# SENSORS AND ACTUATORS

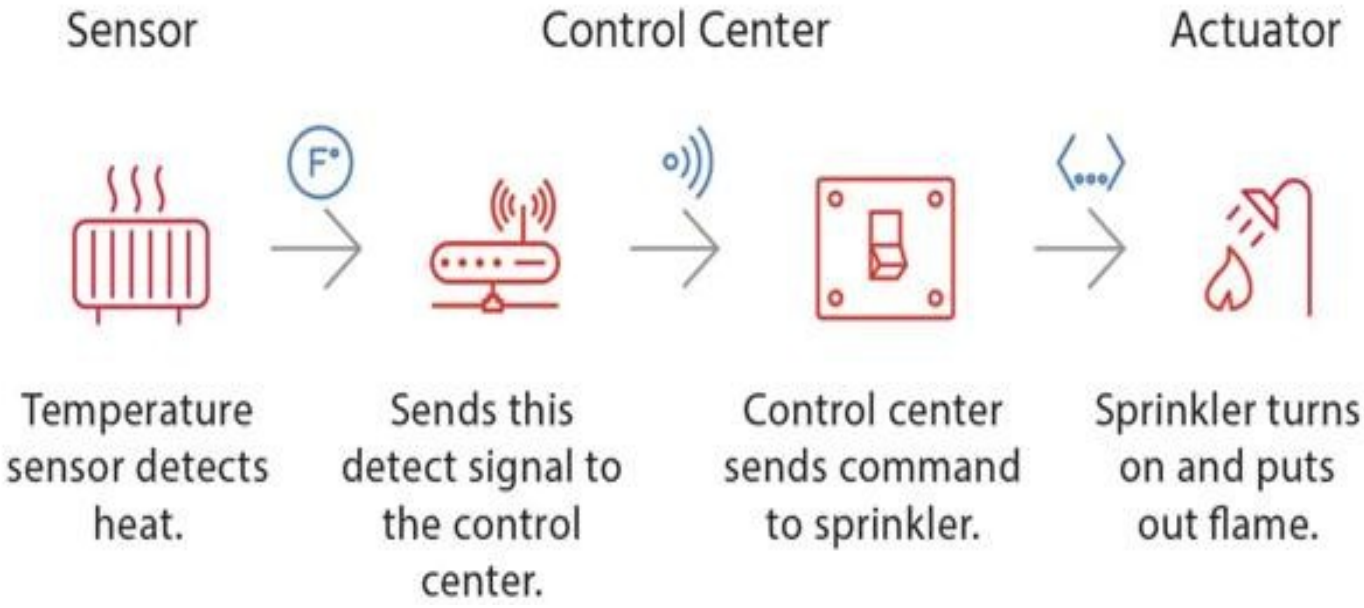
» An embedded system is in constant interaction with the Real world, and the controlling/ monitoring functions, executed by the embedded system is achieved in accordance with the changes happening to the Real world. The changes in system environment or variables are detected by the *sensors* connected to the input port of the embedded system.

» A ***sensor*** is a transducer device that converts energy from one form to another, for any measurement or control purpose.

- Sensor which counts steps for pedometer functionality is an **Accelerometer sensor**.
- Sensor used in smart watch devices to measure the high intensity is an **Ambient Light Sensor (ALS).**

»» ***Actuator*** is a form of transducer device (mechanical or electrical) which converts signals to corresponding physical action (motion). Actuator acts as an output device.

- Smart watches use Ambient Light Sensor to detect the surrounding light intensity and uses an electrical/ electronic actuator circuit to adjust the screen brightness.



» If the embedded system is designed for monitoring purpose only, then there is no need for including an actuator in the system.

» For example, take the case of an ECG machine. It is designed to monitor the heart beat status of a patient and it cannot impose a control over the patient's heart beat and its order. The sensors used here are the different electrode sets connected to the body of the patient. The variations are captured and presented to the user (may be a doctor) through a visual display or some printed chart.

| **Sensors** | **Actuators** |
|---|---|
| Sensor is an input device | Actuator is an output device |
| Convert a physical parameter to electrical output | Convert an electrical signal to a physical output |
| A device that detects events or changes in the environment and send the information to another electronic device | A component of a machine that is responsible for moving and controlling mechanisms |
| Sensor help to monitor the changes in the environment | Actuator helps to control the environment or physical changes |

» ***The I/O Subsystem:***

» The *I/O subsystem* of the embedded system facilitates the interaction of the embedded system with the external world through the sensors and actuators connected to the input and output ports respectively of the embedded system.

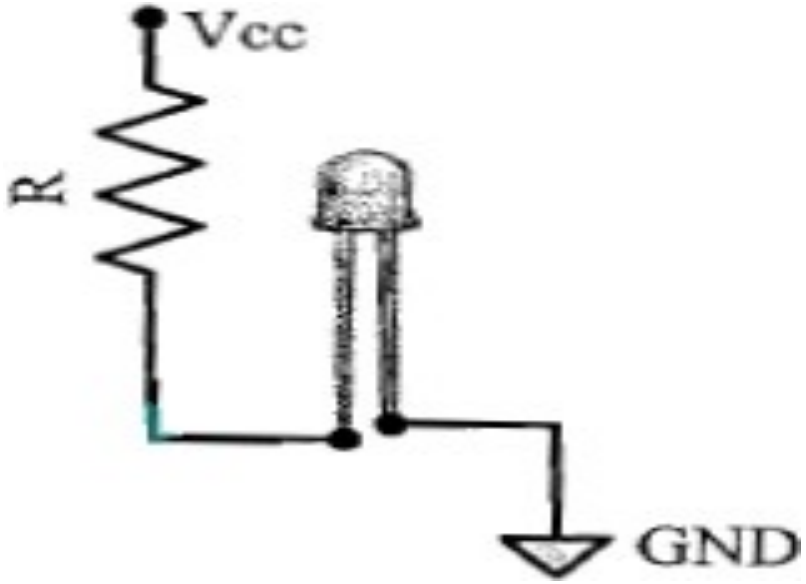
» ***Light Emitting Diode (LED):*** LED is an important output device for visual indication in any embedded system. LED can be used as an indicator for the status of various signals or situations.

» Typical examples are indicating the presence of power conditions like 'Device ON', 'Battery Low' or 'Charging of Battery' for a battery operated handheld embedded devices.

» Light Emitting Diode is a p-n junction diode

- it contains an anode and a cathode.
- Anode should be connected to +ve terminal of the supply voltage
- Cathode to the –ve terminal of supply voltage.
- A resister is used in series between the power supply and the LED to limit the current through the LED.

»



LEDs can be interfaced to the port pin of a processor/ controller in two ways.

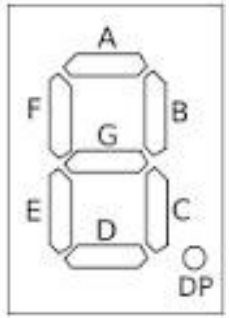
1. Anode is directly connected to the port pin and the port pin drives the LED.
   - LED turns on if Port pin is at logic High (Logic '1').
2. Cathode of the LED is connected to the port pin and the anode to the supply voltage.
   - LED is turned on when the port pin is at logic Low (Logic '0').

» ***7-Segment LED Display:*** The 7-segment LED display is an output device used for displaying alpha-numeric characters.

» It contains 8 light-emitting diode (LED) segments arranged in a special form.

» Out of the 8 LED segments, 7 are used for displaying alpha-numeric characters and 1 is used for representing ‘decimal point'.

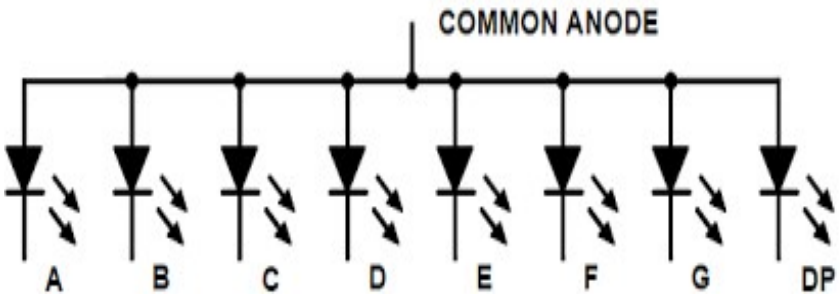
» The LED segments are named *A to G* and the 'decimal point LED segment is named as DP.

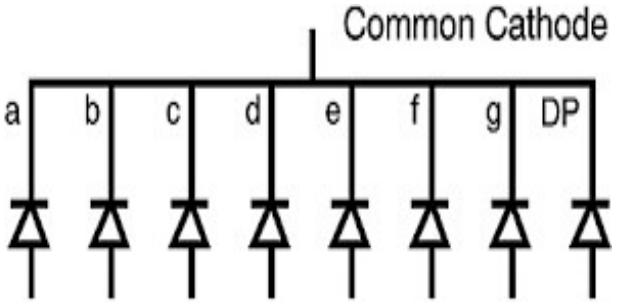


» The 7-segment LED displays are available in two different configurations:

» *common anode configuration*: the anodes of the 8 segments are connected commonly

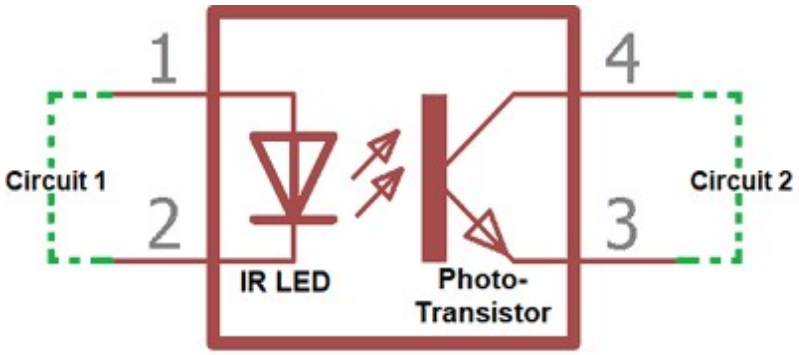
» *common cathode configuration*: the 8 LED segments share a common cathode line.

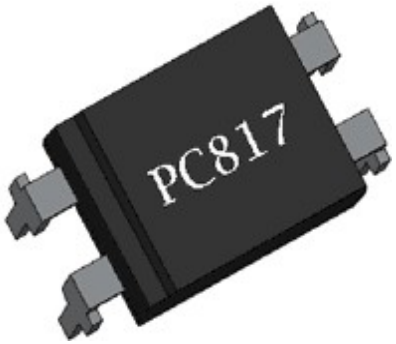


» ***Optocoupler:*** *Optocoupler* is a solid state device to isolate two parts of a circuit.

Optocoupler combines an LED and a photo-transistor in a single housing (package).

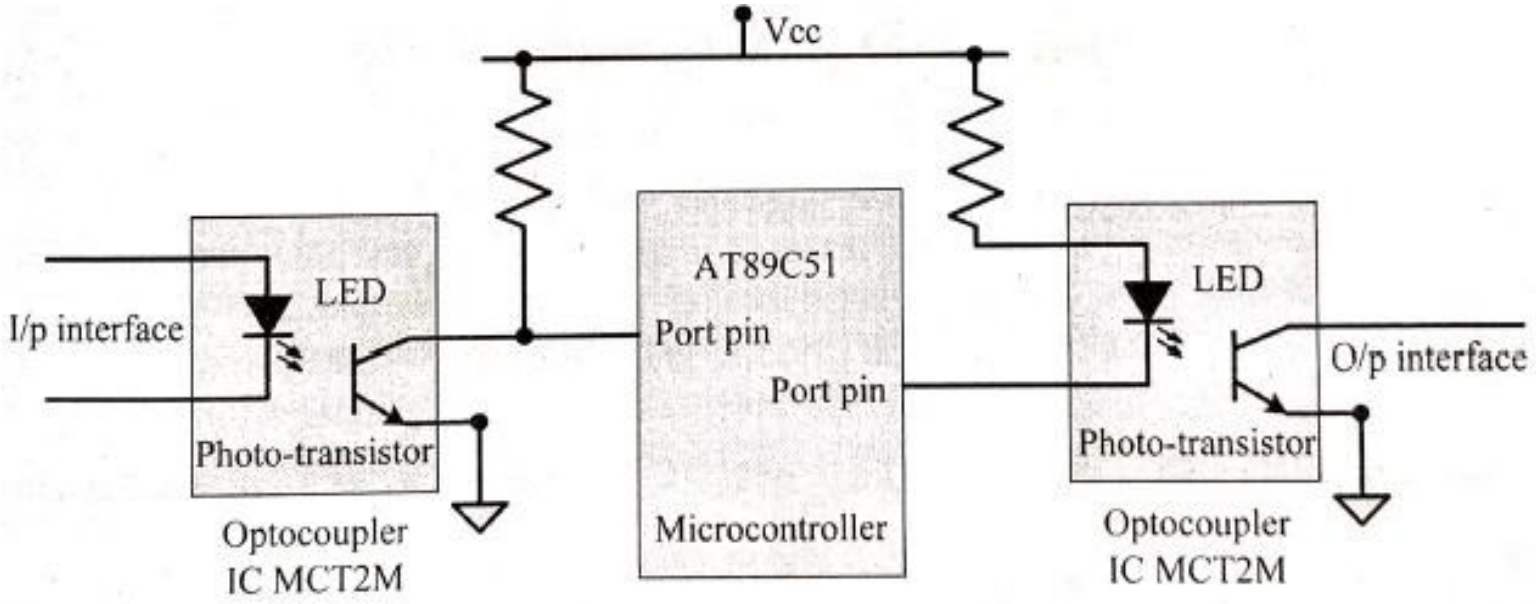



» In electronic circuits, an optocoupler is used for –

1. suppressing interference in data communication,
2. circuit isolation,
3. high voltage separation,
4. simultaneous separation and signal intensification, etc.
5. Optocouplers can be used in either input circuits or in output circuits.

» The following Figure illustrates the usage of optocoupler in input circuit and output circuit of an embedded system with a microcontroller as the system core.




» Optocoupler is available as ICs from different semiconductor manufacturers. The MCT2M IC from Fairchild semiconductor is an example for optocoupler IC.

» ***Stepper Motor:*** A *stepper motor* is an electro-mechanical devic which generates discrete displacement (motion) in response, to de electrical signals.

» A stepper motor produces discrete rotation in response to the DC voltage applied to it.

» Stepper motors are widely used in –

1. Industrial embedded applications,
2. consumer electronic products and robotics control systems.
3. The paper feed mechanism of a printer/ fax makes use of stepper motors for its functioning.

» Based on the coil winding arrangements, a two-phase stepper motor is classified into two. They are: Unipolar and Bipolar

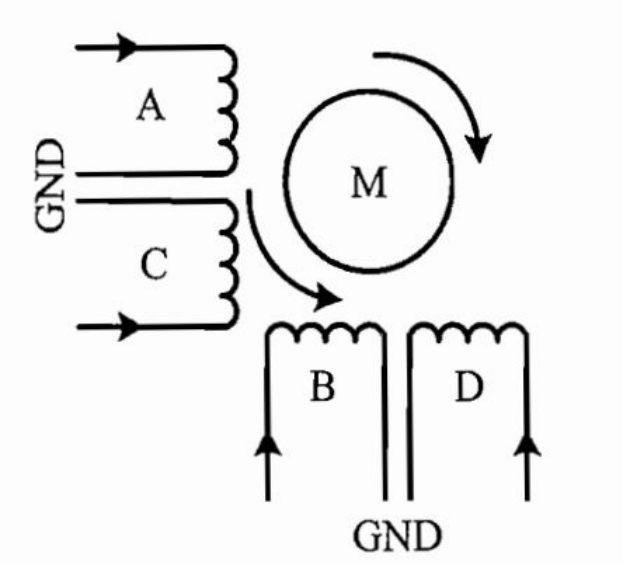
**Unipolar:** A unipolar stepper motor contains two windings per phase.

The direction of rotation (clockwise or anticlockwise) of a stepper motor is controlled by changing the direction of current flow.

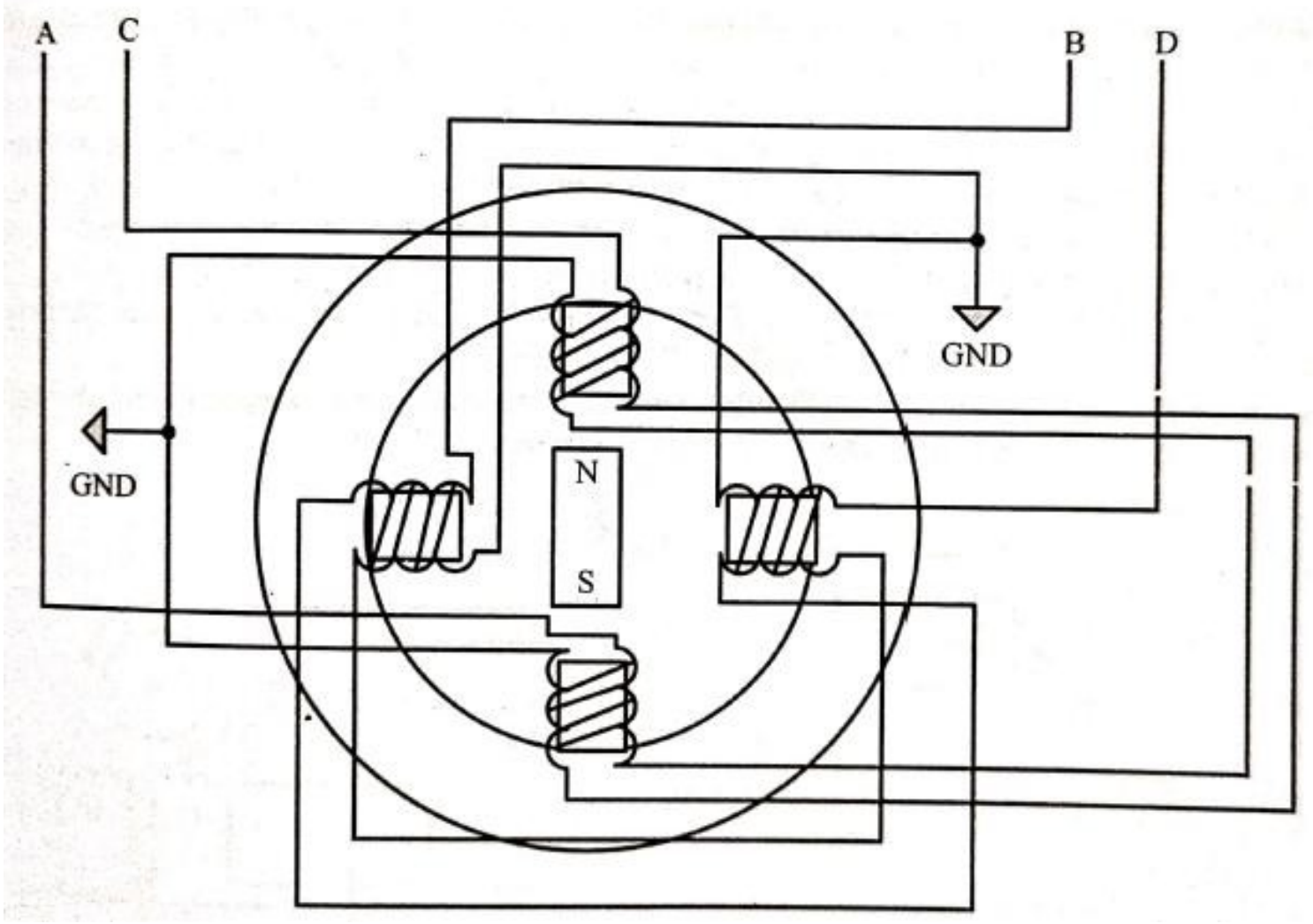
The coils are represented as A, B, C and D.

Coils A and C carry current in opposite directions for phase 1

Similarly, B and D carry current in opposite directions for phase 2.



» The following Figure shows the stator winding details of 2 phase unipolar Stepper motor:



**Bipolar:** A bipolar stepper motor contains single winding per phase.

For reversing the motor rotation the current flow through the windings reversed dynamically It requires complex circuitry for current flow reversal.

» The ***stepping of stepper motor*** can be implemented in different ways by changing the sequence of activation of the stator windings.

The different stepping modes supported by stepper motor are explained below:

» *Full Step* *Wave Step* *Half Step*

| Step | Full Step | | | |
|---|---|---|---|---|
| | Coil A | Coil B | Coil C | Coil D |
| 1 | H | H | L | L |
| 2 | L | H | H | L |
| 3 | L | L | H | H |
| 4 | H | L | L | H |

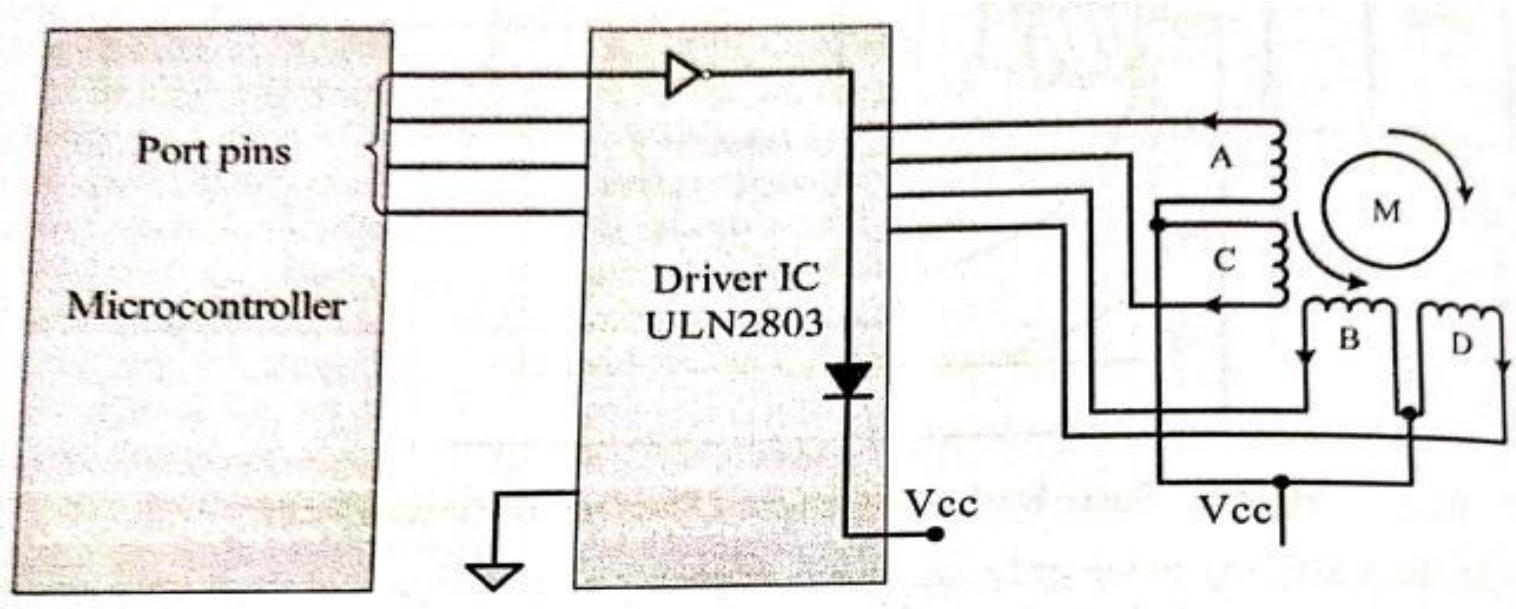
| Wave Step | | | |
|---|---|---|---|
| Coil A | Coil B | Coil C | Coil D |
| H | L | L | L |
| L | H | L | L |
| L | L | H | L |
| L | L | L | H |

| Step | Coil A | Coil B | Coil C | Coil D |
|---|---|---|---|---|
| 1 | H | L | L | L |
| 2 | H | H | L | L |
| 3 | L | H | L | L |
| 4 | L | H | H | L |
| 5 | L | L | H | L |
| 6 | L | L | H | H |
| 7 | L | L | L | H |
| 8 | H | L | L | H |

» ULN2803 is an octal peripheral driver array available from Texas Instruments and ST microelectronics for driving a 5V stepper motor.

» Simple driving circuit can also be built using transistors.

» The following circuit diagram illustrates the interfacing of a stepper motor through a driver circuit connected to the port pins of a microcontroller/ processor.



**7. a).Interface a Stepper motor and rotate it in clockwise and anti-clockwise direction.**

* A stepper motor direction is controlled by shifting the voltage across the coils.
Port lines : P0.12 to P0.15*

```
#include <LPC21xx.H>
void clock_wise(void);
 void anti_clock_wise(void);
 unsigned long int var1,var2;
 unsigned int i=0,j=0,k=0;

 int main(void)
 {
      PINSEL0 = 0x00FFFFFF;            //P0.12 to P0.15 GPIo
      IO0DIR |= 0x0000F000;       //P0.12 to P0.15 output

      while(1)
      {
           for(j=0;j<50;j++)                 // 20 times in Clock wise Rotation
                clock_wise();

           for(k=0;k<65000;k++);         // Delay to show  anti_clock Rotation

           for(j=0;j<50;j++)                 // 20 times in  Anti Clock wise Rotation
                anti_clock_wise();

           for(k=0;k<65000;k++);         // Delay to show clock Rotation
```

```
void clock_wise(void)
{
      var1 = 0x00000800;                    //For Clockwise
   for(i=0;i<=3;i++)                        // for A B C D Stepping
      {
            var1 = var1<<1;                 //For Clockwise

            IO0PIN = var1;

      for(k=0;k<3000;k++);                   //for step speed variation
     }

}

void anti_clock_wise(void)
{
      var1 = 0x00010000;              //For Anticlockwise
    for(i=0;i<=3;i++)                  // for A B C D Stepping
     {
      var1 = var1>>1;                 //For Anticlockwise


            IO0PIN = var1;
      for(k=0;k<3000;k++);                        //for step speed variation

     }
```

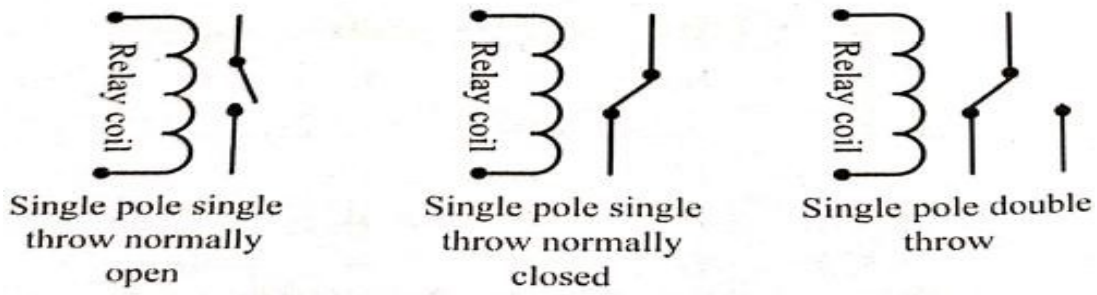
» ***Relay:*** Relay is an electro-mechanical device. In embedded application, the 'Relay' unit acts as dynamic path selectors for signals and power. The 'Relay' unit contains a relay coil made up of insulated wire on a metal core and a metal armature with one or more contacts.

» 'Relay' works on electromagnetic principle.

» When a voltage is applied to the relay coil, current flows through the coil, which in turn generates a magnetic field. The magnetic field attracts the armature core and moves the contact point. The movement of the contact point changes the power/ signal flow path.

» 'Relays' are available in different configurations.

» The following Figure illustrates the widely used relay configurationsfor embedded applications.




» The Single Pole Single Throw configuration has only one path for information flow. The path is either open or dosed in normal condition.

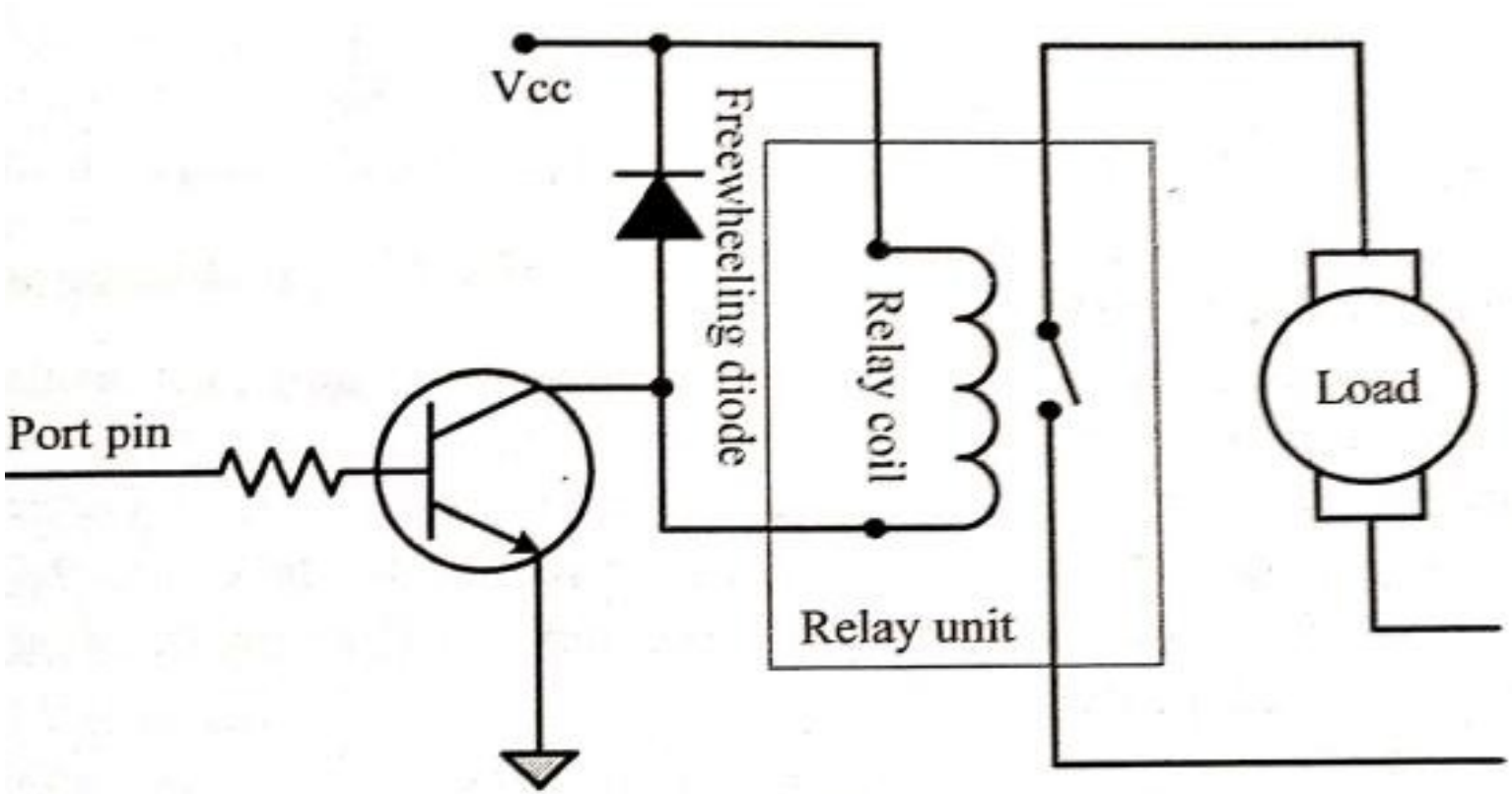
» For *normally open Single Pole Single Throw relay*, the circuit is normally open and it becomes closed when the relay is energized.

» For *normally closed Single Pole Single Throw* configuration, the circuit is normally closed and it becomes open when the relay is energized.

» For Single Pole Double Throw Relay, there are two paths for information flow and they are selected by energizining or deenergizing the relay.

» The Relay is controlled using a relay driver circuit connected to the port pin of the processor/ controller. A transistor is used for building the relay driver circuit.

» The following Figure illustrates the same.



» ***Piezo Buzzer:*** *Piezo buzzer* is a piezoelectric device for generating audio indications in embedded application.

» A piezoelectric buzzer contains a piezoelectric diaphragm which produces audible sound in response to the voltage applied to it.

» Piezoelectric buzzers are available in two types. 'Self driving' and 'External driving'.

» The '*Self-driving*' circuit contains all the necessary components to generate sound at a predefined tone.

» External driving piezo buzzers supports the generation of different tones.

» A piezo buzzer can be directly interfaced to the port pin of the processor/ control. Depending on the driving current requirements, the piezo buzzer can also be interfaced using a transistor based driver circuit as in the case of a 'Relay'.

» ***Push Button Switch:*** *Push button switch* is an input device. Push button switch comes in two configurations, namely 'Push to Make' and 'Push to Break'.

» In the '*Push to Make*' configuration, the switch is in the open state and it makes a circuit contact when it is pushed or pressed.

» In the '*Push to Break*' configuration, the switch is in the closed state and it breaks the circuit contact when it is pushed or pressed.

» The push button stays in the 'closed' (for Push to Make type) or 'open' (For Push to Break type) state.

» Push button is used for generating a momentary pulse. In embedded application push button is generally used as reset and start switch and pulse generator. The Push button is normally connected to the port pin of the host processor/ controller.

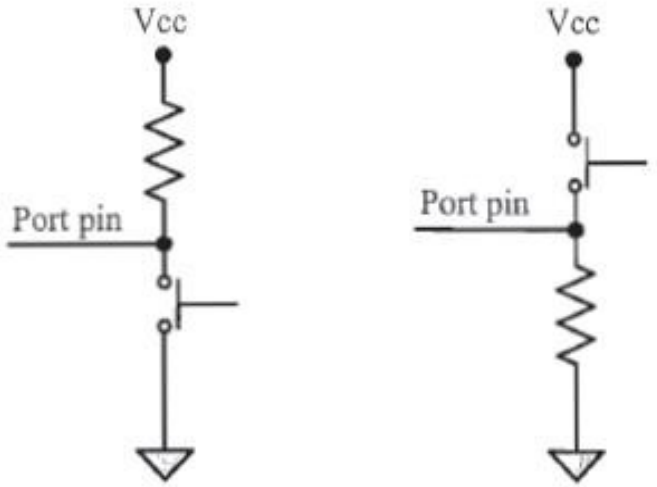
» Depending on the way in which the push button interfaced to the controller, it can generate either a 'HIGH' pulse or a 'LOW' pulse

» Push button is used for generating a momentary pulse.

» In embedded application push button is generally used as reset and start switch and pulse generator. The Push button is normally connected to the port pin of the host processor/ controller.

» Depending on the way in which the push button interfaced to the controller, it can generate either a 'HIGH' pulse or a 'LOW' pulse.

» The following Figure Illustrates how the push button can be used for generating 'LOW' and 'HIGH' pulses.




'LOW' Pulse generator 'HIGH' Pulse generator

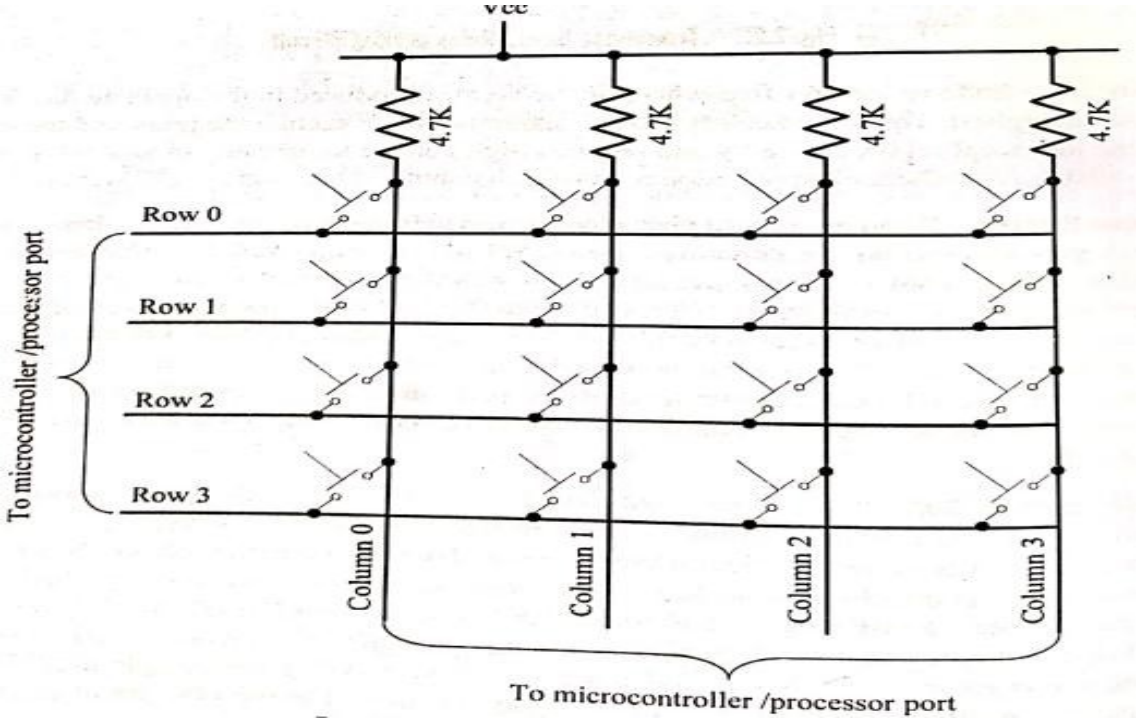
» ***Keyboard:*** *Keyboard* is an input device 'HIGH' Pulse generator for user interfacing.

» If the number of keys required is very limited, push button switches can be used and they can be directly interfaced to the port pins for reading.

- However, there may be situations demanding a large number of keys for user input (e.g. PDA device with alpha-numeric keypad for user data entry).
  - In such situations it may not be possible to interface each keys to a port pin due to the limitation in the number of general purpose port pins available for the processor/ controller in use and moreover it is wastage of port pins.
  - Matrix keyboard is an optimum solution for handling large key requirement. It greatly reduces the number of interface connections.

» For example, for interfacing 16 keys, in the direct interfacing technique, 16 port pins are required, whereas in the matrix keyboard only 8 lines are required.

» The 16 keys are arranged in a 4 column x 4 Row matrix. The following Figure illustrates the connection o keys in a matrix keyboard.



» In a matrix keyboard, the keys are arranged in matrix fashion. For detecting a key press, the keyboard uses the scanning technique, where each row of the matrix is pulled low and the columns are read. After reading the status of each columns corresponding to a row, the row is pulled high and the next row is pulled low and the status of the columns are read.

» This process is repeated until the scanning for all rows are completed. When a row is pulled low and if a key connected to the row is pressed, reading the column to which the key is connected will give logic 0. Since keys are mechanical devices, proper key de-bouncing technique should be applied.

» ***Programmable Peripheral Interface (PPI):***

*PPI* devices are used for extending the I/O capabilities of processors/ controllers.

» Most of the processors/ controllers provide very limited number of I/O and data ports.

*Peripheral*

» A programmable peripheral interface device expands the I/O capabilities of the processor/ controller. ***8255A*** is a popular PPI device for 8-bit processors/ controllers.

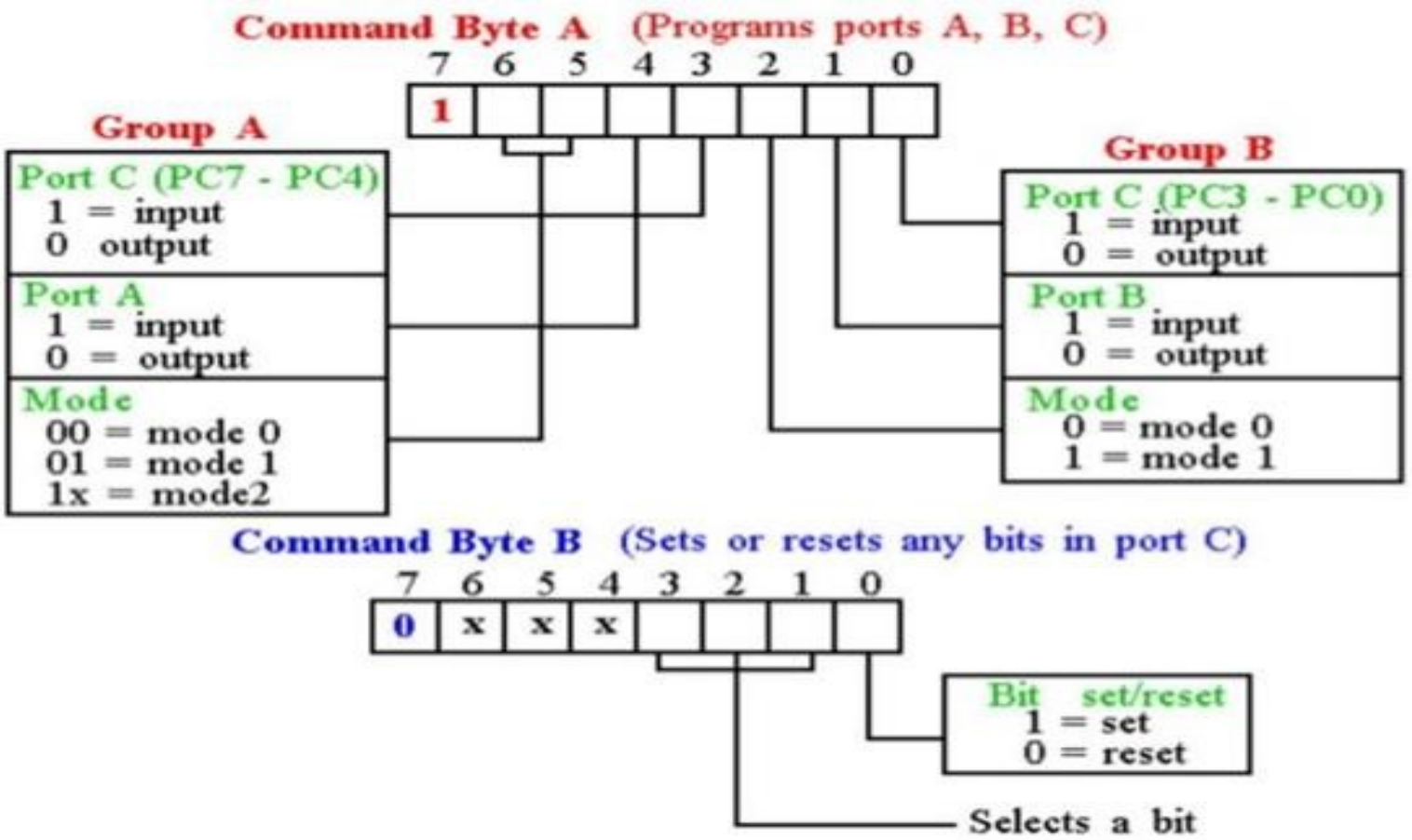
» ***8255A*** supports 24 I/O pins, and these I/O pins can be grouped as either three 8-bit parallel ports (*Port A, Port B and Port C*) or two 8-bit parallel ports (Port A and Port B) with Port C in any one of the following configurations:

» (1) As 8 individual I/O pins

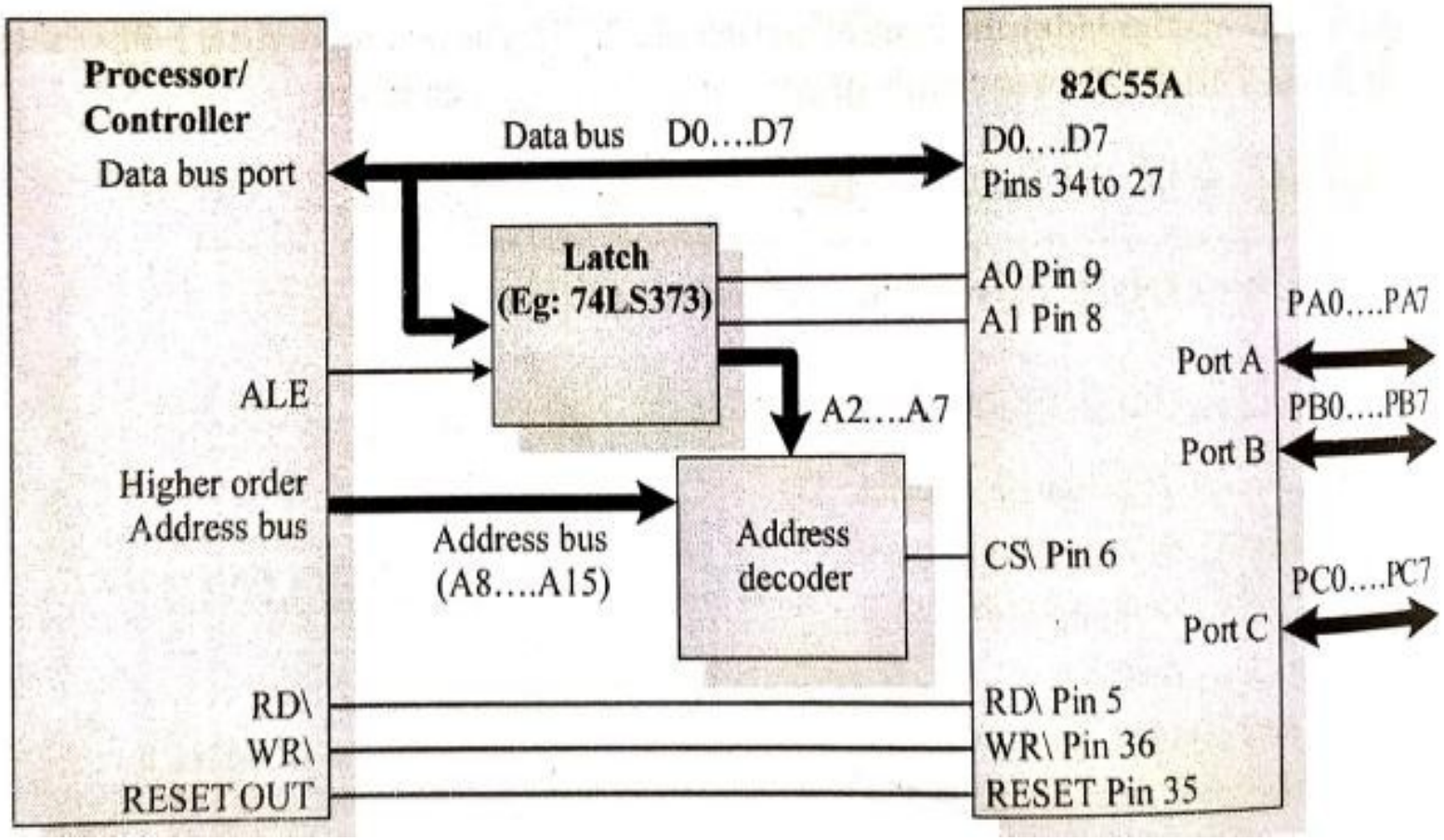
» (2) Two 4-bit ports; namely Port $C_{UPPER}$ (Cu) and Port $C_{LOWER}$ (CL).

» This is configured by manipulating the control register of 8255A. The ***control register*** holds the configuration for Port A, Port B and Port C.

» The bit details of control register is given in the following Figure:



» The following Figure illustrates the generic interfacing of a 8255A device with an 8-bit processor/ controller with 16-bit address bus (Lower order Address bus is multiplexed with data bus).



# COMMUNICATION INTERFACE

» ***Communication Interface*** is essential for communicating with various subsystems of the embedded system and with the external world.

» For an embedded product, the communication interface can be viewed in two different perspectives –

» Device/ Board level communication interface (Onboard Communication Interface)

» Product level communication interface (External Communication Interface)

» Device/ Board level communication interface (Onboard Communication Interface)

» Embedded product is a combination of different types of components (chips/ devices) arranged on a printed circuit board (PCB).

» The communication channel which interconnects the various components within an embedded product is referred as *Device/ Board Level Communication Interface* (*Onboard Communication Interface*).

- Serial interfaces like ***I2C, SPI, UART, 1-Wire***, etc., and **parallel bus interface** are examples of '*Onboard Communication Interface*'.

## » Product level communication interface (External Communication Interface)

» Some embedded systems are self-contained units and they don't require any interaction and data transfer with other sub-systems or external world.

» On the other hand, certain embedded systems may be a part of a large distributed system and they require interaction and data transfer between various devices and sub-modules.

» The *Product level communication interface' (External Communication Interface* is responsible for data transfer between the embedded system and other devices or modules. The external communication interface can be –

» **a wired medium** – *Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves (RF), GPRS/ 3G/ 4GLTE*, etc.

» **a wireless media** – *RS-232C/ RS-422/ RS-485, USB, Ethernet IEEE 1394 port, Parallel port, CF-II interface, SDIO, PCMCIA/ PCIex*, etc.

» ***Onboard Communication Interfaces:***

» *Onboard Communication Interface* refers to the different communication channels/ buses for interconnecting the various integrated circuits and other peripherals within the embedded system.

- » Inter Integrated Circuit (I2C) Interfaces
- » Serial Peripheral Interface (SPI) Bus
- » Universal Asynchronous Receiver Transmitter (UART)
- » 1-Wire Interface
- » Parallel Interface

## *Inter Integrated Circuit (I2C) Bus:*

- a synchronous
- bidirectional
- half duplex
- two wire serial interface bus.

» developed by *'Philips Semiconductors'* in the early 1980s.

- to provide an easy way of connection between a microprocessor/ microcontroller system and the peripheral chips in television sets.

» The I2C bus comprise of two bus lines, namely; *Serial Clock-SCL* and *Serial Data-SDA*.

- *SCL* line : generates synchronization clock pulses.
- *SDA* : transmits the serial data across devices.

» I2C bus is a shared bus system to which many number of I2C devices can be connected.

» Devices connected to the I2C bus can act as either *'Master' device* or *'Slave' device*.

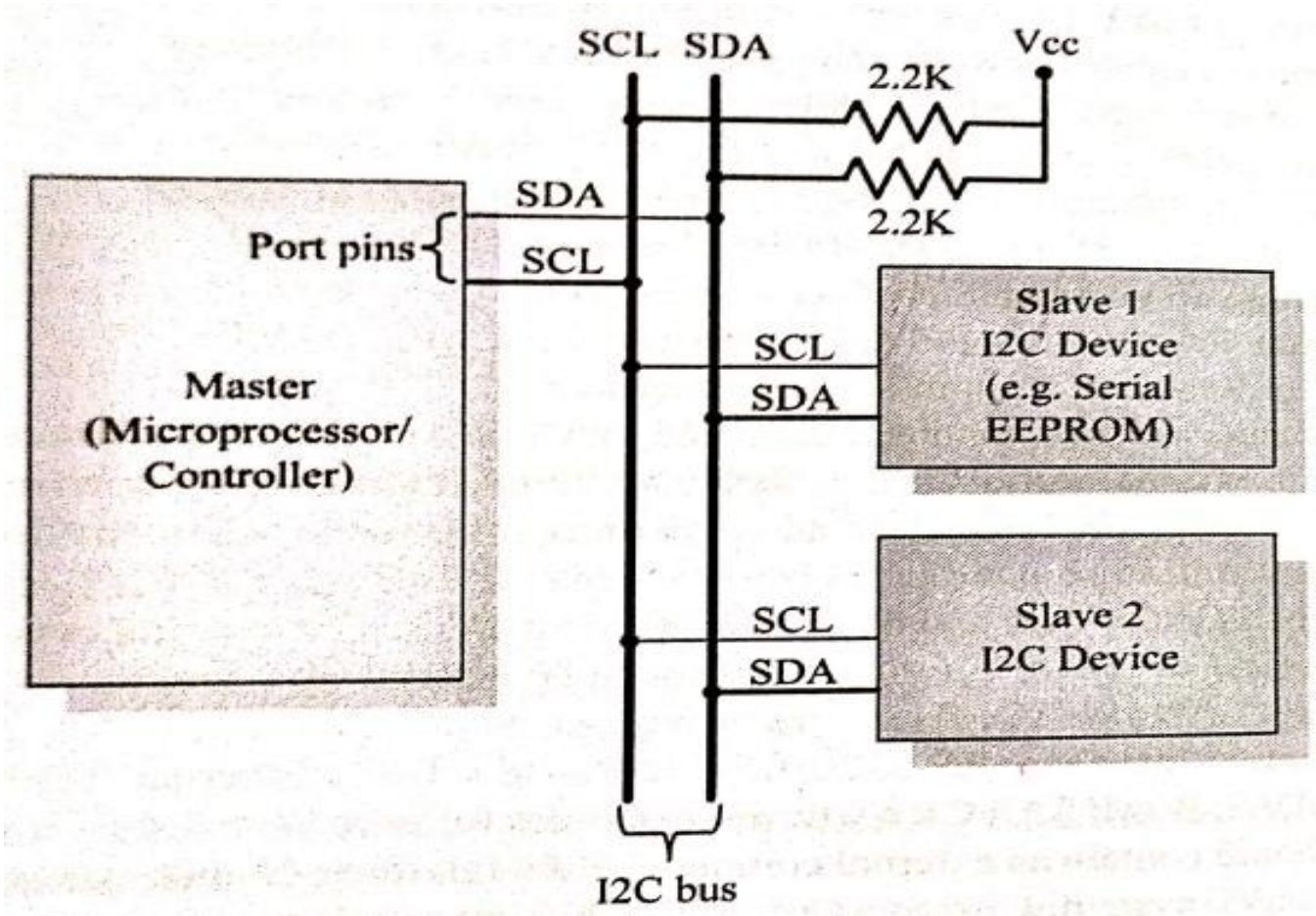
- *'Master' device* controls the communication
  - initiates/ terminates data transfer,
  - sends data and
  - generates necessary synchronization clock pulses.
- *'Slave' devices* wait for the commands from the master and respond upon receiving the commands.

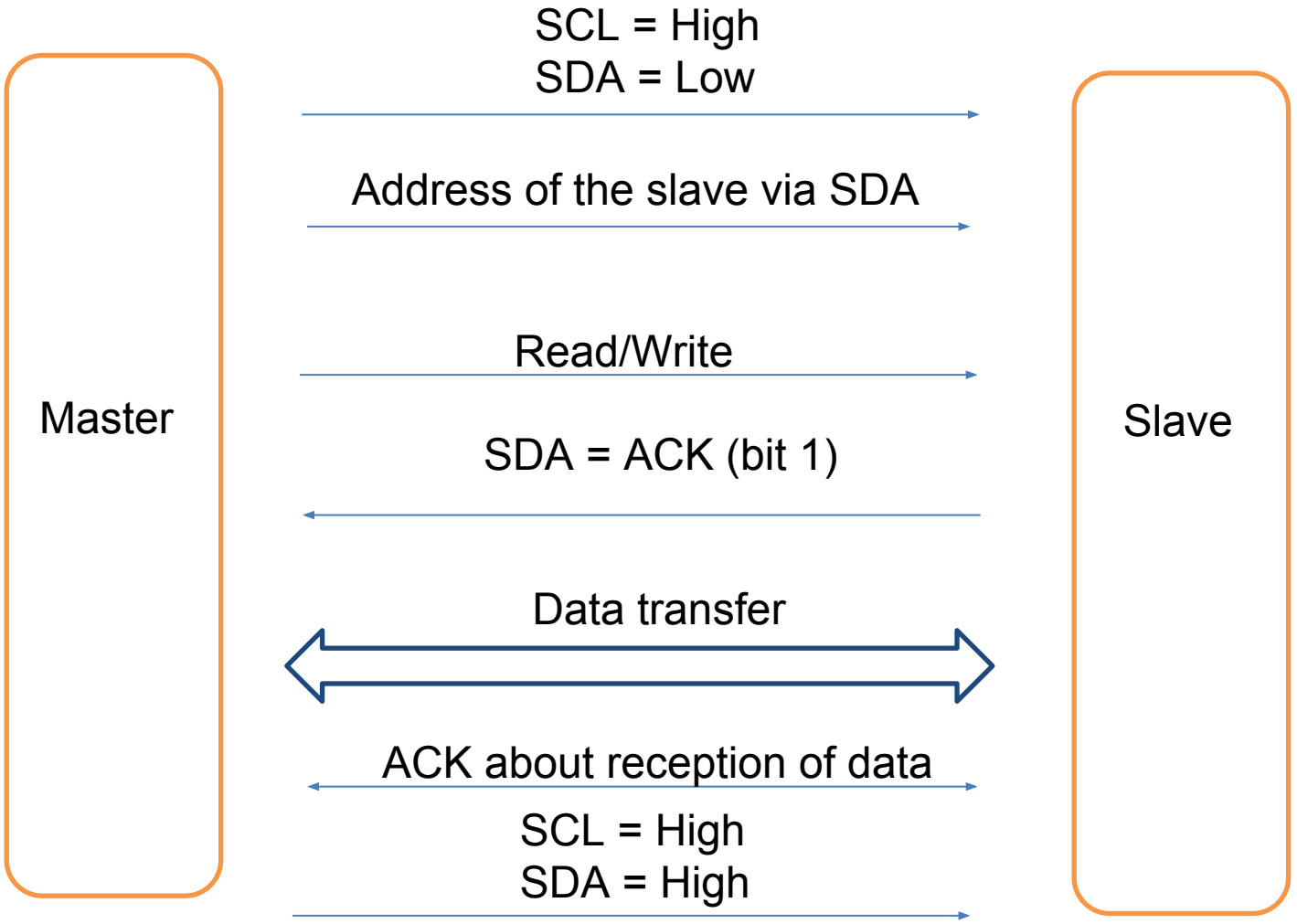
» 'Master' and 'Slave' devices can act as either transmitter or receiver;

» Synchronization clock signal is generated by the 'Master' device only.

» I2C supports multi-masters on the same bus.

» The following Figure shows bus interface diagram, which illustrates the connection of master and slave devices on the I2C bus.




SCL = High
SDA = Low
Address of the slave via SDA
Read/Write
SDA = ACK (bit 1)
Data transfer
ACK about reception of data
SCL = High
SDA = High
Master
Slave

» The sequence of operations for communicating with an I2C slave device is listed below:

1. The master device pulls the clock line (SCL) of the bus to 'HIGH ‘
2. The master device pulls the data line (SDA) 'LOW', when the SCL line is at logic 'HIGH' (This is the 'Start' condition for data transfer)
3. The master device sends the address (7-bit or 10-bit wide) of the 'slave' device to which it wants to communicate, over the SDA line. Clock pulses are generated at the SCL line for synchronizing the bit reception by the slave device.
4. The master device sends the Read or Write bit (Bit value = 1 Read operation; Bit value = 0 Write operation) according to the requirement

5. The master device waits for the acknowledgement bit from the slave device
6. The slave device with the address requested by the master device responds by sending an acknowledge bit (Bit value = 1) over the SDA line
7. Upon receiving the acknowledge bit, the master device sends the 8-bit data to the slave device over SDA line, if the requested operation is 'Write to device'. If the requested operation is 'Read from device', the slave device sends data to the master over the SDA line
8. The master device waits for the acknowledgement bit from the device upon byte transfer complete for a write operation and sends an acknowledge bit to the Slave device for a read operation
9. The master device terminates the transfer by pulling the SDA line 'HIGH' when the clock line SCL is at logic 'HIGH' (Indicating the 'STOP' condition).

» ***Serial Peripheral Interface (SPI):*** *Serial Peripheral Interface Bus (SPI)* is asynchronous bi-directional full duplex four-wire serial interface bus. The concept of SPI was introduced by Motorola.

» SPI is a single master multi-slave system. It is possible to have a system where more than one SPI device can be master, provided the condition only one master device is active at any given point of time, is satisfied.

» SPI requires four signal lines for communication. They are:

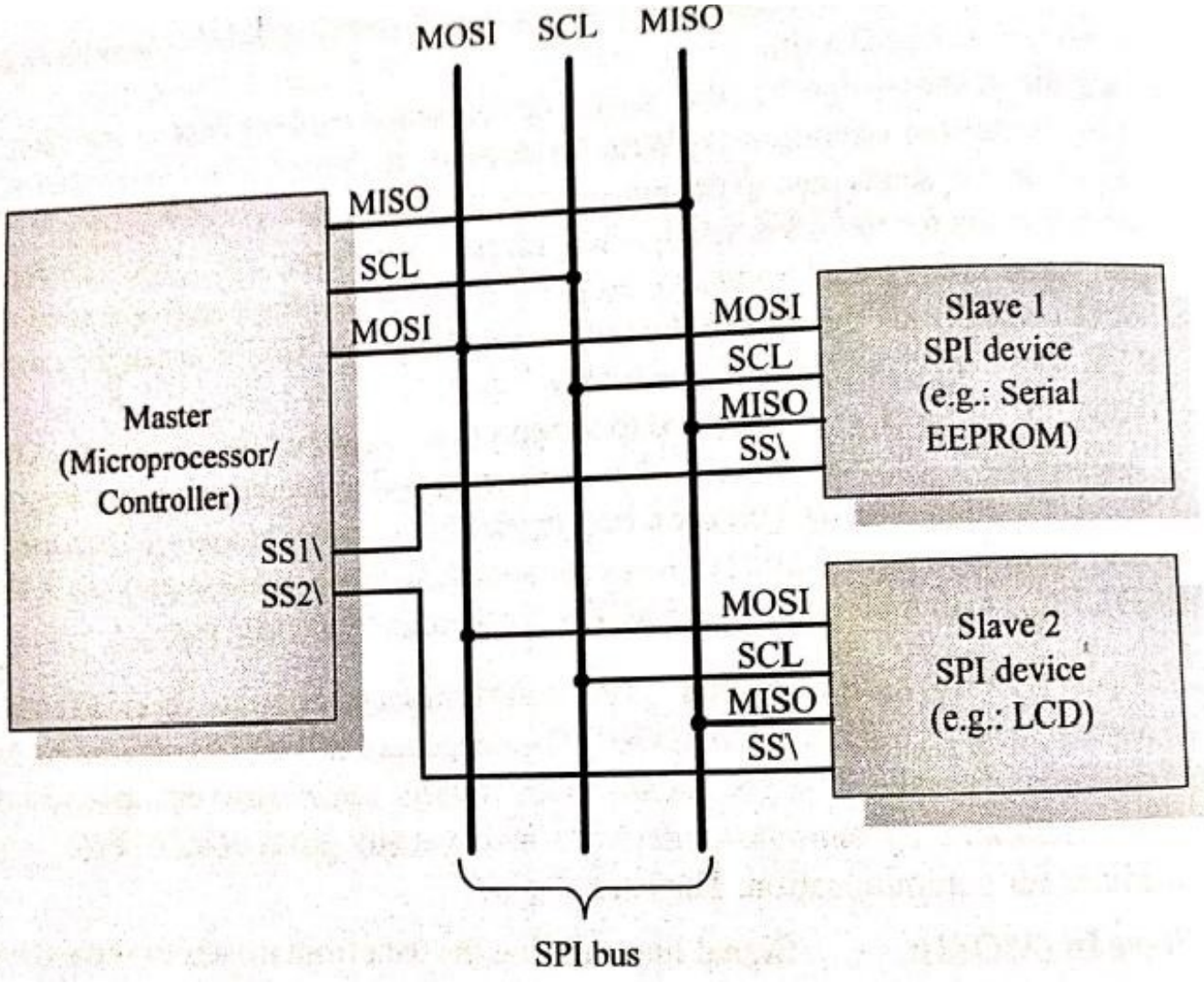
» *Master Out Slave In (MOSI):* Signal line carrying the data from master to slave device. It is also known as Slave Input/Slave Data In (SI/SDI).

» *Master In Slave Out (MISO):* Signal line carrying the data from slave to master device. It is also known as Slave Output (SO/ SDO).

» *Serial Clock (SCL):* Signal line carrying the clock signals

» *Slave Select (SS):* Signal line for slave device select. It is an active low signal.

» The bus interface diagram is shown in the following Figure, illustrates the connection of master and slave devices on the SPI bus.



» The master device is responsible for generating the clock signal. It selects the required slave device by asserting the corresponding slave device's slave select signal 'LOW'. The data out line (MISO) of all the slave devices when not selected floats at high impedance state.

» SPI works on the principle of 'Shift Register'. The master and slave devices contain a special shift register for the data to transmit or receive. The size of the shift register is device dependent. Normally it is a multiple of 8.

» During transmission from the master to slave, the data in the master's shift register is shifted out to the MOSI pin and it enters the shift register of the slave device through the MOSI pin of the slave device. At the same time, the shifted out data bit from the slave device's shift register enters the shift register of the master device through MISO pin. In summary, the shift registers of 'master' and 'slave' devices form a circular buffer.

» When compared to I2C, SPI bus is most suitable for applications requiring transfer of data in 'streams'. The only limitation is SPI doesn’t support an acknowledgement mechanism.

» ***Universal Asynchronous Receiver Transmitter (UART):*** *Universal Asynchronous Receiver Transmitter (UART)* based data transmission is an asynchronous form of serial data transmission.

» UART based serial data transmission doesn't require a clock signal to synchronize the transmitting end arid receiving end for transmission. Instead it relies upon the pre-defined agreement between the transmitting device and receiving device.

» The serial communication settings (Baud rate, number of bits per byte, parity, number of start bits and stop bit and flow control) for both transmitter and receiver should be set as identical.

» The start and stop of communication is indicated through inserting special bits in the data stream.

» While sending a byte of data, a start bit is added first and a stop bit is added at the end of the bit stream. The least significant bit of the data byte follows the 'start' bit.

» The 'start' bit informs the receiver that a data byte is about to arrive. The receiver device starts polling its 'receive line' as per the baud rate settings.

» If the baud rate is '*x*' bits per second, the time slot available for one bit is *1/x* seconds.

» The receiver unit polls the receiver line at exactly half of the time slot available for the bit.

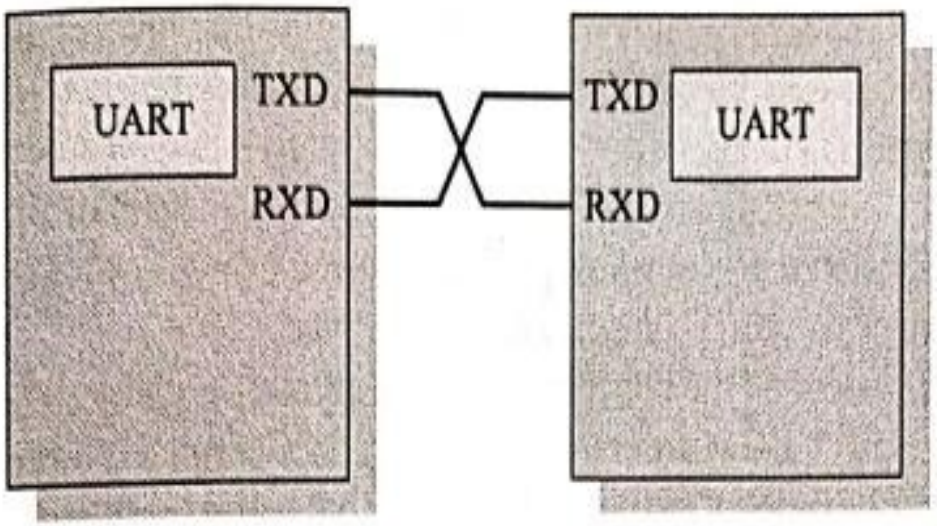
» If parity is enabled for communication, the UART of the transmitting device adds a **parity bit** (bit value is 1 for odd number of 1s in the transmitted bit stream and 0 for even number of 1s). .

» The UART of the receiving device calculates the parity of the bits received and compares it with the received parity bit for error checking.

» The UART of the receiving device discards the 'Start', 'Stop' and 'Parity' bit from the received bit stream and converts the received serial bit data to a word.

» For proper communication, the 'Transmit line' of the sending device should be connected to the 'Receive line' of the receiving device.

» The following Figure illustrates the same.




TXD: Transmitter line
RXD: Receiver line

» In addition to the serial data transmission function, UART provides hardware handshaking signal support for controlling the serial data flow.

» UART chips are available from different semiconductor manufacturers.

» National Semiconductor's 8250 UART chip is considered as the standard setting UART. It was used in the original IBM PC.

» ***1-Wire Interface:*** *1-wire interface* is an asynchronous half-duplex communication protocol developed by Maxim Dallas Semiconductor. It is also known as ***Dallas 1-Wire® protocol***.

» It makes use of only a single signal line (wire) called DQ for communication and follows the master-slave communication model.

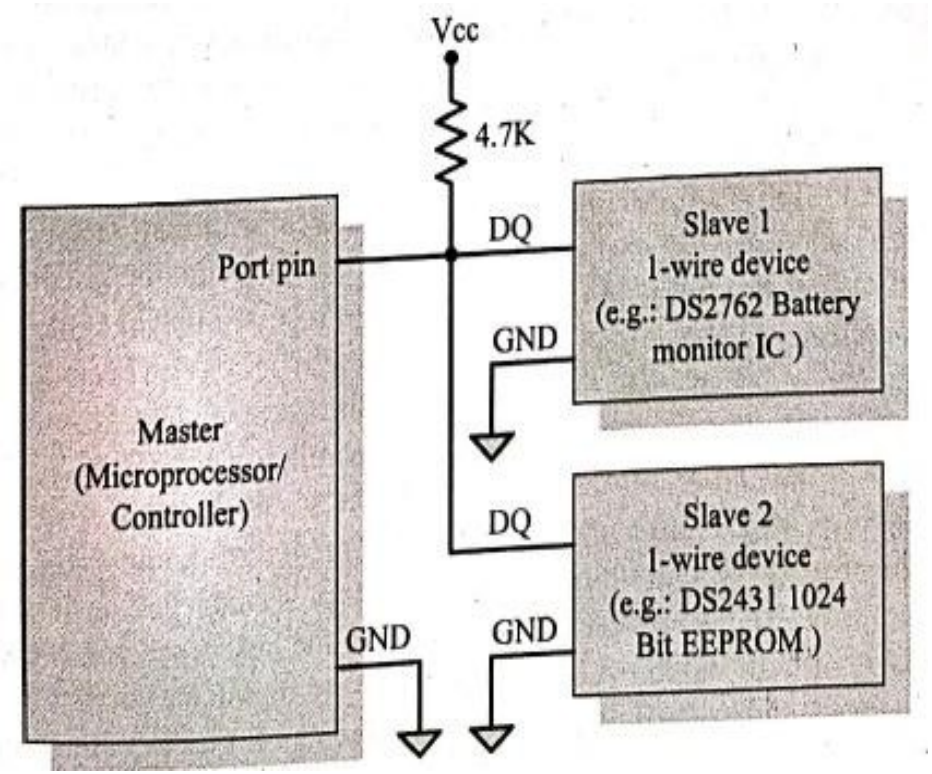
» One of the key feature of 1-wire bus is that it allows power to be sent along the signal wire as well.

» The 1-wire slave devices incorporate internal capacitor (typically of the order of 800 pF) to power the device from the signal line.

» The 1-wire interface supports a single master and one or more slave devices on the bus.

» The bus interface diagram shown in the following Figure illustrates the connection of master and slave devices on the 1-wire bus.

» Every 1-wire device contains a globally unique 64-bit identification number stored within it. This unique identification number can be used for addressing individual devices present on the bus in case there are multiple slave devices




» The identifier has three parts: an 8-bit family code, a 48-bit serial number and an 8-bit CRC computed from the first 56-bits.

» The sequence of operation for communicating with a 1-wire slave device is listed below:

1. The master device sends a 'Reset' pulse on the 1-wire bus.
2. The slave device(s) present on the bus respond with a 'Presence' pulse.
3. The master device sends a ROM command (Net Address Command followed by the 64-bit address of the device). This addresses the slave device(s) to which it wants to initiate a communication.
4. The master device sends a read/ write function command to read/ write the internal memory or register of the slave device.
5. The master initiates a Read data/ Write data from the device or to the device.

» All communication over the 1-wire bus is master initiated.

» The communication over the 1-wire bus is divided into time slots of 60 microseconds for regular speed mode of operation (16.3Kbps).

***Parallel Interface***:

» on-board parallel interface that connects the host/controller to the peripheral device.

» peripheral devices are memory mapped to the host of the system.

» The 'Control Signals' includes 'Read/ Write' and chip select signal.

» Read/Write determines the direction of data transfer

» An address decoder circuit is used for generating the chip select signal for the device.

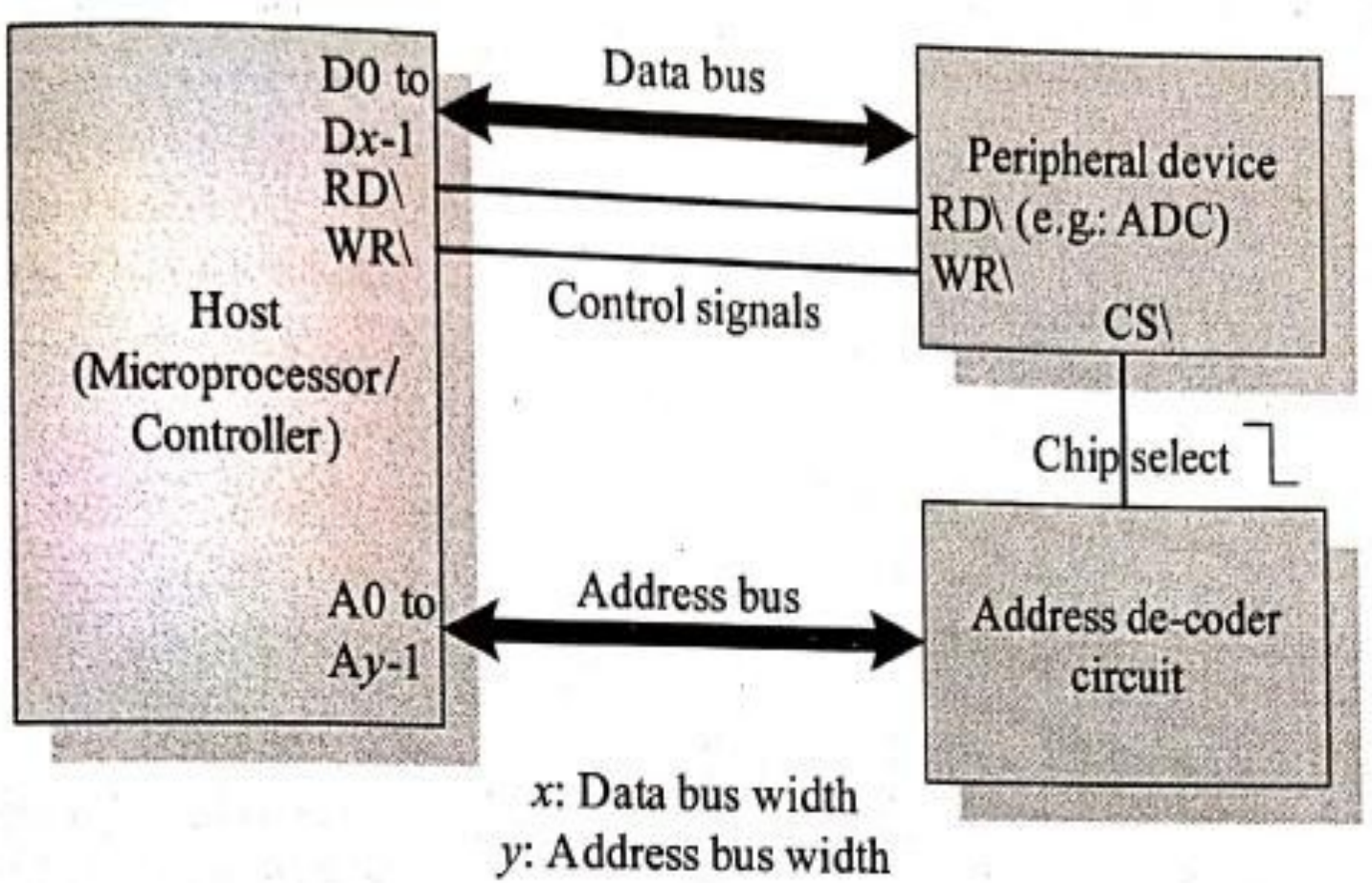
» The device becomes active only when this line is asserted by the host processor.

Strict timing characteristics are followed for parallel communication.

» parallel communication is host processor initiated. If a device wants to initiate the communication, it can inform the same to the processor through interrupts.

» The width of the parallel interface is determined by the data bus width of the host processor. The bus width supported by the device should be same as that of the host processor.

» Parallel data communication offers the highest speed for data transfer.



» ***External Communication Interfaces:***

» The *External Communication Interface* refers to the different communication channels/ buses used by the embedded system to communicate with the external world.

- » RS-232 C & RS-485
- » Universal Serial Bus (USB)
- » IEEE 1394 (Firewire)
- » Infrared (IrDA)
- » Bluetooth (BT)
- » Wi-Fi
- » ZigBee
- » General Packet Radio Service (GPRS), 3G, 4G, LTE

***RS-232 C & RS-485:*** *RS-232 C* is a legacy, full duplex, wired, asynchronous serial communication interface with data rates around 20kbps.

» developed by the Electronics Industries Association (EIA) in early 1960s.

» RS-232 extends the UART communication signals for external data communication.

» RS-232 follows the EIA standard for bit transmission.

- a logic '0'('Space') is represented with voltage between +3 and +25V and
- a logic '1' ('Mark') is represented with voltage between -3 and -25V.

- The RS-232 interface defines various handshaking and control signals for communication apart from the 'Transmit' and. 'Receive' signal lines for data

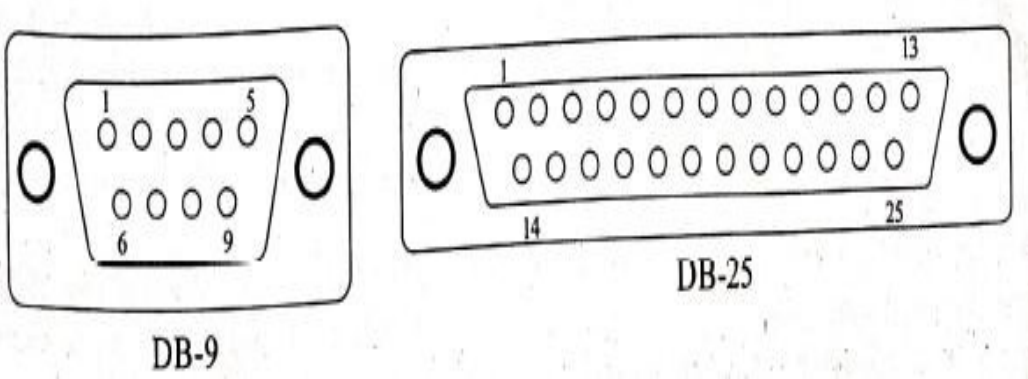
» RS-232 is a point-to-point communication interface and the device involved in RS-232 communication are called '*Data Terminal Equipment (DTE)*' and '*Data Communication Equipment (DCE)*'.

- The **Data Terminal Ready (DTR)** signal is activated by DTE when it is ready to accept data.
- The **Data Set Ready (DSR)** is activated by DCE when it is ready for establishing a communication link.
- DTR should be in the activated state before the activation of DSR.
- The **Data Carrier Detect (DCD)** control signal is used by the DCE to indicate the DTE that a good signal is being received.

» RS-232 supports two different types of connectors:

- DB-9: 9-Pin connector and
- DB-25: 25-Pin connector.

»



» RS-232 supports only point-to-point communication and not suitable for multi-drop communication.

» It is more susceptible to noise and it greatly reduces the operating distance to around 50 ft.

» RS-422 is another serial interface standard from EIA. It supports data rates up to l00Kbps and distance up to 400 ft.

» RS-422 supports multi-drop communication with one transmitter device and receiver devices up to 10.

» RS-485 is the enhanced version of RS-422 and it supports multi-drop communication with up to 32 transmitting devices (drivers) and 32 receiving devices on the bus. The communication between devices in the bus uses the 'addressing' mechanism to identify slave devices.

» ***Universal Serial Bus (USB):*** *Universal Serial Bus* is a wired high speed serial bus for data communication.

» The first version of USB (*USB 1.0*) was released in 1995 and was created by the USB core group members consisting of Intel, Microsoft, IBM, Compaq, Digital and Northern Telecom.

» The USB communication system follows a **star topology** with a USB host at the centre and one or more USB peripheral devices/ USB hosts connected to it.

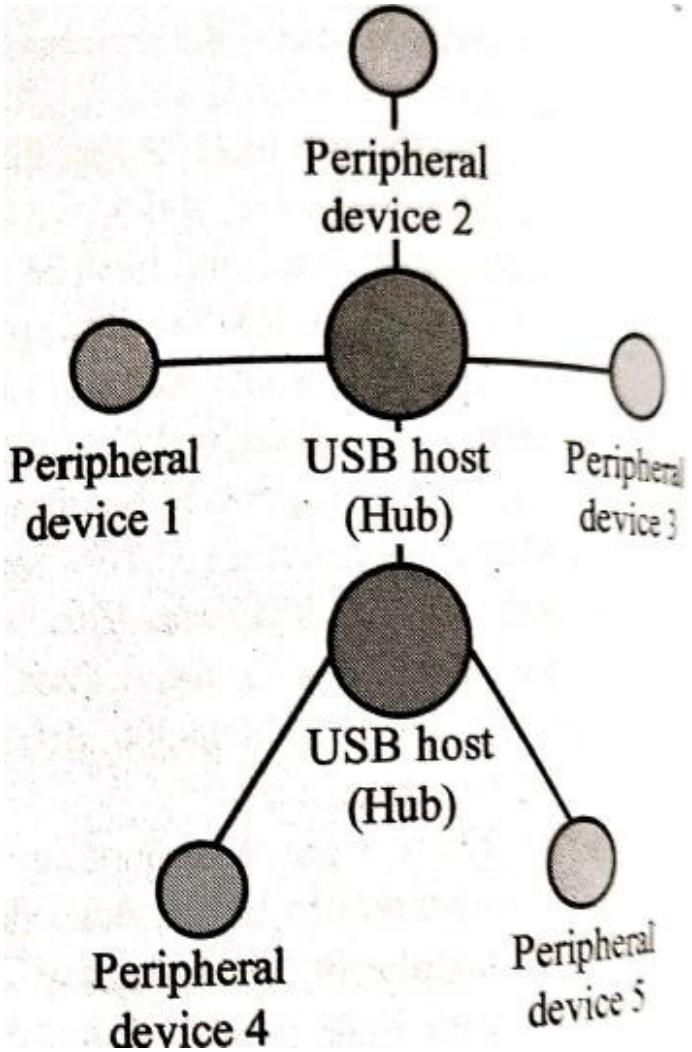
» A USB 2.0 host can support connections up to 127, including slave peripheral devices and other USB hosts.

» The following Figure illustrates the star topology for USB device connection.

» USB transmits data in packet format.

» The USB communication is a host initiated one.

» The USB host contains a host controller which is responsible for controlling the data communication, including establishing connectivity with USB slave devices, packetizing and formatting the data.



» There are different standards for implementing the USB Host Control interface; namely Open Host Control Interface (OHCI) and Universal Host Control Interface (UHCI).

» The physical connection between a USB peripheral device and master device is established with a USB cable. The USB cable in USB 2.0 supports communication distance of up to 5 meters.

» The USB 2.0 standard uses two different types of connector at the ends of the USB cable for connecting the USB peripheral device and host device.

- 'Type A' connector is used for upstream connection (connection with host) and Type B connector is used for downstream connection (connection with slave device).
- The USB connector present in desktop PCs or laptops are examples for 'Type A' USB connector.
- Both Type A and Type B connectors contain 4 pins for communication.

» The Pin details for the connectors are listed in the table given below.

| Pin No. | Pin Name | Description |
|---|---|---|
| 1 | $V_{BUS}$ | Carries power (5V) |
| 2 | D– | Differential data carrier line |
| 3 | D+ | Differential data carrier line |
| 4 | GND | Ground signal line |

» USB uses differential signals for data transmission. It provides noise immunity.

» USB interface has the ability to supply power to the connecting devices. Two connection lines (Ground and Power) of the USB interface are dedicated for carrying power. It can supply power up to 500 rnA at 5 V.

» USB supports four different types of data transfers, namely; Control, Bulk, Isochronous and Interrupt.

» *Control transfer* is used by USB system software to query, configure and issue commands to the USB device.

» *Bulk transfer* is used for sending a block of data to a device. Bulk transfer supports **error checking and correction**.

- Transferring data to a printer is an example for bulk transfer.

» *Isochronous data transfer* is used for real-time data communication as data streams. Isochronous transfer doesn't support error checking and retransmission of data in case of any transmission loss.

- All streaming devices like audio devices and medical equipment for data collection make use of the isochronous transfer.

» *Interrupt transfer* is used for transferring small amount of data.

» Interrupt transfer mechanism makes use of polling technique to see whether the USB device has any data to send.

» The frequency of polling is determined by the USB device and it varies from 1 to 255 milliseconds.

- Devices like Mouse and Keyboard, which transmits fewer amounts of data, uses Interrupt transfer.

» ***IEEE 1394 (Firewire):*** *IEEE 1394* is a wired isochronous high speed serial communication bus. It is also known as ***High Performance Serial Bus (HPSB)***.

» The research on 1394 was started by Apple Inc. in 1985 and the standard for this was coined by IEEE.

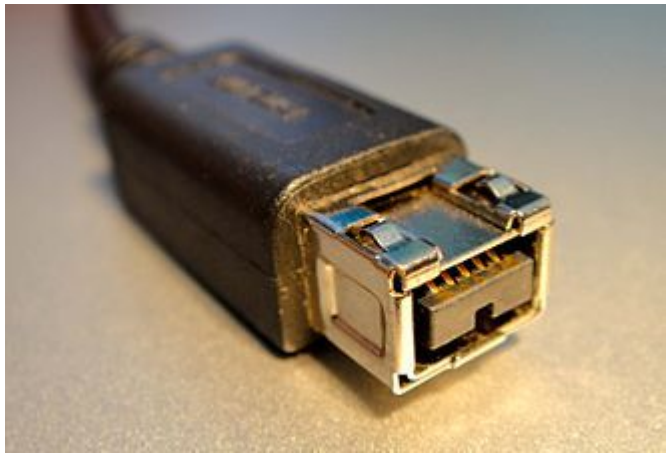
» The implementation of it is available from various players with different names.

- Apple Inc's implementation of 1394 protocol is popularly known as *Firewire*.
- *i.LINK* is the 1394 implementation from Sony Corporation
- *Lynx* is the implementation from Texas Instruments.

» 1394 supports peer-to-peer connection and point-to-multipoint communication allowing 63 devices to be connected on the bus in a tree topology. 1394 is a wired serial interface and it can support a cable length of up to 15 feet for interconnection.

» There are two differential data transfer lines *A* and *B* per connector.

»» 1394 is a popular communication interface for connecting embedded devices like Digital Camera, Camcorder, Scanners to desktop computers for data transfer and storage.



» Unlike USB interface (except USB OTG), IEEE 1394 doesn't require a host for communicating between devices.

- For example, you can directly connect a scanner with a printer for printing.

» The data- rate supported by 1394 is far higher than the one supported by USB2.0 interface.

» The 1394 hardware implementation is much costlier than USB implementation.

» ***Infrared (IrDA):*** *Infrared* is a serial, half duplex, line of sight based wireless technology for data communication between devices.

» IrDA is in use from the olden days of communication and you may be very familiar with it.

- The remote control of your TV, VCD player, etc., works on Infrared data communication principle.

» Infrared communication technique uses infrared waves of the electromagnetic spectrum for transmitting the data.

» IrDA supports **point-point and point-to-multipoint communication**, provided all devices involved in the communication are within the line of sight.

» The typical communication range for IrDA lies in the range l0 cm to 1 m. The range can be increased by increasing the transmitting power of the IR device.

» IR supports data rates ranging from 9600bits/second to 16Mbps.

» Depending on the speed of data transmission IR is classified into Serial IR (SIR), Medium IR (MIR), Fast IR (FIR), Very Fast IR (VFIR) and Ultra Fast IR (UFIR).

- SIR supports transmission rates ranging from 9600bps to 115.2kbps.
- MIR supports data rates of 0.576Mbps and 1.152Mbps.
- FIR supports data rates up to 4Mbps.
- VFIR is designed to support high data rates up to l6Mbps.
- The UFIR supports up to 96Mbps.

» IrDA communication involves a transmitter unit for transmitting the data over IR and a receiver for receiving the data.

» Infrared Light Emitting Diode (LED) is the IR source for transmitter and at the receiving end a photodiode acts as the receiver.

» Both transmitter and receiver unit will be present in each device supporting IrDA communication for bidirectional data transfer. Such IR units are known as '*Transceiver*'.

» Certain devices like a TV require control always require unidirectional communication and so they contain either the transmitter or receiver unit (The remote control unit contains the transmitter unit and TV contains the receiver unit).

» ***Bluetooth (BT):*** *Bluetooth* is a low cost, low power, short range wireless technology for data and voice communication.

» Bluetooth was first proposed by 'Ericsson' in 1994.

» Bluetooth operates at 2.4GHz of the Radio Frequency spectrum and uses the Frequency Hopping Spread Spectrum (FHSS) technique for communication.

» Literally it supports a data rate of up to 1Mbps and a range of approximately 30 to 100 feet (version dependent) for data communication.

» Bluetooth communication also has two essential parts; a physical link part and a protocol part.

- The *physical link* is responsible for the physical transmission of data between devices supporting Bluetooth communication. The physical link works on the wireless principle making use of RF waves for communication. Bluetooth enabled devices essentially contain a Bluetooth wireless radio for the transmission and reception of data.
- The *protocol part* is responsible for defining the rules of communication. The rules governing the Bluetooth communication is implemented in the '*Bluetooth protocol stack*'.

» Each Bluetooth device will have a 48-bit unique identification number.

Bluetooth communication follows packet based data transfer.

» Bluetooth supports point-to-point (device to device) and point-to-multipoint (device to multiple device broadcasting) wireless communication.

» The point-to-point communication follows the master slave relationship. A Bluetooth device can function as either master or slave.

» When a network is formed with one Bluetooth device as master and more than one device as slaves, it is called a ***Piconet***.

» A Piconet supports a maximum of seven slave devices.

» ***Wi-Fi:*** *Wi-Fi* or *Wireless Fidelity* is the popular wireless communication technique for networked communication of devices.

» Wi-Fi follows the IEEE 802.11 standard. Wi-Fi is intended for network communication and supports Internet Protocol (IP) based communication. It is essential to have device identities in a multi-point communication to address specific devices for data communication.

» In an IP based communication each device is identified by an IP address, which is unique to each device on the network.

» Wi-Fi based communications require an intermediate agent called Wi-Fi router/ Wireless Access point to manage the communications.

- The Wi-Fi router is responsible for restricting the access to a network, assigning IP address to devices on the network, routing data packets to the intended devices on the network.

» Wi-Fi enabled devices contain a wireless adaptor for transmitting and receiving data in the form of radio signals through an antenna. The hardware part of it is known as Wi-Fi Radio.

» Wi-Fi operates at 2.4GHz or 5GHz of radio spectrum and they co-exist with other ISM band devices like Bluetooth.

» The following Figure illustrates the typical interfacing of devices in a Wi-Fi network.

» ***ZigBee:*** *ZigBee* is a low power, low cost, wireless network communication protocol based on the IEEE 802.15.4-2006 standard.

» ZigBee is targeted for low power, low data rate and secure applications for Wireless Personal Area Networking (WPAN).

» The ZigBee specifications support a robust mesh network containing multiple nodes. This networking strategy makes the network reliable by permitting messages to travel through a number of different paths to get from one node to another.

» ZigBee operates worldwide at the unlicensed bands of Radio spectrum, mainly at *2.400 to 2.484 GHz*, *902 to 928 MHz* and *868.0 to 868.6 MHz*.

» ZigBee Supports an operating distance of up to 100 meters and a data rate of 20 to 250Kbps.

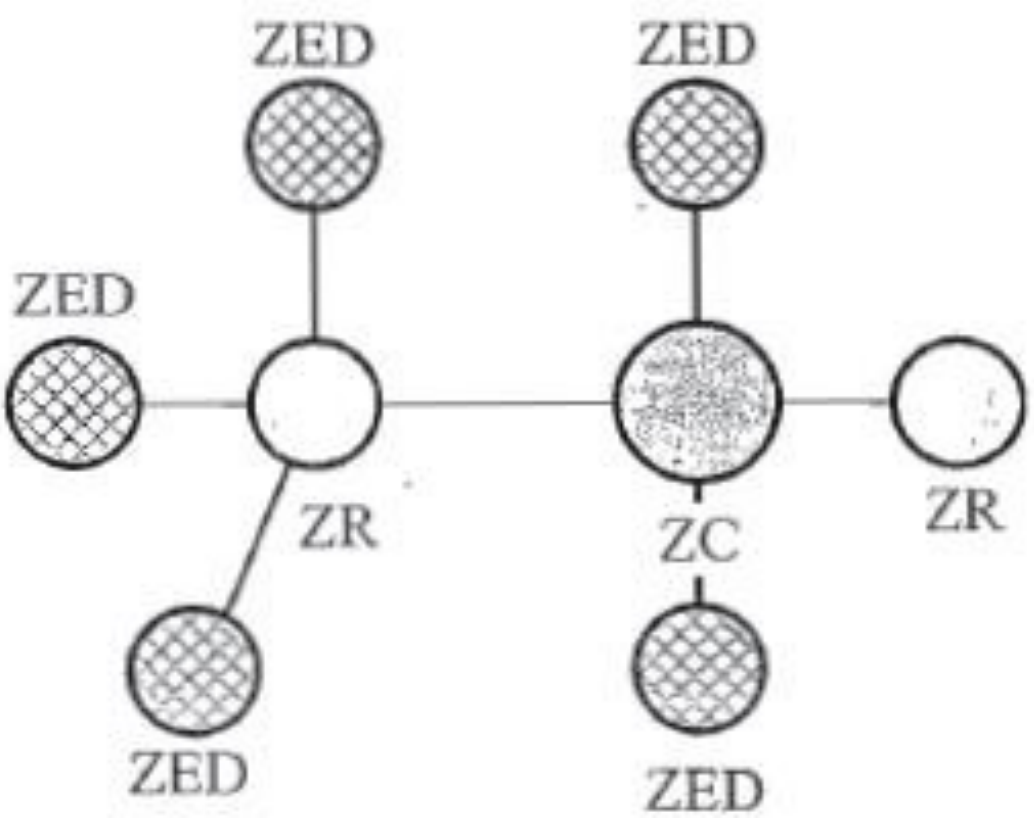
» In the ZigBee terminology, each ZigBee device falls under any one of the following ZigBee device category:

» *ZigBee Coordinator (ZC)/ Network Coordinator:* The ZigBee coordinator acts as the root of the ZigBee network. The ZC is responsible for initiating the ZigBee network and it has the capability to store information about the network.

» *ZigBee Router (ZR)/ Full Function Device (FFD):* Responsible for passing information from device to another device or to another ZR.

» *ZigBee End Device (ZED)/ Reduced Function Device (RFD):* End device containing ZigBee functionality for data communication. It can talk only with a ZR or ZC and doesn't have the capability to act as a mediator for transferring data from one device to another.

» The following Figure gives an overview of *ZigBee Coordinator (ZC)*, *ZigBee Router (ZR)* and *ZigBee End Device (ZED)* in a ZigBee network:



» ***General Packet Radio Service (GPRS), 3G, 4G, LTE:*** *General Packet Radio Service* is a communication technique for transferring data over a mobile communication network like GSM.

» Data is sent as packets in GPRS communication. The transmitting device splits the data into several related packets.

» At the receiving end the data is re-constructed by combining the received data packets.

» GPRS supports a theoretical maximum transfer rate of **171.2kbps.**

» In GPRS communication, the radio channel is concurrently shared between several users instead of dedicating a radio channel to a cell phone user. The GPRS communication divides the channel into 8 timeslots and transmits data over the available channel.

» GPRS is mainly used by mobile enabled embedded devices for data communication. The device should support the necessary GPRS hardware like GPRS modem and GPRS radio.

» To accomplish GPRS based communication, the carrier network also should have support for GPRS communication. GPRS is an old technology and it is being replaced by new generation data communication techniques which offer higher bandwidths for communication.

# EMBEDDED FIRMWARE

» ***Embedded firmware*** refers to the control algorithm (Program instructions) and or the configuration settings that an embedded system developer dumps into the code (Program) memory of the embedded system. It is an unavoidable part of an embedded system.

» There are various methods available for developing the embedded firmware.

1. Write the program in high level languages like Embedded C/ C++ using an Integrated Development Environment (IDE)

» The IDE will contain a editor, compiler, linker, debugger, simulator, etc.

» IDEs are different for different family of processors/ controllers.

» For example, Keil microvision3 IDE is used for all family member of 8051 microcontroller, since it contains the generic 8051 compiler C51.

2. Write the program in Assembly language using the instructions supported by your application's target processor/ controller.

The instruction set for each family of processor/ controller is different and the program written in either of the methods given above should be converted into a processor understandable machine code before loading it into the program memory.

» The process of converting the program to machine readable binary code is called '***HEX File Creation***'.

» The methods used for 'HEX File Creation' is different depending on the programming techniques used.

» If the program is written in Embedded C/ C++ using an IDE, the cross compiler included in the IDE converts it into corresponding processor/ controller understandable 'HEX File'.

» If you are following the Assembly language based programming technique, you can use the utilities supplied by the processor/ controller vendors to convert the source code into 'HEX File'.

» Also third party tools are available, which may be of free of cost, for this conversion.

» For a beginner in the embedded software field, it is strongly recommended to use the high level language based development technique. The reasons for this being:

» Writing codes in a high level language is easy, and highly portable which means you can use the same code to run on different processor/ controller with little or less modification. The only thing you need to do is re-compile the program with the required processor's IDE.

» Also the programs written in high level languages are not developer dependent. Any skilled programmer can trace out the functionalities of the program by just having a look at the program. It will be much easier if the source code contains necessary comments and documentation lines. It is

reduced to a greater extent.

~~very easy to debug and the overall system development time will~~ be

» The embedded software development process in assembly language is tedious and time consuming. The developer needs to know about all the instruction sets of the processor/ controller or at least s/he should carry an instruction set reference manual with her/ him. A programmer using assembly language technique writes the program according to his/ her view and taste. Often he/ she may be writing a method or functionality which can be achieved through a single instruction as an experienced person's point of view, by two or three instructions in his/ her own style. So the program will be highly dependent on the developer. It is very difficult for a second person to understand the code written in Assembly even if it is well documented.

# OTHER SYSTEM COMPONENTS

» The ***other system components*** refer to the components/ circuits/ ICs which are necessary for the proper functioning of the embedded system.

» Some of these circuits may be essential for the proper functioning of the processor/ controller and firmware execution.

» Watchdog timer, Reset IC (or passive circuit), brown-out protection IC etc., are examples of circuits/ ICs which are essential for the proper functioning of the processors/ controllers. Some of the controllers or SoCs, integrate these components within a single IC and doesn't require such components externally connected to the chip for proper functioning.

- Depending on the system requirement, the embedded system may include other integrated circuits for performing specific functions, level translator ICs for interfacing circuits with different logic levels, etc.

» ***Reset Circuit:*** The *reset circuit* is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON.

» The reset signal brings the internal registers and the different hardware systems of the processor/ controller to a known state and starts the firmware execution from the reset vector (Normally from vector address 0x0000 for conventional processors/ controllers.

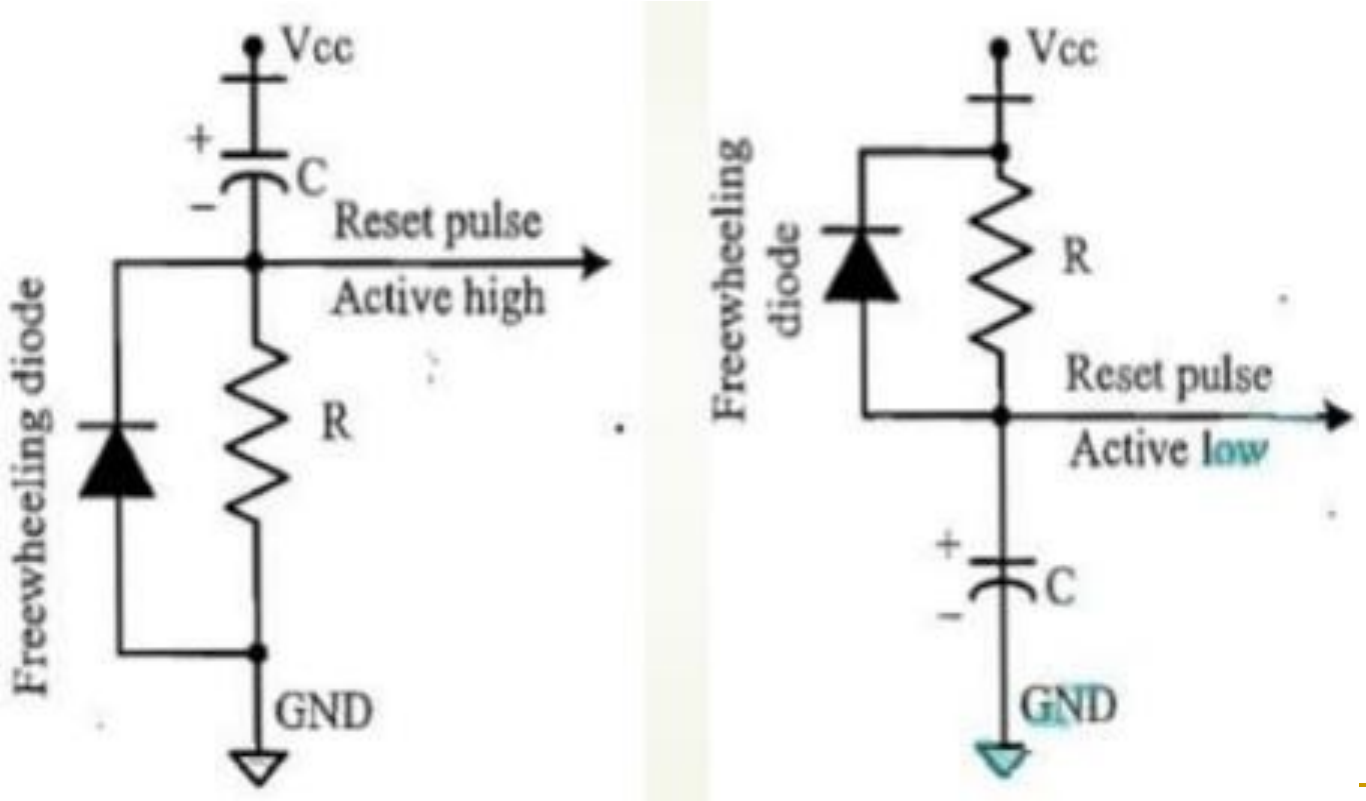
» The reset signal can be either active high or active low.

» Since the processor operation is synchronized to a clock signal, the reset pulse should be wide enough to give time for the clock oscillator to stabilize before the internal reset state starts.

» The reset signal to the processor can be applied at power ON through an external passive reset circuit comprising a Capacitor and Resistor or through a standard Reset IC like MAX810 from Maxim Dallas. Select the reset IC based on the type of reset signal and logic level (CMOS/ TTL) supported by the processor/ controller in use.

» Some microprocessors /controllers contain built-in internal reset circuitry and they don't require external reset circuitry.

» The following Figure illustrates a resistor capacitor based passive reset circuit for active high and low configurations. The reset pulse width can be adjusted by changing the resistance value *R* and capacitance value *C*.



» ***Brown-out Protection Circuit:*** *Brown-out protection circuit* prevents the processor/ controller from unexpected program execution behavior when the supply voltage to the processor/ controller falls below a specified voltage.

» It is essential for battery powered devices since there are greater chances for the battery voltage to drop below the required threshold. The processor behavior may not be predictable if the supply voltage falls below the recommended operating voltage. It may lead to situations like data corruption.

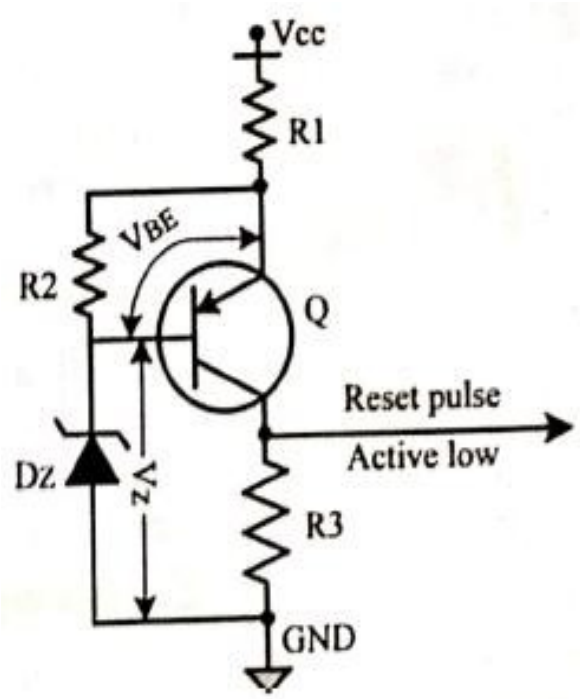
» A brown-out protection circuit holds the processor/ controller in reset state, when operating voltage falls below the threshold, until it rises above the threshold voltage.

» Certain processors/ controllers support built in brown-out protection circuit which monitors the supply voltage internally.

» If the processor/ controller don't integrate a built-in brown-out protection circuit, the same can be implemented using external passive circuits or supervisor ICs.

» The following Figure illustrates a brown-out circuit implementation using Zener diode and transistor for processor/ controller with active low Reset logic.

» The Zener diode, Dz, and transistor, Q, forms the heart of this circuit. The transistor conducts always when the supply voltage $V_{CC}$ is greater than that of the sum of $V_{BE}$ and $V_Z$ (Zener voltage). The transistor stops conducting when the supply voltage falls below the sum of $V_{BE}$ and $V_Z$. Select the Zener diode with required voltage for setting the low threshold value for $V_{CC}$.



» ***Oscillator Unit:*** A microprocessor/ microcontroller is a digital device made up of digital combinational and sequential circuits. The instruction execution of a microprocessor/ controller occurs in synchronization with a clock signal. The *oscillator unit* of the embedded system is responsible for generating the precise clock for the processor.

» Certain processors/ controllers integrate a built-in oscillator unit and simply require an external ceramic resonator/ quartz crystal for producing the necessary clock signals.

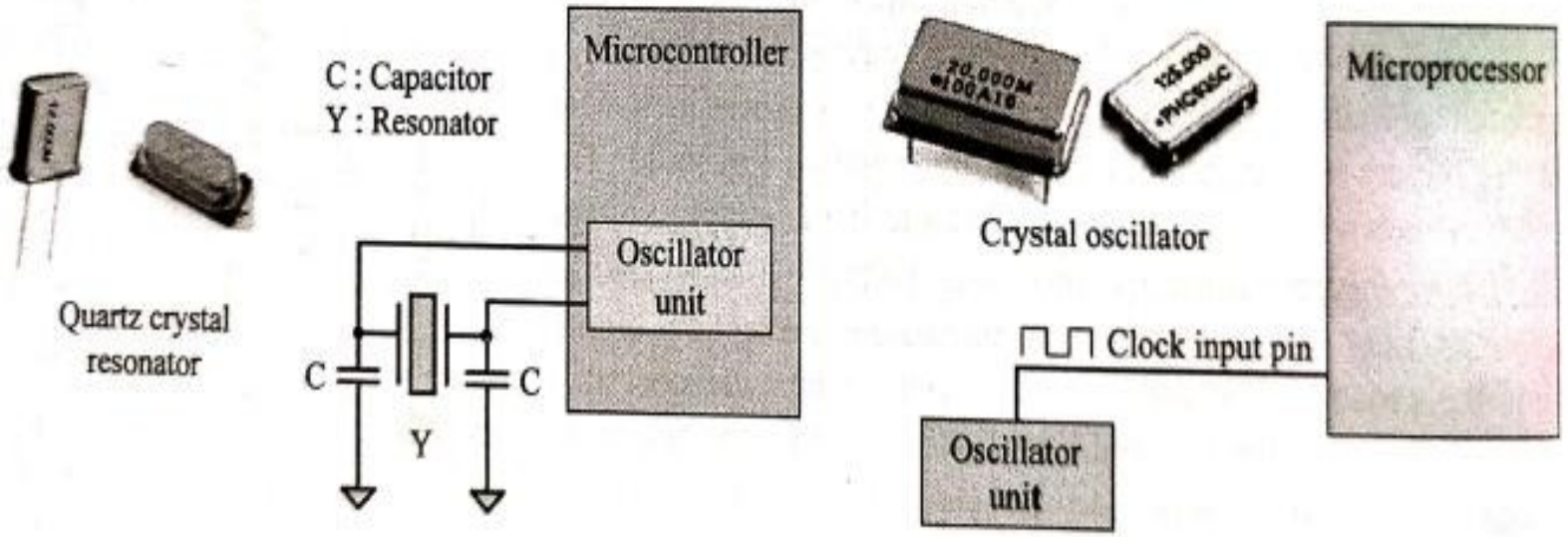
» Certain devices may not contain built-in oscillator unit and require the clock pulses to be generated and supplied externally.

» The speed of operation of a processor is primarily dependent on the clock frequency. However we cannot increase the clock frequency blindly for increasing the speed of execution. The logical circuits lying inside the processor always have an upper threshold value for the maximum clock at which the system can run, beyond which the system becomes unstable and non functional.

» The total system power consumption is directly proportional to the clock frequency. The power consumption increases with increase in clock frequency.

» The accuracy of program execution depends on the accuracy of the clock signal.

» The following Figure illustrates the usage of quartz crystal/ ceramic resonator and external oscillator chip for clock generation.



» ***Real-Time Clock (RTC):*** *Real-Time Clock* is a system component responsible for keeping track of time. RTC holds information like current time (In hours, minutes and seconds) in 12-hour/ 24-hour format, date, month, year, day of the week, etc. and supplies timing reference to the system.

» RTC is intended to function even in the absence of power.

» RTCs are available in the form of Integrated Circuits from different semiconductor manufacturers like Maxim/Dallas, ST Microelectronics etc.

» The RTC chip contains a microchip for holding the time and date related information and backup battery cell for functioning in the absence of power, in a single IC package. The RTC chip is interfaced to the processor or controller of the embedded system.

» For Operating System based embedded devices, a timing reference is essential for synchronizing the operations of the OS kernel.

» The RTC can interrupt the OS kernel

- One Interrupt Request (IRQ) number can be assigned to the RTC interrupt
- The kernel can perform necessary operations like system date time updating, managing software timers etc when an RTC timer tick interrupt occurs.

» The RTC can be configured to interrupt the processor at predefined intervals or to interrupt the processor when the RTC register reaches a specified value (used as alarm interrupt).

» ***Watchdog Timer:*** In desktop Windows systems, if we feel our application is behaving in an abnormally or if the system hangs up, we have the ***'Ctrl + Alt + Del'*** to come out of the situation. What it happens to embedded system?

» We have a watchdog to monitor the firmware execution and reset the system processor/ microcontroller when the program execution hangs up. A *watchdog timer*, or simply a *watchdog*, is a hardware timer for monitoring the firmware execution.

» Depending on the internal implementation, the watchdog timer increments or decrements a free running counter with each clock pulse and generates a reset signal to reset the processor if the count reaches zero for a down counting watchdog, or the highest count value for an up counting watchdog.

» If the watchdog counter is in the enabled state, the firmware can write a zero (for up counting watchdog implementation) to it before starting the execution of a piece of code and the watchdog will start counting.

» If the firmware execution doesn't complete due to malfunctioning, within the time required by the watchdog to reach the maximum count, the counter will generate a reset pulse and this will reset the processor.

» If the firmware execution completes before the expiration of the watchdog, you can reset the count by writing a 0 (for an up counting watchdog timer) to the watchdog timer register.

» If the processor/ controller doesn't contain a built in watchdog timer, the same can be implemented using an external watchdog timer IC circuit.

» The external watchdog timer uses hardware logic for enabling/ disabling, resetting the watchdog count, etc., instead of the firmware based 'writing' to the status and watchdog timer register.

» The microprocessor supervisor IC DS 1232 integrates a hardware watchdog timer in it.

» The following Figure illustrates the implementation of the external watchdog timer based microprocessor based supervisor circuit for a small embedded system.