EXPERIMENT 1

LOGIC GATES AND BOOLEAN ALGEBRA

1. OBJECTIVE

Gain experience in truth table and Boolean algebra.

2. THEORY

2.1 AND GATE

AND gate implements the Boolean ***AND*** function where the output only is logical 1 when all inputs are logical 1. The standard symbol and the truth table for an ***AND*** gate with two inputs is given below.

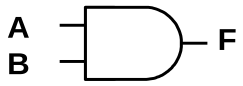


Figure 1 ***AND*** Gate symbol

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

Table 1 Truth table of ***AND*** Gate

The Boolean expression for the ***AND*** gate is F = A.B

2.2 OR GATE

OR gate implements the Boolean ***OR*** function where the output is logical 1 when at least one input is logical 1. The standard symbol and the truth table for an ***OR*** gate with two inputs is given below.

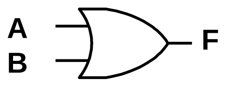


Figure 2 ***OR*** Gate

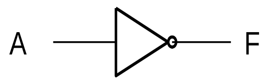
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

Table 2 Truth table of ***OR*** Gate

The Boolean expression for the ***OR*** gate is $F = A + B$.

2.3 NOT GATE

NOT gate implements the Boolean ***NOT*** function where the output is the inverse of the input. The standard symbol and the truth table for the ***NOT*** gate is given below.



A	F
0	1
1	0

Figure 3 ***NOT*** Gate

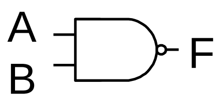
Table 3 Truth table of ***NOT*** Gate

The Boolean expression for the ***NOT*** gate is $F = \overline{A}$

From these three basic logical gates it's to possible implement any Boolean expression into hardware.

2.4 NAND GATE

NAND gate is an ***AND*** gate followed by a ***NOT*** gate. The output is logical 1 when one of the inputs is logical 0. The standard symbol and the truth table for the ***NAND*** gate is given below.



A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

Figure 4 ***NAND*** Gate

Table 4 Truth table of ***NAND*** Gate

The Boolean expression for the ***NAND*** gate is $F = \overline{A \cdot B}$.

2.5 NOR GATE

NOR is a combination of an ***OR*** followed by a ***NOT*** gate. The output is logical 1 when all of the inputs are logical 0.The standard symbol and the truth table for the ***NOR*** gate is given below.

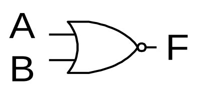


Figure 5 ***NAND*** Gate

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Table 5 Truth table of ***NAND*** Gate

The Boolean expression for the ***NOR*** gate is $R = \overline{A+B}$.

2.6 TRUTH TABLE

A truth table is a list of all the possible inputs and the corresponding outputs for a given system. The amount of possible inputs is determined by the amount of input variables. This value can be obtained by the formula 2^n.

Example: Determine the truth table for: Y=ABC+A'BC.

The indicated Boolean equation would produce a high output when A=1,B=1,C=1 or A=0.B=1,C=1 All other input combinations would produce a low output. The truth table for the expression is shown below.

Truth Table			
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 6 Truth Table

2.7 BUILDING A CIRCUIT

In order to build a circuit from a truth table, we must first determine the Boolean expression for that particular truth table. Then the circuit can be constructed from multiple ***AND*** gates whose outputs all tie into one multiple input ***OR*** gate.

3. PRELIMINARY WORK

Design a circuit that has the truth table shown below.

Truth Table			
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 7: Truth Table for preliminary work

EXPERIMENTAL PROCEDURE

Equation.1 F = A+BC+AC

Equation.2 F = A+AB

1) Complete the Truth Table of equation.1
2) Implement equation 1 by using minimum amount of logic gates.
3) Repeat step 2 for equation 2.

Equipment List:

1) 74LS32 TTL ***OR*** GATE IC
2) 74LS08 TTL ***AND*** GATE IC
3) Standard set equipments

EXPERIMENT 2

COMBINATIONAL LOGIC CIRCUITS AND KARNOUGH MAP

1. OBJECTIVE

To gain experience in designing logic circuits and Karnough map

2. THEORY

Simplification of logic circuits is a responsibility of the designer. Simpler circuits are generally more economic and more reliable.

2.1 SUM OF PRODUCT FORM (SOP)

The sum of product form of a logic circuit output looks like the following examples:

$$F(A,B,D) = A\overline{B}D + A\overline{B}\overline{D}$$
$$F(A,B,C,D,E,F,H) = \overline{A}B + C\overline{D} + EF + \overline{H}F$$

Logic equations may also be simplified using Boolean algebra or Karnough map. Both types of simplification will be covered the logic equations shown in the above examples are called “minterm” expressions.

Minterm expressions are logical equations where logical sum operator separates the logical product terms. Minterm expressions are also called sum of product expressions.

2.2 DESIGNING COMBINATIONAL CIRCUIT

Logic design begins with a problem statement. The problem statement is analyzed and translated into logic variable inputs. A truth table is then constructed to show when a logic one output is to be produced. Next a SOP (minterm) logic equation is then produced. Then a circuit is drawn from the SOP logic equation.

2.3 THE KARNOUGH MAP

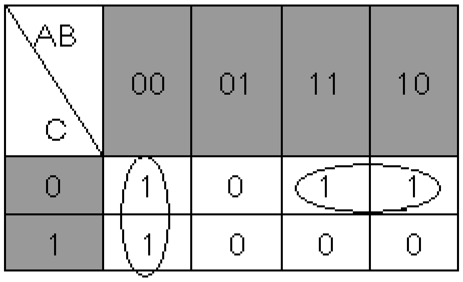
A Karnough map or K-map technique is a graphical device to simplify logic equations or the output of truth tables following a simple orderly process.

A K-map like a truth table displays the relationship between input variables and the desired or true output of logic expression on the truth table.

EXAMPLE: Simplify the logical equation by using K-map $F(A,B,C) = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + AB\overline{C}$

Truth Table			
A	B	C	F
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Table 1. Truth table of logical function



AB \ C	00	01	11	10
0	1	0	1	1
1	1	0	0	0

Fig 3.1 K-map of the function

After using K-map Simplified logical expression is $F = A\overline{C} + \overline{A}\,\overline{B}$

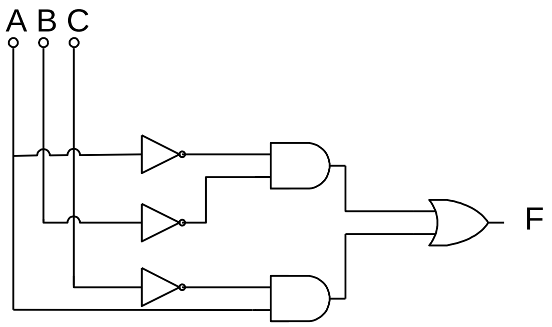


Fig 1. Logic Circuit

3. PRELIMINARY WORK

An alarm is to be used in automated ink bottling plant.

A conveyer belt carries the empty inkbottles past the filling spout. The alarm is to sound if any of the following conditions occur:

A) If there is no ink and bottle in the same time
B) When the user push the emergency button

I = Ink in the tank

B = Bottles on the conveyer belt

E = Emergency button

E	**I**	**B**	**Alarm**

1) Complete the Truth Table of problem
2) Simplify the logical expression of the problem by using K-map

4. EXPERIMENTAL PROCEDURE

Implement simplified logical expression in preliminary work using logic gates

Equipment List:

1) 74LS32 TTL OR GATE IC
2) 74LS08 TTL AND GATE IC
3) 74LS04 TTL NOT GATE IC
4) Standard set equipments

EXPERIMENT 3

HALF ADDER AND FULL ADDER

1. OBJECTIVE

To gain experience in logic circuits and Adders.

2. THEORY

2.1 HALF ADDER

A half adder is a digital logic circuit with two input terminals and two output terminals. The output terminals are called the sum and carry outputs. The sum output of a half-adder circuit is the ***exclusive OR*** (***XOR***) function of the two inputs. That is, the sum output is 0 when the inputs are the same and 1 when they are different. The carry output is the ***AND*** function of the two inputs. It is 1 only when both inputs are 1.

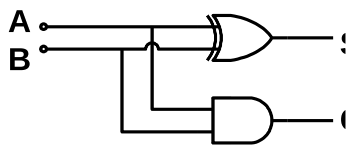


Figure 1. Implementation of a half-adder.

2.2 FULL ADDER

Just like a half-adder, the full adder is also a digital logic circuit. The full adder differs from the half adder in that the full adders consider carry bits from previous stages.

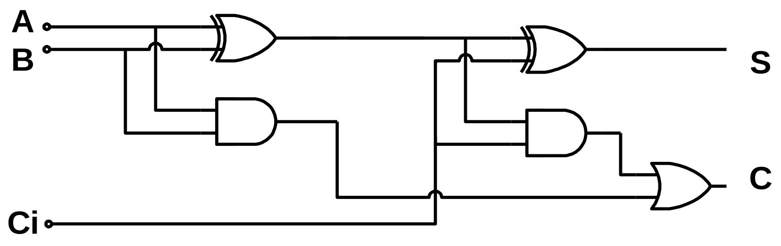


Figure 2. Implementation of a Full-adder.

Suppose that we wish to add two single bit binary numbers A and B. In doing this we have to consider two things:

- What is the sum of the two numbers within the column?
- Do we need to carry a number into the next column?

$$\boxed{\begin{array}{c} A \\ \underline{B} \\ ?\,? \end{array}}$$

It is helpful to construct a truth table for these two operations.
For the sum of the numbers in the column itself the truth table is as shown on the left below. The answer within the column is 1 if only one of A or B is 1 but is zero if neither or both of A and B are

Inputs		*Same column*
A	***B***	***Q***
0	0	0
0	1	1
1	0	1
1	1	0

Table 1

Table1 is the truth table of an ***XOR*** gate. Thus if A and B form the two inputs of an ***XOR*** gate, the output will be the correct answer for the sum of the two numbers within the same column. The truth table that says whether a 1 need to be carried into the next column is as below. Clearly this only needs to happen if both A and B are 1. The truth table therefore is that of an ***AND*** gate. An ***AND*** gate can be used to carry a bit into the next column if required

Inputs		*Carry*
A	***B***	***Q***
0	0	0
0	1	0
1	0	0
1	1	1

Table 2

A suitable logic circuit, which will perform this simple addition of two single bit numbers, is shown below. The circuit is known as a ***half-adder***. This is because it can only be used to add the LSB of two numbers, as it includes no mechanism for also adding a carried bit.

To create a full adder circuit to, say add two 2 bit binary numbers, we need to consider what may happen when we add the second column. The following are all possibilities, which must be allowed for:

We may have to add into the second column a figure that has been carried from the first column.
We may have to carry a figure into the third column because both bits in the second column are 1.
We may have to carry a figure into the third column because even though only one of the bits in the second column is 1 we have also carried a 1 from the first column.

To deal with all these possibilities requires considerably more logic gates.
The full adder circuit to add together 2 two bit binary numbers according to

$$A_2A_1 + B_2B_1 = C_3C_2C_1$$

PRELIMINARY

Implement the Boolean function $F = A\overline{B}C\overline{D} + \overline{A}BC\overline{D} + A\overline{B}\overline{C}D + \overline{A}B\overline{C}D$ with ***XOR*** and ***AND*** gates

EXPERIMENTAL PROCEDURE

1) Connect the circuit of the half adder circuit
2) Connect the circuit of the full adder circuit

Equipment List

1) 74LS32 TTL OR GATE IC
2) 74LS08 TTL AND GATE IC
3) 74LS86 TTL XOR GATE IC
4) Standard set equipment

EXPERIMENT 4

DESIGNING 2-BIT MULTIPLIER CIRCUITS

1. OBJECTIVE

To gain experience in combinational logic circuits. Circuits with both single and multiple outputs will be used.

2. THEORY

There are numerous practical applications of combinational logic circuits. Circuits can be easily designed and built to control outputs for various input combinations or to perform some mathematical or logical function.

Some circuits can be easily designed by directly writing a logical expression to solve the problem. As an example, suppose that we desired to design a circuit to turn on a light using either of two switches. If either switch is up, the light will be lit. If both switches are up or both switches are down, the light will not be lit. If we call the output ***OUT*** and the inputs from the switches ***IN1*** and ***IN2***, we can fairly easily reason that

$$OUT = IN1 \bullet \overline{IN2} + \overline{IN1} \bullet IN2 = IN1 \oplus IN2$$

As problems become more complex, a simple design procedure can be used as is listed below.

Combinational Logic Design Procedure

1. State the problem
2. Determine the required inputs and outputs
3. Assign variables to each input and output
4. Derive a truth table
5. Simplify output expressions
6. Implement each expression

Example: Design a circuit with 4 inputs that has outputs with a binary value equal to the number of inputs that are ***HIGH***. Following the Combinational Logic Design procedure above:

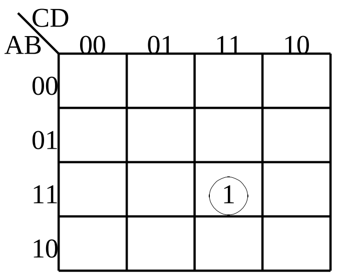
1. State the problem: Specified with the example in this case
2. Determine the required number of inputs and outputs: 4 inputs are specified. If all 4 inputs are ***HIGH***, the output code will be $(100)_2$, so 3 output bits are required.
3. Assign variables to each input and output: Call the 4 inputs ***A***, ***B***, ***C***, and ***D***. Call the 3 outputs ***E***, ***F*****,** and ***G***

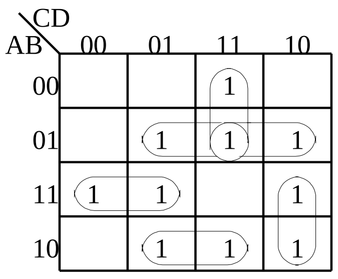
4. Truth table:

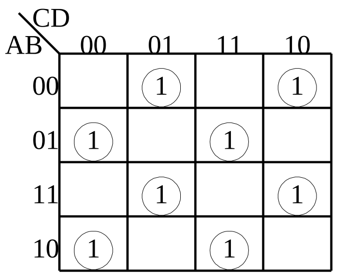
Inputs				***Outputs***		
A	***B***	***C***	***D***	***E***	***F***	***G***
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	0
0	1	1	1	0	1	1
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	0	1	0
1	0	1	1	0	1	1
1	1	0	0	0	1	0
1	1	0	1	0	1	1
1	1	1	0	0	1	1
1	1	1	1	1	0	0

Table-1: Truth Table for given function

5. Simplify output expressions: K-maps for each output are shown below.







E = ABCD
F = A'CD + A'BD + A'BC + ABC' + ACD' + AB'D
G = $A \oplus B \oplus C \oplus D$

6. Implement each expression: (not shown).

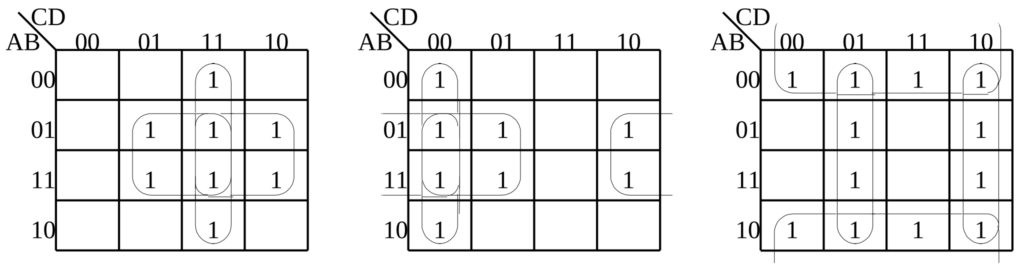
Minimization of Multiple Output Circuits

Minimization problems for circuits with single outputs are relatively straightforward, however, circuits with multiple outputs are more difficult to minimize. The difficulty in minimizing multiple output circuits results from the fact that simply minimizing the output expression for each of the outputs does not always produce the minimal complete circuit. Since product terms generated as a part of one output can be shared by another output, the use of non-minimal product terms can sometimes result in more shared gates and thus results in fewer gates for the complete circuit.

Example:
Suppose that a certain circuit has 4 inputs (***A***, ***B, C***, ***D***) and 3 outputs (***F1***, ***F2***, ***F3***) where the outputs are defined as:

$$F1 = \Sigma(3, 5, 6,7, 11, 13,14,15)$$
$$F2 = \Sigma(0, 4, 5,6, 8, 12, 13,14)$$
$$F3 = \Sigma(0, 1,2,3, 5, 6, 8,9,10,11, 13,14)$$

If these 3 outputs were minimized separately into **SOP** form, the K-maps and output expressions would be:



F1 = CD + BD + BC ***F2 = C'D' + BC' + BD'*** ***F3 = B' + C'D + CD'***

Implementing these three functions would require 11 gates (not counting inverters) as shown in figure 1.

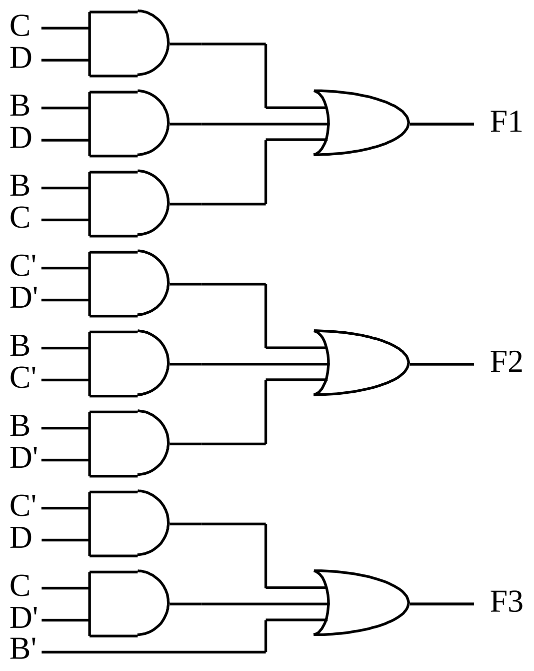
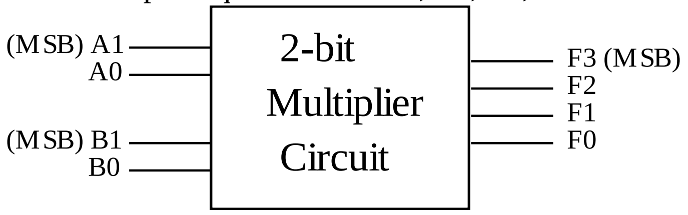


Figure-1 Logic circuit

3. Preliminary Work

Design a combinational logic circuit with multiple outputs that will serve as a 2-bit multiplier. As illustrated below, the circuit will multiply the 2-bit input (***A1*** and ***A0***) by the other 2-bit input (***B1*** and ***B0***). Since the largest value for each input is $(11)_2 = (3)_{10}$, the largest possible output is decimal $(9)_{10}$ or $(1001)_2$.
Thus the output requires 4 bits: ***F3***, ***F2***, ***F1***, and ***F0***.



```
   A1 A0
 x B1 B0
F3 F2 F1 F0
```

Determine a minimal design for the circuit described above. Implement the circuit using only ***AND***, ***OR***, ***NOT***, ***NAND***, ***NOR***, and ***XOR*** gates. Your grade will be based to some extent on the number of gates used. The **fewer** the gates, the better the grade! Display the output using four LEDs. Generate **full circuit documentation** for the circuit.

4. Experimental procedure

Construct the circuit designed in step 1 of the Preliminary Work according to the wire table generated. Note any changes. Test the circuit for all possible input switch combinations and record the results in a truth table.

Equipment List:

1) Various TTL family ICs (depends on design)

EXPERIMENT 5

DECODERS AND MULTIPLEXERS

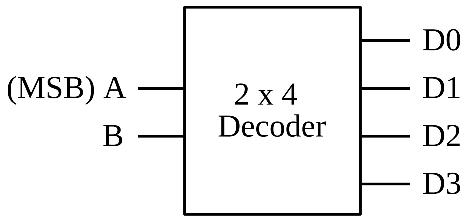
1. Objective

The objective of this laboratory is to investigate the use of decoders and multiplexers to implement combinational logic circuits.

2. Theory

2.1 Decoders

A decoder is a combinational logic circuit that activates one of several output lines based on the input code (typically binary or BCD). Shown below in Figure 1 is a block diagram and a truth table for a 2-line-to-4-line (or 2 x 4) decoder that has active-HIGH inputs and outputs.



Inputs		Outputs			
A	B	D0	D1	D2	D3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Figure 1. 2 x 4 decoder with active-HIGH inputs and outputs

Note that functionally the outputs of the decoder above correspond to minterms.
For example,
$D0 = m_0 = \overline{A} \bullet \overline{B} \bullet \overline{C} \bullet \overline{D}$. A combinational logic function that is expressed as a sum of minterms, therefore, can be implemented by summing decoder outputs.
For example, if f(A,B) = Σ(0, 2, 3) then f (A,B)= D0 + D2 + D3 so f can be implemented by the circuit shown in Figure 2.

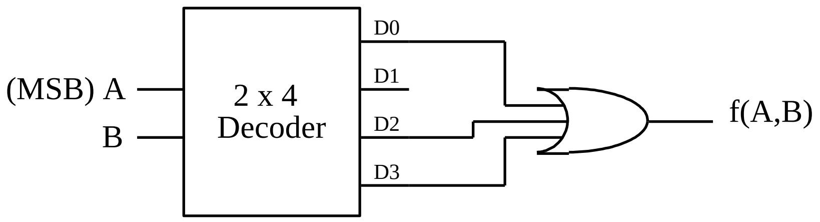
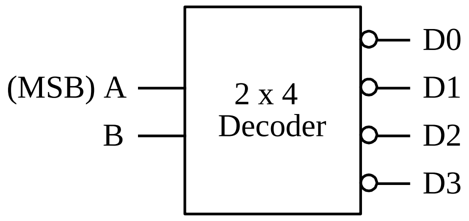


Figure 2. f(A,B) = Σ (0,2,3) implement using a 2 x 4 decoder

Some decoders, such as the 74LS155, have active-LOW outputs. Figure 3 shows a block diagram and a truth table for a 2 x 4 decoder with active-LOW outputs.



Inputs		Outputs			
A	B	D0	D1	D2	D3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Figure 3. 2 x 4 decoder with active-LOW inputs and outputs

Note that functionally the outputs of the decoder above correspond to maxterms.
For example,
$D0 = \overline{m_0} = M_0 = \overline{\overline{A} \bullet \overline{B} \bullet \overline{C} \bullet \overline{D}} = (A + B + C + D)$. A combinational logic function that is expressed as a product of maxterms, therefore, can be implemented by ANDing decoder outputs. For example, if f(A,B) = Π(0, 1, 3) then f (A,B)= D0 • D1 • D3 so f can be implemented by the circuit shown in Figure 4.

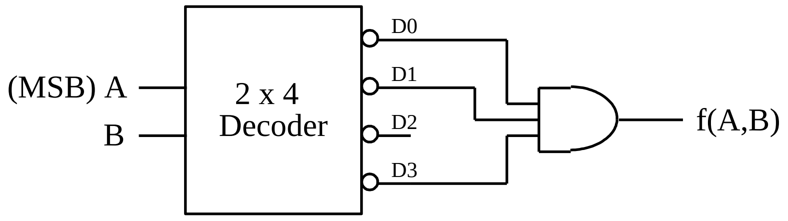


Figure 4. f(A,B) = Π(0,1,3) implement using a 2 x 4 decoder

2.2 Multiplexers

A ***multiplexer***, or ***data selector***, can be also be used to implement combinational logic circuits. A ***multiplexer implementation table*** is used to determine the input connections for the multiplexer.

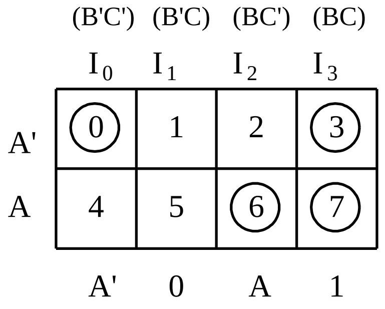
A 2 x 1 multiplexer can be used to implement a function of 2 variables, such as f(A,B)

A 4 x 1 multiplexer can be used to implement a function of 3 variables, such as f(A,B,C)

A 8 x 1 multiplexer can be used to implement a function of 4 variables, such as f(A,B,C,D)

Example: Implement the function f(A,B,C) = Σ(0, 3, 5, 7) using a 4 x 1 multiplexer.

The multiplexer implementation table is shown below in Figure 5.



	(B'C') I_0	(B'C) I_1	(BC') I_2	(BC) I_3
A'	⓪	1	2	③
A	4	5	⑥	⑦
	A'	0	A	1

Figure 5. Multiplexer implementation table for f(A,B,C) = Σ(0,3,6,7)

Note that each minterm in f(A,B,C) is circled in the table. Connections for each input are determined as follows:

If no minterms are circled in a column, a logical 0 is connected to the input (Ex: $I_1 = 0$)
If the only one minterm is circled in a column, the input is equal to the variable shown to the left (Ex: $I_0 = A'$ and $I_2 = A$)
If both minterms are circled in a column, a logical 1 is connected to the input
(Ex: $I_3 = 1$)

The circuit can be implemented as shown in Figure 6.

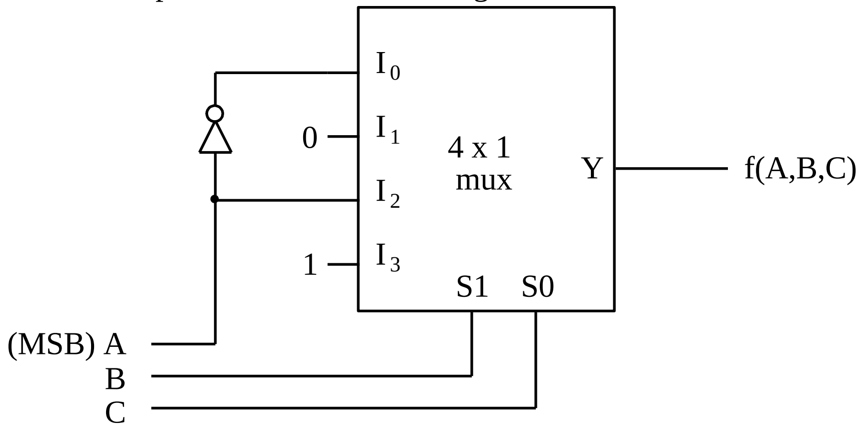


Figure 6. f(A,B,C) = Σ (0,3,6,7) implement using a 4 x 1 multiplexer

Keep in mind in the example above that bit A was the MSB. If another bit is the MSB, if the select lines are reversed, or if any bit except the MSB is connected to the inputs, then the multiplexer implementation table and the circuit will change.

For example, if the same function used above is implemented with input C connected to the inputs and inputs A and B to the select lines, then the multiplexer implementation table and the circuit will appear as shown below in Figure 7.

	(A'B') I_0	(A'B) I_1	(AB') I_2	(AB) I_3
C'	⓪	2	4	⑥
C	1	③	5	⑦
	C'	C	0	1

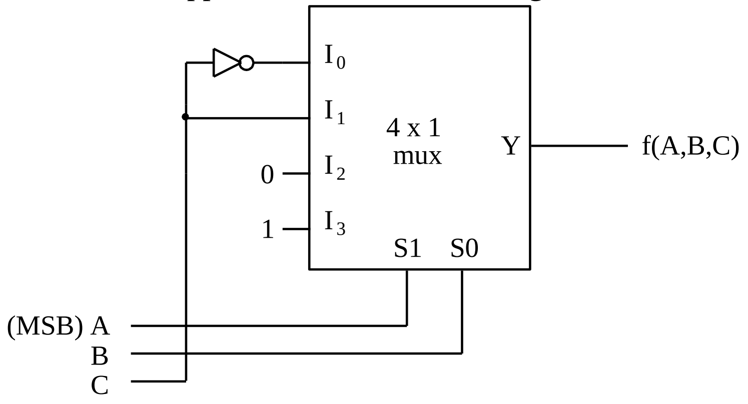
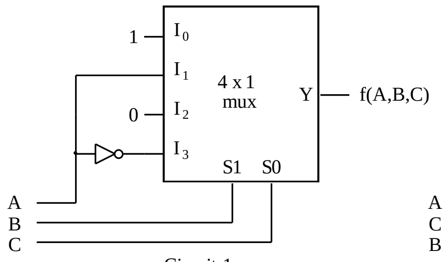


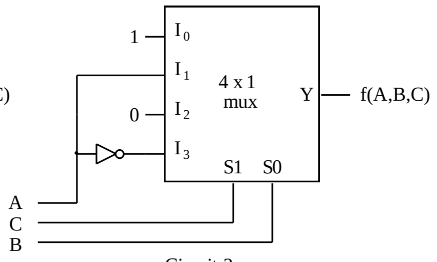
Figure 7. Alternate multiplexer implementation table and circuit

3. Preliminary Work

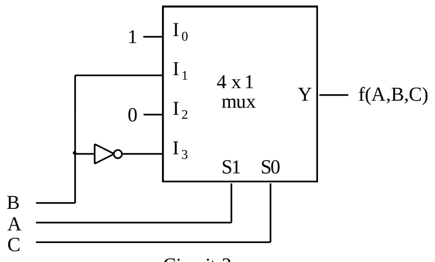
Draw the multiplexer implementation table for each circuit below and determine the output function f(A,B,C) in sum of minterms form.



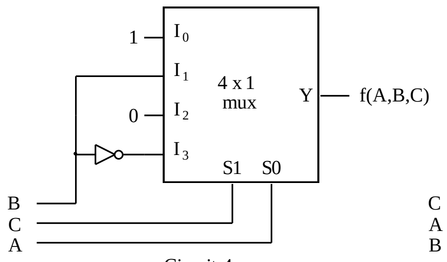
Circuit 1



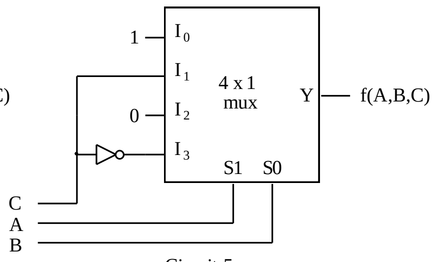
Circuit 2



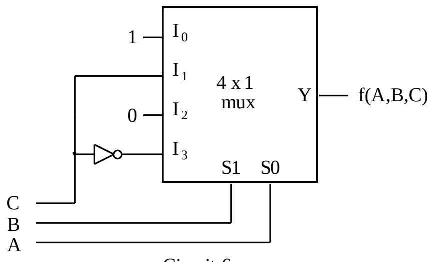
Circuit 3



Circuit 4



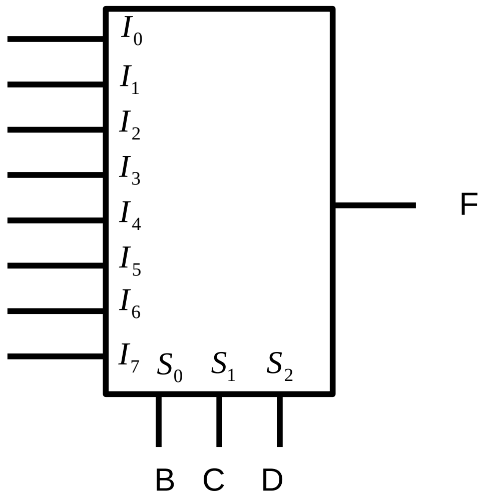
Circuit 5



Circuit 6

4. Experimental Procedure

Implement $F(A,B,C,D)=\sum(1,3,5,6,9,14,15)$ function with 8-to-1 data selector/multiplexer as shown in Figure 9. **<u>Demonstrate proper operation of the circuit to the instructor.</u>**



Equipment List

1) Breadboard
2) 5V Power Supply
3) Wire, switches, etc.
4) 74151 8 x 1 Data Selector (multiplexer)
5) 74155 Dual 2 x 4 Decoder/Demultiplexer
6) Assorted AND, OR, NAND, NOR, XOR, and INVERTER IC's

EXPERIMENT 6
FLIP FLOPS

1. Objective

The objective of this laboratory is to introduce the student to the use of bistable multivibrators (flip-flops), monostable multivibrators (one-shots), and astable multivibrators (Clock-generators). Switch debouncing is also investigated.

2. Theory

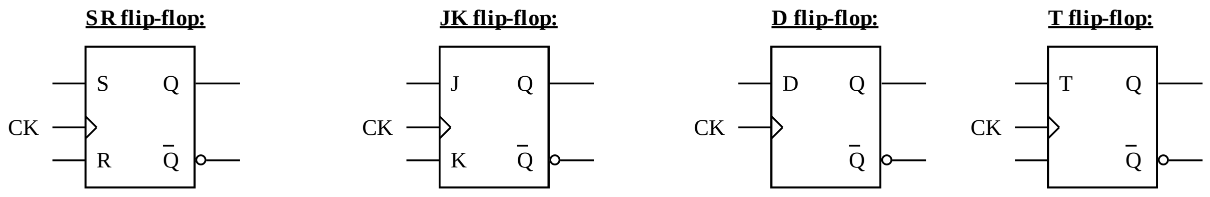
2.1 Multivibrators

A multivibrator is a circuit whose output oscillates between logic HIGH and LOW states, either automatically or due to some input. There are three types of multivibrators:

1) **Bistable multivibrators (flip-flops)** - These devices have two stable states (Q = 0 and Q = 1). They can easily be switch from one stable state to the other.
2) **Monostable multivibrators (one-shots)** - These devices have one stable state, but they may enter another unstable state for a certain period of time.
3) **Astable multivibrator (clock generator)** - These devices oscillate between two unstable states, forming a clock (square wave generator).

2.2 Flip-Flops

A flip-flop is the simplest type of memory cell. Its output, Q, does not depend solely upon its inputs, but also depends on the order in which they are applied. Thus, the flip-flop is not a combinational circuit, but is a sequential circuit. The flip-flop is the key building block of most synchronous sequential circuits. There are four common types of flip-flops. The symbol and truth table for each is shown below.



S	R	Q(t+1)	
0	0	Q(t)	(no change)
0	1	0	(reset)
1	0	1	(set)
1	1	- -	(illegal)

J	K	Q(t+1)	
0	0	Q(t)	(no change)
0	1	0	(reset)
1	0	1	(set)
1	1	$\overline{Q(t)}$	(toggle)

D	Q(t+1)
0	0
1	1

[or Q(t+1) = D]

T	Q(t+1)	
0	Q(t)	(no toggle)
1	Q(t)	(toggle)

Figure 1: Four common types of flip-flops

Flip-flops are synchronous devices meaning that the output responds to the synchronous inputs (S, R, J, K, D, or T) only on certain clock edges. There are three main types of triggering:

1) **positive-edge triggering** - the output Q can only change on the positive (rising) edge of the clock (due to the values of the synchronous inputs).

2) **negative-edge triggering** - the output Q can only change on the negative (falling) edge of the clock (due to the values of the synchronous inputs).
3) **master-slave triggering** - the synchronous inputs are “read” on the positive edge of the clock, but the output Q does not respond until the negative edge of the clock.

The type of triggering is sometimes indicated by the symbol. Shown below in Figure 2 are JK flip-flops with all three types of triggering.

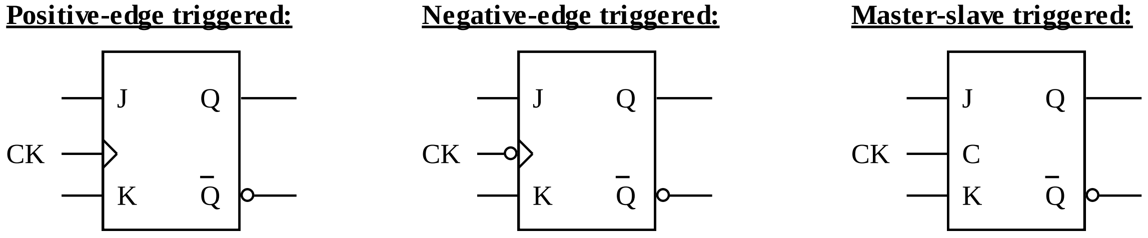


Figure 2: JK Flip-flops with different types of triggering

Flip-flops often have asynchronous inputs available also. These inputs are not synchronized with the clock, therefore, the output may respond immediately to changes in these inputs. There are two types of asynchronous inputs commonly used:
1) **PRESET** (also called **SET**) - used to preset the output Q to 1
2) **CLEAR** (also called **RESET**) - used to clear the output (set Q to 0)

Asynchronous inputs are often active-LOW. Therefore, they are typically tied HIGH for normal flip-flop operation. The PRESET or CLEAR may be momentarily set LOW to initialize the flip-flop to some desired initial value. The symbol for a flip-flop often show the asynchronous inputs as indicated below in Figure 3.

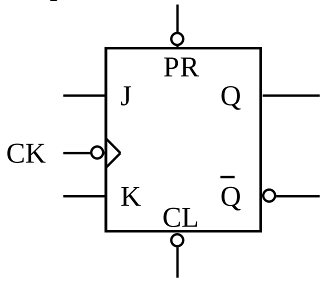


Figure 3: JK Flip-flops with asynchronous PRESET and CLEAR input

2.3 Debounced Switches

If the input to a flip-flop or sequential circuit is applied with a switch, it is important that the switch is ***debounced*** so that only a single transition occurs when the switch is thrown such as is shown in Figure 4A. The contacts in a simple switch will bounce for several milliseconds before settling down allowing several transitions to occur such as is shown in Figure 4B. Since a negative-edge triggered flip-flop reacts to each falling edge of the input clock, the input in Figure 4A would "clock" the flip-flop only once, whereas the input in Figure 4B would clock the flip-flop three times.

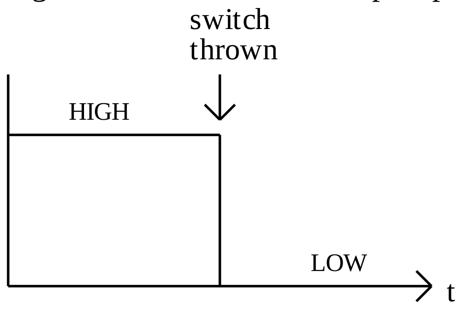


Figure 4A - Debounced switch

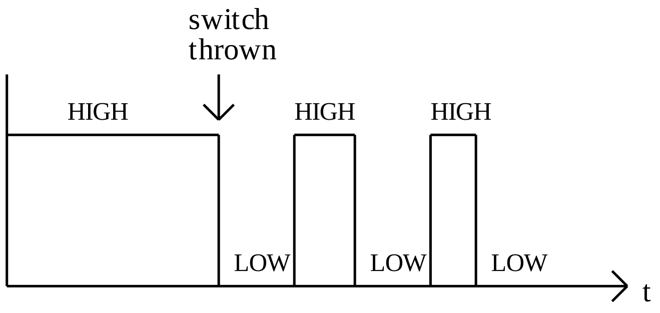


Figure 4B - Switch with contact bounce

Figure 5 shows three circuits that can be used to debounce switches.

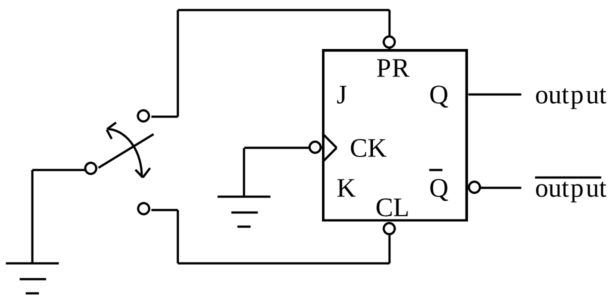


Figure 5A: Debounced switch using a JK flip-flop

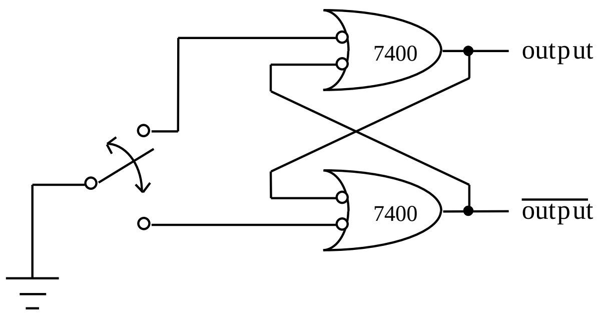


Figure 5B: Debounced switch using NAND gates

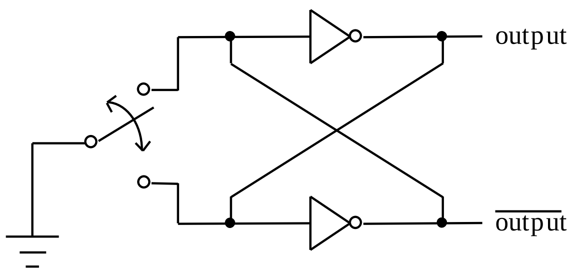


Figure 5C: Debounced switch using inverters

3. Preliminary Work

Design a sequential circuit with two JK flip-flops A and B, and one input x. When x = 0 the state of the circuit remains the same. When x = 1 the circuit goes through the state transitions from 00 to 01 to 11 to 10 back to 00 and then repeats.

4. Experimental Procedure:

Construct the circuit you found in preliminary work.

Equipment List
Breadboard
5V Power Supply
Oscilloscope
555 Timer IC
7476 Dual JK Flip-flop
Assorted AND, OR, NAND, NOR, XOR, and INVERTER IC's

EXPERIMENT 7
COUNTERS

1. OBJECTIVE

To gain experience in counters

2. THEORY

2.1 COUNTERS

Circuits for counting events are frequently used in computers and other digital systems. Since a counter circuit must remember its past states, it has to possess memory. Flip-flops are connected to make a counter. The number of flip-flops used and how they are connected determine the number of states and the sequence of the states that the counter goes through in each complete cycle.

Counters can be classified into two broad categories according to the way they are clocked:

1. **Synchronous Counters** - the same clock simultaneously triggers all memory elements.

2. **Asynchronous (Ripple) Counters** - the first flip-flop is clocked by the external clock pulse, and then each successive flip-flop is clocked by the Q or Q' output of the previous flip-flop.

There are several types of counters under those two categories such as pure binary, decade and up-down counters. In the handout, the synchronous counters will be introduced.

Synchronous Counters

1. Binary Counters

In synchronous counters, the clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel). Figure 8.1 shows a 3-bit synchronous counter using JK Flip-flop. The J and K inputs of FF0 are connected to HIGH. FF1 has its J and K inputs connected to the output of FF0, and the J and K inputs of FF2 are connected to the output of an AND gate that is fed by the outputs of FF0 and FF1.

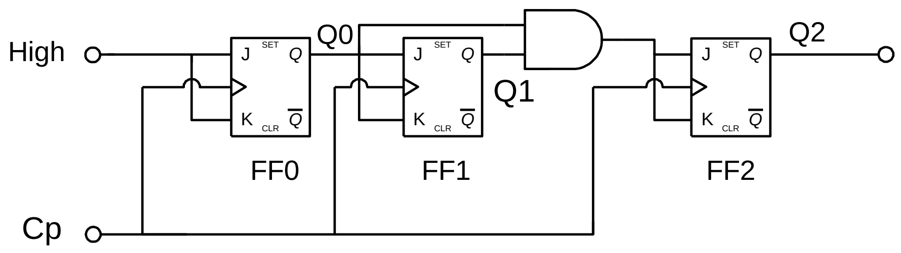
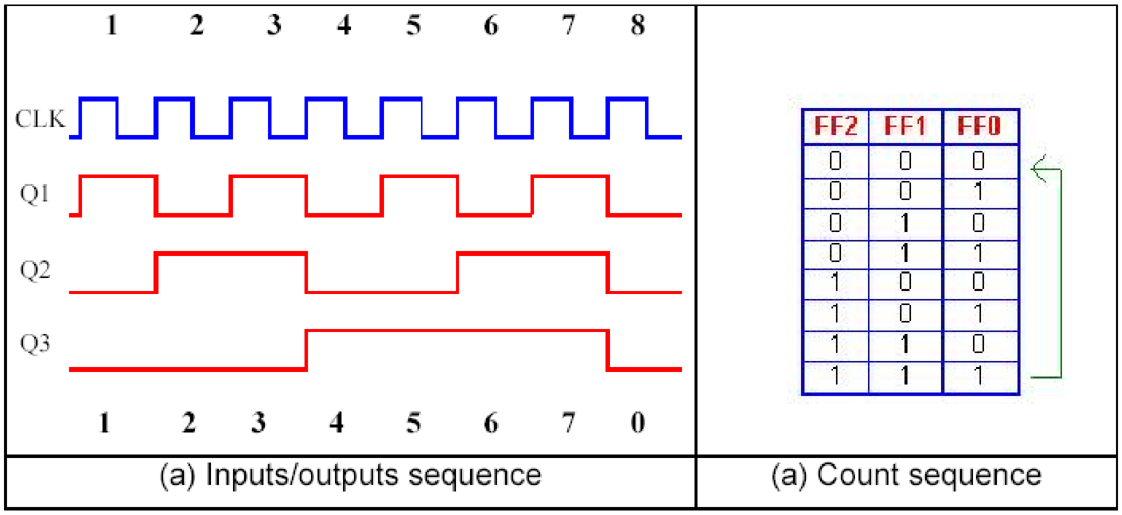


Fig 8.1a 3 bits binary counter

After the 3rd clock pulse, both outputs of FF0 and FF1 are HIGH (Fig. 8.1a). The positive edge of the 4th clock pulse will cause FF2 to change its state due to the AND gate. Figure 8.1b shows the count sequence for the 3-bit counter.



FF2	FF1	FF0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Fig 8.1b 3 bits binary counter

The most important advantage of synchronous counters is that there is no cumulative time delay because all flip-flops are triggered in parallel. Thus, the maximum operating frequency for this counter will be significantly higher than for the corresponding Asynchronous (Ripple) binary counter.

2. Decade Counters

Asynchronous decade counter counts from 0 to 9 and then recycles to 0 again. This is done by forcing the 1010 state back to the 0000 state. This so called truncated sequence can be implemented with the AND/OR logic connected as shown in Fig. 8.2.

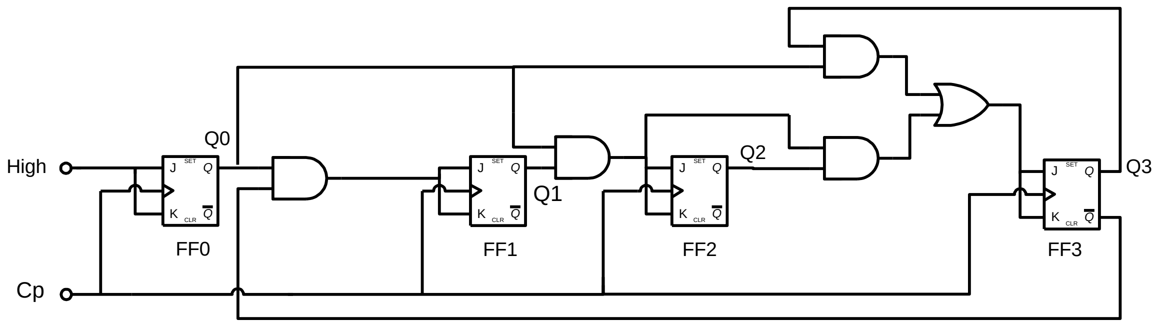
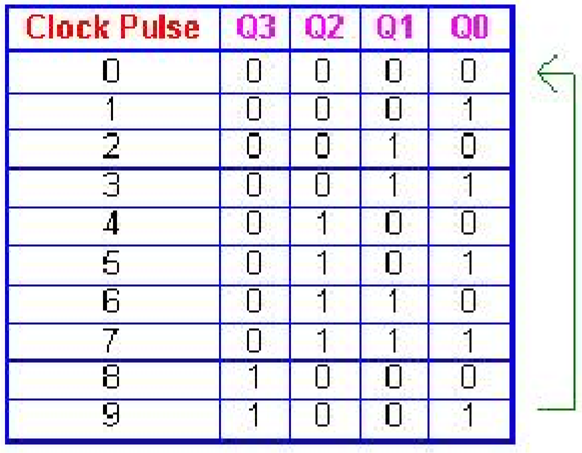


Fig 8.2. Decade Counters



Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Fig 8.3. Count sequence decade counter

From the sequence, you can notice that:

- Q0 toggles on each clock pulse.
- Q1 changes on the next clock pulse each time Q0=1 and Q3=0.
- Q2 changes on the next clock pulse each time Q0=Q1=1.
- Q3 changes on the next clock pulse each time Q0=1, Q1=1 and Q2=1 (count 7), or when Q0=1 and Q3=1 (count 9).

3. Up-Down Counters

A synchronous up-down counter also has an up-down control input. It is used to control the direction of the counter through a certain sequence. It can be implemented with the AND, OR & NOT logic connected as shown in Fig. 8.4

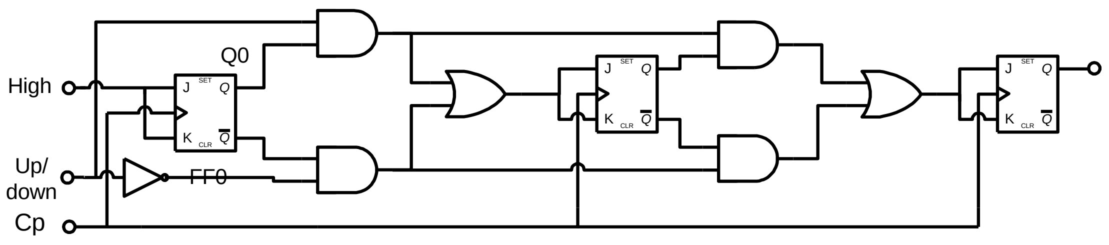
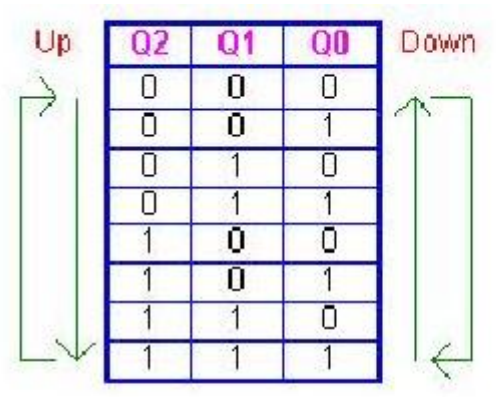


Fig 8.4 Up-down counter



Q2	Q1	Q0
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Fig. 7.14: Count sequence up-down

From the sequence, you can notice that:

- For both the UP and DOWN sequences, Q0 toggles on each clock pulse.
- For the UP sequence, Q1 changes state on the next clock pulse when Q0= 1.
- For the DOWN sequence, Q1 changes state on the next clock pulse when Q0=0.
- For the UP sequence, Q2 changes state on the next clock pulse when Q0=Q1=1.
- For the DOWN sequence, Q2 changes state on the next clock pulse when Q0=Q1=

3. Experimental Procedure:

Build on your breadboard the decade counter

Use a switch to apply the the clock input C. Connect the outputs Q and Q' to two LED's. Fill out the truth table.