# Indian Institute of Information Technology Vadodara International Campus Diu

Practical Workbook

|  |  |  |
|---|---|---|
| Name | : | Shreyas Ladhe |
| Roll No. | : | 202211081 |
| Branch. | : | Computer Science and Engineering |
| Batch | : | 2022 |
| Subject | : | Digital Logic Design |

# Certificate

This is certify that **Shreyas Ladhe** of B.tech of semester III .Enrollment Number **202211081** Branch Computer Science and Engineering (CSE) has been found satisfactory in the continuous internal evaluation of laboratory, practical and term work in the subject EC261 for the academic year 2022-23.

Signature

# List of Experiments

| Lab No. | Lab Name | Date |
|---|---|---|
| 1 | Familiarity with Logic Gates | 22 - 09 - 2023 |
| 2 | Circuits using universal Gates | 29 - 09 - 2023 |
| 3 | 2 bit operations and converters | 13 - 10 - 2023 |
| 4 | Multiplexers, encoders and decoders | 20 - 10 - 2023 |
| 5 | Latches and Flip flops | 27 - 10 - 2023 |
| 6 | Waveform for flip flops | 03 - 11 - 2023 |

## Contents

# 1 Lab-1 : Familiarity with Logic Gates

## 1.1 Aim

Basic familiarity with logic gate. Verify the truth table of given 74 series IC.

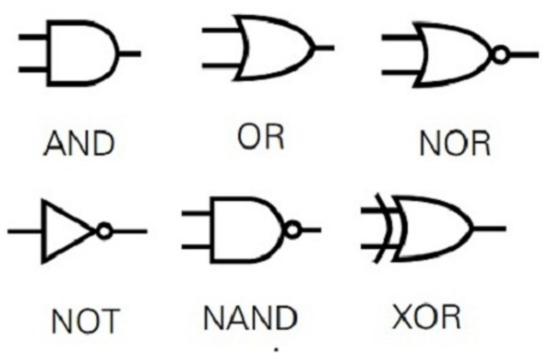
- AND
- NAND
- NOT
- NOR
- OR
- XOR

## 1.2 Apparatus

| Sr No. | Components | Quantity |
|---|---|---|
| 1 | Digital Trainer Kit | 1 |
| 2 | NAND (7400) | 1 |
| 3 | NOR (7402) | 1 |
| 4 | NOT (7404) | 1 |
| 5 | AND (7408) | 1 |
| 6 | OR (7432) | 1 |
| 7 | XOR (7486) | 1 |
| 8 | Connecting Wires | As Required |
| 9 | Bread board | 1 |

## 1.3 Theory

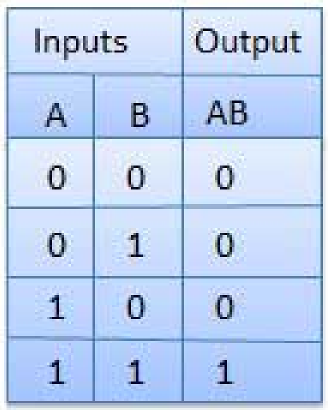
Here is a small rundown about the logic gates we are working on:



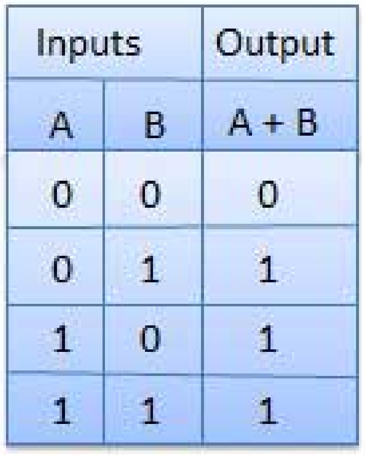
## 1.4 Observation

We verified the following truth tables for the logic gates: The sequence is as follows (left to right):-

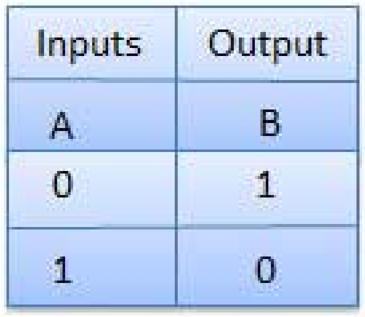
- AND
- OR
- NOT
- NAND
- NOR
- XOR



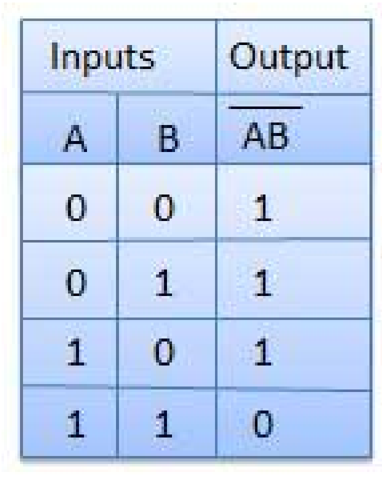
| Inputs | | Output |
|---|---|---|
| A | B | AB |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



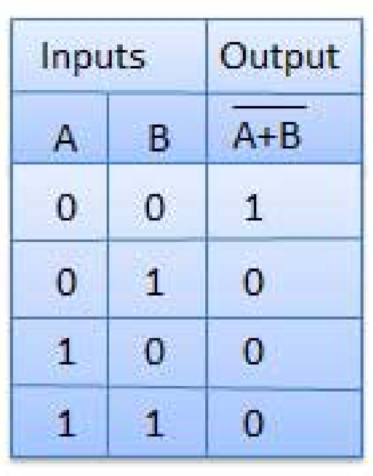
| Inputs | | Output |
|---|---|---|
| A | B | A + B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |



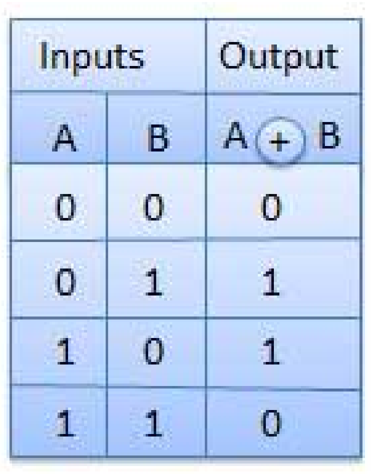
| Inputs | Output |
|---|---|
| A | B |
| 0 | 1 |
| 1 | 0 |



| Inputs | | Output |
|---|---|---|
| A | B | $\overline{AB}$ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



| Inputs | | Output |
|---|---|---|
| A | B | $\overline{A+B}$ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |



| Inputs | | Output |
|---|---|---|
| A | B | $A \oplus B$ |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

## 1.5 Conclusion

Learned about different IC and hands on experience on working with logic gates. Verified the working of all the logic gates.

# 2 Lab-2 : Circuits using Universal Gates

## 2.1 Aim

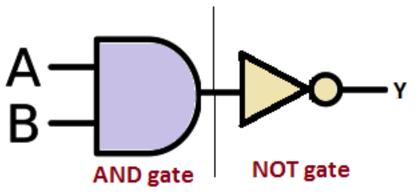
- Make the different basic logic gates using the universal gate.
- Make half adder and subtractor using universal gate.

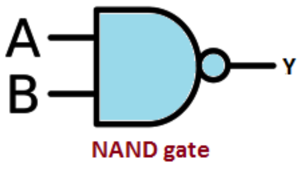
## 2.2 Apparatus

| Sr No. | Components | Quantity |
|---|---|---|
| 1 | Digital Trainer Kit | 1 |
| 2 | NAND (7400) | 1 |
| 3 | NOR (7402) | 1 |
| 4 | Connecting Wires | As Required |
| 5 | Bread board | 1 |

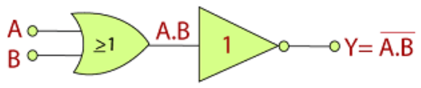
## 2.3 Theory

**NAND Gate**: combination of two basic logic gates, the AND gate and the NOT gate connected in series. The output of a NAND gate is high when either of the inputs is high or if both the inputs are low. In other words, the output is always high and goes low only when both the inputs are high.



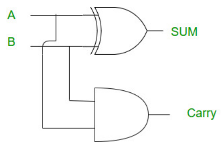


**NOR Gate**: A NOR gate ("not OR gate") is a logic gate that produces a high output (1) only if all its inputs are false, and low output (0) otherwise. Hence the NOR gate is the inverse of an OR gate, and its circuit is produced by connecting an OR gate to a NOT gate. Just like an OR gate, a NOR gate may have any number of input probes but only one output probe.A NOT gate followed by an OR gate makes a NOR gate.

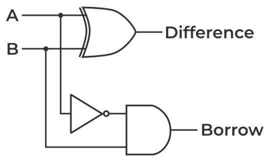


2- Input "AND" gate plus a "NOT" gate

**Half Adder Circuit**: A half adder is a digital logic circuit that performs binary addition of two single-bit binary numbers. It has two inputs, A and B, and two outputs, SUM and CARRY. The SUM output is the least significant bit (LSB) of the result, while the CARRY output is the most significant bit (MSB) of the result, indicating whether there was a carry-over from the addition of the two inputs. The half adder can be implemented using basic gates such as XOR and AND gates.

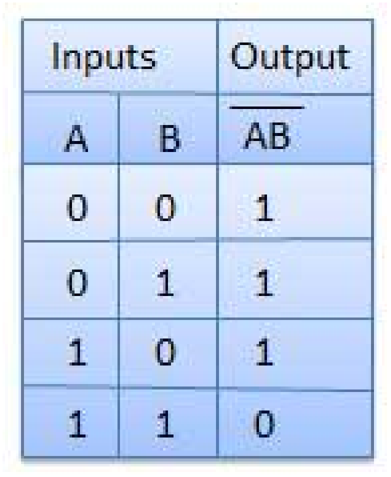


**Half Subtractor Circuit**: A half subtractor is a digital logic circuit that performs binary subtraction of two single-bit binary numbers. It has two inputs, A and B, and two outputs, DIFFERENCE and BORROW. The DIFFERENCE output is the difference between the two input bits, while the BORROW output indicates whether borrowing was necessary during the subtraction.The half subtractor can be implemented using basic gates such as XOR and NOT gates. The DIFFERENCE output is the XOR of the two inputs A and B, while the BORROW output is the NOT of input A and the AND of inputs A and B.

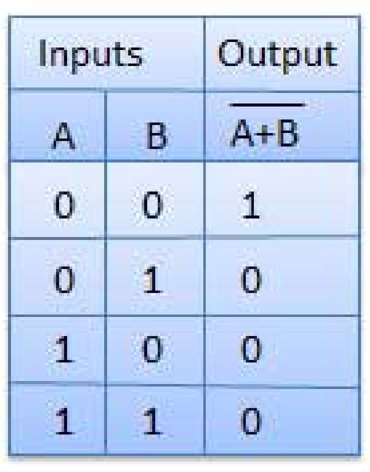


## 2.4 Observation

We made circuits using NAND and NOR Gate. The following are the truth tables for the universal gates:

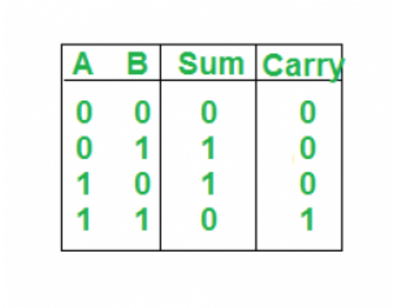


| Inputs | | Output |
|---|---|---|
| A | B | $\overline{AB}$ |
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

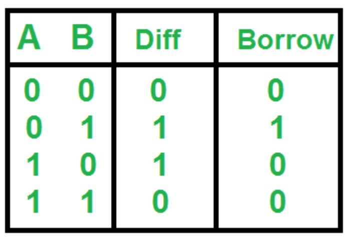


| Inputs | | Output |
|---|---|---|
| A | B | $\overline{A+B}$ |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Half Adder and Subtractor circuits are also made and the truth table for the same was verified:



| A | B | Sum | Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |



| A | B | Diff | Borrow |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

## 2.5 Conclusion

Made Basic Logic gates using universal gates (NAND and NOR) and made half adder and subtractor circuit and verified their truth table.

# 3 Lab-3 : 2-bit operations and converters

## 3.1 Aim

- 2-bit adder and subtractor using logic gate or universal gate.
- Binary to gray code converter.
- Binary to BCD converter.

## 3.2 Apparatus

| Sr No. | Components | Quantity |
|---|---|---|
| 1 | Digital Trainer Kit | 1 |
| 2 | NAND (7400) | 1 |
| 3 | NOR (7402) | 1 |
| 4 | AND (7408) | 1 |
| 5 | OR (7432) | 1 |
| 6 | Connecting Wires | As Required |
| 7 | Bread board | 1 |

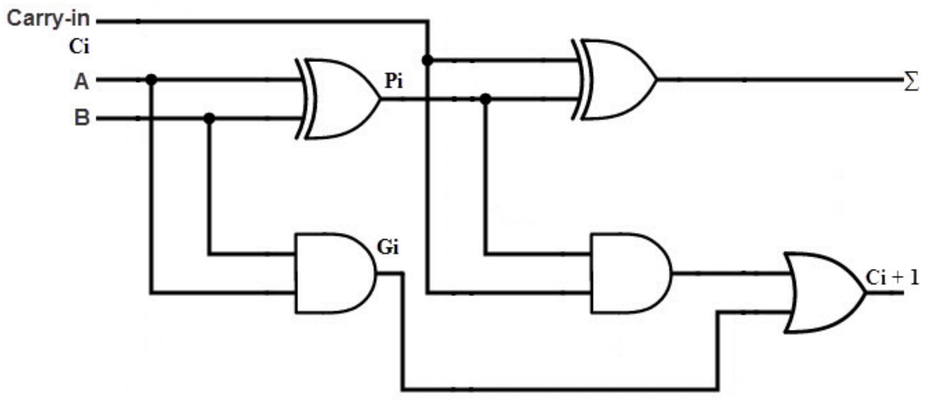
## 3.3 Theory

### 3.3.1 Explanation

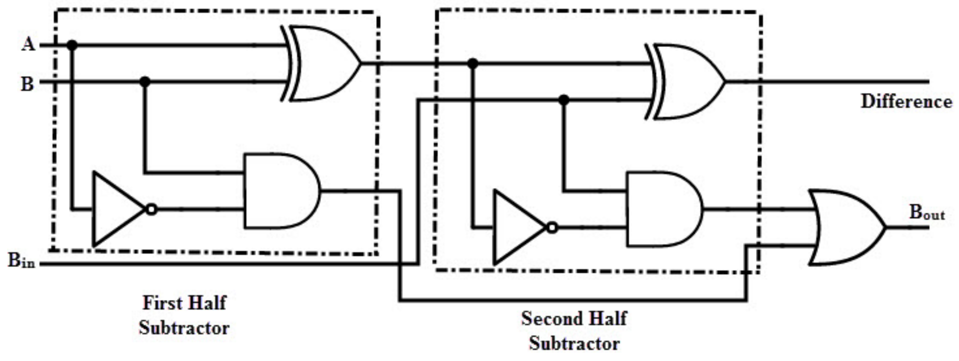
- Addition and Subtraction are two basic Arithmetic Operations that must be performed by any Digital Computer. If both these operations can be properly implemented, then Multiplication and Division tasks become easy (as multiplication is repeated addition and division is repeated subtraction).
- A Full Adder is a combinations logic circuit which performs addition on three bits and produces two outputs: a Sum and a Carry. As we have seen that the Half Adder cannot respond to three inputs and hence the full adder is used to add three digits at a time. It consists of three inputs, of which two are input variables representing the two significant bits to be added, whereas the third input terminal is the carry from the previous addition. The two outputs are a Sum and Carry outputs.

- A Full Subtractor is a combinations logic circuit which performs a subtraction between the two 1-bit binary numbers and it also considers the borrow of the previous bit i.e., whether 1 has been borrowed by the previous minuend bit.So, a Full Subtractor has three inputs, in which two inputs corresponding to the two bits to be subtracted (minuend A and subtrahend B), and a borrow bit, usually represented as Bin, corresponding to the borrow operation. There are two outputs, one corresponds to the difference D output and the other Borrow output Bo.

### 3.3.2 Circuit Diagrams for circuits



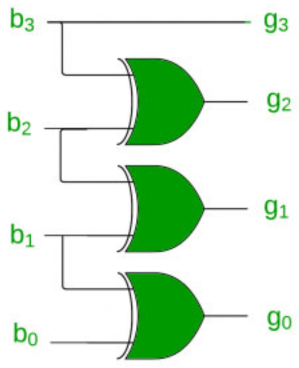
Full Adder Circuit



Full Subtractor Circuit

### 3.3.3 Converters

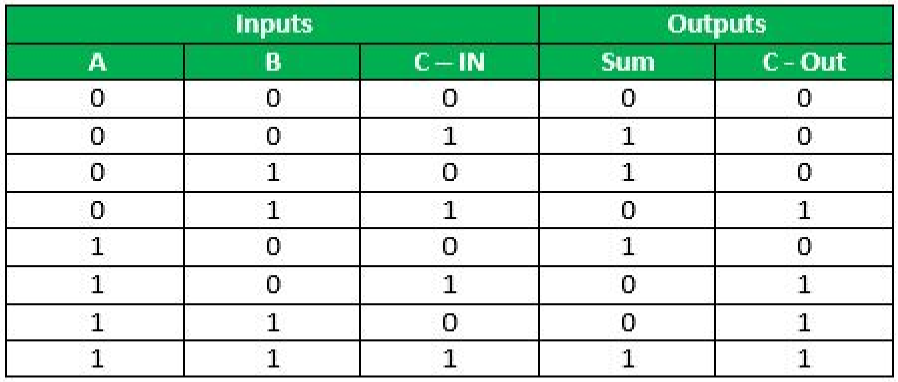
- Gray Code system is a binary number system in which every successive pair of numbers differs in only one bit. It is used in applications in which the normal sequence of binary numbers generated by the hardware may produce an error or ambiguity during the transition from one number to the next. For example, the states of a system may change from 3(011) to 4(100) as- 011 — 001 — 101 — 100. Therefore there is a high chance of a wrong state being read while the system changes from the initial state to the final state. This could have serious consequences for the machine using the information. The Gray code eliminates this problem since only one bit changes its value during any transition between two numbers.
- To find the corresponding digital circuit, we will use the K-Map technique for each of the gray code bits as output with all of the binary bits as input.
- The corresponding digital circuit –



- BCD is binary coded decimal number, where each digit of a decimal number is respected by its equivalent binary number. That means, LSB of a decimal number is represented by its equivalent binary number and similarly other higher significant bits of decimal number are also represented by their equivalent binary numbers.

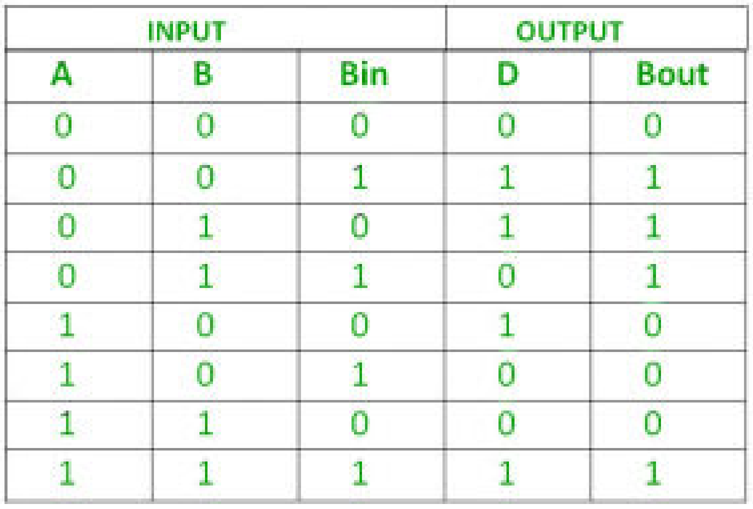
## 3.4 Observations

We made certain circuits to make a full adder and subtractor to perform 2 bit additions and subtractions.



| Inputs | | | Outputs | |
|---|---|---|---|---|
| A | B | C – IN | Sum | C - Out |
| 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 |

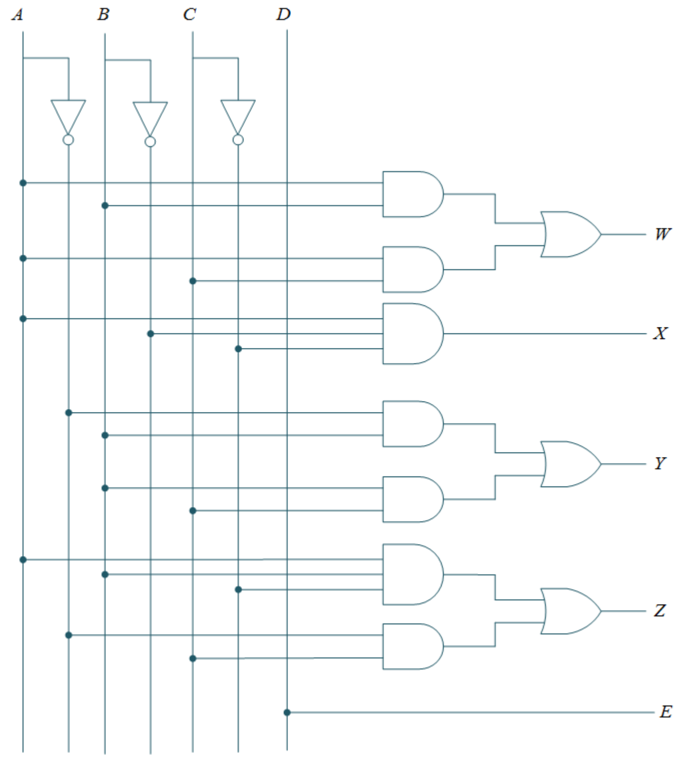
The truth table above is for a Full Adder Circuit.



| INPUT | | | OUTPUT | |
|---|---|---|---|---|
| A | B | Bin | D | Bout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |

The truth table above is for a full Subtract Circuit.

To convert Binary to Gray code we will use the following circuit:



## 3.5 Conclusion

Learned how to make 2 Bit adders and Subtractor Circuits and made Binary to Gray code as well as Binary to BCD converter using logic gates.

# 4 Lab-4 : Multiplexers, encoders and decoders

## 4.1 Aim

- 2-bit magnitude comparator
- $4 \times 1$ multiplexer using $2 \times 1$ multiplexer
- $4 \times 2$ encoder and $2 \times 4$ decoder

## 4.2 Apparatus

| Sr No. | Components | Quantity |
|---|---|---|
| 1 | Digital Trainer Kit | 1 |
| 2 | NAND (7400) | 1 |
| 3 | NOR (7402) | 1 |
| 4 | NOT (7404) | 1 |
| 5 | AND (7408) | 1 |
| 6 | OR (7432) | 1 |
| 7 | Connecting Wires | As Required |
| 8 | Bread board | 1 |

## 4.3 Theory

### 4.3.1 Explanation

- A magnitude digital Comparator is a combinational circuit that compares two digital or binary numbers in order to find out whether one binary number is equal, less than, or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and the other for B and have three output terminals, one for A ¿ B condition, one for A = B condition, and one for A ¡ B condition.
- The circuit works by comparing the bits of the two numbers starting from the most significant bit (MSB) and moving toward the least significant bit (LSB). At each bit position, the two corresponding bits of the numbers are compared. If the bit in the first number is greater than the corresponding bit in the second number, the A¿B output is set to 1, and the circuit immediately determines that the first number is greater than the second. Similarly, if the bit in the second number is greater than the corresponding bit in the first number, the A¡B output is set to 1, and the circuit immediately determines that the first number is less than the second.

- If the two corresponding bits are equal, the circuit moves to the next bit position and compares the next pair of bits. This process continues until all the bits have been compared. If at any point in the comparison, the circuit determines that the first number is greater or less than the second number, the comparison is terminated, and the appropriate output is generated. If all the bits are equal, the circuit generates an A=B output, indicating that the two numbers are equal.

- A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator. It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.

- An encoder is a digital circuit that converts a set of binary inputs into a unique binary code. The binary code represents the position of the input and is used to identify the specific input that is active. Encoders are commonly used in digital systems to convert a parallel set of inputs into a serial code.

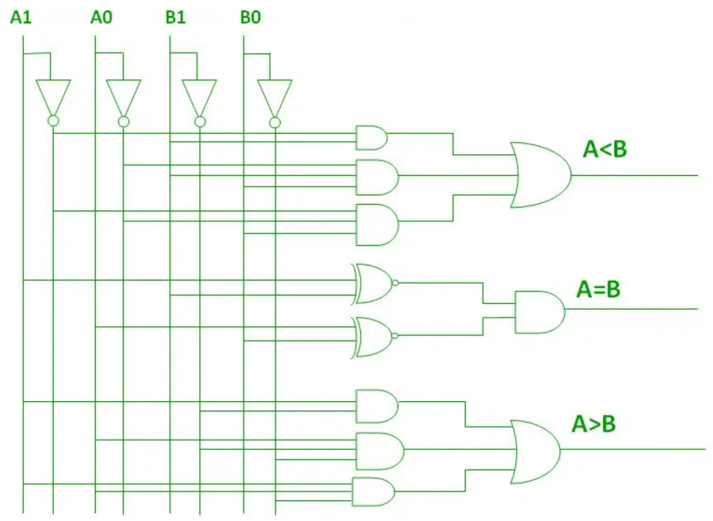
- The basic principle of an encoder is to assign a unique binary code to each possible input. For example, a 2-to-4 line encoder has 2 input lines and 4 output lines and assigns a unique 4-bit binary code to each of the $2^2 = 4$ possible input combinations. The output of an encoder is usually active low, meaning that only one output is active (low) at any given time, and the remaining outputs are inactive (high). The active low output is selected based on the binary code assigned to the active input.

- Decoder is a combinational circuit that has 'n' input lines and maximum of 2n output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the min terms of 'n' input variables lines, when it is enabled.

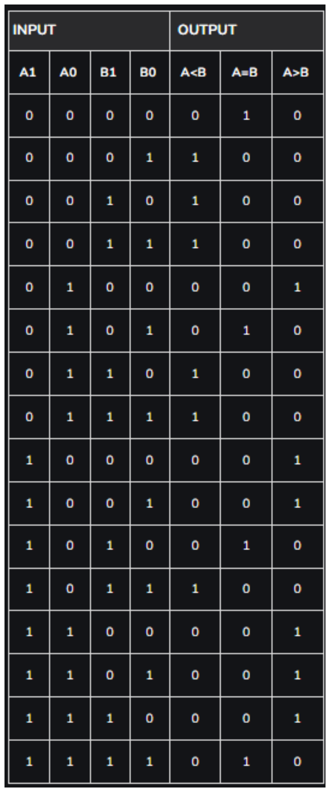
- The 4:1 multiplexer also known as 4:1 mux, offers several advantages in digital circuit design such as space efficiency. It also reduced the complexity of the overall circuit by reducing the no of components required to perform the task.A 4:1 mux can be configured to implement various logical functions. By properly setting the input lines, the mux can perform operations such as AND, OR, XOR and more.

### 4.3.2 Circuit diagrams and Truth Tables

- Circuit diagram for 2 bit comparator

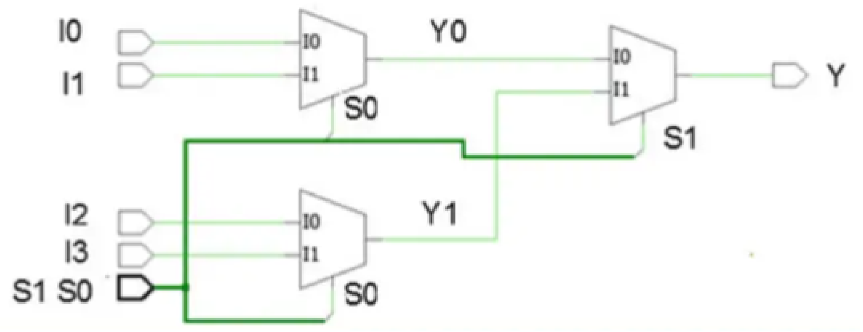


- Truth table for 2 bit comparator

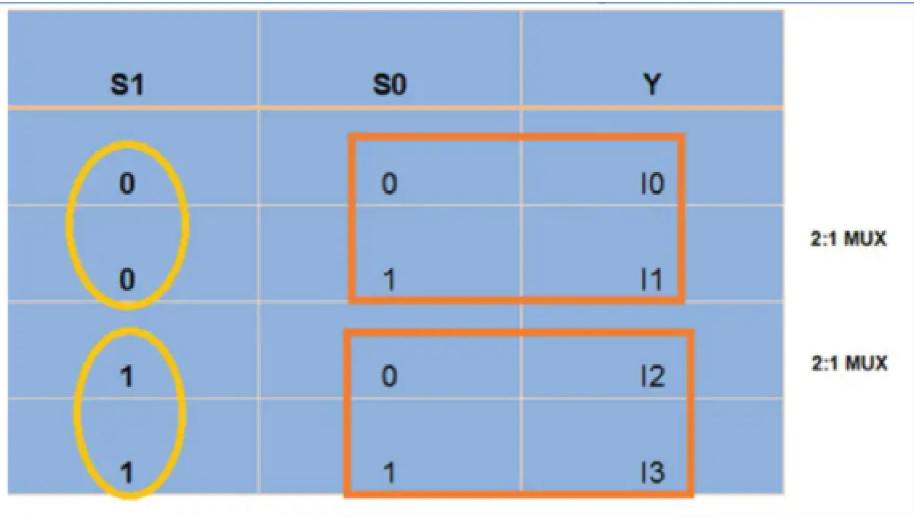


| INPUT | | | | OUTPUT | | |
|---|---|---|---|---|---|---|
| A1 | A0 | B1 | B0 | A<B | A=B | A>B |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

- Circuit diagram for 4 × 1 multiplexer using 2 × 1 multiplexer

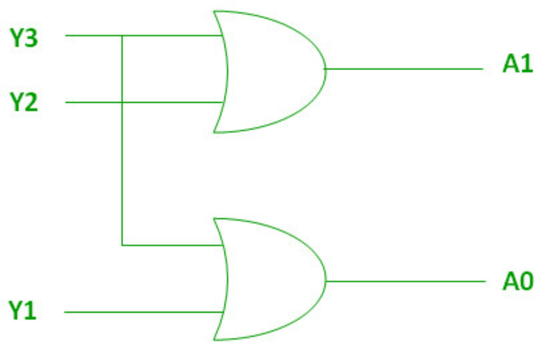


- Truth table for 4 × 1 multiplexer using 2 × 1 multiplexer

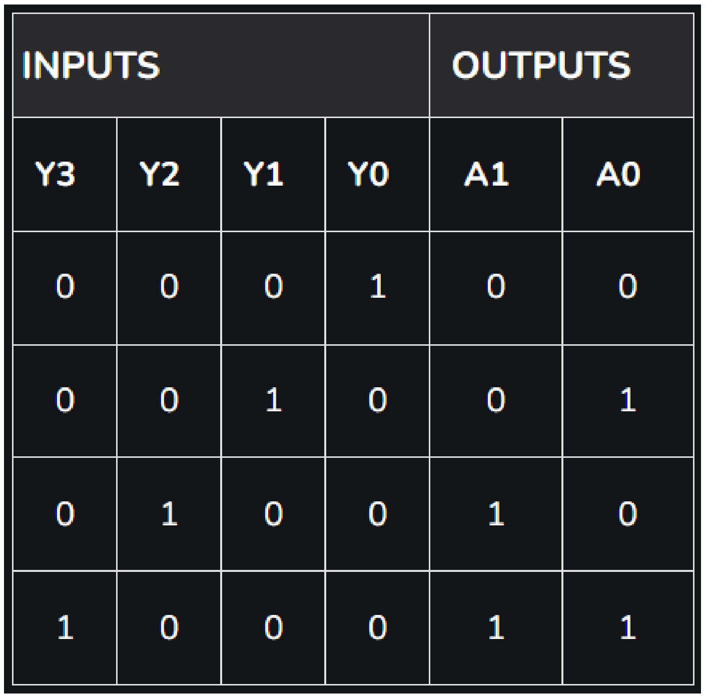


| S1 | S0 | Y | |
|---|---|---|---|
| 0 | 0 | I0 | |
| 0 | 1 | I1 | 2:1 MUX |
| 1 | 0 | I2 | 2:1 MUX |
| 1 | 1 | I3 | |

- Circuit diagram for 4 × 2 encoder

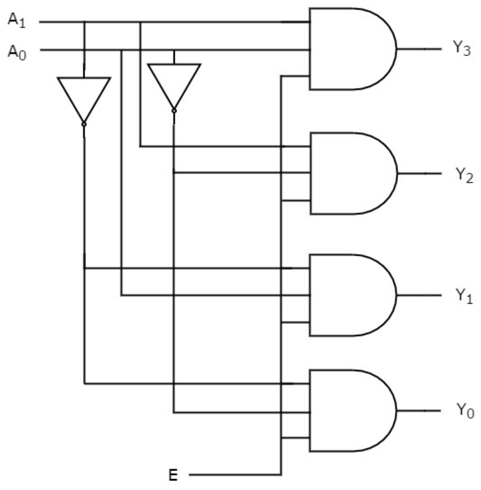


- Truth table for 4 × 2 encoder

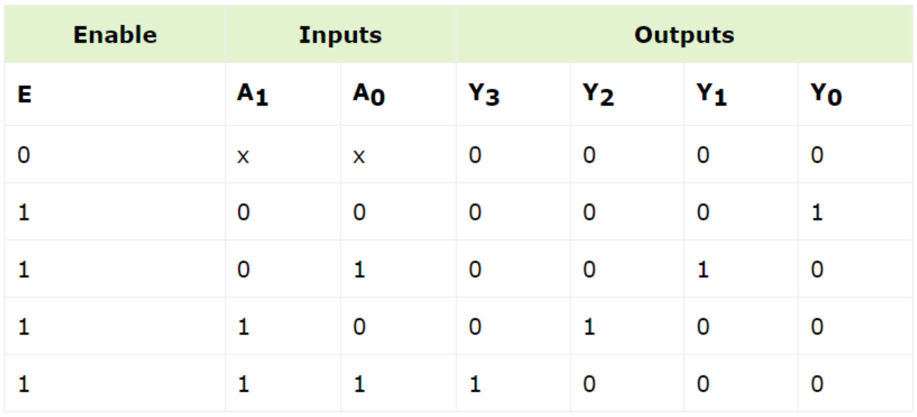


| INPUTS | | | | OUTPUTS | |
|---|---|---|---|---|---|
| Y3 | Y2 | Y1 | Y0 | A1 | A0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

- Circuit diagram for 2 × 4 decoder



- Truth table for $2 \times 4$ decoder



| **Enable** | **Inputs** | | **Outputs** | | | |
|---|---|---|---|---|---|---|
| **E** | **$A_1$** | **$A_0$** | **$Y_3$** | **$Y_2$** | **$Y_1$** | **$Y_0$** |
| 0 | x | x | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 |

## 4.4 Observations

- Boolean expression for 2 bit comparator

```
A>B:A1B1’ + A0B1’B0’ + A1A0B0’
A=B: A1’A0’B1’B0’ + A1’A0B1’B0 + A1A0B1B0 + A1A0’B1B0’
   : A1’B1’ (A0’B0’ + A0B0) + A1B1 (A0B0 + A0’B0’)
   : (A0B0 + A0’B0’) (A1B1 + A1’B1’)
   : (A0 Ex-Nor B0) (A1 Ex-Nor B1)
A<B:A1’B1 + A0’B1B0 + A1’A0’B0
```

- Boolean expression for 4 × 1 multiplexer

$$Y = I0 \cdot \overline{S1} \cdot \overline{S0} + I1 \cdot \overline{S1} \cdot S0 + I2 \cdot S1 \cdot \overline{S0} + I3 \cdot S1 \cdot S0$$

$$Y = \overline{S1}(I0 \cdot \overline{S0} + I1 \cdot S0) + S1 \cdot (I2 \cdot \overline{S0} + I3 \cdot S0)$$

$$Y = Y0 + Y1$$

- Boolean expression for 4 × 2 encoder

```
A1 = Y3 + Y2
A0 = Y3 + Y1
```

- Boolean expression for 2 × 4 decoder

$$Y_3 = E.\,A_1.\,A_0$$

$$Y_2 = E.\,A_1.\,A_0'$$

$$Y_1 = E.\,A_1'.\,A_0$$

$$Y_0 = E.\,A_1'.\,A_0'$$

## 4.5 Conclusion

Learned how to make 2 bit comparators, 4:1 multiplexers using two 2:1 multiplexers and made encoders and decoders using logic gate ICs.

# 5 Lab-5 : Latches and Flip flops

## 5.1 Aim

- JK, SR, D latch using logic gate
- JK to SR flip flop conversion and vice versa
- JK to D and T flip conversion

## 5.2 Apparatus

| Sr No. | Components | Quantity |
|---|---|---|
| 1 | Digital Trainer Kit | 1 |
| 2 | NAND (7400) | 1 |
| 3 | NOR (7402) | 1 |
| 4 | NOT (7404) | 1 |
| 5 | AND (7408) | 1 |
| 6 | OR (7432) | 1 |
| 7 | XOR (7486) | 1 |
| 8 | Connecting Wires | As Required |
| 9 | Bread board | 1 |

## 5.3 Theory

### 5.3.1 Explanation

- Flip Flops are edge-triggered and a latch is level-triggered. For example, let us talk about SR latch and SR flip-flops. In this circuit when you Set S as active, the output Q will be high and Q' will be Low. This is irrespective of anything else. (This is an active-low circuit; so active here means low, but for an active high circuit, active would mean high). A flip-flop, on the other hand, is a synchronous Circuit and is also known as a gated or clocked SR latch.

- Due to the undefined state in the SR flip-flops, another flip-flop is required in electronics. The JK flip-flop is an improvement on the SR flip-flop where S=R=1 is not a problem.The input condition of J=K=1 gives an output inverting the output state. However, the outputs are the same when one tests the circuit practically. In simple words, If J and K data input are different (i.e. high and low), then the output Q takes the value of J at the next clock edge. If J and K are both low, then no change occurs. If J and K are both high at the clock edge, then the output will toggle from one state to the other. JK Flip-Flops can function as Set or Reset Flip-flops.

- This is the most common flip-flop among all. This simple flip-flop circuit has a set input (S) and a reset input (R). In this system, when you Set "S" as active, the output "Q" would be high, and "Q'" would be low. Once the outputs are established, the wiring of the circuit is maintained until "S" or "R" go high, or power is turned off.

- D flip-flop is a better alternative that is very popular with digital electronics. They are commonly used for counters and shift registers and input synchronization. In the D flip-flops, the output can only be changed at the clock edge, and if the input changes at other times, the output will be unaffected. The change of state of the output is dependent on the rising edge of the clock. The output (Q) is the same as the input and can only change at the rising edge of the clock.

- To convert SR to JK J and K will be given as external inputs to S and R. S and R will be the outputs of the combinational circuit. The present state is represented by Qp and Qp+1 is the next state to be obtained when the J and K inputs are applied. For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp.

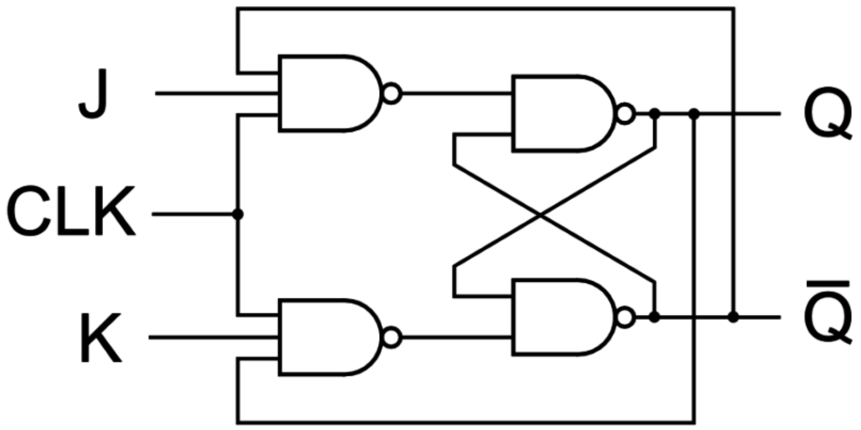
- To convert JK to SR S and R will be the external inputs to J and K. J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Qp. A conversion table is to be written using S, R, Qp, Qp+1, J and K. For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found ut. The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as "don't cares".

- To convert JK to D flip flop D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp.

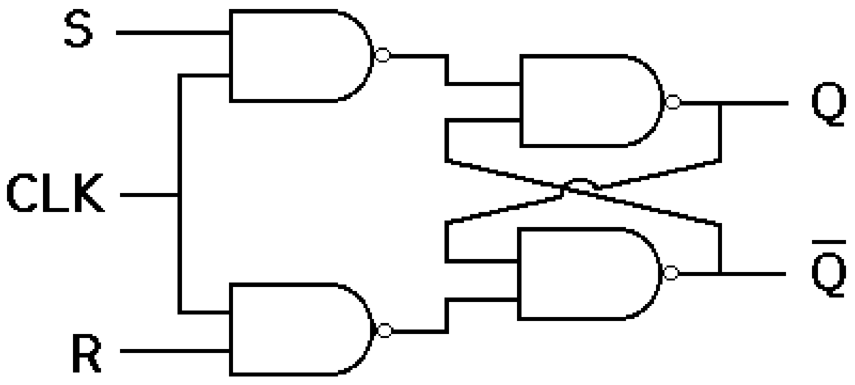
- To convert JK to T flip flop J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp.

### 5.3.2 Circuit diagrams and Truth Tables

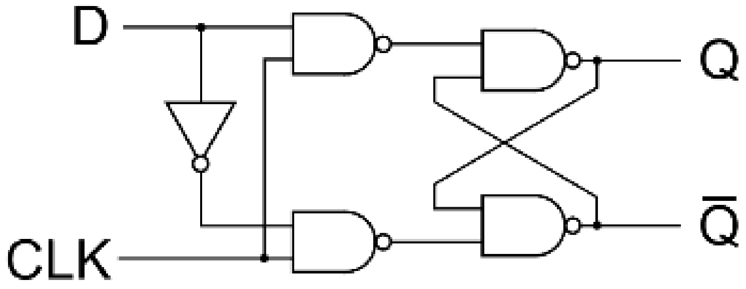
- Circuit Diagram for JK Flip Flop



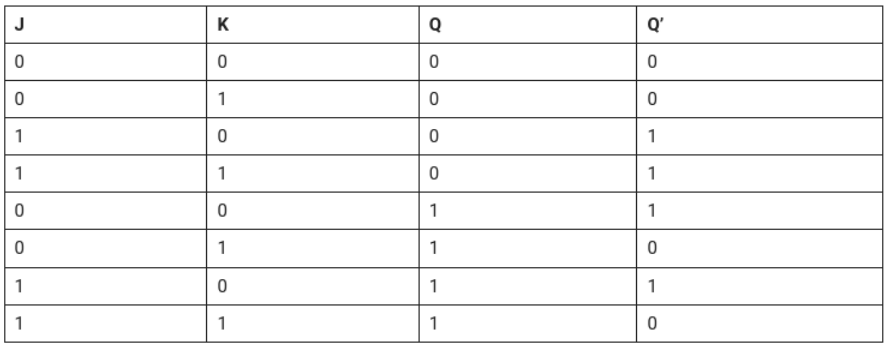
- Circuit Diagram for SR Flip Flop



- Circuit Diagram for D Flip Flop

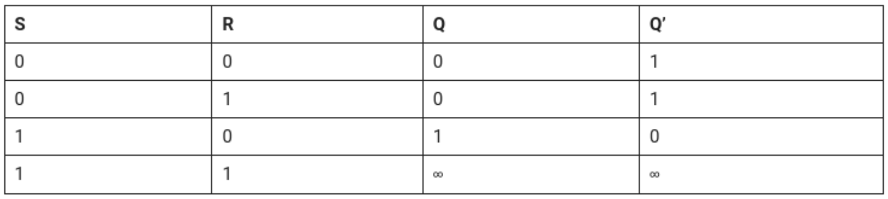


- Truth table for JK Flip Flop



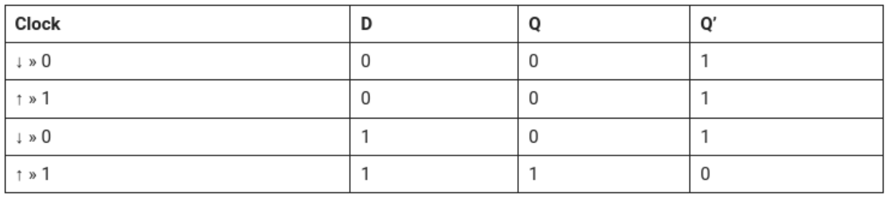
| **J** | **K** | **Q** | **Q’** |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |

- Truth table for SR Flip Flop



| **S** | **R** | **Q** | **Q’** |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | ∞ | ∞ |

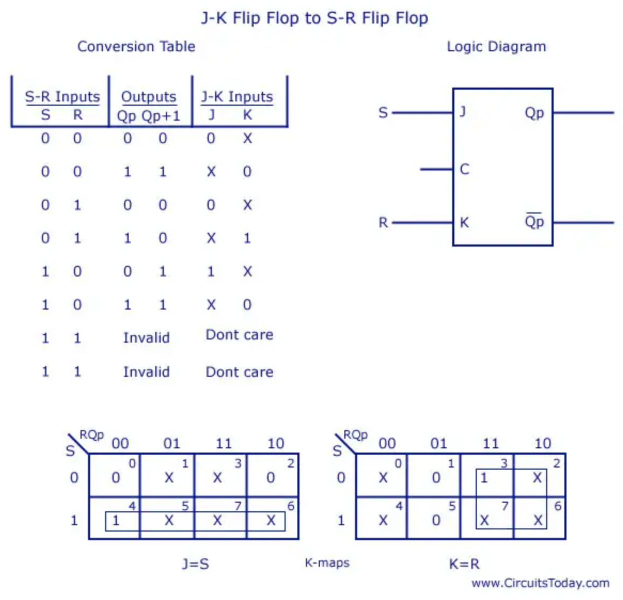
- Truth table for D Flip Flop



| **Clock** | **D** | **Q** | **Q’** |
|---|---|---|---|
| ↓ » 0 | 0 | 0 | 1 |
| ↑ » 1 | 0 | 0 | 1 |
| ↓ » 0 | 1 | 0 | 1 |
| ↑ » 1 | 1 | 1 | 0 |

### 5.3.3 Circuit diagrams and Truth Tables for conversions

- JK to SR conversion



| S-R Inputs | | Outputs | | J-K Inputs | |
|---|---|---|---|---|---|
| S | R | Qp | Qp+1 | J | K |
| 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 1 | 1 | X | 0 |
| 0 | 1 | 0 | 0 | 0 | X |
| 0 | 1 | 1 | 0 | X | 1 |
| 1 | 0 | 0 | 1 | 1 | X |
| 1 | 0 | 1 | 1 | X | 0 |
| 1 | 1 | Invalid | | Dont care | |
| 1 | 1 | Invalid | | Dont care | |

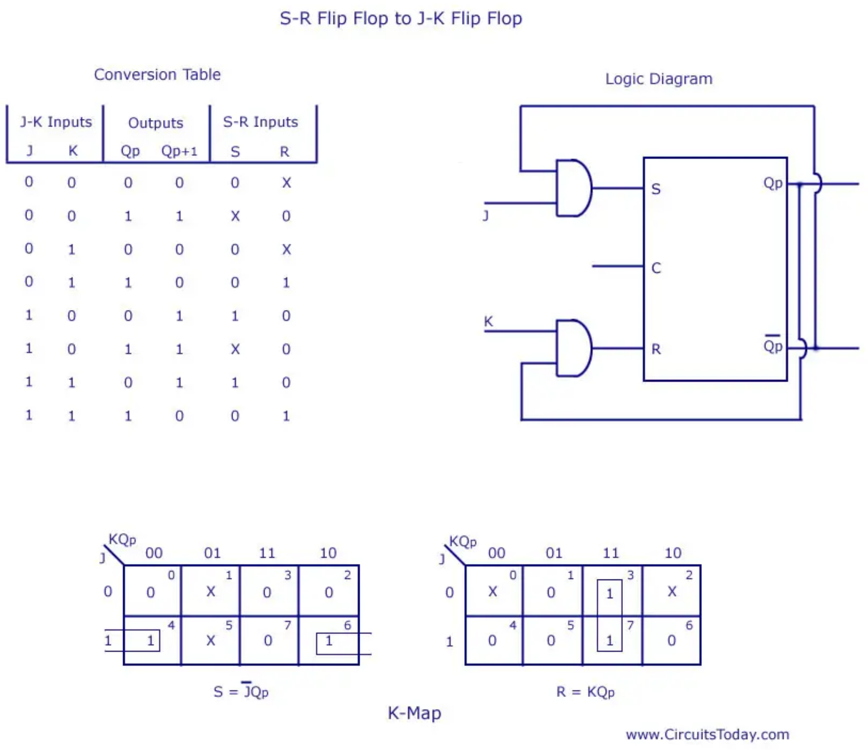
J=S

| S \ RQp | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | X | X | 0 |
| 1 | 1 | X | X | X |

K=R

| S \ RQp | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | 0 | 1 | X |
| 1 | X | 0 | X | X |

- SR to JK conversion



| J-K Inputs | | Outputs | | S-R Inputs | |
|---|---|---|---|---|---|
| J | K | Qp | Qp+1 | S | R |
| 0 | 0 | 0 | 0 | 0 | X |
| 0 | 0 | 1 | 1 | X | 0 |
| 0 | 1 | 0 | 0 | 0 | X |
| 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | X | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 |

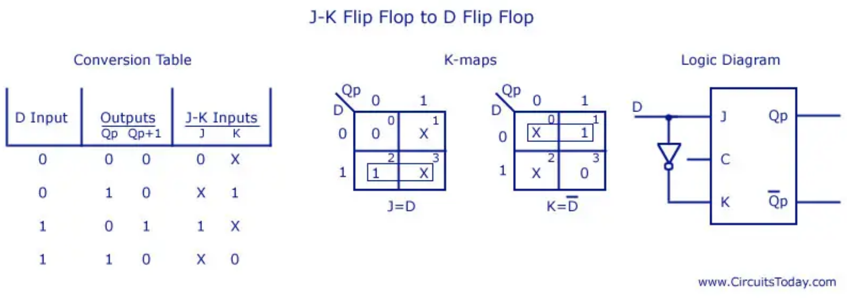
$S = J\overline{Q}p$

| J \ KQp | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | X | 0 | 0 |
| 1 | 1 | X | 0 | 1 |

R = KQp

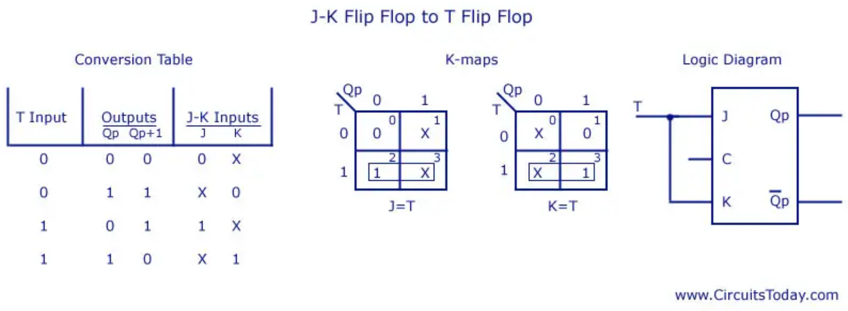
| J \ KQp | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | 0 | 1 | X |
| 1 | 0 | 0 | 1 | 0 |

- JK to D conversion



| D Input | Outputs Qp | Outputs Qp+1 | J-K Inputs J | J-K Inputs K |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X |
| 0 | 1 | 0 | X | 1 |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | X | 0 |

- JK to T conversion



| T Input | Outputs Qp | Outputs Qp+1 | J-K Inputs J | J-K Inputs K |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | X |
| 0 | 1 | 1 | X | 0 |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | X | 1 |

## 5.4 Conclusion

- Learned how to make JK, SR and D flip flop using logic gates.
- Learned how to convert JK to SR and SR to JK flip flop.
- Learned how to convert JK to D and JK to T flip flop.

# 6 Lab-5 : Latches and Flip flops

## 6.1 Aim

Write a program to generate the output of the following waveform

- An active HIGH S-R latch
- Negative edge triggered S-R flip flop with active HIGH PRESET and CLEAR.
- Positive edge-triggered J-K FF with active LOW PRESET and CLEAR.

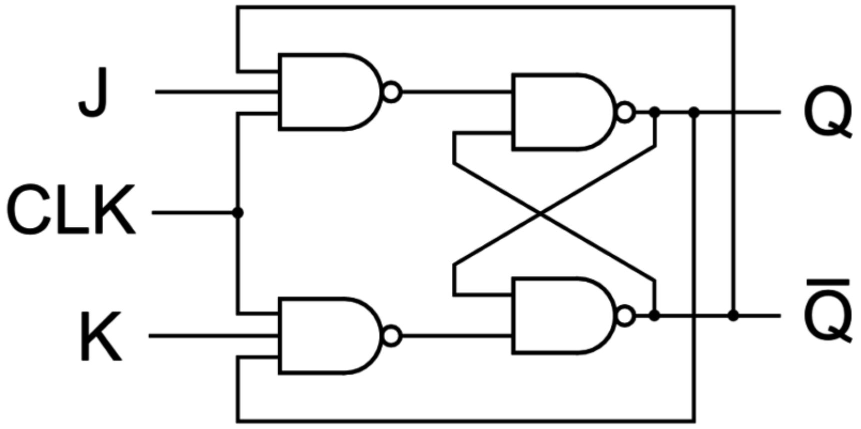
## 6.2 Apparatus

| Sr No. | Components | Quantity |
|---|---|---|
| 1 | Integrated Development Environment (VS Code) | 1 |
| 1 | Python compiler | 1 |
| 1 | Matplotlib | 1 |

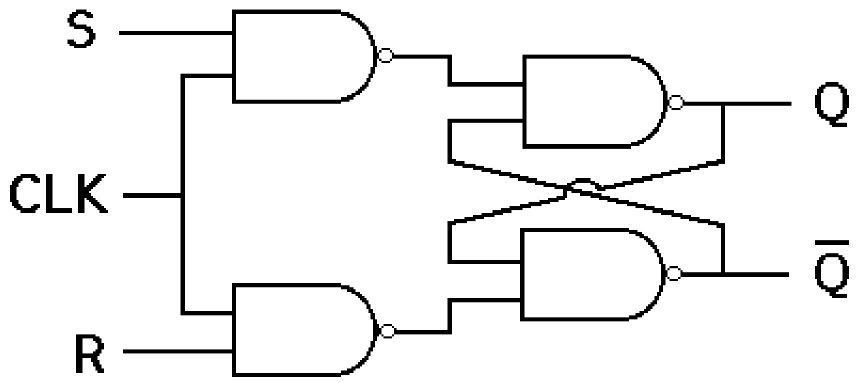
## 6.3 Theory

### 6.3.1 Circuit diagrams and Truth Tables

- Circuit Diagram for JK Flip Flop



- Circuit Diagram for SR Flip Flop



- Truth table for JK Flip Flop

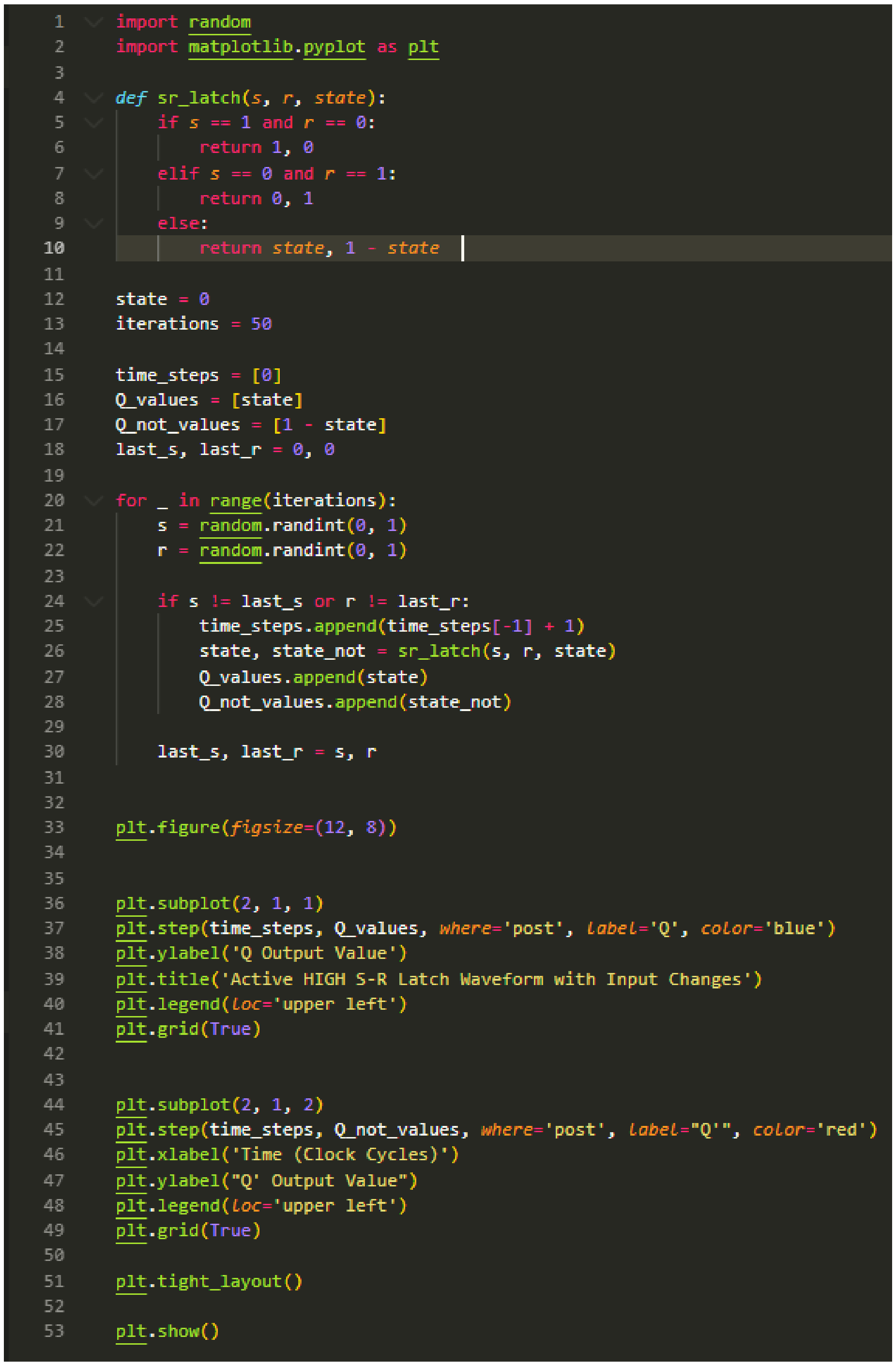
| J | K | Q | Q' |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 |

- Truth table for SR Flip Flop

| S | R | Q | Q' |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | ∞ | ∞ |

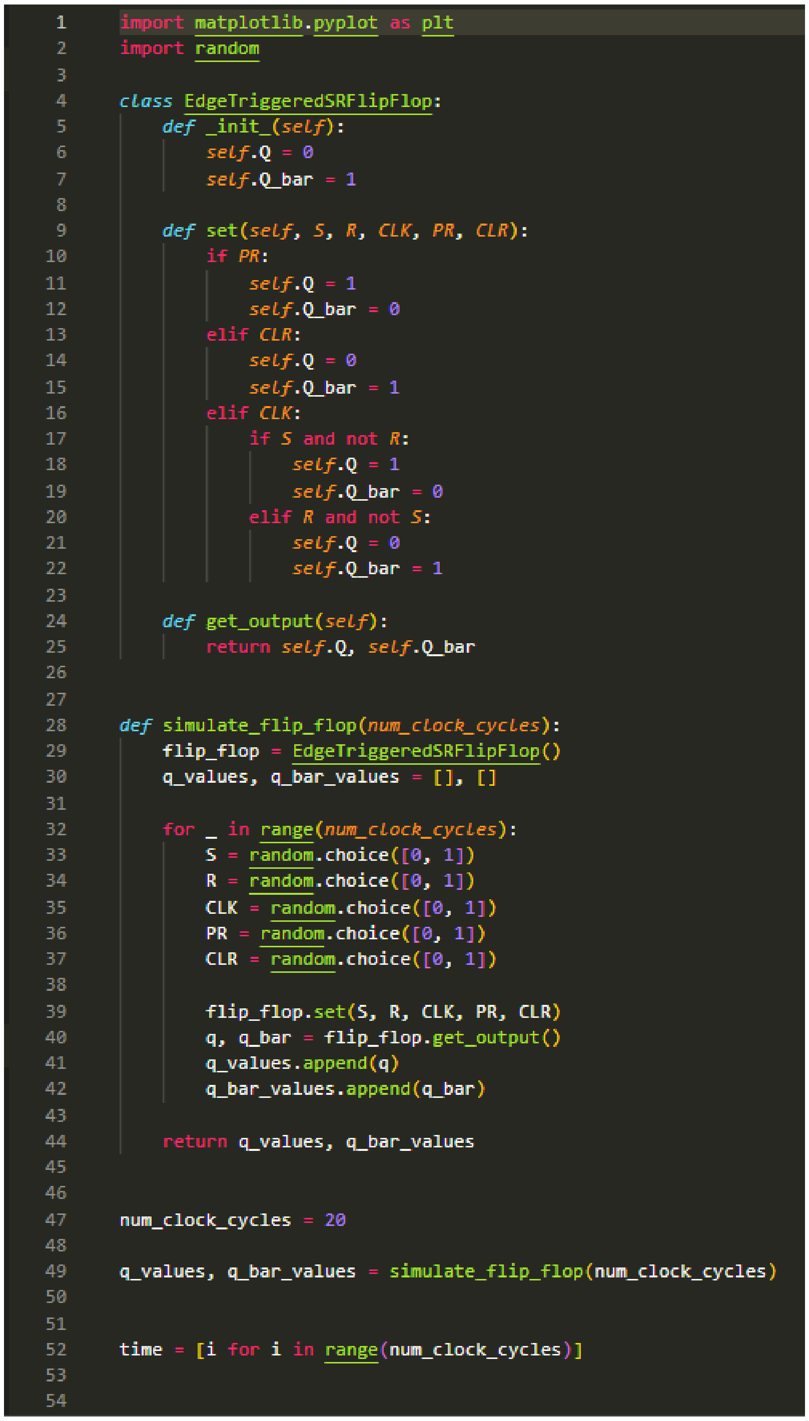
### 6.3.2 Code

- An active HIGH S-R latch

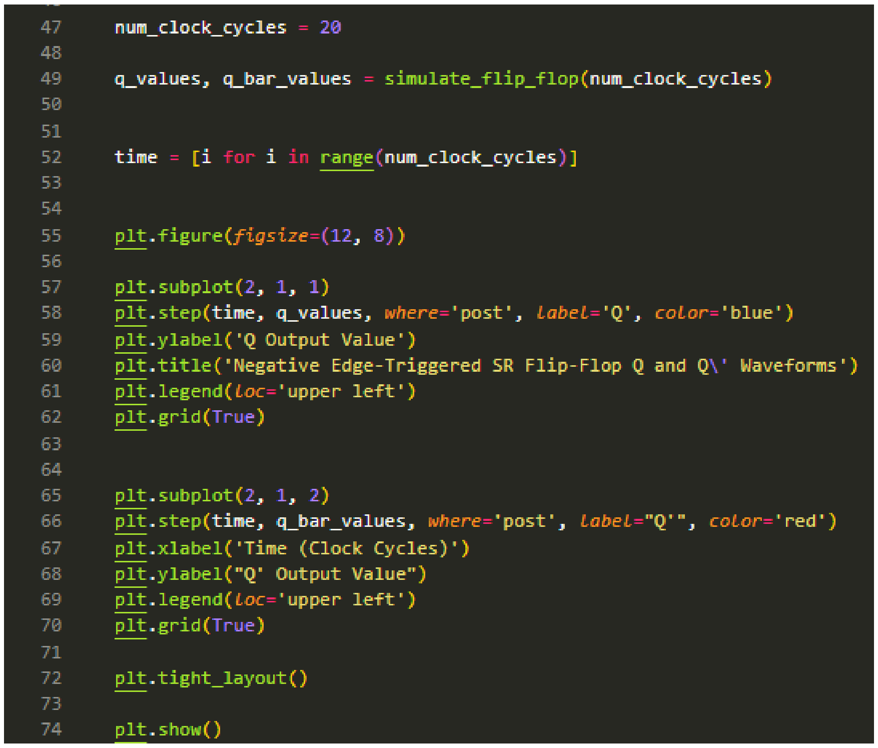


```python
import random
import matplotlib.pyplot as plt

def sr_latch(s, r, state):
    if s == 1 and r == 0:
        return 1, 0
    elif s == 0 and r == 1:
        return 0, 1
    else:
        return state, 1 - state

state = 0
iterations = 50

time_steps = [0]
Q_values = [state]
Q_not_values = [1 - state]
last_s, last_r = 0, 0

for _ in range(iterations):
    s = random.randint(0, 1)
    r = random.randint(0, 1)

    if s != last_s or r != last_r:
        time_steps.append(time_steps[-1] + 1)
        state, state_not = sr_latch(s, r, state)
        Q_values.append(state)
        Q_not_values.append(state_not)

    last_s, last_r = s, r


plt.figure(figsize=(12, 8))


plt.subplot(2, 1, 1)
plt.step(time_steps, Q_values, where='post', label='Q', color='blue')
plt.ylabel('Q Output Value')
plt.title('Active HIGH S-R Latch Waveform with Input Changes')
plt.legend(loc='upper left')
plt.grid(True)


plt.subplot(2, 1, 2)
plt.step(time_steps, Q_not_values, where='post', label="Q'", color='red')
plt.xlabel('Time (Clock Cycles)')
plt.ylabel("Q' Output Value")
plt.legend(loc='upper left')
plt.grid(True)

plt.tight_layout()

plt.show()
```

- Negative edge triggered S-R flip flop with active HIGH PRESET and CLEAR.

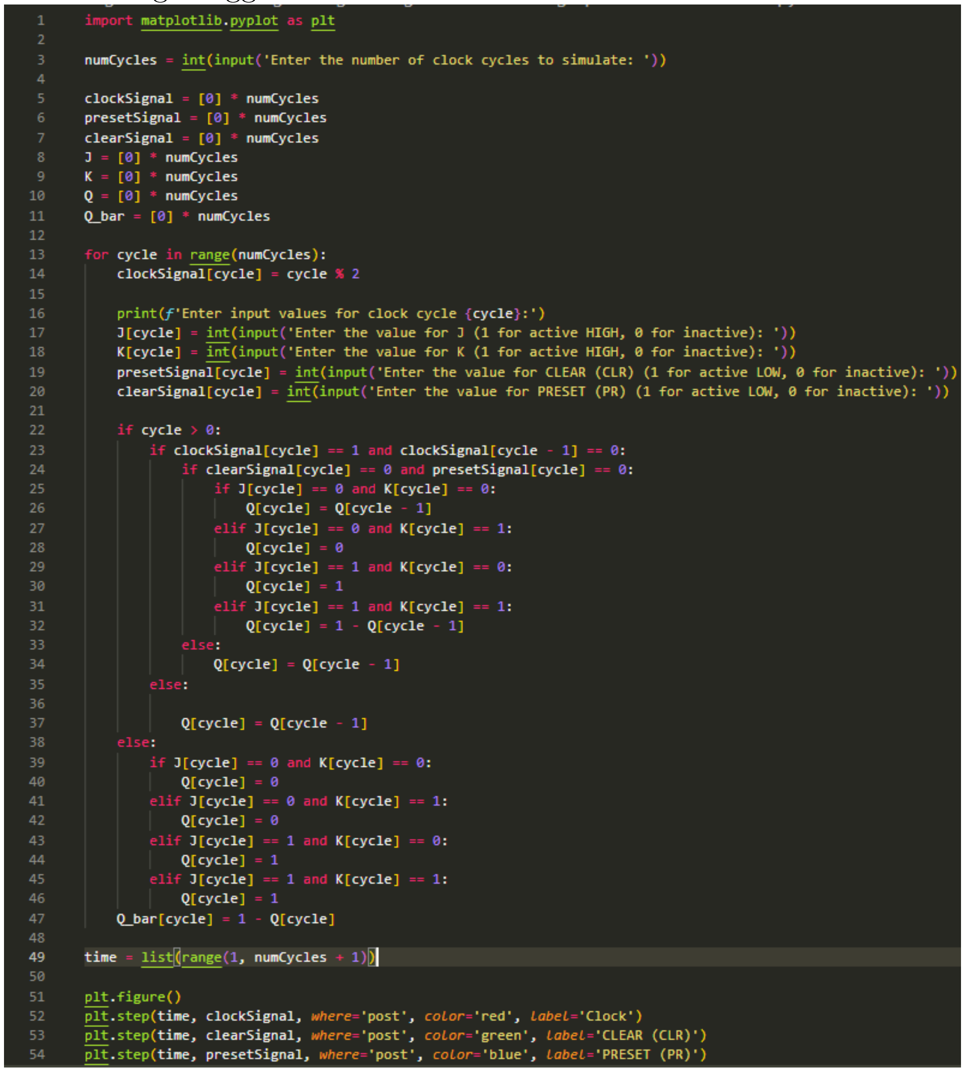


```python
import matplotlib.pyplot as plt
import random

class EdgeTriggeredSRFlipFlop:
    def _init_(self):
        self.Q = 0
        self.Q_bar = 1

    def set(self, S, R, CLK, PR, CLR):
        if PR:
            self.Q = 1
            self.Q_bar = 0
        elif CLR:
            self.Q = 0
            self.Q_bar = 1
        elif CLK:
            if S and not R:
                self.Q = 1
                self.Q_bar = 0
            elif R and not S:
                self.Q = 0
                self.Q_bar = 1

    def get_output(self):
        return self.Q, self.Q_bar


def simulate_flip_flop(num_clock_cycles):
    flip_flop = EdgeTriggeredSRFlipFlop()
    q_values, q_bar_values = [], []

    for _ in range(num_clock_cycles):
        S = random.choice([0, 1])
        R = random.choice([0, 1])
        CLK = random.choice([0, 1])
        PR = random.choice([0, 1])
        CLR = random.choice([0, 1])

        flip_flop.set(S, R, CLK, PR, CLR)
        q, q_bar = flip_flop.get_output()
        q_values.append(q)
        q_bar_values.append(q_bar)

    return q_values, q_bar_values


num_clock_cycles = 20

q_values, q_bar_values = simulate_flip_flop(num_clock_cycles)


time = [i for i in range(num_clock_cycles)]


```

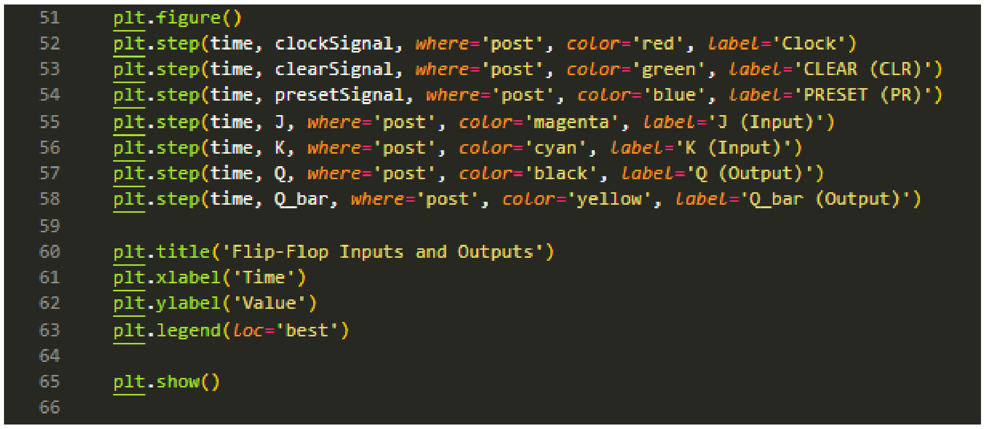


```python
num_clock_cycles = 20

q_values, q_bar_values = simulate_flip_flop(num_clock_cycles)


time = [i for i in range(num_clock_cycles)]


plt.figure(figsize=(12, 8))

plt.subplot(2, 1, 1)
plt.step(time, q_values, where='post', label='Q', color='blue')
plt.ylabel('Q Output Value')
plt.title('Negative Edge-Triggered SR Flip-Flop Q and Q\' Waveforms')
plt.legend(loc='upper left')
plt.grid(True)


plt.subplot(2, 1, 2)
plt.step(time, q_bar_values, where='post', label="Q'", color='red')
plt.xlabel('Time (Clock Cycles)')
plt.ylabel("Q' Output Value")
plt.legend(loc='upper left')
plt.grid(True)

plt.tight_layout()

plt.show()
```

- Positive edge-triggered J-K FF with active LOW PRESET and CLEAR.



```python
import matplotlib.pyplot as plt

numCycles = int(input('Enter the number of clock cycles to simulate: '))

clockSignal = [0] * numCycles
presetSignal = [0] * numCycles
clearSignal = [0] * numCycles
J = [0] * numCycles
K = [0] * numCycles
Q = [0] * numCycles
Q_bar = [0] * numCycles

for cycle in range(numCycles):
    clockSignal[cycle] = cycle % 2

    print(f'Enter input values for clock cycle {cycle}:')
    J[cycle] = int(input('Enter the value for J (1 for active HIGH, 0 for inactive): '))
    K[cycle] = int(input('Enter the value for K (1 for active HIGH, 0 for inactive): '))
    presetSignal[cycle] = int(input('Enter the value for CLEAR (CLR) (1 for active LOW, 0 for inactive): '))
    clearSignal[cycle] = int(input('Enter the value for PRESET (PR) (1 for active LOW, 0 for inactive): '))

    if cycle > 0:
        if clockSignal[cycle] == 1 and clockSignal[cycle - 1] == 0:
            if clearSignal[cycle] == 0 and presetSignal[cycle] == 0:
                if J[cycle] == 0 and K[cycle] == 0:
                    Q[cycle] = Q[cycle - 1]
                elif J[cycle] == 0 and K[cycle] == 1:
                    Q[cycle] = 0
                elif J[cycle] == 1 and K[cycle] == 0:
                    Q[cycle] = 1
                elif J[cycle] == 1 and K[cycle] == 1:
                    Q[cycle] = 1 - Q[cycle - 1]
            else:
                Q[cycle] = Q[cycle - 1]
        else:

            Q[cycle] = Q[cycle - 1]
    else:
        if J[cycle] == 0 and K[cycle] == 0:
            Q[cycle] = 0
        elif J[cycle] == 0 and K[cycle] == 1:
            Q[cycle] = 0
        elif J[cycle] == 1 and K[cycle] == 0:
            Q[cycle] = 1
        elif J[cycle] == 1 and K[cycle] == 1:
            Q[cycle] = 1
    Q_bar[cycle] = 1 - Q[cycle]

time = list(range(1, numCycles + 1))

plt.figure()
plt.step(time, clockSignal, where='post', color='red', label='Clock')
plt.step(time, clearSignal, where='post', color='green', label='CLEAR (CLR)')
plt.step(time, presetSignal, where='post', color='blue', label='PRESET (PR)')
```

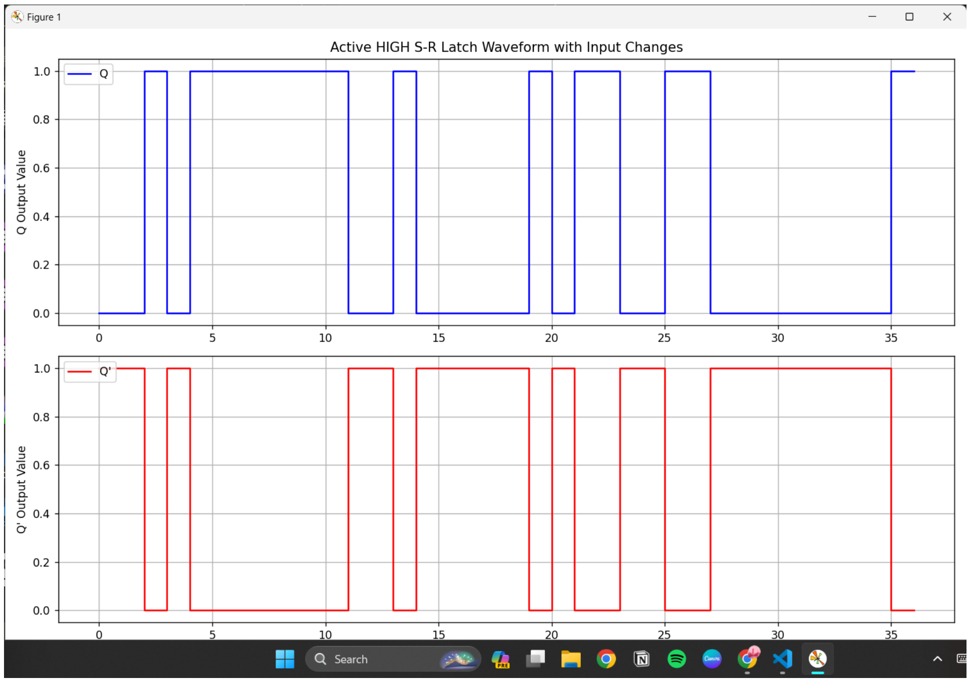


```python
plt.figure()
plt.step(time, clockSignal, where='post', color='red', label='Clock')
plt.step(time, clearSignal, where='post', color='green', label='CLEAR (CLR)')
plt.step(time, presetSignal, where='post', color='blue', label='PRESET (PR)')
```

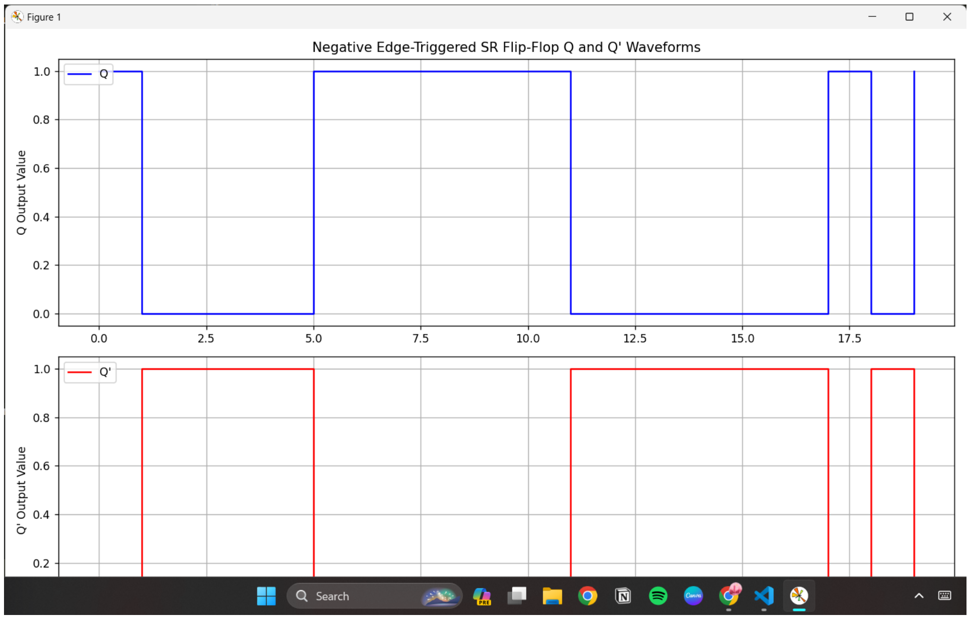
```python
plt.step(time, J, where='post', color='magenta', label='J (Input)')
plt.step(time, K, where='post', color='cyan', label='K (Input)')
plt.step(time, Q, where='post', color='black', label='Q (Output)')
plt.step(time, Q_bar, where='post', color='yellow', label='Q_bar (Output)')

plt.title('Flip-Flop Inputs and Outputs')
plt.xlabel('Time')
plt.ylabel('Value')
plt.legend(loc='best')

plt.show()
```

## 6.4 Observations

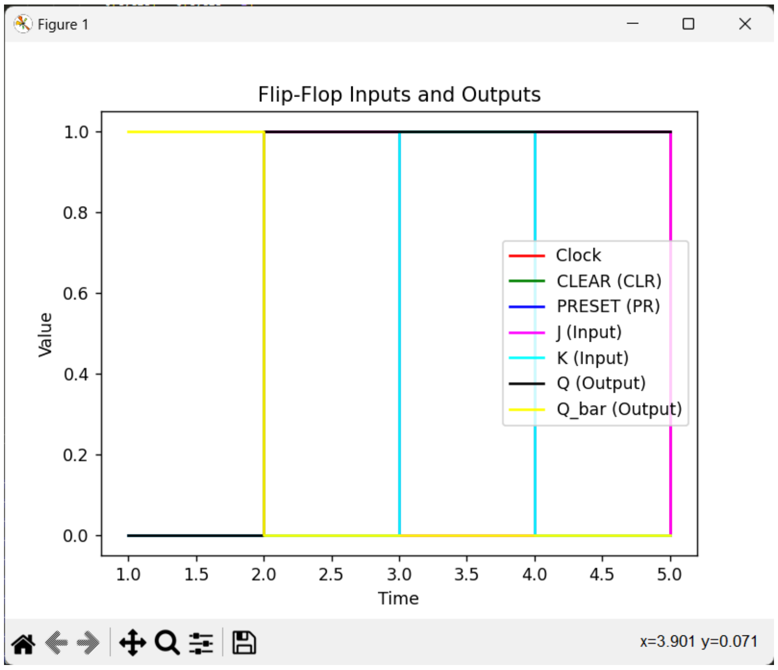
- An active HIGH S-R latch



- Negative edge triggered S-R flip flop with active HIGH PRESET and CLEAR.



- Positive edge-triggered J-K FF with active LOW PRESET and CLEAR.



## 6.5 Conclusion

- Learned how to generate waveform for active HIGH S-R latch.
- Learned how to generate waveform for negative edge triggered S-R flip flop with active HIGH PRESET and CLEAR.
- Learned how to generate waveform for positive edge-triggered J-K FF with active LOW PRESET and CLEAR.