

MODULE – 4: EMBEDDED SYSTEM DESIGN CONCEPTS

Characteristics of an Embedded Systems

Unlike general purpose computing systems, embedded systems possess certain specific characteristics and these characteristics are unique to each embedded system.

- Some of the important characteristics of an embedded system are:

1. Application and domain specific
2. Reactive and Real Time
3. Operates in harsh environments
4. Distributed
5. Small size and weight
6. Power concerns

1. Application and Domain Specific

- An embedded system is designed for a specific purpose only.
 - It will not do any other task.
 - **Ex.** Air conditioner's embedded control unit, it cannot replace microwave oven...
 - **Ex.** A washing machine can only wash, it cannot cook..
 - Because the embedded control units of microwave oven and air conditioner are specifically designed to perform certain specific tasks.
 - Certain embedded systems are specific to a domain
- Ex.** A hearing aid is an application that belongs to the domain of signal processing and telecom with another control unit designed to serve another domain like consumer electronics.

2. Reactive and Real Time

- Certain embedded systems are designed to react to the events that occur in the nearby environment. These events also occur real-time.
- **Ex.** Flight control systems, Antilock Brake Systems (ABS) etc are examples of Real Time systems
- **Ex.** An air conditioner adjusts its mechanical parts as soon as it gets a signal from its sensors to increase or decrease the temperature when the user operates it using a remote control.
- An embedded system uses Sensors to take inputs and has actuators to bring out the required functionality.

3. Operation in Harsh Environment

- Certain embedded systems are designed to operate in harsh environments like a dusty one or a high temperature zone or an area subject to vibrations and shock or very high temperature of the deserts or very low temperature of the mountains or extreme rains.

- Power supply fluctuations, corrosion and component aging etc are the other factors that need to be taken into consideration while designing an embedded system.
- In general, the embedded systems have to be capable of sustaining the environmental conditions it is designed to operate in.

4. Distributed

- The term distributed means that embedded systems may be a part of a larger system.
- These components are independent of each other but have to work together for the larger system to function properly.
- **Ex.** An Automatic Vending Machine is a typical example for this. The vending machine contains a card reader (for pre-paid vending systems), a vending unit, etc. Each of them are independent embedded units but they work together to perform the overall vending function.
- **Ex.** Automatic Teller Machine (ATM) contains a card reader embedded unit, responsible for reading and validating the user's ATM card, transaction unit for performing transactions, a currency counter for dispatching/vending currency to the authorized person and a printer unit for printing the transaction details.
- They can be visualized as independent embedded systems. But they work together to achieve a common goal.

5. Small Size and Weight

- An embedded system that is compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.
- The product aesthetics (size, shape, weight, style etc) will be one of the deciding factors to choose a product.
- It is convenient to handle a compact device than a bulky product.
- **Ex.** Currently available cell phones. The cell phones that have the maximum features are popular but also their size and weight is an important characteristic.

6. Power Concerns

- It is desirable that the power utilization and heat dissipation of any embedded system be low.
- If more heat is dissipated then additional units like heat sinks or cooling fans need to be added to the circuit.
- **Ex.** The production of high amount of heat demands cooling requirements like cooling fans which in turn occupies additional space and make the system bulky. Nowadays ultra low power components are available in the market.
- Select the design according to the low power components like low dropout regulators, and controllers/processors with power saving modes.
- Also power management is a critical constraint in battery operated application.
- The more the power consumption the less is the battery life.

Quality Attributes of Embedded Systems

- Quality attributes are the non-functional requirements that need to be documented properly in any system design.
- If the quality attributes are more concrete and measurable, it will give a positive impact on the system development process and the end product.
- The various quality attributes that need to be addressed in any embedded system development are broadly classified into two, namely
 - a. Operational Quality Attributes
 - b. Non-Operational Quality Attributes

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.
- Operational attributes are as follows:
 - **Response:** System quickness is measured by response. It gives the user an idea about how fast the system is tracking the input variables.
 - Fast response time is required for most of the embedded systems.
 - **Throughput:** It is related to the efficiency of the system. It is the number of events that take place within a given amount of time.
 - The ability to provide appropriate response and processing times and throughput rates when performing its function.
 - Ex: In card reader, throughput is related to the number of transactions performed in one minute or one second.
 - **Reliability:** Probability of system working correctly provided that it was working at $t=0$. Mean Time Between Failures and Mean Time to Repair are terms used in defining system reliability.
 - When function is critical during the mission time, 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:** It is related to maintenance and support to the end user.
 - It is the parameter concerned with how the system in use can be restored after a failure, while also considering concepts like preventive maintenance and Built In Test (BIT), required maintainer skill level, and support equipment.
 - **Security:** Three pillars of security is Confidentiality, Integrity and Availability.
 - Confidentiality deals with protection data from unauthorized disclosure.
 - Integrity gives protection from unauthorized modification.
 - Availability gives protection from unauthorized user.
 - **Safety:** It deals with the possible damage that can happen to the operating person and environment due to the breakdown of an embedded system or due to the emission of hazardous materials from the embedded products.

Non Operational Attributes

1. **Testability and debug ability:** After designing application, testing is performed for checking error. Testing is performed on hardware and software of embedded system.
 - Hardware testing ensures that all devices work properly. Embedded software i.e firmware testing ensure that the firmware is functioning in expected manner.
 - In debug ability, software debugging is performed to find out the possible bugs. In hardware debugging, behavior of device is checked.
2. **Evolvability:** Embedded product is modified according to new hardware technology and software. It takes advantages of new technology.
3. **Portability:** System independence is measured by portability.
 - Product is portable if it is capable of performing its operation as it is intended to do in various environments irrespective of different processor and or controller and embedded OS.
4. **Time to prototype and market:** It is a time laps between the construction of the product and the time at which product is ready for selling or the time required to develop a system to the point that it can be released and sold to customers.
 - Average time to market constraint is about 8 months. Delays can be costly.
 - The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of commercial embedded product.
 - Product prototyping help in reducing time to market in order to shorten the time to prototype, make use of all possible option like use of reuse, off the self component etc.
5. **Per Unit Cost and Revenue**
 - Cost is a factor which is closely monitored by both end user (those who buy the product) and product manufacturer (those who build the product).
 - Cost is a highly sensitive factor for commercial products.
 - Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.
 - From a designer/product development company perspective the ultimate aim of a product is to generate marginal profit.
 - So the budget and total system cost should be properly balanced to provide a marginal profit.
 - **Unit cost:** The monetary cost of manufacturing each copy of the system, excluding NRE cost.
 - **NRE cost (Non Recurring Cost):** The onetime cost of designing the system.
 - $\text{Total cost} = \text{NRE cost} + \text{Unit cost} * \# \text{ of units.}$
 - The product revenue is understood with the help of Product Life Cycle.

Product Life Cycle (PLC)

- ✓ Product Life Cycle has design and development phase.
 - The product idea generation, prototyping, roadmap definition, actual product design and development are the activities carried out during this phase.
 - During the design and development phase there is only investment and no returns.
 - **Product Introduction stage:** Once the product is ready to sell, it is introduced to the market.
 - During the initial period the sales and revenue will be low.
 - There won't be much competition when the product sales and revenue increases with time
 - **Growth phase:** Product grabs high market share.
 - During the **maturing phase**, the growth and sales will be steady and revenue reaches highest point.
 - At **Product retirement time** there will be a drop in sales volume.
- Fig 3.1 shows the graph of revenue v/s time in months.

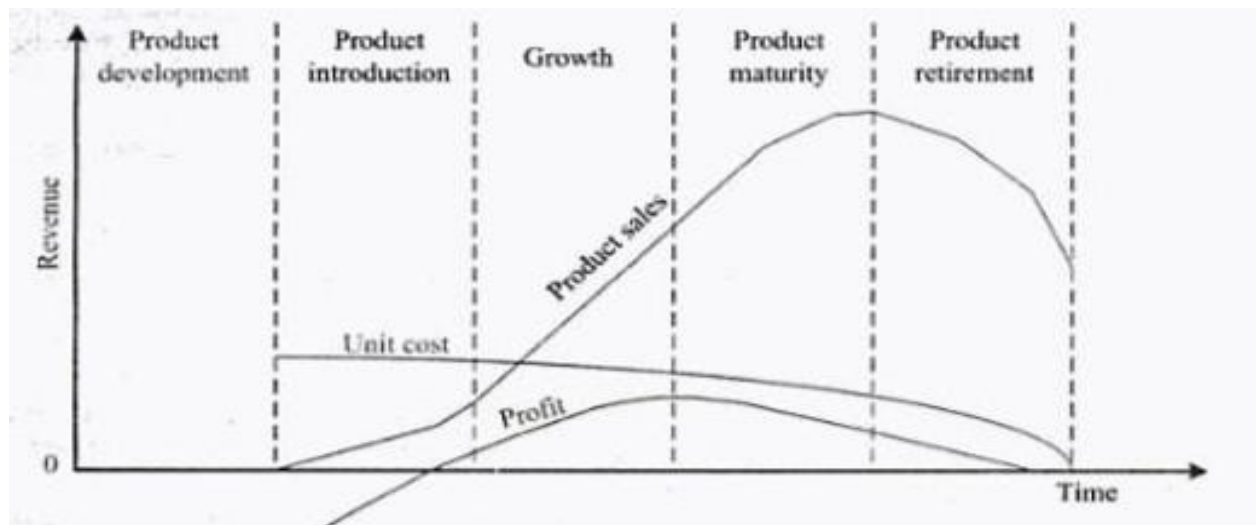


Fig 3.1: Revenue v/s Time graph

Application specific embedded system – Washing machine

- **Washing Machine** is a typical example of an embedded system providing extensive support in home automation applications.
 - It contains status display panel, switches and dials, motor, power supply and control unit, inner water level sensor, solenoid valve, driving motor, water pump, system controller, display panel, sensor and inverter unit.
 - Fig 4.1 shows washing machine.
 - Washing machine supports three functional modes: fully automatic, semi automatic and manual mode.
1. **Fully Automatic Mode:** Once the system is started it performs independently without user interference and after the completion of work it should notify the user about the completion of work.
 2. **Semi Automatic Mode:** In this mode, washing conditions are predefined. Once the predefined mode is started the system performs its job and after completion it informs the user about the completion of work.



Fig 4.1 shows washing machine

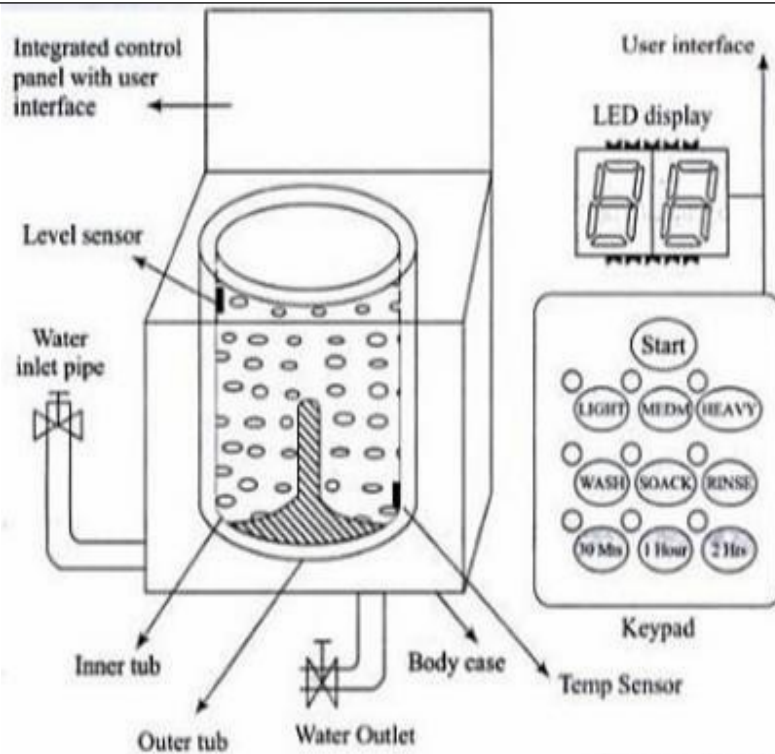


Fig 4.2 shows functional block diagram of washing machine.

3. **Manual Mode:** In this mode, user has to specify which operation she/ he wants to do and has to provide related information to the control system.
 - Fig 4.2 shows functional block diagram of washing machine.
 - An embedded system contains sensors, actuators, control unit and application-specific user interfaces like keyboards, display units etc. Some of them are visible and some of them may be invisible.
 - The **actuator part** of washing machine consists of a motorized agitator, tumble tub, water drawing pump and inlet valve to control the flow of water into the unit.
 - The **sensor part** consists of the water temperature sensor, level sensor etc.
 - **Sensor:** It measures the water level and appropriate amount of soap.
 - **Water level sensor:** It indicates beep sound when water level is low in washing tub.
 - **Door sensor:** It indicates beep sound when all clothes are washed.
 - The **control part** contains a microprocessor/controller based board with interfaces to the sensors and actuators.
 - The sensor data is fed back to the control unit and the control unit generates the necessary actuator outputs.
 - The **control unit** also provides connectivity to user interfaces like keypad for setting the washing time, selecting the type of material to be washed like light, medium, heavy duty etc.
 - **User feedback** is reflected through the display unit and LEDs connected to the control board.
 - **Display panel:** It is a touch panel screen to control all the operations of a machine
 - **Driving motor:** Motor can rotate in two directions either “reverse” or “forward”.
 - Washing machine comes in **two category** front loading as well as top loading.
 1. **Front loading** is the one wherein you are given an opening to put clothes in on the front side.
 - The clothes are tumbled and plunged into the water over and again. This is the ***first phase of washing***.
 - In the ***second phase of washing***, water is pumped out from the tub and the inner tub uses centrifugal force to wring out more water from the clothes by spinning at several hundred ***Rotations Per Minute*** (RPM).
 - This is called a '***Spin phase***'.
 2. **Top loading** is the one wherein you are given an opening to put clothes in on the top.
 - agitator of the machine twists back and forth and pulls the cloth to the bottom of the tub.
 - On reaching the bottom of the tub clothes works their way back up to the top of the tub where the agitator grabs them again and repeats the mechanism.
- **System Controller:** Such component is used to control the motor speed. Motor can move in forward direction as well as reverse direction.
- **Water pump:** The water pump is used to re-circulate water and drain out the dirty water. This pump actually contains two separate pumps inside one.

- The **integrated control panel** consists of a microprocessor/controller based board with I/O interfaces and a control algorithm running in it.
- **Input interface** includes the keyboard which consists of wash type selector namely Wash, Spin and Rinse, cloth type selector namely Light, Medium, Heavy duty and washing time setting etc.
- The **output interface** consists of LED/LCD displays, status indication LED's etc. connected to the I/O bus of the controller.
- It is to be noted that this interface may vary from manufacturer to manufacturer and model to model.
- The **other types of I/O interfaces** which are invisible to the end user are different kinds of sensor interfaces, namely, water temperature sensor, water level sensor etc. and actuator interface including motor control for agitator and tub movement control, inlet water flow control etc.

Working of Washing machine

- Fig 4.3 shows the different parts of washing machine

1) Water inlet control valve: Near the water inlet point of the washing there is water inlet control valve. When you load the clothes in washing machine, this valve gets opened automatically and it closes automatically depending on the total quantity of the water required. The water control valve is actually the solenoid valve.

2) Water pump: The water pump circulates water through the washing machine. It works in two directions, re-circulating the water during wash cycle and draining the water during the spin cycle.

3) Tub: There are two types of tubs in the washing machine: inner and outer. The clothes are loaded in the inner tub, where the clothes are washed, rinsed and dried. The inner tub has small holes for draining the water. The external tub covers the inner tub and supports it during various cycles of clothes washing.

4) Agitator or rotating disc: The agitator is located inside the tub of the washing machine. It is the important part of the washing machine that actually performs the cleaning operation of the clothes. During the wash cycle the agitator rotates continuously and produces strong rotating currents within the water due to which the clothes also rotate inside the tub. The rotation of the clothes within water containing the detergent enables the removal of the dirt particles from the fabric of the clothes. Thus the agitator produces most important function of rubbing the clothes with each other as well as with water.

- In some washing machines, instead of the long agitator, there is a disc that contains blades on its upper side. The rotation of the disc and the blades produce strong currents within the water and the rubbing of clothes that helps in removing the dirt from clothes.

5) Motor of the washing machine: The motor is coupled to the agitator or the disc and produces its rotator motion. These are multispeed motors, whose speed can be changed as per the

requirement. In the fully automatic washing machine the speed of the motor i.e. the agitator changes automatically as per the load on the washing machine.

6) Timer: The timer helps setting the wash time for the clothes manually. In the automatic mode the time is set automatically depending upon the number of clothes inside the washing machine.

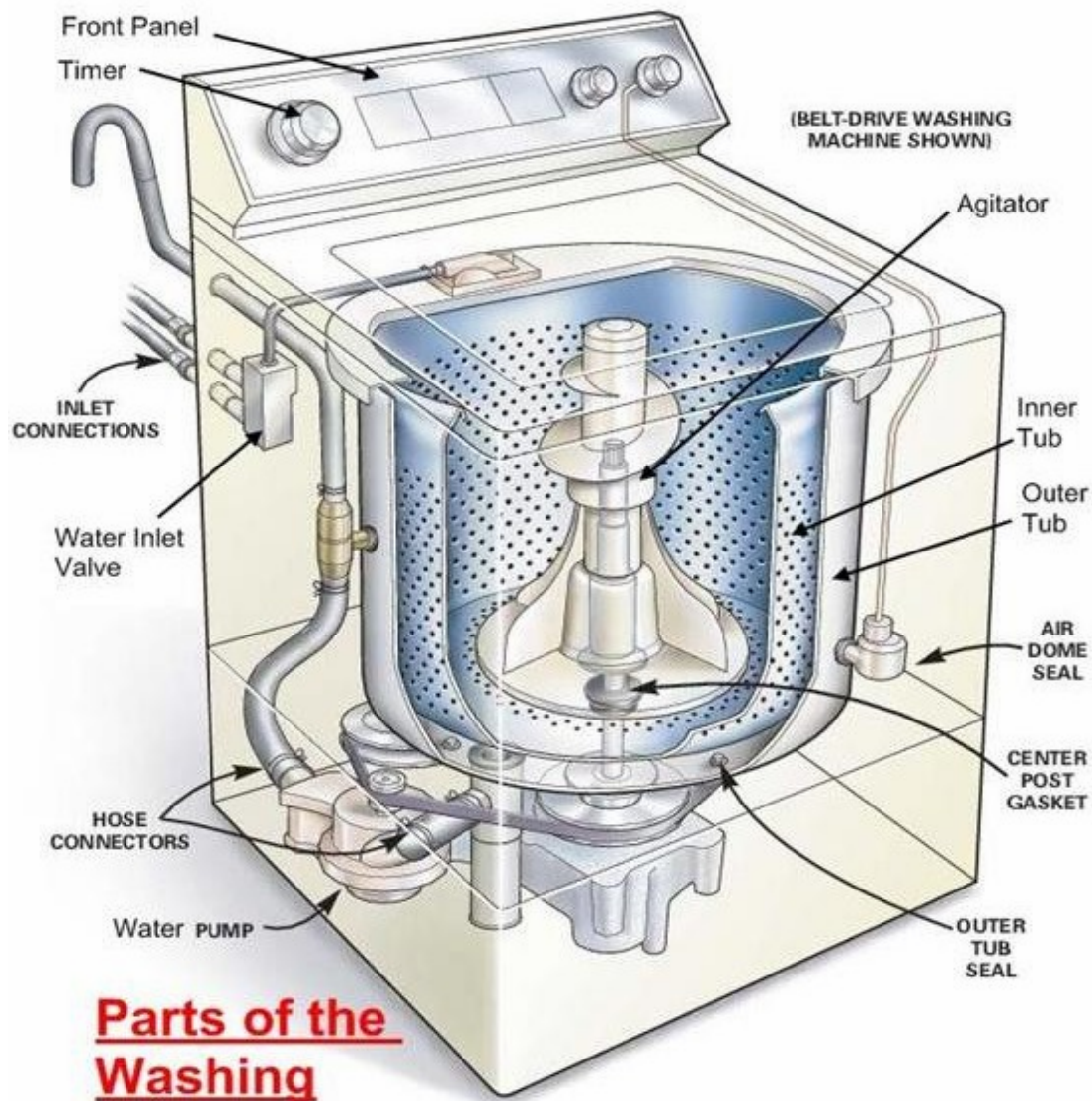


Fig 4.3: Different parts of washing machine

7) Printed circuit board (PCB): The PCB comprises of the various electronic components and circuits, which are programmed to perform in unique ways depending on the load conditions (the condition and the amount of clothes loaded in the washing machine). They are sort of artificial intelligence devices that sense the various external conditions and take the decisions accordingly. These are also called as fuzzy logic systems. Thus the PCB will calculate the total weight of the clothes, and find out the quantity of water and detergent required, and the total time required for washing the clothes. Then they will decide the time required for washing and rinsing.

8) Drain pipe: The drain pipe enables removing the dirty water from the washing that has been used for the washing purpose.

Automotive-Domain-Specific Examples of Embedded System

- The major application domains of embedded systems are consumer, industrial, automotive, telecom etc. of which telecom and automotive industry holds a big market share.

Inner Workings of Automotive Embedded Systems

- Automotive embedded systems are the one where electronics take control over the mechanical systems.
- The presence of automotive embedded system in a vehicle varies from simple mirror and wiper controls to complex air bag controller and Antilock Brake Systems (ABS).
- Automotive embedded systems are normally built around microcontrollers or DSPs or a hybrid of the two and are generally known as Electronic Control Units (ECUs).
- Fig 4.4 shows the embedded system in automotive domain.

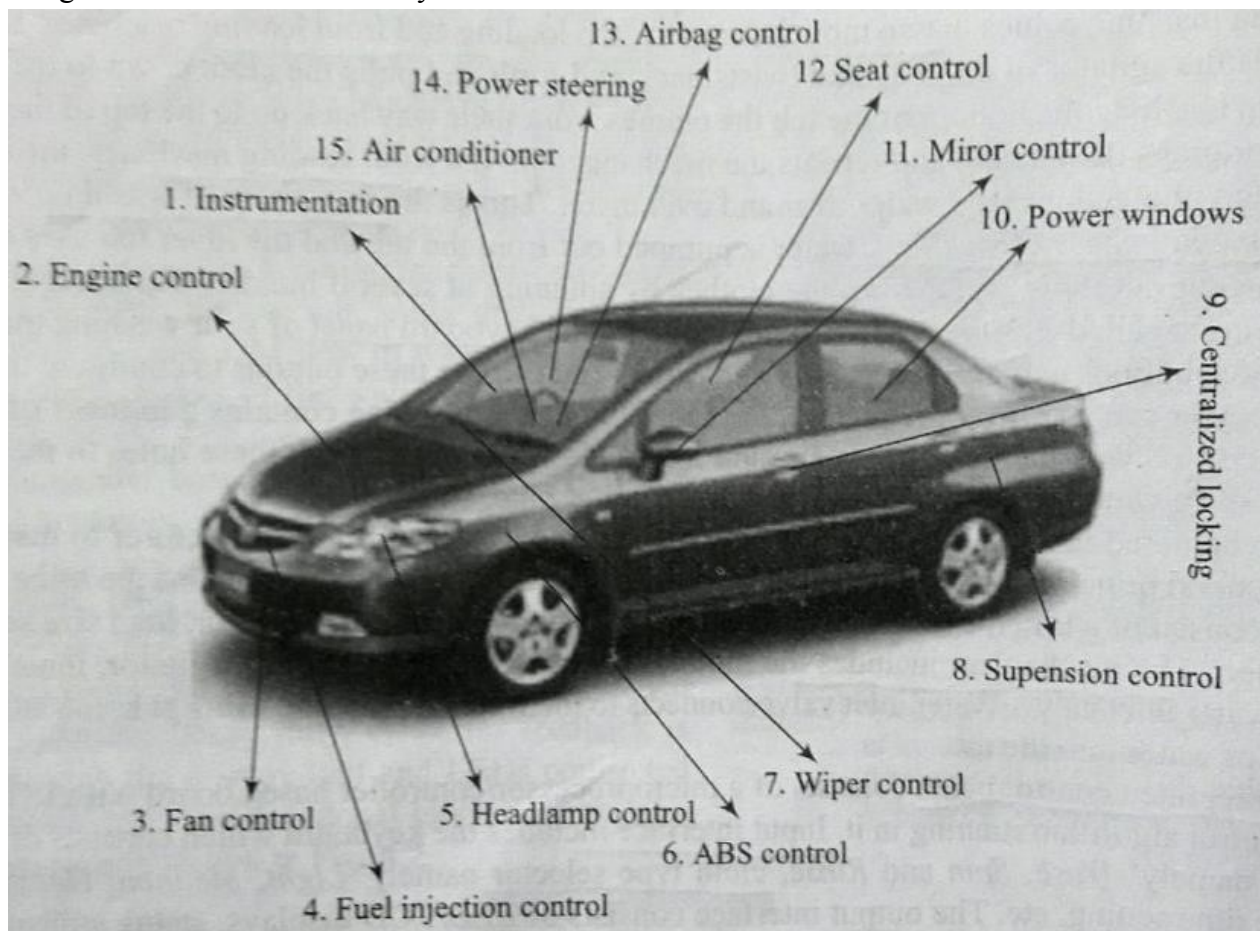


Fig 4.4 shows the embedded system in automotive domain.

NOTE: The first embedded system used in automotive application was the microprocessor based fuel injection system introduced by Volkswagen 1600 in 1968.

- The various types of Electronic Control Units (ECUs) used in the automotive embedded industry can be broadly *classified into two*-High speed embedded control units and Low speed embedded control units.
- **High speed Electronic Control Units (HECUs)** : High speed electronic control units (HECUs) are deployed in critical control units requiring fast response.

Ex: Fuel injection systems, antilock brake systems, Engine control, Steering controls, Transmission control units and Central Control Unit etc.

- **Low speed Electronic Control Units (LECUs)** : Low speed electronic control units are deployed in applications where response time is not so critical. They are generally built around low cost microprocessors/microcontrollers and Digital Signal Processors.

Ex: Audio controllers, Wiper Controllers, Seat Control System, Head and Tail lamp, passenger and driver door locks, door glass controls etc

Automotive Communication Buses

- Automotive applications use serial buses for communication.
- Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented System Transport (MOST) bus, etc. are the important automotive communication buses.

➤ **Controller Area Network (CAN)**

- The CAN bus was originally proposed by Robert Bosch, pioneer in the Automotive embedded solution providers.
- It supports medium speed (ISO11519-class B with data rates up to 125 Kbps) and high speed (ISO11898 class C with data rates up to 1 Mbps) data transfer.
- CAN is an event-driven protocol interface with support for error handling in data transmission.
- It is generally employed in safety system like airbag control, power train systems like engine control and Antilock Brake System (ABS) and navigation systems like GPS.

➤ **Local Interconnect Network (LIN)**

- It is a single master multiple slave (up to 16 independent slave nodes) communication interface.
- LIN is a low speed, single wire communication interface with support for data rates up to 20 kbps and is used for sensor/actuator interfacing.
- LIN bus follows the master communication triggering technique to eliminate the possible bus arbitration problem that can occur by the simultaneous talking of different slave nodes connected to a single interface bus.

- LIN bus is employed in applications like mirror controls, fan controls, seat positioning controls, window controls, and position controls where response time is not a critical issue.

➤ **The Media Oriented System Transport (MOST)**

- This bus is targeted for automotive audio video equipment interfacing.
- MOST bus is a multimedia fiber-optic point-to point network implemented in a star, ring or daisy chained topology over optical fibers cables.
- The MOST bus specifications define the physical (electrical and optical parameters) layer as well as the application layer, network layer, and media access control.
- MOST bus is an optical fibre cable connected between the Electrical Optical Converter (EOC) and Optical Electrical Converter (OEC), which would translate into the optical cable MOST bus.

Key Players of the Automotive Embedded Market

- The key players in Automotive Embedded Market can be Silicon providers, Tools and platform providers, Solution providers.

➤ **Silicon Providers**

- Responsible for providing the necessary chips which are used in the control application development.
- The chip maybe a standard product like microcontroller or DSP or ADC/DAG chips.
- Some applications may require specific chips and they are manufactured as Application Specific Integrated Chip (ASIC).
- The leading silicon providers in the automotive industry are:
 - **Analog Device** (www.analog.com): Provider of world class digital signal processing chips, precision analog microcontrollers, programmable inclinometer/accelerometer, LED drivers etc. for automotive signal processing applications, driver assistance system, audio system, GPS/Navigation system etc.
 - **Xilinx** (www.xilinx.com): Supplier of high performance FPGAs, CPLDs and automotive specific IP cores for GPS navigation systems, driver information systems, distance control, collision avoidance, rear seat entertainment, adaptive cruise control, voice receptionist etc.
 - **Atmel** (www.atmel.com): Supplier of cost-effective high-density Flash controllers and memories. Atmel provides a series of high performance microcontrollers, namely, ARM1, ARM2, and 80C51. A wide range of Application Specific Standard Products (ASSPs) for chassis, body electronics, security, safety and car infotainment and automotive networking products for CAN, LIN and FlexRay are also supplied by Atmel.

- **Maxim/Dallas** (www.maxim-ic.com): Supplier of world class analog, digital and mixed signal products (Microcontrollers, ADC/DAC, amplifiers, comparators, regulators, etc), RF components, etc. for all kinds of automotive solutions.
- **NXP semiconductor** (www.nxp.com): Supplier of 8/16/32 Flash microcontrollers.
- **Renesas** (www.renesas.com): Provider of high speed microcontrollers and Large Scale Integration (LSI) technology for car navigation systems accommodating three transfer speeds: high, medium and low.
- **Texas Instruments** (www.ti.com): Supplier of microcontrollers, digital signal processors and automotive communication control chips for Local Inter Connect (LIN) bus products.
- **Fujitsu** (www.fmal.fujitsu.com): Supplier of fingerprint sensors for security applications, graphic display controller for instrumentation application, AGPS/GPS for vehicle navigation system and different types of microcontrollers for automotive control applications.
- **Infineon** (www.infineon.com): Supplier of high performance microcontrollers and customized application specific chips.
- **NEC** (www.mec.co.jp): Provider of high performance microcontrollers.

➤ ***Tools and Platform Providers***

- They are manufacturers and suppliers of various kinds of development tools and Real Time Embedded Operating Systems for developing and debugging different control unit related applications.
- Tools fall into two categories
 - a. Embedded software application development tools
 - b. Embedded hardware development tools.
 - Sometimes the silicon suppliers provide the development suite for application development using their chip.
 - Some third party suppliers may also provide development kits and libraries.
 - Some of the leading suppliers of tools and platforms in automotive embedded applications are listed below.
- **ENE A** (www.cnea.com): ENEA Embedded Technology is the developer of the OSE Real-Time operating system.
 - The OSE RTOS supports both CPU and DSP and has also been specially developed to support multi-core and fault-tolerant system development.
- **The Math Works** (www.mathworks.com): It is the world's leading developer and supplier of technical software.

- It offers a wide range of tools, consultancy and training for numeric computation, visualization, modeling and simulation across many different industries.
- Math Work's breakthrough product is MATLAB-a high-level programming language and environment for technical computation and numerical analysis.
- Together MATLAB, SIMULINK, State flow and Real-Time Workshop provide top quality tools for data analysis, test & measurement, application development and deployment, image processing and development of dynamic and reactive systems for DSP and control applications.
- **Keil Software** (www.keil.com): The Integrated Development Environment Keil Micro vision from Keil software is a powerful embedded software design tool for 8051 & C166 family of microcontrollers.
- **Lauterbach** (<http://www.lauterbach.com/>): It is the world's number one supplier of debug tools, providing support for processors from multiple silicon vendors in the automotive market.
- **ARTiSAN** (www.artisansw.com): Is the leading supplier of collaborative modelling tools for requirement analysis, specification, design and development of complex applications.
- **Microsoft** (www.microsoft.com): It is a platform provider for automotive embedded applications.
 - WindowsCE is a powerful RTOS platform for automotive applications.
 - Automotive features are included in the new WinCE Version for providing support for automotive application developers.

➤ ***Solution Providers***

- Solution providers supply OEM and complete solution for automotive applications making use of the chips, platforms and different development tools.
- The major players of this domain are listed below.
- **Bosch Automotive** (www.boschindia.com): Bosch is providing complete automotive solution ranging from body electronics, diesel engine control, gasoline engine control, power train systems, safety systems, in-car navigation systems and infotainment systems.
- **DENSO Automotive** (www.globaldensoproducs.com): Denso is an Original Equipment Manufacturer (OEM) and solution provider for engine management, climate control, body electronics, driving control & safety, hybrid vehicles, embedded infotainment and communications.
- **Infosys Technologies** (WWW.infosys.com): Infosys is a solution provider for automotive embedded hardware and software. Infosys provides the competitive edge in integrating technology change through cost effective solutions.
- **Delphi** (www.delphi.com): Delphi is the complete solution provider for engine control, safety, infotainment, etc., and OEM for spark plugs, bearings etc

Hardware Software Co-Design and Program

Traditional embedded system development approach

- The hardware software partitioning is done at an early stage.
- Engineers from the software group take care of the software architecture development and implementation
- Engineers from the hardware group are responsible for building the hardware required for the product.
- There is less interaction between the two teams and the development happens either serially or parallelly.
- Once the hardware and software are ready, the integration is performed.
- The increasing competition in the commercial market and need for reduced time-to-market the product calls for a novel approach for embedded system design in which the hardware and software are co-developed instead of independently developing both..

Hardware and software co design

- During the co-design process, the product requirements captured from the customer are converted into system level needs or processing requirements.
- At this point of time it is not segregated as either hardware requirement or software requirement, instead it is specified as functional requirement.
- The system level processing requirements are then transferred into functions which can be simulated and verified against performance and functionality.
- The Architecture design follows the system design.
- The partition of system level processing requirements into hardware and software takes place during the architecture design phase.
- Each system level processing requirement is mapped as either hardware and/or software requirement.
- The partitioning is performed based on the hardware-software trade-offs.
- The architectural design results in the detailed behavioral description of the hardware requirement and the definition of the software required for the hardware.
- The processing requirement behaviour is usually captured using computational models and ultimately the models representing the software processing requirements are translated into firmware implementation using programming languages.

Fundamental issues in hardware software co-design

- The hardware software co-design is a problem statement and when we try to solve this problem statement in real life we may come across multiple issues in the design.

a. Selecting the model

- In hardware software co-design, models are used for capturing and describing the system characteristics.
- Model is a formal system consisting of objects and composition rules.
- It is hard to make a decision on which model should be followed in a particular system design.
- Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design, the reason being, the objective varies with each phase.
- **Ex:** at the specification stage, only the functionality of the system is in focus and not the implementation information. When the design moves to the implementation aspect, the information about the system components is revealed and the designer has to switch to a model capable of capturing the system's structure.

b. Selecting the Architecture

- A model only captures the system characteristics and does not provide information on 'how the system can be manufactured?'
- **Ex:** architecture specifies how a system is going to implement in terms of the number and types of different components and the interconnection among them.
- Commonly used architectures in system design - Controller architecture, Data path Architecture, Complex Instruction Set Computing (CISC), Reduce Instruction Set Computing (RISC), Very Long Instruction Word Computing (VLIW), Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD), etc.
- Some of them fall into Application Specific Architecture Class (like controller architecture), while others fall into either general purpose architecture class (CISC, RISC etc.) or Parallel processing class (like VLIW, SIMD, MIMD etc.).

1. Controller Architecture:- Implements the Finite State Machine model using a state register and two combinational circuits.

- ✓ The state register holds the present state and the combinational circuits implement the logic for next state and output.

2. Data path architecture:- Is best suited for implementing the data flow graph model where the output is generated as a result of a set of predefined computations on the input data.

- ✓ A data path represents a channel between the input and output and in data path architecture.
- ✓ Data path may contain registers, counters, register tiles, memories and ports along with high speed arithmetic units.
- ✓ Ports connect the data path to multiple buses.
- ✓ Most of the time the arithmetic units are connected in parallel with pipelining support for bringing high performance.

3. The Finite State Machine Data path (FSMD) architecture:- combines the controller architecture with data path architecture.

- ✓ It implements a controller with data path.
- ✓ The controller generates the control input whereas the data path processes the data.
- ✓ The data path contains two types of I/O ports
 - i) One acts as the control port for receiving/sending the control signals from/to the controller unit
 - ii) Second I/O port interfaces the data path with 'external world for data input and data output'.
- ✓ Normally the data path is implemented in a chip and the I/O pins of the chip acts as the data input output ports for the chip resident data path.

4. The Complex Instruction Set Computing (CISC) architecture:- uses an instruction set representing complex operations.

- ✓ It is possible for a CISC instruction set to perform a large complex operation with a single instruction.
- ✓ The use of Single complex instruction in place of multiple simple instructions greatly reduces the program memory access and program memory size requirement.
- ✓ It requires additional silicon for implementing microcode decoder for decoding the CISC instruction.
- ✓ The data path for the CISC processor is complex.
- ✓ On the other hand, Reduced Instruction Set Computing (RISC) architecture uses instruction Set representing simple operations and it requires the execution of multiple RISC instructions to perform a complex operation.
- ✓ The data path of RISC architecture contains a large register file for storing the operands and output.
- ✓ RISC instruction set is designed to operate on registers.
- ✓ RISC architecture supports extensive pipelining.

5. The Very Long Instruction Word (VLIW) architecture:- implements multiple functional units (ALUS, multipliers, etc.) in the data path.

- ✓ The VLIW instruction packages one standard instruction per functional unit of the data path.

6. Parallel processing architecture:- implements multiple concurrent Processing Elements (PEs)

- ✓ Each processing element may associate a data path containing register and local memory.
- ✓ Ex: Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD).

i. SIMD architecture:- a single instruction is executed in parallel with the help of the Processing Elements.

- The scheduling of the instruction execution and controlling of each PE is performed through a single controller.
- The SIMD architecture forms the basis of re-configurable processor.

ii. MIMD architecture:-The processing elements execute different instructions at a given point of time.

- The MIMD architecture forms the basis of multiprocessor systems.
- The PBS in a multiprocessor system communicates through mechanisms like shared memory and message passing.

c. Selecting the Language

- A programming language captures a 'Computational Model' and maps it into architecture.
- There is no hard and fast rule to specify this language should be used for capturing this mode.
- A model can be captured using multiple programming
- Software implementation languages : C, C++, C#, Java.
- Hardware implementation languages: VHDL, System C, Verilog.
- Single language can be used for capturing a variety of models. Certain languages are good in capturing certain computational model.
- **Ex:** C++ is a good candidate for capturing an object oriented model.
- **The only pre-requisite in selecting a programming language is that the language should capture the model easily.**

d. Partitioning System Requirements into hardware and software

- It is possible to implement the system requirements in either hardware or software (firmware).
- It is a tough decision making task to figure out which one to opt.
- Various hardware software trade-offs are used for making a decision on the hardware-software partitioning.

Computational models in embedded design

- The commonly used computational models in embedded system design are Data Flow Graph (DFG) model, State Machine model, Concurrent Process model, Sequential Program model, Object Oriented model.

Data Flow Graph/Diagram (DFG) Model

- It's the model that translates the data processing requirements into a data flow graph.
- A DFG model translates the program as a single sequential process execution.
- It is a data driven model in which the program execution is determined by data.
- This model emphasis on the data and operations on the data which transforms the input data to output data.

- Data Flow Graph (DFG) is a visual model in which the operation on the data (process) is represented using a block (circle) and data flow is represented using arrows.
- In DFG notation, An inward arrow to the process (circle) represents input data
- An Outward arrow from the process (circle) represents output data.
- The computationally intensive embedded applications and data driven are modeled using DFG model.
- Ex: DSP applications using DFG model.
- Fig 7.1 represents DFG for the equation: $x = a+b$ and $y=x-c$

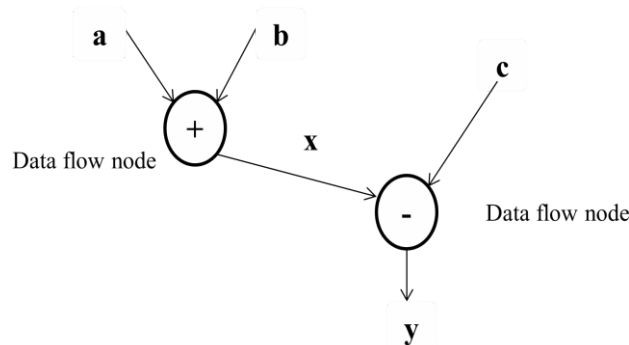


Fig 7.1: DFG for the equation: $x = a+b$ and $y=x-c$

- **Data path** : data flow path from input to output.
- **Acyclic DFG**: DFG which does not contain multiple values for the input variable and multiple output values for a given set of inputs.
- **Non cyclic input**: Ex: inputs, events etc

Control Data Flow Graph/Diagram (CDFG)

- The DFG model is a data driven model in which the execution is controlled by data and it doesn't involve any control operations (conditionals).
- The Control DFG (CDFG) model is used for modeling applications involving conditional program execution.
- CDFG models contains both data and control operations.
- The CDFG uses Data Flow Graph (DFG) as element and conditional (constructs) as decision makers.
- CDFG contains both data flow nodes and decision nodes, whereas DFG contains only data flow nodes.
- Implementation of the CDFG for the following requirement - if flag=1, $x=a +b$; else $y=a-b$ is as shown in Fig 7.2.
- This requirement contains a decision making process.

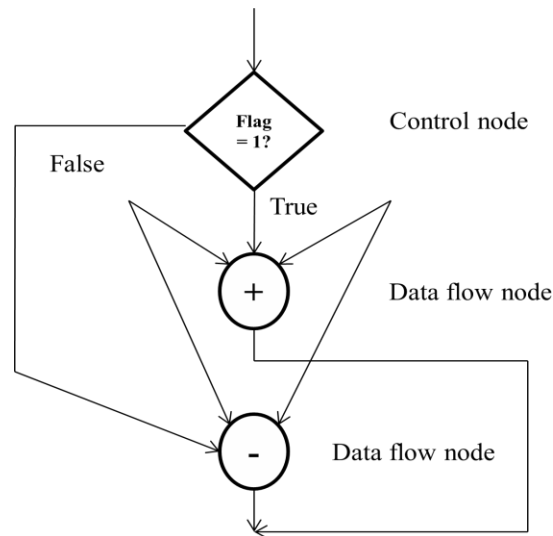


Fig 7.2: Control Data Flow Graph (CDFG) Model for equation
if flag=1, $x=a +b$; else $y=a-b$

- The control node is represented by a 'Diamond' block which is the decision making element in formal flow chart based design.
- CDFG translates the requirement, which is modeled to a concurrent process model.
 - The decision on which process is to be executed is determined by the control node.
 - **Ex:** for modeling the embedded application using CDFG is the capturing and saving of the image to a format set by the user in a digital still camera where everything is data driven
 - The analog signal from CCD sensor is converted to Digital Signal.
 - The task stores the data from ADC to a frame buffer.
 - Media processor performs various operations like, auto correction, white balance adjusting, etc.
 - The decision on, in which format the image is stored (formats like JPEG, TIFF, BMP etc.) is controlled by the camera settings configured by the user.

State Machine Model Space

- It is used for modeling reactive or event driven embedded systems whose processing behavior are dependent on state transitions.
- State machine model describes the system with states, events, actions and transitions.
- **State:** representation of a current situation.
- **Event:** It is an input to the state. And acts as stimuli for state transition.
- **Transition:** is the movement from one state to another.
- **Action:** is an activity to be performed by the state machine.
- In FSM model, the numbers of states are finite. (defines all possible states)
- **Ex:** Design of an embedded system for driver/passenger 'seat belt warning' using FSM model is as shown in Fig 7.3.
- System requirements are:

1. When the vehicle ignition is turned ON and the seat belt is not fastened within 10 sec of ignition ON, the system generates an alarm signal for 5 seconds.
2. The Alarm is turned off when the alarm time (5 sec) expires or if the driver/passenger fastens the belt or if the ignition switch is turned off, whichever happens first.
 - States: Alarm OFF, Waiting, Alarm ON
 - Events: Ignition key ON, Ignition key OFF, Alarm OFF, Timer Expire, Alarm Time Expire, Seat belt ON.
 - Wait state is implemented by a timer.
 - The 'ignition key ON' event triggers the 10 second timer and transitions the state to 'waiting'.
 - If a 'seat belt ON' or 'Ignition key OFF' event occurs during the wait state, the state transitions into 'Alarm Off'.
 - When the wait timer expires in the 'waiting state'.
 - The 'Alarm ON' state continues until a 'Seat belt ON' or 'Ignition key OFF' event or 'Alarm Time Expire' event, whichever occurs first.
 - The occurrence of any of these events transitions the state to 'Alarm off'.
 - The wait state is implemented using a timer.
 - The timer also has certain set of states and events for state transitions.

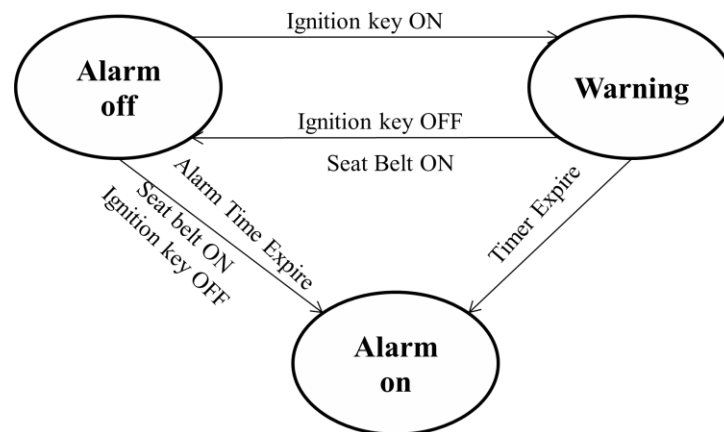


Fig 7.3: FSM Model for Automatic seat belt warning system

- Timer states: ready, running, idle.
 - Idle state: Normal condition, when the timer is not running.
 - Ready state: When the timer is loaded with the count corresponding to the required time delay.
- Timer remains in ready state until a 'start timer' event occurs.
- Running state: Upon receiving start timer event, the timer moves from ready state to running state.
- Timer remains in running state until a 'stop timer' event occurs.
- The timer state changes to idle from running on receiving a stop timer or timer expire event.
 - Fig 7.4 shows the timer model for the above application.

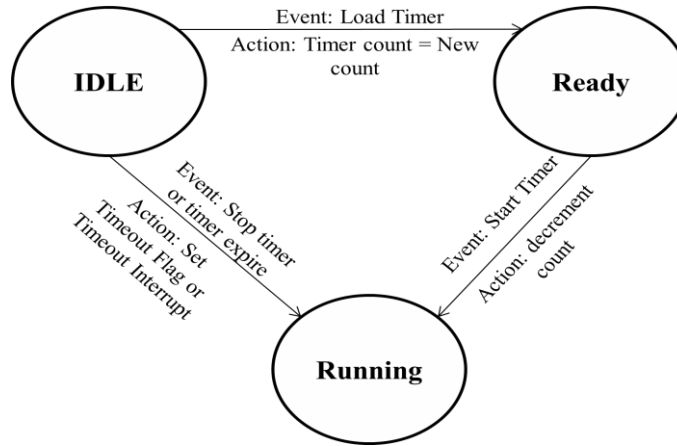


Fig 7.4: FSM Model for Timer

➤ **Example 1**

Problem: Design an automatic tea/coffee vending machine based on FSM model for the following requirements.

The tea/coffee vending is initiated by user inserting a 5 rupee coin. After inserting the coin, the user can either select ‘coffee’ or ‘tea’ or press ‘cancel’ to cancel the order and take back the coin.

Solution: Fig 7.5 shows the FSM model for automatic tea/coffee vending machine.

- Four states: Wait for coin, wait for user input, dispense tea, dispense coffee.
- The ‘coin insertion’ event makes transition to ‘wait for user input’.

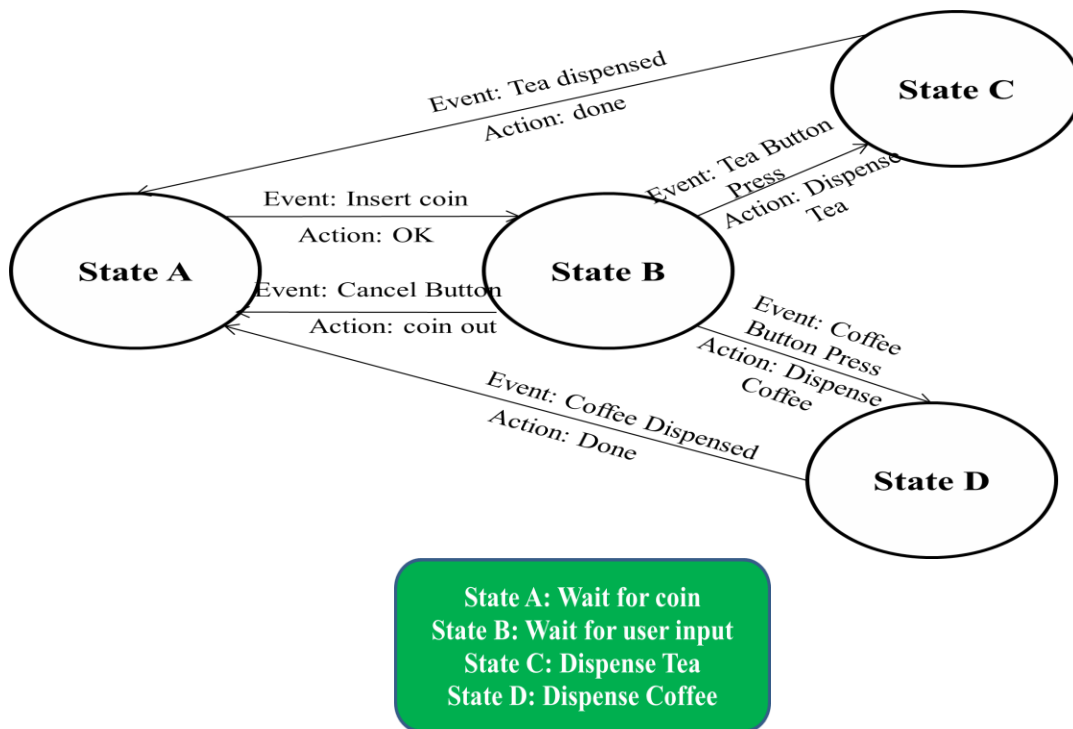


Fig 7.5: FSM Model for Automatic Tea/Coffee Vending machine

- System stays in this state until a user input is received from the button ‘cancel’, ‘tea’ or ‘coffee’.
- If the event triggered in ‘wait state’ is ‘cancel’ button press, the coin is pushed out and the state transitions to ‘wait for coin’.
- If the event received in ‘wait state’ is either ‘tea’ button press or ‘coffee’ button press, the state changes to ‘dispense tea’ or ‘dispense coffee’ respectively.
- Once the tea/coffee vending is over, the respective state transitions back to the ‘wait for coin state’.

Modifications: Adding time out for wait state and capturing another events like ‘water not available’, ‘tea/coffee mix not available’ and changing the state to an ‘error state’ can be added to enhance this design.

➤ Example 2

Problem: Design a coin operated public telephone unit based on FSM model for the following requirements.

1. The calling process is initiated by lifting the receiver (off hook) of the telephone unit.
2. After lifting the phone the user needs to insert a 1 rupee coin to make the call.
3. If the line is busy, the coin is returned on placing the receiver back on the hook (on hook).
4. If the line is through, the user is allowed to talk till 6 seconds and at the end of 45th second, prompt for inserting another 1 rupee coin for continuing the call is initiated.
5. If the user does not insert another 1 rupee coin, the call is terminated on completing the 60 seconds time slot.
6. The system is ready to accept new call request when the receiver is placed back on the hook (on hook).
7. The system goes to the ‘out of order’ state when there is a line fault.

Solution: Fig 7.6 shows the FSM Model for Coin operated telephone system

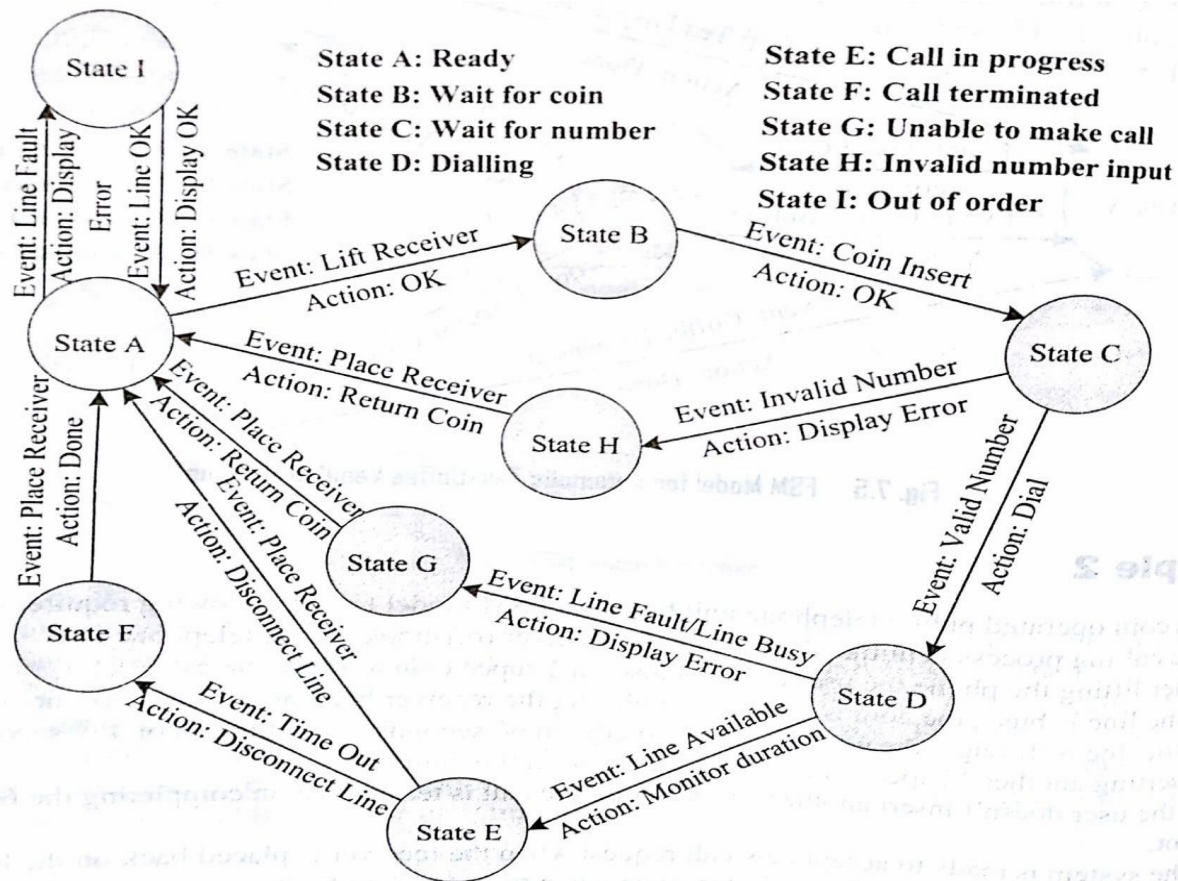


Fig 7.6: FSM Model for Coin operated telephone system

Sequential program model

- Here, the functional and programming requirements are executed in sequence.
- It is same as conventional procedural programming.
- Program instructions are iterated and executed conditionally and the data gets transferred through a series of operations.
- Ex: FSM
- FSM Model: represents state, events, transitions, actions etc.
- Flow chart: models the execution flow.
- The execution of functions in seat belt monitoring system is as given.

```
#define ON 1
```

```
#define OFF 0
```

```
#define YES 1
```

```
#define NO 0
```

```
void seat_belt_warn()
```

```

{
wait_10sec();
if(check_ignition_key()==ON)
{
if(check_seat_belt()==OFF)
{
set_timer(5);
start_alarm();
while((check_seat_belt()==OFF)&&(check_ignition_key()==OFF)&&(timer_expire()==NO));
stop_alarm();    } } }

```

- Fig 7.7 shows the flowchart for seat belt monitoring in sequential program model.

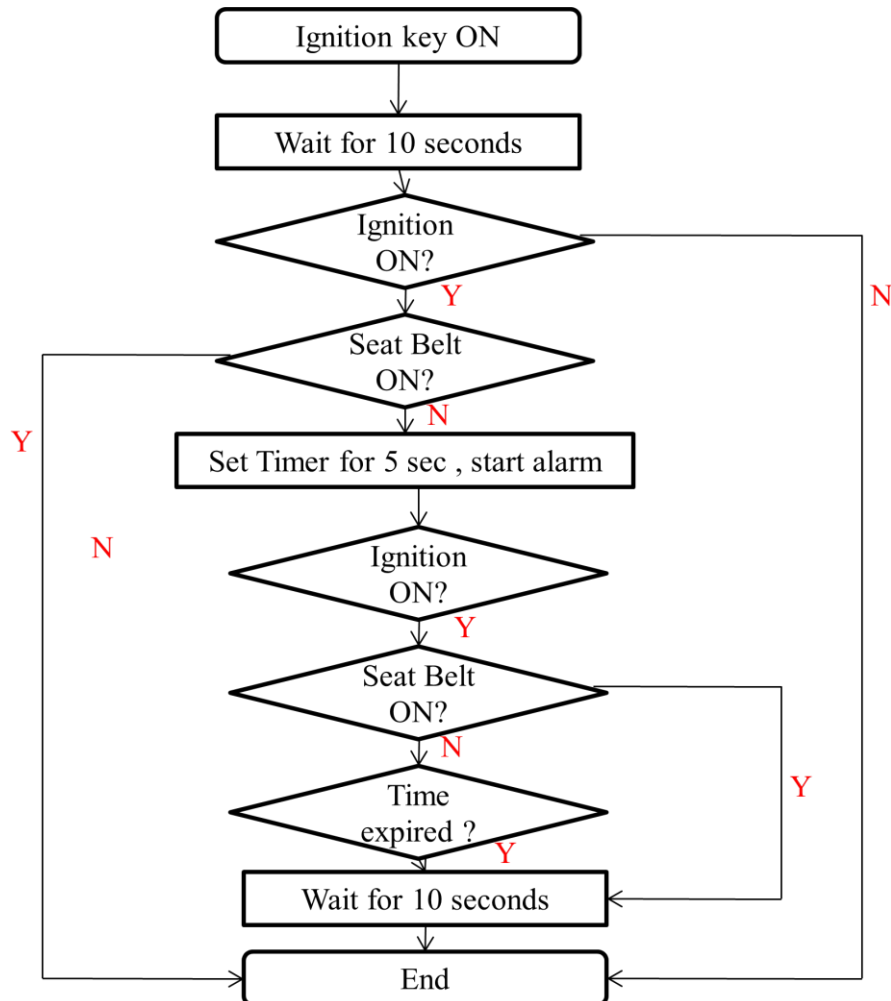


Fig 7.7: Sequential program model for seat belt warning system

Concurrent/Communicating process

- The Concurrent/Communicating process model models concurrently executing tasks/processes.
- It is easier to implement some tasks concurrently than sequentially
- Sequential execution leads to poor processor utilization. **Ex:** When task involves I/O waiting, sleeping etc.
- If the task is split into multiple task, it is possible to utilize CPU effectively, when the subtask goes to sleep, waiting etc.
- ✓ Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication because the processing requirements are split in to multiple tasks, Tasks are executed concurrently and 'Events' are used for synchronizing the execution of tasks
- The concurrent processing model is commonly used for modeling of 'Real time' systems.
- Techniques such as shared memory, message passing, events etc are used for communication and synchronizing between concurrently executing processes.
- **Ex:** Seat belt warning system in concurrent processing is as shown in Fig 7.8 with different tasks

-- The main task is split into the following

1. Timer task for waiting 10 seconds (wait timer task)
2. Task for checking the ignition key (ignition key monitoring task)
3. Task for checking the seat belt status (seat belt status monitoring task)
4. Task for stopping and starting the alarm (alarm control task)
5. Alarm timer task for waiting 5 sec (alarm timer task)

(a)

Fig 7.8 Tasks for seat belt warning system

- Five tasks here sequentially
- Tasks should be execution
- Start the alarm of 10 seconds only if seat belt is ON.
- Alarm control task is executed only when wait timer expires and if the ignition key is in the ON state and the seat belt is in the OFF state.

Create and initialize events *wait_timer_expire*, *ignition_on*, *ignition_off*, *seat_belt_on*, *seat_belt_off*, *alarm_timer_start*, *alarm_timer_expire* Create task *Wait Timer*
Create task *Ignition Key Status Monitor* Create task *Seat Belt Status Monitor* Create task *Alarm Control*
Create task *Alarm Timer*

seat belt warning

can't be executed randomly or synchronized in their through some mechanism. only after the expiration wait timer and that too is OFF and ignition key

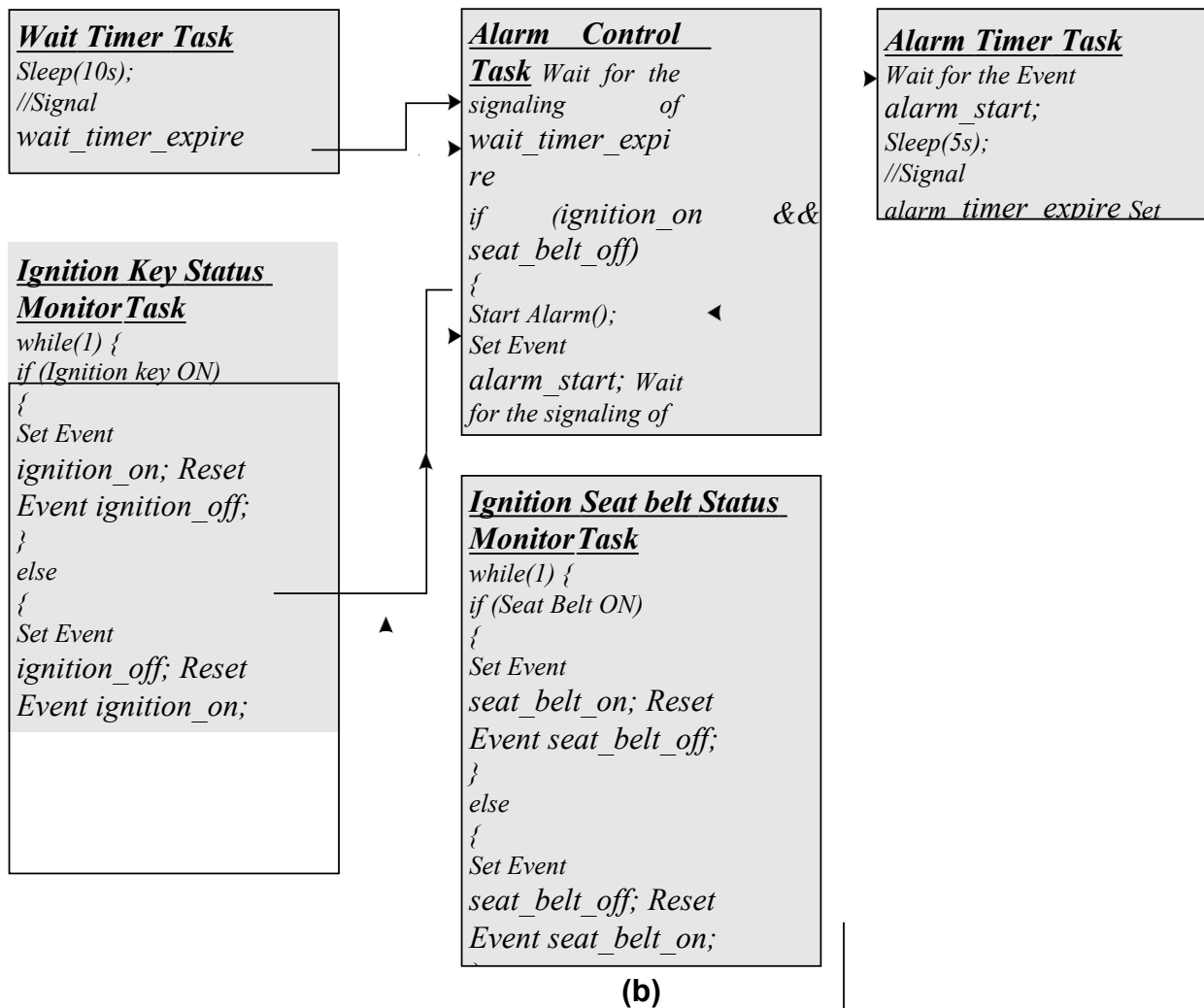


Fig 7.9: Concurrent processing program model for 'Seat Belt Warning System'

- 'wait timer expire' event is associated with the timer task event and it will be reset state initially.
- It is set when the timer expires.
- Events 'ignition on' and 'ignition off' are associated with the task ignition key status monitoring.
- Events 'seat_belt_on' and 'seat_belt_off' are associated with the task seat belt status monitoring.
- The events 'ignition_off' and 'ignition on' are set and reset respectively when the ignition key status is OFF and reset and set respectively when the ignition key status is ON, by the ignition key status monitoring task.

- Initially, seat_belt_off and seat_belt_on are set and reset respectively when the seat belt status is OFF and reset and set respectively when the seat belt status is ON, by the seat belt monitoring task.
- The events alarm_timer_start and alarm_timer_expire are associated with the alarm timer task.
- The alarm_timer_start event will be in the reset state initially and it is set by the alarm control task when the alarm is started.
- The alarm_timer_expire event will be in the reset state initially and it is set when the alarm timer expires.
- The alarm control task waits for the signaling of the event wait_timer_expire and starts the alarm timer and alarm if both the events ignition_on and seat_belt_off are in the set state when the event wait_timer_expire signals.
- If not the alarm control task simply completes its execution and returns.
- In case the alarm is started, the alarm control task waits for the signaling of any one of the events alarm_timer_expire or ignition_off or seat_belt_on.
- Upon signaling any one of these events, the alarm is stopped and the alarm control task simply completes the execution and returns.

Object Oriented Model

- It is an object based model for modeling system requirements.
- It reduces complex software requirements into a simple one with well defined pieces called objects.
- Object oriented modeling brings reusability, maintainability and productivity in system design.
- Object is an entity used for representing or modeling a particular piece of the system.
- Each object is characterized by a set of unique behavior and state.
- Class: is an abstract description of a set of objects and it can be considered as a blueprint of an object.
- Class represents the state of an object through member variables and object behavior through member functions.
- The member variables and member functions of a class can be private, public and protected.
- Private member variables and functions are accessible only within the class
- Public variables and functions are accessible within the class as well as outside the class.
- The protected variables and functions are protected from external access.
- Classes derived from parent class can also access the protected member functions and variables.
- The concept of object and class brings abstraction hiding and protection.

Embedded Firmware Design and Development

- Embedded Firmware Design Approaches
 - *Super loop* based approach
 - *Embedded Operating System* based approach
- Embedded Firmware Development Languages
 - *Assembly Language*
 - *High Level Language*
- The embedded firmware is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the functional requirements of the product.
- The embedded firmware is usually stored in a permanent memory (ROM) and it is non alterable by end users.
- Designing Embedded firmware requires understanding of the particular embedded product hardware, like various component interfacing, memory map details, I/O port details, configuration and register details of various hardware chips used and some programming language (either low level Assembly Language or High level language like C/C++ or a combination of the two).
- The embedded firmware development process starts with the conversion of the firmware requirements into a program model using various modeling tools
- There exist two basic approaches for the design and implementation of embedded firmware, namely;
 - The *Super loop* based approach
 - The *Embedded Operating System* based approach
- The decision on which approach needs to be adopted for firmware development is purely dependent on the complexity and system requirements

Embedded firmware Design Approaches – The Super loop

- Suitable for applications that are not time critical and where the response time is not so important (Embedded systems where missing deadlines are acceptable)
- Very similar to a conventional procedural programming where the code is executed task by task
- The tasks are executed in a never ending loop. The task listed on top on the program code is executed first and the tasks just below the top are executed after completing the first task.
- A typical super loop implementation will look like:
- Configure the common parameters and perform initialization for various hardware components memory, registers etc.

- Start the first task and execute it
- Execute the second task
- Execute the next task
- :
- :
- Execute the last defined task
- Jump back to the first task and follow the same flow

```
void main ()
{
  Configurations ();
  Initializations ();
  while (1)
  {
    Task 1 ();
    Task 2 ();
    //:
    //:
    Task n ();
  }
}
```

Pros

- ✓ Doesn't require an Operating System for task scheduling and monitoring and free from OS related overheads
- ✓ Simple and straight forward design
- ✓ Reduced memory footprint

Cons

- ✓ Non Real time in execution behavior (As the number of tasks increases the frequency at which a task gets CPU time for execution also increases)
- ✓ Any issues in any task execution may affect the functioning of the product (This can be effectively tackled by using Watch Dog Timers for task execution monitoring)

Enhancements

- ✓ Combine Super loop based technique with interrupts
- ✓ Execute the tasks (like keyboard handling) which require Real time attention as Interrupt Service routines

Embedded firmware Design Approaches – Embedded OS based Approach

- ✓ The embedded device contains an Embedded Operating System which can be one of:
 - A Real Time Operating System (RTOS)
 - A Customized General Purpose Operating System (GPOS)

- ✓ The Embedded OS is responsible for scheduling the execution of user tasks and the allocation of system resources among multiple tasks
- ✓ Involves lot of OS related overheads apart from managing and executing user defined tasks
- ✓ Examples of GPOS for embedded devices: Microsoft® Windows XP Embedded.
- ✓ Examples of embedded devices running on embedded GPOSs: Point of Sale (PoS) terminals, Gaming Stations, Tablet PCs etc.
- ✓ Examples of RTOSs employed in Embedded Product development: ‘Windows CE’, ‘Windows Mobile’, ‘QNX’, ‘VxWorks’, ‘ThreadX’, ‘MicroC/OS-II’, ‘Embedded Linux’, ‘Symbian’ etc
- ✓ Examples of embedded devices that runs on RTOSs: Mobile Phones, PDAs, Flight Control Systems etc

Embedded firmware Development Languages/Options

- ✓ Assembly Language
- ✓ High Level Language
 - ✓ Subset of C (Embedded C)
 - ✓ Subset of C++ (Embedded C++)
 - ✓ Any other high level language with supported Cross-compiler
- ✓ Mix of Assembly & High level Language
 - ✓ Mixing High Level Language (Like C) with Assembly Code
 - ✓ Mixing Assembly code with High Level Language (Like C)
 - ✓ Inline Assembly

Assembly Language

- ✓ ‘*Assembly Language*’ is the human readable notation of ‘*machine language*’
- ✓ ‘*machine language*’ is a processor understandable language
- ✓ Machine language is a binary representation and it consists of 1s and 0s
- ✓ Assembly language and machine languages are processor/controller dependent.
- ✓ An Assembly language program written for one processor/controller family will not work with others
- ✓ **Assembly language programming is the process of writing processor specific machine code in mnemonic form, converting the mnemonics into actual processor instructions (machine language) and associated data using an assembler.**
- ✓ The general format of an assembly language instruction is an Opcode followed by Operands
- ✓ The Opcode tells the processor/controller what to do.
- ✓ The Operands provide the data and information required to perform the action specified by the opcode
- ✓ It is not necessary that all opcode should have Operands following them.
- ✓ Some of the Opcode implicitly contains the operand and in such situation no operand is required.

- ✓ The operand may be a single operand, dual operand or more.

❑ The 8051 Assembly Instruction

MOV A, #30

- ✓ Moves decimal value 30 to the 8051 Accumulator register.
- ✓ Here *MOV A* is the Opcode and 30 is the operand (single operand).
- ✓ The same instruction when written in machine language will look like

01110100 00011110

- ✓ The first 8 bit binary value 01110100 represents the opcode for *MOV A*
- ✓ The second 8 bit binary value 00011110 represents the operand 30.
- ✓ Assembly language instructions are written one per line
- ✓ A machine code program consists of a sequence of assembly language instructions, where each statement contains a mnemonic (Opcode + Operand)
- ✓ Each line of an assembly language program is split into four fields as:

LABEL OPCODE OPERAND COMMENTS

- ✓ LABEL is an optional field.
- ✓ A 'LABEL' is an identifier used extensively in programs to reduce the reliance on programmers for remembering where data or code is located.
- ✓ LABEL is commonly used for representing a memory location, address of a program, sub-routine, code portion etc.
- ✓ The maximum length of a label differs between assemblers.
- ✓ Assemblers insist strict formats for labeling.
- ✓ Labels are always suffixed by a colon and begin with a valid character. Labels can contain number from 0 to 9 and special character _ (underscore).
- ✓ Fig 9.2 gives assembly level program to generate delay

```
#####
##
;          SUBROUTINE FOR GENERATING DELAY
;          DELAY PARAMETR PASSED THROUGH REGISTER R1
;          RETURN VALUE NONE
;          REGISTERS USED: R0, R1
#####
##      DELAY: MOV    R0, #255 ; Load Register R0 with 255
              DJNZ    R1, DELAY      ; Decrement R1 and loop till
              R1= 0
              RET                    ; Return to calling program
```

Fig 9.2: Sample assembly code to generate delay

- ✓ The symbol ; represents the start of a comment. Assembler ignores the text in a line after the ; symbol while assembling the program
- ✓ DELAY is a label for representing the start address of the memory location where the piece of code is located in code memory

- ✓ The above piece of code can be executed by giving the label `DELAY` as part of the instruction. E.g. `LCALL DELAY; LJMP DELAY`

Assembly Language – Source File to Object File Translation

- ✓ The Assembly language program written in assembly code is saved as *.asm* (Assembly file) file or a *.src* (source) file or a format supported by the assembler.
- ✓ Similar to 'C' and other high level language programming, it is possible to have multiple source files called modules in assembly language programming.
- ✓ Each module is represented by a '*.asm*' or '*.src*' file or the assembler supported file format similar to the '*.c*' files in C programming.
- ✓ The software utility called 'Assembler' performs the translation of assembly code to machine code.
- ✓ The assemblers for different family of target machines are different.
- ✓ A51 Macro Assembler from Keil software is a popular assembler for the 8051 family micro controller
- ✓ Each source file can be assembled separately to examine the syntax errors and incorrect assembly instructions

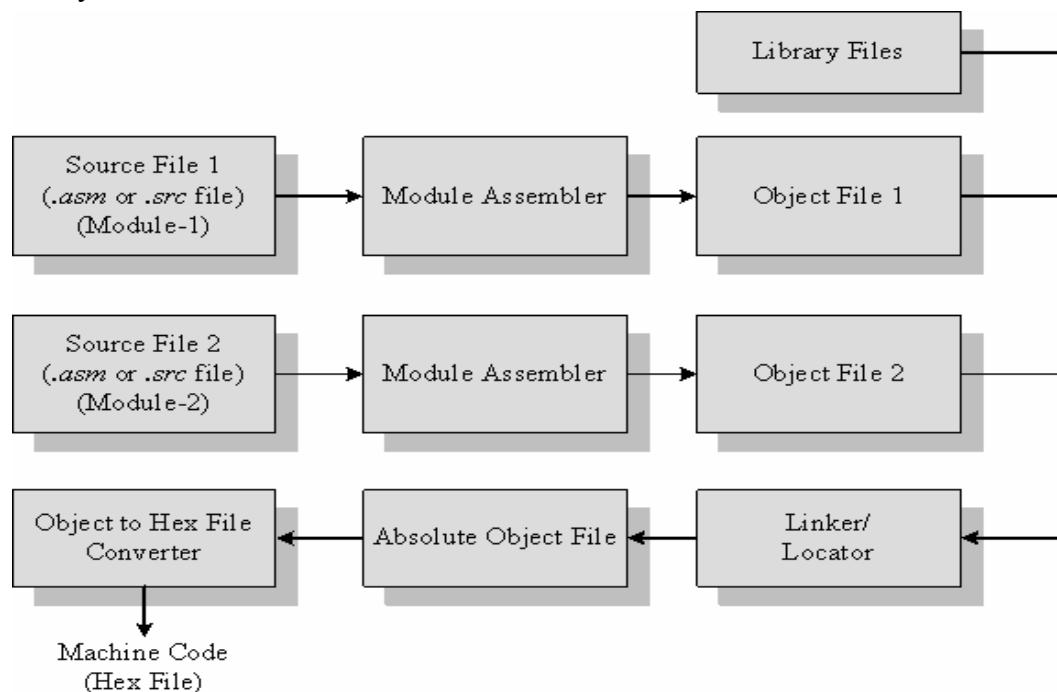


Fig 9.3: Assembly language to machine language Conversion process

- ✓ Assembling of each source file generates a corresponding object file.
- ✓ The object file does not contain the absolute address of where the generated code needs to be placed (a re-locatable code) on the program memory
- ✓ The software program called linker/locator is responsible for assigning absolute address to object files during the linking process.

- ✓ The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory
- ✓ A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file).
- ✓ Fig 9.3 shows assembly to machine language conversion process.

Library file creation and usage

- ✓ Libraries are specially formatted, ordered program collections of object modules that may be used by the linker at a later time.
- ✓ When the linker processes a library, only those object modules in the library that are necessary to create the program are used.
- ✓ Library files are generated with extension '.lib'.
- ✓ Library file is some kind of source code hiding technique.
- ✓ If you don't want to reveal the source code behind the various functions you have written in your program and at the same time you want them to be distributed to application developers for making use of them in their applications, you can supply them as library files and give them the details of the public functions available from the library (function name, function input/output, etc).
- ✓ For using a library file in a project, add the library to the project.
- ✓ If you are using a commercial version of the assembler/compiler suite for your development, the vendor of the utility may provide you pre-written library files for performing multiplication, floating point arithmetic, etc. as an add-on utility or as a bonus.
- ✓ 'LIB51' from Keil Software is an example for a library creator and it is used for creating library files for A51 Assembler/C51 Compiler for 8051 specific controller.

Linker and Locator

- ✓ It is another software utility responsible for "linking the various Object modules in a multi module project and assigning absolute address to each module".
- ✓ Linker generates an absolute object module by extracting the Object modules from the library, if any and those obj files created by the assembler, which is generated by assembling the individual modules of a project.
- ✓ It is the responsibility of the linker to link any external dependent variables or functions declared on various modules and resolve the external dependencies among the modules.
- ✓ An absolute Object file Or module does not contain any re-locatable code or data.
- ✓ All code and data reside at fixed memory locations.
- ✓ The absolute Object file is used for creating hex files for dumping into the code memory of the processor/controller.

- ✓ BL51' from Keil Software is an example for a Linker & Locator for A51 Assembler/C51 Compiler for 8051 specific controller.

Object to Hex File Converter

- ✓ This is the final stage in the conversion of Assembly language (mnemonics) to machine understandable language (machine code).
- ✓ Hex File is the representation of the machine code and the hex file is dumped into the code memory of the processor/controller.
- ✓ The hex file representation varies depending on the target processor/controller make.
- ✓ For Intel processors/controllers the target hex tile format will be 'Intel HEX' and for Motorola, the hex tile should be in 'Motorola HEX' format.
- ✓ HEX files are ASCII files that contain Hexadecimal representation Of target application. Hex tile is created from the final 'Absolute Object File' using the Object to Hex File Converter utility.
- ✓ 'OH51 'from Keil software is an example for Object to Hex File Converter utility for A51 Assembler, C51 Compiler for 8051 specific controller.

Advantages of assembly language based development

- ✓ Efficient Code Memory & Data Memory Usage (Memory Optimization)
 - Since the developer is well versed with the target processor architecture and memory organization, optimised code can be written for performing operations.
 - This leads to less utilization of code memory and efficient utilization of data memory.
- ✓ High Performance
 - Optimised code(through effective coding techniques) not only improves the code memory usage but also improves the total system performance.
- ✓ Low level Hardware Access
 - Most of the code for low level programming like accessing external device specific registers from the operating system kernel, device drivers and low level interrupt routines etc are making use of direct assembly coding.
- ✓ Code Reverse Engineering
 - Reverse engineering is the process of understanding the technology behind a product by extracting the information from a finished product.
 - If hawkers cracks the memory protection and read the code memory, it can be easily converted into assembly code using a disassemble program for target machine.

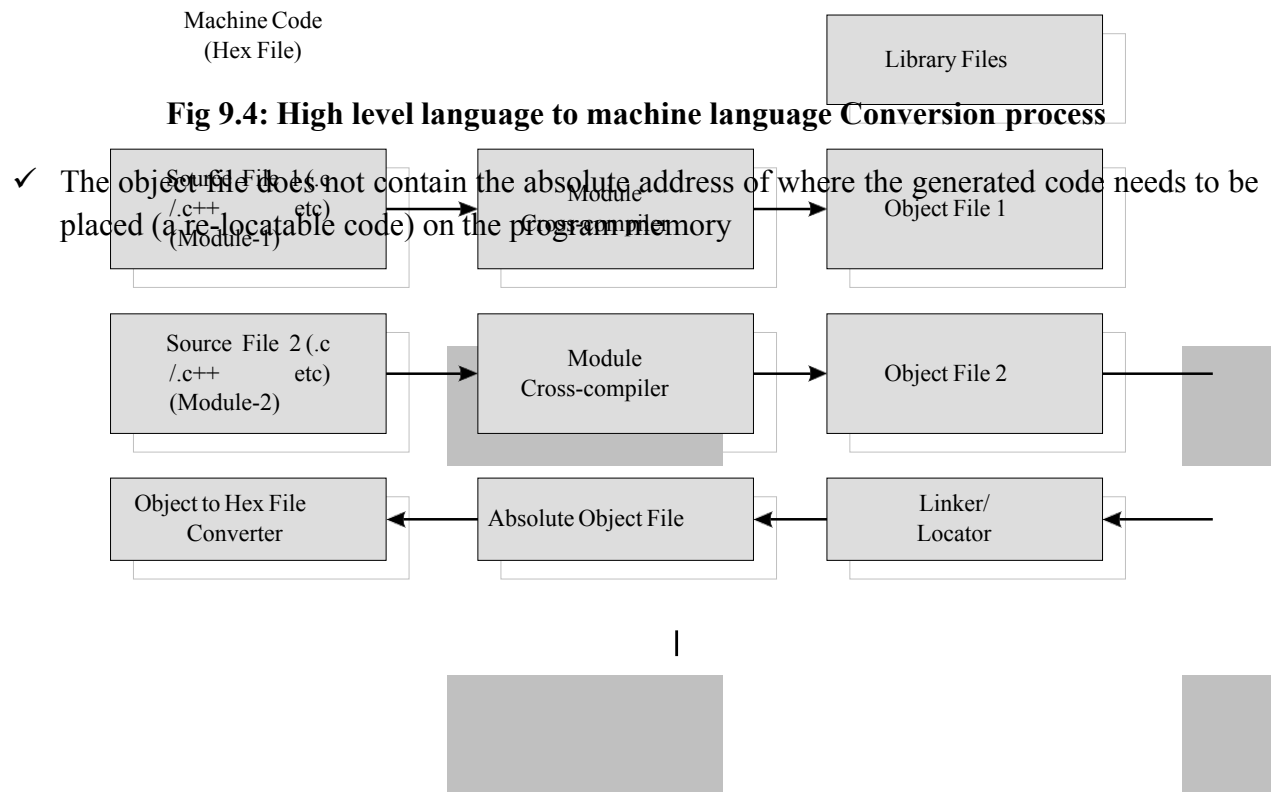
Drawbacks of assembly language based development

- ✓ High Development time

- Assembly developer must pay attention to inner details and must have thorough knowledge of the architecture, memory organization and register details of the target processor.
- So the design is highly time consuming, results in delayed product development.
- ✓ Developer dependency
 - In assembly, the developer can choose any register and different memory for coding.
 - Programming approach differs from developer to developer.
 - If the approach done by a developer is not documented properly at development stage, he/she may not be able to recollect why this approach is followed at a later stage or when a new developer is instructed to analyse this code, he/she also may not be able to understand what is done and why it is done so.
- ✓ Non portable
 - Target applications written in assembly instructions are valid only for that particular family of processors and cannot be reused for another target processor/controller.

High Level Language

- ✓ The embedded firmware is written in any high level language like C, C++
- ✓ A software utility called 'cross-compiler' converts the high level language to target processor specific machine code
- ✓ The cross-compilation of each module generates a corresponding object file.



- ✓ The software program called linker/locator is responsible for assigning absolute address to object files during the linking process
- ✓ The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory.
- ✓ A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file).
- ✓ Fig 9.4 represents high level language to machine language conversion process.

Advantages of high level language

- ✓ Reduced Development time
 - Developer needs less/little knowledge on the internal hardware details and architecture of the target processor/controller.
 - Minimum knowledge on memory organization and registers with syntax of high level language are the only prerequisites for high level language based firmware development.
- ✓ Developer independency
 - The syntax used by most of the high level language is universal.
 - Program written in high level language can be easily understood by second person knowing the syntax of the language.
- ✓ Portability
 - An application written in high level language for a particular target processor can easily be converted to another target processor/controller specific application, with little or less effort by simply recompiling/little code modification followed by recompiling the application for the required target processor/controller, provided, the cross compiler has support for the processor/controller selected.

Limitations of high level language

- ✓ The investments required for high level language based development tools is high compared to assembly language based firmware tools.
- ✓ Some cross compilers available for high level languages may not be so efficient in generating optimized target processor specific instructions. Target images created by such compilers may be messy and non optimized in terms of performance as well as code size.

Mixing of Assembly Language and High Level Language

- ✓ Certain situations in Embedded firmware development may require the mixing of Assembly Language with high level language or vice versa.
- ✓ 3 ways of mixing:
 - Mixing Assembly Language with High level language like 'C'

- Mixing High level language like 'C' with Assembly Language
 - In line Assembly
 - The passing of parameters and return values between the high level and low level language is cross-compiler specific

Mixing Assembly Language with High level language like 'C'

- Here, the entire program is written in C and cross compiler in use do not have a built in support for implementing some features like ISR.
- While accessing certain low level hardware, timing specifications are critical. Cross compiler generated binary may not be able to offer required time specifications accurately.
- Mixing of C and assembly is little complicated. Because passing of parameters from a C function to an assembly program should be well known by the programmer.
- Same with parameter returning from assembly to C language.
- All these are compiler dependent.

Mixing High level language like 'C' with Assembly Language

- It is useful in following scenarios
- The source code is already available in Assembly language and routine written in a high level language like C needs to be included to the existing code.
- The entire source code is planned in assembly(reasons: optimized code, optimal performance, efficient code memory utilization and expertise in handling assembly etc. Ex: 16 bit multiplication and division in assembly is difficult to code. So C can be used for this coding.
- To include built-in functions written in C language provided by cross compilers. Ex: built-in graphics library functions and string operations supported in C.
- Ex: LCALL __Cfunction
Where Cfunction is a function written in C language, Prefix _ informs the cross compiler that the parameters to the function are passed through register. If the functions is invoked without the _prefix, it is understood that the parameters are passed through fixed memory locations.

In line Assembly

- It's a technique to insert target processor/ controller specific assembly instructions at any location of a source code written in high level language C.
- Avoids the delay in calling an assembly routine from C
- #pragma asm --- keyword to indicate the start of program
- #pragma endasm --- keyword to indicate the end of program.
- Keywords are cross compiler specific.
- Ex: #pragma asm
MOV A, #13H

#pragma endasm

Programming in Embedded C

- ✓ Whenever the conventional C language and its extensions are used for programming embedded systems, it is referred as Embedded C programming.
- ✓ For a desktop application developer, the resources available are surplus in quantity and s/he can be very lavish in the usage of RAM and ROM and no restrictions are imposed at all.
- ✓ All embedded systems are limited in both storage and working memory resources.
- ✓ Embedded applications must be developed in a best possible way which optimizes the code memory and working memory usage as well as performance.

‘C’ V/s Embedded C

- ✓ ‘C’ is a well structured, well defined and standardized general purpose programming language with extensive bit manipulation support.
- ✓ ‘C’ offers a combination of the features of high level language and assembly and helps in hardware access programming (system level programming) as well as business package developments (Application developments like pay roll systems, banking applications etc)
- ✓ The conventional ‘C’ language follows ANSI standard and it incorporates various library files for different operating systems.
- ✓ A platform (Operating System) specific application, known as, compiler is used for the conversion of programs written in ‘C’ to the target processor (on which the OS is running) specific binary files.
- ✓ Embedded C can be considered as a subset of conventional ‘C’ language.
- ✓ Embedded C supports all ‘C’ instructions and incorporates a few target processor specific functions/instructions.
- ✓ The standard ANSI ‘C’ library implementation is always tailored to the target processor/controller library files in Embedded C.
- ✓ The implementation of target processor/controller specific functions/instructions depends upon the processor/controller as well as the supported cross-compiler for the particular Embedded C language.
- ✓ A software program called ‘Cross-compiler’ is used for the conversion of programs written in Embedded C to target processor/controller specific instructions.

‘Compiler’ V/s ‘Cross-Compiler’

- ✓ Compiler is a software tool that converts a source code written in a high level language on top of a particular operating system running on a specific target processor architecture (E.g. Intel x86/Pentium).
- ✓ Here, the operating system, the compiler program and the application making use of the source code run on the same target processor.
- ✓ The source code is converted to the target processor specific machine instructions

- ✓ A **native compiler** generates machine code for the same machine (processor) on which it is running.
- ✓ **Cross compiler** is the software tools used in cross-platform development applications
- ✓ In cross-platform development, the compiler running on a particular target processor/OS converts the source code to machine code for a target processor whose architecture and instruction set is different from the processor on which the compiler is running or for an operating system which is different from the current development environment OS
- ✓ Embedded system development is a typical example for cross-platform development where embedded firmware is developed on a machine with Intel/AMD or any other target processors and the same is converted into machine code for any other target processor architecture (E.g. 8051, PIC, ARM9 etc).
- ✓ Keil C51 compiler from Keil software is an example for cross-compiler for 8051 family architecture

ADDITIONAL INFO

1. Difference between C and C++

C	C++
C was developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs.	C++ was developed by Bjarne Stroustrup in 1979 with C++'s predecessor "C with Classes".
When compared to C++, C is a subset of C++.	C++ is a superset of C. C++ can run most of C code while C cannot run C++ code.
C supports procedural programming paradigm for code development.	C++ supports both procedural and object oriented programming paradigms; therefore C++ is also called a hybrid language.
C does not support object oriented programming; therefore it has no support for polymorphism, encapsulation, and inheritance.	Being an object oriented programming language C++ supports polymorphism, encapsulation, and inheritance.
In C (because it is a procedural programming language), data and functions are separate and free entities.	In C++ (when it is used as object oriented programming language), data and functions are encapsulated together in form of an object. For creating objects class provides a blueprint of structure of the object.
In C, data are free entities and can be manipulated by outside code. This is because C does not support information hiding.	In C++, Encapsulation hides the data to ensure that data structures and operators are used as intended.

C, being a procedural programming, it is a function driven language.	While, C++, being an object oriented programming, it is an object driven language.
C does not support function and operator overloading.	C++ supports both function and operator overloading.
C does not allow functions to be defined inside structures.	In C++, functions can be used inside a structure.
C does not have namespace feature.	C++ uses NAMESPACE which avoid name collisions. A namespace is a declarative region that provides a scope to the identifiers (the names of types, functions, variables, etc) inside it. Namespaces are used to organize code into logical groups and to prevent name collisions that can occur especially when your code base includes multiple libraries. All identifiers at namespace scope are visible to one another without qualification. Identifiers outside the namespace can access the members by using the fully qualified name for each identifier.
C uses functions for input/output. For example scanf and printf.	C++ uses objects for input output. For example cin and cout.
C does not support reference variables.	C++ supports reference variables.
C has no support for virtual and friend functions.	C++ supports virtual and friend functions.

2. Differences between C programming and Embedded C programming

C programming	Embedded C programming
Possesses native development in nature.	Possesses cross development in nature.
Independent of hardware architecture.	Dependent on hardware architecture (microcontroller or other devices).
Used for Desktop applications, OS and PC memories.	Used for limited resources like RAM, ROM and I/O peripherals on embedded controller.

3. Differences between Compiler and Assembler

BASIS FOR COMPARISON	COMPILER	ASSEMBLER
Basic	Generates the assembly language code or directly the executable code.	Generates the relocatable machine code.
Input	Preprocessed source code.	Assembly language code.
Phases/ Passes	The compilation phases are lexical analyzer, syntax analyzer, semantic analyzer, intermediate code generation, code optimization, code generation.	Assembler makes two passes over the given input.
Output	The assembly code generated by the compiler is a mnemonic version of machine code.	The relocatable machine code generated by an assembler is represented by binary code.

4. Describe Controller Area Network (CAN) protocol

- The CAN bus uses bit serial transmission. CAN runs at rates of 1MBps over a twisted pair connection of 40m. As optical link can also be used. The bus protocol supports multiple masters on the bus.
- CAN is very widely used in cars as well as in other applications.
- An automotive serial bus system developed to satisfy the following requirements.
 - a. Network multiple microcontroller with 1 pair of wires.
 - b. Allow microcontrollers communicate with each other.
 - c. High speed, real time communication.
 - d. Provide noise immunity in an electrically noisy environment.
 - e. Low cost.
- At each CAN device, the start of the frame bit notifies a transmission is being sent. The identifier bit shows the priority of the message along with determining which device the data belongs to.
- In CAN terminology, a logical 1 on the bus is called recessive and a logical 0 is dominant. When all nodes are transmitting 1s, the bus is said to be in the recessive state; when a node transmits a 0, the bus is in the dominant state.
- CAN is a synchronous bus, all transmitters must send at the same time for bus arbitration to work.

- The four different message types or frames, that can be transmitted on a CAN bus are the data frame, the remote frame, the error frame and the overload frame.
- The data frame is the most common message type and comprises the Arbitration Field, the Data Field, the CRC Field and the Acknowledgement Field.
- The error frame is a special message that violates the formatting rules of a CAN message. It is transmitted when a node detects an error in a message, and causes all other nodes in the network to send an error frame as well.

CAN logic and Arbitration

- a. CAN 2.0A messages begins with an 11bit message ID which identifies the message type and also establishes the message priority.
- b. As with many computer interfaces, the CAN transceivers invert the microcontroller signal. Thus, the dominant bus state occurs when a logic 0 is transmitted and the recessive state occurs when a logic 1 is transmitted.
- c. CAN uses the message ID to perform bus access arbitration between nodes.
- d. Each node waits for an idle bus state then begins to transmit its message ID.
- e. Each node also listens to the bus to see if the bus state match its transmission.
- f. If a node detects a dominant bus state while transmitting a recessive message ID bit (logic 1), it drops out of the current arbitration round and will try again the next time the bus is idle.

Advantages of CAN

- a. Low cost network infrastructure which is often built into microcontroller.
- b. Large market segment with broad availability of hardware, software and system engineering tools.
- c. Light weight, low latency, highly deterministic design specifically for real time embedded applications.
- d. Reliable with strong error detection, fault tolerant versions available.
- e. Flexible and highly configurable with various higher level application protocols.
- f. Foundation for next generation technology Controller Area Network.

5. Describe Local Interconnect Network

- LIN [Local Interconnect Network] is used as an in vehicle communication and networking serial bus between intelligent sensors and actuators operating at 12V.
- The LIN bus is a sub bus system based on a serial communication protocol. The bus is a single master/multiple slave bus that uses a single wire to transmit data.
- Only the master is able to initiate a communication. LIN frame consists of a header and a response part. To initiate a communication with a slave the master sends the header part. If the master wants to send data to the slave it goes on sending the response part. If the master requests data from the slave the slave sends the response part.

- A typical LN node consists of a microcontroller for handling the LIN protocol and a LIN transceiver for interfacing the digital part and the physical line.

6. Describe Media Oriented System Transport (MOST) Bus

- The MOST bus specification is fast becoming an auto industry standard for building in vehicle multimedia systems. A network can consists of upto 64 devices and sample rates of or 48kHz.
- MOST network must have a number of masters for different functions. Th masters can be containing in the same devices.
 1. Timing master: Controls the timing of the network and thereby the synchronization between the devices.
 2. Network Master: Sets up the network and allocates addresses between devices.
 3. Connection Master: Sets up the synchronous communication channels between devices.
 4. Power Master: Monitors the power supplies. Handles power up and shut down.
- MOST devices is not connected to a bus in the common sense. It has an in port and an out port and passes the information from the in port to the out port.
- There are three communication channels open to applications.
 1. **Control Channel:** For event oriented transmission with low bandwidth (10kBits/s) and short package length.
 2. **Asynchronous Channel:** Packet oriented transmission with large block size and high bandwidth.
 3. **Synchronous Channel:** Continuous data streams that require high bandwidth.