Bachelor of Computer Applications (BCA)

Digital Electronics Lab

(OBCASE108P24)

Self-Learning Material
(SEM 1)

Jaipur National University
Centre for Distance and Online Education

Established by Government of Rajasthan
Approved by UGC under Sec 2(f) of UGC ACT 1956
&
NAAC A+ Accredited

TABLE OF CONTENTS

COURSE INTRODUCTION

"Clean code always looks like it was written by someone who cares."
- Robert C. Martin

The Digital Electronics Lab is a hands-on course designed to complement theoretical knowledge gained in digital electronics studies with practical, real-world applications. This course provides students with the opportunity to explore digital circuit design and analysis through experimental work, allowing them to apply theoretical concepts in a controlled laboratory environment.

In this lab course, students will engage in a series of experiments that cover the fundamental principles of digital electronics. They will work with various digital components, including logic gates, flip-flops, counters, and memory devices, gaining practical experience in assembling, testing, and troubleshooting digital circuits.

The laboratory sessions are structured to reinforce learning by allowing students to design and build circuits based on specific requirements. They will use digital simulation tools and hardware platforms to test their designs, analyze performance, and make necessary adjustments. This hands-on approach helps bridge the gap between theoretical knowledge and practical application, providing valuable insights into the behavior and limitations of digital systems.

Students will also be introduced to advanced digital design techniques, including the use of programmable logic devices and hardware description languages. Throughout the course, students will document their experimental work, analyze results, and present their findings. This process helps develop critical thinking and problem-solving skills, as well as the ability to communicate technical information effectively.

By the end of the course, students will have a comprehensive understanding of digital electronics through practical experience. They will be able to design, build, and test digital circuits, and will be well-prepared to apply their skills in real-world engineering contexts. This hands-on experience is crucial for understanding the complexities of digital systems and for preparing students for careers in electronics and related fields.

Course Outcomes:

At the completion of the course, a student will be able to:

1. Understand and Recall and learn the basics of logic gates & code conversion.
2. Design capability in the binary arithmetic logic circuit.
3. Apply knowledge in Combinational Logic Problem formulation and verify their functionalities.
4. Distinguish and examine the design capability in synchronous and asynchronous sequential circuits like flip flops, Shift registers, and counters.
5. Evaluate the basic understanding of digital circuits and to verify their operation.
6. Create more complex digital systems, including memory architectures.

Acknowledgements:

The content we have utilized is solely educational in nature. The copyright proprietors of the materials reproduced in this book have been tracked down as much as possible. The editors apologize for any violation that may have happened, and they will be happy to rectify any such material in later versions of this book.

Digital Electronics Lab

1. Basic Logic Gates

Q: Design and implement a circuit using AND, OR, and NOT gates to realize the Boolean function $F(A,B,C)=(A\cdot B)+\underline{C}$. Draw the truth table and verify the output.

A: Start by understanding the given Boolean function $F(A,B,C)=(A\cdot B)+\underline{C}$. This function uses AND, OR, and NOT gates. First, create the truth table for all possible combinations of inputs A, B, and C.

A	B	C	A·B	$\underline{C}$	F(A, B, C)
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

Next, draw the circuit. Start with AND gate, which takes inputs A and B. The output of the AND gate (A·B) goes to one input of the OR gate. Use a NOT gate to invert input C, producing $\underline{C}$, which is the other input to the OR gate. The output of OR gate is the function F.

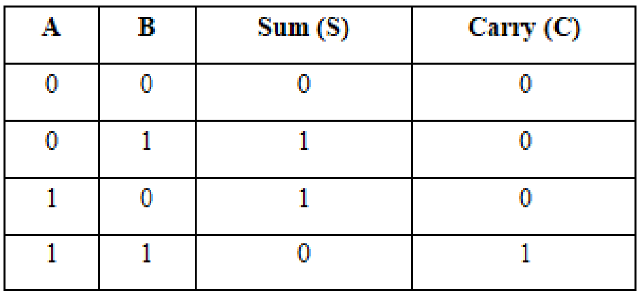
Finally, implement this circuit on a breadboard or using simulation software like Logisim. Connect the inputs A, B, and C to switches or logic input devices. Verify the output of the circuit for each combination of inputs and compare it with the truth table. This practical helps reinforce understanding of basic Boolean algebra and the functioning of basic logic gates.

2. Half Adder

Q: Design and implement a half adder circuit using XOR and AND gates. Verify the outputs for the SUM and Carry, also draw the truth table.

A: A half adder adds two single-bit binary numbers (A and B) and produces carry(C) and sum (S). The Boolean expressions for the sum and carry are $S=A\oplus B$ and $C=A\cdot B$.

The truth table for the half adder:



A	B	Sum (S)	Carry (C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Draw the circuit. Use an XOR gate for the sum (S = A ⊕ B) and an AND gate for the carry (C = A · B). Connect the inputs A and B to both gates. The output of the AND gate will be the carry, and the output XOR gate will be the sum.

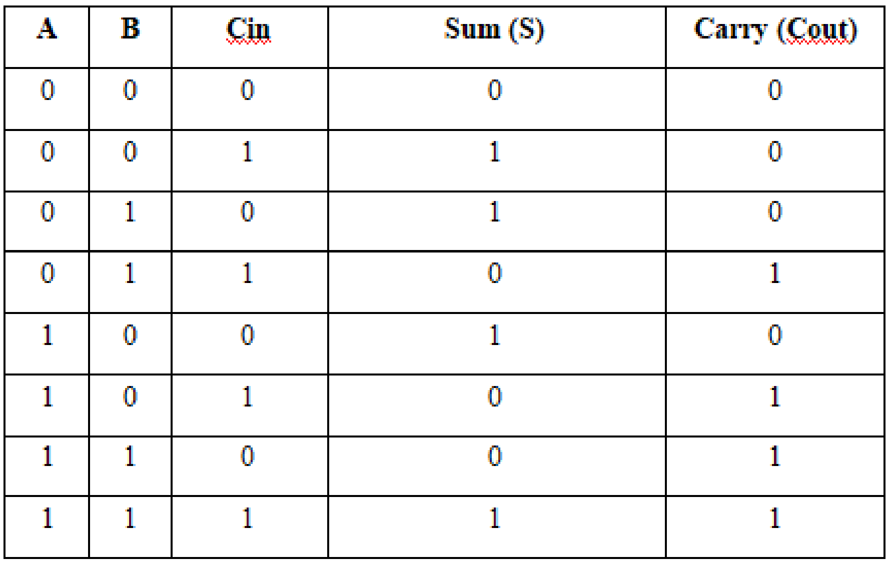
Implement this circuit on a breadboard or with simulation software. Use switches or logic inputs for A and B, and connect LEDs or output devices to the sum and carry outputs. Verify the outputs by setting different combinations of A and B and comparing the results with the truth table. This practical demonstrates the fundamental principles of binary addition and introduces students to combinational logic circuits.

3. Full Adder

Q: Design and implement a full adder circuit using an OR gate and two half adders. Draw the truth table and verify the sum and carry outputs.

A: A full adder adds three binary digits (A, B, and Cin) and produces a sum (S) and a carry-out (Cout). The Boolean expressions are $S=(A\oplus B)\oplus Cin$ and $Cout=(A\cdot B)+(Cin\cdot(A\oplus B))$.

The truth table for the full adder:



A	B	Cin	Sum (S)	Carry (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

Draw the circuit. First, make two half adders. The first half adder adds A and B, resulting in a carry (C1) and an intermediate sum (S1). S1 and Cin are added by the second half adder, generating the final sum (S) and an additional carry (C2). Finally, use an OR gate to combine the two carry outputs (Cout = C1 + C2).

Implement this circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A, B, and Cin, and comparing the results with the truth table. This practical helps student understands the concept of hierarchical design in digital circuits, where complex circuits are built from simpler ones.

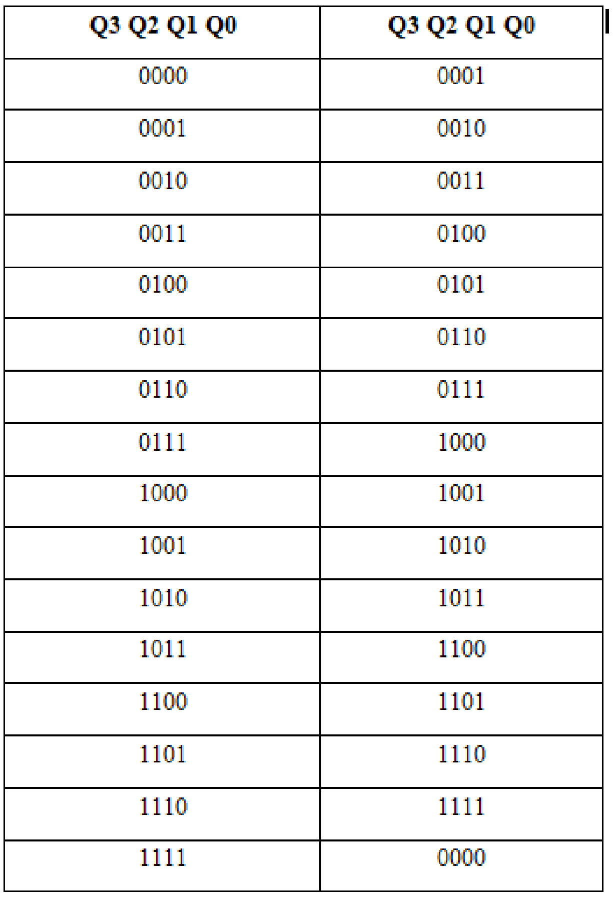
4. 4-Bit Binary Counter Using Flip-Flops

Q: Design and implement a 4-bit binary counter using JK flip-flops. Draw the state transition table and verify the counter operation.

A: A 4-bit binary counter uses flip-flops to count from 0 to 15 in binary. JK flip-flops are ideal because they can toggle states based on input conditions. Four JK flip-flops should be connected in series, with each flip-flop's output acting as the subsequent one's clock input.

Create the state transition table:

| Present State | Next State |



Q3 Q2 Q1 Q0	Q3 Q2 Q1 Q0
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

Connect the J and K inputs of each flip-flop to logic high (1) to enable toggling. The clock input for the first flip-flop receives the external clock signal. The output (Q) of each flip-flop connects to the input of the next, forming a ripple counter.

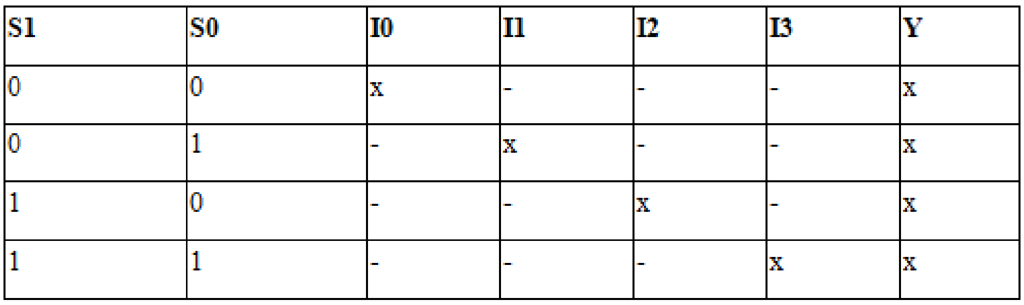
Implement this on a breadboard or using simulation software. Verify the counter's operation by applying a clock signal and observing the flip-flops' outputs. The outputs should represent the binary count sequence from 0000 to 1111. This practical demonstrates sequential logic design and the application of flip-flops in counters.

5. 4-to-1 Multiplexer

Q: Design and implement a 4-to-1 multiplexer using basic logic gates. Draw the truth table and verify the output.

A: A 4-to-1 multiplexer selects one of four input lines (I0, I1, I2, I3) based on two select lines (S0, S1) and outputs the selected input. The Boolean expression for the output Y is $Y=(\overline{S0}\cdot\overline{S1}\cdot I0)+(\overline{S0}\cdot S1\cdot I1)+(S0\cdot\overline{S1}\cdot I2)+(S0\cdot S1\cdot I3)Y=(S0\cdot S1\cdot I0)+(S0\cdot S1\cdot I1)+(S0\cdot S1\cdot I2)+(S0\cdot S1\cdot I3)$.

Create the truth table:



S1	S0	I0	I1	I2	I3	Y
0	0	x	-	-	-	x
0	1	-	x	-	-	x
1	0	-	-	x	-	x
1	1	-	-	-	x	x

Draw the circuit. Use AND gates to implement each product term and an OR gate to combine them. For example, the first term S0· S1· I0 uses two NOT gates to invert S0 and S1, followed by an AND gate with I0.

Implement this on a breadboard or using simulation software. Verify the output by setting different combinations of S0 and S1, and applying inputs to I0, I1, I2, and I3. The output should match the selected input based on the truth table. This practical illustrates how multiplexers function and their application in digital systems.

6. D Flip-Flop Operation

Q: Design and implement a D flip-flop using NAND gates. Draw the truth table and verify the outputs for various inputs.

A: A D flip-flop captures the value of the D input at the moment the clock (CLK) transitions from low to high (positive edge-triggered) and holds that value until the next clock edge. The output (Q) changes based on the D input only at the clock's rising edge.

Create the truth table:

CLK	**D**	**Q (Next)**	**$\bar{Q}$ (Next)**
↑	0	0	1
↑	1	1	0
0	0	Q	$\bar{Q}$
0	1	Q	$\bar{Q}$

To design a D flip-flop using NAND gates:

1. Create a SR latch using NAND gates.
2. Modify the SR latch to accept a single D input.
3. Connect the D input directly to the S input of the SR latch and through an inverter to the R input.

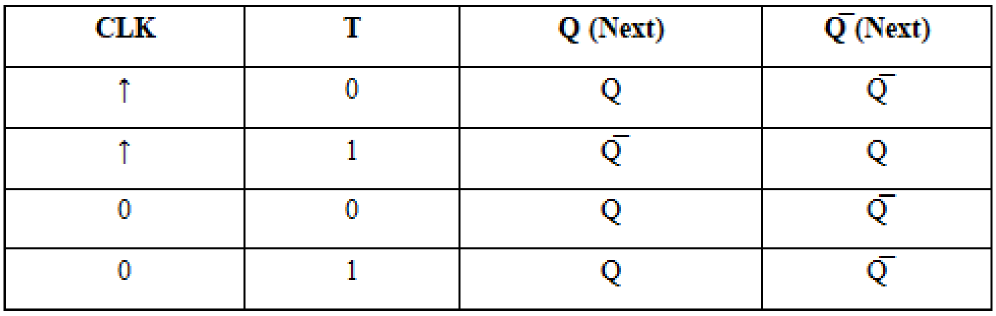
Connect the clock signal (CLK) to the enable inputs of the latch. Implement this circuit on a breadboard or using simulation software like Multisim. Verify the operation by applying a clock signal and varying the D input. Observe and record the Q and Q̄ outputs. This practical helps understand how flip-flops store binary data and the role of the clock signal in synchronous circuits.

7. T Flip-Flop Operation

Q: Design and implement a T flip-flop using JK flip-flops. Draw the truth table and verify the output for various inputs.

A: When T input is high (1), T flip-flop toggles its output on each clock pulse. If T is low (0), the output remains unchanged.

Create the truth table:



CLK	T	Q (Next)	$\overline{Q}$ (Next)
↑	0	Q	$\overline{Q}$
↑	1	$\overline{Q}$	Q
0	0	Q	$\overline{Q}$
0	1	Q	$\overline{Q}$

To design a T flip-flop using JK flip-flops:

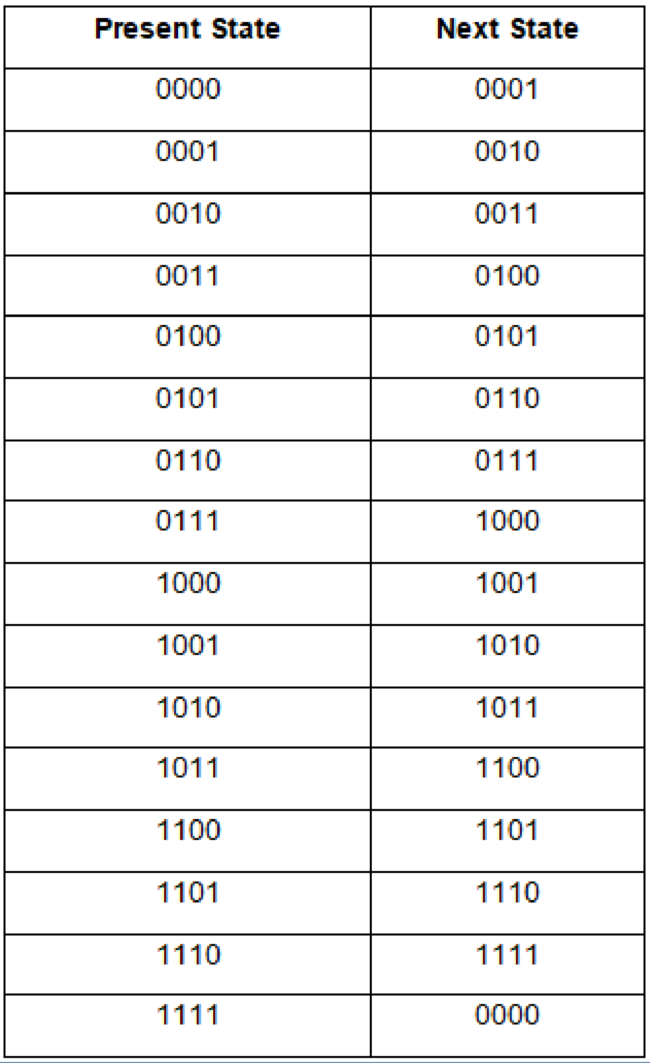
1. Connect the J and K inputs of the JK flip-flop together and label this combined input as T.
2. When T is 1, the JK flip-flop toggles its output with each clock pulse.
3. When T is 0, the output remains the same.

Implement this circuit on a breadboard or using simulation software. Verify the operation by applying a clock signal and varying the T input. Observe and record the Q and $\overline{Q}$ outputs. This practical illustrates the concept of toggle operation and the versatility of JK flip-flops in implementing various flip-flop configurations.

8. **Synchronous 4-Bit Counter**

Q: Design and implement a synchronous 4-bit counter using T flip-flops. Draw the state transition table and verify the counter operation.

A: A synchronous 4-bit counter counts from 0 to 15 in binary and then resets to 0. An identical clock signal triggers each flip-flop, guaranteeing simultaneous transitions. Create the state transition table:



Present State	Next State
0000	0001
0001	0010
0010	0011
0011	0100
0100	0101
0101	0110
0110	0111
0111	1000
1000	1001
1001	1010
1010	1011
1011	1100
1100	1101
1101	1110
1110	1111
1111	0000

Connect four T flip-flops in series, with each flip-flop's output driving the next flip-flop's T input. The first flip-flop's T input is connected to a logic high (1). The clock signal is connected to all flip-flops simultaneously.

Implement this circuit on a breadboard or using simulation software. Verify the counter's operation by applying a clock signal and observing the outputs. The outputs should represent a binary count sequence from 0000 to 1111. This practical demonstrates the principles of synchronous counters and their use in digital systems.

9. Multiplexer Implementation

Q: Design and implement an 8-to-1 multiplexer using 2-to-1 multiplexers. Draw the circuit diagram and verify the output for various inputs.

A: An 8-to-1 multiplexer selects one of eight inputs (I0 to I7) based on three select lines (S0, S1, S2) and outputs the selected input.

Use 2-to-1 multiplexers to construct an 8-to-1 multiplexer:

1. Use four 2-to-1 multiplexers for the first stage, each taking two inputs (I0 and I1, I2 and I3, I4 and I5, I6 and I7) and controlled by S0.
2. Use two 2-to-1 multiplexers for the second stage, each taking the output of two first-stage multiplexers and controlled by S1.
3. Use one 2-to-1 multiplexer for the final stage, taking the output of the second-stage multiplexers and controlled by S2.

Draw the circuit diagram and implement it on a breadboard or using simulation software. Verify the output by setting different combinations of S0, S1, and S2, and applying inputs to I0 to I7. The output should match the selected input based on the select lines. This practical helps understand the hierarchical design and implementation of complex multiplexers using simpler components.

10. Demultiplexer Implementation

Q: Design and implement a 1-to-8 demultiplexer using 1-to-2 demultiplexers. Draw the circuit diagram and verify the outputs for various inputs.

A: A 1-to-8 demultiplexer routes a single input (I) to one of eight outputs (Y0 to Y7) based on three select lines (S0, S1, S2).

Use 1-to-2 demultiplexers to construct a 1-to-8 demultiplexer:

1. Use one 1-to-2 demultiplexer for the first stage, controlled by S2, splitting the input I into two intermediate signals.
2. Use two 1-to-2 demultiplexers for the second stage, each taking an intermediate signal and controlled by S1.
3. Use four 1-to-2 demultiplexers for the final stage, each taking an output from the second stage and controlled by S0.

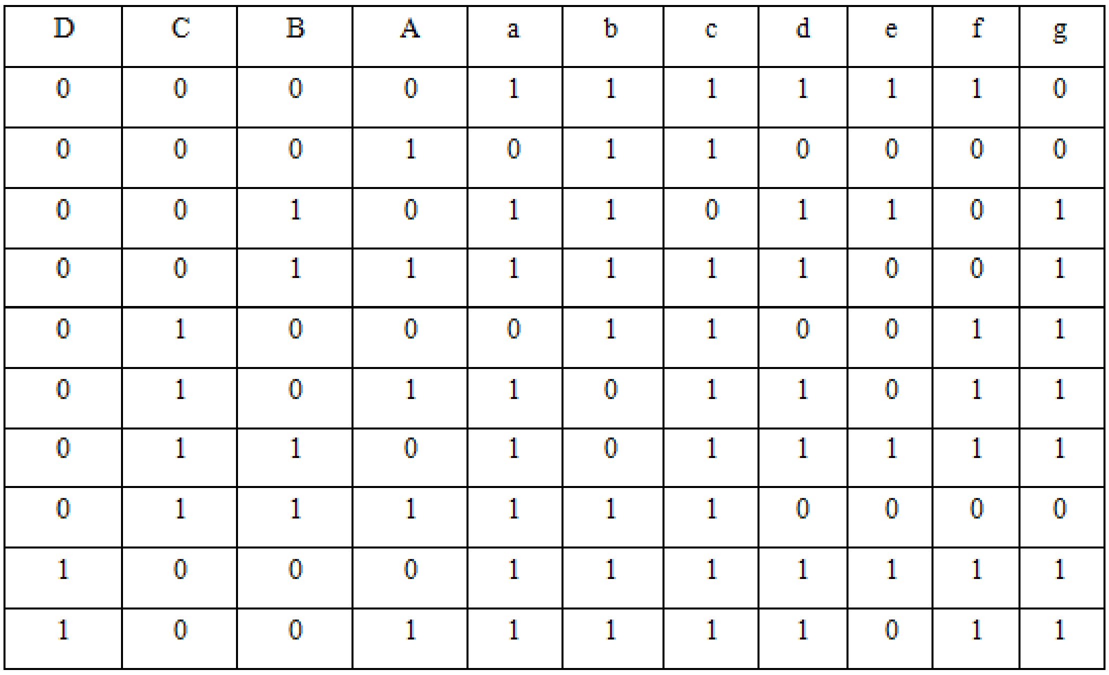
Draw the circuit diagram and implement it on a breadboard or using simulation software. Verify the outputs by setting different combinations of S0, S1, and S2, and applying input I. The active output should correspond to the selected path. This practical illustrates the implementation of complex demultiplexers and their use in digital systems.

11. BCD to 7-Segment Decoder

Q: Design and implement a BCD (Binary-Coded Decimal) to 7-segment display decoder using logic gates. Verify the output for each BCD input.

A: A BCD to 7-segment decoder converts a BCD input (0000 to 1001) to the corresponding 7-segment display signals (a to g). Each segment is controlled by a specific combination of BCD inputs.

Create the truth table for each segment (a to g) based on BCD inputs (D, C, B, A):



D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1

Design the logic circuits for each segment using AND, OR, and NOT gates based on the truth table. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different BCD inputs and observing the 7-segment display. This practical helps understand the conversion of BCD to visual representation on 7-segment displays.

12. 4-Bit Comparator

Q: Design and implement a 4-bit comparator using logic gates. Draw the truth table and verify the output for various inputs.

A: A 4-bit comparator compares two 4-bit binary numbers (A3 A2 A1 A0) and (B3 B2 B1 B0) and outputs whether one number is equal to, less than, and greater than.

Create the truth table for the comparator:

A3 A2 A1 A0	B3 B2 B1 B0	A > B	A = B	A < B
0000	0000	0	1	0
0001	0000	1	0	0
0010	0000	1	0	0
...	...	...	...	...
1110	1111	0	0	1
1111	1111	0	1	0

Design the circuit using AND, OR, and NOT gates:

1. Compare each bit of A and B using XOR gates to determine if they are equal.
2. Use AND gates to determine if one number is greater or less than the other.
3. Combine the outputs using OR gates to generate the final comparison signals.

Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A and B, and observing the comparator outputs. This practical demonstration demonstrates the design and operation of digital comparators, which are fundamental components in digital systems.

13. 4-Bit Shift Register

Q: Design and implement a 4-bit shift register using D flip-flops. Draw the timing diagram and verify the operation for various inputs.

A: A 4-bit shift register stores 4 bits of data and shifts the bits left or right on each clock pulse. Use four D flip-flops connected in series, where each flip-flop's output serves as the input for the next flip-flop.

Create the timing diagram:

CLK	D	Q3	Q2	Q1	Q0
↑	1	0	0	0	0
↑	1	1	0	0	0
↑	0	1	1	0	0
↑	1	0	1	1	0
↑	0	1	0	1	1

Connect the D input to the first flip-flop and the output of each flip-flop to the input of the next. Apply the clock signal to all flip-flops simultaneously.

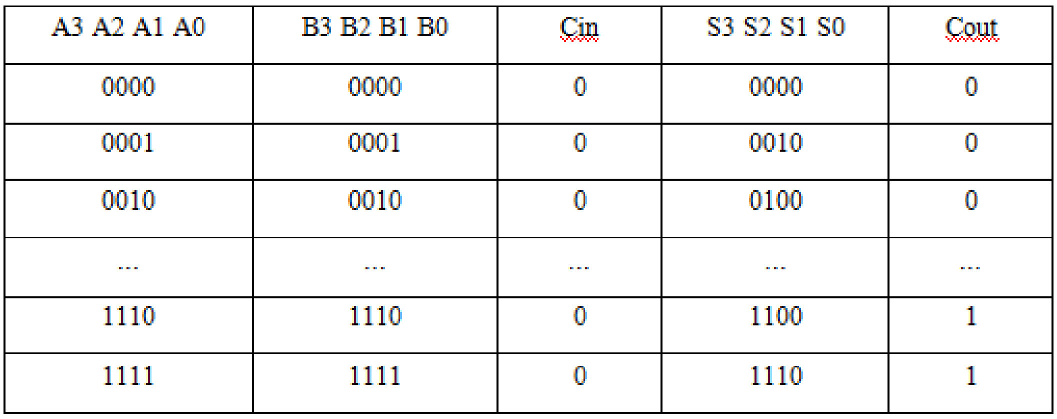
Implement this circuit on a breadboard or using simulation software. Verify the operation by applying different input sequences and observing the outputs. The timing diagram should match the expected shifting behavior. This practical illustrates the operation of shift registers and their application in data storage and transfer.

14. 4-Bit Ripple Carry Adder

Q: Design and implement a 4-bit ripple carry adder using full adders. Draw the truth table and verify the sum and carry outputs for various inputs.

A: A 4-bit ripple carry adder adds two 4-bit binary numbers (A3 A2 A1 A0) and (B3 B2 B1 B0) and produces a carry-out (Cout) and 4-bit sum (S3 S2 S1 S0).

Create the truth table for each full adder stage:



A3 A2 A1 A0	B3 B2 B1 B0	Cin	S3 S2 S1 S0	Cout
0000	0000	0	0000	0
0001	0001	0	0010	0
0010	0010	0	0100	0
...	...	...	...	...
1110	1110	0	1100	1
1111	1111	0	1110	1

Connect four full adders in series, with the carry-out of each adder connected to the carry-in of the next. The first adder's carry-in is set to 0.

Implement this circuit on a breadboard or using simulation software. Verify the sum and carry outputs by setting different combinations of A and B, and comparing the results with the truth table. This practical demonstrates the design of multi-bit adders and the concept of carry propagation in digital arithmetic.

15. Ring Counter

Q: Design and implement a 4-bit ring counter using D flip-flops. Draw the state transition table and verify the counter operation.

A: A ring counter is a type of counter composed of a circular shift register. It cycles through a predefined sequence of states. Utilize four D flip-flops interconnected in series, with the first flip-flop's input and the last flip-flop's output connected.

Create the state transition table:

CLK	Q3	Q2	Q1	Q0
↑	1	0	0	0
↑	0	1	0	0
↑	0	0	1	0
↑	0	0	0	1
↑	1	0	0	0

Initialize the flip-flops to a state where only one flip-flop is set to 1, and the others are 0. Connect the output of each flip-flop to the input of the next flip-flop in a circular fashion.

Implement this circuit on a breadboard or using simulation software. Verify the counter's operation by applying a clock signal and observing the outputs. The outputs should represent the predefined sequence of states. This practical illustrates the operation of ring counters and their application in digital systems.

16. Johnson Counter

Q: Design and implement a 4-bit Johnson counter using D flip-flops. Draw the state transition table and verify the counter operation.

A: A Johnson counter is a type of counter composed of a shift register where the complement of the output of the last flip-flop is fed back to the input of the first flip-flop.

Create the state transition table:

CLK	Q3	Q2	Q1	Q0
↑	0	0	0	0
↑	1	0	0	0
↑	1	1	0	0
↑	1	1	1	0
↑	1	1	1	1
↑	0	1	1	1
↑	0	0	1	1
↑	0	0	0	1

Connect four D flip-flops in series, with the complement of the last flip-flop's output connected to the input of the first flip-flop.

Implement this circuit on a breadboard or using simulation software. Verify the counter's operation by applying a clock signal and observing the outputs. The outputs should represent the predefined sequence of states. This practical demonstrates the operation of Johnson counters and their application in digital systems.

17. Binary to Gray Code Converter

Q: Design and implement a binary to Gray code converter using logic gates. Draw the truth table and verify the output for various inputs.

A: A binary to Gray code converter converts a binary number to its corresponding Gray code representation. The Gray code is a binary numeral system where two successive values differ in only one bit.

Create the truth table:

Binary	**Gray**
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

The Boolean expressions for converting 4-bit binary to Gray code are:

- G3 = B3
- G2 = B3 $\oplus$B2
- G1 = B2 $\oplus$B1
- G0 = B1 $\oplus$B0

Design the logic circuits for each Gray code bit using XOR gates based on the expressions. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different binary inputs and comparing the results with the truth table. This practical helps understand the conversion between binary and Gray code, which is used in error correction and digital communication.

18. Gray Code to Binary Converter

Q: Design and implement a Gray code to binary converter using logic gates. Draw the truth table and verify the output for various inputs.

A: A Gray code to binary converter converts a Gray code number to its corresponding binary representation.

Create the truth table:

Gray	**Binary**
0000	0000
0001	0001
0010	0011
0011	0010
0100	0110
0101	0111
0110	0101
0111	0100
1000	1100
1001	1101
1010	1111
1011	1110
1100	1010
1101	1011
1110	1001
1111	1000

The Boolean expressions for converting 4-bit Gray code to binary are:

- $B3 = G3$
- $B2 = G3 \oplus G2$
- $B1 = B2 \oplus G1$
- $B0 = B1 \oplus G0$

Design the logic circuits for each binary bit using XOR gates based on the expressions. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different Gray code inputs and comparing the results with the truth table. This practical helps understand the conversion between Gray code and binary, which is essential in digital systems and coding theory.

19. BCD Adder

Q: Design and implement a BCD adder using 4-bit binary adders and logic gates. Verify the output for various BCD inputs.

A: A BCD adder adds two BCD numbers and produces a BCD sum and a carry-out. A BCD number is represented in binary but only uses digits 0 to 9 (0000 to 1001).

The circuit involves:

1. Adding two 4-bit BCD numbers using a 4-bit binary adder.
2. Checking if the sum is greater than 9 (1001 in binary) to adjust the result.
3. Adding 6 (0110) to the sum if it exceeds 9 to convert it back to a valid BCD number.

Create the truth table:

A	B	Sum	Carry	Adjusted Sum	BCD Carry
0000	0000	0000	0	0000	0
0001	0001	0010	0	0010	0
0101	0101	1010	0	0000	1
...	...	...	...	...	...
1001	1001	10010	1	1000	1

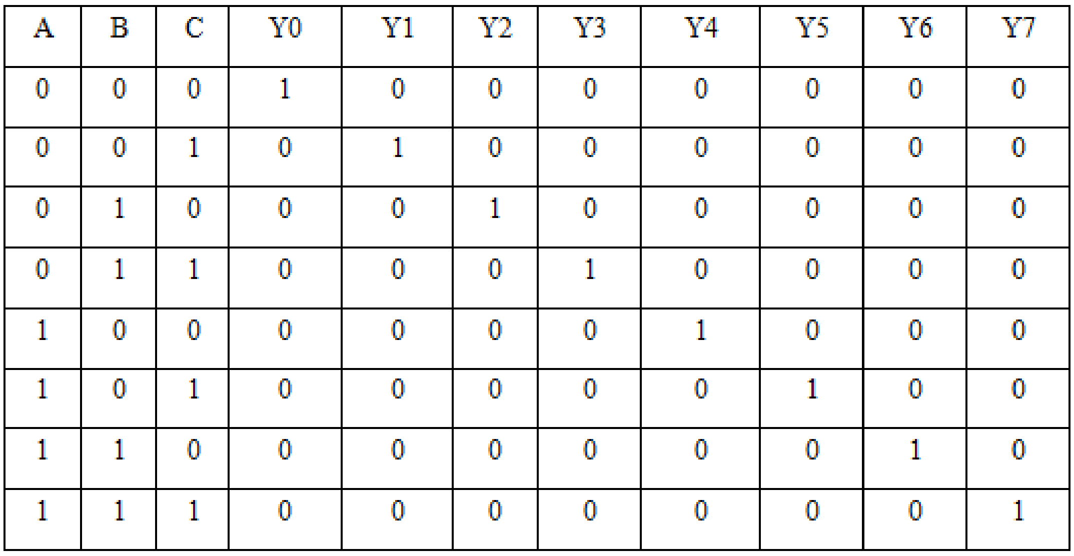
Design the circuit using two 4-bit binary adders and additional logic gates for the adjustment. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different BCD inputs and comparing the results with the truth table. This practical demonstrates the design of BCD arithmetic circuits, which are used in digital calculators and other applications.

20. 3-to-8 Decoder

Q: Design and implement a 3-to-8 decoder using basic logic gates. Draw the truth table and verify the output for various inputs.

A: A 3-to-8 decoder converts a 3-bit binary input (A, B, C) into one of eight outputs (Y0 to Y7), with only one output active at a time.

Create the truth table:



A	B	C	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Design the logic circuits for each output using AND gates and NOT gates based on the truth table. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A, B, and C, and comparing the results with the truth table. This practical helps understand the design and operation of decoders, which are essential components in digital systems for addressing and data routing.

21. Binary Multiplier

Q: Design and implement a 2-bit binary multiplier using logic gates. Draw the truth table and verify the output for various inputs.

A: A 2-bit binary multiplier multiplies two 2-bit binary numbers (A1 A0) and (B1 B0) and produces a 4-bit product (P3 P2 P1 P0).

Create the truth table:

A1	A0	B1	B0	P3 P2 P1 P0
0	0	0	0	0000
0	0	0	1	0000
0	0	1	0	0000
0	0	1	1	0000
0	1	0	0	0000
0	1	0	1	0001
0	1	1	0	0010
0	1	1	1	0011
1	0	0	0	0000
1	0	0	1	0010
1	0	1	0	0100
1	0	1	1	0110
1	1	0	0	0000
1	1	0	1	0011
1	1	1	0	0110
1	1	1	1	1001

Design the circuit using AND gates to generate partial products, and use half adders and full adders to combine these products.

mplement the circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A1, A0, B1, and B0, and comparing the results with the truth table. This practical illustrates the design of binary multipliers, which are fundamental components in digital arithmetic and signal processing.

22. 4-Bit Binary Subtractor

Q: Design and implement a 4-bit binary subtractor using full adders and logic gates. Draw the truth table and verify the output for various inputs.

A: A 4-bit binary subtractor subtracts two 4-bit binary numbers (A3 A2 A1 A0) and (B3 B2 B1 B0) and produces a 4-bit difference (D3 D2 D1 D0) and a borrow-out (Bout).

Create the truth table for each full adder stage in a subtractor:

A3 A2 A1 A0	B3 B2 B1 B0	Bin	D3 D2 D1 D0	Bout
0000	0000	0	0000	0
0001	0000	0	0001	0
0010	0001	0	0001	0
...	...	...	...	...
1110	0110	0	1000	0
1111	1111	0	0000	0

Design the circuit using four full adders, where the B inputs are inverted and an additional logic gate is used to handle the borrow-in and borrow-out logic.

Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A and B, and comparing the results with the truth table. This practical demonstrates the design of binary subtractors, essential for digital arithmetic operations.

23. 4-Bit Magnitude Comparator

Q: Design and implement a 4-bit magnitude comparator using logic gates. Draw the truth table and verify the output for various inputs.

A: A 4-bit magnitude comparator compares two 4-bit binary numbers (A3 A2 A1 A0) and (B3 B2 B1 B0) and outputs whether one number is less than, greater than or equal to the other.

Create the truth table for the comparator:

A3 A2 A1 A0	B3 B2 B1 B0	A > B	A = B	A < B
0000	0000	0	1	0
0001	0000	1	0	0
0010	0000	1	0	0
...	...	...	...	...
1110	1111	0	0	1
1111	1111	0	1	0

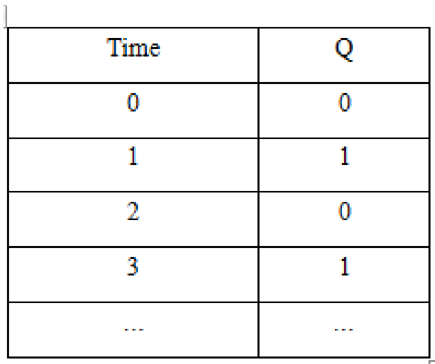
Design the circuit using AND, OR, and NOT gates to implement the comparator logic. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A and B, and comparing the results with the truth table. This practical demonstrates the design and operation of magnitude comparators in digital systems.

24. Ring Oscillator

Q: Design and implement a ring oscillator using inverters. Draw the timing diagram and verify the oscillation frequency.

A: A ring oscillator consists of an odd number of inverters connected in a loop, creating an oscillating output signal due to the propagation delay of the inverters.

Create the timing diagram for a 3-inverter ring oscillator:



Time	Q
0	0
1	1
2	0
3	1
...	...

Connect three inverters in a loop, with the output of the last inverter connected to the input of the first inverter.

Implement the circuit on a breadboard or using simulation software. Verify the oscillation by observing the output with an oscilloscope or simulation tool. Measure the oscillation frequency and compare it with the theoretical value based on the inverter delay. This practical illustrates the principles of ring oscillators, which are used in clock generation and timing applications.

25. 4-to-16 Decoder

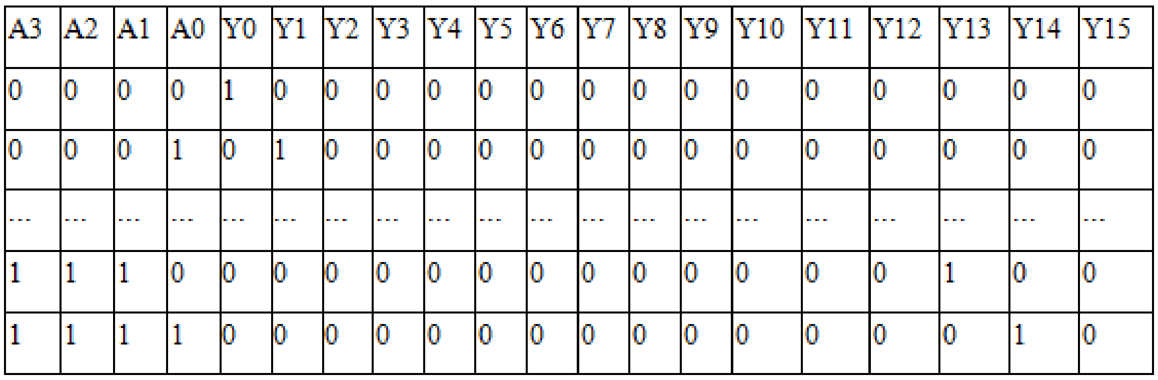
Q: Design and implement a “4-to-16” decoder using two “3-to-8” decoders. Draw the circuit diagram and verify the output for various inputs.

A: A “4-to-16” decoder converts a 4-bit binary input (A3 A2 A1 A0) into one of sixteen outputs (Y0 to Y15), with only one output active at a time.

Use two 3-to-8 decoders to construct a 4-to-16 decoder:

1. Use the first decoder to decode the lower three bits (A2, A1, A0) and generate eight intermediate signals.
2. Use the fourth bit (A3) to enable one of the two decoders, allowing the selection of outputs Y0 to Y7 or Y8 to Y15.

The 4-to-16 decoder truth table:



A3	A2	A1	A0	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8	Y9	Y10	Y11	Y12	Y13	Y14	Y15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0

Design the circuit using two 3-to-8 decoders and additional logic gates for enabling the correct decoder based on A3. Implement the circuit on a breadboard or using simulation software. Verify the outputs by setting different combinations of A3, A2, A1, and A0, and comparing the results with the truth table. This practical helps understand the design and implementation of larger decoders using smaller ones.