

## **UNIT-I**

### **INTRODUCTION**

Embedded System-Definition, History, Classification, application areas and purpose of embedded systems, the typical embedded system-Core of the embedded system, Memory, Sensors and Actuators, Communication Interface, Embedded firmware, PCB and passive components. Characteristics, Quality attributes of an Embedded systems, Application-specific and Domain-Specific examples of an embedded system.

## **UNIT-II**

### **EMBEDDED HARDWARE DESIGN**

Analog and digital electronic components, I/O types and examples, Serial communication devices, Parallel device ports, Wireless devices, Timer and counting devices, Watchdog timer, Real time clock.

## **UNIT-III**

### **EMBEDDED FIRMWARE DESIGN**

Embedded Firmware design approaches, Embedded Firmware development languages, ISR concept, Interrupt sources, Interrupt servicing mechanism, Multiple interrupts, DMA, Device driver programming, Concepts of C versus Embedded C and Compiler versus Cross-compiler.

## **UNIT-IV**

### **REAL TIME OPERATING SYSTEM**

Operating system basics, Types of operating systems, Tasks, Process and Threads, Multiprocessing and Multitasking, Threads, Processes and Scheduling, Task Scheduling, Communication, Synchronization, Device Drivers, How to choose an RTOS.

Hardware Software Co-Design: Fundamental Issues in Hardware Software Co-Design, Computational models in embedded design, Hardware software Trade-offs, Integration of Hardware and Firmware, ICE.

## **UNIT-V**

### **EMBEDDED SYSTEM DEVELOPMENT**

The integrated development environment, Types of files generated on cross-compilation, Deassembler/Decompiler, Simulators, Emulators and Debugging, Target hardware debugging, Boundary Scan, Embedded Software development process and tools.

## **UNIT-VI**

### **EMBEDDED SYSTEM IMPLEMENTATION AND TESTING**

The main software utility tool, CAD and the hardware, Translation tools-Pre-processors, Interpreters, Compilers and Linkers, Debugging tools, Quality assurance and testing of the design, Testing on host machine, Simulators, Laboratory Tools.

# **UNIT-I**

## **INTRODUCTION**

### **EMBEDDED SYSTEM DEFINITION:**

An embedded system is an applied computer system, as distinguished from other types of computer systems such as personal computers (PCs) or supercomputers. However, you will find that the definition of “embedded system” is fluid and difficult to pin down, as it constantly evolves with advances in technology and dramatic decreases in the cost of implementing various hardware and software components. An embedded system is designed to perform a dedicated function. Most embedded devices are primarily designed for one specific function.

An embedded system is a computer system with higher quality and reliability requirements than other types of computer systems. Some families of embedded devices have a very high threshold of quality and reliability requirements. For example, if a car’s engine controller crashes while driving on a busy freeway or a critical medical device malfunctions during surgery, very serious

problems result. However, there are also embedded devices, such as TVs, games, and cell phones, in which a malfunction is an inconvenience but not usually a life-threatening situation.

Table: Examples of embedded systems and their markets

Market	Embedded Device
Automotive	Ignition System
	Engine Control
	Brake System (i.e., Antilock Braking System)
Consumer Electronics	Digital and Analog Televisions
	Set-Top Boxes (DVDs, VCRs, Cable Boxes, etc.)
	Personal Data Assistants (PDAs)
	Kitchen Appliances (Refrigerators, Toasters, Microwave Ovens)
	Automobiles
	Toys/Games
	Telephones/Cell Phones/Pagers
	Cameras
	Global Positioning Systems (GPS)
Market	Embedded Device
Industrial Control	Robotics and Control Systems (Manufacturing)
Medical	Infusion Pumps
	Dialysis Machines
	Prosthetic Devices
	Cardiac Monitors
Networking	Routers
	Hubs
	Gateways
Office Automation	Fax Machine
	Photocopier
	Printers
	Monitors
	Scanners

## HISTORY:

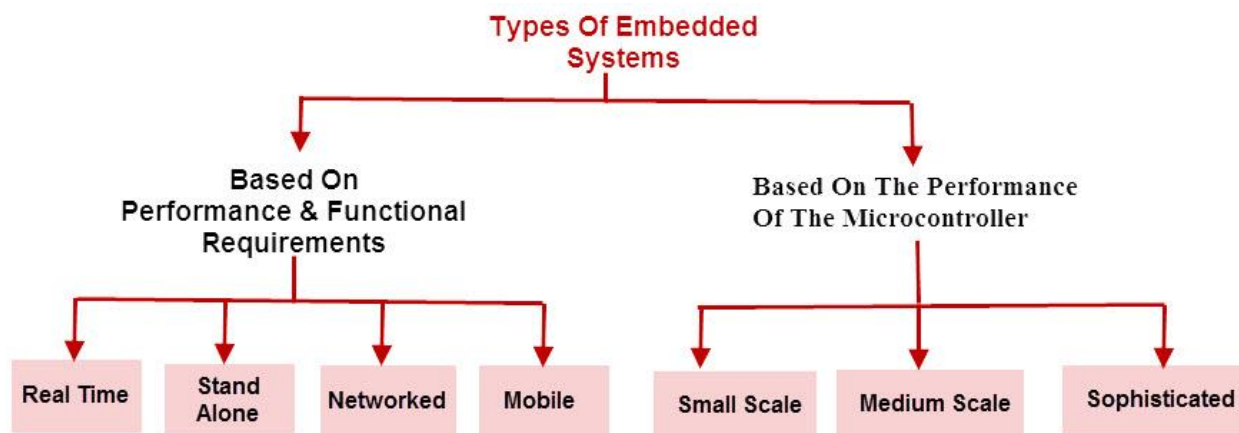
One of the very first recognizably modern embedded systems was the Apollo Guidance Computer, developed ca. 1965 by Charles Stark Draper at the MIT Instrumentation Laboratory. At the project's inception, the Apollo guidance computer was considered the riskiest item in the Apollo project as it employed the then newly developed monolithic integrated circuits to reduce the size and weight. An early mass-produced embedded system was the Autonetics D-17 guidance computer for the Minuteman missile, released in 1961. When the Minuteman II went into production in 1966, the D-17 was replaced with a new computer that was the first high-volume use of integrated circuits.

Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. An early microprocessor for example, the Intel 4004 (released in 1971), was designed for calculators and other small systems but still required external memory and support chips. In 1978 National Engineering Manufacturers Association released a "standard" for programmable microcontrollers, including almost any computer-based controllers, such as single board computers, numerical, and event-based controllers.

As the cost of microprocessors and microcontrollers fell it became feasible to replace expensive knob-based analog components such as potentiometers and variable capacitors with up/down buttons or knobs read out by a microprocessor even in consumer products. By the early 1980s, memory, input and output system components had been integrated into the same chip as the processor forming a microcontroller. Microcontrollers find applications where a general-purpose computer would be too costly.

A comparatively low-cost microcontroller may be programmed to fulfill the same role as a large number of separate components. Although in this context an embedded system is usually more complex than a traditional solution, most of the complexity is contained within the microcontroller itself. Very few additional components may be needed and most of the design effort is in the software. Software prototype and test can be quicker compared with the design and construction of a new circuit not using an embedded processor

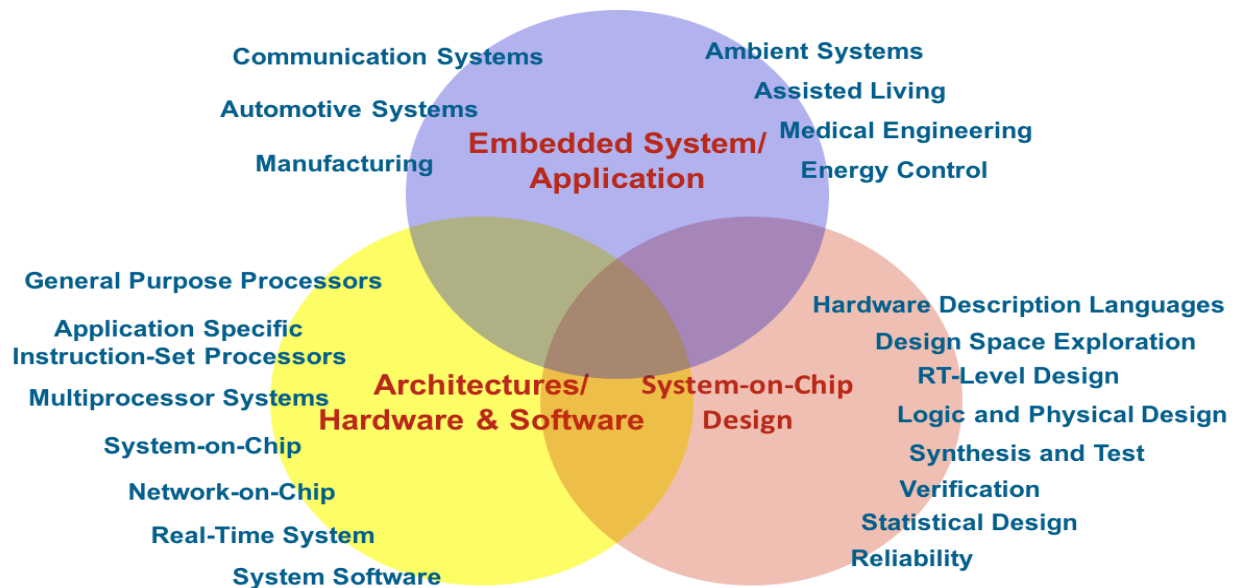
## **CLASSIFICATION OF EMBEDDED SYSTEMS:**



## **APPLICATION AREAS OF EMBEDDED SYSTEMS:**

Embedded systems find numerous applications in various fields such as digital electronics, telecommunications, computing network, smart cards, satellite systems, military defense system equipment, research system equipment, and so on. Let us discuss a few practical applications of

embedded systems that are used in designing embedded projects as a part of engineering final year electronics projects.



### CORE OF THE EMBEDDED SYSTEMS:

Embedded systems are domain and application specific and are built around a central core. The core of the embedded system falls into any of the following categories:

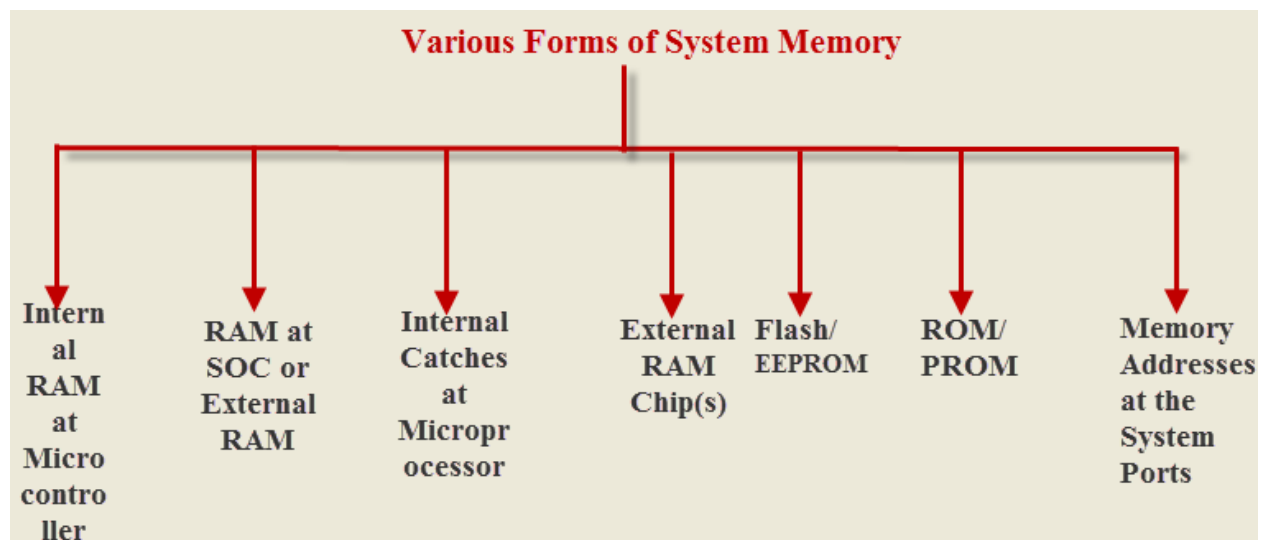
General purpose and Domain Specific Processors

- Microprocessors
- Microcontrollers
- Digital Signal Processors
- Application Specific Integrated Circuits. (ASIC)
- Programmable logic devices(PLD's)
- Commercial off-the-shelf components (COTs)

### MEMORY:

Different Types of Memory Modules used in Embedded System

- Different types of memory.
- Volatile Memory Module-RAM.
- Internal Data storage circuit for RAM memory chip.
- Non volatile memory-ROM Memory.
- Static random Access memory (SRAM)
- Dynamic Access Random Memory (DRAM)
- Programmable Read Only Memory.
- Erasable Programmable Read Only Memory



## SENSORS AND ACTUATORS:

### SENSORS:

A transducer is a device that converts energy from one form to another. Usually a transducer converts a signal in one form of energy to a signal in another. Transducers are often employed at the boundaries of automation, measurement, and control systems, where electrical signals are converted to and from other physical quantities (energy, force, torque, light, motion, position, etc.). The process of converting one form of energy to another is known as transduction.

### Mechanical and electrical transducers

Transducers that convert physical quantities into mechanical ones are called mechanical transducers; Transducers that convert physical quantities into electrical are called electrical transducers. Examples are a thermocouple that changes temperature differences into a small voltage, or a Linear variable differential transformer (LVDT) used to measure displacement.

### Sensors and actuators

Transducers can be categorized by which direction information passes through them:

- A *sensor* is a transducer that receives and responds to a signal or stimulus from a physical system.<sup>[3][4][2]</sup> It produces a signal, which represents information about the system, which is used by some type of telemetry, information or control system.
- An *actuator* is a device that is responsible for moving or controlling a mechanism or system. It is controlled by a signal from a control system or manual control. It is operated by a source of energy, which can be mechanical force, electrical current, hydraulic fluid pressure, or pneumatic pressure, and converts that energy into motion. An actuator is the mechanism by which a control system acts upon an environment. The control system can be simple (a fixed mechanical or electronic system), software-based (e.g. a printer driver, robot control system), a human, or any other input.<sup>[2]</sup>
- Bidirectional transducers convert physical phenomena to electrical signals and also convert electrical signals into physical phenomena. An example of an inherently bidirectional transducer is an antenna, which can convert radio waves (electromagnetic

waves) into an electrical signal to be processed by a radio receiver, or translate an electrical signal from a transmitter into radio waves. Another example is voice coils, which are used in loudspeakers to translate an electrical audio signal into sound and in dynamic microphones to translate sound waves into an audio signal.<sup>[2]</sup>

### **Active vs Passive sensors**

- *Active* sensors require an external power source to operate, which is called an excitation signal. The signal is modulated by the sensor to produce an output signal. For example, a thermistor does not generate any electrical signal, but by passing an electric current through it, its resistance can be measured by detecting variations in the current or voltage across the thermistor.<sup>[5][2]</sup>
- *Passive* sensors, in contrast, generate an electric current in response to an external stimulus which serves as the output signal without the need of an additional energy source. Such examples are a photodiode, and a piezoelectric sensor, thermocouple.<sup>[6]</sup>

### **ACTUATOR:**

An actuator is a component of a machine that is responsible for moving and controlling a mechanism or system, for example by opening a valve. In simple terms, it is a "mover". An actuator requires a control signal and a source of energy. The control signal is relatively low energy and may be electric voltage or current, pneumatic or hydraulic pressure, or even human power. Its main energy source may be an electric current, hydraulic fluid pressure, or pneumatic pressure. When it receives a control signal, an actuator responds by converting the signal's energy into mechanical motion. An actuator is the mechanism by which a control system acts upon an environment. The control system can be simple (a fixed mechanical or electronic system), software-based (e.g. a printer driver, robot control system), a human, or any other input.

### **Hydraulic**

A hydraulic actuator consists of cylinder or fluid motor that uses hydraulic power to facilitate mechanical operation. The mechanical motion gives an output in terms of linear, rotatory or oscillatory motion. As liquids are nearly impossible to compress, a hydraulic actuator can exert a large force. The drawback of this approach is its limited acceleration.

The hydraulic cylinder consists of a hollow cylindrical tube along which a piston can slide. The term *single acting* is used when the fluid pressure is applied to just one side of the piston. The piston can move in only one direction, a spring being frequently used to give the piston a return stroke. The term *double acting* is used when pressure is applied on each side of the piston; any difference in pressure between the two sides of the piston moves the piston to one side or the other.<sup>[3]</sup>

### **Pneumatic rack and pinion actuators for valve controls of water pipes**

#### **Pneumatic**

Pneumatic actuators enable considerable forces to be produced from relatively small pressure changes. A pneumatic actuator converts energy formed by vacuum or compressed air at high pressure into either linear or rotary motion. Pneumatic energy is desirable for main engine controls because it can quickly respond in starting and stopping as the power source does not need to be stored in reserve for operation. Moreover, pneumatic actuators are safer, cheaper, and

often more reliable and powerful than other actuators. These forces are often used with valves to move diaphragms to affect the flow of air through the valve.<sup>[4][5]</sup>

## **Electric valve actuator controlling a ½ needle valve.**

### **Electric**

An electric actuator is powered by a motor that converts electrical energy into mechanical torque. The electrical energy is used to actuate equipment such as multi-turn valves. Additionally, a brake is typically installed above the motor to prevent the media from opening valve. If no brake is installed, the actuator will uncover the opened valve and rotate it back to its closed position. If this continues to happen, the motor and actuator will eventually become damaged.<sup>[6]</sup> It is one of the cleanest and most readily available forms of actuator because it does not directly involve oil or other fossil fuels.<sup>[7]</sup>

### **Twisted and coiled polymer (TCP) or supercoiled polymer (SCP)**

Twisted and coiled polymer (TCP) actuator also known as supercoiled polymer (SCP) actuator is a coiled polymer that can be actuated by electric power <sup>[8]</sup>. A TCP actuator look like a helical spring. TCP actuators are usually made from silver coated Nylon. TCP actuators can also be made from other electrical conductance coat such as gold. TCP actuator should be under a load to keep the muscle extended. The electrical energy transforms to thermal energy due to electrical resistance, which is also known as Joule heating, Ohmic heating, and resistive heating. As the temperature of the TCP actuator increases by Joule heating, the polymer contracts and it causes the actuator contraction <sup>[8]</sup>.

### **Thermal or magnetic**

Actuators which can be actuated by applying thermal or magnetic energy have been used in commercial applications. Thermal actuators tend to be compact, lightweight, economical and with high power density. These actuators use shape memory materials (SMMs), such as shape-memory alloys (SMAs) or magnetic shape-memory alloys (MSMAs). Some popular manufacturers of these devices are Finnish Modti Inc., American Dynalloy and Rotork.

### **Mechanical**

A mechanical actuator functions to execute movement by converting one kind of motion, such as rotary motion, into another kind, such as linear motion. An example is a rack and pinion. The operation of mechanical actuators is based on combinations of structural components, such as gears and rails, or pulleys and chains.

## **QUALITY ATTRIBUTES OF AN EMBEDDED SYSTEM:**

These are the attributes that together form the deciding factor about the quality of an embedded system. There are two types of quality attributes are:-

### **Operational Quality Attributes.**

These are attributes related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.



### **Non-Operational Quality Attributes.**

These are attributes **not** related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality. These are the attributes that are associated with the embedded system before it can be put in operation.

### **Operational Attributes**

#### **Response**

Response is a measure of quickness of the system.

It gives you an idea about how fast your system is tracking the input variables.

Most of the embedded system demand fast response which should be real-time.

#### **Throughput**

Throughput deals with the efficiency of system.

It can be defined as rate of production or process of a defined process over a stated period of time. In case of card reader like the ones used in buses, throughput means how much transaction the reader can perform in a minute or hour or day.

#### **Reliability**

Reliability is a measure of how much percentage you rely upon the proper functioning of the system. Mean Time between failures and Mean Time To Repair are terms used in defining system reliability. Mean Time between failures can be defined as the average time the system is functioning before a failure occurs. Mean time to repair can be defined as the average time the system has spent in repairs.

#### **Maintainability**

Maintainability deals with support and maintenance to the end user or a client in case of technical issues and product failures or on the basis of a routine system checkup

It can be classified into two types :-

#### **Scheduled or Periodic Maintenance**

o This is the maintenance that is required regularly after a periodic time interval.

Example :

Periodic Cleaning of Air Conditioners

Refilling of printer cartridges.

### **2. Maintenance to unexpected failure**

This involves the maintenance due to a sudden breakdown in the functioning of the system.

Example:

Air conditioner not powering on

Printer not taking paper in spite of a full paper stack

### **APPLICATION SPECIFIC EMBEDDED SYSTEMS:**

#### **Embedded Systems in Automobiles and in telecommunications**

- Motor and cruise control system

- Body or Engine safety
- Entertainment and multimedia in car
- E-Com and Mobile access
- Robotics in assembly line
- Wireless communication
- Mobile computing and networking

### **Embedded Systems in Smart Cards, Missiles and Satellites**

- Security systems
- Telephone and banking
- Defense and aerospace
- Communication

### **Embedded Systems in Peripherals & Computer Networking**

- Displays and Monitors
- Networking Systems
- Image Processing
- Network cards and printers

### **Embedded Systems in Consumer Electronics**

- Digital Cameras
- Set top Boxes
- High Definition TVs
- DVDs

# **UNIT-II**

## **EMBEDDED HARDWARE DESIGN**

### **ANALOG AND DIGITAL ELECTRONIC COMPONENTS:**

An electronic component is any basic discrete device or physical entity in an electronic system used to affect electrons or their associated fields. Electronic components are mostly industrial products, available in a singular form and are not to be confused with electrical elements, which are conceptual abstractions representing idealized electronic components.

Electronic components have a number of electrical terminals or leads. These leads connect to create an electronic circuit with a particular function (for example an amplifier, radio receiver, or oscillator). Basic electronic components may be packaged discretely, as arrays or networks of like components, or integrated inside of packages such as semiconductor integrated circuits, hybrid integrated circuits, or thick film devices.

### **Semiconductors**

## Diodes

**Conduct electricity easily in one direction, among more specific behaviors.**

- Diode, Rectifier, Bridge rectifier
- Schottky diode, hot carrier diode – super fast diode with lower forward voltage drop
- Zener diode – Passes current in reverse direction to provide a constant voltage reference
- Transient voltage suppression diode (TVS), Unipolar or Bipolar – used to absorb high-voltage spikes
- Varactor, Tuning diode, Varicap, Variable capacitance diode – A diode whose AC capacitance varies according to the DC voltage applied.



### Various examples of Light-emitting diodes

- Light-emitting diode (LED) – A diode that emits light
- Photodiode – Passes current in proportion to incident light
  - Avalanche photodiode Photodiode with internal gain
  - Solar Cell, photovoltaic cell, PV array or panel, produces power from light
- DIAC (Diode for Alternating Current), Trigger Diode, SIDAC) – Often used to trigger an SCR
- Constant-current diode
- Peltier cooler – A semiconductor heat pump
- Tunnel diode - very fast diode based on quantum mechanical tunneling

## Transistors

Transistors were considered the invention of the twentieth century that changed electronic circuits forever. A transistor is a semiconductor device used to amplify and switch electronic signals and electrical power.

- **Transistors**
  - Bipolar junction transistor (BJT, or simply "transistor") – NPN or PNP
    - Photo transistor – Amplified photodetector
  - Darlington transistor – NPN or PNP
    - Photo Darlington – Amplified photodetector
  - Sziklai pair (Compound transistor, complementary Darlington)
- **Field-effect transistor (FET)**
  - JFET (Junction Field-Effect Transistor) – N-CHANNEL or P-CHANNEL
  - MOSFET (Metal Oxide Semiconductor FET) – N-CHANNEL or P-CHANNEL
  - MESFET (Metal Semiconductor FET)
  - HEMT (High electron mobility transistor)
- **Thyristors**
  - Silicon-controlled rectifier (SCR) – Passes current only after triggered by a sufficient control voltage on its gate
  - TRIAC (TRIode for Alternating Current) – Bidirectional SCR
  - Unijunction transistor (UJT)
  - Programmable Unijunction transistor (PUT)

- SIT (Static induction transistor)
  - STh (Static induction thyristor)
- **Composite transistors**
  - IGBT (Insulated-gate bipolar transistor)

## Integrated circuits

- Digital electronics
- Analog
  - Hall effect sensor –senses a magnetic field.
  - Current sensor – Senses a current through it

## Optoelectronic devices

- **Opto-electronics**
  - Opto-Isolator, Opto-Coupler, Photo-Coupler – Photodiode, BJT, JFET, SCR, TRIAC, Zero-crossing TRIAC, Open collector IC, CMOS IC, Solid state relay (SSR)
  - Opto switch, Opto interrupter, Optical switch, Optical interrupter, Photo switch, Photo interrupter
  - LED display – Seven-segment display, Sixteen-segment display, Dot-matrix display

## Display technologies

### Current:

- Filament lamp (indicator lamp)
- Vacuum fluorescent display (VFD) (preformed characters, 7 segment, starburst)
- Cathode ray tube (CRT) (dot matrix scan, radial scan (e.g. radar), arbitrary scan (e.g. oscilloscope)) (monochrome & colour)
- LCD (preformed characters, dot matrix) (passive, TFT) (monochrome, colour)
- Neon (individual, 7 segment display)
- LED (individual, 7 segment display, starburst display, dot matrix)
- Flap indicator (numeric, preprinted messages)
- Plasma display (dot matrix)

### Obsolete:

- Incandescent filament 7 segment display (aka 'Numitron')
- Nixie Tube
- Dekatron (aka glow transfer tube)
- Magic eye tube indicator
- Penetron (a 2 colour see-through CRT)

## Vacuum tubes (valves)

A vacuum tube is based on current conduction through a vacuum (see Vacuum tube).

- Diode or rectifier tube
- Amplification

- Triode
- Tetrode
- Pentode
- Hexode
- Pentagrid
- Octode
- Traveling-wave tube
- Klystron
- Oscillation
  - Magnetron
  - Reflex Klystron (obsolete)
  - Carcinotron

#### Optical detectors or emitters

- Phototube or Photodiode – tube equivalent of semiconductor photodiode
- Photomultiplier tube – phototube with internal gain
- Cathode ray tube (CRT) or television picture tube (obsolete)
- Vacuum fluorescent display (VFD) – modern non-raster sort of small CRT display
- Magic eye tube – small CRT display used as a tuning meter (obsolete)
- X-ray tube – generates x-rays

#### Discharge devices

- Gas discharge tube
- Ignitron
- Thyatron

#### Obsolete:

- Mercury arc rectifier
- Voltage regulator tube
- Nixie tube

### **Power sources**

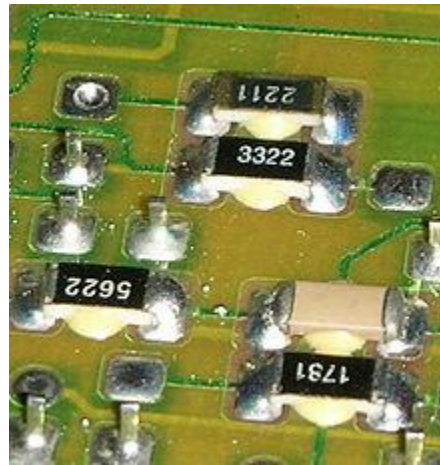
#### Sources of electrical power:

- Battery – acid- or alkali-based power supply.
- Fuel cell – an electrochemical generator
- Power supply – usually a main hook-up
- Photo voltaic device – generates electricity from light
- Thermo electric generator – generates electricity from temperature gradients
- Electrical generator – an electromechanical power source
- Piezoelectric generator - generates electricity from mechanical strain
- Van de Graaff generator - generates electricity from friction

### **Passive components**

Components incapable of controlling current by means of another electrical signal are called passive devices. Resistors, capacitors, inductors, and transformers are all considered passive devices.

## Resistors



### SMD resistors on a backside of a PCB

Pass current in proportion to voltage (Ohm's law) and oppose current.

- Resistor – fixed value
  - Power resistor – larger to safely dissipate heat generated
  - SIP or DIP resistor network – array of resistors in one package
- Variable resistor
  - Rheostat – two-terminal variable resistor (often for high power)
  - Potentiometer – three-terminal variable resistor (variable voltage divider)
  - Trim pot – Small potentiometer, usually for internal adjustments
  - Thermistor – thermally sensitive resistor whose prime function is to exhibit a large, predictable and precise change in electrical resistance when subjected to a corresponding change in body temperature.<sup>[3]</sup>
  - Humistor – humidity-varied resistor
  - Photoresistor
  - Memristor
  - Varistor, Voltage Dependent Resistor, MOV – Passes current when excessive voltage is present
- Resistance wire, Nichrome wire – wire of high-resistance material, often used as a heating element
- Heater – heating element

## Capacitors



### Some different capacitors for electronic equipment

Capacitors store and release electrical charge. They are used for filtering power supply lines, tuning resonant circuits, and for blocking DC voltages while passing AC signals, among numerous other uses.

- Capacitor
  - Integrated capacitors
    - MIS capacitor
    - Trench capacitor
  - Fixed capacitors
    - Ceramic capacitor
    - Film capacitor
    - Electrolytic capacitor
      - Aluminum electrolytic capacitor
      - Tantalum electrolytic capacitor
      - Niobium electrolytic capacitor
      - Polymer capacitor, OS-CON
    - Supercapacitor (Electric double-layer capacitor)
      - Nanoionic supercapacitor
      - Lithium-ion capacitor
    - Mica capacitor
    - Vacuum capacitor
  - Variable capacitor – adjustable capacitance
    - Tuning capacitor – variable capacitor for tuning a radio, oscillator, or tuned circuit
    - Trim capacitor – small variable capacitor for seldom or rare adjustments of LC-circuits
    - Vacuum variable capacitor
  - Capacitors for special applications



- Power capacitor
- Safety capacitor
- Filter capacitor
- Light-emitting capacitor
- Motor capacitor
- Photoflash capacitor
- Reservoir capacitor
- Capacitor network (array)
- Varicap diode – AC capacitance varies according to the DC voltage applied

## **Magnetic (inductive) devices**

**Electrical components that use magnetism in the storage and release of electrical charge through current:**

- Inductor, coil, choke
- Variable inductor
- Saturable Inductor
- Transformer
- Magnetic amplifier (toroid)
- ferrite impedances, beads
- Motor / Generator
- Solenoid
- Loudspeaker and microphone

## **Memristor**

Electrical components that pass charge in proportion to magnetism or magnetic flux, and have the ability to retain a previous resistive state, hence the name of Memory plus Resistor.

- Memristor

## **Networks**

Components that use more than one type of passive component:

- RC network – forms an RC circuit, used in snubbers
- LC Network – forms an LC circuit, used in tunable transformers and RFI filters.

## **Transducers, sensors, detectors**

1. Transducers generate physical effects when driven by an electrical signal, or vice versa.
  2. Sensors (detectors) are transducers that react to environmental conditions by changing their electrical properties or generating an electrical signal.
  3. The transducers listed here are single electronic components (as opposed to complete assemblies), and are passive (see Semiconductors and Tubes for active ones). Only the most common ones are listed here.
- Audio
    - Loudspeaker – Electromagnetic or piezoelectric device to generate full audio
    - Buzzer – Electromagnetic or piezoelectric sounder to generate tones
  - Position, motion

- Linear variable differential transformer (LVDT) – Magnetic – detects linear position
- Rotary encoder, Shaft Encoder – Optical, magnetic, resistive or switches – detects absolute or relative angle or rotational speed
- Inclinator – Capacitive – detects angle with respect to gravity
- Motion sensor, Vibration sensor
- Flow meter – detects flow in liquid or gas
- Force, torque
  - Strain gauge – Piezoelectric or resistive – detects squeezing, stretching, twisting
  - Accelerometer – Piezoelectric – detects acceleration, gravity
- Thermal
  - Thermocouple, thermopile – Wires that generate a voltage proportional to delta temperature
  - Thermistor – Resistor whose resistance changes with temperature, up PTC or down NTC
  - Resistance Temperature Detector (RTD) – Wire whose resistance changes with temperature
  - Bolometer – Device for measuring the power of incident electromagnetic radiation
  - Thermal cutoff – Switch that is opened or closed when a set temperature is exceeded
- Magnetic field (see also Hall Effect in semiconductors)
  - Magnetometer, Gauss meter
- Humidity
  - Hygrometer
- Electromagnetic, light
  - Photo resistor – Light dependent resistor (LDR)

## **Antennas**

### **Antennas transmit or receive radio waves**

- Elemental dipole
- Yagi
- Phased array
- Loop antenna
- Parabolic dish
- Log-periodic dipole array
- Biconical
- Feedhorn

## **Assemblies, modules**

### **Multiple electronic components assembled in a device that is in itself used as a component**

- Oscillator
- Display devices
  - Liquid crystal display (LCD)
  - Digital voltmeters
- Filter

## **Prototyping aids**

- Wire-wrap
- Breadboard

## **Electromechanical**



**A quartz crystal (left) and a crystal oscillator**

## **Piezoelectric devices, crystals, resonators**

### **Passive components that use piezoelectric effect:**

- Components that use the effect to generate or filter high frequencies
  - Crystal – a ceramic crystal used to generate precise frequencies (See the Modules class below for complete oscillators)
  - Ceramic resonator – Is a ceramic crystal used to generate semi-precise frequencies
  - Ceramic filter – Is a ceramic crystal used to filter a band of frequencies such as in radio receivers
  - surface acoustic wave (SAW) filters
- Components that use the effect as mechanical transducers.
  - Ultrasonic motor – Electric motor that uses the piezoelectric effects
  - For piezo buzzers and microphones, see the Transducer class below

## **Terminals and connectors**

### **Devices to make electrical connection**

- Terminal
- Connector
  - Socket
  - Screw terminal, Terminal Blocks
  - Pin header

## **Cable assemblies**

### **Electrical cables with connectors or terminals at their ends**

- Power cord
- Patch cord
- Test lead



**2 different miniature pushbutton switches**

## Switches

**Components that can pass current ("closed") or break the current ("open"):**

- Switch – Manually operated switch
  - Electrical description: SPST, SPDT, DPST, DPDT, NPNT (general)
  - Technology: slide switches, toggle switches, rocker switches, rotary switches, pushbutton switches
- Keypad – Array of pushbutton switches
- DIP switch – Small array of switches for internal configuration settings
- Footswitch – Foot-operated switch
- Knife switch – Switch with unenclosed conductors
- Micro switch – Mechanically activated switch with snap action
- Limit switch – Mechanically activated switch to sense limit of motion
- Mercury switch – Switch sensing tilt
- Centrifugal switch – Switch sensing centrifugal force due to rate of rotation
- Relay or contactor – Electro-mechanically operated switch (see also solid state relay above)
- Reed switch – Magnetically activated switch
- Thermostat – Thermally activated switch
- Humidistat – Humidity activated switch
- Circuit breaker – Switch opened in response to excessive current: a resettable fuse

## Protection devices

**Passive components that protect circuits from excessive currents or voltages:**

- Fuse – over-current protection, one time use
- Circuit breaker – resettable fuse in the form of a mechanical switch
- Resettable fuse or PolySwitch – circuit breaker action using solid state device
- Ground-fault protection or residual-current device – circuit breaker sensitive to mains currents passing to ground
- Metal oxide varistor (MOV), surge absorber, TVS – Over-voltage protection
- Inrush current limiter – protection against initial Inrush current
- Gas discharge tube – protection against high voltage surges
- Spark gap – electrodes with a gap to arc over at a high voltage
- Lightning arrester – spark gap used to protect against lightning strikes

## Mechanical accessories

- Enclosure (electrical)

- Heat sink
- Fan

## Other

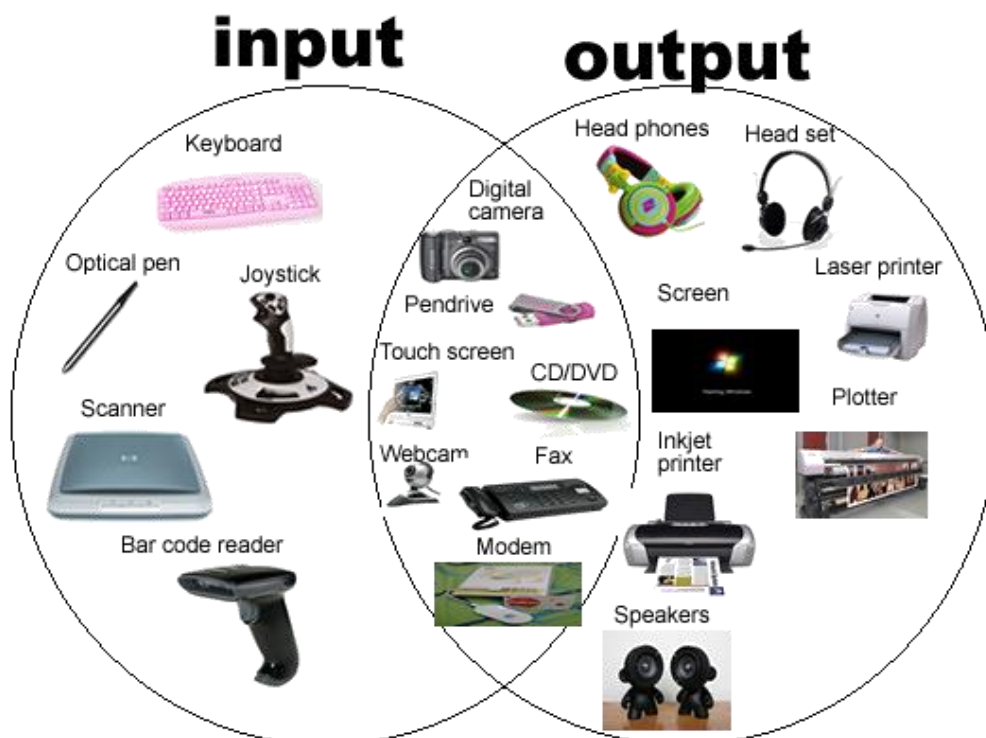
- Printed circuit boards
- Lamp
- Waveguide
- Memristor

## Obsolete

- Carbon amplifier (see Carbon microphones used as amplifiers)
- Carbon arc (negative resistance device)
- Dynamo (historic rf generator)
- Coherer

## I/O TYPES AND EXAMPLES:

In computing, input/output or I/O (or, informally, io or IO) is the communication between an information processing system, such as a computer, and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation. I/O devices are the pieces of hardware used by a human (or other system) to communicate with a computer. For instance, a keyboard or computer mouse is an input device for a computer, while monitors and printers are output devices. Devices for communication between computers, such as modems and network cards, typically perform both input and output operations.



## SERIAL COMMUNICATION:

### Serial Data Communication

**Data Communication** is one of the most challenging fields today as far as technology development is concerned. Data, essentially meaning information coded in digital form, that is, 0s and 1s, is needed to be sent from one point to the other either directly or through a network.

And when many such systems need to share the same information or different information through the same medium, there arises a need for proper organization (rather, “socialization”) of the whole network of the systems, so that the whole system works in a cohesive fashion. Therefore, in order for a proper interaction between the data transmitter (the device needing to commence data communication) and the data receiver (the system which has to receive the data sent by a transmitter) there has to be some set of rules or (“protocols”) which all the interested parties must obey. The requirement above finally paves the way for some **DATA COMMUNICATION STANDARDS**.

Depending on the requirement of applications, one has to choose the type of communication strategy. There are basically two major classifications, namely SERIAL and PARALLEL, each with its variants. The discussion about serial communication will be undertaken in this lesson.

Any data communication standard comprises

- The protocol.
- Signal/data/port specifications for the devices or additional electronic circuitry involved.

What is Serial Communication?

Serial data communication strategies and, standards are used in situations having a limitation of the number of lines that can be spared for communication. This is the primary mode of transfer in long-distance communication. But it is also the situation in embedded systems where various subsystems share the communication channel and the speed is not a very critical issue.

**Standards** incorporate both the software and hardware aspects of the system while **buses** mainly define the cable characteristics for the same communication type. **Serial data communication** is the most common **low-level** protocol for communicating between two or more devices. Normally, one device is a computer, while the other device can be a modem, a printer, another computer, or a scientific instrument such as an oscilloscope or a function generator.

As the name suggests, the serial port sends and receives bytes of information, rather characters (*used in the other modes of communication*), in a serial fashion - one bit at a time. These bytes are transmitted using either a **binary (numerical) format** or a **text format**.

All the data communication systems follow some specific set of standards defined for their communication capabilities so that the systems are not Vendor specific but for each system the user has the advantage of selecting the device and interface according to his own choice of make and range. The most common serial communication system protocols can be studied under the following categories: *Asynchronous, Synchronous and Bit-Synchronous communication standards.*

## Asynchronous Communication and Standards

### The Protocol

- This protocol allows bits of information to be transmitted between two devices at an arbitrary point of time.
- The protocol defines that the data, more appropriately a “character” is sent as “frames” which in turn is a collection of bits.
- The start of a frame is identified according to a **START** bit(s) and a **STOP** bit(s) identifies the end of data frame. Thus, the **START** and the **STOP** bits are part of the frame being sent or received.
- The protocol assumes that both the transmitter and the receiver are configured in the same way, i.e., follow the same definitions for the start, stop and the actual data bits.
- Both devices, namely, the transmitter and the receiver, need to communicate at an agreed upon data rate (baud rate) such as **19,200 KB/s or 115,200 KB/s.**
- This protocol has been in use for 15 years and is used to connect PC peripherals such as **modems** and the applications include the classic **Internet dial-up** modem systems.
- Asynchronous systems allow a number of variations including the number of **bits in a character** (5, 6, 7 or 8 bits), the number of **stops bits** used (1, 1.5 or 2) and an optional parity bit. Today the most common standard has 8 bit characters, with 1 stop bit and no parity and this is frequently abbreviated as '**8-1-n**'. A single **8-bit** character, therefore, consists of **10 bits on the line**, i.e., One **Start** bit, Eight **Data** bits and One **Stop** bit (as shown in the figure below).
- Most important observation here is that the individual characters are **framed** (unlike all the other standards of serial communication) and **NO CLOCK** data is communicated between the two ends.

### The Typical Data Format (*known as “FRAME”*) for Asynchronous Communication



Interface Specifications for Asynchronous Serial Data Communication standard published by the Telecommunications Industry Association; both the physical and electrical characteristics of the interfaces have been detailed in these publications.

[RS-232](#), [RS-422](#), [RS-423](#) and [RS-485](#) are each a recommended standard (RS-XXX) of the Electronic Industry Association (EIA) for asynchronous serial communication and have more recently been rebranded as *EIA-232*, *EIA-422*, *EIA-423* and *EIA-485*.

It must be mentioned here that, although, some of the more advanced standards for serial communication like the [USB](#) and [FIREWIRE](#) are being popularized these days to fill the gap for high-speed, relatively short-run, heavy-data-handling applications, but still, the above four satisfy the needs of all those high-speed and longer run applications found most often in industrial settings for plant-wide security and equipment networking.

RS-232, 423, 422 and 485 specify the *communication system characteristics* of the hardware such as *voltage levels, terminating resistances, cable lengths, etc.* *The standards, however, say nothing about the software protocol or how data is framed, addressed, checked for errors or interpreted*

## THE RS-232

This is the original serial port interface “standard” and it stands for “***Recommended Standard Number 232***” or more appropriately *EIA Recommended Standard 232* is the oldest and the most popular serial communication standard. It was first introduced in 1962 to help ensure connectivity and compatibility across manufacturers for simple serial data communications.

### Applications

- Peripheral connectivity for PCs (*the PC COM port hardware*), which can range beyond modems and printers to many different handheld devices and modern scientific instruments.

All the various characteristics and definitions pertaining to this standard can be summarized according to:

- The maximum bit transfer rate capability and cable length.
- Communication Technique: names, electrical characteristics and functions of signals.
- The mechanical connections and pin assignments.



## The Standard

### Maximum Bit Transfer Rate, Signal Voltages and Cable Length

- RS-232's capabilities range from the original slow data rate of up to 20 kbps to over 1 Mbps for some of the modern applications.
- RS-232 is mainly intended for short cable runs, or local data transfers in a range up to **50 feet** maximum, but it must be mentioned here that it also depends on the **Baud Rate**.
- It is a robust interface with speeds to 115,200 baud, and
- It can **withstand a short circuit** between any 2 pins.
- It can handle signal voltages as high / low as  $\pm 15$  volts.

## Signal States and the Communication Technique

Signals can be in either an **active** state or an **inactive** state. RS232 is an Active LOW voltage driven interface where:

**ACTIVE STATE:** An active state corresponds to the binary value 1. An active signal state can also be indicated as **logic "1", "on", "true", or a "mark"**.

**INACTIVE STATE:** An inactive signal state is stated as **logic "0", "off", "false", or a "space"**.

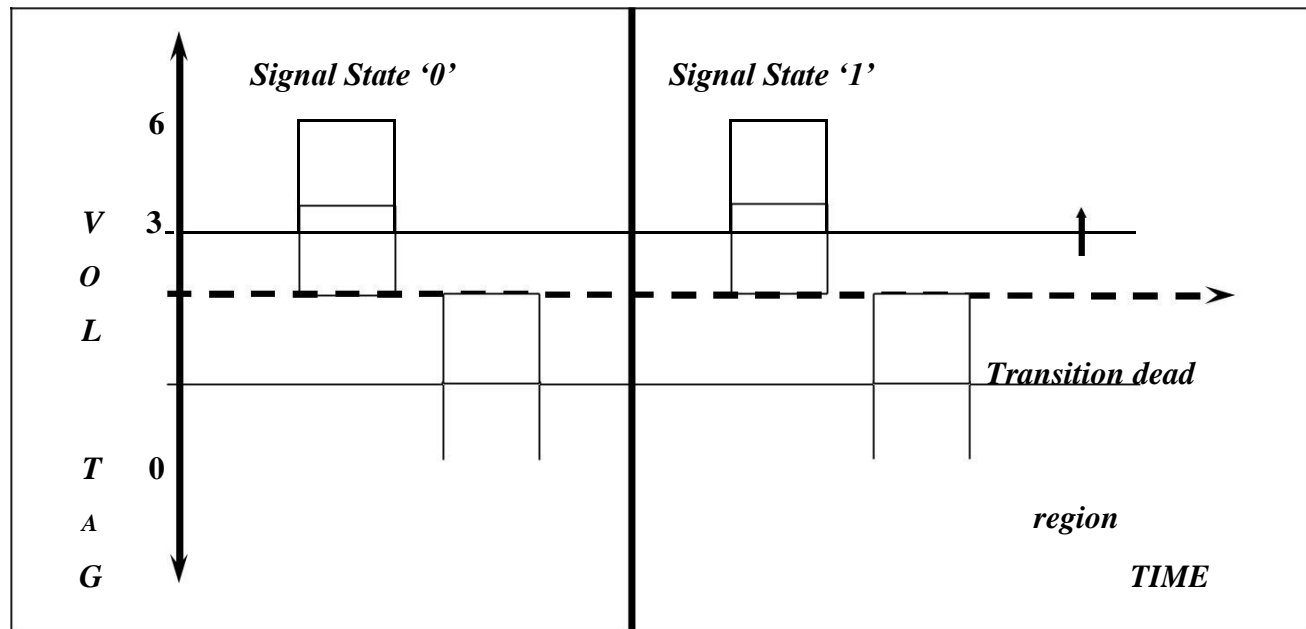
- For data signals, the **"true"** state occurs when the received signal voltage is more negative than **-3 volts**, while the **"false"** state occurs for voltages more positive than **3 volts**.
- For control signals, the **"true"** state occurs when the received signal voltage is more positive than **3 volts**, while the **"false"** state occurs for voltages more negative than **-3 volts**.

### Transition or "Dead Area"

**Signal voltage region in the range  $>-3.0V$  and  $< +3.0V$  is regarded as the 'dead area' and allows for absorption of noise.** This same region is considered a transition region, and the **signal state is undefined**.

To bring the signal to the "true" state, the controlling device **unasserts** (or **lowers**) the value for data pins and **asserts** (or **raises**) the value for control pins. Conversely, to bring the signal to the "false" state, the controlling device asserts the value for data pins and unasserts the value for control pins. The "true" and "false" states for a data signal and for a control signal are as shown below.

## The Communication Technique



*Data Signal Status*

*Control Signal Status*

A factor that limits the distance of reliable data transfer using RS-232 is the signaling technique that it uses.

- This interface is “*single-ended*” meaning that communication occurs over a ***SINGLE WIRE referenced to GROUND***, the ground wire serving as a second wire. Over that single wire, *marks and spaces* are created.
- While this is very adequate for slower applications, it is not suitable for faster and longer applications.

## The communication technique

- RS-232 is designed for a ***unidirectional half-duplex communications mode***. That simply means that a transmitter (driver) is feeding the data to a receiver over a copper line. The data always follows the direction from driver to receiver over that line. If ***return transmission*** is desired, another set of driver- receiver pair and separate wires are needed. In other words, if ***bi-directional or full-duplex capabilities*** are needed, two separate communications paths are required.

## Disadvantage

Being a *single-ended system* it is more susceptible to induced noise, ground loops and ground shifts, a ground at one end not the same potential as at the other end of the cable e.g. in applications under the proximity of heavy electrical installations and machineries. But these vulnerabilities at very high data rates and for those applications a different standard, like the RS-422 etc., is required which have been explained further.

## Some Modern Perspectives/Advantages

Most applications for RS-232 today are for data connectivity between portable handheld devices and a PC. Some of the differences between the modern RS-232 integrals from the older versions are:

- Such devices require that the RS-232 IC to be very small, have low current drain, operate from a +3 to +5-V supply.
- They provide [ESD](#) protection on all transmit and receive pins. For example, some RS-232 interfaces have specifically been designed *for handheld devices* and support *data rates greater than 250 kbps*, can operate down to +2.7 V.
- They can automatically go into a *standby mode* drawing very small currents of the order of only *150 nA* when not in use, provide *15 kV ESD* protection on data pins and are in the *near-chip-scale 5 X 5 mm quad flat no-lead package*.

Nevertheless, for portable and handheld applications the older RS-232 is still the most popular one.

## RS-422 and RS-423 (EIA Recommended Standard 422 and 423)

These were designed, specifically; to overcome the distance and speed limitations of RS-232. Although they are similar to the more advanced RS-232C, but can accommodate higher baud rates and longer cable lengths and, accommodate multiple receivers.

## The Standard

### Maximum Bit Transfer Rate, Signal Voltages and Cable Length

- For both of these standards the data lines can be up to **4,000 feet** with a data rate around **100 kbps**.
- The **maximum data rate** is around **10 Mbps for short runs**, trading off distance for speed.
- The maximum signal voltage levels are **±6 volts**.
- The signaling technique for the RS-422 and RS-423 is mainly responsible for there superiority over RS-232 in terms of speed and length of transmission as explained in the next subsection.

### Communication Technique

- The flair of this standard lies in its capability in tolerating the ground voltage differences between sender and receiver. Ground voltage differences can occur in **electrically noisy** environments where **heavy electrical machinery** is operating.
- The criterion here is the **differential-data communication technique**, also referred to as **balanced-differential signaling**. In this, the driver uses two wires over which the signal is transmitted. However, each wire is driven and floating separate from ground, meaning, neither is grounded and in this respect this system is different to the single-ended systems. Correspondingly, the receiver has two inputs, each floating above ground and electrically balanced with the other when no data is being transmitted. Data on the line causes a desired electrical imbalance, which is recognized and amplified by the receiver. The **common-mode signals**, such as induced electrical noise on the lines caused from machinery or radio transmissions, are, for the most part, canceled by the receiver. That is because the induced noise is identical on each wire and the receiver **inverts** the signal on one wire to place it **out of phase** with the other causing a **subtraction** to occur which results in a **Zero difference**. Thus, noise picked up by the long data lines is eliminated at the receiver and does not interfere with data transfer. Also, because the line is balanced and separate from ground, there is no problem associated with ground shifts or ground loops.
- It may be mentioned here to avoid any ambiguity in understanding the RS-422 and the RS-423 standards, that, the standard RS-423 is an advanced counterpart of RS-422 which has been designed to tolerate the ground voltage differences between the sender and the receiver for the more advanced version of RS-232, that is, the RS-232C.
- Unlike RS-232, an RS-422 driver can service up to **10 receivers** on the **same line (bus)**. This is often referred to as a **half-duplex single-source multi-drop network**, (not to be confused with **multi-point networks** associated with RS-485), this will be explained further in conjugation with RS-485.

- Like RS-232, however, RS-422 is still ***half-duplex*** one-way data communications over a two-wire line. If ***bi-directional or full-duplex*** operation is desired, another set of driver, receiver(s) and two-wire line is needed. In which case, RS-485 is worth considering.

## Applications

This fits well in process control applications in which instructions are sent out to **many *actuators or responders***. **Ground voltage differences can occur in electrically noisy environments where heavy electrical machinery is operating.**

## RS-485

This is an improved RS-422 with the capability of connecting a number of devices (transceivers) on ***one serial bus to form a network***.

## The Standard

### Maximum Bit Transfer Rate, Signal Voltages and Cable Length

- Such a network can have a "daisy chain" topology where each device is connected to two other devices except for the devices on the ends.
- Only one device may drive data onto the bus at a time. The standard does not specify the rules for deciding who transmits and when on such a network. That solely depends upon the system designer to define.
- Variable data rates are available for this standards but the standard max. data rate is 10 Mbps, however ,some manufacturers do offer up to double the standard range i.e. around 20 Mbps, but of course, it is at the expense of cable width.
- It can connect upto 32 drivers and receivers in fully differential mode similar to the RS – 422.

## Communication Technique

- EIA Recommended Standard 485 is designed to provide ***bi-directional half-duplex multi-point data communications over a single two-wire bus***.
- Like RS-232 and RS-422, full-duplex operation is possible using a ***four-wire, two-bus network but the RS-485 transceiver ICs*** must have separate transmit and receive pins to accomplish this.

- RS-485 has *the same distance and data rate specifications as RS-422* and uses differential signaling but, *unlike RS-422, allows multiple drivers on the same bus*. As depicted in the Figure below, each node on the bus can include both a driver and receiver forming a multi-point star network. Each driver at each node remains in a *disabled high-impedance state* until called upon to transmit. This is different than drivers made for RS-422 where there is only one driver and it is always enabled and cannot be disabled.
- With automatic repeaters and tri-state drivers the 32-node limit can be greatly exceeded. In fact, the ANSI-based SCSI-2 and SCSI-3 bus specifications use RS-485 for the physical (hardware) layer.

## Advantages

- Among all of the asynchronous standards mentioned above this standard offers the maximum data rate.
- Apart from that special hardware for avoiding bus contention and ,
- A higher receiver input impedance with lower Driver load impedances are its other assets.

## Differences between the various standards at a glance

All together the important electrical and mechanical characteristics for application purposes may be classified and summarized according to the table below.

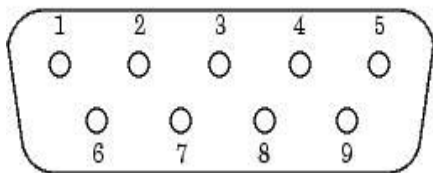
	RS-232	RS-422/423	RS-485
<b>Signaling Technique</b>	Single-Ended (Unbalanced)	Differential (Balanced)	Differential (Balanced)
<b>Drivers and Receivers on Bus</b>	1 Driver  1 Receiver	1 Driver  10 Receivers	32 Drivers  32 Receivers
<b>Maximum Cable Length</b>	50 feet	4000 feet	4000 feet
<b>Original Standard Maximum Data Rate</b>	20 kbps	10 Mbps  down to 100 kbps	10 Mbps  down to 100 kbps

<b>Minimum Loaded Driver Output Voltage Levels</b>	+/-5.0 V	+/-2.0 V	+/-1.5 V
--	----------	----------	----------

<b>Driver Load Impedance</b>	3 to 7 k	100	54
<b>Receiver Input Impedance</b>	3 to 7 k	4 k or greater	12 k or greater

## Interfacing of Peripherals Involving the Rs-232 Asynchronous Communication Standards

The RS-232 standard defines the two devices connected with a serial cable as the *Data Terminal Equipment (DTE)* and *Data Circuit-Terminating Equipment (DCE)*. This terminology reflects the RS-232 origin as a standard for communication between a *computer terminal* and a *modem*. Primary communication is accomplished using three pins: the *Transmit Data (TD)* pin, the *Receive Data (RD)* pin, and the *Ground pin* (not shown). Other pins are available for *data flow control*. The serial port pins and the signal assignments for a typical asynchronous serial communication can be shown in the scheme for a 9-pin male connector (*DB9*) on the *DTE* as under:



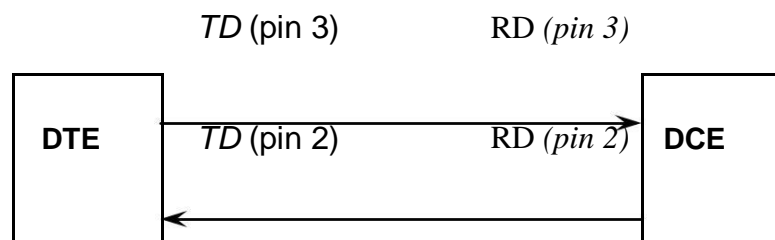
*The DB9 male connector*

Serial Port Pin and Signal Assignments			
<i>Pin</i>	<i>Label</i>	<i>Signal Name</i>	<i>Signal Type</i>
1	CD	<i>Carrier Detect</i>	Control
2	RD	<i>Received Data</i>	Data
3	TD	<i>Transmitted Data</i>	Data
4	DTR	<i>Data Terminal Ready</i>	Control
5	GND	<i>Signal Ground</i>	Ground
6	DSR	<i>Data Set Ready</i>	Control
7	RTS	<i>Request to Send</i>	Control

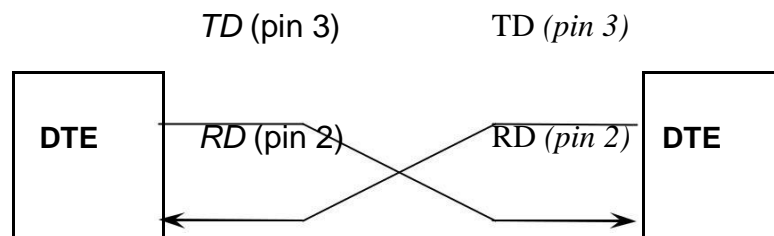
8	CTS	<i>Clear to Send</i>	Control
9	RI	<i>Ring Indicator</i>	Control

(The RS-232 standard can be referred for a description of the signals and pin assignments used for a 25-pin connector)

Because RS-232 mainly involves connecting a DTE to a DCE, the pin assignments are defined such that ***straight-through cabling*** is used, where pin 1 is connected to pin 1, pin 2 is connected to pin 2, and so on. A DTE to DCE serial connection using the ***Transmit Data (TD)*** pin and the ***Receive Data (RD)*** pin is shown below.



Connecting two DTE's or two DCE's using a straight serial cable, means that the TD pin on each device are connected to each other, and the RD pin on each device are connected to each other. Therefore, to connect two like devices, a ***null modem*** cable has to be used. As shown below, null modem cables crosses the transmit and receive lines in the cable.



Serial ports consist of two signal types: data signals and control signals. To support these signal types, as well as the signal ground, the RS-232 standard defines a 25-pin connection. However, most PC's and UNIX platforms use a 9-pin connection. In fact, only three pins are required for serial port communications: one for receiving data, one for transmitting data, and one for the signal ground.

Throughout this discussion computer is considered a DTE, while peripheral devices such as modems and printers are considered DCE's. Note that many scientific instruments function as DTE's.



The term "data set" is synonymous with "modem" or "device," while the term "data terminal" is synonymous with "computer."

## (Detail PC – PC communication....)

**Note:** The serial port pin and signal assignments are with respect to the DTE. For example, data is transmitted from the TD pin of the DTE to the RD pin of the DCE.

## The Data Pins

Most serial port devices support *full-duplex* communication meaning that they can send and receive data at the same time. Therefore, separate pins are used for transmitting and receiving data. For these devices, the TD, RD, and GND pins are used. However, some types of serial port devices support only one-way or *half-duplex* communications. For these devices, only the TD and GND pins are used. In the course of explanation, it is assumed that a full-duplex serial port is connected to the DCE.

The TD pin carries data transmitted by a DTE to a DCE. The RD pin carries data that is received by a DTE from a DCE.

## The Control Pins

9-pin serial ports provide several control pins whose functions are to:

- Signal the presence of connected devices
- Control the flow of data

The control pins include RTS and CTS, DTR and DSR, CD, and RI.

## The RTS and CTS Pins

The RTS and CTS pins are used to signal whether the devices are ready to send or receive data. This type of data flow control - called hardware handshaking - is used to prevent data loss during transmission. When enabled for both the DTE and DCE, hardware handshaking using RTS and CTS follows these steps:

1. The DTE asserts the RTS pin to instruct the DCE that it is ready to receive data.

2. The DCE asserts the CTS pin indicating that it is clear to send data over the TD pin. If data can no longer be sent, the CTS pin is unasserted.
3. The data is transmitted to the DTE over the TD pin. If data can no longer be accepted, the RTS pin is unasserted by the DTE and the data transmission is stopped.

## The DTR and DSR Pins

Many devices use the DSR and DTR pins to signal if they are connected and powered. Signaling the presence of connected devices using DTR and DSR follows these steps:

1. The DTE asserts the DTR pin to request that the DCE connect to the communication line.
2. The DCE asserts the DSR pin to indicate it's connected.
3. DCE unasserts the DSR pin when it's disconnected from the communication line.

The DTR and DSR pins were originally designed to provide an alternative method of hardware handshaking. However, the RTS and CTS pins are usually used in this way, and not the DSR and DTR pins. However, you should refer to your device documentation to determine its specific pin behavior.

## The CD and RI Pins

The CD and RI pins are typically used to indicate the presence of certain signals during modem-modem connections.

CD is used by a modem to signal that it has made a connection with another modem, or has detected a carrier tone. CD is asserted when the DCE is receiving a signal of a suitable frequency. CD is unasserted if the DCE is not receiving a suitable signal.

RI is used to indicate the presence of an audible ringing signal. RI is asserted when the DCE is receiving a ringing signal. RI is unasserted when the DCE is not receiving a ringing signal (for example, it's between rings).

## A Practical Example: PC-PC Communication

**PROBLEM:** Suppose one PC needs to send data to another computer located far away from its vicinity. Now, the actual data is in the parallel form, it needs to be converted into its serial counterpart. This is done by a [Parallel-in-Serial-out Shift register](#) and a [Serial-in-Parallel-out Shift register](#) (some electronic component).

It has to be made sure that the transmitter must not send the data at a rate faster than with which the receiver can receive it. This is done by introducing some handshaking signals or circuitry in conjugation with the actual system.

For very short distances, devices like [UART](#)(Universal Asynchronous Receiver Transmitter: IN8250 from National Semiconductors Corporation) and [USART](#)\_(Universal Synchronous Asynchronous Receiver Transmitter; Intel [8251A](#) from Intel Corporation.) incorporate the essential circuitry for handling this serial communication with handshaking.

For long distances *Telephone lines (switched lines)* are more practically feasible because of there pre-availability.

**ONE COMPLICATION:** ...[BANDWIDTH](#) is only 300 – 3000Hz.

**REMEDY:** [Convert the digital signal to audio tones](#). The device, which is used to do this conversion and vice-versa, is known as a **MODEM**.

## But how all the above Principles are Applied in Practice?

Consider the control room of a steel plant where one main computer is time-sharing and communicating data to and fro with some other computers or I/O modules in a [DCS](#) or [SCADA](#) hierarchy.

### A TYPICAL DIGITAL TRANSMISSION SYSTEM

## Overall Procedure of Communication.....

(Note: This is actually the initialization and handshaking description for a typical UART, the Intel 8251A)..... (for more details click the box here )



To start with, it should be mentioned that the signals alongside the arrowheads represent the minimum number of necessary signals for the execution of a typical communication standard or a protocol; being elaborated later. These signals occur when the main control terminal wants to send some control signal to the end device or if the end device wants to send some data, say an alarm or some process output, to the main controller.

Both the main microcomputer and the end-device or the time-shared device can be referred to as terminals.

Whenever a terminal is switched on it first performs a self-diagnostic test, in which it checks itself and if it finds that its integrity is fully justified it asserts the DTR (*data-terminal ready*) signal low. As the modem senses it getting low, it understands that the terminal is ready.

The modem then “replies” the terminal by asserting DSR (*data-set ready*) signal low. Here the direction of the arrows is of prime importance and must be remembered to get the full understandability of the whole procedure.

If the terminal is actually having some valuable data to convey to the end -terminal it will assert the RTS (*request-to-send*) signal low back to the modem and, in turn, the modem will assert the CD (*carrier-detect*) signal to the terminal indicating as if now it has justified the connection with the terminal computer.

But it may be possible that the modem may not be fully ready to transmit the actual data to the telephone, this may be because of its buffer saturation and several other reasons. When the modem is fully ready to send the data along the telephone line it will assert the CTS (Clear-to-send) signal back to the terminal.

The terminal then starts sending the serial data to the modem and the modem. When the terminal gets exhausted of the data it asserts the RTS signal low indicating the modem that it has not got any more data to be sent. The modem in turn unasserts its CTS signal and stops transmitting.

The same way initialization and the handshaking processes are executed at the other end. Therefore, it must be noted here that the very important aspect of data communication is the definition of the handshaking signals defined for transferring serial data to and from the modem.

## Current loops

Current loops are a standard, which are used widely in process automation. 20 mA are widely used for transmitting serial communication data to programmable process controlling devices. Other widely used standard is 4-20mA current loop, which is used for transmitting analogue measurement signals between the sensor and measurement device.

## Serial communication using current loop

In digital communications 20 mA current loop is a standard. The transmitters will only source 20 mA and the receiver will only sink 20 mA. Current loops often use opto-couplers. Here it is the current which matters and not the voltages.

For measurement purposes a small resistance, say of value  $1k$ , is connected in series with the receiver/transmitter and the current meter. The current flowing into the receiver indicates the *scaled* data, which is actually going inside it. The data transmitted through this kind of interface is usually a standard RS-232 signal just converted to current pulses. Current “on” and “off” the

transmission line depends on how the RS-232 circuit distinguishes between the value of currents and in what way it interprets the logic state thus obtained.

## 4-20 mA current loop

4-20 mA current loop interface is the standard for almost all the process control instruments. This interface works as follows. The sensor is connected to a process controlling equipment, which reads the sensor value and supplies a voltage to the loop where the sensor is connected and reads the amount of current it takes. The typical supply voltage for this arrangement is around **12-24 Volts** through a resistor and the measured output is the voltage drop across that resistor converted into its current counterpart.

The current loop is designed so that a sensor takes 4 mA current when it is at its minimum value and 20 mA when it is in its maximum value.

Because the sensor will always pass at least 4 mA current and there is usually a voltage drop of many volts over the sensor, many sensor types can be made to be powered from only that loop current.

### **WATCH DOG TIMER:**

A **watchdog timer** (sometimes called a *computer operating properly* or *COP* timer, or simply a *watchdog*) is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the computer regularly resets the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to reset the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation.

Watchdog timers are commonly found in embedded systems and other computer-controlled equipment where humans cannot easily access the equipment or would be unable to react to faults in a timely manner. In such systems, the computer cannot depend on a human to invoke a reboot if it hangs; it must be self-reliant. For example, remote embedded systems such as space probes are not physically accessible to human operators; these could become permanently disabled if they were unable to autonomously recover from faults. A watchdog timer is usually employed in cases like these. Watchdog timers may also be used when running untrusted code in a sandbox, to limit the CPU time available to the code and thus prevent some types of denial-of-service attacks.

**UNIT-III**  
**EMBEDDED FIRMWARE DESIGN**

Two basic approaches are used for desing they are

1. Conventional procedure based firmware design
2. Embedded operating system(OS) based design

### **CONVENTIONAL PROCEDURE BASED FIRMWARE DESIGN:**

This is also called super loop based design. When programming an embedded system, it is important to meet the time deadlines of the system, and to perform all the tasks of the system in a reasonable amount of time, but also in a good order. This page will talk about a common program architecture called the **Super-Loop Architecture** that is very useful in meeting these requirements.

A super loop is a program structure comprised of an infinite loop, with all the tasks of the system contained in that loop. Here is a general pseudocode for a superloop implementation:

```
Function Main_Function()
{
    Initialization();
    Do_Forever
    {
        Check_Status();
        Do_Calculations();
        Output_Response();
    }
}
```

We perform the initialization routines before we enter the super loop, because we only want to initialize the system once. Once the infinite loop begins, we don't want to reset the values, because we need to maintain persistent state in the embedded system.

The loop is in fact a variant of the classic "batch processing" control flow: Read input, calculate some values, write out values. Do it until you run out of input data "cards". So, embedded systems software is not the only type of software which uses this kind of architecture. For example, computer games often use a similar loop. There the loop is called (*tight*) (*main*) *game loop*.

### **EMBEDDED OPERATING SYSTEM(OS) BASED DESIGN**

An **embedded operating system** is an operating system for embedded computer systems. This type of operating system is typically designed to be resource-efficient and reliable. Resource efficiency comes at the cost of losing some functionality or granularity that larger computer operating systems provide, including functions which may not be used by the specialized applications they run. Depending on the method used for multitasking, this type of OS is frequently considered to be a real-time operating system, or *RTOS*.

**Ex:** VxWorks, Microkernels, ThreadX, Symbian, Embedded Linux.



## EMBEDDED FIRMWARE DEVELOPMENT LANGUAGES

### ASSEMBLY LANGUAGE BASED DEVELOPMENT:

Assembly languages were developed to provide **mnemonics** or symbols for the machine level code instructions. Assembly language programs consist of mnemonics, thus they should be translated into machine code. A program that is responsible for this conversion is known as **assembler**. Assembly language is often termed as a low-level language because it directly works with the internal structure of the CPU. To program in assembly language, a programmer must know all the registers of the CPU.

Different programming languages such as C, C++, Java and various other languages are called high-level languages because they do not deal with the internal details of a CPU. In contrast, an assembler is used to translate an assembly language program into machine code (sometimes also called **object code** or **opcode**). Similarly, a compiler translates a high-level language into machine code. For example, to write a program in C language, one must use a C compiler to translate the program into machine language.

#### Structure of Assembly Language

An assembly language program is a series of statements, which are either assembly language instructions such as ADD and MOV, or statements called **directives**.

An **instruction** tells the CPU what to do, while a **directive** (also called **pseudo-instructions**) gives instruction to the assembler. For example, ADD and MOV instructions are commands which the CPU runs, while ORG and END are assembler directives. The assembler places the opcode to the memory location 0 when the ORG directive is used, while END indicates to the end of the source code. A program language instruction consists of the following four fields –

[ label: ] mnemonics [ operands ] [;comment ]

### HIGH LEVEL LANGUAGE BASED DEVELOPMENT

"High-level language" refers to the higher level of abstraction from machine language. Rather than dealing with registers, memory addresses and call stacks, high-level languages deal with variables, arrays, objects, complex arithmetic or boolean expressions, subroutines and functions, loops, threads, locks, and other abstract computer science concepts, with a focus on usability over optimal program efficiency. Unlike low-level assembly languages, high-level languages have few, if any, language elements that translate directly into a machine's native opcodes. Other features, such as string handling routines, object-oriented language features, and file input/output, may also be present. One thing to note about high-level programming languages is that these languages allow the programmer to be detached and separated from the machine. That is, unlike low-level languages like assembly or machine language, high-level programming can amplify the programmer's instructions and trigger a lot of data movements in the background without their knowledge. The responsibility and power of executing instructions have been handed over to the machine from the programmer.

There are three general modes of execution for modern high-level languages:

### **Interpreted**

When code written in a language is interpreted, its syntax is read and then executed directly, with no compilation stage. A program called an *interpreter* reads each program statement, following the program flow, then decides what to do, and does it. A hybrid of an interpreter and a compiler will compile the statement into machine code and execute that; the machine code is then discarded, to be interpreted anew if the line is executed again. Interpreters are commonly the simplest implementations of the behavior of a language, compared to the other two variants listed here.

### **Compiled**

When code written in a language is compiled, its syntax is transformed into an executable form before running. There are two types of compilation:

Machine code generation

Some compilers compile source code directly into machine code. This is the original mode of compilation, and languages that are directly and completely transformed to machine-native code in this way may be called "truly compiled" languages. See assembly language.

### **Intermediate representations**

When code written in a language is compiled to an intermediate representation, that representation can be optimized or saved for later execution without the need to re-read the source file. When the intermediate representation is saved, it may be in a form such as byte code. The intermediate representation must then be interpreted or further compiled to execute it. Virtual machines that execute byte code directly or transform it further into machine code have blurred the once clear distinction between intermediate representations and truly compiled languages.

## **INTERRUPTS**

An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine (ISR)** or **Interrupt Handler**. ISR tells the processor or controller what to do when the interrupt occurs. The interrupts can be either hardware interrupts or software interrupts.

## Hardware Interrupt

A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like a disk controller or an external peripheral. For example, when we press a key on the keyboard or move the mouse, they trigger hardware interrupts which cause the processor to read the keystroke or mouse position.

## Software Interrupt

A software interrupt is caused either by an exceptional condition or a special instruction in the instruction set which causes an interrupt when it is executed by the processor. For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero exception, thus causing the computer to abandon the calculation or display an error message. Software interrupt instructions work similar to subroutine calls.

## What is Polling?

The state of continuous monitoring is known as **polling**. The microcontroller keeps checking the status of other devices; and while doing so, it does no other operation and consumes all its processing time for monitoring. This problem can be addressed by using interrupts.

In the interrupt method, the controller responds only when an interruption occurs. Thus, the controller is not required to regularly monitor the status (flags, signals etc.) of interfaced and inbuilt devices.

## Interrupts v/s Polling

Here is an analogy that differentiates an interrupt from polling –

### Interrupt

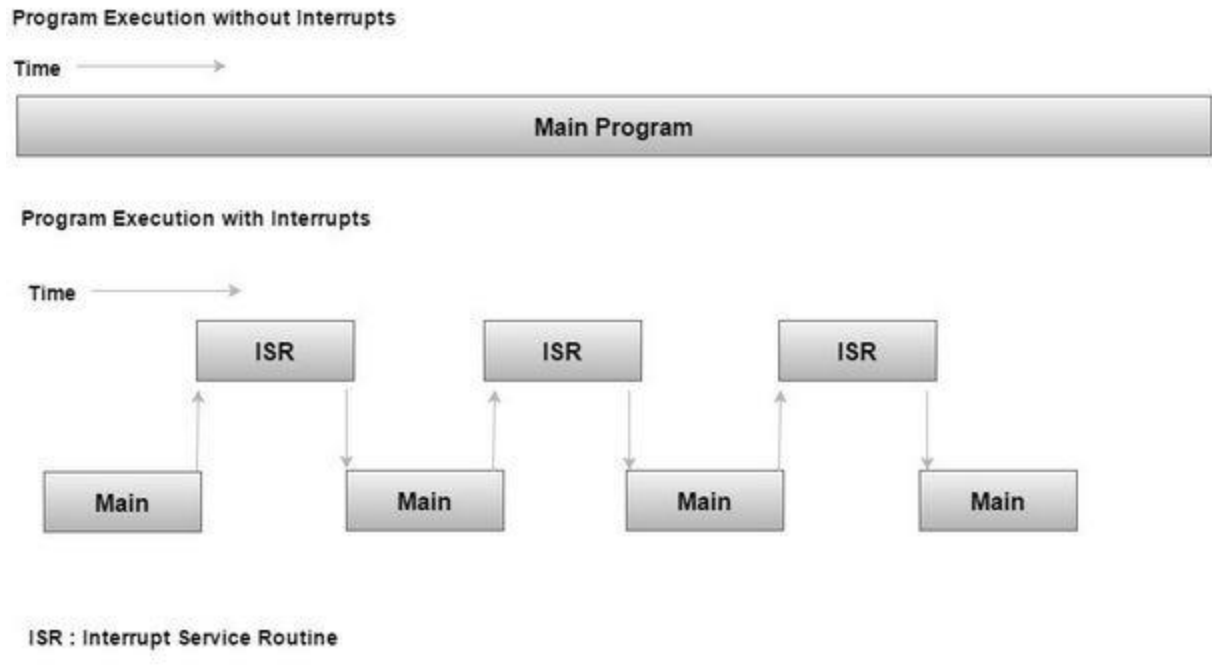
An interrupt is like a **shopkeeper**. If one needs a service or product, he goes to him and appraises him of his needs. In case of an interrupt, when the flags or signals are received, they notify the controller that they need to be serviced.

### Polling

The polling method is like a **salesperson**. The salesman goes from door to door while requesting to buy a product or service. Similarly, the controller keeps monitoring the flags or signals one by one for received, they notify the controller that they all devices and provides service to whichever component that needs its service.

## Interrupt Service Routine

For every interrupt, there must be an interrupt service routine (ISR), or **interrupt handler**. When an interrupt occurs, the microcontroller runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its interrupt service routine, ISR. The table of memory locations set aside to hold the addresses of ISRs is called as the Interrupt Vector Table.



## Steps to Execute an Interrupt

When an interrupt gets active, the microcontroller goes through the following steps –

- The microcontroller closes the currently executing instruction and saves the address of the next instruction (PC) on the stack.
- It also saves the current status of all the interrupts internally (i.e., not on the stack).
- It jumps to the memory location of the interrupt vector table that holds the address of the interrupt service routine.
- The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it. It starts to execute the interrupt service subroutine, which is RETI (return from interrupt).
- Upon executing the RETI instruction, the microcontroller returns to the location where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then, it starts to execute from that address.

## Interrupt inside Interrupt

What happens if the 8051 is executing an ISR that belongs to an interrupt and another one gets active? In such cases, a high-priority interrupt can interrupt a low-priority interrupt. This is known as **interrupt inside interrupt**. In 8051, a low-priority interrupt can be interrupted by a high-priority interrupt, but not by any other low-priority interrupt.

## Triggering an Interrupt by Software

There are times when we need to test an ISR by way of simulation. This can be done with the simple instructions to set the interrupt high and thereby cause the 8051 to jump to the interrupt vector table. For example, set the IE bit as 1 for timer 1. An instruction **SETB TF1** will interrupt the 8051 in whatever it is doing and force it to jump to the interrupt vector table.

## DMA

**Direct memory access (DMA)** is a feature of computer systems that allows certain hardware subsystems to access main system memory (random-access memory), independent of the central processing unit (CPU).

Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the CPU first initiates the transfer, then it does other operations while the transfer is in progress, and it finally receives an interrupt from the DMA controller when the operation is done. This feature is useful at any time that the CPU cannot keep up with the rate of data transfer, or when the CPU needs to perform work while waiting for a relatively slow I/O data transfer. Many hardware systems use DMA, including disk drive controllers, graphics cards, network cards and sound cards. DMA is also used for intra-chip data transfer in multi-core processors. Computers that have DMA channels can transfer data to and from devices with much less CPU overhead than computers without DMA channels. Similarly, a processing element inside a multi-core processor can transfer data to and from its local memory without occupying its processor time, allowing computation and data transfer to proceed in parallel.

DMA can also be used for "memory to memory" copying or moving of data within memory. DMA can offload expensive memory operations, such as large copies or scatter-gather operations, from the CPU to a dedicated DMA engine. An implementation example is the I/O Acceleration Technology. DMA is of interest in network-on-chip and in-memory computing architectures.

## ***DEVICE DRIVER PROGRAMMING***

A device driver has a set of routines (functions) used by a high-level language programmer, which does the interaction with the device hardware, sends control commands to the device, communicates data to the device and runs the codes for reading device data.

Each device in a system needs device driver routine with number of device functions.

- An ISR relates to a device driver command (device-function). The device driver uses SWI to call the related ISR (device-function routine)
- The device driver also responds to device hardware interrupts. Device driver routine

## **Device driver generic commands**

- A programmer uses generic commands for device driver for using a device. The operating system provides these generic commands.
- Each command relates to an ISR. The device driver command uses an SWI to call the related ISR device-function routine)

## **Generic functions**

- Generic functions used for the commands to the device are device create ( ), open ( ), connect ( ), bind ( ), read ( ), write ( ), ioctl ( ) [for IO control], delete ( ) and close ( ).

## **Device driver code**

Different in different operating system. Same device may have different code for the driver when system is using different operating system

Interrupt service routines Interrupt service routines An Interrupt service routine (ISR) accesses a device for service (configuring, initializing, activating, opening, attaching, reading, writing, resetting, deactivating or closing). Interrupt service routines thus implements the device functions of the device driver

## **C Vs Embedded C**

In the C standard, a standalone implementation doesn't have to provide all of the library functions that a hosted implementation has to provide. The C standard doesn't care about embedded, but vendors of embedded systems usually provide standalone implementations with whatever amount of libraries they're willing to provide.

C is a widely used general purpose high level programming language mainly intended for system programming.

Embedded C is an extension to C programming language that provides support for developing efficient programs for embedded devices

## **Compiler Vs Cross Compiler**

A **compiler** is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. Typically, a programmer writes language statements in a language such as Pascal or C one line at a time using an editor.

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. Cross compiler tools are generally found in use to generate compiles for embedded system or multiple platforms. It is a tool that one must use for a platform where it is inconvenient or impossible to compile on that platform, like microcontrollers that run

with a minimal amount of memory for their own purpose. It has become more common to use this tool for paravirtualization where a system may have one or more platforms in use.

The fundamental use of a cross compiler is to separate the build environment from the target Embedded computers where a device has extremely limited resources. For example, a microwave oven will have an extremely small computer to read its touchpad and door sensor, provide output to a digital display and speaker, and to control the machinery for cooking food. This computer will not be powerful enough to run a compiler, a file system, or a development environment. Since debugging and testing may also require more resources than is available on an embedded system, cross-compilation can be more involved and prone to errors than native compilation.

Compiling for multiple machines. For example, a company may wish to support several different versions of an operating system or to support several different operating systems. By using a cross compiler, a single build environment can be set up to compile for each of these targets. Bootstrapping to a new platform. When developing software for a new platform, or the emulator Use of virtual machines (such as Java's JVM) resolves some of the reasons for which cross Typically the hardware architecture differs (e.g. compiling a program destined for the MIPS architecture on an x86 computer) but cross-compilation is also applicable when only the operating system environment differs, as when compiling a FreeBSD program under Linux, or even just the system library, as when compiling programs with uClibc on a glibc host.

**UNIT-IV**  
**REAL TIME OPERATING SYSTEM**



## **OPERATING SYSTEM BASICS**

An **operating system (OS)** is system software that manages computer hardware and software resources and provides common services for computer programs.

Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware,<sup>[1][2]</sup> although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or is interrupted by it. Operating systems are found on many devices that contain a computer – from cellular phones and video game consoles to web servers and supercomputers.

The dominant desktop operating system is Microsoft Windows with a market share of around 82.74%. macOS by Apple Inc. is in second place (13.23%), and the varieties of Linux are collectively in third place (1.57%).<sup>[3]</sup> In the mobile (smartphone and tablet combined) sector, use in 2017 is up to 70% of Google's Android<sup>[4]</sup> and according to third quarter 2016 data, Android on smartphones is dominant with 87.5 percent and a growth rate 10.3 percent per year, followed by Apple's iOS with 12.1 percent and a per year decrease in market share of 5.2 percent, while other operating systems amount to just 0.3 percent.<sup>[5]</sup> Linux distributions are dominant in the server and supercomputing sectors. Other specialized classes of operating systems, such as embedded and real-time systems, exist for many applications.

### **Types of operating systems**

#### **Batch operating system**

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows –

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

## Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if  $n$  users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows –

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows –

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

## Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred to as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred to as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows –

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

## **Network operating System**

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows –

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows –

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

## **Real Time operating System**

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

### **Hard real-time systems**

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

### **Soft real-time systems**

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers, etc.

## **TASKS of OS**

Following are some of important tasks of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

### **Memory Management**

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management –

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part are not in use.
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

## Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management –

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

## Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management –

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

## File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management –

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

## Other Important Activities

Following are some of the important activities that an Operating System performs –

- **Security** – By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** – Recording delays between request for a service and response from the system.
- **Job accounting** – Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** – Production of dumps, traces, error messages, and other debugging and error detecting aids.

- **Coordination between other softwares and users** – Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

## MULTITASKING AND MULTI THREADING

In computing, **multitasking** is the concurrent execution of multiple tasks (also known as processes) over a certain period of time. New tasks can interrupt already started ones before they finish, instead of waiting for them to end. As a result, a computer executes segments of multiple tasks in an interleaved manner, while the tasks share common processing resources such as central processing units (CPUs) and main memory. Multitasking automatically interrupts the running program, saving its state (partial results, memory contents and computer register contents) and loading the saved state of another program and transferring control to it. This "context switch" may be initiated at fixed time intervals (pre-emptive multitasking), or the running program may be coded to signal to the supervisory software when it can be interrupted (cooperative multitasking).

Multitasking does not require parallel execution of multiple tasks at exactly the same time; instead, it allows more than one task to advance over a given period of time.<sup>[1]</sup> Even on multiprocessor computers, multitasking allows many more tasks to be run than there are CPUs.

Multitasking is a common feature of computer operating systems. It allows more efficient use of the computer hardware; where a program is waiting for some external event such as a user input or an input/output transfer with a peripheral to complete, the central processor can still be used with another program. In a time sharing system, multiple human operators use the same processor as if it was dedicated to their use, while behind the scenes the computer is serving many users by multitasking their individual programs. In multiprogramming systems, a task runs until it must wait for an external event or until the operating system's scheduler forcibly swaps the running task out of the CPU. Real-time systems such as those designed to control industrial robots, require timely processing; a single processor might be shared between calculations of machine movement, communications, and user interface.<sup>[2]</sup>

Often multitasking operating systems include measures to change the priority of individual tasks, so that important jobs receive more processor time than those considered less significant. Depending on the operating system, a task might be as large as an entire application program, or might be made up of smaller threads that carry out portions of the overall program.

A processor intended for use with multitasking operating systems may include special hardware to securely support multiple tasks, such as memory protection, and protection rings that ensure the supervisory software cannot be damaged or subverted by user-mode program errors.

The term "multitasking" has become an international term, as the same word is used in many other languages such as German, Italian, Dutch, Danish and Norwegian.

## Cooperative multitasking

Early multitasking systems used applications that voluntarily ceded time to one another. This approach, which was eventually supported by many computer operating systems, is known today as cooperative multitasking. Although it is now rarely used in larger systems except for specific applications such as CICS or the JES2 subsystem, cooperative multitasking was once the only scheduling scheme employed by Microsoft Windows and Classic Mac OS to enable multiple applications to run simultaneously. Cooperative multitasking is still used today on RISC OS systems.

## Preemptive multitasking

Preemptive multitasking allows the computer system to more reliably guarantee to each process a regular "slice" of operating time. It also allows the system to deal rapidly with important external events like incoming data, which might require the immediate attention of one or another process. Operating systems were developed to take advantage of these hardware capabilities and run multiple processes preemptively. Preemptive multitasking was implemented in the PDP-6 Monitor and MULTICS in 1964, in OS/360 MFT in 1967, and in Unix in 1969, and was available in some operating systems for computers as small as DEC's PDP-8; it is a core feature of all Unix-like operating systems, such as Linux, Solaris and BSD with its derivatives,<sup>[4]</sup> as well as modern versions of Windows.

## THREAD

A thread is a flow of execution through the process code, with its own program counter that keeps track of which instruction to execute next, system registers which hold its current working variables, and a stack which contains the execution history. A thread shares with its peer threads few information like code segment, data segment and open files. When one thread alters a code segment memory item, all other threads see that. A thread is also called a **lightweight process**. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

## Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- It is more economical to create and context switch threads.
- Threads allow utilization of multiprocessor architectures to a greater scale and efficiency.

## Types of Thread

Threads are implemented in following two ways –

- **User Level Threads** – User managed threads.
- **Kernel Level Threads** – Operating System managed threads acting on kernel, an operating system core.

## User Level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.

### Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.
- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

### Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

## Kernel Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individuals threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

### Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can be multithreaded.



## Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

## PROCESS AND SCHEDULING

The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy. Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

### Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues –

- **Job queue** – This queue keeps all the processes in the system.
- **Ready queue** – This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** – The processes which are blocked due to unavailability of an I/O device constitute this queue.

### Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types –

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

### Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

## Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

## Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

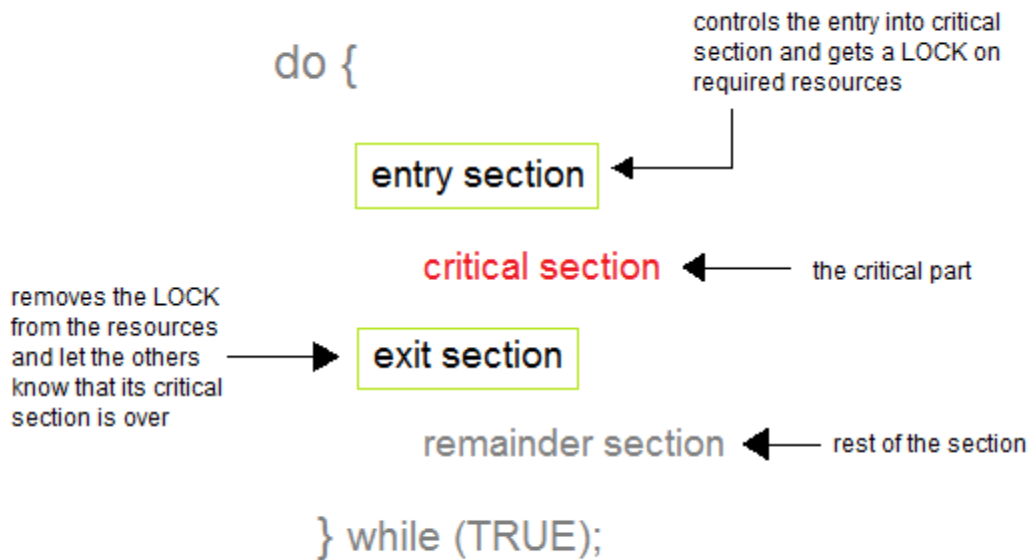
## Process Synchronization

Process Synchronization means sharing system resources by processes in a such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes.

Process Synchronization was introduced to handle problems that arose while multiple process

## Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



## Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

### 1. Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

### 2. Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

### 3. Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, system must grant the process permission to get into its critical section.

## Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

Disabling interrupt on a multiprocessor environment can be time consuming as the message is passed to all the processors.

This message transmission lag, delays entry of threads into critical section and the system efficiency decreases.

## **Mutex Locks**

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside critical section, and in the exit section that LOCK is released.

## **DEVICE DRIVER**

A device driver has a set of routines (functions) used by a high-level language programmer, which does the interaction with the device hardware, sends control commands to the device, communicates data to the device and runs the codes for reading device data.

Each device in a system needs device driver routine with number of device functions.

- An ISR relates to a device driver command (device-function). The device driver uses SWI to call the related ISR (device-function routine)
- The device driver also responds to device hardware interrupts. Device driver routine

## **Device driver generic commands**

- A programmer uses generic commands for device driver for using a device. The operating system provides these generic commands.
- Each command relates to an ISR. The device driver command uses an SWI to call the related ISR device-function routine)

## **Generic functions**

- Generic functions used for the commands to the device are device create ( ), open ( ), connect ( ), bind ( ), read ( ), write ( ), ioctl ( ) [for IO control], delete ( ) and close ( ).

## **Device driver code**

Different in different operating system. Same device may have different code for the driver when system is using different operating system

Interrupt service routines Interrupt service routines An Interrupt service routine (ISR) accesses a device for service (configuring, initializing, activating, opening, attaching, reading, writing, resetting, deactivating or closing). Interrupt service routines thus implements the device functions of the device driver

## HOW TO CHOOSE AN RTOS

Ranking RTOS is both a tricky and difficult task because there are so many competent choices that are available in the market [44]. The developer can choose from commercial RTOS (44% developers are using) or open- source RTOS (20 %) or internally developed RTOS (17 %). This shows that almost 70% of developers are using the RTOS for their current projects [43] and are migrating from one RTOS to another due to various reasons. To handle the current requirements of the customers, developers are using 32 bit controllers in their projects; in which 92% projects/products are using RTOS [44] and 50% of developers are migrating to another RTOS for their next project. This influences importance of the selection of right RTOS for a particular project so that it meets all the requirements, and fulfills its intended task. Literature survey revealed most of the authors used the elimination criteria, which are manual and which takes more time, and also require the detailed specifications of all the existing commercial RTOSs. In order to select the RTOS, the designer first identifies the parameters for selection based on the application and the intended requirements are provided to the systems through an interactive GUI shown in Figure 6.1. The designer has the freedom to omit and or include parameters, and also he/she can edit the database of RTOS for efficient selection under multi-user environment. Subsequently, genetic algorithm is used to arrive at the RTOS taking into account the parameters that are specified.

# HARDWARE SOFTWARE CODESIGN

In hardware software co-design, the designer specifies the structure and behavior of the system using finite state machines which communicate among themselves. Then a series of testing, simulation and formal verification are done on these state machines before deciding which components go into the hardware and which of these into the software. The hardware is usually done in field programmable gate arrays (FPGAs) or application specific integrated circuits (ASICs), whereas the software part is translated into low-level programming language. This approach mostly applies in embedded systems which is defined as a collection of programmable parts that interact continuously with environment through sensors. Existing techniques<sup>[18]</sup> are intended for generating simple micro-controllers and their drivers.

Two approaches for the embedded system design device programmer system design device programmer (1)When the software development cycle ends then the cycle begins for the process of integrating the software into the hardware at the time when a system is designed. (2) Both cycles concurrently proceed when co-designing a time critical sophisticated system

#### Software Hardware Tradeoff Software Hardware Tradeoff

It is possible that certain subsystems in hardware (microcontroller), IO memory accesses, real-time clock, system clock, pulse width modulation, timer and serial communication are also implemented by the software.

A serial communication real-time clock and timers featuring microcontroller may cost more than the microprocessor with external memory and a software implementation. Hardware implementations provide advantage of processing speed

#### Hardware implementation advantages Hardware implementation advantages

- (i) Reduced memory for the program. (ii) Reduced number of chips but at an increased cost.
- (iii) Simple coding for the device drivers.
- (iv) Internally embedded codes, which are more secure than at the external ROM (v) Energy dissipation can be controlled by controlling the clock rate and voltage

#### Software implementation advantages Software implementation advantages

- (i) Easier to change when new hardware versions become available. (ii) Programmability for complex operations. (iii) Faster development time. (iv) Modularity and portability.
- (v) Use of standard software engineering, modeling and RTOS tools. (vi) Faster speed of operation of complex functions with high-speed microprocessors. (vii) Less cost for simple systems.

#### Choosing the right platform

Units to be chosen chosen

Processor ASIP or ASSP Multiple processors System-on-Chip Memory Other Hardware Units of System Buses

Units to be chosen Units to be chosen

Software Language RTOS (real-time programming OS) Code generation tools Tools for finally embedding the software into binary image

Embedded System Processors Choice Embedded System Processors Choice

Processor Less System System with Microprocessor or Microcontroller or DSP System with Single purpose processor or ASSP in VLSI or FPGA

Processor Less System

Factors and Needed Features Taken into Factors and Needed Features Taken into Consideration Consideration When the 32-bit system, 16kB+ on chip memory and need of cache, memory management unit or SIMD or MIMD or DSP instructions arise, we use a microprocessor or DSP. Video game, voice recognition and image-filtering systems– need a DSP.

Factors and Needed Features Taken into Factors and Needed Features Taken into Consideration... Consideration... Microcontroller provides the advantage of on-chip memories and subsystems like the timers.

Factors for On Factors for On-Chip Feature Chip Feature

8 bit or 16 bit or 32 bit ALU Cache, Memory Management Unit or DSP calculationsI Intensive computations at fast rate Total external and internal Memory up to or more than Internal RAM Internal ROM/EPROM/EEPROM Flash Timer 1, 2 or 3 Watchdog Timer Serial Peripheral Interface Full duplex Serial Synchronous Communication Interface (SI) Half Duplex

Serial UART Input Captures and Out-compares PWM Single or multi-channel ADC with or without programmable Voltage reference (single or dual reference) DMA Controller Power Dissipation

Hardware Sensitive Programming Hardware Sensitive Programming

Processor Sensitive Processor Sensitive

Can have memory mapped IOsor IO mapped IOs. IO instructions are processor sensitive. Fixed point ALU only or Floating-point operations preetn Provision for execution of SIMD (single instruction multiple data) and VLIW (very large instruction word) instructions.

Programming of the modules needing SIMD and VLIW instructions is handled differently in different processors. Assembly language– sometimes facilitate an optimal use of the processor's special features and instructions. Advanced processors– usually provide the compiler or optimizing compiler sub-unit to obviate need for programming in assembly.

Memory Sensitive Memory Sensitive

Real-time programming model and algorithm used by a programmer depend on memory available and processor performance. Memory address of IO device registers, buffers, control-registers and vector addresses for the interrupt sources or source groups are prefixed in a microcontroller.

Memory Sensitive Memory Sensitive

Programming for these takes into account the addresses. Same addresses must be allotted for these by the RTOS. Memory-sensitive programs need to be optimized for the memory use by skillful programming for example, ARM Thumb® instruction set use



ALLOCATION OF ADDRESSES TO MEMORY ALLOCATION OF ADDRESSES TO MEMORY Program segments and device addresses Different sets and different structures of data at the memory Device, Internal Devices and I/O devices Addresses and Device Drivers

Porting Issues Porting Issues

Byte order Data Alignment Linked Lists Memory Page Size Time Intervals

Performance Metrics Performance Metrics

Performance Modeling for metric System Performance Index as Performance Metric Multiprocessor system performance as Performance Metric MFLOPs and DMIPS (Dhrystone/s) as Performance Indices as Performance Metrics Buffer Requirement, IO performance and Bandwidth Requirement as Performance Metrics

Performance Accelerators Performance Accelerators

Conversion of CDFGs into DFGs for example by using loop flattening (loops are converted to straight program flows) and using look-up tables instead of control condition tests to decide a program flow path

Performance Accelerators Performance Accelerators Reusing the used arrays in memory, appropriate variable selection, appropriate memory allocation and de-allocation strategy • Using stacks as data structure when feasible in-place of queue and using queue in place of list, whenever feasible.

Computing slowest cycle first and examining the possibilities of its speed-up. Code such that more words are fetched from ROM as a byte than the multibyte words

**UNIT-V**  
**EMBEDDED SYSTEM DEVELOPMENT**

## INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

An **integrated development environment (IDE)** is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools, and a debugger. Most modern IDEs have intelligent code completion. Some IDEs, such as NetBeans and Eclipse, contain a compiler, interpreter, or both; others, such as SharpDevelop and Lazarus, do not. The boundary between an integrated development environment and other parts of the broader software development environment is not well-defined. Sometimes a version control system, or various tools to simplify the construction of a graphical user interface (GUI), are integrated. Many modern IDEs also have a class browser, an object browser, and a class hierarchy diagram, for use in object-oriented software development.

## TYPES OF FILES GENERATED ON CROSS COMPILER

.hex file is the actually hexadecimal code that will be read by your programmer and feeded into your microcontroller. It's the machine code and varies from machine to machine.

.o file is the object file. When the compiler compiles the file, an object file is generated. This file contains all the individual parts of the program packaged into pluggable units for the deployment environment i.e the microcontroller.

.lst is the listing file. It is actually useful only if you are using multiple C programming files to be compiled. For single files the listing file just acts as directive for linkers.

.map is the mapping file. It has the real address location mapping for the program to be deployed via hex. The map file is only used during compilation. Only file that is burnt into the microprocessor is the .hex file.

.asm is the assembler code. This file contains the assembly language code. It is never generated. In most embedded C compilers or IDE's you can write either C code or assembly code. If you write C code then you'll use .C extension and if you write assembly code then you will use the .asm file extension.

## DEASSEMBLER AND DECOMPILER

In essence, a **disassembler** is the exact opposite of an assembler. Where an assembler converts code written in an assembly language into binary machine code, a disassembler reverses the process and attempts to recreate the assembly code from the binary machine code. Since most assembly languages have a one-to-one correspondence with underlying machine instructions, the process of disassembly is relatively straight-forward, and a basic disassembler can often be implemented simply by reading in bytes, and performing a table lookup. Of course, disassembly has its own problems and pitfalls, and they are covered later in this chapter. Many disassemblers have the option to output assembly language instructions in Intel, AT&T, or (occasionally) HLA

syntax. Examples in this book will use Intel and AT&T syntax interchangeably. We will typically not use HLA syntax for code examples, but that may change in the future.

Akin to Disassembly, **Decompilers** take the process a step further and actually try to reproduce the code in a high level language. Frequently, this high level language is C, because C is simple and primitive enough to facilitate the decompilation process. Decompilation does have its drawbacks, because lots of data and readability constructs are lost during the original compilation process, and they cannot be reproduced. Since the science of decompilation is still young, and results are "good" but not "great", this page will limit itself to a listing of decompilers, and a general (but brief) discussion of the possibilities of decompilation.

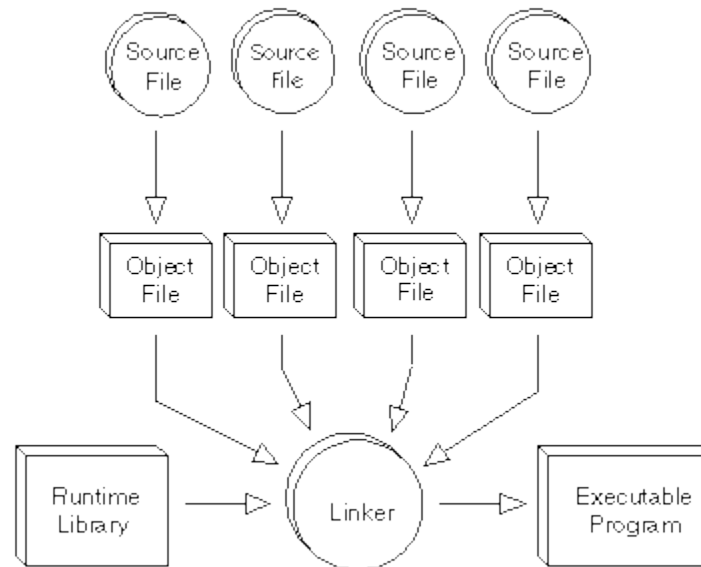
## EMULATOR AND DEBUGGING:

In computing, an **emulator** is hardware or software that enables one computer system (called the *host*) to behave like another computer system (called the *guest*). An emulator typically enables the host system to run software or use peripheral devices designed for the guest system. Emulation refers to the ability of a computer program in an electronic device to emulate (or imitate) another program or device. Many printers, for example, are designed to emulate Hewlett-Packard LaserJet printers because so much software is written for HP printers. If a non-HP printer emulates an HP printer, any software written for a real HP printer will also run in the non-HP printer emulation and produce equivalent printing. Since at least the 1990s, many video game enthusiasts have used emulators to play classic (and/or forgotten) arcade games from the 1980s using the games' original 1980s machine code and data, which is interpreted by a current-era system. A hardware emulator is an emulator which takes the form of a hardware device. Examples include the DOS-compatible card installed in some 1990s-era Macintosh computers like the Centris 610 or Performa 630 that allowed them to run personal computer (PC) software programs and FPGA-based hardware emulators. In a theoretical sense, the Church-Turing thesis implies that (under the assumption that enough memory is available) any operating environment can be emulated within any other environment. However, in practice, it can be quite difficult, particularly when the exact behavior of the system to be emulated is not documented and has to be deduced through reverse engineering. It also says nothing about timing constraints; if the emulator does not perform as quickly as the original hardware, the emulated software may run much more slowly than it would have on the original hardware, possibly triggering timer interrupts that alter behavior.

A **debugger** or **debugging tool** is a computer program that is used to test and debug other programs (the "target" program). The code to be examined might alternatively be running on an *instruction set simulator* (ISS), a technique that allows great power in its ability to halt when specific conditions are encountered, but which will typically be somewhat slower than executing the code directly on the appropriate (or the same) processor. Some debuggers offer two modes of operation, full or partial simulation, to limit this impact. A "trap" occurs when the program cannot normally continue because of a programming bug or invalid data. For example, the program might have tried to use an instruction not available on the current version of the CPU or attempted to access unavailable or protected memory. When the program "traps" or reaches a preset condition, the debugger typically shows the location in the original code if it is a **source-level debugger** or **symbolic debugger**, commonly now seen in integrated development

environments. If it is a **low-level debugger** or a **machine-language debugger** it shows the line in the disassembly (unless it also has online access to the original source code and can display the appropriate section of code from the assembly or compilation).

## EMBEDDED SOFTWARE DEVELOPMENT PROCESS AND TOOLS



### Embedded Software

- Editor. The first tool you need for **Embedded Systems Software Development Tools** is text editor. ...
- Compiler. The second among **Embedded Systems Software Development Tools** is a compiler. ...
- Assembler. ...
- Debugger. ...
- Linker. ...
- Libraries. ...
- Simulator.

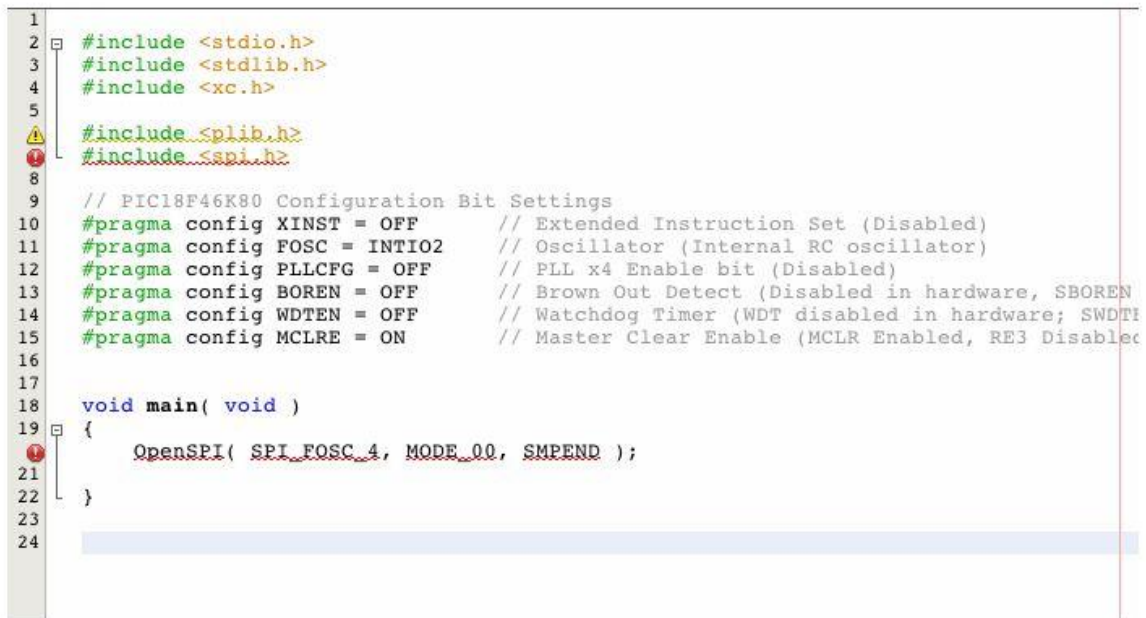
**UNIT-VI**  
**EMBEDDED SYSTEM IMPLEMENTATION**  
**AND TESTING**

## TYPES OF EMBEDDED SYSTEMS DEVELOPMENT TOOLS

There are two types of embedded system development tools

- Embedded software development tools
- Embedded Hardware development tools
  - All kinds of Embedded Systems need software's to run them for performing specific functions. The microcontroller contains the software for handling all the operations. For the development of software of the embedded system, there are different tools that include a compiler, editor, debugger, and assembler. Let's discuss these embedded system software development tools in detail.

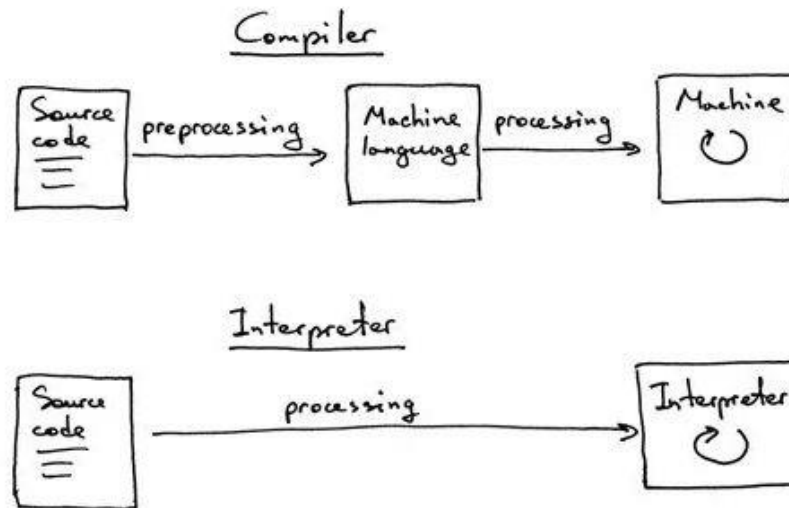
### Editor



```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <xc.h>
5
6 #include <plib.h>
7 #include <spi.h>
8
9 // PIC18F46K80 Configuration Bit Settings
10 #pragma config XINST = OFF      // Extended Instruction Set (Disabled)
11 #pragma config FOSC = INTIO2    // Oscillator (Internal RC oscillator)
12 #pragma config PLLCFG = OFF     // PLL x4 Enable bit (Disabled)
13 #pragma config BOREN = OFF      // Brown Out Detect (Disabled in hardware, SBOREN
14 #pragma config WDTEN = OFF      // Watchdog Timer (WDT disabled in hardware; SWDT
15 #pragma config MCLRE = ON       // Master Clear Enable (MCLR Enabled, RE3 Disab
16
17
18 void main( void )
19 {
20     OpenSPI( SPI_FOSC_4, MODE_00, SMPEND );
21 }
22
23
24
```

- The very first tool in the development of software for an embedded system is a text editor.
- You need to write source code. In Embedded System Development Tools, Editor is used to writing code for embedded systems applications.
- It is the editor where you write that code.
- The code is written in programming language either C++ or C.
- There is a standard ASCII text editor that is used to write source code and you save your file as ASCII text file.

# Translate the code by Compiler or Assembler



- Compiler/Assembler is the second tool in your embedded system software development.
- Once you are done with your source code, you need to translate that code into the instructions on the basis of which, the microcontroller will operate.
- The set of instructions in the microcontroller is called as 'Op Codes'.
- Now, you might be thinking what Op Codes are. These bits are decoded and then executed.
- Most of the times, the Op Codes are not written in bits but in hexadecimal numbers.
- And one hexadecimal number means 4 bits.
- Two hexadecimal number will represent 8 bits that mean 1 byte.
- Op Codes are actually bits (0 and 1) that are present in a sequence.
- The compiler is used to for translating the source code into another code called 'hex code'. This code now represents the machine's instruction code.
- One can say that the purpose of the compiler is the conversion of a high-level programming language into a low-level programming language.

## Linkers

- The codes are written into smaller parts for ease.
- The linker is a program that combines the number of codes for execution.
- Linkers are used for linking the codes that are saved in different files into one single final program.
- It also takes much care of allocation of memory of chips so that the different modules saved into a single program do not overlap.

## Libraries

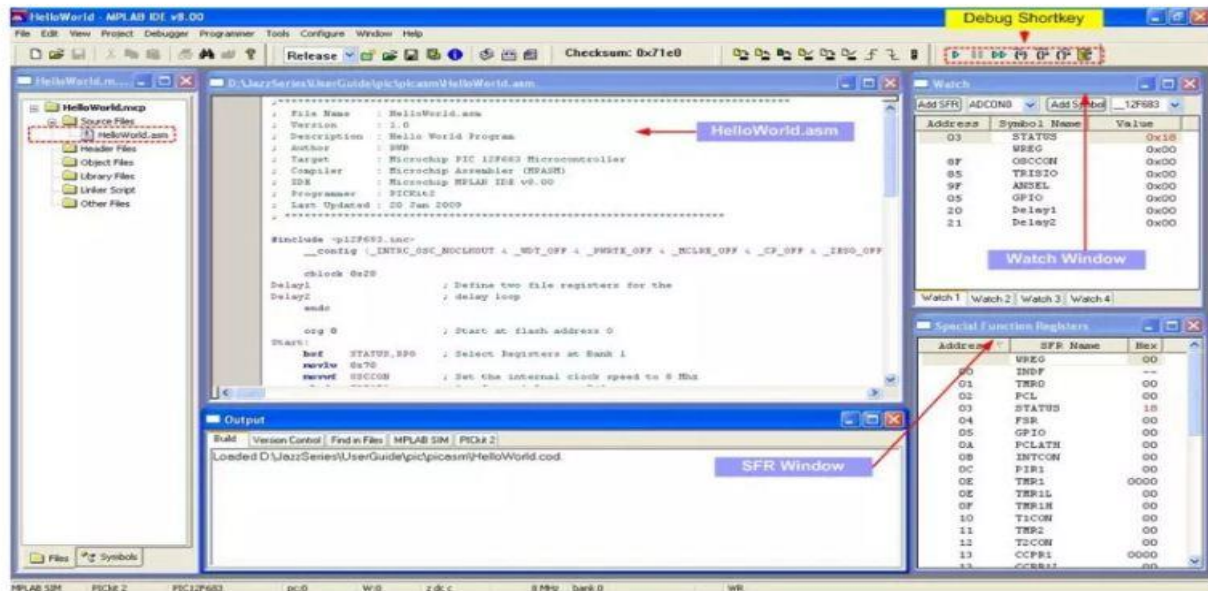
- You can say a library is an already written program that you can use instantly and some specific function is provided by that program. For the software development tools of the embedded system, the library is very significant and appropriate.



- Say for instance, you may download an Arduino microcontroller that is available with different libraries and you can use them in the development of your software for the embedded system.
- Using library you can control LED's and or read sensors like encoders.

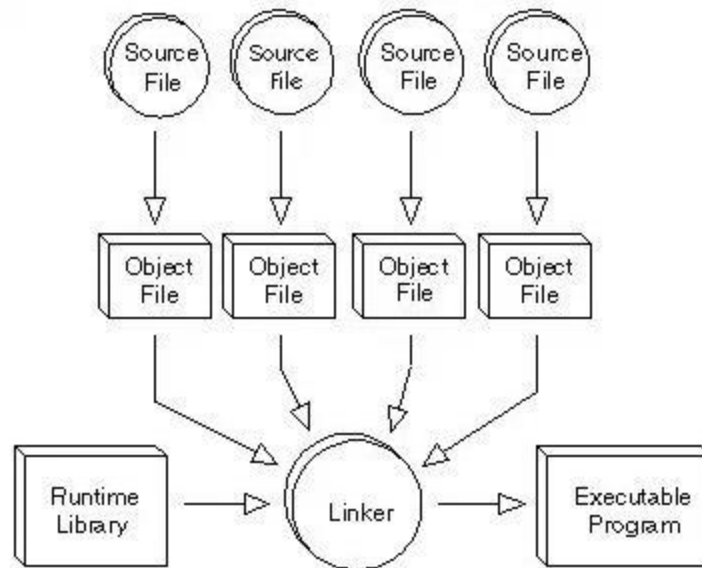
## Debugger

## Debugger



- The name debugger speaks itself. This tool is used for debugging your code.
- The debugger is actually as a tester and is used to test whether your code contains error or not.
- The debugger has a complete look at the code and test if there are any errors or bugs.
- It tests different kinds of errors like any error in your syntax or if there is any runtime error and it tells where the error is actually taking place.
- The place where the error occurs is highlighted by the debugger so that you can easily remove your error by doing some changes.
- So, you get to know how important debugger in the development of software is in embedded systems.

The figure shows the cycle for the development of software.



### Stimulator

- Among all other tools used for development of software for embedded system, there is another tool stimulator.
- The simulator enables you to know how the code that you created actually works in reality.
- You might be able to see the interaction of sensors by changing the input entries from sensors.
- You may analyze what type of function different components performing and what is the effect, created by changing input values.
- After knowing about some basic tools for the software development of the embedded system, you may need to know about a software that we have here in detail.
- Examples of simulators are Proteus which is used for simulation of the microcontroller based project and microprocessor based projects.
- Following video lecture will show you show to use Proteus for microcontrollers simulation to check your program output

### Proteus ISIS as a simulator

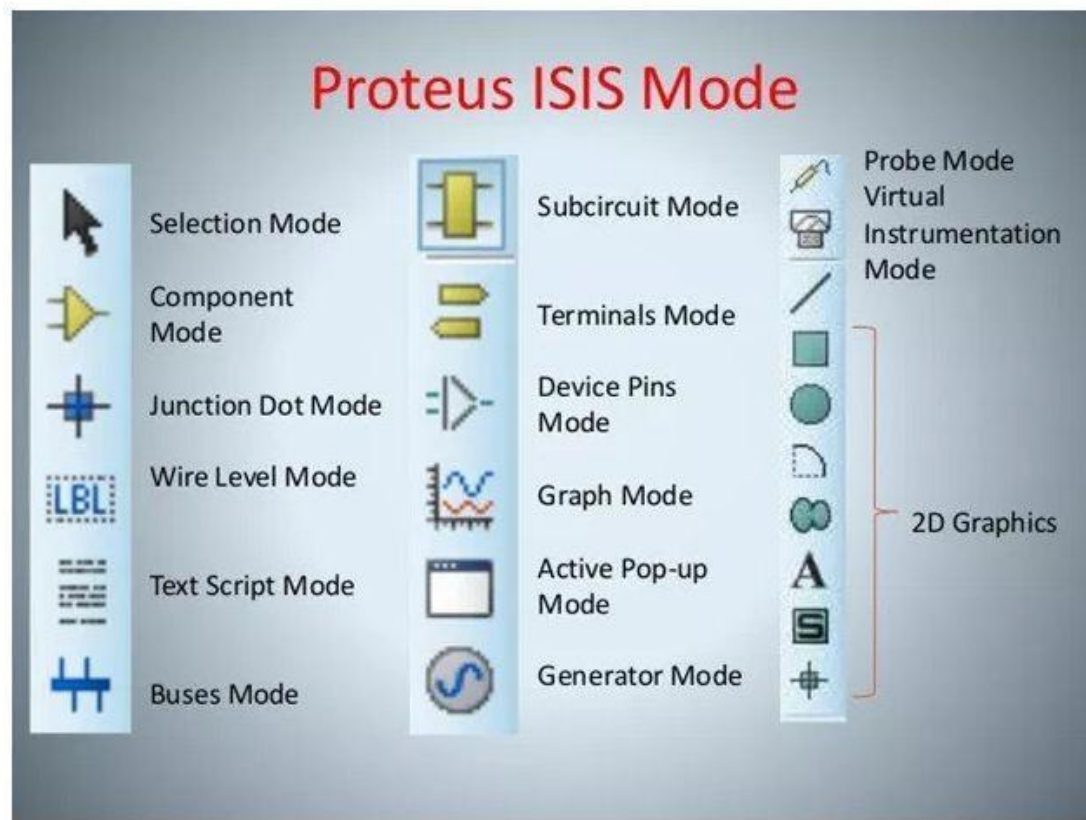
Proteus is software in which we can make an easily schematic capture, PCB, and simulation of a microprocessor. It's a simple but more effective interface that simplifies the task required to be performed. It's more attracted to the user also can say that its user-friendly. It provides a powerful working environment. The user can design the different electronics circuits with all necessary components like simple resistance, power supply, and different microprocessors or microcontroller. This application mostly used in educational institutes because it easy to use and easy to understand the students.

### Proteus Feature

- Easy to use.
- User-Friendly.
- Effective interface.
- Circuit designing and schematic makes easily.

- Provides working environment.

### Microcontroller Simulation in Proteus



The microcontroller simulation in Proteus works by applying “hex” file or another file to the microcontroller. here some important microcontrollers which we simulate in Proteus

- Microchips
- PIC10, PIC12, PIC16, PIC18, PIC24, dsPIC33 Microcontrollers.

- ARM microcontrollers
- MSP microcontrollers
- Microprocessors

### Example of embedded systems development tools

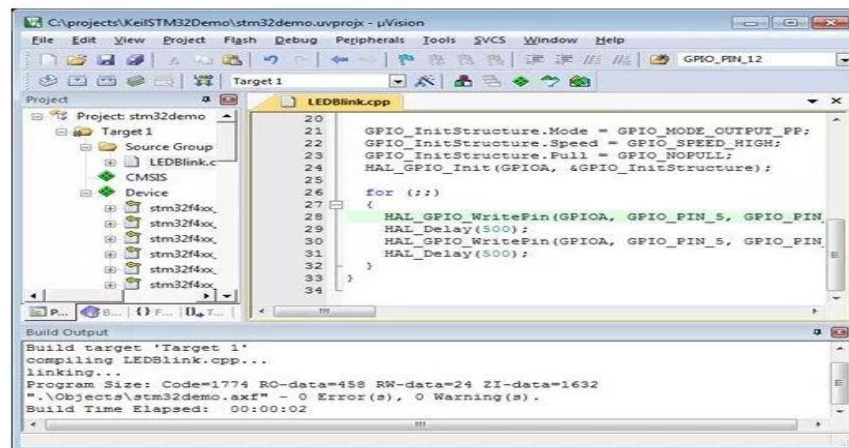
- Integrated Development Environment is an example of software that consists of all the tools for the development of software for the embedded system.
- You need to have all the above-mentioned tools for the development of software for your embedded system.
- Integrated Development Environment Software usually consists of the debugger, compiler, and a code editor.
- Microsoft Visual Studio is an example of this software. This software is used for making many different programs of computer and is also able to support a number of different computer languages.
- There are many Integrated Development Environment is available in the market but your choice is dependent on the type of microcontroller and processor you are using for your development.
- For example, if you are using pic microcontroller, you can use Mikro c for pic and MPLAB for your development. if you are using ARM microcontroller, you can use Keil MDK. if you are working on embedded Linux projects, you have to choose your IDE accordingly.

Other examples of Integrated Development Environment include:

- Xcode
- Eclipse
- Android Studio
- Code Blocks
- Adobe Flash Builder etc.
- BlueJ

## KEIL MDK FOR ARM-BASED MICROCONTROLLERS

### Keil MDK for ARM-based Microcontrollers



Keil is the world's number one developer of embedded Software's. Keil MDK is the concise software for ARM microcontroller. Keil MDK development tools include IDE, Compiler, and Debugger. It contains all the features like creating a project, building it and debugging the project. It also contains very useful and important embedded applications. There are three different Keil development softwares for ARM, which is as follow:

- ARM Keil Microcontroller Tools for Embedded and microcontroller based projects.
- ARM DS Development Tool for SoC, Bare-metal, Kernel, and Applications
- ARM Models Virtual Prototyping for System Simulations.

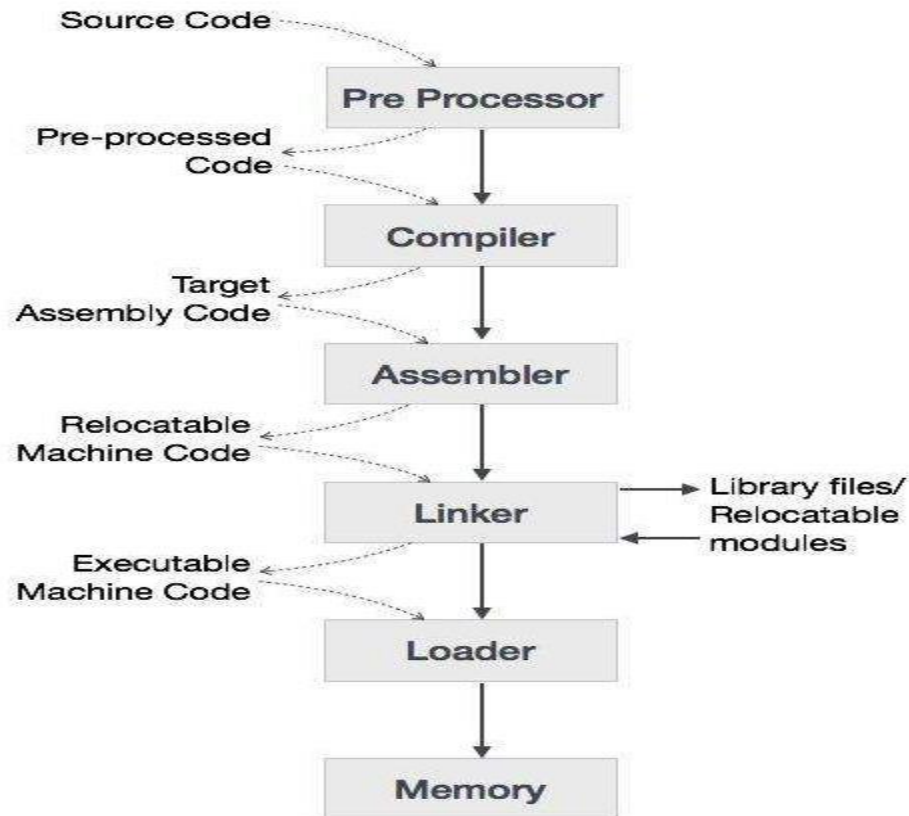
It can control up to 3750 different devices of silicon vendors. It has user-friendly interface and easy to use in any ARM project.

## **TRANSLATION TOOLS**

Computers are a balanced mix of software and hardware. Hardware is just a piece of mechanical device and its functions are being controlled by a compatible software. Hardware understands instructions in the form of electronic charge, which is the counterpart of binary language in software programming. Binary language has only two alphabets, 0 and 1. To instruct, the hardware codes must be written in binary format, which is simply a series of 1s and 0s. It would be a difficult and cumbersome task for computer programmers to write such codes, which is why we have compilers to write such codes.

### **Language Processing System**

We have learnt that any computer system is made of hardware and software. The hardware understands a language, which humans cannot understand. So we write programs in high-level language, which is easier for us to understand and remember. These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine. This is known as Language Processing System.



The high-level language is converted into binary language in various phases. A **compiler** is a program that converts high-level language to assembly language. Similarly, an **assembler** is a program that converts the assembly language to machine-level language.

Let us first understand how a program, using C compiler, is executed on a host machine.

- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program (low-level language).
- An assembler then translates the assembly program into machine code (object).
- A linker tool is used to link all the parts of the program together for execution (executable machine code).
- A loader loads all of them into memory and then the program is executed.

Before diving straight into the concepts of compilers, we should understand a few other tools that work closely with compilers.

### **Preprocessor**

A preprocessor, generally considered as a part of compiler, is a tool that produces input for compilers. It deals with macro-processing, augmentation, file inclusion, language extension, etc.

### **Interpreter**

An interpreter, like a compiler, translates high-level language into low-level machine language. The difference lies in the way they read the source code or input. A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes. In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it. whereas a compiler reads the whole program even if it encounters several errors.

### **Assembler**

An assembler translates assembly language programs into machine code. The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.

### **Linker**

Linker is a computer program that links and merges various object files together in order to make an executable file. All these files might have been compiled by separate assemblers. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

### **Loader**

Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and creates memory space for it. It initializes various registers to initiate execution.

### **Cross-compiler**

A compiler that runs on platform (A) and is capable of generating executable code for platform (B) is called a cross-compiler.

### **Source-to-source Compiler**

A compiler that takes the source code of one programming language and translates it into the source code of another programming language is called a source-to-source compiler.

## **QUALITY ASSURANCE AND TESTING OF THE DESIGN**

The software system needs to be checked for its intended behavior and direction of progress at each development stage to avoid duplication of efforts, time and cost overruns, and to assure completion of the system within stipulated time. The software system needs to be checked for its intended behavior and direction of progress at each development stage to avoid duplication of efforts, time and cost overruns, and to assure completion of the system within stipulated time.

System testing and quality assurance come to aid for checking the system. It includes –

- Product level quality (Testing)
- Process level quality.

Let us go through them briefly –

### **Testing**

Testing is the process or activity that checks the functionality and correctness of software according to specified user requirements in order to improve the quality and reliability of system. It is an expensive, time consuming, and critical approach in system development which requires proper planning of overall testing process.

A successful test is one that finds the errors. It executes the program with explicit intention of finding error, i.e., making the program fail. It is a process of evaluating system with an intention of creating a strong system and mainly focuses on the weak areas of the system or software.

### **Characteristics of System Testing**



System testing begins at the module level and proceeds towards the integration of the entire software system. Different testing techniques are used at different times while testing the system. It is conducted by the developer for small projects and by independent testing groups for large projects.

### Stages of System Testing

The following stages are involved in testing –

#### **Test Strategy**

It is a statement that provides information about the various levels, methods, tools, and techniques used for testing the system. It should satisfy all the needs of an organization.

#### **Test Plan**

It provides a plan for testing the system and verifies that the system under testing fulfils all the design and functional specifications. The test plan provides the following information –

- Objectives of each test phase
- Approaches and tools used for testing
- Responsibilities and time required for each testing activity
- Availability of tools, facilities, and test libraries
- Procedures and standards required for planning and conducting the tests
- Factors responsible for successful completion of testing process

#### **Test Case Design**

- Test cases are used to uncover as many errors as possible in the system.
- A number of test cases are identified for each module of the system to be tested.
- Each test case will specify how the implementation of a particular requirement or design decision is to be tested and the criteria for the success of the test.

- The test cases along with the test plan are documented as a part of a system specification document or in a separate document called **test specification** or **test description**.

### **Test Procedures**

It consists of the steps that should be followed to execute each of the test cases. These procedures are specified in a separate document called test procedure specification. This document also specifies any special requirements and formats for reporting the result of testing.

### **Test Result Documentation**

Test result file contains brief information about the total number of test cases executed, the number of errors, and nature of errors. These results are then assessed against criteria in the test specification to determine the overall outcome of the test.

### Types of Testing

Testing can be of various types and different types of tests are conducted depending on the kind of bugs one seeks to discover –

#### Unit Testing

Also known as Program Testing, it is a type of testing where the analyst tests or focuses on each program or module independently. It is carried out with the intention of executing each statement of the module at least once.

- In unit testing, accuracy of program cannot be assured and it is difficult to conduct testing of various input combination in detail.
- It identifies maximum errors in a program as compared to other testing techniques.

#### Integration Testing

In Integration Testing, the analyst tests multiple module working together. It is used to find discrepancies between the system and its original objective, current specifications, and systems documentation.

- Here the analysts are try to find areas where modules have been designed with different specifications for data length, type, and data element name.
- It verifies that file sizes are adequate and that indices have been built properly.

### Functional Testing

Function testing determines whether the system is functioning correctly according to its specifications and relevant standards documentation. Functional testing typically starts with the implementation of the system, which is very critical for the success of the system.

Functional testing is divided into two categories –

- **Positive Functional Testing** – It involves testing the system with valid inputs to verify that the outputs produced are correct.
- **Negative Functional Testing** – It involves testing the software with invalid inputs and undesired operating conditions.

### Rules for System Testing

To carry out system testing successfully, you need to follow the given rules –

- Testing should be based on the requirements of user.
- Before writing testing scripts, understand the business logic should be understood thoroughly.
- Test plan should be done as soon as possible.
- Testing should be done by the third party.
- It should be performed on static software.
- Testing should be done for valid and invalid input conditions.
- Testing should be reviewed and examined to reduce the costs.
- Both static and dynamic testing should be conducted on the software.
- Documentation of test cases and test results should be done.

## Quality Assurance

It is the review of system or software products and its documentation for assurance that system meets the requirements and specifications.

- Purpose of QA is to provide confidence to the customers by constant delivery of product according to specification.
- Software quality Assurance (SQA) is a techniques that includes procedures and tools applied by the software professionals to ensure that software meet the specified standard for its intended use and performance.
- The main aim of SQA is to provide proper and accurate visibility of software project and its developed product to the administration.
- It reviews and audits the software product and its activities throughout the life cycle of system development.

## Objectives of Quality Assurance

The objectives of conducting quality assurance are as follows –

- To monitor the software development process and the final software developed.
- To ensure whether the software project is implementing the standards and procedures set by the management.
- To notify groups and individuals about the SQA activities and results of these activities.
- To ensure that the issues, which are not solved within the software are addressed by the upper management.
- To identify deficiencies in the product, process, or the standards, and fix them.

## Levels of Quality Assurance

There are several levels of QA and testing that need to be performed in order to certify a software product.

### **Level 1 – Code Walk-through**

At this level, offline software is examined or checked for any violations of the official coding rules. In general, the emphasis is placed on examination of the documentation and level of in-code comments.

### **Level 2 – Compilation and Linking**

At this level, it is checked that the software can compile and link all official platforms and operating systems.

### **Level 3 – Routine Running**

At this level, it is checked that the software can run properly under a variety of conditions such as certain number of events and small and large event sizes etc.

### **Level 4 – Performance test**

At this final level, it is checked that the performance of the software satisfies the previously specified performance level.

## **LABORATORY TOOLS**

Hardware Diagnostic Laboratory Tools Hardware Diagnostic Laboratory Tools • Volt-Ohm meter— useful for checking the power supply voltage at source and voltage levels at chips power input pins, and port pins initial at start and final voltage levels after the software run, checking broken connections, improper ground connections, and burnout resistances and diodes.

Hardware Diagnostic Laboratory Tools Hardware Diagnostic Laboratory Tools • LED test — useful for testing port outputs and when using port conditions 1 or 0 for debugging a particular set of conditions

Hardware Diagnostic Laboratory Tools... • Logic Probe • Oscilloscope • Logic Analyser • Bit Rate meter • ICE • System Monitor Codes

Use of Logic Probe Use of Logic Probe • Simplest hardware test device. • Handheld pen like device with LEDs – Glows green for '1' and red for '0' • Important tool when studying the long port-delay effects (>1s). • Delay program tests the presence of system clock ticks

Uses of Oscilloscope Uses of Oscilloscope • Screen to display two signal voltages as a function of time • Displays analog as well as digital signals as a function of time • Noise detection tool • Mal-function detection of a sudden transitions between '0' and '1' states during a period.

Uses of Logic Analyser Uses of Logic Analyser • A powerful hardware tool for checking multiple lines carrying address, data and control bits, IO buses, ports, peripherals and clock. • Recognizes only discrete voltage conditions, '1' and '0'.

Logic Analyser ... Logic Analyser ... • Collects, stores and tracks multiple signals and bus transactions simultaneously and successively. • Reads multiple input lines (24 or 48) and later displays, using this tool, each transaction on each of these on computer monitor (screen) observed

Logic Analyser ... Logic Analyser ... • Also shows states on the horizontal axis instead of time in its state mode of display. • Displays the logic states of a particular line as a function of '0' and '1' on another line. • Catches intermittent bugs

Logic Analyser ... Logic Analyser ... • Does not help on a program halt due to a bug. • Does not show processor register and memory contents, if the processor uses caches then a bus examination alone may not help.

Logic Analyser Analyser ... • Cannot modify the memory contents and input parameters and study their effects [simulator needed as that helps in studying these effects]

Logic Analyzer two modes of functioning Logic Analyzer two modes of functioning One mode — to show time on x-axis, and logic states of the clock signal, bus signals and other signals on y-axis. Second mode — to give address, data bus and other signal states from a trigger point to examine illegal opcodes, access in protected address space and other states as a function of a reference state.