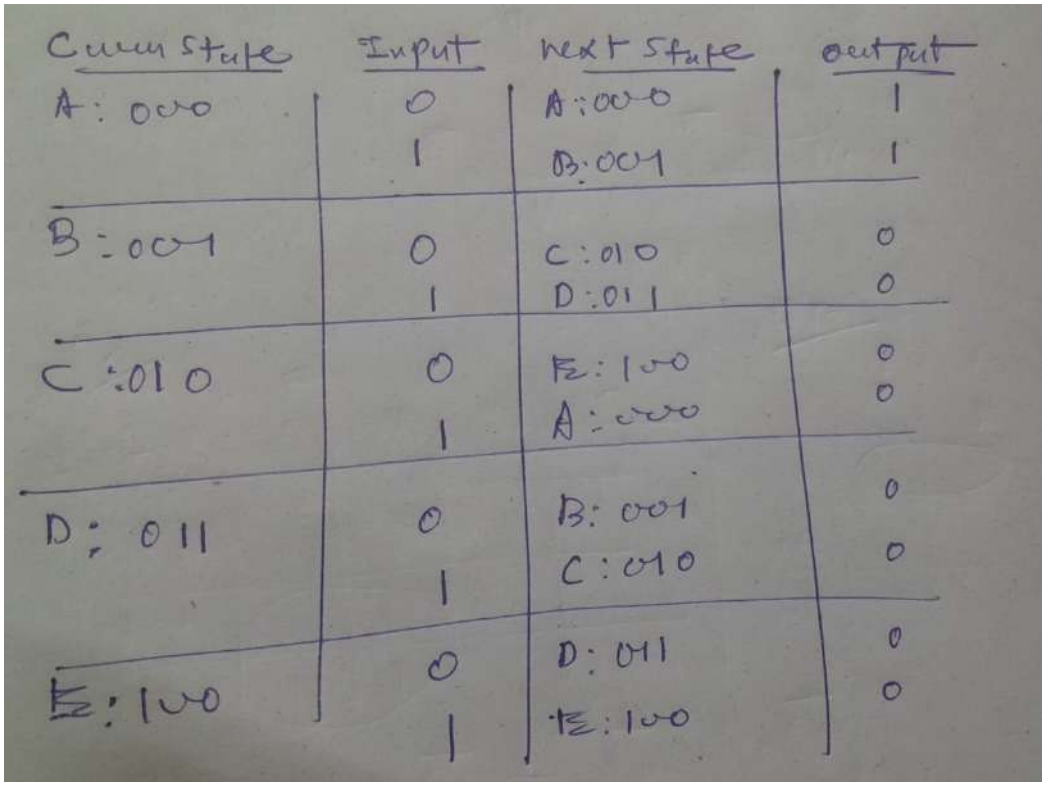


Figure 7: State Table Moore Model

Results/Observations

Part 1: Mealy Model

- We used the following code for making the Mealy Model using the behavioral approach:

```
##################EXP-4#################
module behavioural_fsm(
input wire data,
input wire clk,
output reg[2:0] curr_state,
output reg out
    );
    initial begin
        curr_state=0;
    end

    always @(posedge clk) begin
    if(curr_state==0)begin
        if(data==0) begin
            out=1;
            curr_state=0;
        end
        else begin
            out=0;
            curr_state=1;
        end
    end else if (curr_state==1) begin
        if (data==0) begin
            out=0;
            curr_state=2;
        end
        else begin
            out=0;
            curr_state=3;
        end
    end else if (curr_state==2) begin
        if (data==0) begin
            out=0;
            curr_state=4;
        end
        else begin
            out=1;
            curr_state=0;
        end
    end else if (curr_state==3) begin
        if (data==0) begin
            out=0;
            curr_state=1;
        end
        else begin
            out=0;
            curr_state=2;
        end
     end else if (curr_state==4) begin
        if (data==0) begin
            out=0;
            curr_state=3;
        end
        else begin
            out=0;
            curr_state=4;
        end
    end
    end

endmodule
```

Figure 8: Mealy Model Behavioral Code

- We used the following code for making the Mealy Model using the structural approach:

```
---------------
############structural################
module exp_4_mealy(
input wire data,
input wire clk,

output reg out
    );

    //states=5: corr to n%5: from 0 to 4
    wire a=data;
    wire b,c,d;

    wire d_a=(~a & ~b & c & ~d) | (a & b & ~c & ~d);
    wire d_b=(~b & ~c & d) | (~a & b & ~c & ~d) | (a & ~b & d);
    wire d_c=(~a & ~b & c & d) | (~a & b & ~c & ~d) | (a & ~b & ~c);

    always @(posedge clk) begin
        out=(~a & ~b & ~c & ~d) | (a & ~b & c & ~d);
    end


    dff Fa(.data(d_a),.clk(clk),.rst(0),.out(b));
    dff Fb(.data(d_b),.clk(clk),.rst(0),.out(c));
    dff Fc(.data(d_c),.clk(clk),.rst(0),.out(d));

endmodule

-------------------------

module mealy_tb;
reg clk;
    reg data;
    wire out;
    //wire[2:0] curr_state;
    // Instantiate the Sequence Detector module
    exp_4_mealy uut (
        .clk(clk),
        .data(data),
        .out(out)
    );
    // Generate Clock Signal (50% Duty Cycle)
    always #5 clk = ~clk;  // 10ns period
    initial begin
        // Initialize signals
        $monitor("Time = %0t | Data = %b  | Output = %b", $time, data, out);
        clk = 0;
        data = 0;
        // Apply test sequence: 100110 should produce output = 1
        #5 data = 1;
        #10 data = 0;
        #10 data = 1;
        #10 data = 0;
        #10 data = 0;
        #10 data = 1;
        #10 data = 0;
        #10 data = 1;

        #50 $stop; // Stop simulation
    end

endmodule
```

Figure 9: Mealy Model Structural Code

- We used the following test bench for testing the divisibility by 5 detector

```
 ---------------------
 module fsm_tb;
    reg clk;
     reg data;
     wire out;
     wire[2:0] curr_state;
     // Instantiate the Sequence Detector module
     moore_model uut (
         .clk(clk),
         .data(data),
         .out(out),
         .curr_state(curr_state)
     );
     // Generate Clock Signal (50% Duty Cycle)
     always #5 clk = ~clk;  // 10ns period
     initial begin
         // Initialize signals
         $monitor("Time = %0t | Data = %b | State=%0d | Output = %b", $time, data, curr_state, out);
         clk = 0;
         data = 0;
         // Apply test sequence: 100110 should produce output = 1
         #5 data = 1;
         #10 data = 0;
         #10 data = 1;
         #10 data = 0;
         #10 data = 0;
         #10 data = 1;
         #10 data = 0;
         #10 data = 1;

         #50 $stop; // Stop simulation
     end

 endmodule
```

Figure 10: Test Bench

- The following is the Simulation Result:

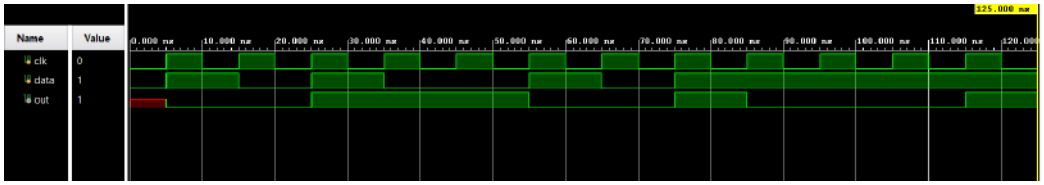


Figure 11: Simulation Result

- The following is the output of the console:

```
# run 1000ns
Time = 0 | Data = 0  | Output = x
Time = 5000 | Data = 1  | Output = 0
Time = 15000 | Data = 0  | Output = 0
Time = 25000 | Data = 1  | Output = 1
Time = 35000 | Data = 0  | Output = 1
Time = 55000 | Data = 1  | Output = 0
Time = 65000 | Data = 0  | Output = 0
Time = 75000 | Data = 1  | Output = 1
Time = 85000 | Data = 1  | Output = 0
Time = 115000 | Data = 1  | Output = 1
```

Figure 12: Console Output

- The following is the schematic of the Model:

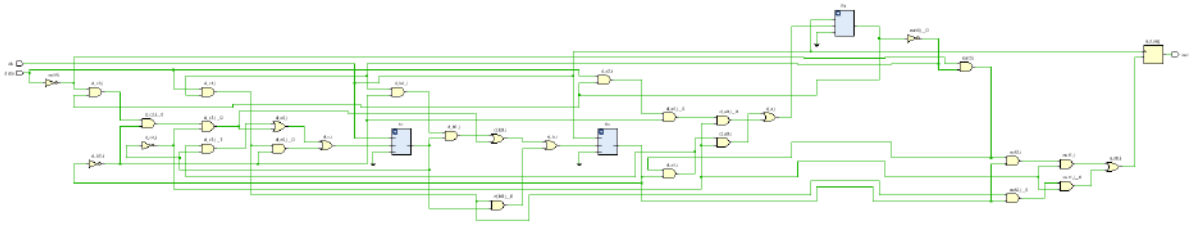


Figure 13: Schematic

Part 1: Moore Model

- We used the following code for making the Moore Model using the behavioral approach:

```
module moore_model(
input wire data,
input wire clk,
output reg[2:0] curr_state,
output reg out
    );
    initial begin
        curr_state=0;
    end

    always @(posedge clk) begin
    if(curr_state==0)begin
        out=1;//only
        if(data==0) begin
            curr_state=0;
        end
        else begin

            curr_state=1;
        end
    end else if (curr_state==1) begin
        out=0;
        if (data==0) begin
            curr_state=2;
        end
        else begin
            curr_state=3;
        end
    end else if (curr_state==2) begin
        out=0;
        if (data==0) begin
            curr_state=4;
        end
        else begin
            curr_state=0;
        end
    end else if (curr_state==3) begin
        out=0;
        if (data==0) begin
            curr_state=1;
        end
        else begin
            curr_state=2;
        end
     end else if (curr_state==4) begin
        out=0;
        if (data==0) begin
            curr_state=3;
        end
        else begin
            curr_state=4;
        end
    end
    end
endmodule
```

Figure 14: Code

- We used the following test bench for testing the divisibility by 5 detector

```
--------------------
module fsm_tb;
  reg clk;
   reg data;
   wire out;
   wire[2:0] curr_state;
   // Instantiate the Sequence Detector module
   moore_model uut (
       .clk(clk),
       .data(data),
       .out(out),
       .curr_state(curr_state)
   );
   // Generate Clock Signal (50% Duty Cycle)
   always #5 clk = ~clk;  // 10ns period
   initial begin
       // Initialize signals
       $monitor("Time = %0t | Data = %b | State=%0d | Output = %b", $time, data, curr_state, out);
       clk = 0;
       data = 0;
       // Apply test sequence: 100110 should produce output = 1
       #5 data = 1;
       #10 data = 0;
       #10 data = 1;
       #10 data = 0;
       #10 data = 0;
       #10 data = 1;
       #10 data = 0;
       #10 data = 1;

       #50 $stop; // Stop simulation
   end

endmodule
```

Figure 15: Test Bench

- The following is the Simulation Result:

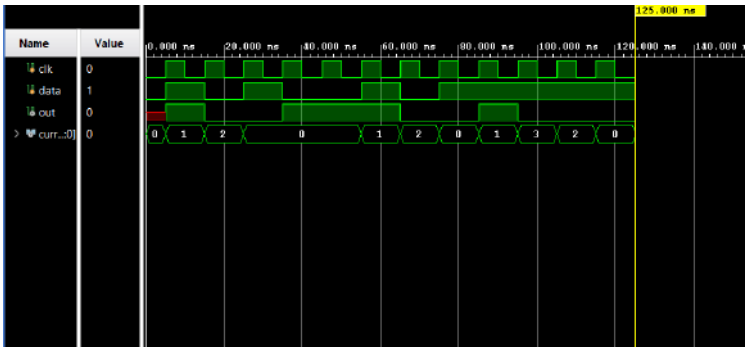


Figure 16: Simulation Result

- The following is the output of the console:

```
# run 1000ns
Time = 0 | Data = 0 | State=0 | Output = x
Time = 5000 | Data = 1 | State=1 | Output = 1
Time = 15000 | Data = 0 | State=2 | Output = 0
Time = 25000 | Data = 1 | State=0 | Output = 0
Time = 35000 | Data = 0 | State=0 | Output = 1
Time = 55000 | Data = 1 | State=1 | Output = 1
Time = 65000 | Data = 0 | State=2 | Output = 0
Time = 75000 | Data = 1 | State=0 | Output = 0
Time = 85000 | Data = 1 | State=1 | Output = 1
Time = 95000 | Data = 1 | State=3 | Output = 0
Time = 105000 | Data = 1 | State=2 | Output = 0
Time = 115000 | Data = 1 | State=0 | Output = 0
```

Figure 17: Console Output

- The following is the schematic of the Model:

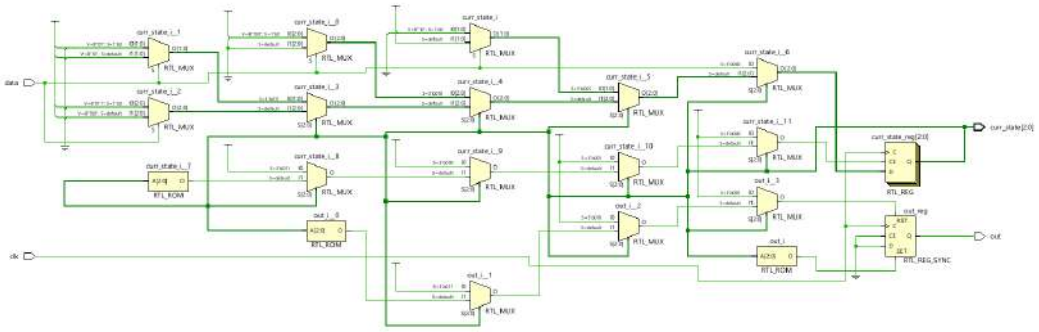


Figure 18: Schematic

Discussion Atif Faizan 22EC10096

- In this experiment we developed a divisibility by 5 detector for a continuous bit stream using Finite State Machine using behavioral approach.
- The logic was based on the fact that if the input data is 0 then the new remainder of the number will be 2*previous remainder. While if the input data is 1 then the new remainder of the number will be 2*previous remainder + 1. In this was we were able to determine the change in state when input data is 1 or 0.
- If the value of the remainder was greater than equal to 5 we took modulo again with 5 to find the actual remainder.
- The main difference between the structural and behavioral approach in this experiment was that in structural approach we explicitly designed the logic using 3 d flip flops and by minimizing the K map and determining the complete logic like we did in the previous experiment. In the behavioral approach we coded the features of the finite state machine by determining the next state depending on the data input and the current state.
- In the mealy model there was no delay between the occurrence of a number divisible by 5 and the output begin 1. While in Moore Model there was one cycle delay as is expected from theory.
- The advantage of Moore model over Mealy model is that there is no error due to propagation delay of the gates because in Mealy Model the output is a function of both input and the outputs from the flip flops. Since the output from the flip flops may be delayed due to gate propagation delay while the input data is available instantaneously to the output logic, it give rise to errors. While in the Moore Model the output is not a function of input, hence no such error.
- The number of states needed in this experiment for both Mealy and Moore Model was 5.
- We also completed the bonus part of the experiment of designing an FSM which outputs 1 for an aggregate serial binary string that detects if the last 3 bits are alternating (101 or 010). Here is the code:

```
########## EXP-4 EXTRA PART ##############
module fsm_mealy(
input wire data,
input wire clk,
output reg[2:0] curr_state,
output reg out
    );

    initial begin
        curr_state=0;
    end

    always @(posedge clk) begin
        if(curr_state==0) begin
            if(data==0) begin
                out = 0;
                curr_state = 0;
             end
             else begin
                out = 0;
                curr_state = 1;
             end
        end else if(curr_state==1)begin
            if(data==0) begin
                out = 1;
                curr_state = 2;
            end
            else begin
                out = 0;
                curr_state = 3;
            end
        end else if(curr_state==2) begin
            if(data==0) begin
                out = 0;
                curr_state = 4;
            end
            else begin
                out = 1;
                curr_state = 5;
            end
        end else if(curr_state==3) begin
            if(data==0) begin
                out = 0;
                curr_state = 6;
            end
            else begin
                out = 0;
                curr_state = 7;
            end
        end else if(curr_state==4) begin
            if(data==0) begin
                out = 0;
                curr_state = 0;
            end
            else begin
                out = 0;
                curr_state = 1;
            end
        end else if(curr_state==5) begin
            if(data==0) begin
                out= 1;
                curr_state = 2;
            end
            else begin
                out = 0;
                curr_state = 3;
            end
        end else if(curr_state==6) begin
            if(data==0) begin
                out = 0;
                curr_state = 4;
            end
            else begin
                out = 1;
                curr_state = 5;
            end
        end else if(curr_state==7) begin
            if(data==0) begin
                out = 0;
                curr_state = 6;
            end
            else begin
                out = 0;
                curr_state = 7;
            end
        end
    end

endmodule
```

Figure 19: Extra Part Code

Discussion Chiradip Biswas 22EC30016

- In this experiment we made a FSM to check the divisibility of any serial binary stream by 5. For this we made both structural and behavioural realization of the mealy model and the behavioural implementation of the moore model.
- The main idea behind the state diagram was the fact that after the new bit comes, if it is 0, $rem_{n+1} = 2 * rem_n$ and if it is 1, $rem_{n+1} = 2 * rem_n + 1$. If the result exceeds 5, we take modulo again. Thus the states are made as modulo we obtained after dividing the current bitstream. The state transitions were drawn keeping this rule in mind.
- We used 3 d flip flops as there were 5 states in both Meanly and Moore realizations. $ceil(log_2(5)) = 3$. The number of states are the same as the output is 1 when the modulo is zero. So in Moore the output=1 does not require an additional state.
- The structural realization was done by specifying the connection of each flip-flop input and output after minimizing k-maps. For the behavioural case we just mentioned the next state when the current state is a given one and for a particular input data. The schematic realization is done in the behavioural case by the compiler using default elements, whereas in structural realization, we explicitly specified them.
- For the extra part of the experiment, we made a FSM to detect whether the last 3 bits of a sequence are alternating or not. This was just an FSM to detect sequence (010 or 101 overlapping) at the last of bit-stream.
- The Moore model made FSM is safer compared to the Mealy model one, because in Moore the output only depends upon the current state,whereas for the mealy model, the output directly depends upon the current input as well. So in case of mealy, there can be multiple parallel layers of logic gates, so it may happen that the gate output did not stabilize within a given clock period, as the input can change in each period. This can give rise to erronous output due to the timing isuue. But in moore model, as the output is obtained after reaching the state, there is no possibility of error. So moore model is safer compared to mealy model.

Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 6

NAND, NOR Gates JK Latch,D flip-Flop Design

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016

- **Group no.:** 35 (Monday)
- **Date of Submission:** 23-03-2025

Aim

1. Design and simulate smallest possible 2-input and 3-input NAND gates in 180 nm CMOS technology with their worst case input-to-output transfer characteristic curve goes through (Vdd/2, Vdd/2) point (i.e., Inverting point is Vdd/2). Simulate and observe their worst case rise and fall time with fan-out load of 3 (i.e. driving three gates identical to itself) and driven by a gate identical to itself.

2. Design and simulate smallest possible 2-input and 3-input NOR gates in 180 nm CMOS technology with their worst case input-to-output transfer characteristic curve goes through (Vdd/2, Vdd/2) point (i.e., Inverting point is Vdd/2). Simulate and observe their worst case rise and fall time with fan-out load of 3 (i.e. driving three gates identical to itself) and driven by a gate identical to itself.

3. Design and simulate JK-Latch using NAND gates (designed in part (a)) and NOR gates (designed in part (b)). Observe Data-to-Out and Clock-to-Out delays.

4. Design and simulate a D-Flip-Flop using NAND gates based JK-Latch (designed in part (c)). Observe Setup time and hold time of the Flip-Flop.

Theory

- CMOS NAND and NOR gates are fundamental logic gates built using complementary MOSFETs (PMOS and NMOS) to achieve low power consumption and high noise margins.
- In a CMOS NAND gate, PMOS transistors are connected in parallel, and NMOS transistors are in series, ensuring that the output is low only when all inputs are high.
- Conversely, a CMOS NOR gate has PMOS transistors in series and NMOS transistors in parallel, making the output high only when all inputs are low.
- Both NAND and NOR gates are universal gates, meaning they can be combined to implement any Boolean function, including AND, OR, and NOT operations.
- Due to their energy efficiency and scalability, CMOS NAND and NOR gates are widely used in digital circuits, including microprocessors, memory units, and FPGAs.
- For NAND gate: $Y = \overline{A \cdot B}$ and for NOR gate: $Y = \overline{A + B}$. For NAND "0" is the dominant input and for NOR is "1".

Waveform of a CMOS logic gate with finite propagation delay, rise and fall time:

- **Rise Time or tR**: the time for a signal to go from 10% to 90% of its final value.
- **Peak Time or tP**: Time taken for a response to reach maximum value.
- **Fall Time or tF**: the time taken for 90%-10% drop in the signal depending on its value.

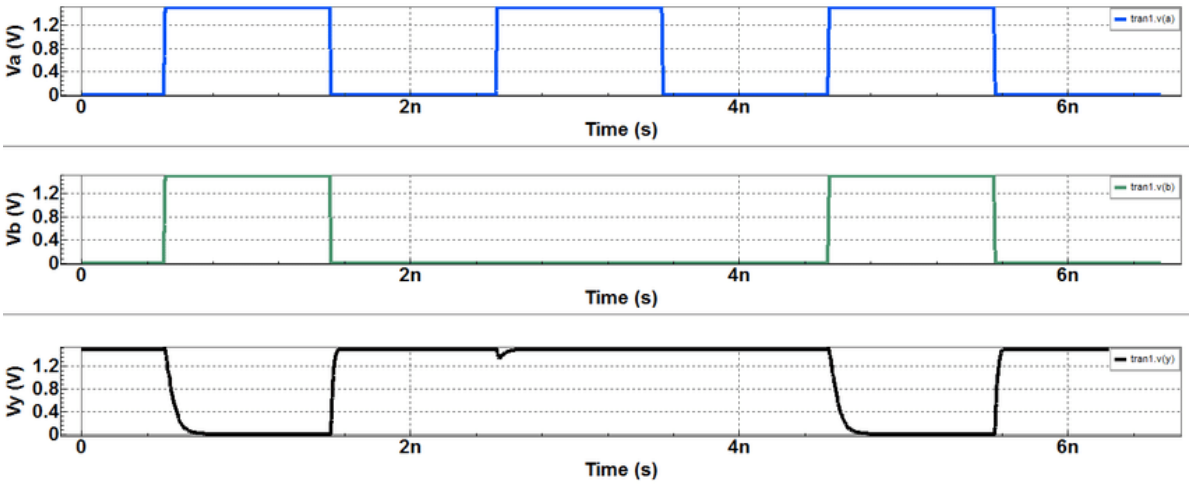


Figure 1: Dynamic Characteristics of CMOS NAND gate

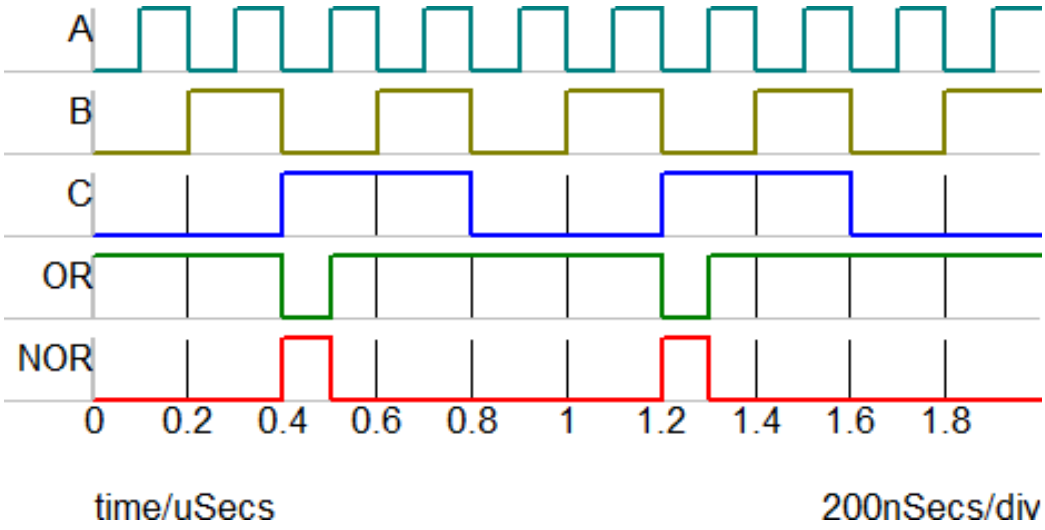
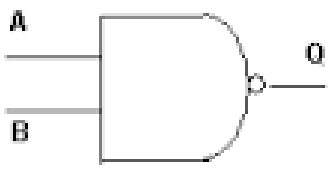
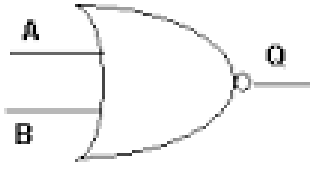


Figure 2: Waveform of CMOS NOR gate



NAND

A	B	Q
0	0	1
0	1	1
1	0	1
1	1	0

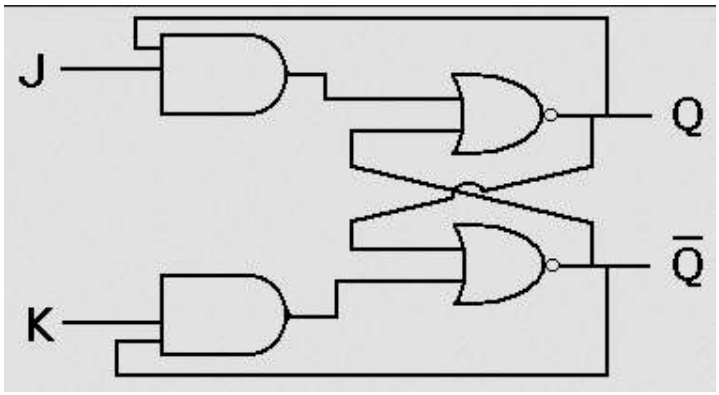


NOR

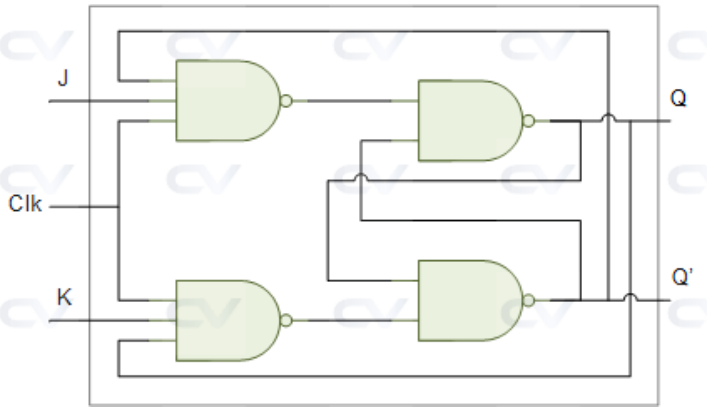
A	B	Q
0	0	1
0	1	0
1	0	0
1	1	0

Figure 3: NAND and NOR gates Truth Table

- The Propagation delay, rise and fall time depends upon factors like:
  - MOS widths, channel-length (current drawn (by n-MOS) and current supplied (by p-MOS) increases with Width and decreases with channel length increase).
  - Also the capacitance of MOS increase with their Widths, so beyond a certain width, the charge and discharge time starts to increase.
  - The load capacitance at output of gates also increases propagation delay, rise and fall time, as they take longer time to charge or discharge to a certain voltage. So with increased fanout the rise, fall time and delay increase.
- **JK-Latch:** A JK latch is a bistable circuit with two inputs J and K, which control its state based on clock or enable signals. Unlike an SR latch, the JK latch eliminates the invalid state by toggling the output when both J and K are high. This toggling happens due to:
  - Delayed effect of Q,$\overline{Q}$ to input.
  - The state of "10" or "01" at output combined with J,K both "1", toggles output continuously.



(a) JK latch using NAND NOR both



(b) JK latch using NAND gates only

Clock	J	K	Qn+1	State
0	X	X	Qn	
1	0	0	Qn	Hold
1	0	1	0	Reset
1	1	1	1	Set
1	1	1	Qn	Toggle

Figure 5: JK latch truth table

- **D-flip flop:** Flip flops are edge triggered analogue of latches. D-flip flop loads the input value at D to output (Q) directly when active edge of clock is detected (either rising or falling edge).
- Flip-flop can be designed by using master-slave topology of latches, where 2 latches of opposite level triggered clock signals are cascaded and given clock signal(complement of each other), the output of master connected to input of slave latch.
- The delays in flip-flop are:
  - **Data-output delay:** The time taken for the output to change after the clock or input changes.

- **Setup Time:** The minimum time before the clock edge that the input must remain stable.
- **Hold Time:** The minimum time after the clock edge that the input must remain stable.
- **Clock-to-Q Delay:** The time between the active clock edge and the change in the output Q.

Part 1: NAND Gate Design

Design

Schematic for 2 input NAND

We chose the following design parameter:

- $(W/L)_p/(W/L)_n = 1.6u/0.84u$
- $V_{dd} = 1.8V$
- $V_{pulse} = 1.8V$ with $TimePeriod = 100ns$ $PulseWidth = 50ns$ $RiseTime = 10ns$ $FallTime = 10ns$ $DutyCycle = 50\%$
- Load Capacitance of $C_L = 10fF$

The following is the schematic of the 2 input CMOS NAND gate:

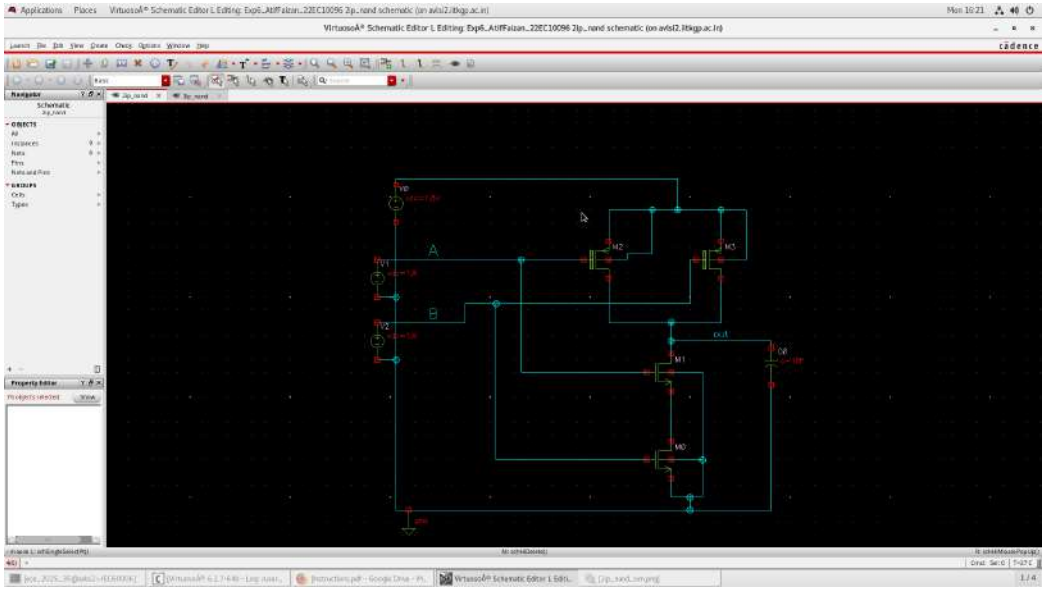


Figure 6: CMOS 2 input nand Schematic

The following is the schematic of the 2 input CMOS NAND gate driving a load of fanout 3:

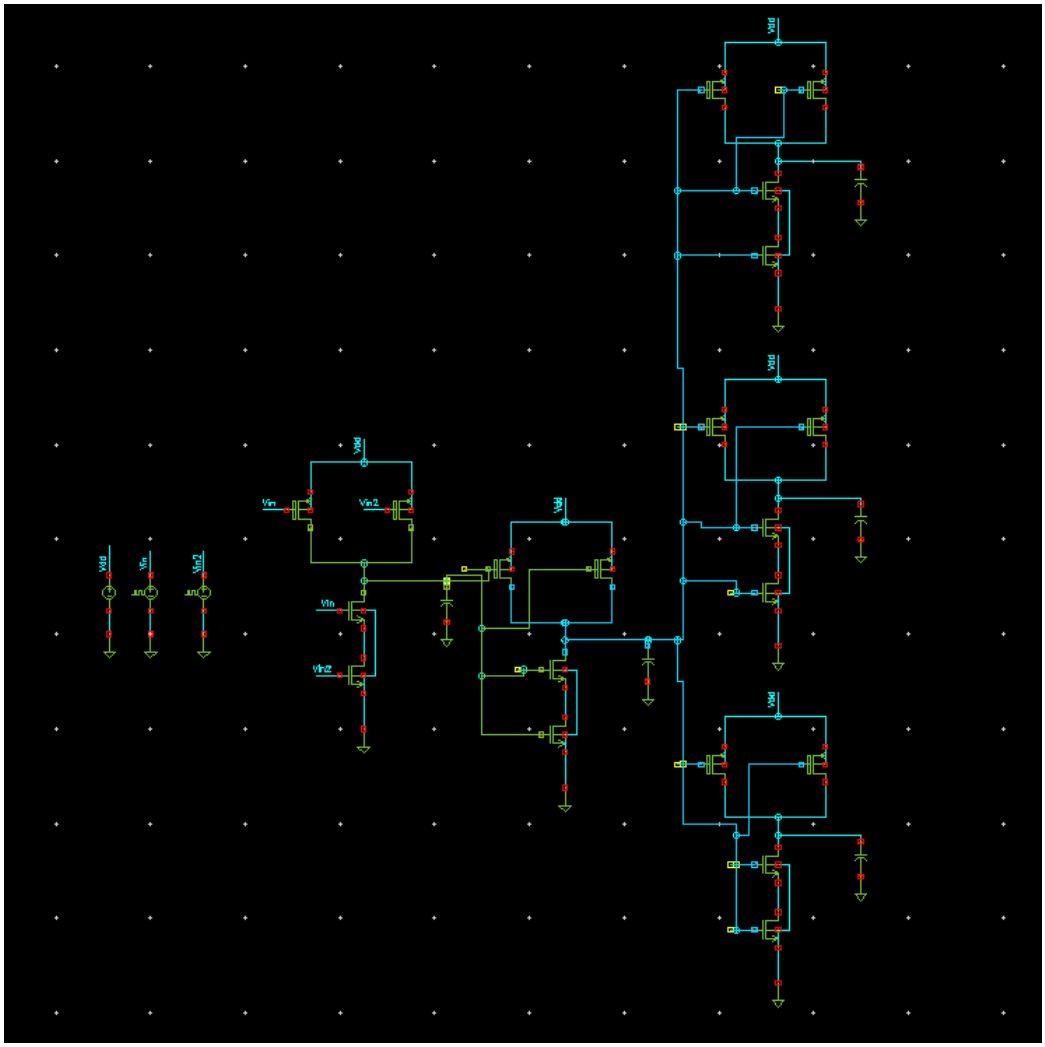


Figure 7: 2 input NAND gate driving a load of fanout 3

Simulation Results for 2 input NAND gate

The following is the simulation result of the testbench:

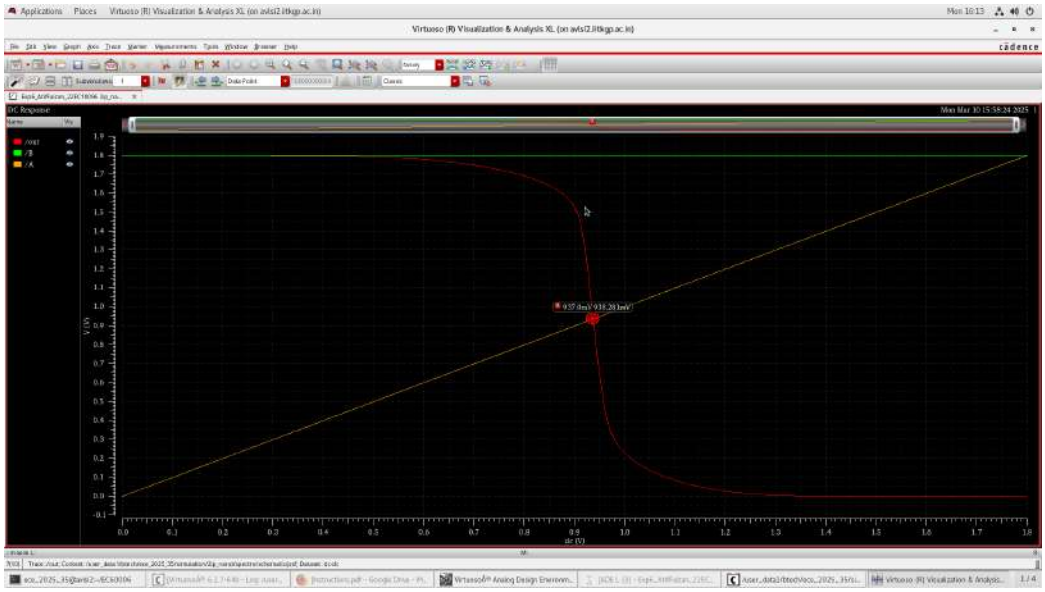


Figure 8: DC sweep of 1 input, keeping the other at 1.8v

The following is the Rise Time Observed:

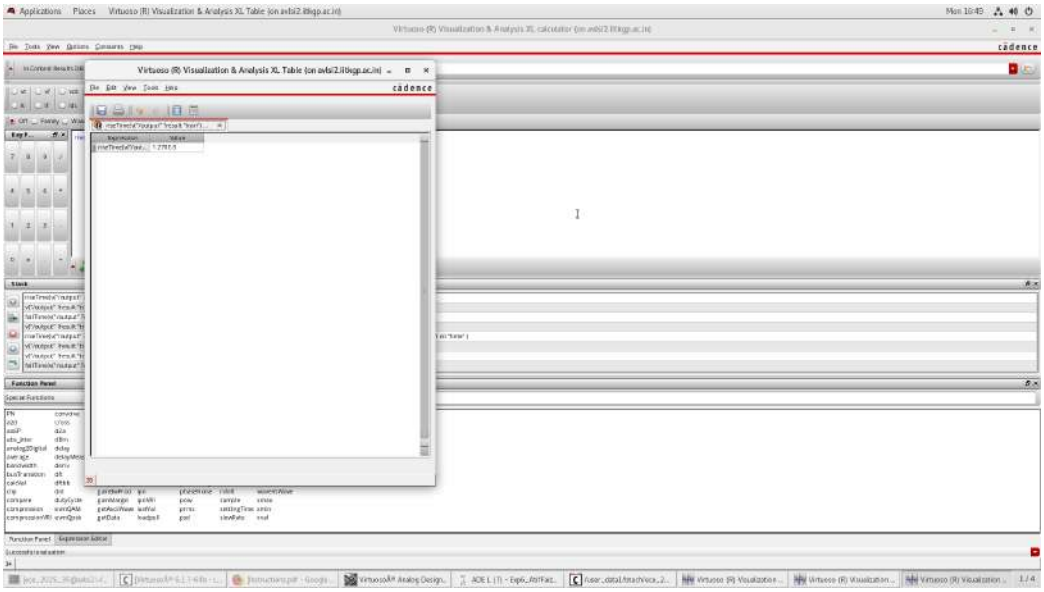


Figure 9: Rise Time

The following is the Fall Time Observed:

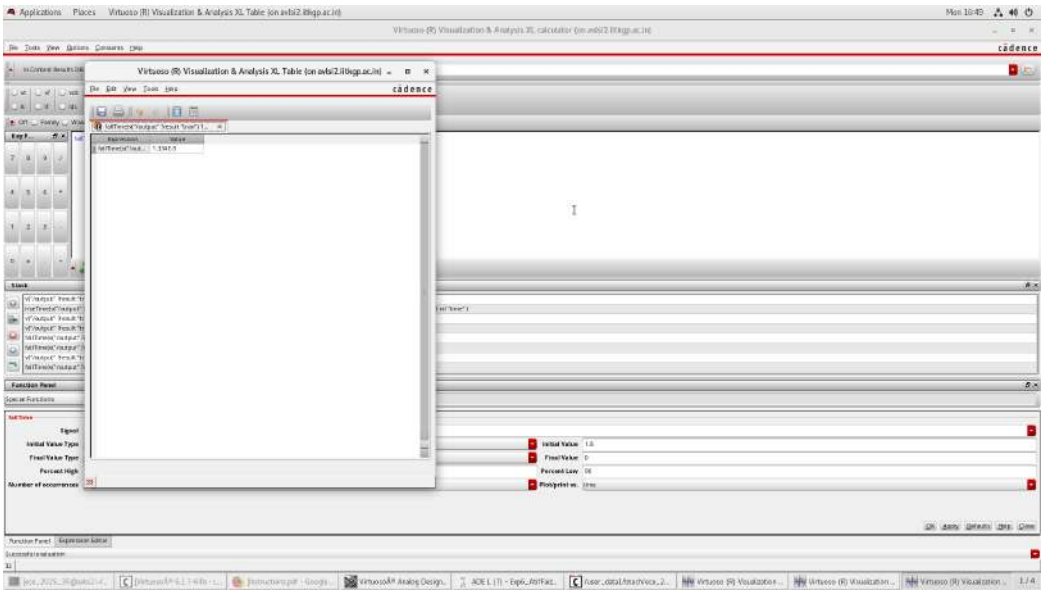


Figure 10: Fall Time

The following is the Propagation Delay (midpoint to midpoint) observed: 671.8ps

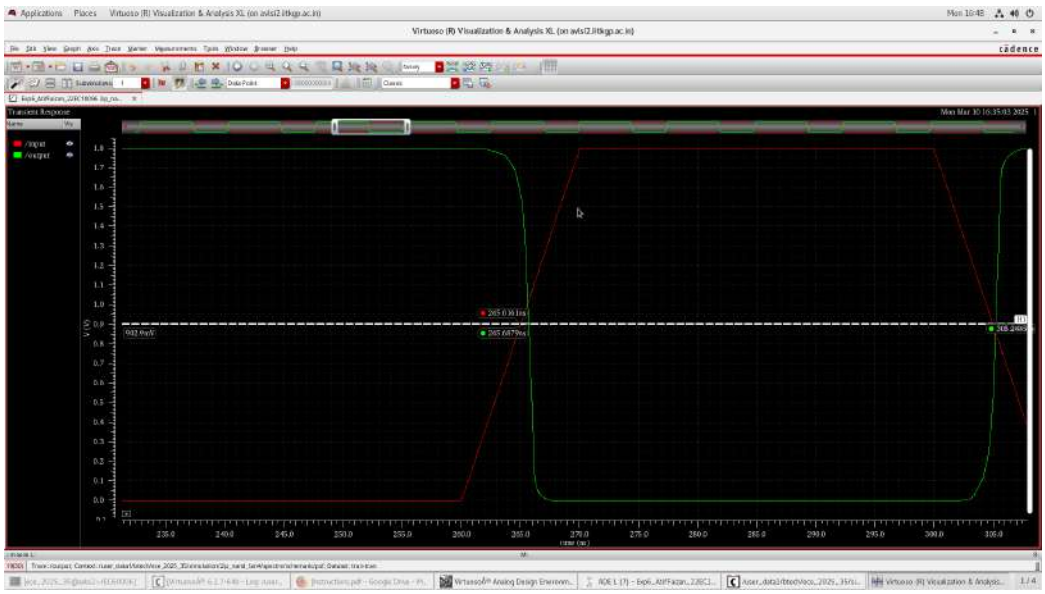


Figure 11: Propagation delay measurement of the fanout circuit

Observations

We observed the following things:

- Rise Time = 1.27 ns [fanout-3]
- Fall Time = 1.334 ns [fanout-3]
- Logic Threshold Voltage = 938 mV

We can clearly see that the rise time and fall time are approximately same and the logic threshold voltage is approximately $V_{dd}/2$

Design for 3 input NAND gate

We chose the following design parameter:

- $(W/L)_p/(W/L)_n = 1.6u/1.26u$
- $V_{dd} = 1.8V$
- $V_{pulse} = 1.8V$ with $TimePeriod = 100ns$ $PulseWidth = 50ns$ $RiseTime = 10ns$ $FallTime = 10ns$ $DutyCycle = 50\%$
- Load Capacitance of $C_L = 10fF$

The following is the schematic of the 3 input CMOS NAND gate:

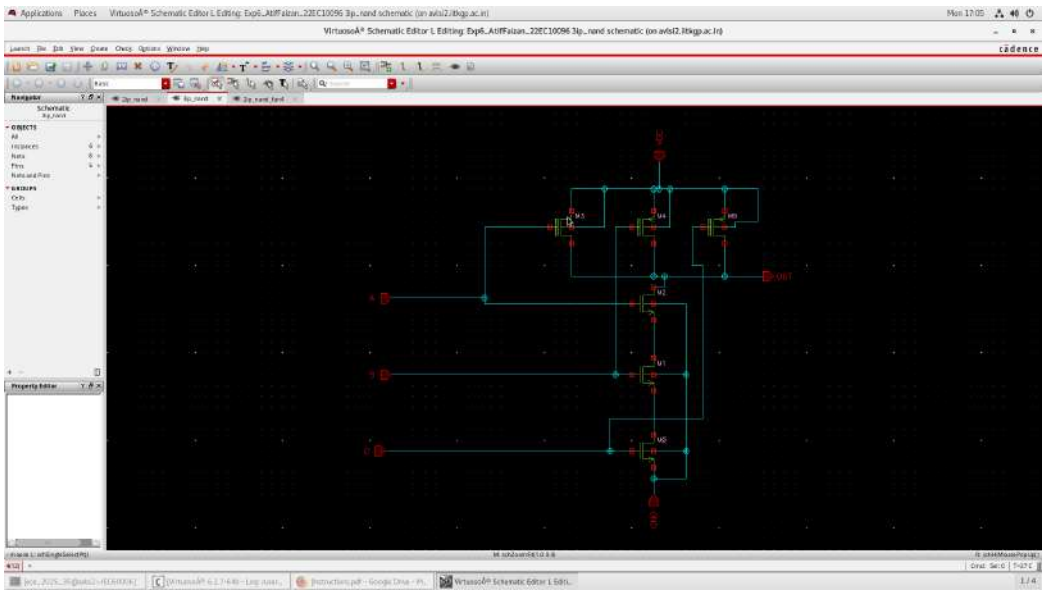


Figure 12: CMOS 3 input nand Schematic

The following is the schematic of the 3 input CMOS NAND gate driving a load of fanout 3:

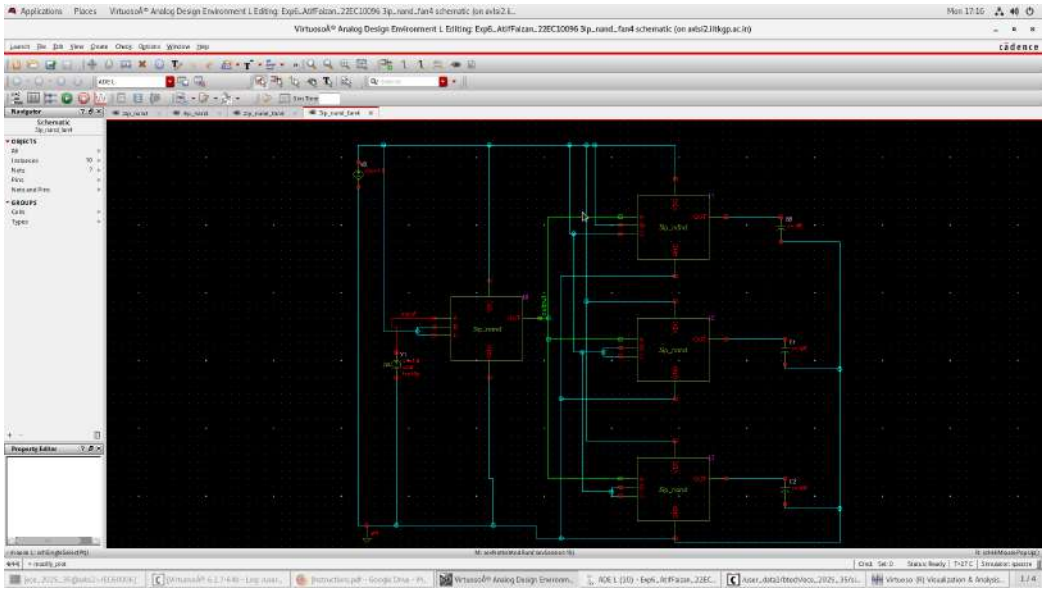


Figure 13: 3 input NAND gate driving a load of fanout 3

Simulation Results for 3 input NAND gate

The following is the simulation result of the testbench:

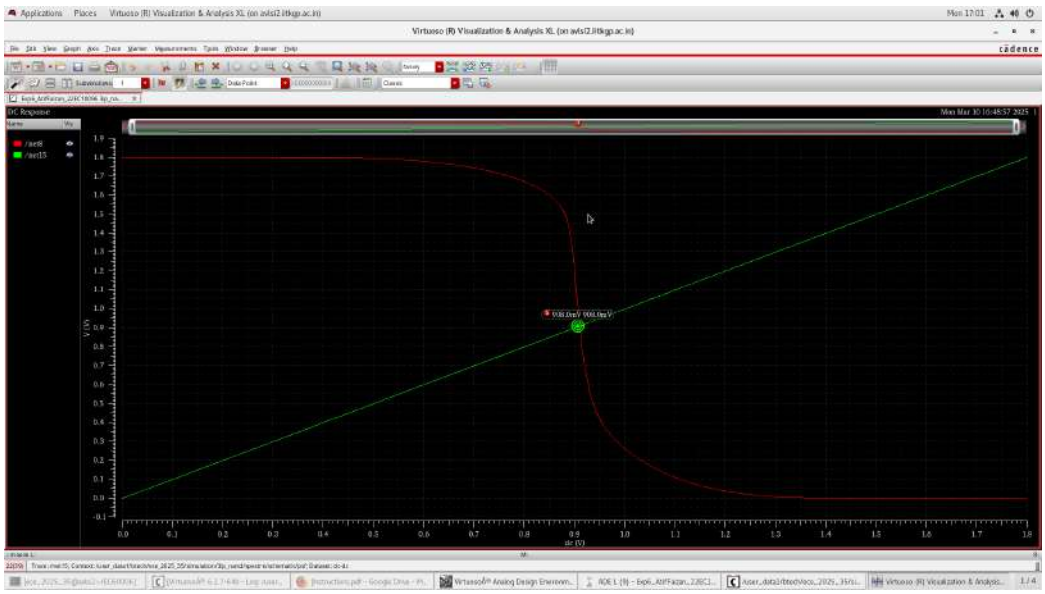


Figure 14: DC sweep of 1 input, keeping the other 2 at 1.8v

The following is the Rise Time Observed:

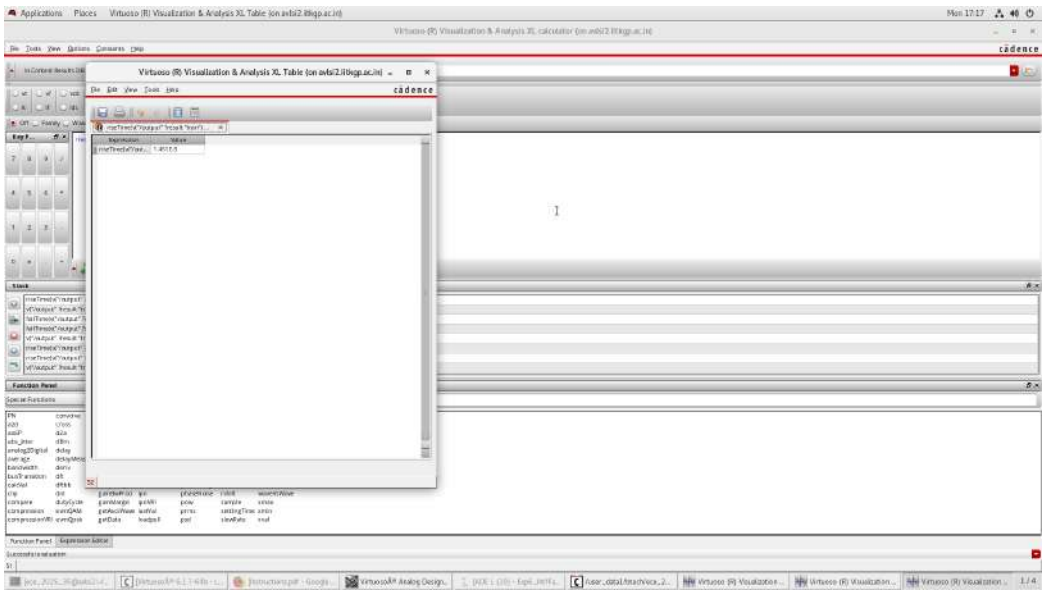


Figure 15: Rise Time with fanout-3 circuit

The following is the Fall Time Observed:

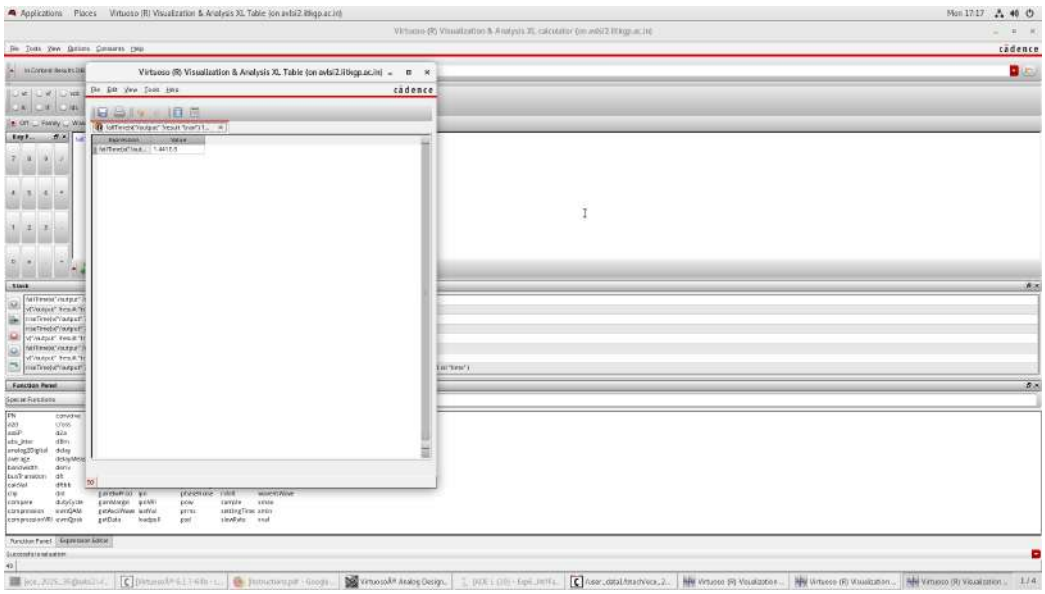


Figure 16: Fall Time with fanout-3 circuit

The following is the Propagation Delay (midpoint to midpoint) observed: 414.5ps

Figure 17: Propagation delay measurement of the fanout circuit

Observations

We observed the following things:

- Rise Time = 1.451 ns [fanout-3]
- Fall Time = 1.441 ns [fanout-3]
- Logic Threshold Voltage = 908 mV

We can clearly see that the rise time and fall time are approximately same and the logic threshold voltage is approximately $V_{dd}/2$

Part 2: Designing CMOS NOR Gate

Schematic of 2 Input NOR gate:

Design

We chose the following design parameter:

- $(W/L)_p/(W/L)_n = 3.2u/0.42u$
- $V_{dd} = 1.8V$
- $V_{pulse} = 1.8V$ with $TimePeriod = 100ns$ $PulseWidth = 50ns$ $RiseTime = 10ns$ $FallTime = 10ns$ $DutyCycle = 50\%$
- Load Capacitance of $C_L = 10fF$

The following is the schematic of the 2 input CMOS NOR gate:

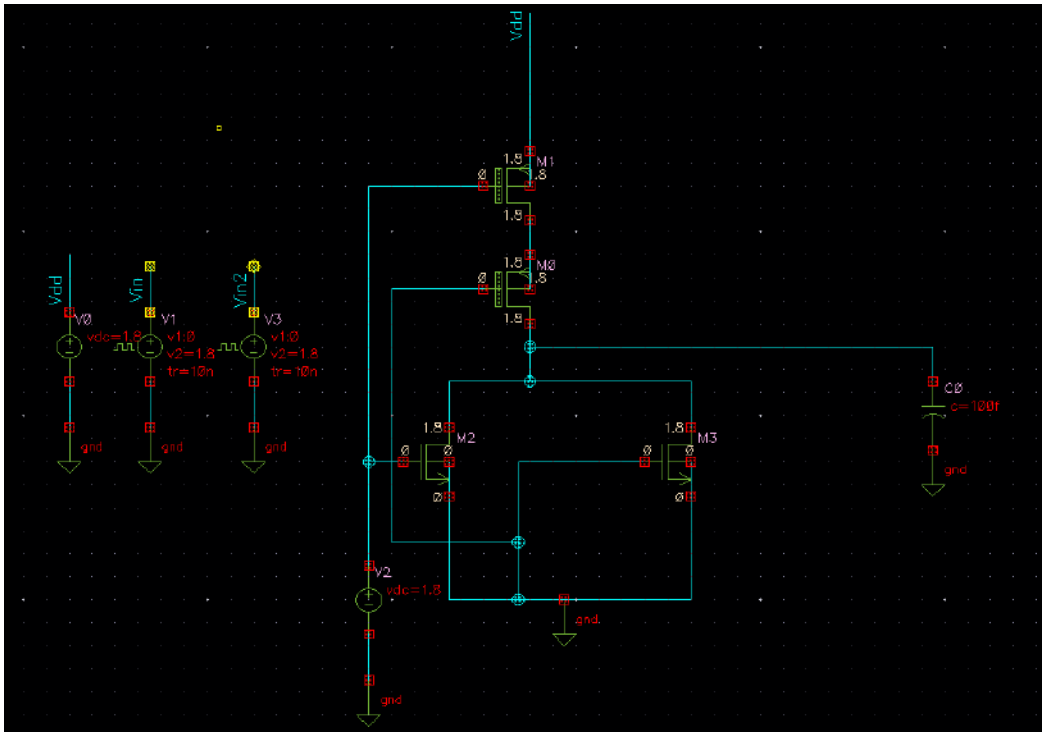


Figure 18: CMOS 2 input NOR Schematic

The following is the schematic of the 2 input CMOS NOR gate driving a load of fanout 3:

Figure 19: 2 input NOR gate driving a load of fanout 3

Simulation Results for 2 input NOR gate

The following is the simulation result of the testbench:

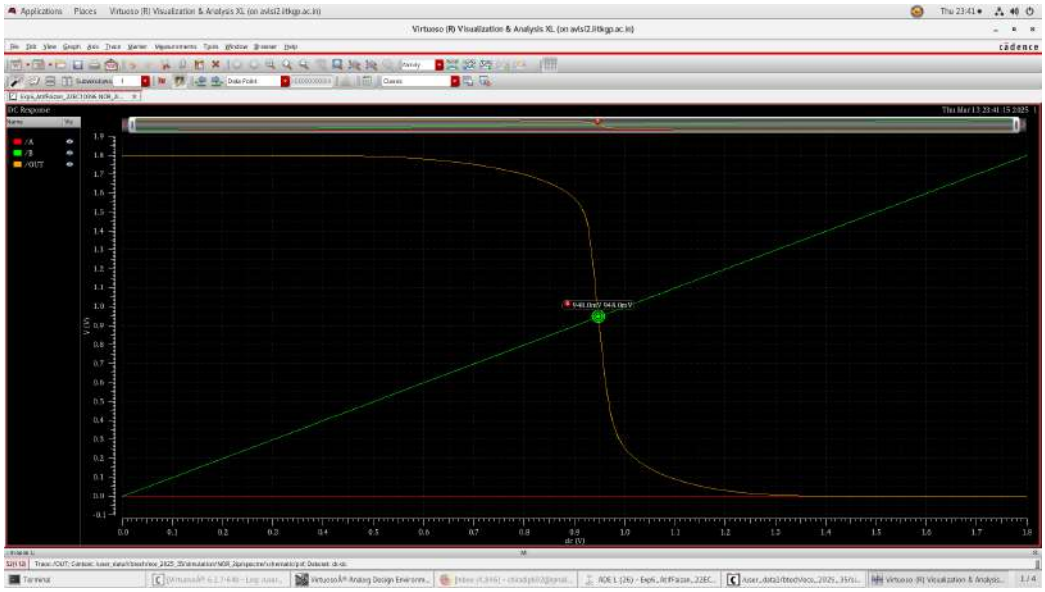


Figure 20: DC sweep of 1 input, keeping the other at 0v

The following is the Rise Time Observed:

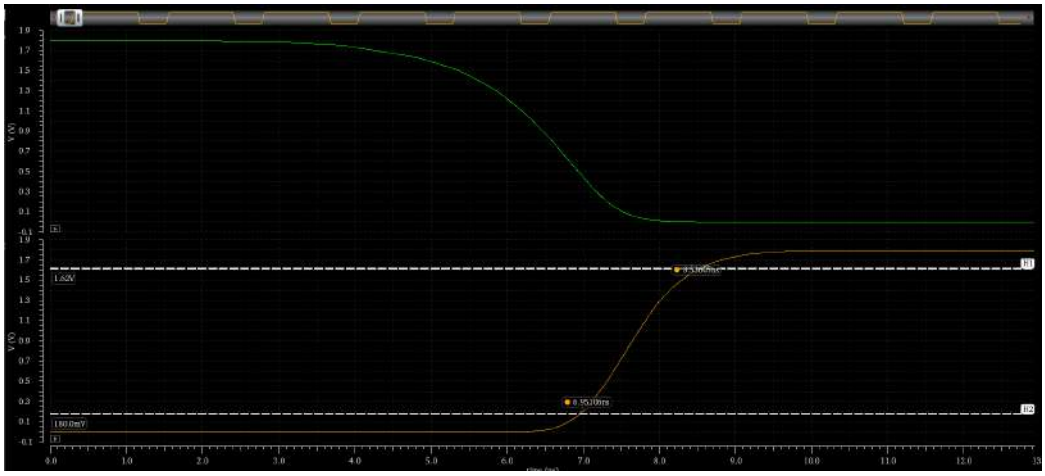


Figure 21: Rise Time

The following is the Fall Time Observed:

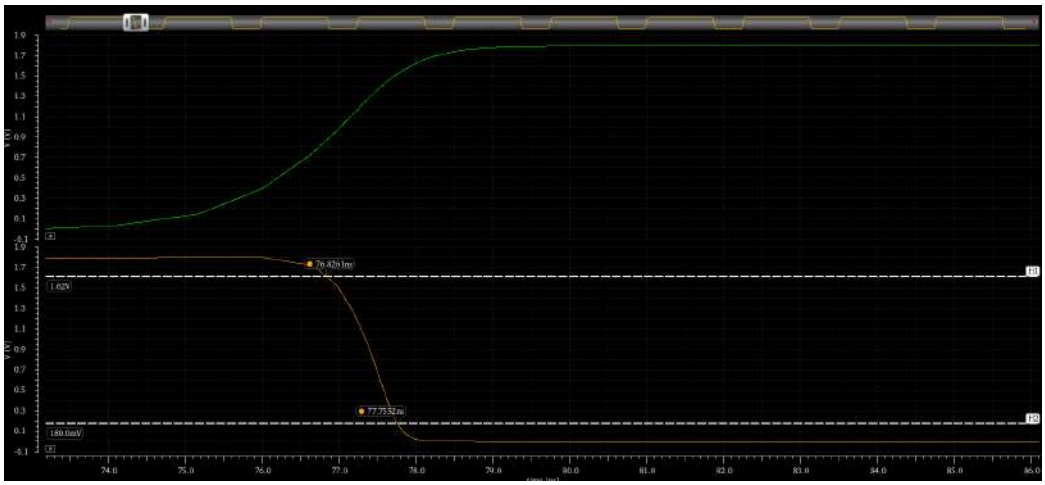


Figure 22: Fall Time

Observations

We observed the following things:

- Rise Time = 1.585 ns [fanout-3]
- Fall Time = 0.929 ns [fanout-3]

We can clearly see that the rise time and fall time are approximately same.

Design for 3 input NOR gate

We chose the following design parameter:

- $(W/L)_p/(W/L)_n = 4.8u/0.42u$
- $V_{dd} = 1.8V$
- $V_{pulse} = 1.8V$ with $TimePeriod = 100ns$ $PulseWidth = 50ns$ $RiseTime = 10ns$ $FallTime = 10ns$ $DutyCycle = 50\%$
- Load Capacitance of $C_L = 10fF$

The following is the schematic of the 3 input CMOS NOR gate:

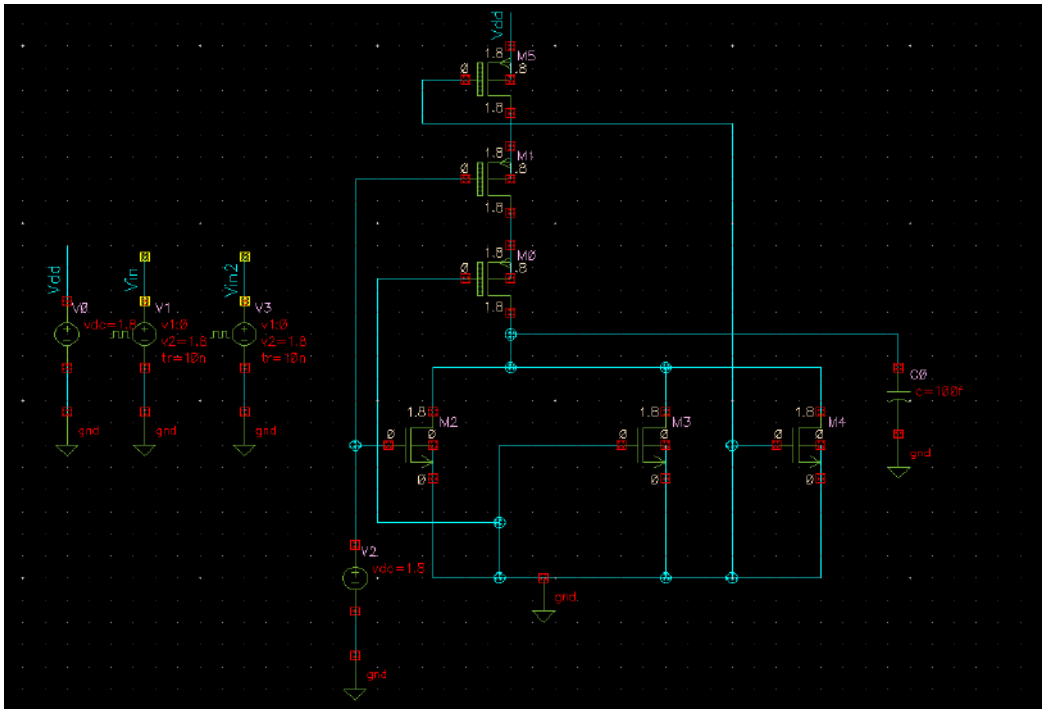


Figure 23: CMOS 3 input NOR Schematic

The following is the schematic of the 3 input CMOS NOR gate driving a load of fanout 3:

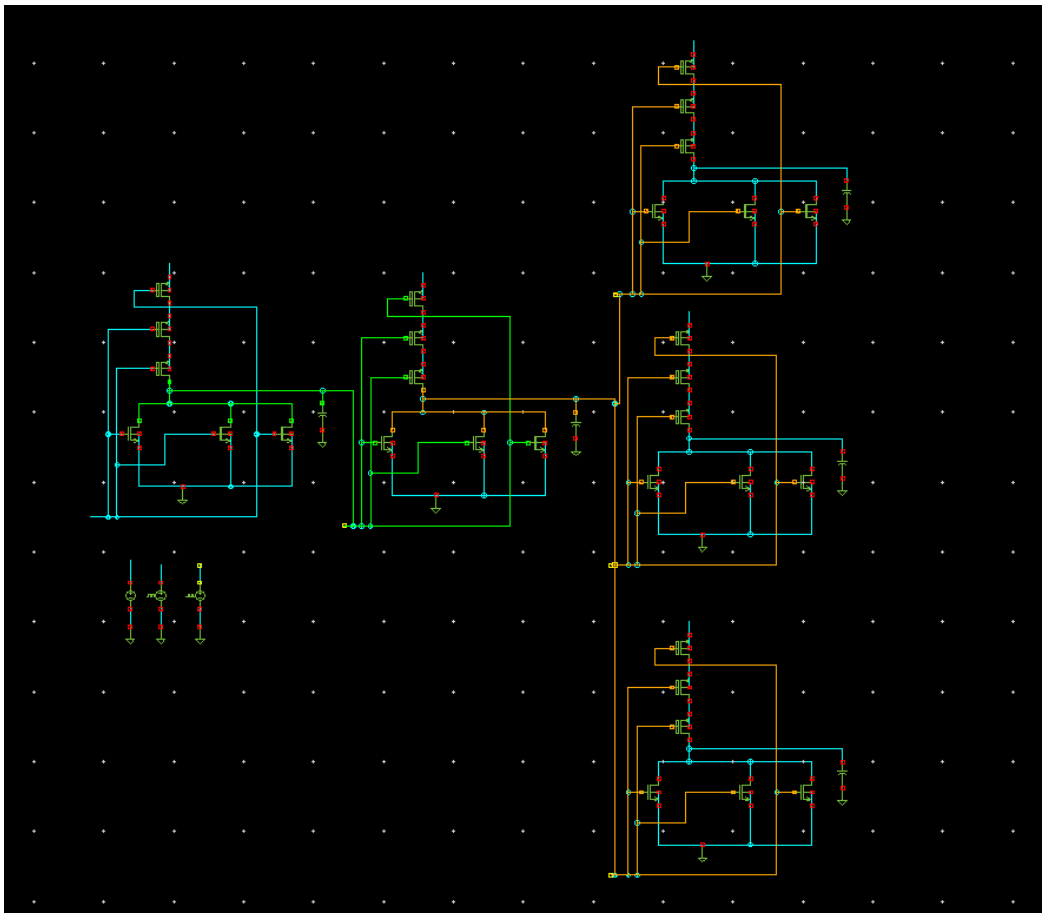


Figure 24: 3 input NOR gate driving a load of fanout 3

Simulation Results for 3 input NOR gate

The following is the simulation result of the testbench:

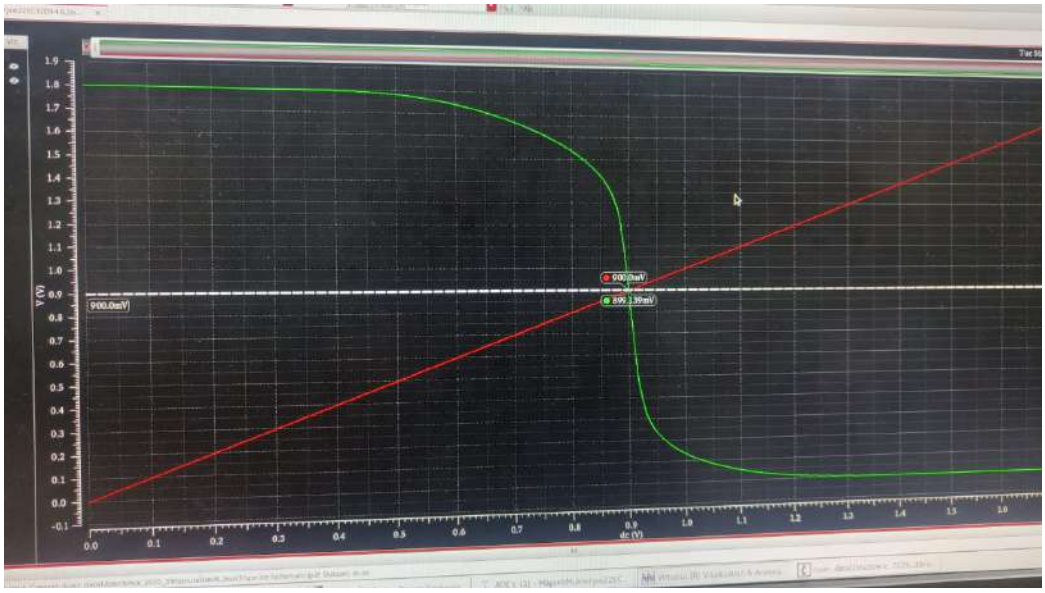


Figure 25: DC sweep of 1 input, keeping the other 2 at 0v

The following is the Rise Time Observed:

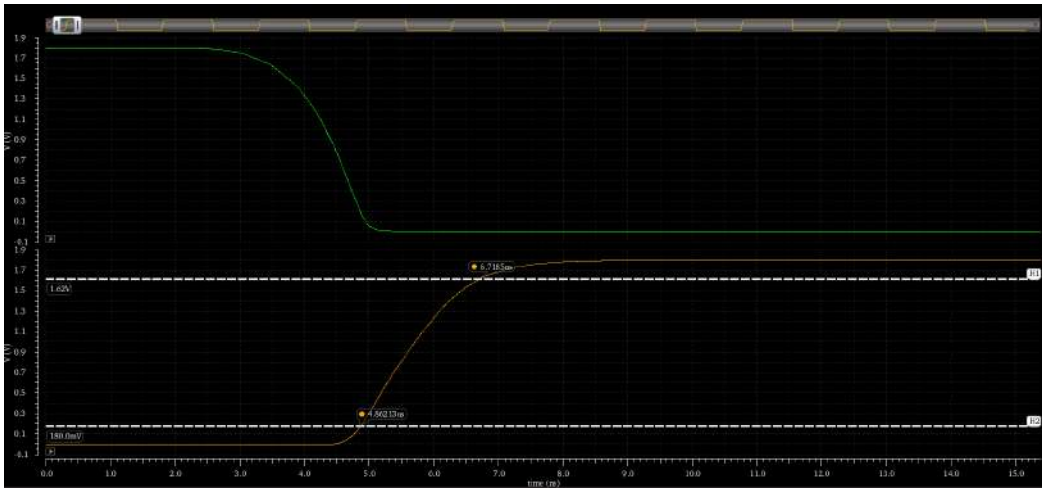


Figure 26: Rise Time with fanout-3 circuit

The following is the Fall Time Observed:

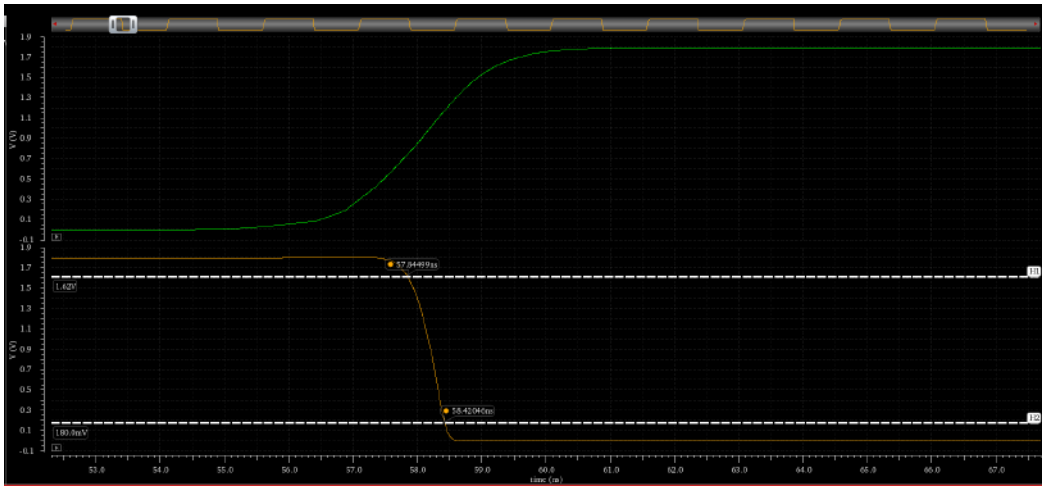


Figure 27: Fall Time with fanout-3 circuit

Observations

We observed the following things:

- Rise Time = 1.856 ns [fanout-3]
- Fall Time = 0.5775 ns [fanout-3]
- Logic Threshold Voltage = 899.14 mV

We can clearly see that the rise time and fall time are approximately same and the logic threshold voltage is approximately $V_{dd}/2$

Part 3: Designing of JK Latch

Design

- Complete NAND gate topology and the topology with NAND, NOR gate mixed topology were used to make positive level triggered JK Latch. Out of the 2 topologies, the fully NAND one was unable to toggle fully when J,K both were "1", due to the feedback signal coming from output Q,$\overline{Q}$ to K and J reached final stage gates even before the outputs reached VDD or GND level.
- For the NAND, NOR gates mixed topology, as the NOR has a delay more than equivalent NAND gate, it did not require any additional delay in the feedback path to function correctly.

Schematic

The schematics of JK-Latch are:

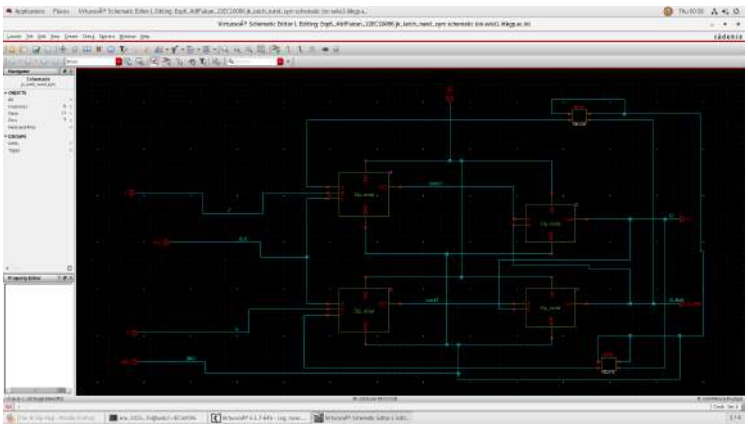


Figure 28: JK Latch made of NAND gates only. The delay element can either be replaced with a significantly long inverter chain (to observe toggling) or can be removed (considering the state to be invalid)

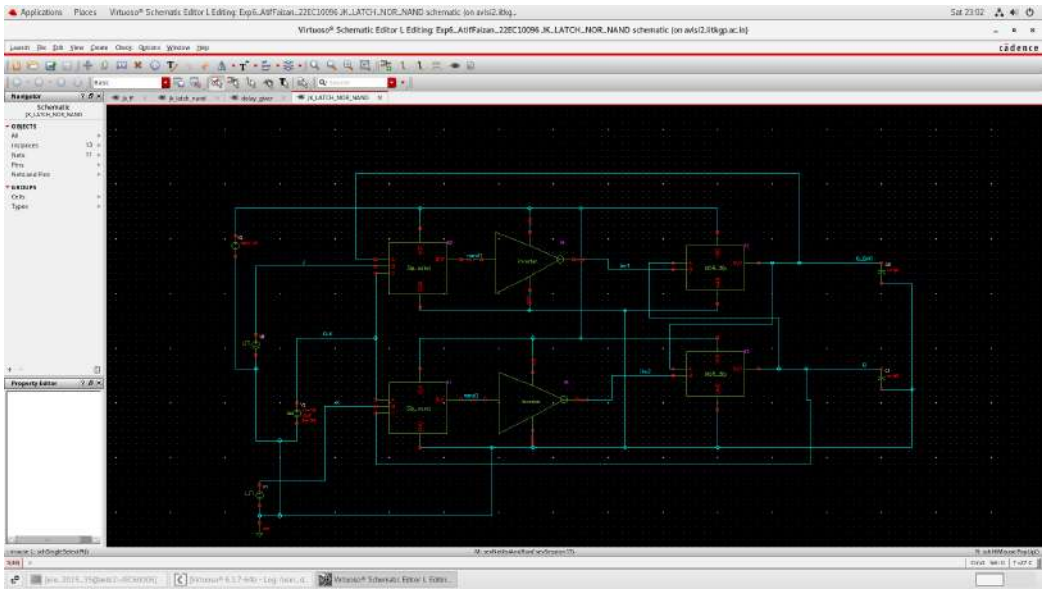


Figure 29: JK Latch made of NAND,NOR gates

0.1 Simulation Results

For the purely NAND gate based JK latch with a 5ns delay element, following are the results:

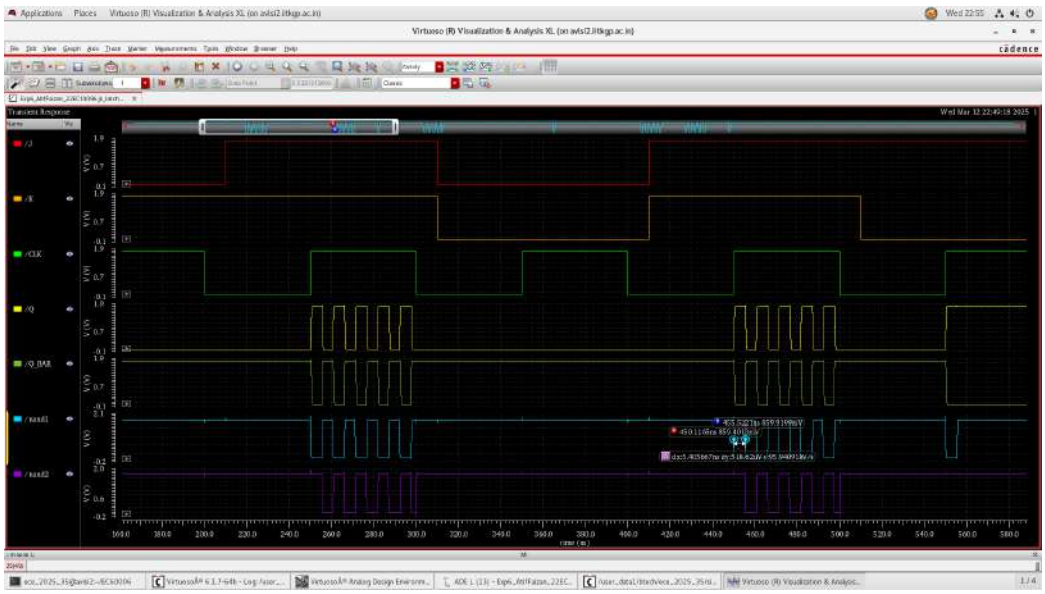


Figure 30: Input output waveform

The clock-data delay: Measured difference between midpoints of transition, when inputs are constant, just the clocks level change triggers output change.

Figure 31: Clock-data delay of JK Latch

The data-out delay: Measured in similar way, the clock level remains high, just the inputs change.

Figure 32: Data-out delay of JK latch

Observations for NAND gate based JK latch

- **Clock to Out Delay:** 227.5 ps
- **Data to Out Delay:** 426 ps

For the JK Latch made of NAND,NOR gates, following are the results:

Figure 33: Input output waveform

The clock-data delay: Measured difference between midpoints of transition, when inputs are constant, just the clocks level change triggers output change.

Figure 34: Clock-data delay of JK Latch

The data-out delay: Measured in similar way, the clock level remains high, just the inputs change.

Figure 35: Data-out delay of JK latch

Figure 36: Toggling Time-Period for JK latch made of NAND,NOR gates

Observations for NAND gate based JK latch

- **Clock to Out Delay:** 266.64 ps
- **Data to Out Delay:** 280 ps
- **Toggling Period:** 790 ps

Part 4: Designing of D Flip-flop

Design

- The Flip-flop can be made in master-slave topology, using 2 opposite level triggered latches in cascaded manner.
- D-latch was made from JK-latch,using an inverter in between data (D) and K, feeding the Data (D) directly to J.
- The D latches are cascaded, first the positive level triggered,then negative level triggered,thus making it negative edge triggered.

Schematic

The schematics used to make D flip-flop:

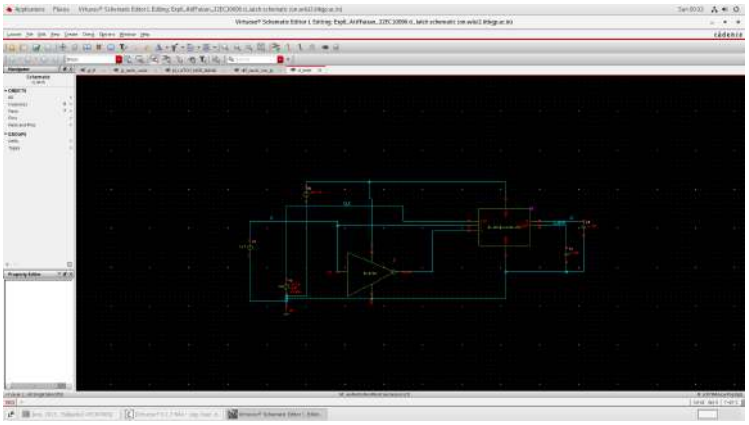


Figure 37: D-latch from JK latch

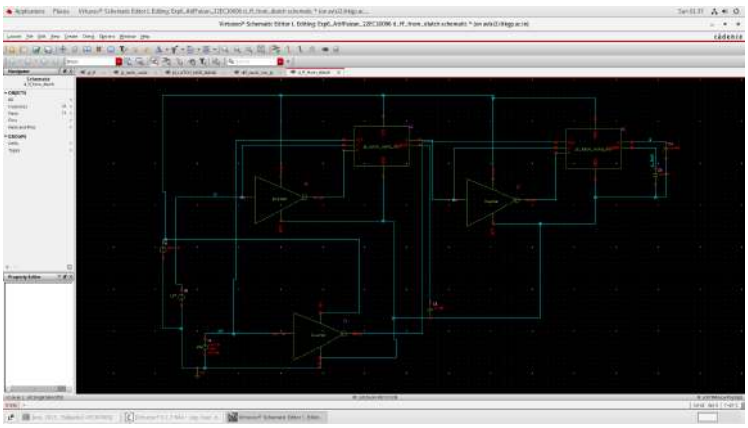


Figure 38: D-Flip-flop from JK latch

0.2 Simulation Results

Figure 39: Input output waveform

The clock-data delay: Measured difference between midpoints of transition, when inputs are constant, just the clock's falling edge triggers output change.

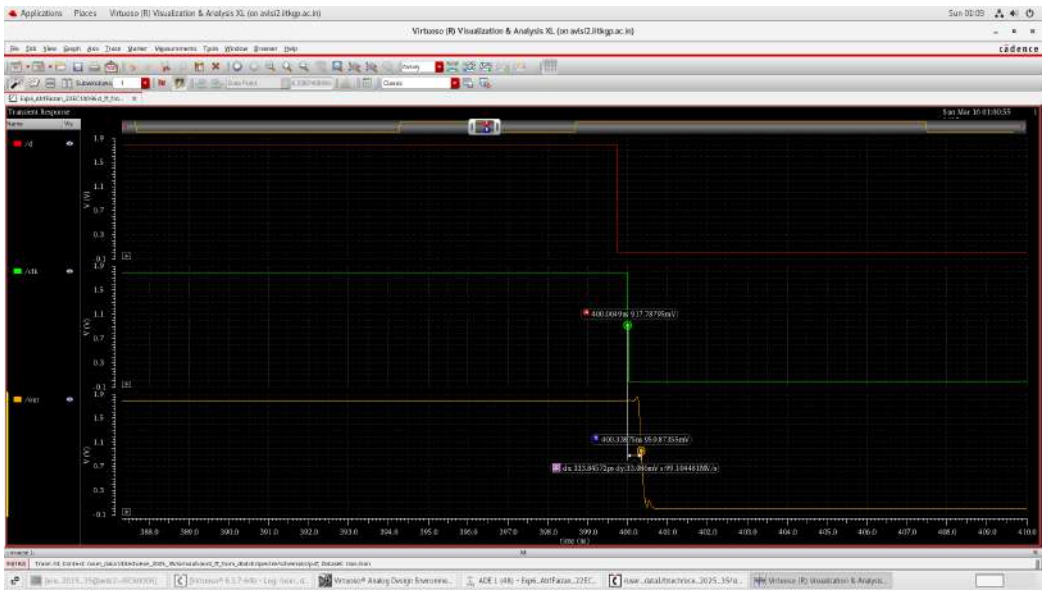


Figure 40: Clock-data delay of D-Flip-flop

Setup time: Minimum time before the active edge of clock, the input signal needs to remain constant.

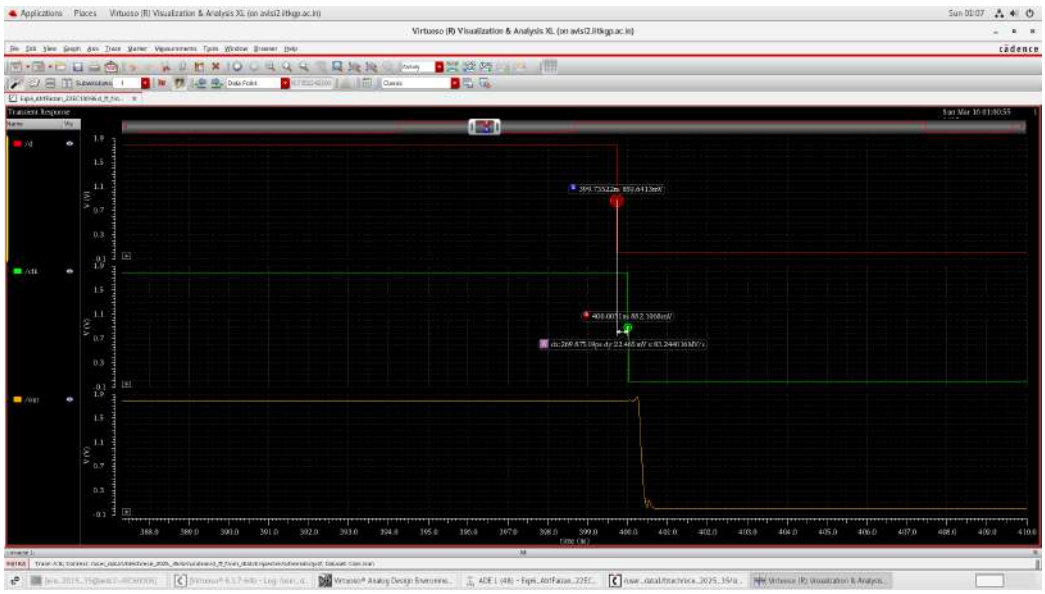


Figure 41: Minimum required setup time

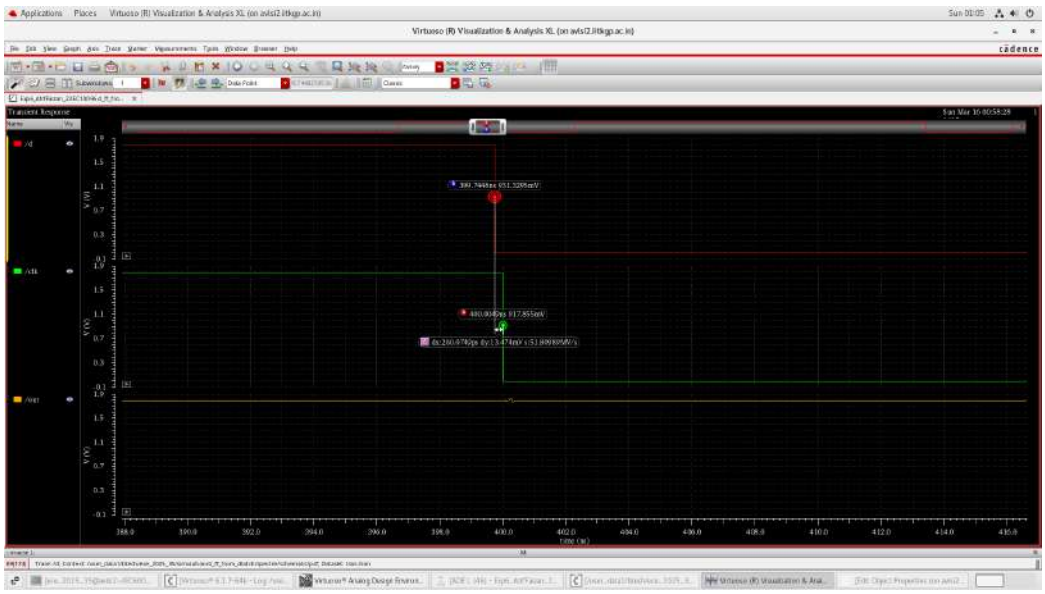


Figure 42: setup time at the critical limit

Hold time: Minimum time after the active edge of clock, till which input data needs to stay constant.

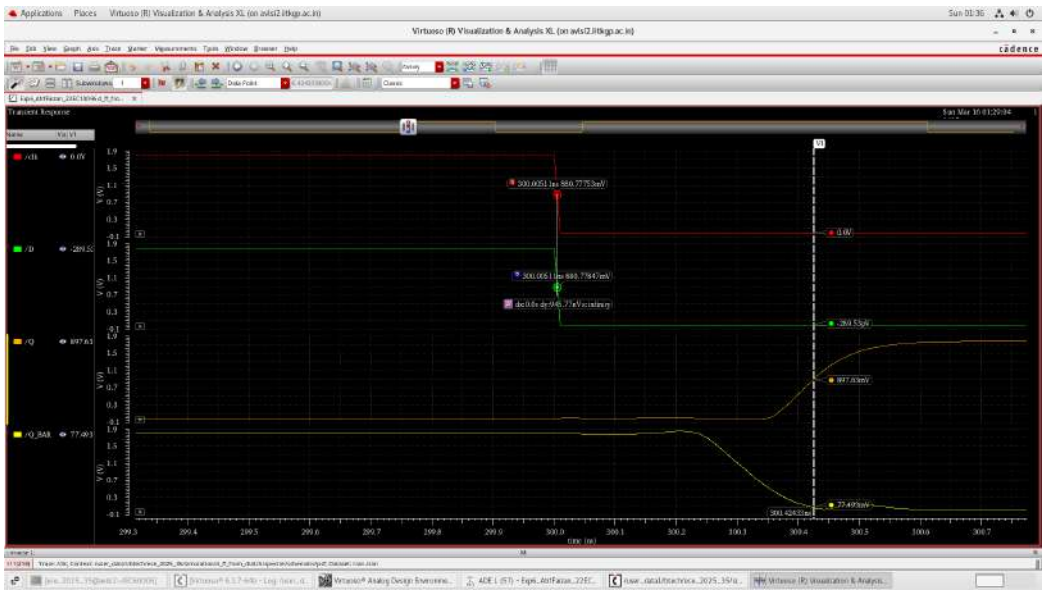


Figure 43: Hold time for D-flip flop

Observations for D-flip-flop

- **Clock to Out Delay:** 333.845 ps
- **Setup Time:** 269 ps
- **Hold Time:** 0 ps (approx) (from the plots)

Discussion Atif Faizan 22EC10096

- In this experiment we designed a 2-input and 3-input NAND and NOR gates, JK Latch and a D flip flop using the gates. We optimized our design by reducing the size of the NMOS and PMOS to reduce to size of the gates ensuring logical threshold voltage to be $V_{dd}/2$ and equalizing the rise time and fall time to maintain symmetry. Apart from that in the JK latch design we observed the data-to-out and clock-to-out delay and we experimented with different topologies (only using NAND gates,only using NOR gates, using both NAND and NOR gates)

to design a stable JK latch. We used the JK latch to design a D flip flop and observed the setup time and hold time which illustrate why changing the input and the clock at the same time in the D-flip flop may lead to indeterminate output.

- We used the Complementary (CMOS) technology for designing the NAND and NOR gates because there are many advantages CMOS technology has over any other methods like no static power consumption (if we ignore the leakage current), high noise margins, less area (using resistive loads increases the area), non-ratioed logic, symmetric propagation delay (for this all the mosfets must be appropriately sized), etc.

- Sizing of mosfets: if n NMOS are connected in parallel then size of each of the NMOS = size of the NMOS of a unit sized inverter and if n NMOS are connected in series the size of each of the NMOS is n times the size of the NMOS of a unit sized buffer. The same is done with PMOS. This is done to ensure the symmetric propagation delay (keeping in mind the worst case delay).

- We designed and tested our NAND and NOR gates against a fanout of 3. Fanout of x means the maximum number of similar devices that a given device can drive without significant output distortion. As the number of devices in the load side increases the capacitance of the load increases and hence the delay increases. We tested out gates for a fanout of 3 i.e. a delay of $t_{p0} * (1 + 3C/C) = 4t_{p0}$. If these gates need to drive a load of larger capacitance appropriately sized inverter chain must be used in between the load the gates so as to reduce the delays similar to what we did in the Clock Driver design in the previous experiment.

- Latches and flip-flops are both types of sequential circuits used for storing binary data, but they differ in how they operate and respond to inputs. Latches are level-sensitive devices that change output as long as the enable signal is active. Flip-flops are edge-triggered devices that store data only at clock edges, making them more reliable in synchronous systems.

- Complete NAND gate topology and the topology with NAND, NOR gate mixed topology were used to make positive level triggered JK Latch. Out of the 2 topologies, the fully NAND one was unable to toggle fully when J,K both were "1", due to the feedback signal coming from output Q,Q to K and J reached final stage gates even before the outputs reached VDD or GND level.

- For the NAND, NOR gates mixed topology, as the NOR has a delay more than equivalent NAND gate, it did not require any additional delay in the feedback path to function correctly.

- JK Latch made of NAND gates only. The delay element can either be replaced with a significantly long inverter chain (to observe toggling) or can be removed (considering the state to be invalid as announced by professor in the lab).

- Master Slave Configuration for Designing Flip Flops: The Master-Slave flip-flop configuration uses two latches to create a reliable, edge-triggered flip-flop. It consists of a Master latch (active during the clock HIGH phase) and a Slave latch (active during the clock LOW phase). When the clock is HIGH, the master captures the input but does not pass it to the output. When the clock transitions to LOW, the master becomes inactive, and the slave latch updates the output with the master's stored value. This ensures that changes in input affect the output only at the clock edge, eliminating race conditions and making it suitable for synchronous circuits.

- For the latch, We observed the data-to-out delay which is the delay between the input and output when clock is 1 and input changes. We also observed the clock-to-out delay which is the delay between the output and the clock when input remains constant and the clock changes.

- A D flip-flop can be designed using a JK latch by modifying its inputs to ensure predictable behavior. In a JK latch, when $J = K = 0$, the state remains unchanged; when $J = 1, K = 0$, it sets; when $J = 0, K = 1$, it resets; and when $J = K = 1$, it toggles. To convert it into a D flip-flop, the J input is connected to D, and K is connected to the complement of D ($K = \overline{D}$). This ensures that the flip-flop always follows the D input—storing '1' when D is high and '0' when D is low—eliminating the toggle state and making it a reliable edge-triggered storage element.

Discussion Chiradip Biswas 22EC30016

- In the 1st part of experiment, we made NAND and NOR gate using CMOS logic.The CMOS logic is popular in logic gates design because rail to rail output voltage swing and a very small power leakage. The sizing of pMOS and nMOS were done keeping the dimensions of CMOS inverter in mind. When the MOS are connected in series, in place of a single M"originatingScript":"m2","payload":"guid":"471d388f-1191-455d-847b-25d3f09f497f3528ae","muid":"cd0abdac-2e8b-46eb-b3c1-939a56702f3511ca68","sid":"0defe86e-6f93-42fb-bfe3-97d7080 the Widths are set to be double the width of single nMOS. When they are connected in parallel, the widths are kept same. This is done instead of halving the width, to combat the worst case delay, which arises when only 1 of the parallel MOS is on. At that time so that the current drawing capability of the gate remains same as the inverter, the width is chosen in this way.
- In the NAND gate, the pMOS are connected in parallel in pull up network, and in NOR the nMOS are in parallel in pull down network. As the width of pMOS is kept higher than that of nMOS, the intrinsic delay caused by NOR gate is more than that of NAND gate, for a given number of inputs and a given fanout.
- when the fanout of these gates are increased, the rise time, fall time, propagation delay increases, as there are more capacitance added at output.
- NAND and NOR gates are universal logic gates. So any boolean logic can be implemented using these gates, individually.
- A latch is a basic memory element that stores one bit of data and is generally level-triggered. The basic motto is to make the output stable for a given input and store the output accordingly. JK latch is a special type of latch which is a derivative of SR latch, just eliminating the invalid state (racing condition), arising from S=R="1". Instead, when J=K=1, the output toggles.
- The output of a NAND gate based JK latch is affected by the 1st stage of NAND gates and also by the feedback connection from output to the input side. This is the reason, why the memory or the data storing states are stabilized more, and also why the flip flop toggles when J=K=1. When this pair of input is given, if the previous Q and $\overline{Q}$ were 0 and 1 respectively, in the next cycle the new output will be 1,0. This new output will again come via the feedback path and will change the output from 1,0 to 0,1 again. Now if the feedback path signal arrives at the final stage of NAND gates before the output settles to VDD or GND (risetime, falltime are high), a glitch is observed at the output (the voltage trying to fall down, but in between was pulled up via feedback voltage.).
- For our case, when we made the jk latch using NAND gates only, this glitch was observed. To correct this, we found the required delay to obtain toggling by adding a delay element in the feedback path. Then the delay element was replaced by a buffer made of chain of inverters.
- Flip flop is an edge triggered version of latch. this can be made by either master-slave configuration or using the hazard. For Master-slave configuration, 2 latches are used one as positive level triggered and the other as negative level triggered. If the master one is positive level triggered it becomes negative edge triggered flip flop. If the other one happens it becomes rising edge triggered.
- For the Hazard based flip flop, the clock is given through a 2 input and gate with 1 input as clock and the other is clock bar. As the NOT gate will have a finite propagation delay, the inputs of AND gate will be "11" for a short period of time. So the output will be 1 for that period. Thus we are able to detect the rising transition of clock. This AND gate output when fed as the clock of JK latch, it acts as rising edge triggered JK flip flop.
- The D latch can be made from JK latch, by giving D to J and $\overline{D}$ to K. For D latch, the output becomes whatever be the input data, when active level of clock arrives. So To make the D flip flop, we cascade 2 D latches with opposite level triggered and feed the data of slave latch with output of master latch.
- The evaluation parameters of a Latch are clock to out delay and data to out delay. The first one is defined as the time delay, after the clock transitions from its inactive to active level, after which the output changes in respond to the input, given that input remains stable throughout the clock transition. The data-out delay is defined as time delay after the change in input data, given the clock level remains same throughout, causing the change in output.

- For a good Flip-flop we want these time parameters to be as small as possible.

Dept. of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

VLSI ENGINEERING LABORATORY [EC39004]

EXPERIMENT - 8

Physical design & Layout Checks of CMOS Inverter

Group No. 35

- **Member 1:** Atif Faizan 22EC10096
- **Member 2:** Chiradip Biswas 22EC30016
- **Group no.:** 35 (Monday)
- **Date of Submission:** 06-04-2025

Aim

1. Layout design of a smallest possible inverter with "equal" rise-to-fall and fall-to-rise

2. Make it DRC (Design Rule Checker) and LVS (Layout vs. Schematic check) clear physical design.

Theory

- Layout of any schematic is the real life positioning and fabrication blue-print, that is given to the manufacturing Foundry, to manufacture the IC, using silicon wafers.
- It involves placing and routing components such as transistors, interconnects, and other elements. Also this part of VLSI, deals with the materials to be used for designing each layer of the circuit. For example, the polysilicon gates of the MOS are to be connected to metal-to-poly silicon Via before connecting to metal directly. Using metals for making the electrical connections.
- The layout must follow design rules set by the fabrication process to ensure manufacturability. It directly affects the performance, area, and power consumption of the chip. A well-optimized layout reduces parasitic capacitances and resistances, enhancing speed. It also ensures proper signal integrity and minimizes noise and crosstalk.
- There are some rule-checks that, the layout of the circuit must follow. These are:
  - **DRC:** Design Rule Check. The purpose of this check is to ensure that the layout follows all the fabrication process rules defined by the foundry.It checks for issues like minimum width of metal lines, spacing between layers, overlapping requirements, enclosure rules, etc. Violating design rules can result in fabrication defects or failure of the chip during operation. eg: If two metal lines are too close, it might cause a short circuit due to bridging during manufacturing.
  - **LVS:** Layout Vs Schematic. LVS verifies that the physical layout (what's to be fabricated) matches the original schematic (design intent).It compares the connectivity, number of devices, and device types in the layout against the schematic.A mismatch means the fabricated chip may not function as intended, even if the layout passes DRC. eg: If a transistor is missing or connected incorrectly in the layout compared to the schematic, LVS will detect it.

Part 1: Designing a Minimum Sized Inverter,From Expt:5

Design

We chose the following design parameter:

- $(W/L)_p/(W/L)_n = 1.6u/0.42u$
- $V_{dd} = 1.8V$
- $V_{pulse} = 1.8V$ with $TimePeriod = 100ns$ $PulseWidth = 50ns$ $RiseTime = 10ns$ $FallTime = 10ns$ $DutyCycle = 50\%$

Schematic

The following is the schematic of the CMOS inverter:

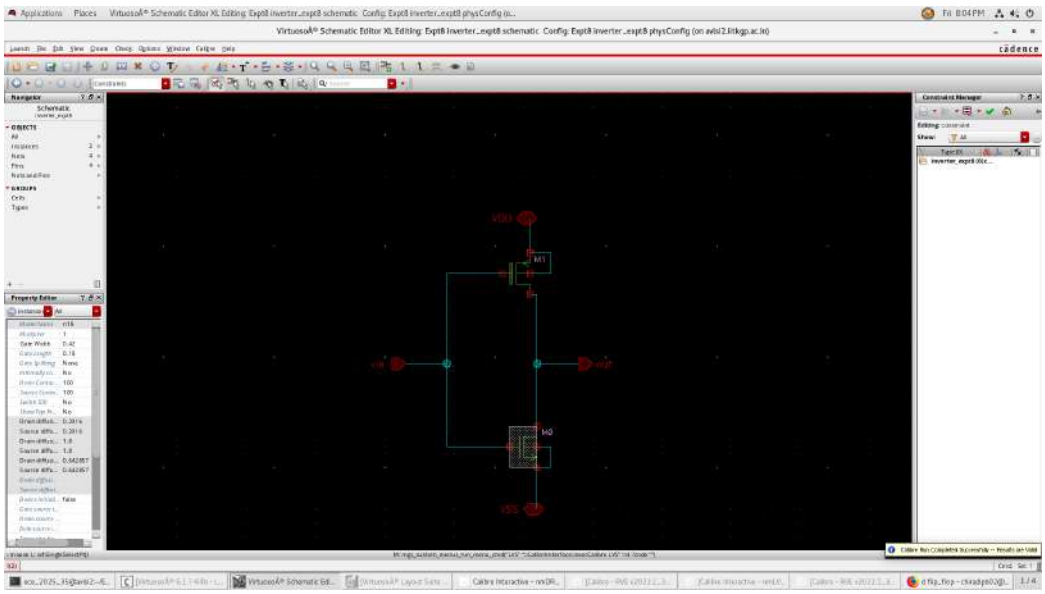


Figure 1: CMOS Inverter Schematic

0.1 Schematic ADEL simulation results:

We observed the following specicfications of this design schematic:

- Rise Time = 636.7 ps
- Fall Time = 739.4 ps
- Logic Threshold Voltage = 940.374 mV

We can clearly see that the rise time and fall time are approximately same and the logic threshold voltage is approximately $V_{dd}/2$

We then make a symbol from this schematic and then use the Layout XL option to make the layout of this.

Layout From this Inverter Schematic

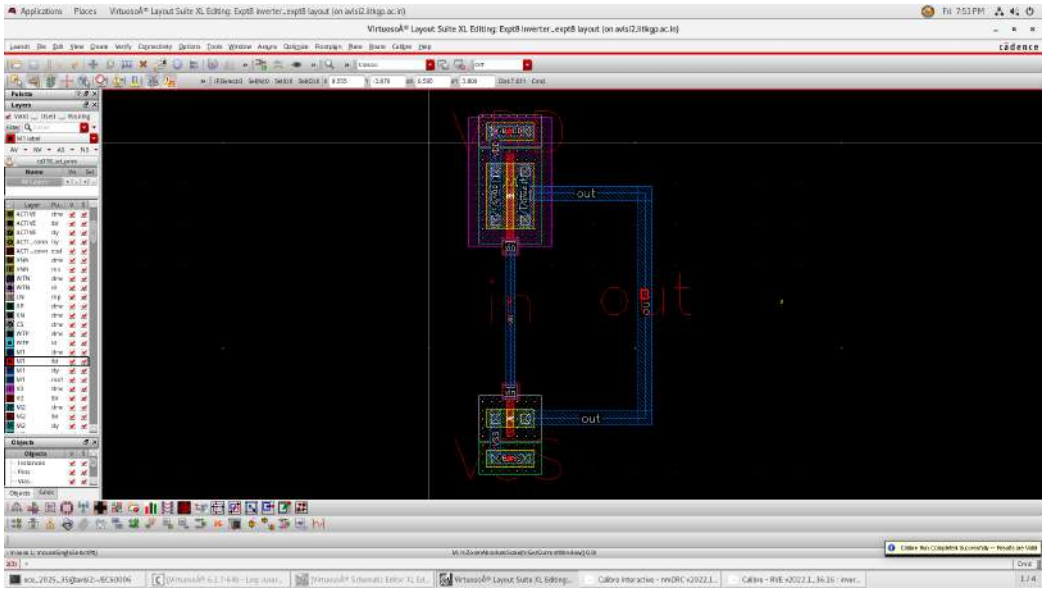


Figure 2: The layout of the inverter

After completing the Layout and pin making on the metal lines, we need to check whether the layout design is usable and it how closely will it resemble the original schematic, by testing: DRC and LVS checks.

The Various post-layout Checks

- **DRC check:** Using Calibre nmDRC using the rule files "DRC.header" we performed DRC check. The following are the DRC check and results:

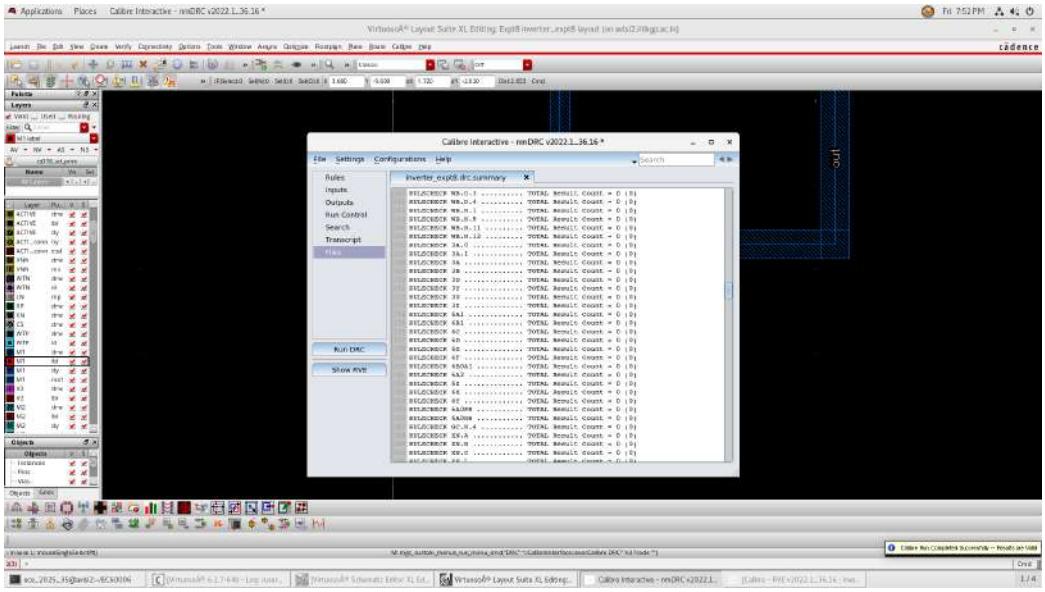


Figure 3: The DRC checking of inverter

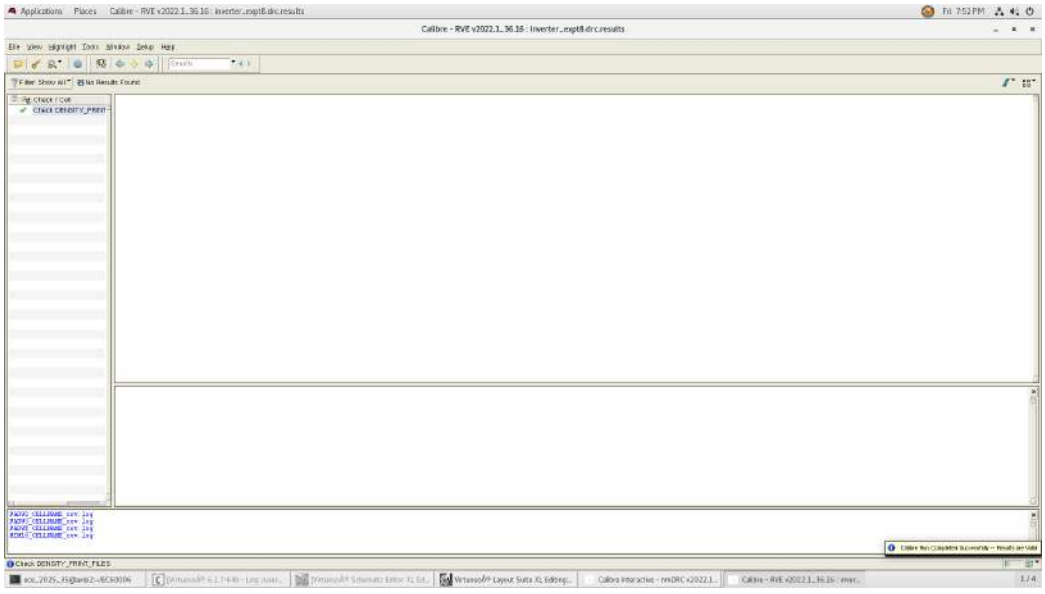


Figure 4: The DRC results of the inverter

- **LVS Check:** Using Calibre nmLVS, using the rulefiles: "LVS.header" we performed the LVS (Layout VS Schematic Check). During this, the additional SPICE file included was "scale.cdl". Following is the LVS result of the same layout:

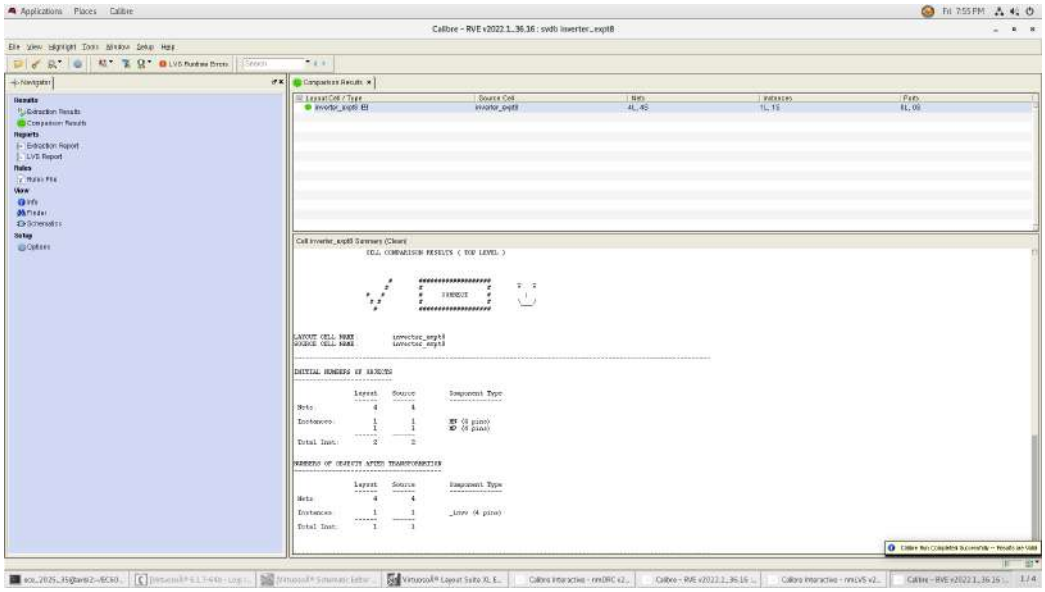


Figure 5: The LVS result of the Inverter

Discussion Atif Faizan 22EC10096

- This experiment focused on next step of making a VLSI design i.e. layout for final fabrication. The layout of a schematic serves as the physical representation and fabrication blueprint of an integrated circuit. It is provided to the manufacturing foundry, which uses it to produce the IC on silicon wafers.
- The layout is required to adhere to specific design rules defined by the fabrication process to guarantee successful manufacturing. It plays a crucial role in determining the chip's performance, size, and power efficiency. An efficiently designed layout minimizes parasitic capacitance and resistance, leading to faster operation. Additionally, it helps maintain signal integrity while reducing noise and crosstalk.
- We used a specialized Software called Calibre in Cadence for DRC LVS check of the layout of an inverter. DRC and LVS are two checks that are done on a layout before manufacturing.
- Design Rule Check (DRC) is a critical step that verifies whether the layout complies with all the manufacturing guidelines specified by the foundry. It inspects parameters such as the minimum width of metal traces, spacing between layers, overlap conditions, and enclosure constraints. Any violation of these rules can lead to serious fabrication issues or operational failures. For instance, if two metal lines are placed too close to each other, they might bridge during the manufacturing process, causing a short circuit.
- Layout Versus Schematic (LVS) is a validation process used to ensure that the physical layout accurately reflects the original schematic design. It checks whether the connectivity, device count, and types in the layout align with the intended circuit described in the schematic. Even if the layout passes Design Rule Check, a mismatch here could mean the final chip won't operate correctly. For example, if a transistor is absent or misconnected in the layout compared to the schematic, LVS will catch the discrepancy.
- Initially when we were doing the experiment we ran into a DRC error because the spacing between the via and the polysilicon line was not enough. After correcting this we passed the DRC check but encountered LVS check error because we did not create any pins in the layout.
- For creating the pins and for creating the connections metal M1 must be selected but it should be noted that the metal must be selected in drawing mode for making the connections and in label mode for making the pins. It should also be noted that the pins must be named the same (case sensitive) as we named various pins in the schematic otherwise it will throw LVS error. Also pin type should be properly selected. The vdd pin and gnd pin must be made Power Type and the vin and vout pin must be made of type signal.

- For connecting the gates of the nmos and pmos we must use polysilicon which can be selected using the shortcut key "P". It should be ensured that the gate of the nmos and pmos must align vertically otherwise it will not be possible for a straight polysilicon line to connect to two without any problem.
- We should not forget to enable the top tap parameters for the pmos and bottom tap parameters for the nmos connecting the source to gnd for nmos and source to vdd for pmos otherwise it will throw LVS error.
- The rule files for DRC were included in the DRC.header and the rule files for LVS were included in the LVS.header.
- A via is a small conductive hole or connection used in IC layout to link different metal layers vertically within a chip. Since modern integrated circuits have multiple layers of metal for routing signals, vias allow electrical connectivity between these layers. They are essential for efficient routing, enabling complex interconnections while optimizing chip area and maintaining signal integrity.
- It should be noted that we need to change the resolution to 0.001 and stoptime to 10 to see what is inside the nmos and pmos layout in Calibre Cadence.

Discussion Chiradip Biswas 22EC30016

- In this experiment we learnt to make the layout of the CMOS inverter that we prepared in experiment 5, with equal rise time and fall time. Then performed various checks on this prepared layout to ensure this is workable one.
- For this experiment, we had to install Calibre software in the cadence to perform the DRC and LVS checks.
- Even after making the schematic, we cannot directly use it to make the IC. For this purpose, we make the layout of the schematic and this s the file we send to the Foundry to make the IC. But even after making the layout this may not be be usable due to design problems or deviation from the schematic it was referred from. The design of the layout is cross validated by a set of layout rules, using Design Rule Check (DRC). The deviation of the prepared layout is checked using LVS (Layout VS Schematic Check).
  - DRC: This feature,for example warns us about the spacing between components or nmos, pmos placement on the substrate if it is not adequate, also warns us if the pmos and nmos are not alligned properly etc.
  - LVS: This feature warns us about the mismatch between design parameters of the schematic component and that in the layout one.
- Using A metal-polysilicon via is a must to make a conducting contact between the silicon and metal.
- To connect the gates of the NMOS and PMOS transistors, we use polysilicon. Make sure that the gates of the NMOS and PMOS are vertically aligned. This alignment is crucial for a straight polysilicon line to connect both gates without layout problems.
- To create pins and interconnections, Metal 1 (M1) layer must be used. However, it's important to switch to the "Drw" mode when making connections, and to the "Lbl" mode when placing the pins. Ensure that the pin names exactly match those used in the schematic—including case sensitivity—to avoid LVS errors. Also, set the correct pin types: assign "Power" type to vdd and gnd, and "Signal" type to vin and vout
- The shorting of body with the source is very crucial. The body terminal is shown in the pmos and nmos layout by enabling top and the bottom taps of pmos and nmos respectively. Then shorting them using metal lines.
- The tricky part was balancing the rise and fall times—how fast the inverter turns on and off. Our earlier simulations showed they were nearly equal (636ps vs. 739ps), which is good for performance. The "logic threshold" (where the inverter flips its output) was also near the midpoint of the supply voltage (940mV out of 1.8V), making it reliable.
- The post layout simulation can be done to test the layout generated component after obtaining the parasitic capacitance files (pex files). Efficient layout is necessary to avoid these parasitic caps.