MODULE-2

Chapter- Logic Circuit

Logic Circuits – Logic gates, Bistables, R-S Bistables, D-type Bistables, J-K Bistables.

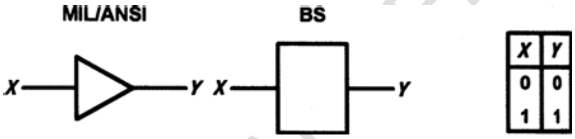
Logic Gates:

- Logic gates are circuits designed to produce the basic logic functions, AND, OR, etc.
- These circuitsare designed to be interconnected into larger, more complex, logic circuit arrangements. Since these circuits form the basic building blocks of all digital systems,
- For each gate we have included its British Standard (BS) symbol together with its American Standard (MIL/ANSI) symbol.
- We have also included the truth tables and Boolean expressions (using ‘ + ’ to denote OR, ‘·’ to denote AND, and ‘-’ to denote NOT).

Buffers (Fig. 10.9)

- Buffers do not affect the logical state of a digital signal (i.e., a logic 1 input results in a logic 1 output whereas a logic 0 input results in a logic 0 output).
- Buffers are normally used to provide extra current drive at the output but can also be used to regularize the logic levels present at an interface.
- The Boolean expression for the output, *Y*, of a buffer with an input, *X*, is:

$Y = X$



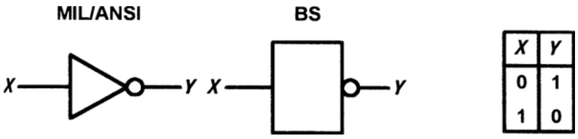
X	Y
0	0
1	1

Figure 10.9 Symbols and truth table for a buffer

Inverters (Fig. 10.10)

- Inverters are used to complement the logical state (i.e., a logic 1 input results in a logic 0 output and vice versa).
- Inverters also provide extra current drive and, like buffers, are used in interfacing applications where they provide a means of regularizing logic levels present at the input or output of a digital system.
- The Booleanexpression for the output, Y, of a buffer with an input, X, is:

Y= $\overline{X}$



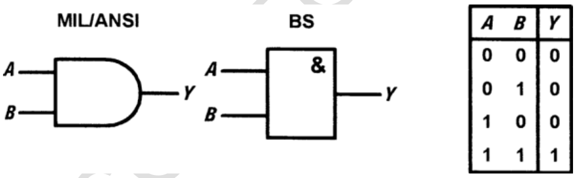
X	Y
0	1
1	0

Figure 10.10 Symbols and truth table for aninverter

AND gates (Fig. 10.11)

- AND gates will only produce a logic 1 output when all inputs are simultaneously at logic 1. Any other input combination results in a logic 0 output.
- The Boolean expression for the output, *Y*, of an AND gate with inputs, *A* and *B*, is:

$Y = A \cdot B$



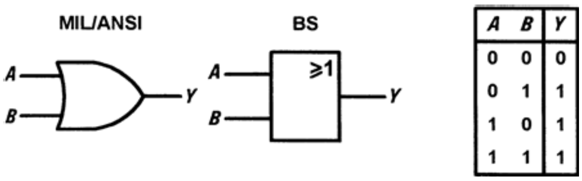
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Figure 10.11 Symbols and truth table for an AND gate

OR gates (Fig. 10.12)

- OR gates will produce a logic 1 output whenever anyone, or more, inputs are at logic 1. Putting this another way, an OR gate will only produce a logic 0 output whenever all of its inputs are simultaneously at logic 0.
- The Boolean expressionfor the output, *Y*, of an OR gate with inputs *A* and*B* is:

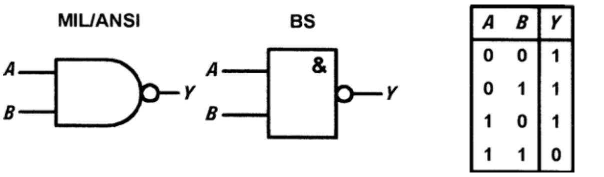
$Y = A + B$



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Figure 10.12 Symbols and truth table for an ORgate

NAND gates (Fig. 10.13)



A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

Figure 10.13 Symbols and truth table for a NAND gate

- NAND (i.e., NOT-AND) gates will only produce a logic 0 output when all inputs are simultaneously at logic 1. Any other input combination will produce a logic 1 output.
- A NAND gate, therefore,is nothing more than an AND gate with its output inverted! The circle shown at the

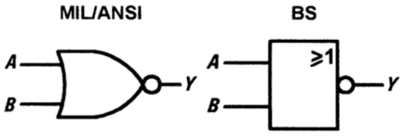
output denotes this inversion. The Boolean expression for the output, *Y*, of a NAND gate with inputs *A* and *B* is:

$$Y = \overline{A.B}$$

NOR gates (Fig. 10.14)

- NOR (i.e., NOT-OR) gates will only produce a logic 1 output when all inputs are simultaneously at logic 0. Any other input combination will produce a logic 0 output.
- A NOR gate, therefore, is simplyan OR gate with its output inverted. A circle is again used to indicate inversion. The Boolean expression for the output, *Y*, of a NOR gate with inputs, *A* and *B*, is:

$$Y = \overline{A + B}$$



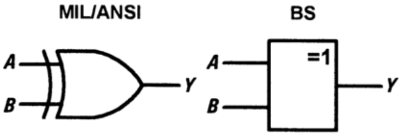
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Figure 10.14 Symbols and truth table for a NOR gate

Exclusive-OR gates (Fig. 10.15)

- Exclusive-OR gates will produce a logic 1 output whenever either one of the inputs is at logic 1 andthe other is at logic 0.
- Exclusive-OR gates producea logic 0 output whenever both inputs have the same logical state (i.e., when both are at logic 0 orboth are at logic 1).
- The Boolean expression for the output, *Y*, of an exclusive-OR gate with inputs*A* and *B* is:

$$Y = A \cdot \bar{B} + B \cdot \bar{A}$$



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 10.15 Symbols and truth table for anexclusive-OR gate

Combinational logic

By using a standard range of logic levels (i.e. voltage levels used to represent the logic 1 and logic O states) logic circuits can be combined in order to solve complex logic functions.

Example 10. 1

A logic circuit is to be constructed that will produce a logic 1 output whenever two or more of its three Inputs are at logic 1

Solution

This circuit could be more aptly referred to as a **majority vote** circuit. It s truth table is shown In Fig. 10.16. Fig. 10.17 shows the logIc circuitry required.

A	B	C	Y	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	← B·C
1	0	0	0	
1	0	1	1	← A·C
1	1	0	1	← A·B
1	1	1	1	← A·B·C

$$Y = (B \cdot C) + (A \cdot C) + (A \cdot B) + A \cdot B \cdot C$$

Figure 10.16 See Example 10.2

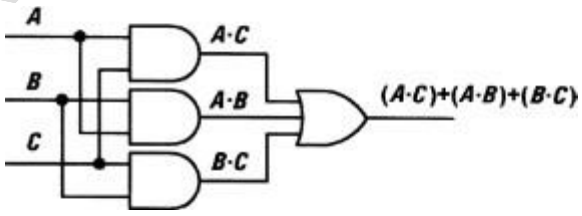


Figure 10.17 See Example 10.2

Example 10.2

Show how an arrangement of basic logic gates (AND, OR and NOT) can be used to produce the exclusive-OR function.

Solution

In order to solve this problem, consider the Boolean expression for the exclusive-OR function:

$$Y = A.\ \bar{B} + B.\ \bar{A}$$

This expression takes the form:

Y= P+Q where P= A. $\bar{B}$ and Q= B. $\bar{A}$

A. $\bar{B}$ and Q= B. $\bar{A}$ can be obtained using two two-input AND gates and the result (i.e., P and Q) can then be applied to an OR gate with two inputs.

The complete solution shown in the figure 10.18

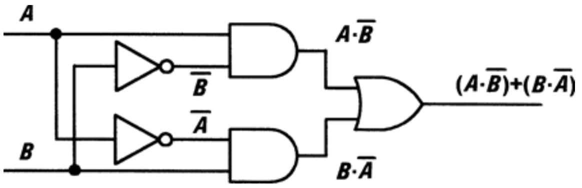


Figure 10.18 See Example 10.3

Bistables

The output of a bistable has two stable states (logic 0 or logic 1) and, once set in one or other of these states, the device will remain at a particular logic level for an indefinite period until reset. A bistable thus constitutes a simple form of 'memory cell' because it will remain in its latched state (whether **set** or **reset**) until a signal is applied to it in order to change its state (or until the supply is disconnected).

R-S bistables

- The simplest form of bistable is the R-S bistable.This device has two inputs, SET and RESET, and complementary outputs, Q and $\bar{Q}$
- A logic 1 applied to the SET input will cause the Q output to become (or remain at) logic 1 while a logic 1 applied to the RESET input will cause the Q output to become (or remain at) logic 0.
- In either case, the bistable will remain in its SET or RESET state until an input is applied in such a sense as to change the state.
- Two simple forms of R-S bistable based on cross-coupled logic gates are shown in Fig. 10.19. Fig. 10.19(a) is based on NAND gates while Fig.10.19(b) is based on NOR gates.

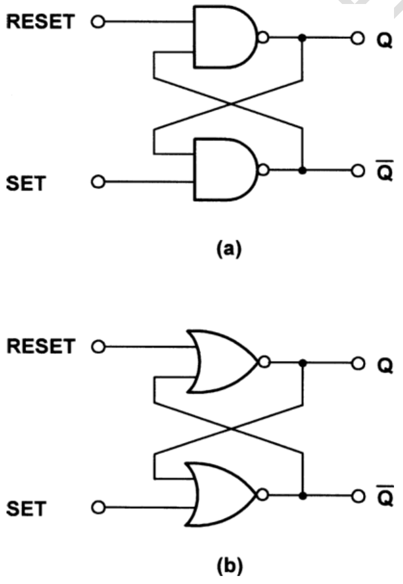


Figure 10.19 R-S bistables using cross-coupled NAND and NOR gates

- The simple cross-coupled logic gate bistable has a number of serious short comings (consider what would happen if a logic 1 was simultaneously present on both the SET and RESET inputs!) and practical forms of bistable make use of much improved purpose-designed logic circuits such as D-type and J-K bistables.

D-type bistables

- The D-type bistable has two inputs: D (standing variously for 'data' or 'delay') and CLOCK (CLK). The data input (logic 0 or logic 1) is clocked into the bistable such that the output state only changes when the clock changes state.
- Operationis thus said to be synchronous. Additional subsidiary inputs (which are invariably active low) are provided which can be used to directlyset or reset the bistable. These are usually called PRESET (PR) and CLEAR (CLR).
- D-type bistables are used both as latches (a simple form of memory) and as binary dividers. The simple circuit arrangement in Fig. 10.20 together with the timing diagram shown in Fig. 10.21 illustrate the operation of D-type bistables.

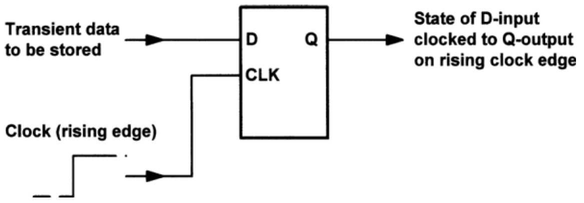

Transient data to be stored
D
Q
CLK
State of D-input clocked to Q-output on rising clock edge
Clock (rising edge)

Figure 10.20 D-type bistable operation

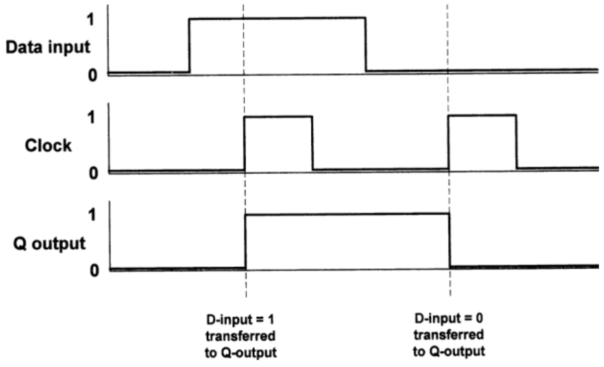


Figure 10.21 Timing diagram for the D-type bistable

J-K bistables

- J-K bistables have two clocked inputs (J and K), two direct inputs (PRESET and CLEAR), a CLOCK(CK) input, and outputs (Q and Q).
- As with R-S bistables, the two outputs are

complementary (i.e., when one is 0 the other is 1, and vice versa).

- Similarly, the PRESET and CLEAR inputs are invariably both active HIGH (i.e., a 1 on the PRESET input will set the Q output to 1 whereas a 1 on the CLEAR input will set the Q output to 0).
- Tables 10.4 and 10.5 summarize the operation of a J-K bistable, respectively, for the PRESET and CLEAR inputs and for clocked operation.

Table 10.4 **Input and output states for a J-K bistable (PRESET and CLEAR inputs)**

Inputs		Output	Comments
PRESET	CLEAR	Q_{N+1}	
0	0	?	Indeterminate
0	1	0	Q output changes to 0 (i.e., Q is reset) regardlessof the clock state
1	0	1	Q output changes to 1 (i.e., Q is set) 3 regardless of the clock state
1	1	See below	Enables clocked operation (refer to Table 10.5)

Note: The pre-set and clear inputs operate regardless of the clock

Table 10.5 Input and output states for a J-Kbistable (clocked operation)

Inputs		Output	Comments
J	K	Q_{N+1}	
0	0	Q_N	No change in state of the Q output on the nextclock transition
0	1	0	Q output changes to 0(i.e., Q is reset) on the next clock transition
1	0	1	Q output changes to 1 (i.e., Q is set) on the next clock transition
1	1	Q_N	Q output changes to the opposite state on the next clock transition

Note: Q_{N+1} means 'Q after next clock transition' while $Q_N$means 'Q in whatever state it was before'

- J-K bistables are the most sophisticated and flexible of the bistable types and they can be configured in various ways including binary dividers, shift registers and latches.
- Fig. 10.22 shows the arrangement of a four-stage binary counter based on J-K bistables. The **timing diagram** for this circuit is shown in Fig. 10.23.

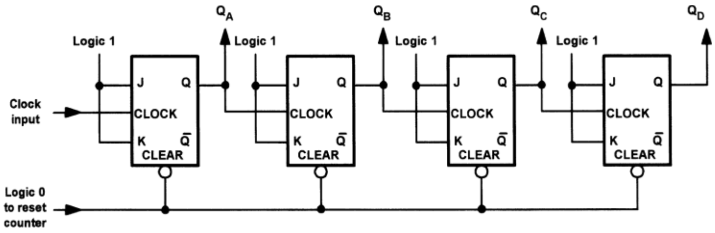


Figure 10.22 Four-stage binary counter using J-K bistables

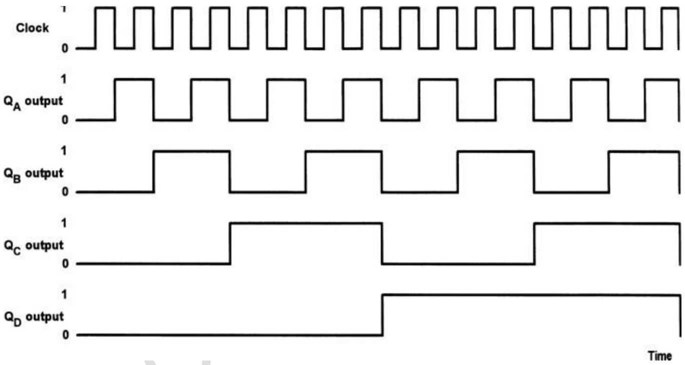


Figure 10.23 Timing diagram for the four-stage binary counter shown in Fig. 10.22

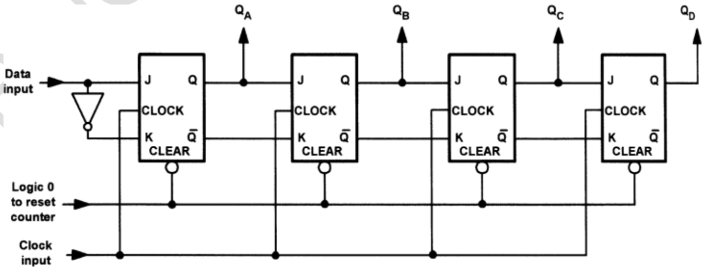


Figure 10.24 Four-stage shift register using J-K bistables

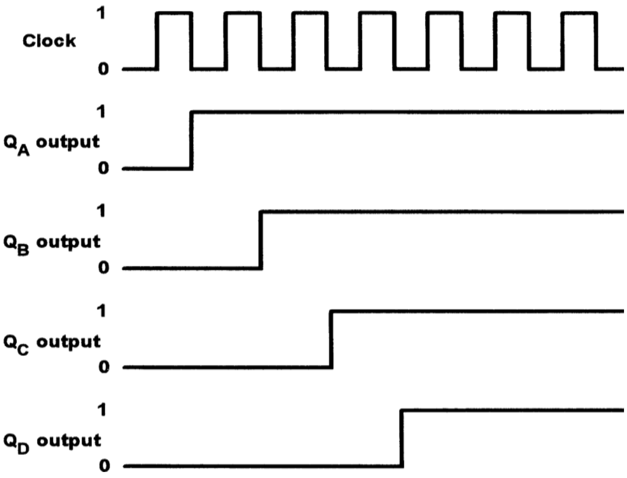


Figure 10.25 Timing diagram for the four-stage shift register shown in Fig. 10.24

- Each stage successively divides the clock input signal by a factor of two. Note that a logic 1 input is transferred to the respective Q-output on the falling edge of the clock pulse and all J and K inputs must be taken to logic 1 to enable binary counting.

- Fig. 10.24 shows the arrangement of a four- stage shift register based on J-K bistables. The timing diagram for this circuit is shown in Fig. 10.25.
- Note that each stage successively feeds data to the next stage. Note that all data transfer occurs on the falling edge of the clock pulse.

Example 10.3

A logic arrangement has to be designed so that it produces the pulse train shown in Fig. 10.27. Devise a logic circuit arrangement that will generate this pulse train from a regular square wave input.

Solution

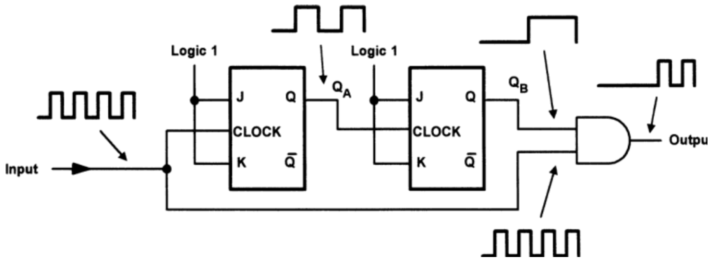


Figure 10.26 See Example 10.4

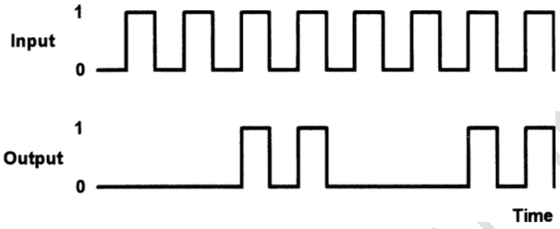


Figure 10.27 See Example 10.4

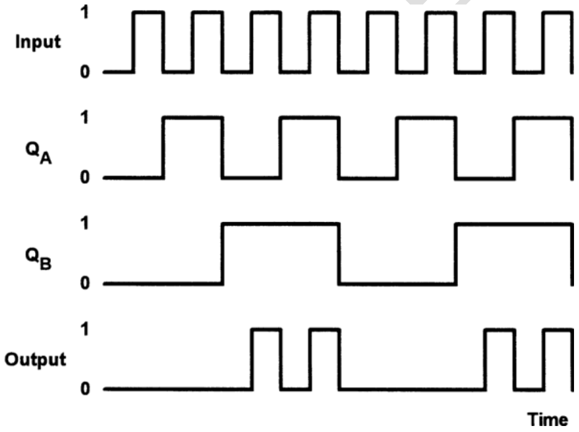


Figure 10.28 Waveforms for the logic arrangement shown in Fig. 10.26

A two-stage binary divider (based on J-K bistables)can be used together with a two-input AND gate as shown in Fig. 10.26. The waveforms for this logic arrangement are shown in Fig. 10.28.

Chapter - Microprocessor

Data representation

- Binary numbers – particularly large ones – are not very convenient. To make numbers easier to handle we often convert binary numbers to **hexadecimal (base 16)**.
- This format is easier for mere humans to comprehend and offers the advantage over denary (base 10) in that it can be converted to and from binary with ease. The first 16 numbers in binary, denary and hexadecimal are shown in **Table 11.1.**
- A single hexadecimal character (in the range zero to F) is used to represent a group of four binary digits (bits). This group of four bits (or single hex character) is sometimes called a **nibble.**
- A **byte** of data comprises a group of eight bits.Thus, a byte can be represented by just two hexadecimal (hex) characters. A group of 16 bits (a word) can be represented by four hex characters, 32 bits (a double word by eight hexcharacters, and so on).
- The value of a byte expressed in binary can be easily converted to hex by arranging the bits in groups of four and converting each nibble into hexadecimal using Table 11.1.
- Note that, to avoid confusion about whether a number is hexadecimal or decimal, we often place a **$** symbol before a hexadecimal number or add an **H** to the end of the number. For example, 64 means decimal 'sixty-four'; whereas $64 means hexadecimal 'six-four', which is equivalent to decimal 100. Similarly, 7FH means hexadecimal 'seven-F', which is equivalent to decimal 127.

Table 11.1 **Binary, denary and hexadecimal**

Binary (base 2)	Denary (base 10)	Hexadecimal (base 16)
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

Example 11.1

Convert hexadecimal A3 into binary.

Solution

From Table 11.1, A = 1010 and 3 = 0011. Thus, A3 in hexadecimal is equivalent to 10100011 inbinary.

Example 11.2

Convert binary 11101000 binary to hexadecimal.

Solution

From Table 11.1, 1110 = E and 1000 = 8. Thus 11101000 in binary is equivalent to E8 in hexadecimal.

Data types

- A byte of data can be stored at each address within the total memory space of a microprocessor system. Hence one byte can be stored at each of the 65,536 memory locations within a microprocessor system having a 16-bit address bus.
- Individual bits within a byte are numbered from 0 (least significant bit) to 7 (most significant bit). In the case of 16-bit words, the bits are numbered from 0 (least significant bit) to 15 (most significantbit).
- Negative (or signed) numbers can be representedusing **two's complement** notation where the leading (most significant) bit indicates the sign of the number (1 = negative, 0 = positive).
- For example, the signed 8-bit number 10000001 represents the denary number −1. The range of integer data values that can be represented as bytes, words and long words are shown in Table 11.2.

Table 11.2 Data types

Data type	Bits	Range of values
Unsigned byte	8	0 to 255
Signed byte	8	−128 to +127
Unsigned word	16	0 to 65,535
Signed word	16	−32,768 to + 32,767

Data storage

- The semiconductor ROM within a microprocessor system provides storage for the program code as well as any permanent data that requires storage.
- All of these data are referred to as non-volatile because they remain intact when the power supply is disconnected. The semiconductor RAM within a microprocessor system provides storage for the transient data and variables that are used by programs.
- Part of the RAM is also used by the microprocessor as a temporary store for data while carrying out its normal processing tasks.
- It is important to note that any program or datastored in RAM will be lost when the power supply is switched off or disconnected. The only exception to this is CMOS RAM, which is kept alive by means of a small battery. This **battery- backed memory** is used to retain important data,such as the time and date.
- When expressing the amount of storage provided by a memory device, we usually use Kilobytes (Kbyte). It is important to note that a Kilobyte of memory is actually 1,024 bytes (not 1,000 bytes).
- The reason for choosing the Kbyte rather than the k byte (1,000 bytes) is that 1,024 happens to be the nearest power of 2 (note that $\mathbf{2^{10}} = \mathbf{1{,}024}$).
- The capacity of a semiconductor ROM is usually specified in terms of an address range and the number of bits stored at each address. For example, 2 K × 8 bits (capacity 2 Kbytes), 4 K × 8 bits (capacity 4 Kbytes), and so on.
- Note that it is not always necessary (or desirable) for the entire memory space of a microprocessor to be populated by memory devices.

A microcontroller system

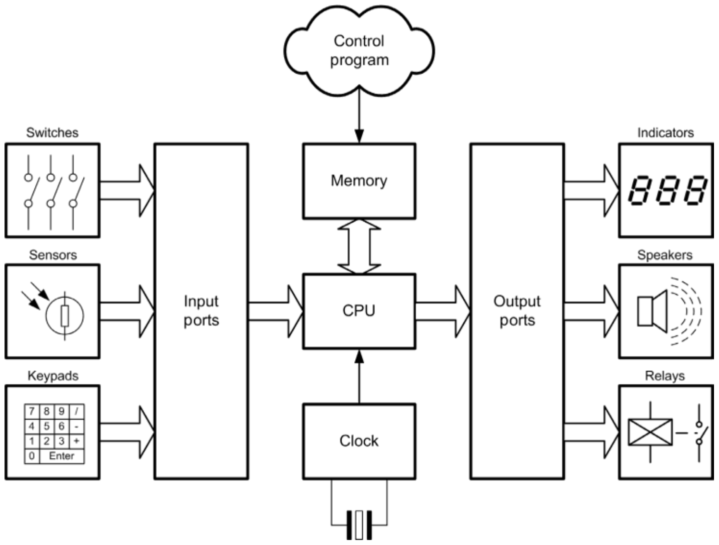


Figure 11.11 A microcontroller system with typical inputs and outputs

- Fig. 11.11 shows the arrangement of a typical microcontroller system. The sensed quantities (temperature, position, etc.) are converted to corresponding electrical signals by means of a number of sensors.
- The outputs from the sensors (in either digital or

analogue form) are passed as input signals to the microcontroller.

- The microcontroller also accepts inputs from the user. These user-set options typically include target values for variables (such as desired room temperature), limit values (such as maximum shaft speed) or time constraints (such as 'on' time and 'off' time, delay time, etc.).
- The operation of the microcontroller is controlled by a sequence of software instructions known as a control program. The control program operates continuously, examining inputs from sensors, user settings and time data before making changes to the output signals sent to one or more controlled devices.
- The controlled quantities are produced by the controlled devices in response to output signals from the microcontroller. The controlled device generally converts energy from one form into energy in another form. For example, the controlled device might be an electrical heater that converts electrical energy from the AC mainssupply into heat energy, thus producing a given temperature (the controlled quantity).
- In most real-world systems there is a requirement for the system to be automatic or self-regulating. Once set, such systems will continue to operate without continuous operator intervention. The output of a self-regulating system is fed back to its input in order to produce what is known as a closed-loop system.
- A good example of a closed-loop system is a heating control system that is designed to maintain a constant room temperature and humidity within a building regardless of changes in the outside conditions.
- In simple terms, a microcontroller must produce a specific state on each of the lines connected to its output ports in response to a particular combination of states present on each of the lines connected to its input ports (see Fig. 11.11).
- Microcontrollers must also have a central processing unit (CPU) capable of performing simple arithmetic, logical and timing operations. The input port signals can be derived from anumber of sources, including:
  - switches (including momentary action push-buttons);
  - sensors (producing logic-level compatible outputs);
  - keypads (both encoded and unencoded types). The output port signals can be connected to a number of devices, including:
  - LED indicators (both individual and multiple bar types);
  - LED seven-segment displays (via a suitable interface);
  - motors and actuators (both linear and rotary types) via a suitable buffer/driver or a dedicated interface;
  - relays (both conventional electromagnetic types and optically couple solid-state types);
  - transistor drivers and other solid-state switching devices.

Input devices

- Input devices supply information to the computer system from the outside world. In an ordinary personal computer, the most obvious input device is the keyboard.
- Other input devices available on a PC are the mouse (pointing device), scanner, and modem. Microcontrollers use much simpler input devices.
- This need be nothing more than individual switches or contacts that make and break, but many other types of devices are also used including many types of sensors that provide logic-level outputs (such as float switches, proximity detectors, light sensors, etc.).

Output devices

- Output devices are used to communicate information or actions from a computer system to the outside world. In a personal computer system, the most common output device is the flat-screen display. Other output devices include printers and modems. As with input devices, microcontroller systems often use much simpler output devices.
- These may be nothing more than LEDs, piezoelectric sounders, relays and motors. In order to be connected directly to the output port of a microcontroller, an output device must, once again, be able to accept a logic compatible signal.
- Where analogue quantities (rather than simple digital on/off operation) are required at the output a digital-to-analogue converter (DAC) willbe needed.
- All of the functions associated with a DAC can be provided by a single integrated circuit. As with an ADC, the output resolution of a DAC depends on the number of bits and 8, 10 and12 bits are common in control applications.

Interface circuits

- Finally, where input and output signals are not logic compatible (i.e., when they are outside the range of signals that can be connected directly to the microcontroller) some additional interface circuitry may be required in order to shift the voltage levels or to provide additional current drive.
- Additional circuitry may also be required when a load (such as a relay or motor) requires more current than is available from a standard logic device or output port. For example, a common range of interface circuits

(solid-state relays) is available that will allow a microcontroller to be easily interfaced to an a.c. mains-connectedload.

- It then becomes possible for a small microcontroller (operating from only a 5 V d.c. supply) to control a central heating system operating from 240 V a.c. mains.

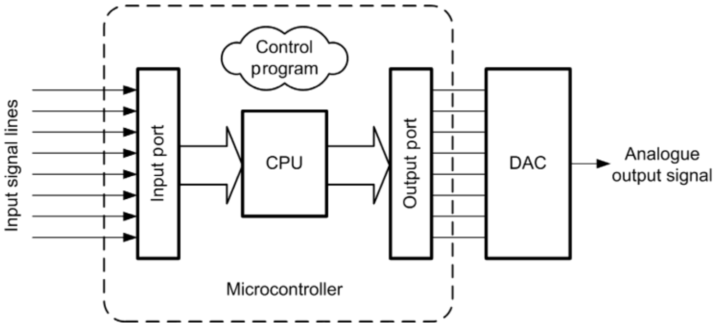
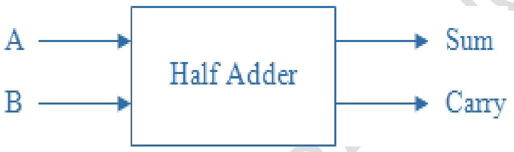


Figure 11.13 An analogue output signal can be produced by connecting a digital-to-analogue converter (DAC) to a microcontroller output port

Chapter - Boolean Algebra and Combinational Circuits

Half Adder

- A half adder is a combinational circuit which adds two one-bit binary numbers and produces two binary outputs.



- The logic circuit, truth table and logic symbol for a half adder is shown below.

Truth table

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

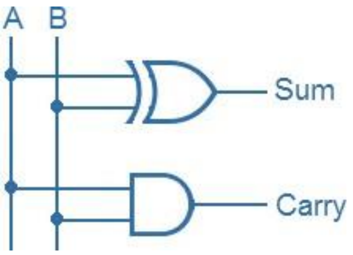
From the truth table

Sum = $A \oplus B$

Sum = $A\bar{B} + \bar{A}B$

Carry = AB

- Realization of half adder using logic gates

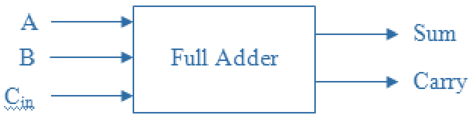


Limitations (disadvantages) of half-adder

- In multi-digit addition we have to add two bits along with the carry of previous digit addition. Such addition requires addition of 3 bits. This is not possible in half-adders.

Full Adder

- A full adder is a combinational circuit which adds three one-bit binary numbers and produces two binary outputs.



- The logic circuit, truth table and logic symbol for a full adder is shown below.

Truth table

Inputs			Outputs	
A	B	C_{in}	Sum	Carry
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

From the truth table

Sum = $\bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in}$

Sum = $\bar{A}\bar{B}C_{in} + ABC_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in}$

Sum = $C_{in}(\bar{A}\bar{B} + AB) + \bar{C}_{in}(\bar{A}B + A\bar{B})$

Sum = $C_{in}\overline{(A \oplus B)} + \bar{C}_{in}(A \oplus B)$

Sum = $A \oplus B \oplus C_{in}$

Carry = $\bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in}$

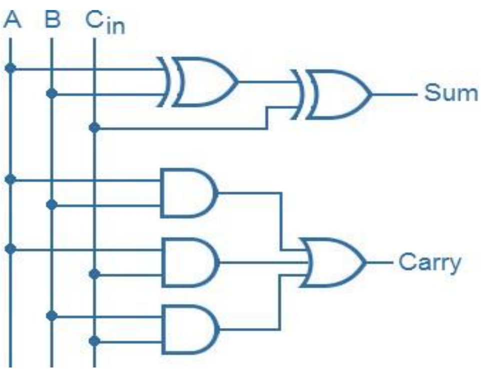
Carry = $\bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in} + ABC_{in} + ABC_{in}$

Carry = $\bar{A}BC_{in} + ABC_{in} + A\bar{B}C_{in} + ABC_{in} + AB\bar{C}_{in} + ABC_{in}$

Carry = $BC_{in}(\bar{A} + A) + AC_{in}(\bar{B} + B) + AB(\bar{C}_{in} + C_{in})$

Carry = $AB + BC_{in} + AC_{in}$

Realization of full adder using logic gates



Multiplexer

- The objective of a multiplexer is to select one signal from a group of 2^n inputs, to be an output on a single output line.
- For example, an 8-to-1 multiplexer(mux) is diagramed as a block box as shown in Fig.10.12. Lines D_0…D_7 are the data input lines and F is the output line.
- Lines A, B, and C are called the selected lines. They are interpreted as a three-bit binary number, which is used to choose one of the D lines to be output on the line F.

Implementation

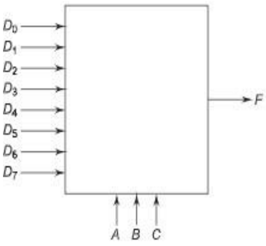


Fig. 10.12 8-to-1 multiplexer

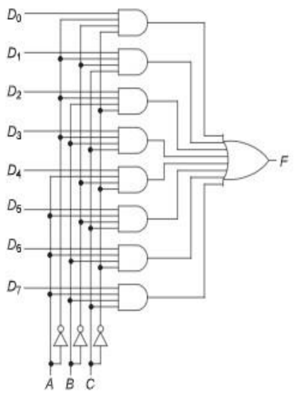


Fig. 10.13 Implementation of an 8-to-1 multiplexer

- Design of an 8-to-1 multiplexer using traditional minimization techniques would require minimizing a Boolean function 11 variables.
- Instead, a mux can be designed with a regular pattern of AND OR gates, as shown in Fig. 10.13
- Any literal A=1, $\bar{A}$=0 Suppose we want to output Y=D_5, then the three inputs to the corresponding AND gate should be 10s. Then

  The corresponding select inputs are

  5 → 101 or $A\bar{B}C$

  And the output is $A\bar{B}C$ → D_5

  Similarly, to output, D_6

  6 → 110 or $AB\bar{C}$

  Output is $AB\bar{C}$ → D_6

  The output Y will be sum (+) of all eight (2^3) outputs.

Application

Choose one of several registers to be used as ALU (Athematic Logic Unit) input. 2^n to -1 multiplexer can be used to implement an arbitrary Boolean function of n variables, by associating each input line of the multiplexer with arow of the truth table for the function.

Decoders

- The objective of the decoder is to decode an n-bit binary number, producing a signal on one of 2^n output lines. Figure 10.14 shown an 3-to-8 decoder

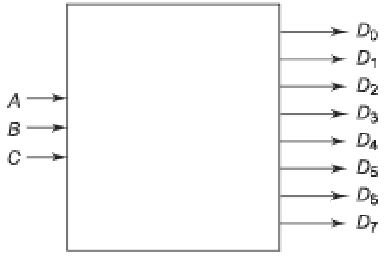


Fig. 10.14 3-to-8 decoder

Implementation

- A decoder is often implemented with an additional input called an "enable" line. When the line is enabled. The circuit is the decoder. When it is disabled, all the outputs are 0. The same circuit can be used as a de-multiplexer, which directs a single data input line to one of 2^n output lines, depending on the values of n select lines.
- Figure 10.15 shows the 3-to-8 decoder implementation.

  To produce 1 at D_5

  5 → 101

  If input is A =1, B=0, C=1 or $A\bar{B}C$, the output is 1 and D_5. It can be checked that all other outputs are 0.

To produce 1 at D_5 we input 3 → 011 or $\bar{A}BC$

Reader: Find the input to produce 1 at D_7

Application: Decode memory address for reads and writes to random access memory

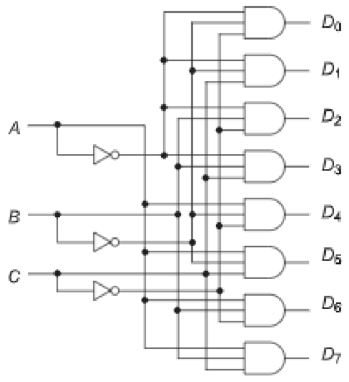


Fig. 10.15 3-to-8 decoder implementation

Chapter Boolean Algebra and Combinational Circuits

Shift Register

- Data are entered and stored in a shift register. It can then be taken out when needed.
- A 4-bit shift register is shown in Figure13.1. It comprises 4 JK flip-flops in which data are entered.
- FF3 J, K terminals are connected through an inverter so that it acts as D flip-flop through which the data are entered;
- the flip flops are cleared by placing 0 at CLR. During normal operation, the CLR is held at 1. Also, Pr is held at 0 so that these do not interfere with normal operation of the registers

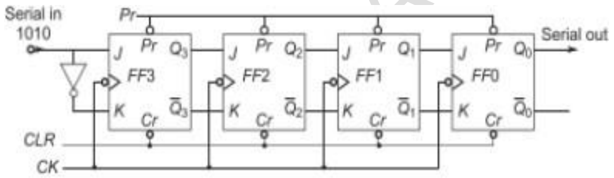
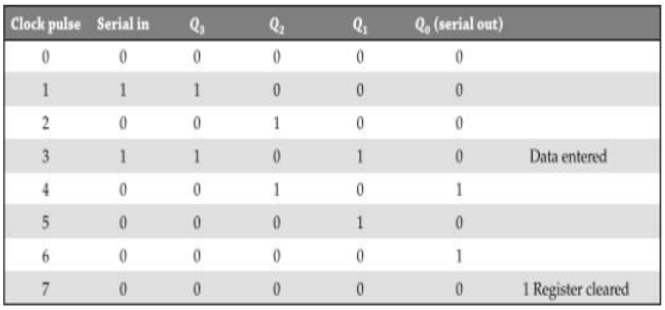


Fig. 13.1 4-bit shift register

Operation

- All the flip flops are triggered by common clock pulses (CK), data (1010) is now fed in Least Significant Digit (LSD) FIRST and Most significant Digit (MSD) last as shown in Table 13.1

Table 13.1: operation of shift register



Clock pulse	Serial in	Q_3	Q_2	Q_1	Q_0 (serial out)	
0	0	0	0	0	0	
1	1	1	0	0	0	
2	0	0	1	0	0	
3	1	1	0	1	0	Data entered
4	0	0	1	0	1	
5	0	0	0	1	0	
6	0	0	0	0	1	
7	0	0	0	0	0	1 Register cleared

Register Type

- **Serial In Serial Out (SISO)**
- **Serial In Parallel Out (SIPO)**
- **Parallel In Serial Out (PIPO)**
- **Parallel In Parallel out (PIPO)**

- A 4-bit shift register which can operate as all-fuse type is shown in Fig 13.2. The SISO operation is same as illustrated in Fig. 13.1. All we need to explain is the parallel operation.

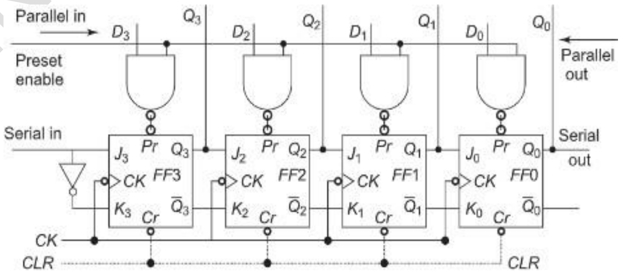


Fig. 13.2 4-bit shift register

❖ **Parallel Out**

Connections are directly taken from Q_3 to Q_0 output of flip-flops

❖ **Parallel Out**

After clearing the register, 'parallel enable' input is made. This result in NAND gates to act as inverters.

Any D_i input causes

D_i =0, P_r =1 and Qi = 0

D_i =1, P_r =0 and Qi = 1

Data is thus entered in parallel (Simultaneously) at Q_I s. It can then be read at Q_3 – Q_0 or it can be serially output by applying four pulses.

❖ **Shifting of data to left**

In the registers of the figs 13.1, and 13.2, the data is shifted from left by one bit at clock pulse by feeding output of FF0 to input of FF1, and FF1 to FF2, FF2 to FF3. On the occurrence of a pulse, the data word will shift one bit to left; This logic is shown in Fig. 13.3

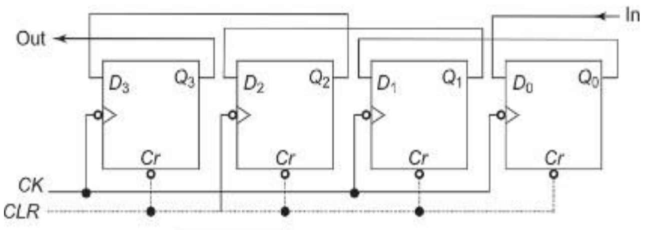


Fig. 13.3 Register shift left

- **Bidirectional Register**
  shown in Fig. 13.4. in which one AND gets 1 input, while the other AND gets 0, and the connection in one is Q, while the other will be $\overline{Q}$

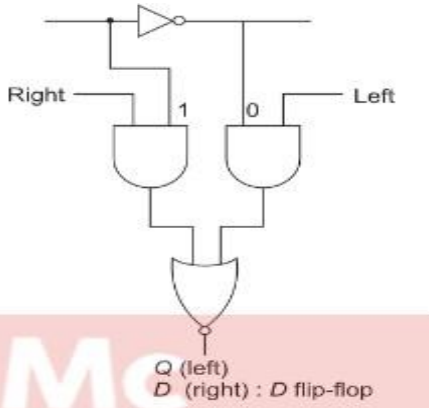


Fig. 13.4 Bidirectional register

By combining Fig 13.1 and 13.3 through AND-OR logic, we can build a bidirectional shift register. This logic is

Counters

Counter circuits count the applied clock pulses. These are usually designed using flip-flops.

- Based on the way clock is applied for their functioning counters are classified as **Synchronous and Asynchronous counters**. In **synchronous counter all the flip-flops are working in sync** with clock pulse as well as each other. Here clock pulse is applied to every flip flop
- In asynchronous counter clock pulse is applied only to the initial flip flop whose value would be considered as LSB. Instead of the clock pulse, the output of first flip-flop acts as a clock pulse to the next flip flop, whose output is used as a clock to the next in line flip-flop and so on.
- Asynchronous counter is a ripple counter as the clock pulse ripples through the circuit. An n-MOD ripple counter contains n number of flip-flops and the circuit can count up to 2^n values before it resets itself to the initial value

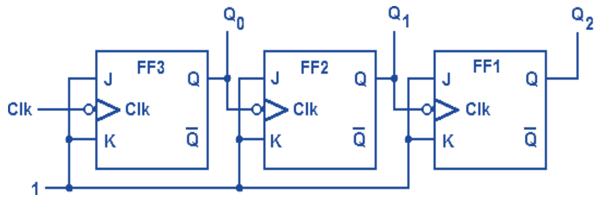


Fig.5.11: 3-bit binary ripple counter

Clock pulse	Q_2	Q_1	Q_0
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1
9	0	0	0

- The **binary ripple counter** is as shown in the fig.5.11 uses three **JK flip flops** FF3, FF2 and FF1.
- JK inputs of flip flops are tied to logic 1. The bubble at the clock input indicates a negative triggered clock pulse.
- From the figure, it can be observed that the output Q_0 of the first flip flop is applied as a clock pulse to the second flip flop and output Q_1 of the second flip flop is applied as a clock pulse to the third flip flop.
- Here the output Q_0 is the LSB and the output Q_2 is the MSB bit.

The 3-bit ripple counter can count up to $2^3 = 8$ values.i.e., 000, 001, 010, 011, 100, 101, 110, 111.

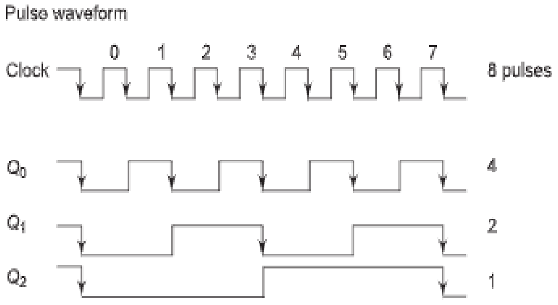


Fig. 13.7 Pulse waveform

- **Down Counter:** $\bar{Q}$s are connected to the next flip-flop. As $\bar{Q}$ toggles from 1 to 0, the corresponding Q toggles 1 to 0, which is down count.
- **Up Counter:** CLK is connected to Q for up-down and $\bar{Q}$ through AND-OR logic.