

INDIRA GANDHI DELHI TECHNICAL UNIVERSITY FOR WOMEN

Computer organization

And

Architecture

(b2)

# Submitted by: Submitted to:

Name: Meghavi Tomar Prof. SRNReddySir

EnrollmentNo:18601012020 CSEDepartment

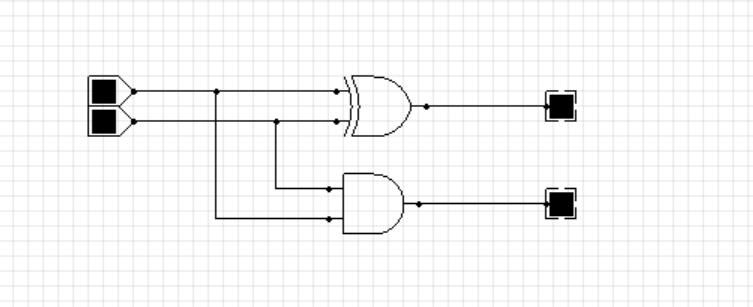
Batch:CSE-2, B2 IGDTUW

B.Tech, 4thSemester

**CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.No.** | **Name of the Experiment** | **Date** |
| 1. | Design a half adder and verify its truth table. | 04-02-22 |
| 2. | Design a full adder and verify its truth table. | 04-02-22 |
| 3. | Design a 4-Bit Parallel adder circuit and verify various cases. | 08-02-22 |
| 4. | Design and implement a) SR Flip Flop b) D Flip Flop c) JK Flip Flop d) T Flip Flop. Also explain the truth tables along with wave forms. | 25-02-22 |
| 5. | Design a 4-Bit Shift register using D Flip Flops and explain its truth table. | 25-02-22 |
| 6. | To identify high level language code, corresponding assembly language code, instructions passed and its characteristics. | 04-03-22 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# EXPERIMENT - 1



**AIM**: Design a half adder circuit and verify its truth table.

**Hardware Requirement:** Laptop

(Inspiron14 5408)

[Device Name: DESKTOP-JPH3QS6]

**Software Requirement:** a. Windows 10 (Operating System)

b. CEDAR logic (simulator)

## Procedure:

1. Take a XOR and AND logic gates.
2. Take 2 input devices.
3. Connect both the inputs to the XOR and AND logic gates.
4. Take 2 output devices.
5. Connect 1 output device with XOR and 1 with AND logic gaterespectively.
6. Half adder circuit is ready.
7. Sum = A’B + B’A and Carry\_out = AB

## Circuit Diagram:

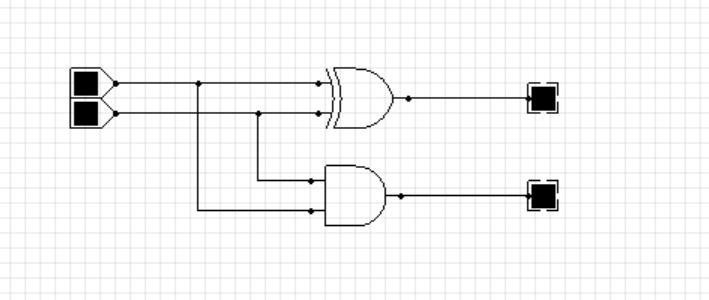
**Half adder Circuit [ black =’0’ ; red=’1’ ]**

## Truth Table:

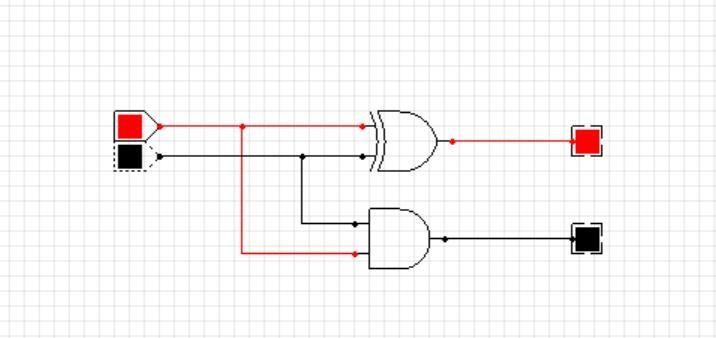
|  |  |  |  |
| --- | --- | --- | --- |
| A (input 1) | B (input 2) | Su m | Carry\_O ut |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**Observations:**

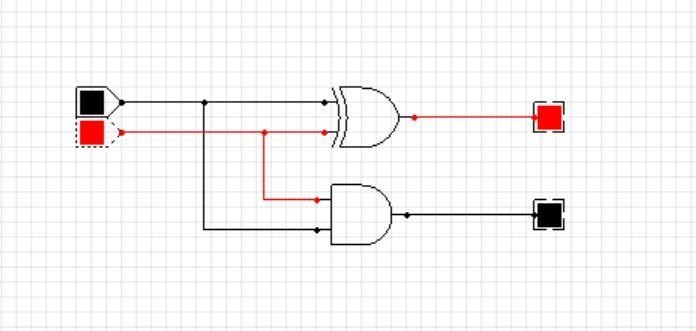
1. A = 0 and B = 0



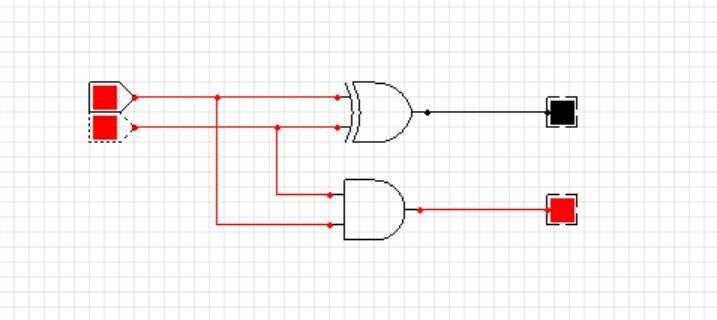
1. A = 1 and B = 0



1. A = 0 and B = 1



1. A = 1 and B = 1



## Conclusion:

The half adder circuit has been implemented and its truth table is verified.

# EXPERIMENT - 2

**AIM**: Design a full adder circuit and verify its truth table.

**Hardware Requirement:** Laptop

(Inspiron14 5408)

[Device Name: DESKTOP-JPH3QS6]

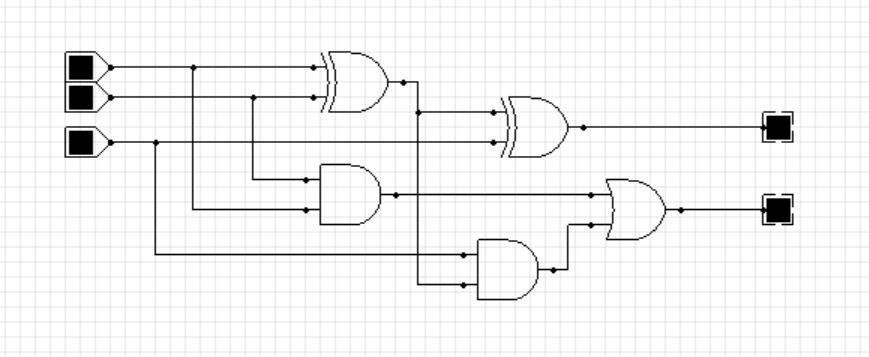
**Software Requirement:** a. Windows 10 (Operating System)

b. CEDAR logic (simulator)

## Procedure:

* 1. Take a 2 XOR, 2 AND and 1 OR logic gates.
  2. Take 3 input devices.
  3. Connect 2 inputs to one pair of XOR and AND logicgates.
  4. Connect the output of 1st XOR gate and 3rd input to another pair of XOR and AND gates.
  5. Connect the outputs of both the AND gates to the OR gate./
  6. Take 2 output devices.
  7. Connect output devices with 2nd XOR and OR logic gaterespectively.
  8. Full adder circuit is ready.
  9. Sum = (A’B + B’A)’Cin + Cin’(A’B + B’A) and Carry\_out = AB + Cin(A’B +B’A)

## Circuit Diagram:



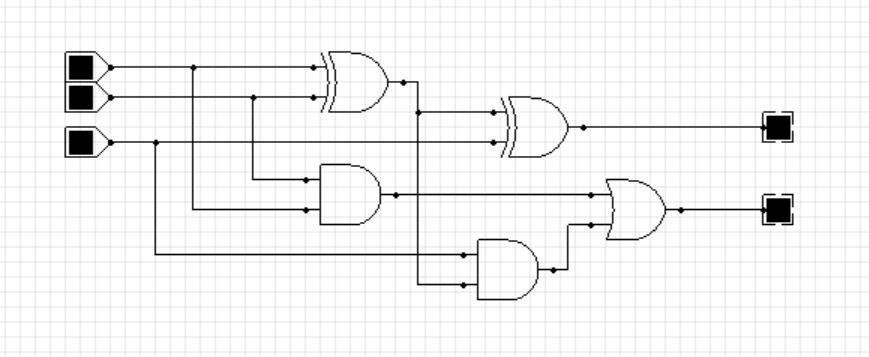
**Full adder Circuit [ black =’0’ ; red=’1’ ]**

## Truth Table:

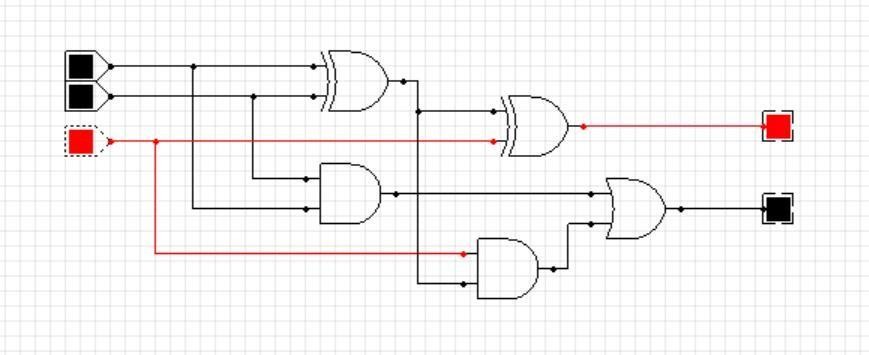
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A (input 1) | B (input 2) | Cin | Sum | Cout |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

**Observations:**

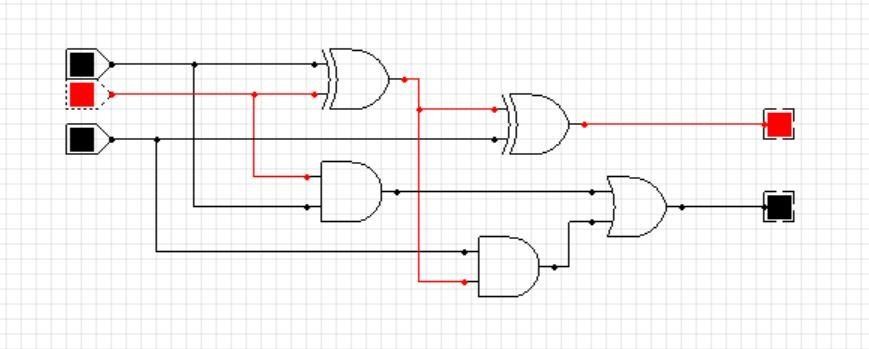
1. A = 0 and B = 0 and Cin = 0



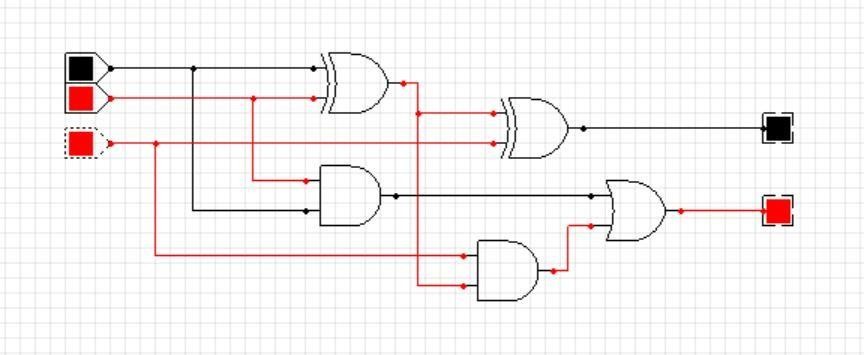
1. A = 0 and B = 0 and Cin = 1



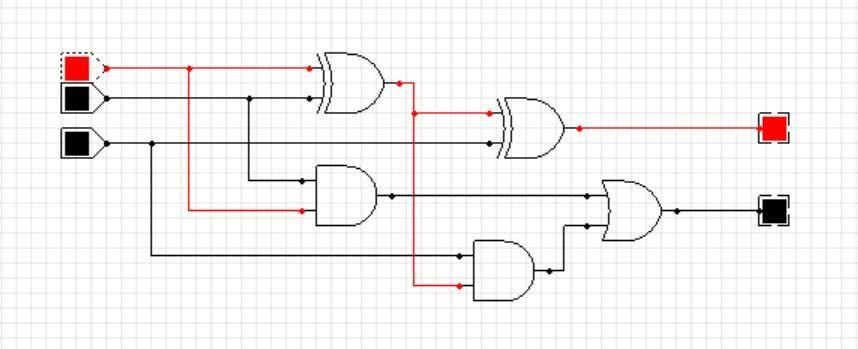
1. A = 0 and B = 1 and Cin = 0



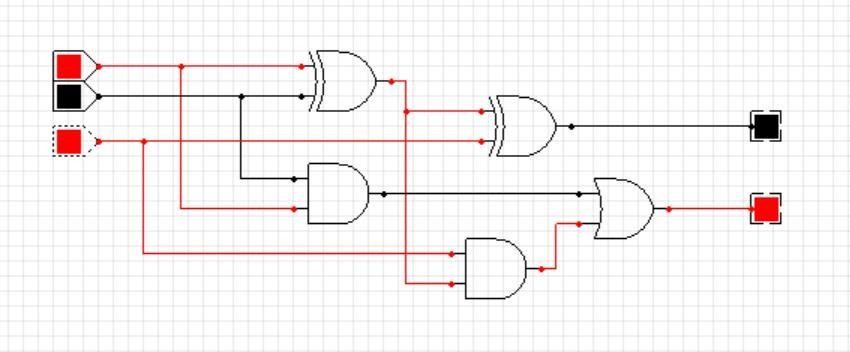
1. A = 0 and B = 1 and Cin = 1



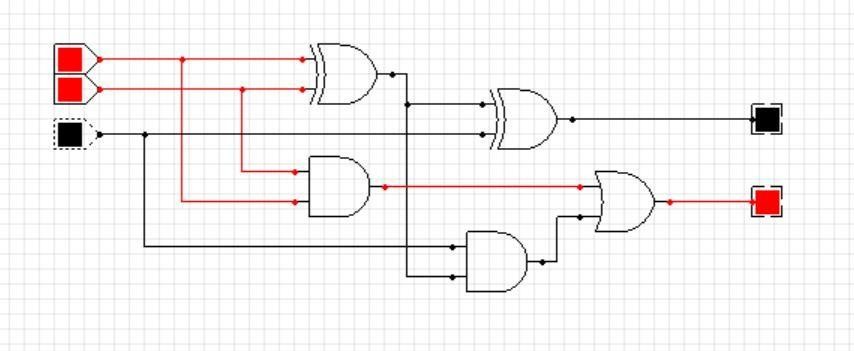
1. A = 1 and B = 0 and Cin = 0



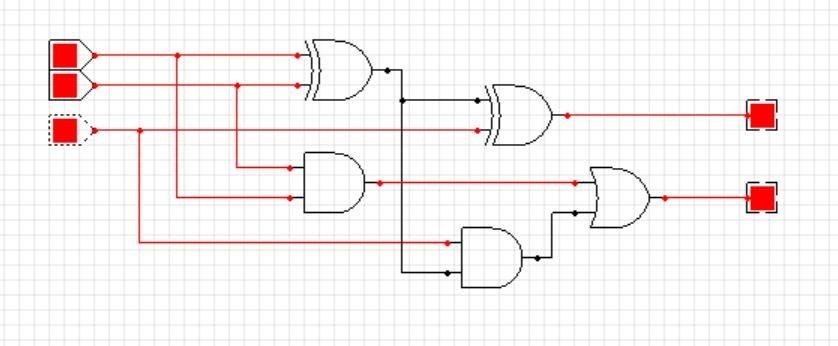
1. A = 1 and B = 0 and Cin = 1



1. A = 1 and B = 1 and Cin = 0

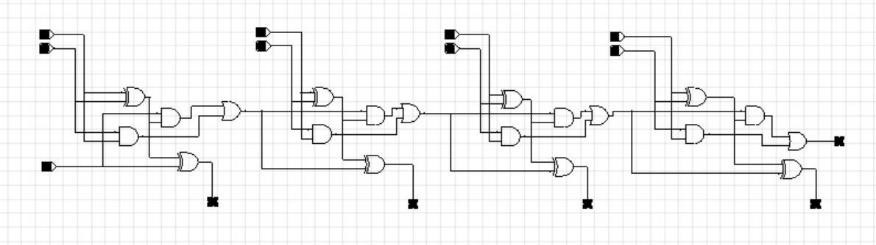


1. A = 1 and B = 1 and Cin = 1



## Conclusion:

The full adder circuit has been implemented and its truth table is verified.



# EXPERIMENT - 3

**AIM**: Design a 4 Bit Parallel adder circuit and verify its truth table.

**Hardware Requirement:** Laptop

(Inspiron14 5408)

[Device Name: DESKTOP-JPH3QS6]

**Software Requirement:** a. Windows 10 (Operating System)

b. CEDAR logic (simulator)

## Procedure:

* 1. Take 4 full adder circuits namely A, B, C, D.
  2. Replace the carry-in inputs of 3 circuits (B, C, D) by the carry-out of 3 circuits (A, B, C) respectively.
  3. 4-Bit Parallel adder is completed.
  4. Sum (Sn) = (An’Bn + Bn’An)’Cni + Cni’(An’Bn + Bn’An)
  5. Carry\_Out (Coi) = AnBn + Cni(An’Bn + Bn’An)

## Circuit Diagram:

**4-Bit Parallel Full adder Circuit [ black =’0’ ; red=’1’ ]**

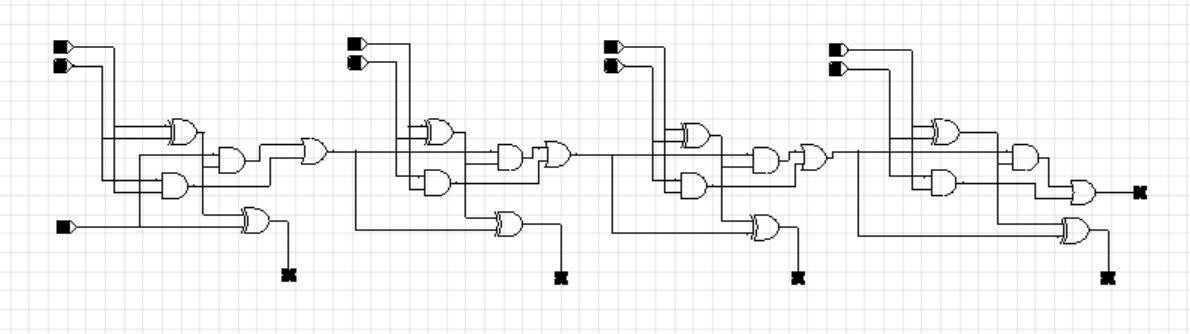
## Truth Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A  (A3A2A1A0) | B  (B3B2B1B0) | Cin | S  (S3S2S1S0) | Cout |
| 0000 | 0000 | 0 | 0000 | 0 |
| 0001 | 0001 | 0 | 0010 | 0 |
| 0010 | 0010 | 0 | 0100 | 0 |
| 0011 | 0011 | 0 | 0110 | 0 |
| 0100 | 0100 | 0 | 1000 | 0 |
| 0101 | 0101 | 0 | 1010 | 0 |
| 0110 | 0110 | 0 | 1100 | 0 |
| 0111 | 0111 | 0 | 1110 | 0 |
| 1000 | 1000 | 0 | 0000 | 1 |
| 1001 | 1001 | 0 | 0010 | 1 |
| 1010 | 1010 | 0 | 0100 | 1 |
| 1011 | 1011 | 0 | 0110 | 1 |
| 1100 | 1100 | 0 | 1000 | 1 |
| 1101 | 1101 | 0 | 1010 | 1 |
| 1110 | 1110 | 0 | 1100 | 1 |
| 1111 | 1111 | 0 | 1110 | 1 |
| 1111 | 1111 | 1 | 1111 | 1 |

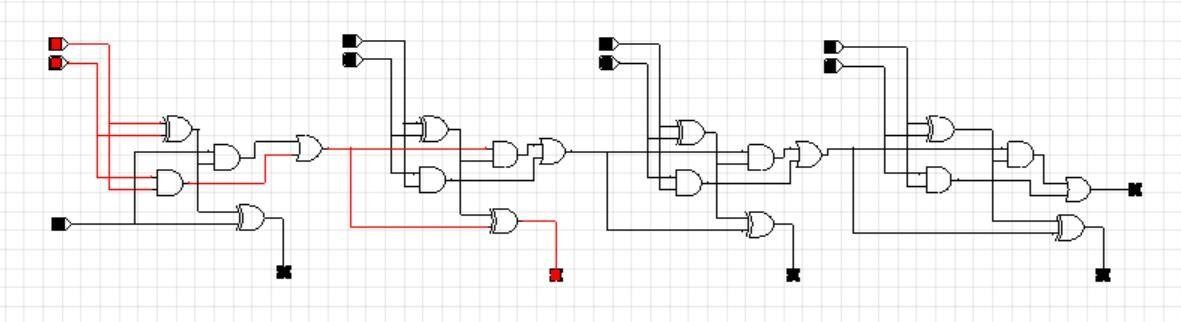
**––**

## Observations:

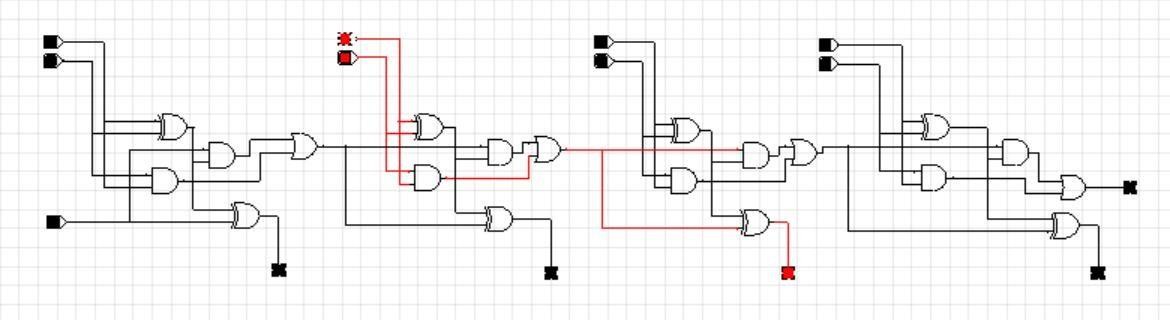
**1.** A = 0000 and B = 0000 and Cin = 0



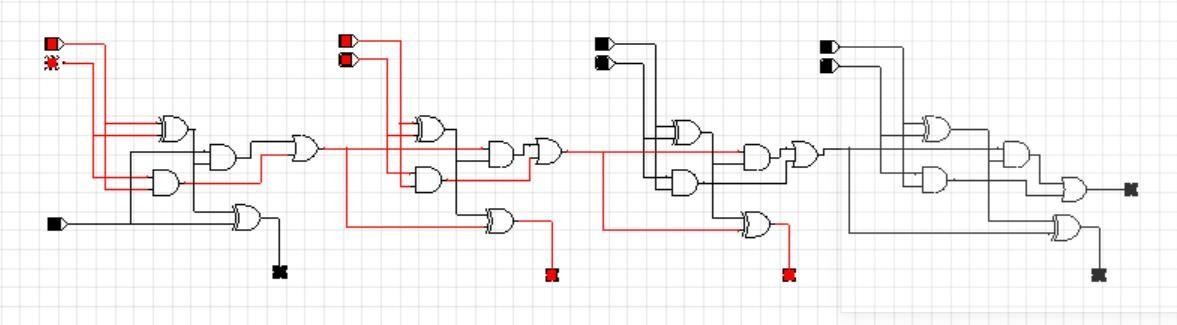
**2.** A = 0001 and B = 0001 and Cin = 0



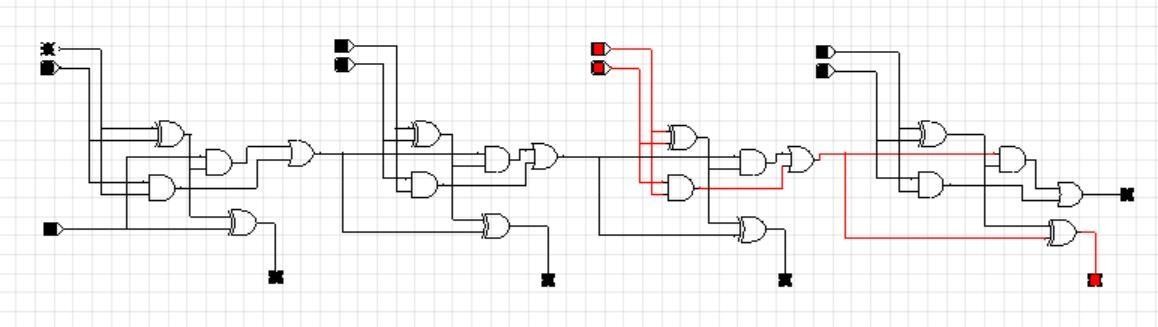
**3.** A = 0010 and B = 0010 and Cin = 0



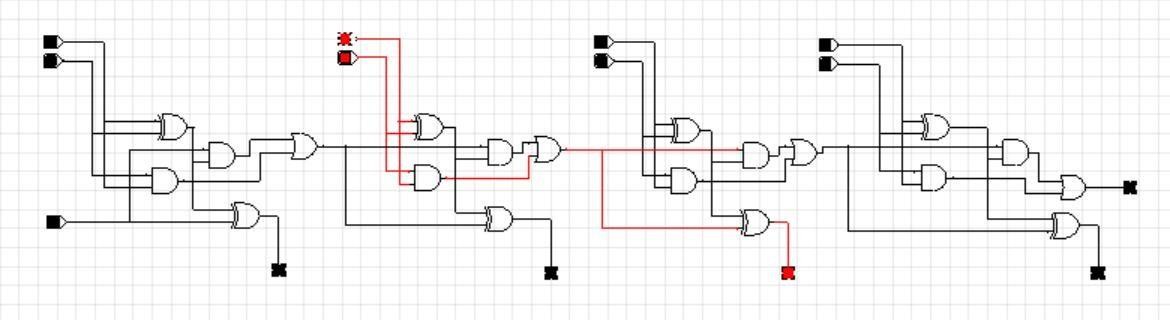
**4.** A = 0011 and B = 0011 and Cin = 0



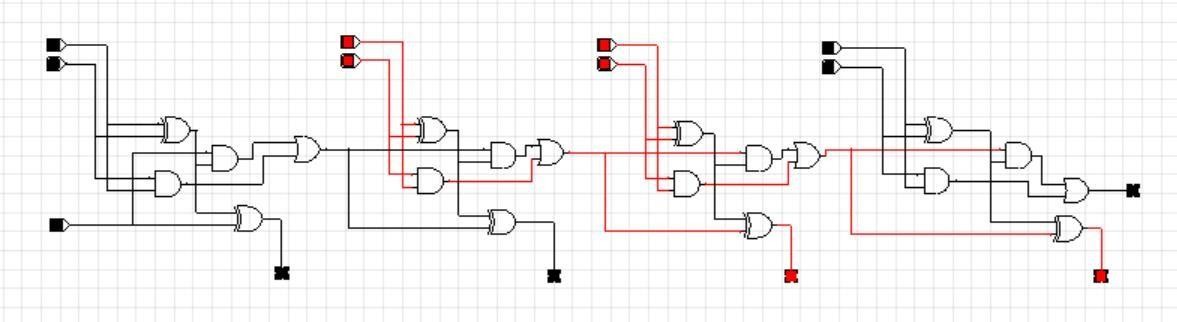
**5.** A = 0100 and B = 0100 and Cin = 0



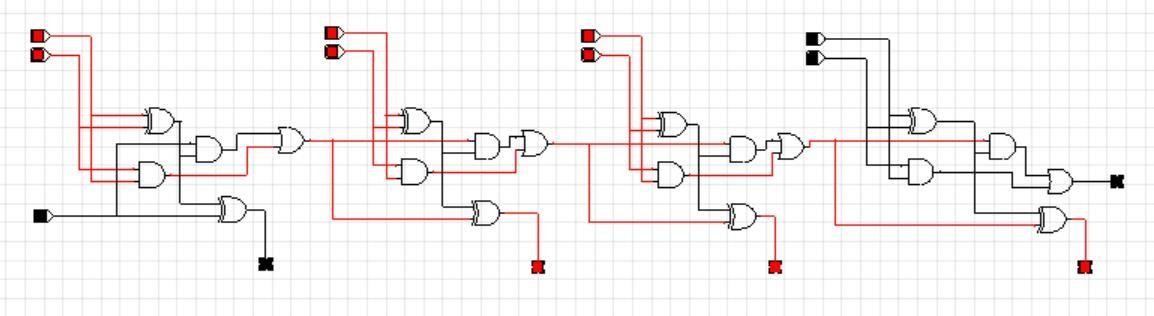
**6.** A = 0101 and B = 0101 and Cin = 0



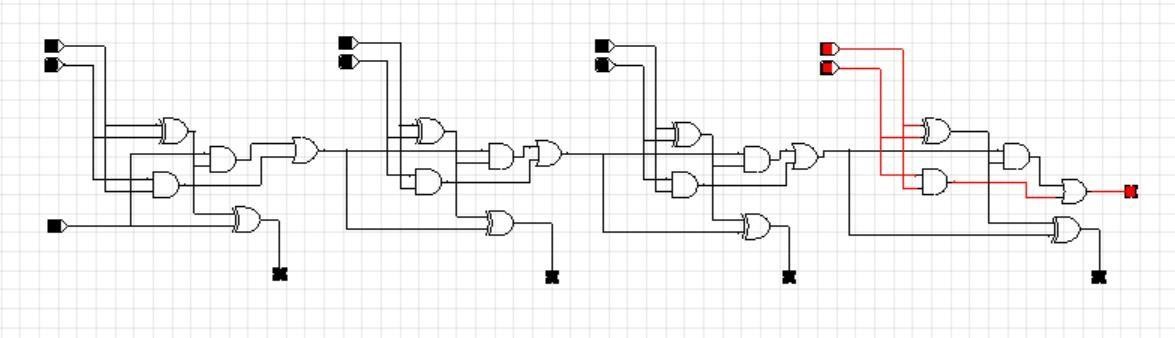
**7.** A = 0110 and B = 0110 and Cin = 0



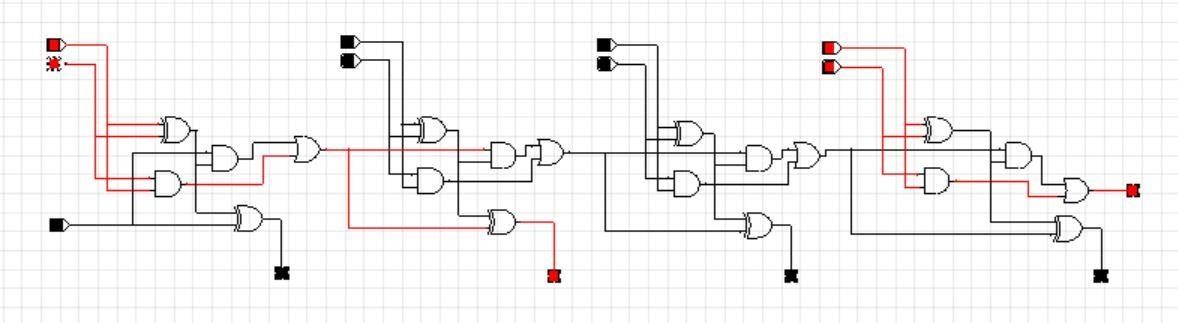
**8.** A = 0111 and B = 0111 and Cin = 0



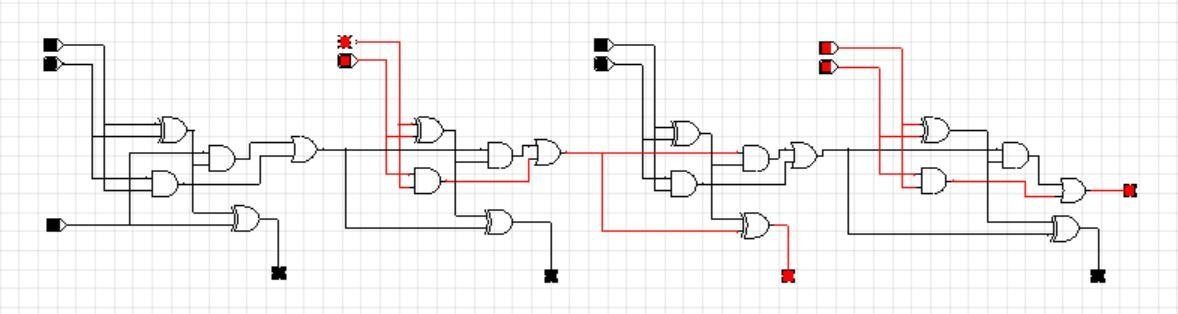
**9.** A = 1000 and B = 1000 and Cin = 0



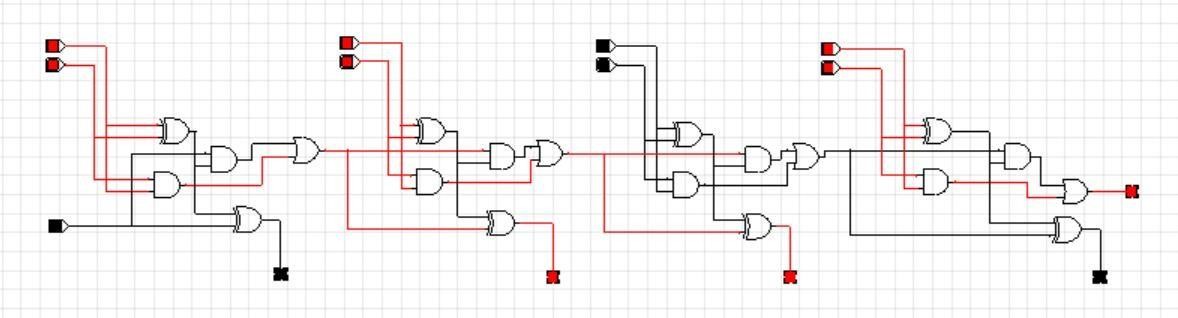
**10.** A = 1001 and B = 1001 and Cin = 0



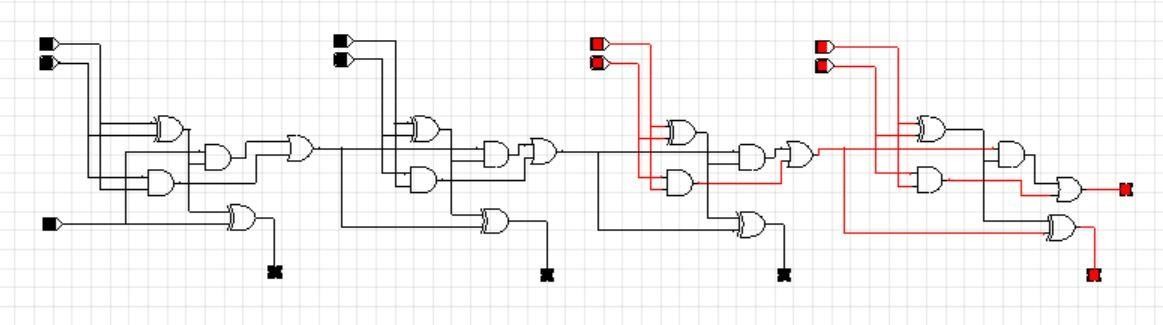
**11.** A = 1010 and B = 1010 and Cin = 0



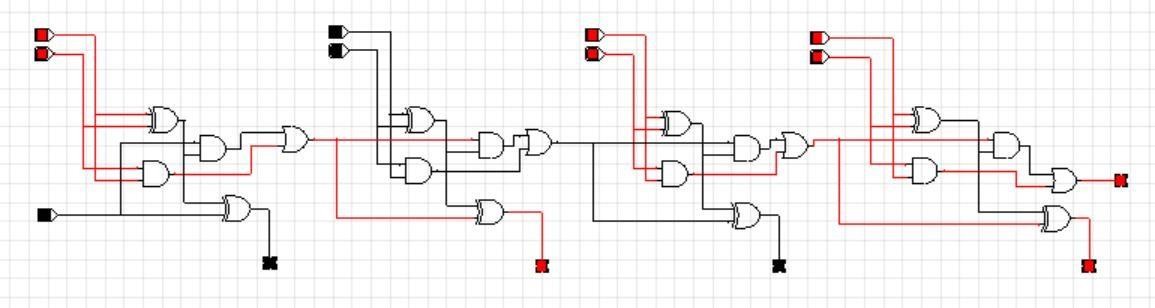
**12.** A = 1011 and B = 1011 and Cin = 0



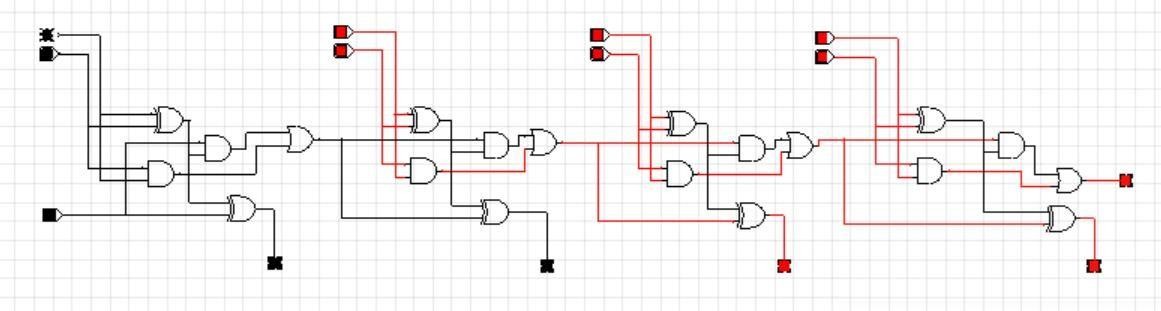
**13.** A = 1100 and B = 1100 and Cin = 0



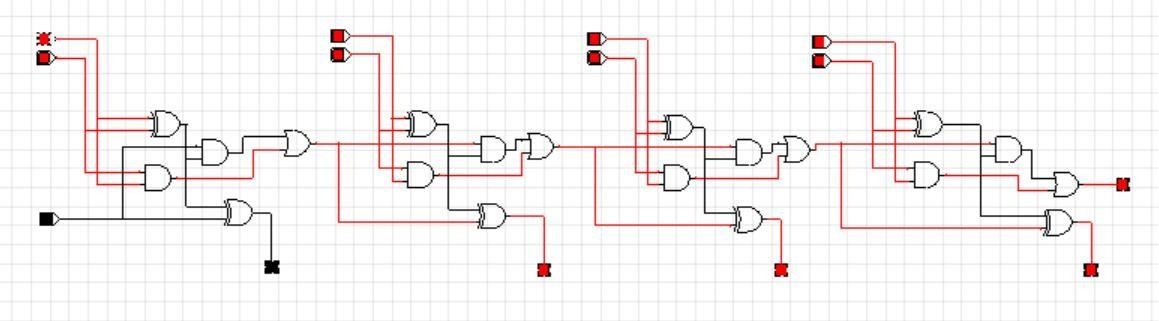
**14.** A = 1101 and B = 1101 and Cin = 0



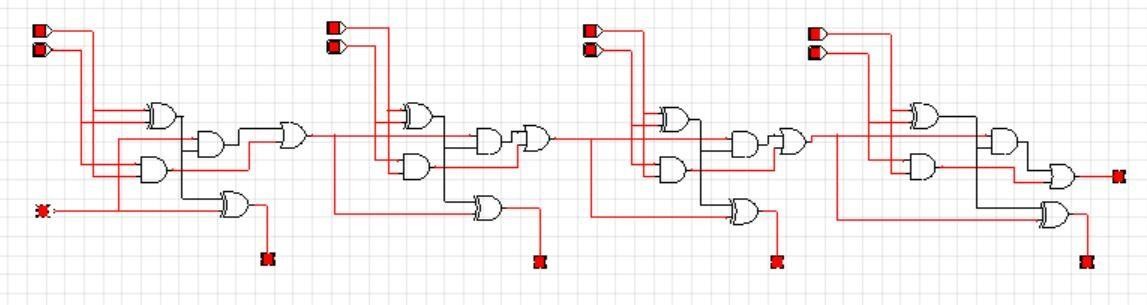
**15.** A = 1110 and B = 1110 and Cin = 0



**16.** A = 1111 and B = 1111 and Cin = 0

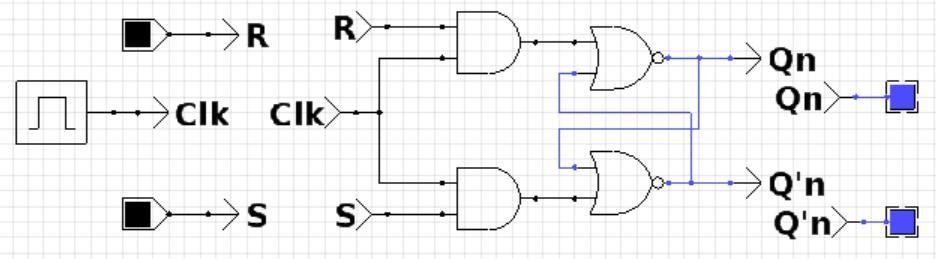


**17.** A = 1111 and B = 1111 and Cin = 1



## Conclusion:

The 4-Bit parallel adder circuit has been implemented and its truth table is verified.



# EXPERIMENT - 4

**AIM**: Design and implement a) SR Flip Flop b) D Flip Flop c) JK Flip Flop

d) T Flip Flop. Also explain the truth tables along with the wave forms.

**Hardware Requirement:** Laptop

(Inspiron14 5408)

[Device Name: DESKTOP-JPH3QS6]

**Software Requirement:** a. Windows 10 (Operating System)

b. CEDAR logic (simulator)

1. SR Flip Flop:

## Procedure:

* 1. Take 2 NOR and 2 AND logic gates.
  2. Take 2 inputs and 1 clock signal.
  3. Connect the 2 inputs to the 2 AND gates input and connect the other 2 terminals of AND gate to the clock signal.
  4. Connect the output of AND gates to one input terminals of NOR gates.
  5. Connect the output terminals of NOR gates to the other NOR gate’s input

terminal as shown.

* 1. Qn+1 = S + R’Qn.

## Circuit Diagram:

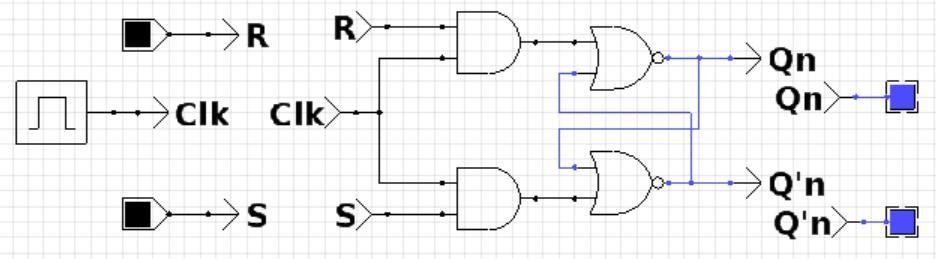
**SR Flip Flop Circuit [ black = ‘0’ ; red = ‘1’; purple = ‘X’ ]**

## Truth Table:

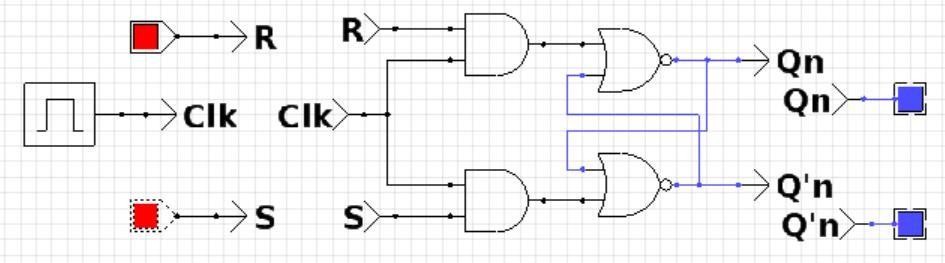
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cloc k | S (Set i/p) | R (Reset i/p) | Qn+1 (Main o/p) | Q’n+1 (other o/p) | State |
| 0 | X | X | Qn | Q’n | Hold |
| 1 | 0 | 0 | Qn | Q’n | Hold |
| 1 | 0 | 1 | 0 | 1 | Rese t |
| 1 | 1 | 0 | 1 | 0 | Set |
| 1 | 1 | 1 | X | X | Inval id |

**Observations:**

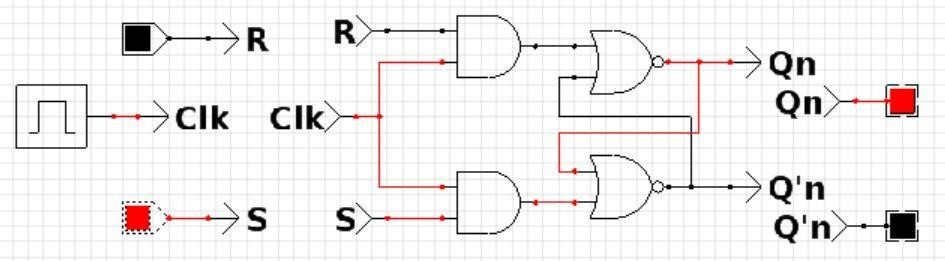
1. Clock = 0 and S = 0 and R = 0



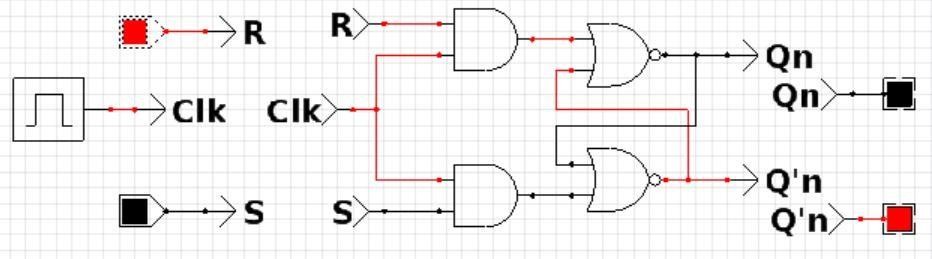
1. Clock = 0 and S = 1 and R = 1



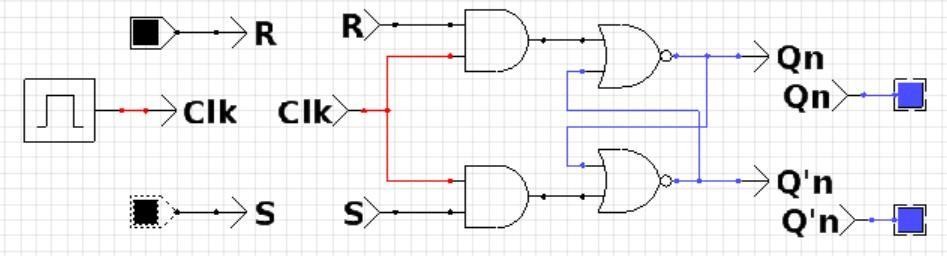
1. Clock = 1 and S = 1 and R = 0



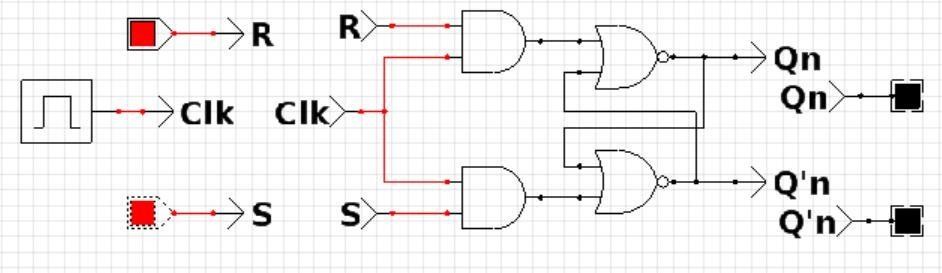
1. Clock = 1 and S = 0 and R = 1



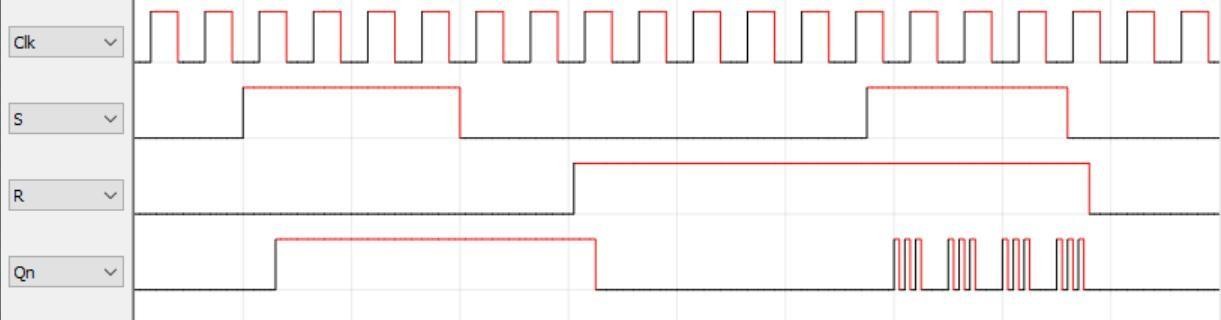
1. Clock = 1 and S = 0 and R = 0



1. Clock = 1 and S = 1 and R = 1



## Waveform:



1. D Flip Flop:

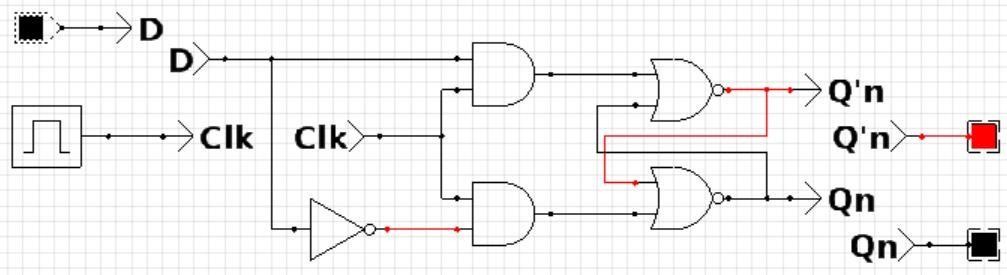
## Procedure:

* 1. Take 2 NOR and 2 AND logic gates.
  2. Take 1 inputs and 1 clock signal and 1 inverter.
  3. Connect the input connect it to the inverter and connect the input and it’s complement to the 2 AND gates input and connect the other 2 terminals of AND gate to the clock signal.
  4. Connect the output of AND gates to one input terminals of NOR gates.
  5. Connect the output terminals of NOR gates to the other NOR gate’s input

terminal as shown.

* 1. Qn+1 =D

## Circuit Diagram:



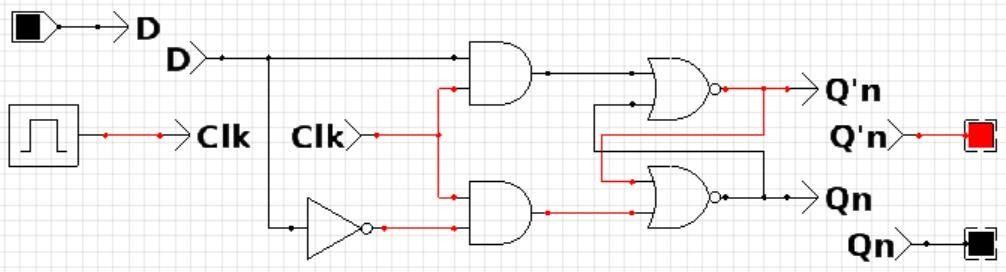
**D Flip Flop Circuit [ black = ‘0’ ; red = ‘1’; purple = ‘X’ ]**

## Truth Table:

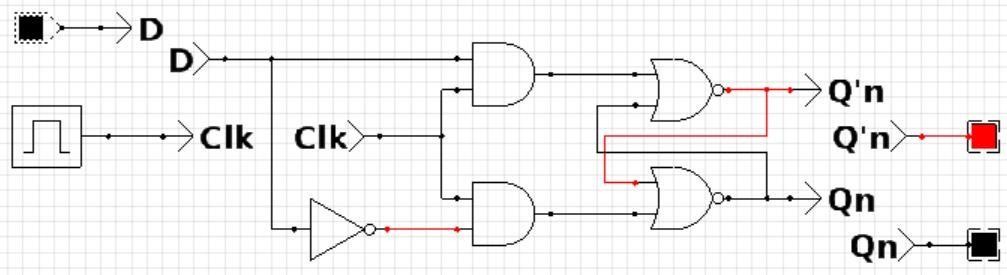
|  |  |  |  |
| --- | --- | --- | --- |
| Cloc k | D  (i/p) | Qn+1 (Main o/p) | State |
| 0 | X | Qn | Hold |
| 1 | 0 | 0 | Rese t |
| 1 | 1 | 1 | Set |

**Observations:**

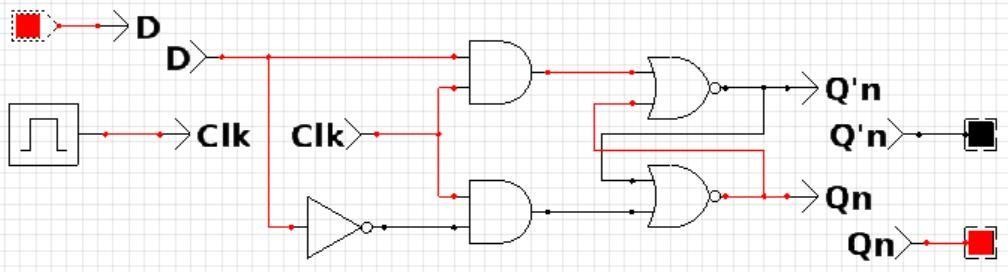
1. Clock = 1 and D = 0



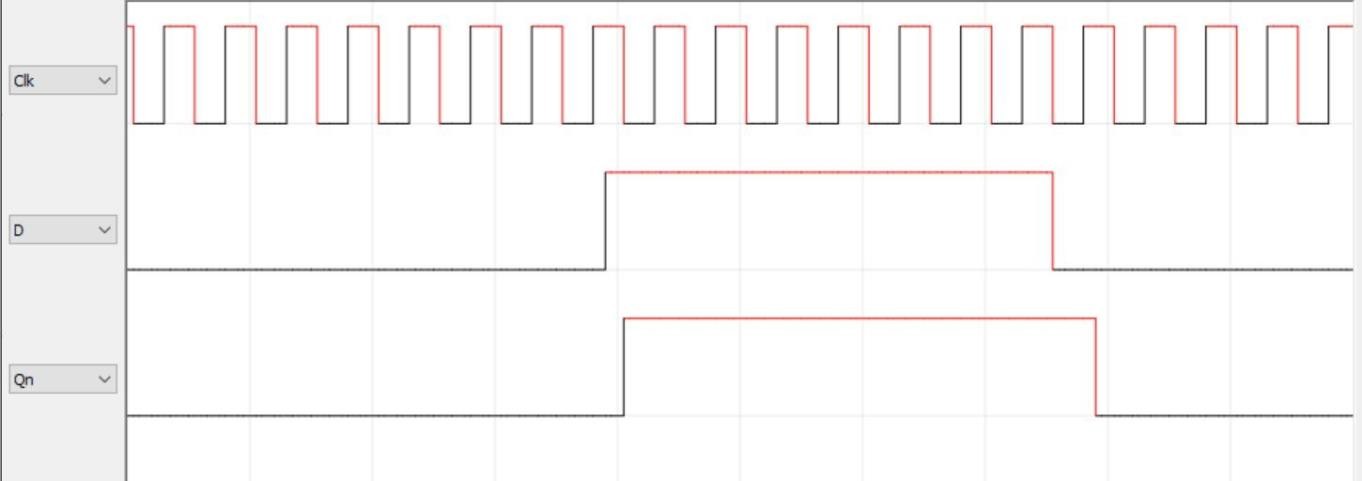
1. Clock = 0 and D = 0



1. Clock = 1 and D = 1



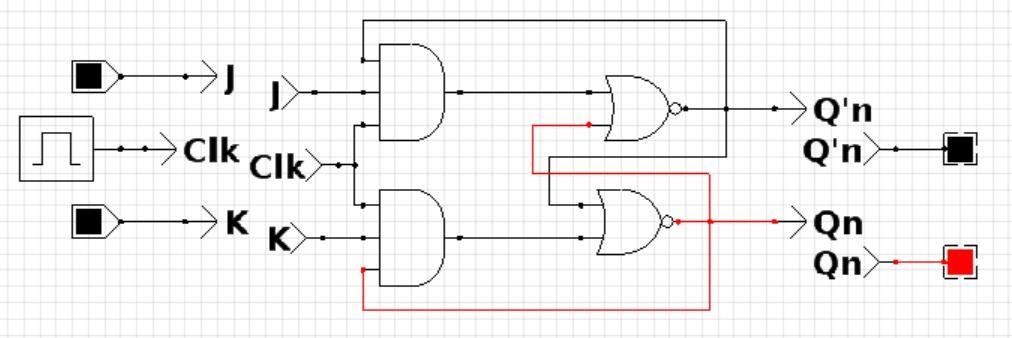
## Waveform:



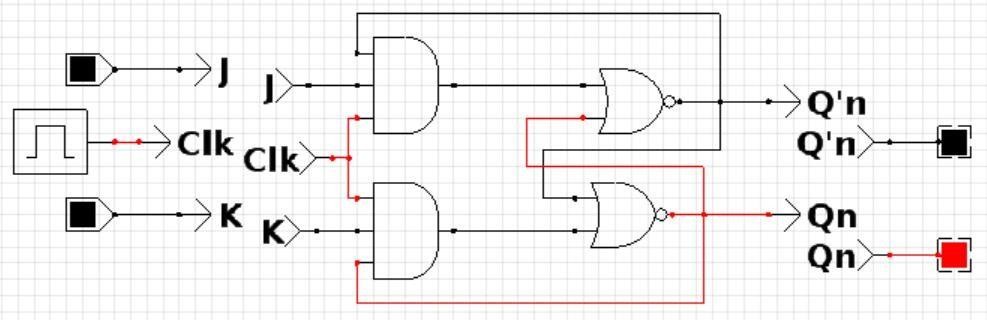
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 25 **| P a g**  C. JK Flip Flop:  **Procedure:**   1. Take 2 (3 input) AND logic gates and 2 (2 input) NOR logic gates. 2. Take 2 inputs and 1 clock signal. 3. Connect the 2 inputs to the 2 AND gates input and connect the other 2 terminals of AND gate to the clock signal. The third input of the AND gate is connected to the alternate output of the NOR gate. 4. Connect the output of NOR gates to one input terminals of alternate AND gates. 5. Connect the output terminals of NOR gates to the other input of NOR gate’s   input terminal as shown.   1. Qn+1 = JQ’n + K’Qn.   **Circuit Diagram:**    **JK Flip Flop Circuit [ black = ‘0’ ; red = ‘1’ ]**  **Truth Table:** | | | | | | | |
|  | Cloc k | J (i/p) | K (i/p) | Qn+1 (Main o/p) | Q’n+1 (other o/p) | State |  |
| 0 | X | X | Qn | Q’n | Hold |
| 1 | 0 | 0 | Qn | Q’n | Hold |
| 1 | 0 | 1 | 0 | 1 | Rese t |
| 1 | 1 | 0 | 1 | 0 | Set |
| 1 | 1 | 1 | Q’n | Qn | Togg  le |
|  |  |  |  |  |  |  |  |

**Observations:**

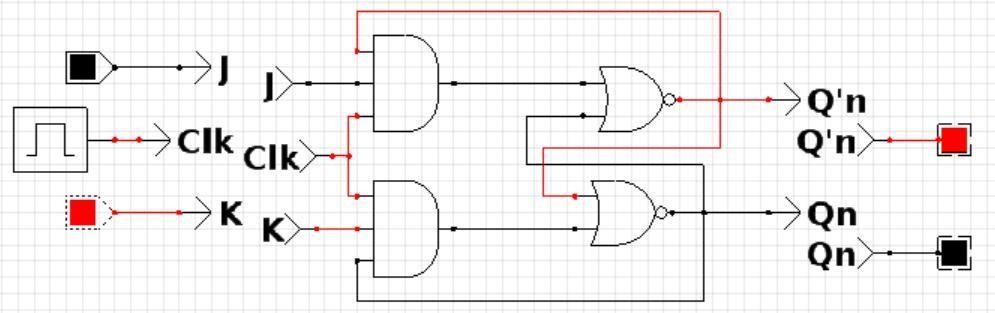
1. Clock = 0 and J = 0 and K = 0



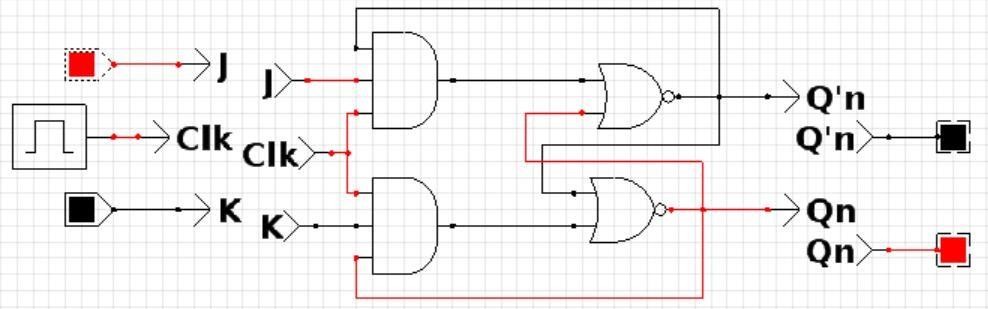
1. Clock = 1 and J = 0 and K = 0



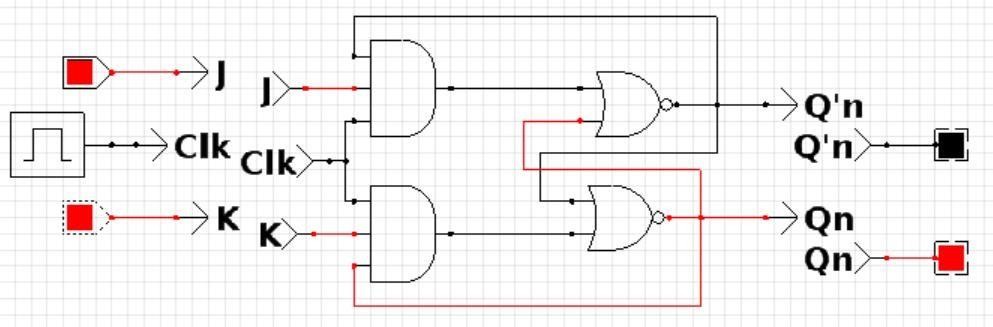
1. Clock = 1 and J = 0 and K = 1



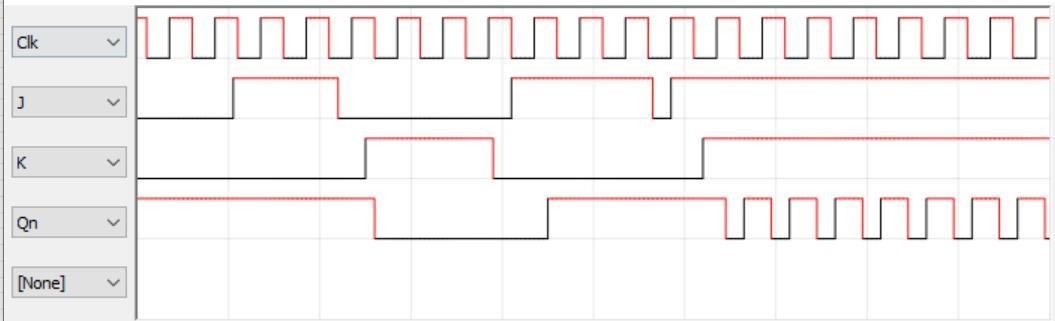
1. Clock = 1 and J = 1 and K = 0



1. Clock = 1 and S = 1 and R = 1



## Waveform:



D. T Flip Flop:

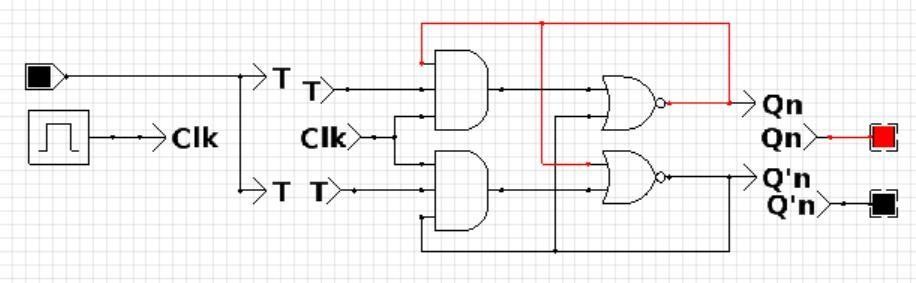
## Procedure:

1. Take 2 (3 input) AND gate and 2 (2 input) NOR gate.
2. Take 1 inputs and 1 clock signal.
3. Connect the 1 input to the 2 AND gates input and connect the other 2 terminals of AND gate to the clock signal. The third input of the AND gate is connected to the alternate output of the NOR gate.
4. Connect the output of NOR gates to one input terminals of alternate AND gates.
5. Connect the output terminals of NOR gates to the other input of NOR gate’s

input terminal as shown.

1. Qn+1 = T’Qn+ Q’nT

## Circuit Diagram:



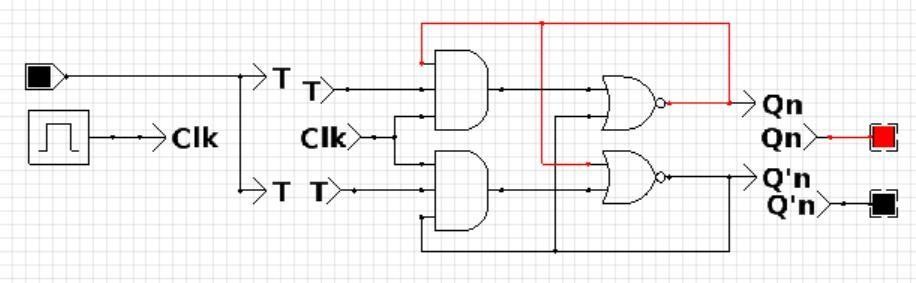
**T Flip Flop Circuit [ black = ‘0’ ; red = ‘1’]**

## Truth Table:

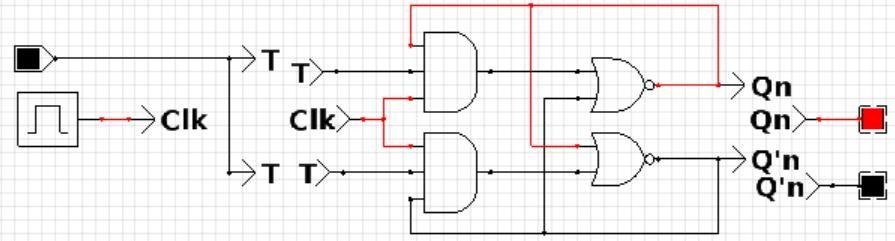
|  |  |  |  |
| --- | --- | --- | --- |
| Cloc k | T  (i/p) | Qn+1 (Main o/p) | State |
| 0 | X | Qn | Hold |
| 1 | 0 | Qn | Hold |
| 1 | 1 | Q’n | Togg le |

**Observations:**

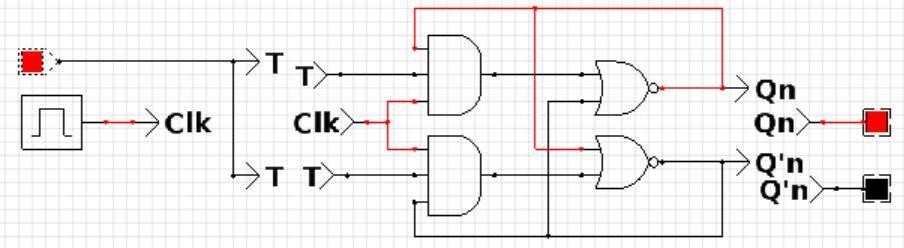
1. Clock = 0 and T = 0



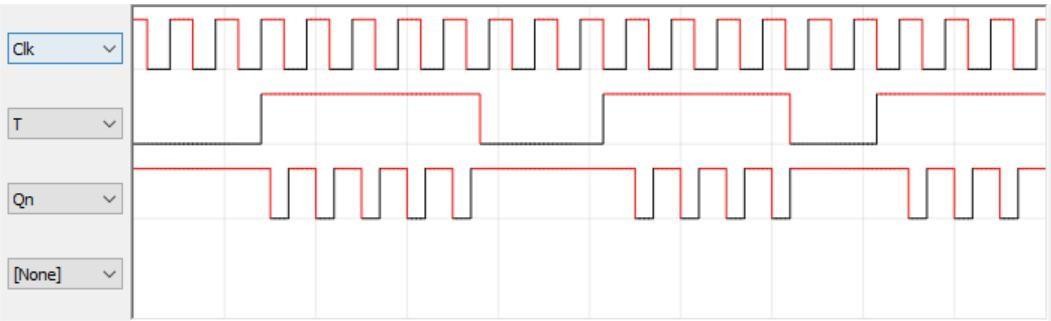
1. Clock = 1 and T = 0



1. Clock = 1 and T = 1

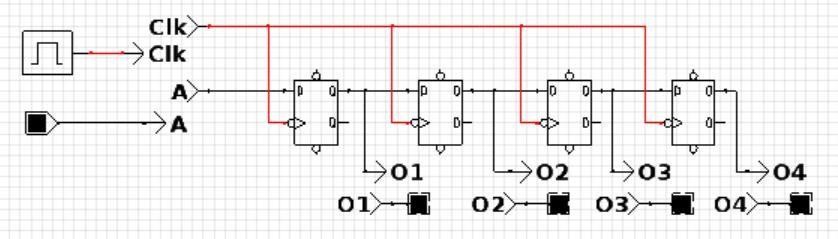


## Waveform:



**Conclusion:**

All the flip flop circuits have been implemented and their truth table is verified.



# EXPERIMENT - 5

**AIM**: Design a 4-Bit shift register using D flip flops and explain its truth table.

**Hardware Requirement:** Laptop

(Inspiron14 5408)

[Device Name: DESKTOP-JPH3QS6]

**Software Requirement:** a. Windows 10 (Operating System)

b. CEDAR logic (simulator)

## Procedure:

* 1. Take 4 D-Flip Flop, 1 input and 1 clock.
  2. Connect the input to any one of the D-Flip Flops.
  3. As for the rest of the rest of the D-Flip Flops, connect the output of 1 D-Flip Flop to the input of next.
  4. Connect the clock to all the D-Flip Flops.

## Circuit Diagram:

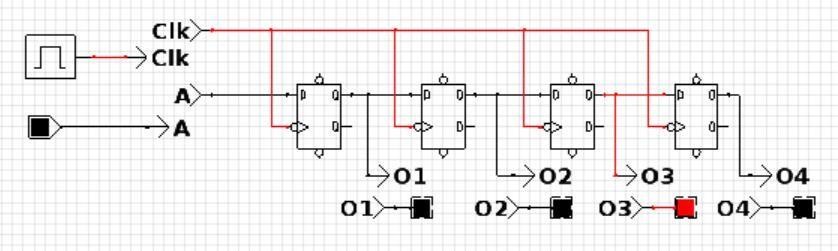
**Full adder Circuit [ black =’0’ ; red=’1’ ]**

## Truth Table:

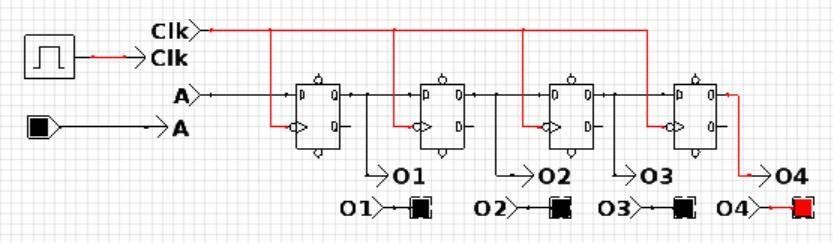
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Clock Pulse | A (input) | O1 | O2 | O3 | O4 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 |
| 2 | 0 | 0 | 0 | 0 | 0 |
| 3 | 1 | 1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 0 | 0 |
| 5 | 0 | 0 | 1 | 1 | 0 |
| 6 | 0 | 0 | 0 | 1 | 1 |
| 7 | 0 | 0 | 0 | 0 | 1 |
| 8 | 0 | 0 | 0 | 0 | 0 |

**Observations:**

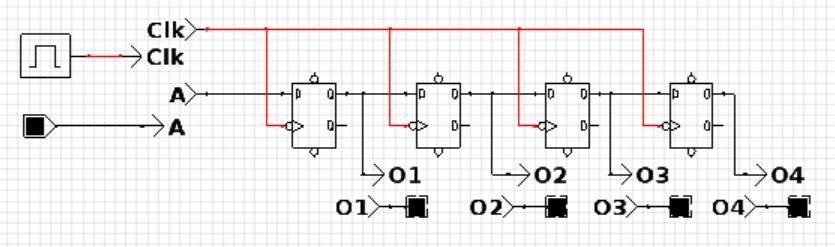
1. Clock Pulse = 0 and A = 0 and O1O2O3O4 = 0010



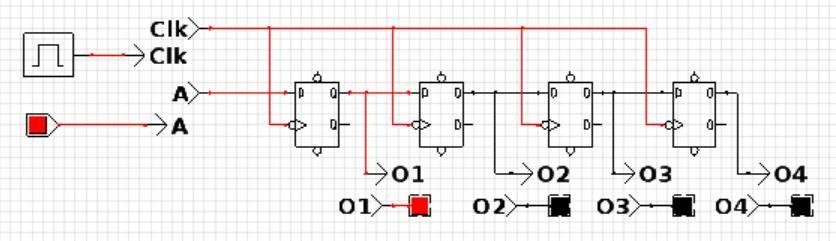
1. Clock Pulse = 1 and A = 0 and O1O2O3O4 = 0001



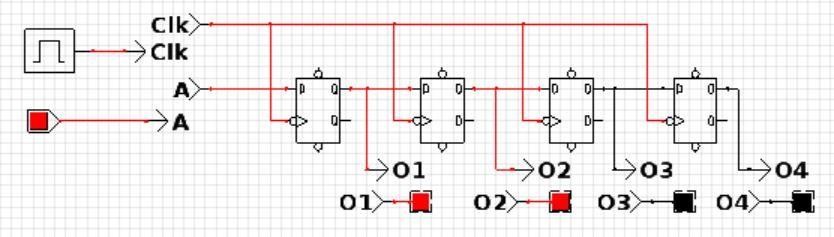
1. Clock Pulse = 2 and A = 0 and O1O2O3O4 = 0000



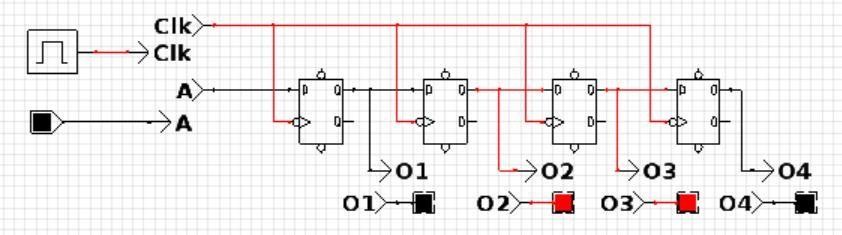
1. Clock Pulse = 3 and A = 1 and O1O2O3O4 = 1000



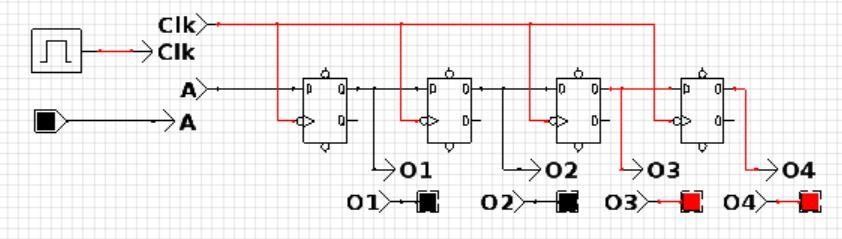
1. Clock Pulse = 4 and A = 1 and O1O2O3O4 = 1100



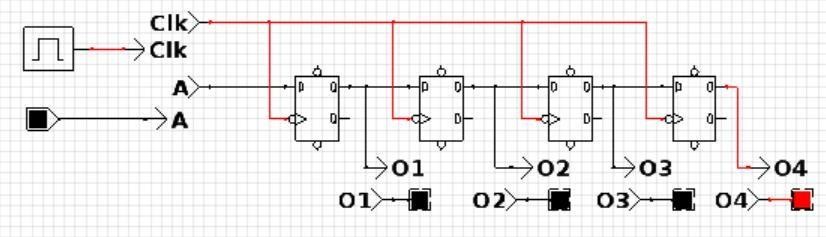
1. Clock Pulse = 5 and A = 0 and O1O2O3O4 = 0110



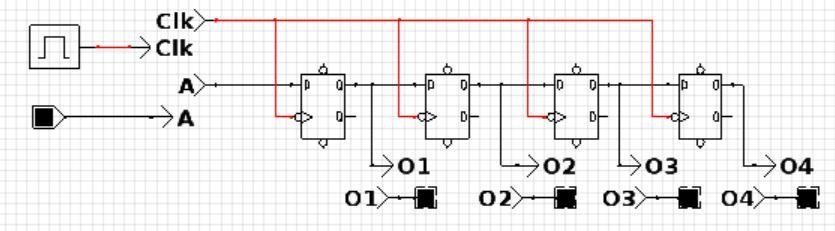
1. Clock Pulse = 6 and A = 0 and O1O2O3O4 = 0011



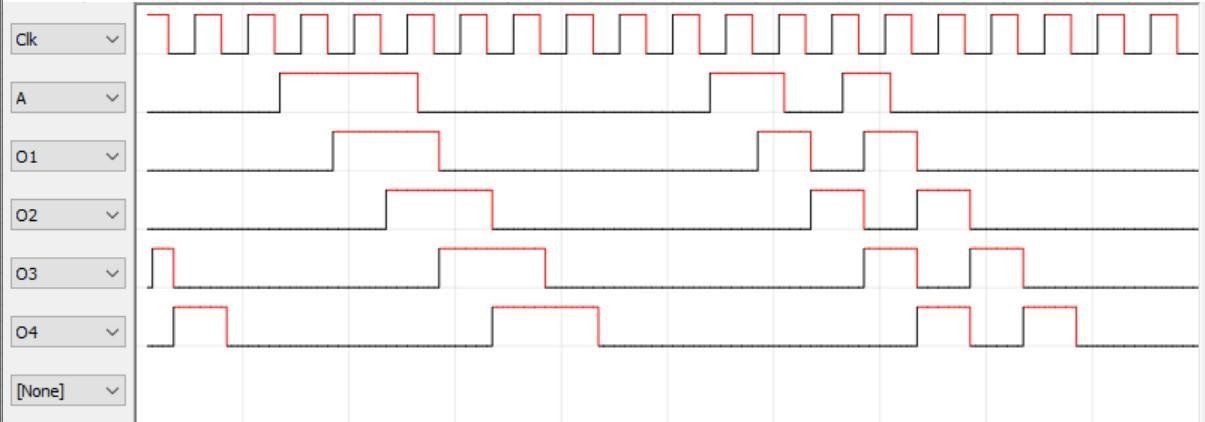
1. Clock Pulse = 7 and A = 0 and O1O2O3O4 = 0001



1. Clock Pulse = 8 and A = 0 and O1O2O3O4 = 0000



## Waveform:



**Conclusion:**

The 4-Bit shift register circuit has been implemented and its truth table is verified.

# EXPERIMENT - 6

**AIM**: To identify high level language code, corresponding assembly language code, instructions passed and its characteristics.

**Hardware Requirement:** Laptop

(Inspiron14 5408)

[Device Name: DESKTOP-JPH3QS6]

**Software Requirement:** a. Windows 10 (Operating System)

b. KEIL - IDE

## Procedure:

* 1. We take the example code of LED Blinking.
  2. Study and understand all the statements in the high level language code.
  3. Debug and build the code.
  4. Open the disassembly window and study and understand the assembly language code.
  5. Observe and understand the usage of memory according to the different instructions.
  6. Observe and understand the instructions framework, instruction length, various registers used (both general purpose and special purpose registers).
  7. Observe and understand the addressing mode and its execution.
  8. Along with different examples try to understand which addressing mode is better.

## Observations:

1. High Level Language Code:

#include <REG51F.H> void wait (void)

{

;

}

void main (void) { unsigned int i; unsigned char j; while (1) {

for (j=0x01; j< 0x80; j<<=1)

{

P1 = j;

for (i = 0; i < 10000; i++)

{

wait ();

}

}

for (j=0x80; j> 0x01; j>>=1)

{

P1 = j;

for (i = 0; i < 10000; i++)

{

wait ();

}

}

} }

1. Assembly Level Language Code:

|  |  |  |  |
| --- | --- | --- | --- |
| C:0x0000 | 02084A LJMP | | C:084A |
| C:0x0003 | 00 | NOP |  |
| C:0x0004 | 00 | NOP |  |
| C:0x0005 | 00 | NOP |  |
| . |  |  |  |
| . |  |  |  |
| . |  |  |  |
| C:0x07FF | 00 | NOP |  |

17: void main (void) { 18: unsigned int i; 19: unsigned char j; 20: while (1) {

21: for (j=0x01; j< 0x80; j<<=1) {

|  |  |  |  |
| --- | --- | --- | --- |
| C:0x0800 | 7D01 | MOV | R5,#0x01 |
| C:0x0802 | ED | MOV | A,R5 |
| C:0x0803 | C3 | CLR | C |
| C:0x0804 | 9480 | SUBB | A,#P0(0x80) |
| C:0x0806 | 7480 | MOV | A,#P0(0x80) |
| C:0x0808 | 9480 | SUBB | A,#P0(0x80) |

C:0x080A 5019 JNC C:0825 22: P1 = j;

C:0x080C 8D90 MOV P1(0x90),R5

|  |  |  |  |
| --- | --- | --- | --- |
| 23: | for (i = 0; i < 10000; i++) { | | |
| C:0x080E | E4 | CLR | A |
| C:0x080F | FE | MOV | R6,A |
| C:0x0810 | FF | MOV | R7,A |

24: wait ();

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C:0x0811 | | 120856 LCALL wait(C:0856) | | |
| 25: | } |  |  |  |
| C:0x0814 | | 0F INC R7 | | |
| C:0x0815 | | BF0001 CJNE R7,#0x00,C:0819 | | |
| C:0x0818 | | 0E INC R6 | | |
| C:0x0819 | | BE27F5 CJNE | | R6,#0x27,C:0811 |
| C:0x081C | | BF10F2 CJNE | | R7,#0x10,C:0811 |
| 26: | } |  |  |  |
| 27: |  |  |  |  |
| C:0x081F | | ED | MOV | A,R5 |
| C:0x0820 | | 25E0 | ADD | A,ACC(0xE0) |
| C:0x0822 | | FD | MOV | R5,A |
| C:0x0823 | | 80DD | SJMP | C:0802 |
| 28: | for (j=0x80; j> 0x01; j>>=1) { | | | |
| C:0x0825 | | 7D80 | MOV | R5,#P0(0x80) |
| C:0x0827 | | ED | MOV | A,R5 |
| C:0x0828 | | D3 | SETB | C |
| C:0x0829 | | 9401 | SUBB | A,#0x01 |
| C:0x082B | | 7480 | MOV | A,#P0(0x80) |
| C:0x082D | | 9480 | SUBB | A,#P0(0x80) |
| C:0x082F | | 40CF | JC main(C:0800) | |
| 29: | P1 = j; | |  |  |
| C:0x0831 | | 8D90 | MOV | P1(0x90),R5 |
| 30: | for (i = 0; i < 10000; i++) { | | | |
| C:0x0833 | | E4 CLR A | | |
| C:0x0834 | | FE MOV R6,A | | |
| C:0x0835 | | FF MOV R7,A | | |
| 31: | wait (); | |  |  |
| C:0x0836 | | 120856 LCALL wait(C:0856) | | |

32: }

C:0x0839 0F INC R7

C:0x083A BF0001 CJNE R7,#0x00,C:083E C:0x083D 0E INC R6

C:0x083E BE27F5 CJNE R6,#0x27,C:0836 C:0x0841 BF10F2 CJNE R7,#0x10,C:0836

33: }

C:0x0844 ED MOV A,R5 C:0x0845 C3 CLR C C:0x0846 13 RRC A C:0x0847 FD MOV R5,A C:0x0848 80DD SJMP C:0827

C:0x084A 787F MOV R0,#0x7F C:0x084C E4 CLR A C:0x084D F6 MOV @R0,A

C:0x084E D8FD DJNZ R0,C:084D C:0x0850 758107 MOV SP(0x81),#0x07 C:0x0853 020800 LJMP main(C:0800)

13: void wait (void) { C:0x0856 22 RET

C:0x0857 00 NOP

C:0x0858 00 NOP

.

.

.

C:0xFFFB 00 NOP

1. Instructions and its Characteristics:

## Instruction Format:

It is a sequence of bits contained in a machine instruction that defines the layout of the instruction.

a. C:0x0815 BF0001 CJNE R7,#0x00,C:0819

|  |  |  |
| --- | --- | --- |
| OPCODE | Operand 1 | Operand 2 |

(1 Byte) (1 Byte) (2 Byte)

b. C:0x081F ED MOV A,R5

OPCODE

(1 Byte)

c. C:0x0820 25E0 ADD A,ACC(0xE0)

Operand 1

OPCODE

(1 Byte) (1 Byte)

## Instruction Length:

a. C:0x0815 BF0001 CJNE R7,#0x00,C:0819

Instruction length is 4 bytes.

b. C:0x081F ED MOV A,R5 Instruction length in 1 byte.

c. C:0x0820 25E0 ADD A,ACC(0xE0) Instruction length is 2 bytes.

## Addressing Mode:

a. C:0x0800 7D01 MOV R5,#0x01

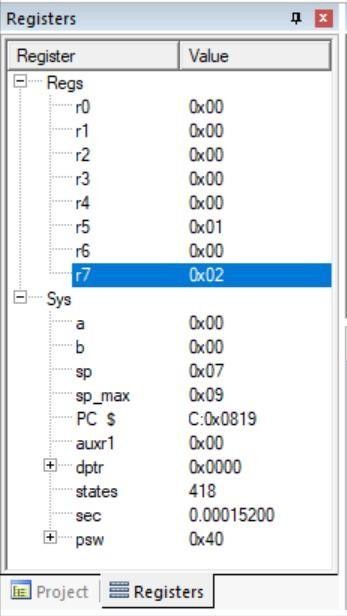
Immediate Addressing Mode

b. C:0x0802 ED MOV A,R5 Direct Addressing Mode

c. C:0x084D F6 MOV @R0,A Indirect Addressing mode

d. C:0x0850 758107 MOV SP(0x81),#0x07

Indirect Addressing Mode

1. CPU Registers used:

Some Special purpose registers used:

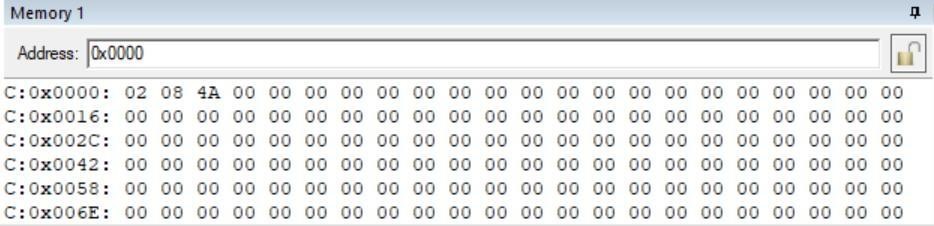
* + Stack pointer register
  + Data pointer
  + Program counter
  + PSW (Program Status Word)

Some General purpose registers used:

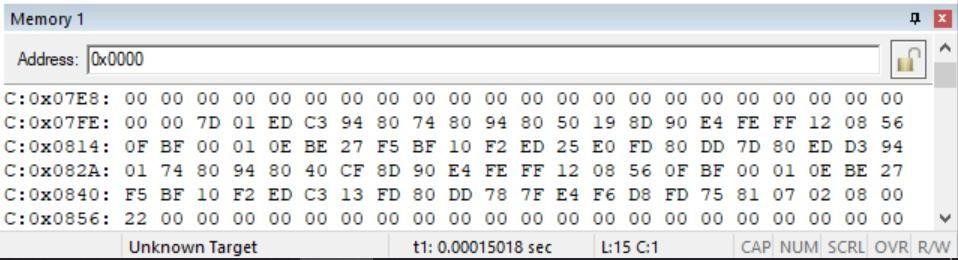
* + r0 to r7

1. Memory Used:

a. C: 0x0000 to C: 0x0002 (3 bytes)

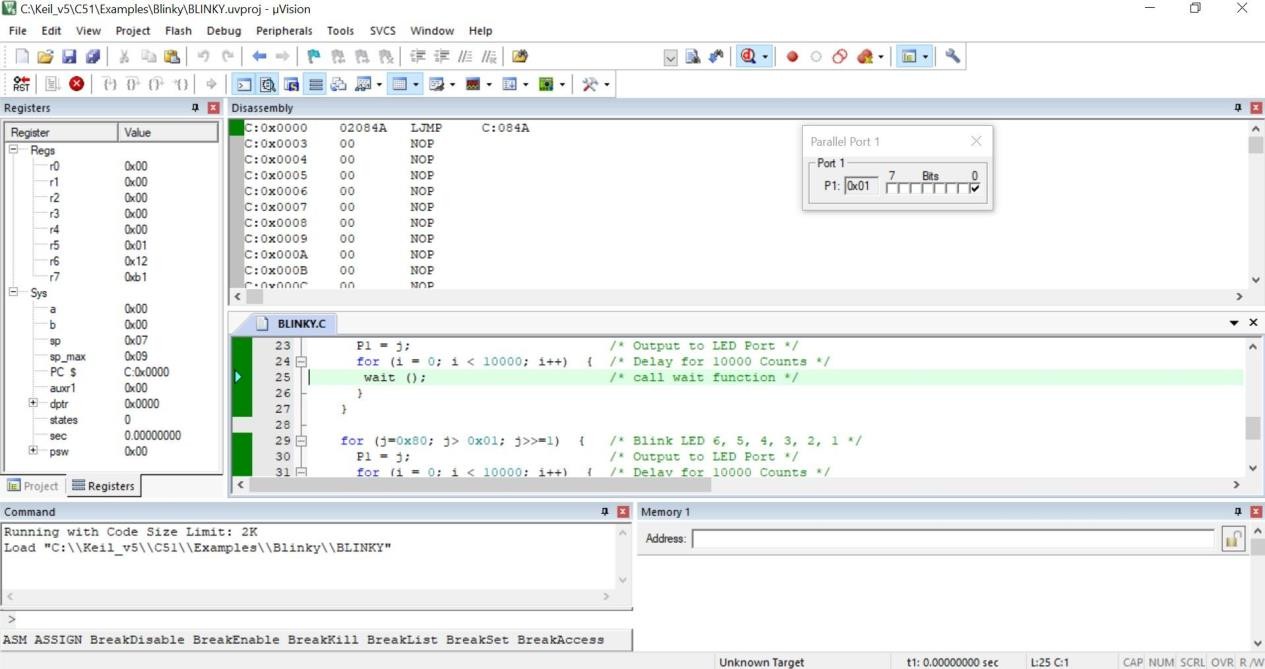


b. C:0x0800 to C:0x0856 (87 bytes)



Total memory used: 90 bytes

1. Output :



## Conclusion:

The high level language and assembly language code is understood. The instructions and their characteristics are also observed and understood. The best addressing mode used in the above instructions will be the Direct Mode.