

**Vidyavardhini’s**

**College of Engineering & Technology**

Vasai Road (W)

**Department of Artificial Intelligence & Data Science**

**Laboratory Manual**

**AI(DS) - 28**

| Semester | III | Class | S.E |
| --- | --- | --- | --- |
| Course Code | CSL302 | | |
| Course Name | Digital Logic and Computer Organization and Architecture Lab | | |

****

**Vidyavardhini’s College of Engineering & Technology**

**Vision**

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

**Mission**

* To provide technologically inspiring environment for learning.
* To promote creativity, innovation and professional activities.
* To inculcate ethical and moral values.
* To cater personal, professional and societal needs through quality education.

**Department Vision:**

To foster proficient artificial intelligence and data science professionals, making remarkable contributions to industry and society.

**Department Mission:**

* To encourage innovation and creativity with rational thinking for solving the challenges in emerging areas.
* To inculcate standard industrial practices and security norms while dealing with Data.
* To develop sustainable Artificial Intelligence systems for the benefit of various sectors.

**Program Specific Outcomes (PSOs):**

PSO1: Analyze the current trends in the field of Artificial Intelligence & Data Science and contribute to the technological advancements by presenting / publishing at national / international forums..

PSO2: Design and develop Artificial Intelligence & Data Science applications and solutions in various domains to cater to the needs of industry and society.

**Program Outcomes (POs):**

Engineering Graduates will be able to:

* **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
* **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
* **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
* **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
* **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
* **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
* **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
* **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
* **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
* **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
* **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
* **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

**Course Objectives**

| 1 | To implement operations of the arithmetic unit using algorithms. |
| --- | --- |
| 2 | Design and simulate different digital circuits. |
| 3 | To design memory subsystems including cache memory. |
| 4 | To demonstrate CPU and ALU design. |

**Course Outcomes**

| CO | At the end of course students will be able to: | **Action verbs** | **Bloom’s**  **Level** |
| --- | --- | --- | --- |
| **CSL302.1** | Verify the truth table of logic, universal gates, and realize logic circuits using hardware. | Verify,  Realize | Apply (level - 3) |
| **CSL302.2** | Implement combinational circuits design using hardware. | Implement | Evaluate (level - 5) |
| **CSL302.3** | Implement sequential & code conversion circuits design using hardware. | Implement | Evaluate (level - 5) |
| **CSL302.4** | Write Booth's, Restoring, and Non-Restoring algorithms for arithmetic operations using C-Programming language. | Write | Apply (level - 3) |
| **CSL302.5** | Implement ripple carry adder, carry look ahead adder, ALU design using virtual lab. | Implement | Evaluate (level - 5) |
| **CSL302.6** | Implement CPU, memory and Cache memory designs using a virtual lab. | Implement | Evaluate (level - 5) |

**Mapping of Experiments with Course Outcomes**

| **List of**  **Experiments** | **Course Outcomes** | | | | |  |
| --- | --- | --- | --- | --- | --- | --- |
| **CSL404**  **.1** | **CSL404**  **.2** | **CSL404**  **.3** | **CSL404**  **.4** | **CSL404**  **.5** | **CSL40**  **4.6** |
| 1 | To verify the truth table of various logic gates using ICs. | 3 | - | - | - | - |
| 2 | To realize the gates using universal gates. | - | 3 | - | - | - |
| 3 | To perform code conversion | - | 3 | - | - | - |
| 4 | To realize half adder and full adder. | - | - | 3 | - | - |
| 5 | To implement ripple carry adder. | - | - | 3 | - | - |
| 6 | To implement carry look ahead adder. | - | - | - | 3 | - |
| 7 | To implement Booth’s algorithm. | - | - | - | 3 | - |
| 8 | To implement restoring division algorithm. | - | - | - | - | 3 |
| 9 | To implement non restoring division algorithm. | - | - | - | - | 3 |
| 10 | To implement ALU design. | - | - | - | - | - |

**List of Experiments**

| **Sr.**  **No.** | **Name of Experiment** | **DOP** | **DOC** | **Marks** | **Sign** |
| --- | --- | --- | --- | --- | --- |
|  | **Basic Experiments** | |  |  |  |
| 1 | To verify the truth table of various logic gates using ICs. |  |  |  |  |
| 2 | To realize the gates using universal gates. |  |  |  |  |
| 3 | To realize half adder and full adder. |  |  |  |  |
| 4\* | Study of flip flop IC. |  |  |  |  |
| 5 | To implement ripple carry adder. |  |  |  |  |
| 6 | To implement carry look ahead adder. |  |  |  |  |
| 7 | To implement Booth’s algorithm. |  |  |  |  |
| 8 | To implement restoring division algorithm. |  |  |  |  |
| 9 | To implement non restoring division algorithm. |  |  |  |  |
| 10\* | To implement memory design |  |  |  |  |

| **Assignment** | | | | | |
| --- | --- | --- | --- | --- | --- |
| 11 | Assignment 1: Number System |  |  |  |  |
| 12 | Assignment 2: Booth’s Algorithm |  |  |  |  |
| 13 | Assignment 3: Process Organization & Architecture |  |  |  |  |
| 14 | Assignment 4: Control Unit Design |  |  |  |  |
| 15 | Assignment 5: Memory Organization |  |  |  |  |
| 16 | Assignment 6: Principal of Advanced Processor & Bussses |  |  |  |  |
| **Formative Assessment** | | | | | |
| 17 | Th - Quiz 1: Number System |  |  |  |  |
| 18 | Th - Quiz 2: Booth’s Algorithm |  |  |  |  |
| 19 | Th - Quiz 3: Process Organization & Architecture |  |  |  |  |
| 20 | Th - Quiz 4: Control Unit Design |  |  |  |  |
| 21 | Th - Quiz 5: Memory Organization |  |  |  |  |
| 22 | Th - Quiz 6: Principal of Advanced Processor & Bussses |  |  |  |  |
| 23 | Pr - Quiz 1: Logic Gates & Universal Gates |  |  |  |  |
| 24 | Pr - Quiz 2:Combinational Circuits |  |  |  |  |
| 25 | Pr - Quiz 3: Sequential & code conversion circuits |  |  |  |  |
| 26 | Pr - Quiz 4: Booth’s Algorithm |  |  |  |  |
| 21 | Pr - Quiz 5: Adder & ALU design |  |  |  |  |
| 22 | Pr - Quiz 6: Memory Design |  |  |  |  |

D.O.P: Date of performance

D.O.C : Date of correction

**INDEX**

| **Sr. No.** | **Name of Experiment** | **D.O.P.** | **D.O.C.** | **Page No.** | **Remark** |
| --- | --- | --- | --- | --- | --- |
| 1 | To verify the truth table of various logic gates using ICs. | 24/07 |  |  |  |
| 2 | To realize the gates using universal gates. |  |  |  |  |
| 3 | To realize half adder and full adder. |  |  |  |  |
| 4 | Study of flip flop IC |  |  |  |  |
| 5 | To implement ripple carry adder. |  |  |  |  |
| 6 | To implement carry look ahead adder. |  |  |  |  |
| 7 | To implement Booth’s algorithm. |  |  |  |  |
| 8 | To implement restoring division algorithm. |  |  |  |  |
| 9 | To implement non restoring division algorithm. |  |  |  |  |
| 10 | To implement ALU design. |  |  |  |  |

| Experiment No. 1 |
| --- |
| Truth table of various logic gates using ICs. |
| Name: **James Lewis** |
| Roll Number: **28** |
| Date of Performance: **24/07/2024** |
| Date of Submission: |

**Aim -** To verify the truth table of various logic gates using ICs.

**Objective -**

1. Understand how to use the breadboard to patch up, test your logic design and debug it.
2. The principal objective of this experiment is to fully understand the function and use of logic gates.
3. Understand how to implement simple circuits based on a schematic diagram using logic gates.

**Components required -**

1. IC’s 7408, 7432, 7404

2. Bread Board.

3. Connecting wires.

**Theory -**

In digital electronics, a gate is logic circuits with one output and one or more inputs. Logic gates are available as integrated circuits.

**AND gate** :

AND gate performs logical multiplication, more commonly known as AND operation. The AND gate output will be in high state only when all the inputs are in high state.7408 is a Quad 2 input AND gate.

**OR gate:**

It performs logical addition. Its output become high if any of the inputs is in logic high. 7432 is a Quad 2 input OR gate.

**NOT gate:**

It performs basic logic function for inversion or complementation. The purpose of the inverter is to change one logic level to the opposite level. IC 7404 is a Hex inverter.

**Circuit Diagram, Truth Table -**

**AND Gate -**

A grid of numbers and letters

Description automatically generated

**OR Gate -**

A grid of numbers and letters

Description automatically generated with medium confidence

**NOT Gate -**

A diagram of a circuit

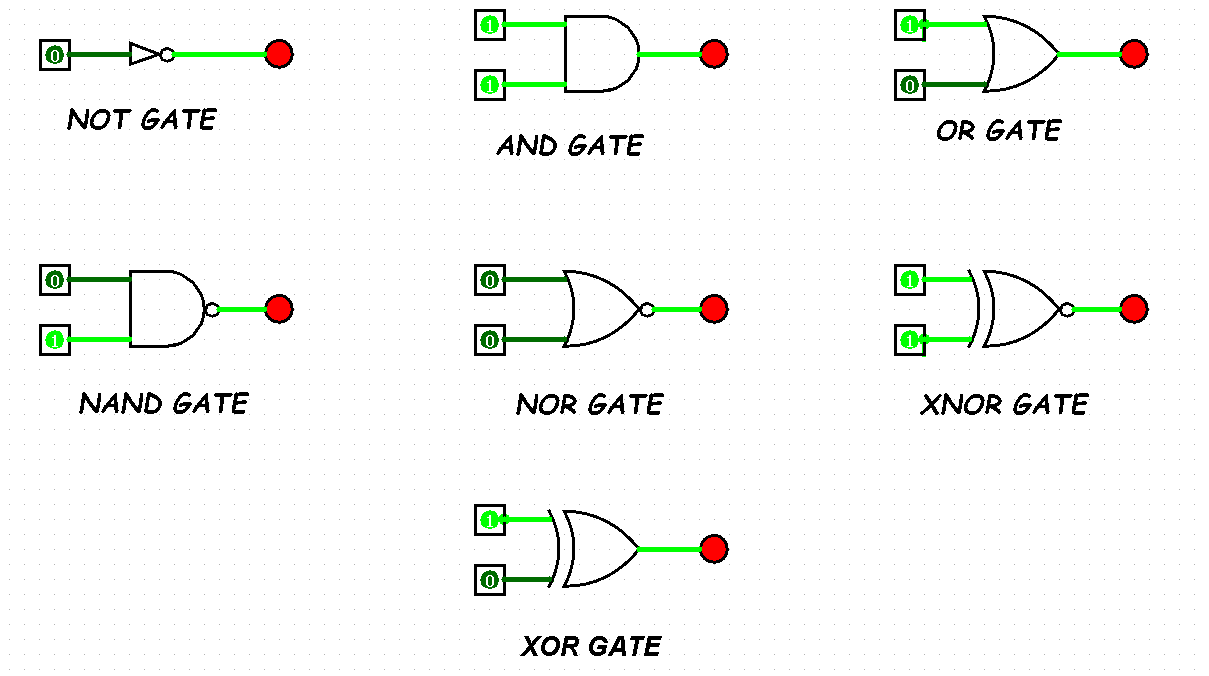
Description automatically generated

**Procedure:**

1.Test all the components in the Ic packages using a digital IC tester. Also assure whether all the connecting wires are in good condition by testing for the continuity using a Multimeter or a trainer kit.

2.Verify the dual in line package (DIP) inout of the IC before feeding the inputs.

3.Set up the circuits and observe the outputs.

****

**Conclusion -**

The experiment successfully verifies the truth tables for the AND, OR, and NOT gates using the respective ICs. The outputs correspond correctly to the expected results based on the defined truth tables, demonstrating a clear understanding of the functionality of these basic logic gates in digital circuits. This hands-on experience also enhances skills in using breadboards and debugging circuit designs.

| Experiment No. 2 |
| --- |
| Basic gates using universal gates. |
| Name: James Lewis |
| Roll Number: 28 |
| Date of Performance: 31-07-2024 |
| Date of Submission: |

**Aim -** To realize the gates using universal gates.

**Objective -**

1. To study the realization of basic gates using universal gates.
2. Understanding how to construct any combinational logic function using NAND or NOR gates only.

**Theory -**

AND, OR, NOT are called basic gates as their logical operation cannot be simplified further.

NAND and NOR are called universal gates as using only NAND or only NOR, any logic function can be implemented.

**Components required -**

1. IC’s 7400(NAND) 7402(NOR)

2. Bread Board.

3. Connecting wires.

**Circuit Diagram -**

A diagram of a circuit

Description automatically generated with medium confidence

A diagram of a block

Description automatically generated

A diagram of a mathematical equation

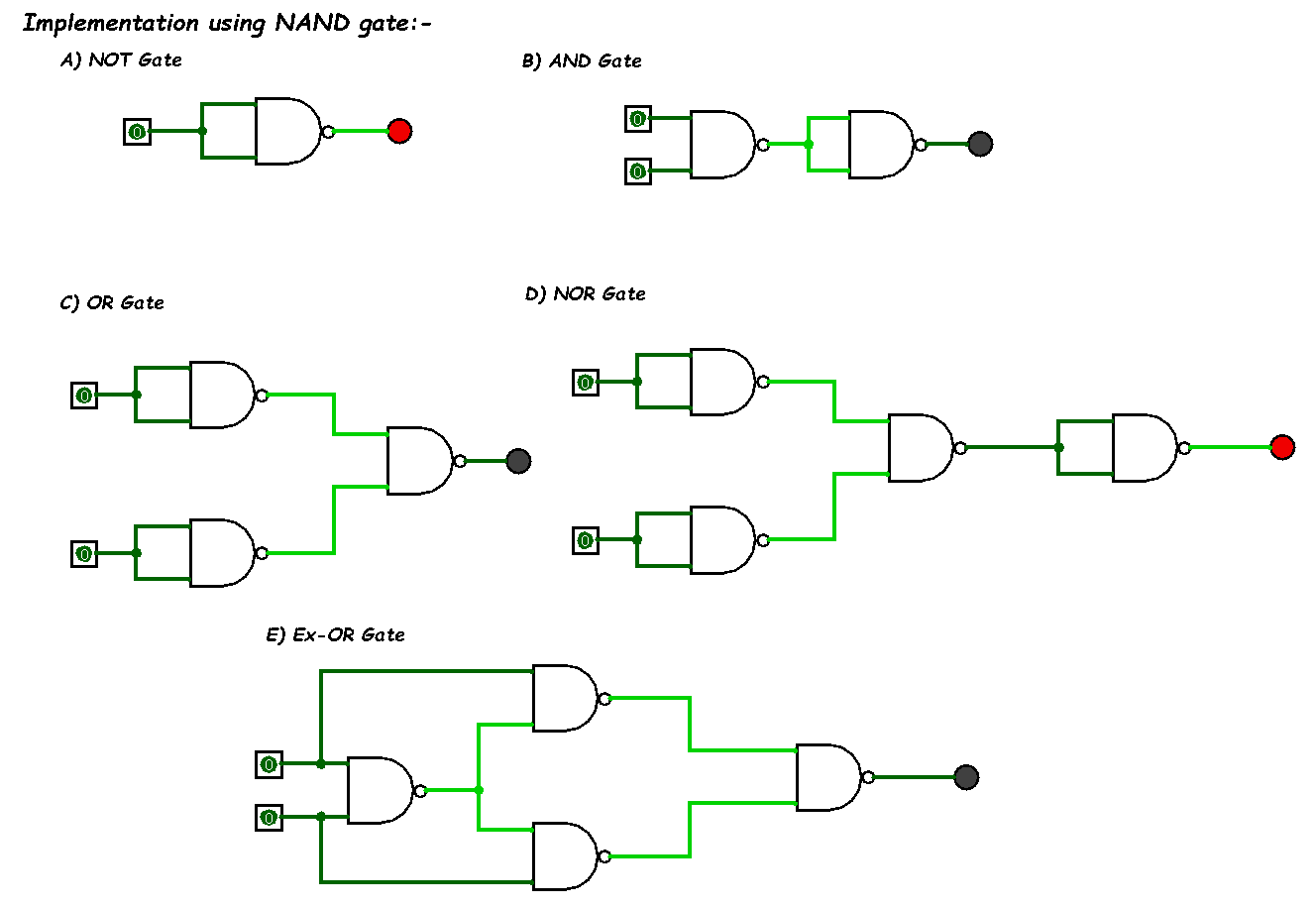
Description automatically generated with medium confidenceA black line drawing of a rocket

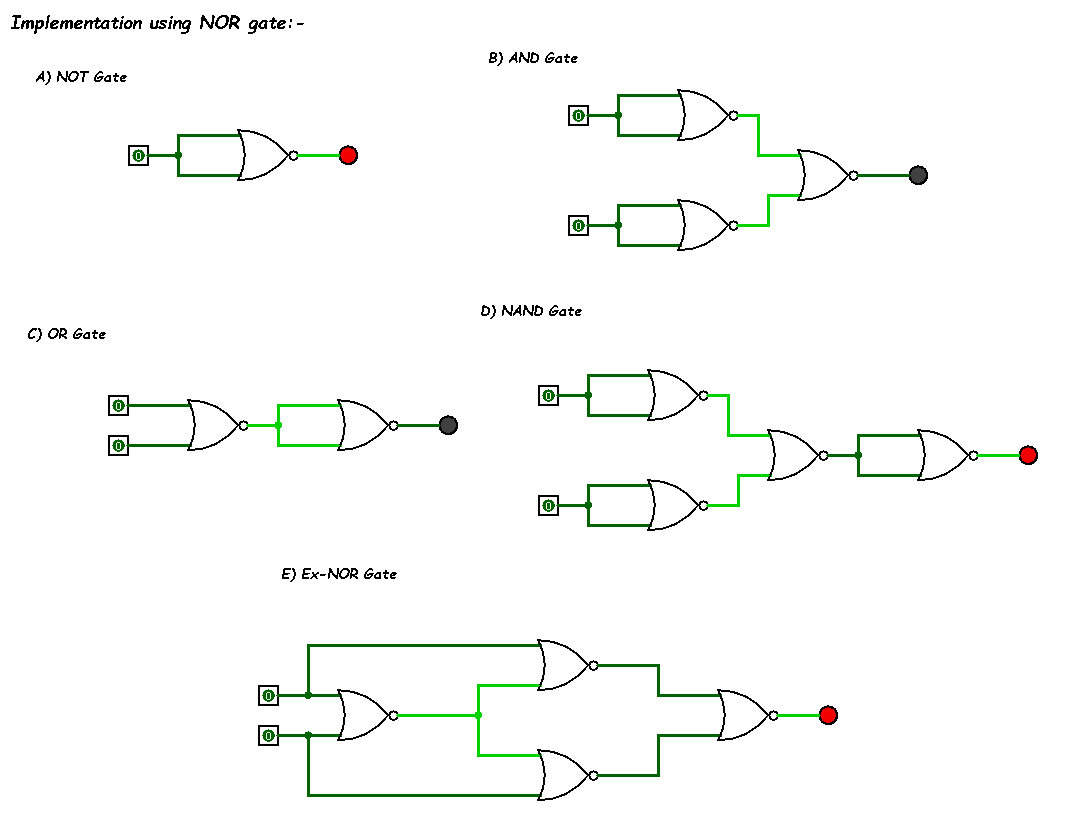
Description automatically generated

**Procedure:**

a) Connections are made as per the circuit diagrams.

b) By applying the inputs, the outputs are observed and the operations are verified with the help of truth table.

****

****

**Conclusion** -

The experiment successfully demonstrates the realization of basic logic gates using universal gates (NAND and NOR). By constructing AND, OR, and NOT gates with these universal gates, we confirm their versatility and the principle that any combinational logic function can be implemented using just NAND or NOR gates. This exercise enhances understanding of digital logic design and the fundamental relationships between different types of logic gates.

| Experiment No. 3 |
| --- |
| To realize half adder and full adder. |
| Name: James Lewis |
| Roll Number: 28 |
| Date of Performance: 14/08/2024 |
| Date of Submission: |

**Aim -** To realize half adder and full adder.

**Objective -**

1. The objective of this experiment is to understand the function of Half-adder, Full-adder, Half-subtractor and Full-subtractor.
2. Understand how to implement Adder and Subtractor using logic gates.

**Components required** -

1. IC’s - 7486(X-OR), 7432(OR), 7408(AND), 7404 (NOT)

2. Bread Board

3. Connecting wires.

**Theory** -

Half adder is a combinatio]

‘

n nal logic circuit with two inputs and two outputs. The half adder circuit is designed to add two single bit binary numbers A and B. It is the basic building block for addition of two single bit numbers. This circuit has two outputs CARRY and SUM.

Sum =A ⊕ B

Carry = A B

Full adder is a combinational logic circuit with three inputs and two outputs. Full adder is developed to overcome the drawback of HALF ADDER circuit. It can add two one bit umbers A and B. The full adder has three inputs A, B, and CARRY in,the circuit has two outputs CARRY out and SUM.

Sum = (A⊕B) ⊕ Cin

Carry = AB + Cin (A⊕B)  
 Subtracting a single-bit binary value B from another A (i.e. A -B) produces a difference bit D and a borrow out bit B-out. This operation is called half subtraction and the circuit to realize it is called a half subtractor. The Boolean functions describing the half- Subtractor are

Sum =A ⊕B

Carry = A’ B

Subtracting two single-bit binary values, B, Cin from a single-bit value A produces a difference bit D and a borrow out Br bit. This is called full subtraction. The Boolean functions describing the full-subtractor are

Difference = (A ⊕ B) ⊕Cin

Borrow = A’B + A’(Cin) + B(Cin)

**Circuit Diagram and Truth Table -**

**Half-adder**

A diagram of a diagram

Description automatically generated

**Full-adder**

A diagram of a number and a number

Description automatically generated

**Procedure -**

1. Verify the gates.

2. Make the connections as per the circuit diagram.

3. Switch on VCC and apply various combinations of input according to truth table.

4. Note down the output readings for half/full adder and half/full subtractor, Sum/difference and the carry/borrow bit for different combinations of inputs verify their truth tables.

**Conclusion -**

The experiment successfully demonstrates the realization of half adder, full adder, half subtractor, and full subtractor circuits using logic gates. The outputs corresponded accurately to the expected results based on the defined truth tables, illustrating a clear understanding of binary addition and subtraction in digital electronics. This hands-on experience enhances the comprehension of combinational logic circuits and their applications in arithmetic operations.

| Experiment No. 4 |
| --- |
| Study of flip flop IC |
| Name: James Lewis |
| Roll Number: 28 |
| Date of Performance: 21/08/2024 |
| Date of Submission: |

# **Aim -** Study of flip flop IC

**Objective -**

1. To familiarize students with the operation of various types of flip-flops, including RS, JK, D, and T flip-flops.
2. To verify the truth tables for each type of flip-flop through practical circuit implementations.
3. To explore the role of flip-flops in memory storage and sequential logic circuits.

**Components required** -

1. ICs:
   * 7474 (Dual D Flip-flop)
   * 7476 (Dual JK Flip-flop)
   * 7400 (Quad 2-input NAND gate)
   * 7402 (Quad 2-input NOR gate)
   * 7404 (Hex Inverter)
   * 7408 (Quad 2-input AND gate)
   * 7432 (Quad 2-input OR gate)
   * 7486 (Quad 2-input XOR gate)
2. Breadboard
3. Connecting wires
4. Digital IC Tester
5. Clock Pulse Generator or Manual Switch for Clock Input
6. Power Supply (5V)

**Theory** -

Flip-flops are bistable devices that can store one bit of data. They have two stable states and can be used to store state information. The main types of flip-flops include:

## 1. RS Flip-flop

1. Inputs: Set (S) and Reset (R)
2. Outputs: Q and Q' (complement of Q)
3. Operation:
   * When S = 1 and R = 0, Q = 1 (Set state).
   * When S = 0 and R = 1, Q = 0 (Reset state).
   * When S = 0 and R = 0, Q remains unchanged.
   * S = 1 and R = 1 is an invalid state.

## 2. D Flip-flop

1. Input: Data (D)
2. Output: Q
3. Operation: Captures the value of D at the rising edge of the clock. The output Q follows the input D only at the clock transition.

## 3. JK Flip-flop

1. Inputs: J and K
2. Outputs: Q and Q'
3. Operation:
   * J = 1, K = 0: Set (Q = 1).
   * J = 0, K = 1: Reset (Q = 0).
   * J = 1, K = 1: Toggle the output.
   * J = 0, K = 0: No change.

## 4. T Flip-flop

1. Input: Toggle (T)
2. Output: Q
3. Operation: Toggles the output state on each clock pulse when T = 1; holds the previous state when T = 0.

## Applications of Flip-flops

Flip-flops are used in various digital applications, including:

* Memory storage elements (registers)
* Frequency dividers
* Counters
* Shift registers
* State machines

**Conclusion -**

The experiment successfully demonstrates the functionality of various flip-flops, including RS, JK, D, and T types. The outputs observed during the experiments matched the expected results based on the truth tables. This practical experience enhances the understanding of flip-flops as essential components in digital circuits, particularly in memory storage and sequential logic applications. By implementing these circuits, students gain hands-on experience with digital electronics, preparing them for more complex systems involving memory and data storage. The knowledge acquired will be beneficial for future studies in digital systems and electronics design.

| Experiment No. 5 |
| --- |
| Implement ripple carry adder |
| Name: |
| Roll Number: |
| Date of Performance: |
| Date of Submission: |

**Aim:** To implement ripple carry adder.

**Objective:** To understand the operation of a ripple carry adder, specifically how the carry ripples through the adder.

1. examining the behavior of the working module to understand how the carry ripples through the adder stages
2. to design a ripple carry adder using full adders to mimic the behavior of the working module
3. the adder will add two 4 bit numbers

**Theory:** Arithmetic operations like addition, subtraction, multiplication, division are basic operations to be implemented in digital computers using basic gates like AND, OR, NOR, NAND etc. Among all the arithmetic operations if we can implement addition then it is easy to perform multiplication (by repeated addition), subtraction (by negating one operand) or division (repeated subtraction).

Half Adders can be used to add two one bit binary numbers. It is also possible to create a logical circuit using multiple full adders to add N-bit binary numbers. Each full adder inputs a Cin, which is the Cout of the previous adder. This kind of adder is a Ripple Carry Adder, since each carry bit "ripples" to the next full adder. The first (and only the first) full adder may be replaced by a half adder. The block diagram of 4-bit Ripple Carry Adder is shown here below -

A diagram of a computer component

Description automatically generated

The layout of ripple carry adder is simple, which allows for fast design time; however, the ripple carry adder is relatively slow, since each full adder must wait for the carry bit to be calculated from the previous full adder. The gate delay can easily be calculated by inspection of the full adder circuit. Each full adder requires three levels of logic. In a 32-bit [ripple carry] adder, there are 32 full adders, so the critical path (worst case) delay is 31 \* 2(for carry propagation) + 3(for sum) = 65 gate delays.

**Design Issues:**

The corresponding Boolean expressions are given here to construct a ripple carry adder. In the half adder circuit the sum and carry bits are defined as

sum = A ⊕ B

carry = AB

In the full adder circuit the the Sum and Carry outpur is defined by inputs A, B and Carryin as

Sum=ABC + ABC + ABC + ABC

Carry=ABC + ABC + ABC + ABC

Having these we could design the circuit. But, we first check to see if there are any logically equivalent statements that would lead to a more structured equivalent circuit.

With a little algebraic manipulation, one can see that

Sum= ABC + ABC + ABC + ABC

= (AB + AB) C + (AB + AB) C

= (A ⊕ B) C + (A ⊕ B) C

=A ⊕ B ⊕ C

Carry= ABC + ABC + ABC + ABC

= AB + (AB + AB) C

= AB + (A ⊕ B) C

**Procedure:**

Procedure to perform the experiment: Design of Ripple Carry Adders

1. Start the simulator as directed. This simulator supports 5-valued logic.
2. To design the circuit we need 3 full adder, 1 half adder, 8 Bit switch(to give input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or presses the 'show pin config’button. Pin numbering starts from 1 and from the bottom left corner (indicating with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1, For full adder input is in pin-5,6,8 output sum is in pin-4 and carry is pin-1
5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 3 full adders(from the Adder drawer in the pallet), 8 Bit switches, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet, if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect 4 bit switches to the 4 terminals of a digital display and another set of 4 bit switches to the 4 terminals of another digital display. connect the pin-1 of the full adder which will give the final carry output. connet the sum(pin-4) of all the adders to the terminals of the third digital display(according to the circuit diagram shown in screenshot). After the connection is over click the selection tool in the pallet.
7. To see the circuit working, click on the Selection tool in the pallet then give input by double clicking on the bit switch, (let it be 0011(3) and 0111(7)) you will see the output on the output(10) digital display as sum and 0 as carry in bit display.

**Circuit diagram of Ripple Carry Adder:**

A diagram of a circuit

Description automatically generated

**Components required:**

The components needed to create 4 bit ripple carry adder is listed here -

* 4 full-adders
* wires to connect
* LED display to obtain the output

OR we can use

* 3 full-adders
* 1 half adder
* wires to connect
* LED display to obtain the output

**Screenshots of Ripple Carry Adder:**

A screenshot of a computer

Description automatically generated

**Conclusion:**

| Experiment No.6 |
| --- |
| Implement Carry Look Ahead Adder. |
| Name: |
| Roll Number: |
| Date of Performance: |
| Date of Submission: |

**Aim:**  . To implement carry look ahead adder.

**Objective:**

It computes the carries parallely thus greatly speeding up the computation.

1. To understanding behaviour of carry lookahead adder from module designed by the student as part of the experiment
2. To understand the concept of reducing computation time with respect of ripple carry adder by using carry generate and propagate functions.
3. The adder will add two 4 bit numbers

**Theory:**

To reduce the computation time, there are faster ways to add two binary numbers by using carry lookahead adders. They work by creating two signals P and G known to be Carry Propagator and Carry Generator. The carry propagator is propagated to the next level whereas the carry generator is used to generate the output carry ,regardless of input carry. The block diagram of a 4-bit Carry Lookahead Adder is shown here below -

A diagram of a computer program

Description automatically generated

The number of gate levels for the carry propagation can be found from the circuit of full adder. The signal from input carry Cin to output carry Cout requires an AND gate and an OR gate, which constitutes two gate levels. So if there are four full adders in the parallel adder, the output carry C5 would have 2 X 4 = 8 gate levels from C1 to C5. For an n-bit parallel adder, there are 2n gate levels to propagate through.

**Design Issues :**

The corresponding boolean expressions are given here to construct a carry lookahead adder. In the carry-lookahead circuit we ned to generate the two signals carry propagator(P) and carry generator(G),

Pi = Ai ⊕ Bi

Gi = Ai · Bi

The output sum and carry can be expressed as

Sumi = Pi ⊕ Ci

Ci+1 = Gi + ( Pi · Ci)

Having these we could design the circuit. We can now write the Boolean function for the carry output of each stage and substitute for each Ci its value from the previous equations:

C1 = G0 + P0 · C0

C2 = G1 + P1 · C1 = G1 + P1 · G0 + P1 · P0 · C0

C3 = G2 + P2 · C2 = G2 P2 · G1 + P2 · P1 · G0 + P2 · P1 · P0 · C0

C4 = G3 + P3 · C3 = G3 P3 · G2 P3 · P2 · G1 + P3 · P2 · P1 · G0 + P3 · P2 · P1 · P0 · C0

**Procedure:**

Procedure to perform the experiment: Design of Carry Look ahead Adders

1. Start the simulator as directed. This simulator supports 5-valued logic.
2. To design the circuit we need 7 half adder, 3 OR gate, 1 V+(to give 1 as input), 3 Digital display(2 for seeing input and 1 for seeing output sum), 1 Bit display(to see the carry output), wires.
3. The pin configurations of a component are shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner (indicating with the circle) and increases anticlockwise.
4. For half adder input is in pin-5,8 output sum is in pin-4 and carry is pin-1
5. Click on the half adder component(in the Adder drawer in the pallet) and then click on the position of the editor window where you want to add the component(no drag and drop, simple click will serve the purpose), likewise add 6 more full adders(from the Adder drawer in the pallet), 3 OR gates(from Logic Gates drawer in the pallet), 1 V+, 3 digital display and 1 bit Displays(from Display and Input drawer of the pallet, if it is not seen scroll down in the drawer)
6. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components; connect V+ to the upper input terminals of 2 digital displays according to you input. Connect the OR gates according to the diagram shown in the screenshot connect the pin-1 of the half adder which will give the final carry output. Connect the sum (pin-4) of those adders to the terminals of the third digital display which will give output sum. After the connection is over click the selection tool in the pallet.
7. See the output; in the screenshot diagram we have given the value 0011(3) and 0111(7) so get 10 as sum and 0 as carry. You can also use many bit switches instead of V+ to give input and by double clicking those bit switches can give different values and check the result.

**Circuit diagram of Carry Look Ahead Adder:**

A diagram of a diagram

Description automatically generated

**Components required:**

The components needed to create 4 bit carry look ahead adder is listed here -

1. 7 half-adders: 4 to create the look adder circuit, and 3 to evaluate Si and Pi · Ci
2. 3 OR gates to generate the next level carry Ci+1
3. wires to connect
4. LED display to obtain the output

**Screenshots of Carry Look Ahead Adder:**

A screenshot of a computer

Description automatically generated

**Conclusion:**

| Experiment No. 7 |
| --- |
| Implement Booth’s algorithm using c-programming |
| Name: |
| Roll Number: |
| Date of Performance: |
| Date of Submission: |

**Aim:** To implement Booth’s algorithm using c-programming.

**Objective -**

1. To understand the working of Booths algorithm.
2. To understand how to implement Booth’s algorithm using c-programming.

**Theory:**

Booth’s algorithm is a multiplication algorithm that multiplies two signed binary numbers in 2’s complement notation. Booth used desk calculators that were faster at shifting than adding and created the algorithm to increase their speed.

The algorithm works as per the following conditions :

1. If Qn and Q-1 are same i.e. 00 or 11 perform arithmetic shift by 1 bit.

2. If Qn Q-1 = 10 do A= A - B and perform arithmetic shift by 1 bit.

3. If Qn Q-1 = 01 do A= A + B and perform arithmetic shift by 1 bit.

A diagram of a mathematical algorithm

Description automatically generated

A table with numbers and symbols

Description automatically generated

**Program:**

**Output:**

**Conclusion -**

| Experiment No. 8 |
| --- |
| Implement Restoring algorithm using c-programming |
| Name: |
| Roll Number: |
| Date of Performance: |
| Date of Submission: |

**Aim:** To implement Restoring division algorithm using c-programming.

**Objective -**

1. To understand the working of Restoring division algorithm.
2. To understand how to implement Restoring division algorithm using c-programming.

**Theory:**

1) The divisor is placed in M register, the dividend placed in Q register.

2) At every step, the A and Q registers together are shifted to the left by 1-bit

3) M is subtracted from A to determine whether A divides the partial remainder. If it does, then Q0 set to 1-bit. Otherwise, Q0 gets a 0 bit and M must be added back to A to restore the previous value.

4) The count is then decremented and the process continues for n steps. At the end, the quotient is in the Q register and the remainder is in the A register.

**Flowchart**

A diagram of a number system

Description automatically generated

**Program-**

**Output -**

**Conclusion -**

| Experiment No. 9 |
| --- |
| Implement Non-Restoring algorithm using c-programming |
| Date of Performance: |
| Date of Submission: |

**Aim -** To implement Non-Restoring division algorithm using c-programming.

**Objective -**

1. To understand the working of Non-Restoring division algorithm.
2. To understand how to implement Non-Restoring division algorithm using c-programming.

**Theory:**

In each cycle content of the register, A is first shifted and then the divisor is added or subtracted with the content of register A depending upon the sign of A. In this, there is no need of restoring, but if the remainder is negative then there is a need of restoring the remainder. This is the faster algorithm of division.

A diagram of a program

Description automatically generated

**Program -**

**Output:**

**Conclusion -**

| Experiment No.10 |
| --- |
| Implement Memory design. |
| Name: |
| Roll Number: |
| Date of Performance: |
| Date of Submission: |

**Aim:** To implement Memory design

**Objective :** Objective of 4 bit arithmetic logic unit (with AND, OR, XOR, ADD operation):

1. To understand behaviour of arithmetic logic unit from working module.
2. To Design an arithmetic logic unit for given parameter.

**Theory:**

Design of Memory :

A memory unit is a collection of storage cells together with associated circuits needed to transform insformation in and out of the device. Memory cells which can be accessed for information transfer to or from any desired random location is called random access memory(RAM). The block diagram of a memory unit-

A blue rectangle with black background

Description automatically generated

Internal Construction: The internal construction of a random-access memory of m words with n bits per word consists of m\*n binary storage cells and associated decoding circuits for selecting individual words. The binary cell is the basic building block of a memory unit.

RAM Design:

Design of a RAM cell :

The binary cell has three inputs and one output. The select input enables the cell for reading or writing and the reda/write input determines the cell operation when it is selected. A 1 in the read/write input provides the read operation by forming a path from the flip-flop to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the flip-flop. the logic diagram is-

A black background with a black square

Description automatically generated with medium confidence

Design of a 4X4 RAM :

The logical construction of a small RAM 4X3 is shown below. It consists of 4 words of 3 bits each and has a total of 12 binary cells. Each block labeled BC represents the binary cell with its 3 inputs and 1 output. The block diagram of a binary cell-

A memory with 4 words needs two address lines. The two address inputs go through a 2\*4 decoder to select one of the four words. The decoder is enabled with the memory enable input. When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected. With the memory enable at 1, one of the four words is selected, dictated by the value in the two address lines. Once a word has been selected, the read/write input determines the operation. the logic diagram is-

A screenshot of a computer

Description automatically generated

Design Issues :

A basic RAM cell has been provided here as a component which can be used to design larger memory units. An IC memory consisting of 4 words each having 3 bits has been aslo provided.

**Procedure**

1. Procedure to perform the experiment:Design of 4X3 RAM memory:
2. Start the simulator as directed.This simulator supports 5-valued logic.To design the circuit we need 12 binary RAM cell, 9 OR gate, 7 bit switch (to give input,which will toggle its value with a double click), 3 bit display (to see the output), wires.
3. The pin configuration of a component is shown whenever the mouse is hovered on any canned component of the palette or press the 'show pinconfig' button. Pin numbering starts from 1 and from the bottom left corner (indicated with the circle) and increases anticlockwise.
4. For a binary RAM cell input is in pin-5, output is in pin-4 and select is pin-8, Read/Write is in pin-6, for read operation give 1 input to Read/Write pin. For write operation give 0 input to Read/Write pin.
5. For a 'decoder with enable', input A is in pin-6, B is in pin-5, output D0 is in pin-4, D1 is in pin-3, D2 is in pin-2, D3 is in pin-1 and Enable is in pin-8
6. Click on the 'decoder with enable' component (in the Other Components drawer in the pallet) and then click on the position of the editor window where you want to add the component (no drag and drop, simple click will serve the purpose), likewise add 12 binary RAM cell (from the Other Components drawer in the pallet), 9 OR gates (from Logic Gates drawer in the pallete), 7 bit switches (which will toggle its value with a double click), 3 bit displays (from Display and Input drawer of the pallet,if it is not seen scroll down in the drawer)
7. To connect any two components select the Connection menu of Palette, and then click on the Source terminal and click on the target terminal. According to the circuit diagram connect all the components, connect 2 bit switches to the inputs of the 'decoder with enable' (which will act as address input), 1 bit switch to the enable pin of the 'decoder with enable' (which will act as memory enable input), connect a bit switch to the Read/Write(R/W') line, 3 bit switches to the data inputs line, 3 bit displays to the data output line and OR gates according to the diagram shown in the circuit diagram. after athe connection is over click the selection tool in the pallete.
8. To see the circuit working, Do some read or write operation by properly setting the R/W', memory enable then give input and check the output. suppose you give, R/W'=1, memory enable=1, address input=01, data input=101, then it will be a read operation and you will not see 101 as output, it will store 101 in the word-1. now again set, R/W'=0, memory enable=1, address input=01, then it will be a write operation and you will see 101 as the content of word-1 on the output display.

**Circuit diagram of 4 bit ALU:**

A black background with a black square

Description automatically generated with medium confidence

**Components required :**

For Designing a RAM Cell

To build a RAM Cell, we need :

AND Gate(2 input)-6

NOT Gate-2

RS Flip Flop-1

For Designing a 4X3 RAM

To build a 4X3 RAM, we need :

OR Gate(2 input)-11

RAM Cell-12

2X4 Decoder with Enable-1

**Screenshots of Memory design:**

**A diagram of a computer

Description automatically generated**

**Conclusion:**