Module 4
EMBEDDED SYSTEM DESIGN CONCEPTS

Characteristics & Quality Attributes of Embedded Systems

The characteristics of embedded system are different from those of a general purpose computer and so are its Quality metrics. This chapter gives a brief introduction on the characteristics of an embedded system and the attributes that are associated with its quality.

CHARACTERISTICS OF EMBEDDED SYSTEM

Following are some of the characteristics of an embedded system that make it different from a general purpose computer:

1. Application and Domain specific

- Each embedded system is having certain functions to perform and they are developed in such a manner to do the intended functions only.
- They cannot be used for any other purpose.
- It is the major criterion which distinguishes an embedded system from a general purpose system.
- For example, you cannot replace the embedded control unit of your microwave oven with your air conditioner's embedded control unit, because the embedded control units of microwave oven and air conditioner are specifically designed to perform certain specific tasks.

2. Reactive and Real time

- Embedded systems are in constant interaction with the Real world through sensors and user-defined input devices which are connected to the input port of the system.
- Any changes happening in the Real world (which is called an Event) are captured by the sensors or input devices in Real Time and the control algorithm running inside the unit reacts in a designed manner to bring the controlled output variables to the desired level.
- Real Time System operation means the timing behavior of the system should be deterministic; meaning the system should respond to requests or tasks.in a known amount of time. A Real Time system should not miss any deadlines for tasks or operations.

- Embedded applications or systems which are mission critical, like flight control systems, Antilock Brake Systems (ABS), etc. are examples of Real Time systems.

3. Operation in harsh environment:

- It is not necessary that all embedded systems should be deployed in controlled environments. The environment in which the embedded system deployed may be a dusty one or a high temperature zone or an area subject to vibrations and shock. Systems placed in such areas should be capable to withstand all these adverse operating conditions.
- The design should take care of the operating conditions of the area where the system is going to implement.
- For example, if the system needs to be deployed in a high temperature zone, then all the components used in **the** system should be of high temperature grade.
- Also proper shock absorption techniques should be provided to systems which are going to be commissioned in places subject to high shock. Power supply fluctuations, corrosion and component aging, etc. are the other factors that need to be taken into consideration for embedded systems to work in harsh environments.

4. Distributed systems:

- Certain embedded systems are part of a larger system and thus form components of a distributed system.
- These components are independent of each other but have to work together for the larger system to function properly.
- Ex. A car has many embedded systems controlled to its dash board. Each one is an independent embedded system yet the entire car can be said to function properly only if all the systems work together.

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.
- 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.
- If more power is required then a battery of higher power or more batteries need to be accommodated in the embedded system.

QUALITY ATTRIBUTES OF 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:-

1. 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.

2. 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

a) Response

- Response is a measure of quickness of the system. It gives you an idea about how fast your system is tracking the changes in input variables.
- Most of the embedded systems demand fast response which should be almost Real Time.
- For example, an embedded system deployed in flight control application should respond in a Real Time manner. Any response delay in the system will create potential damages to the safety of the flight as well as the passengers.

b) Throughput

- Throughput deals with the efficiency of a system. In general it can be defined as the rate of production or operation of a defined process over a stated period of time.
- The rates can be expressed in terms of units of products, batches produced, or any other meaningful measurements.
- In the case of a Card Reader, throughput means how many transactions the Reader can perform in a minute or in an hour or in a day. Throughput is generally measured in terms of 'Benchmark'.

c) 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.

d) Maintainability

- Maintainability deals with support and maintenance to the end user or client in case of technical issues and product failures or on the basis of a routine system checkup.
- Reliability and maintainability are considered as two complementary disciplines. A more reliable system means a system with less corrective maintainability requirements and vice versa.
- As the reliability of the system increases, the chances of failure and non-functioning also reduces, thereby the need for maintainability is also reduced.
- Maintainability is closely related to the system availability.
- Maintainability can be broadly classified into two categories, namely, 'Scheduled or Periodic Maintenance (preventive maintenance)' and 'Maintenance to unexpected failures (corrective maintenance)'. It can be classified into two types :-

1. Scheduled or Periodic Maintenance

- 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:
  1. Air conditioner not powering on
  2. Printer not taking paper in spite of a full paper stack

e) Security

- Confidentiality, Integrity and Availability are three corner stones of information security.
- Confidentiality deals with protection data from unauthorized disclosure.
- Integrity gives protection from unauthorized modification.
- Availability gives protection from unauthorized user
- Certain Embedded systems have to make sure they conform to the security measures.
- Ex. An Electronic Safety Deposit Locker can be used only with a pin number like a password.

Non Operational Attributes

a) Testability and Debug-ability:

- It deals with how easily one can test his/her design, application and by which mean he/she can test it.
- In hardware testing the peripherals and total hardware function in designed manner
- Firmware testing is functioning in expected way
- Debug-ability is means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system

b) Evolvability:

- For embedded system, the qualitative attribute "Evolvability" refer to ease with which the embedded product can be modified to take advantage of new firmware or hardware technology.

c) Portability

- Portability is a measure of 'system independence'. An embedded product is said to be portable if the product is capable of functioning 'as such' in various environments, target processors/controllers and embedded operating systems.

- The ease with which an embedded product can' be ported on to a new platform is a direct measure of the re-work required. A standard embedded product should always be flexible and portable.

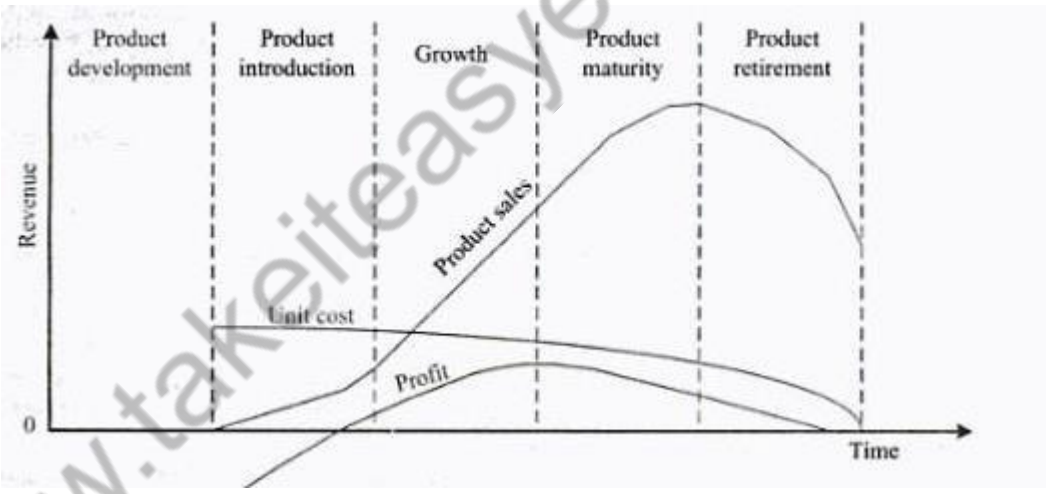
d) Time to prototype and market

- Time to Market is the time elapsed between the conceptualization of a product and time at which the product is ready for selling or use

- Product prototyping help in reducing time to market.
- Prototyping is an informal kind of rapid product development in which important feature of the under consider are develop.
- In order to shorten the time to prototype, make use of all possible option like use of reuse, off the self-component etc**.**

e) Per unit and total cost

- Cost is an important factor which needs to be carefully monitored. Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.
- When the product is introduced in the market, for the initial period the sales and revenue will be low
- There won't be much competition when the product sales and revenue increase.
- During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.



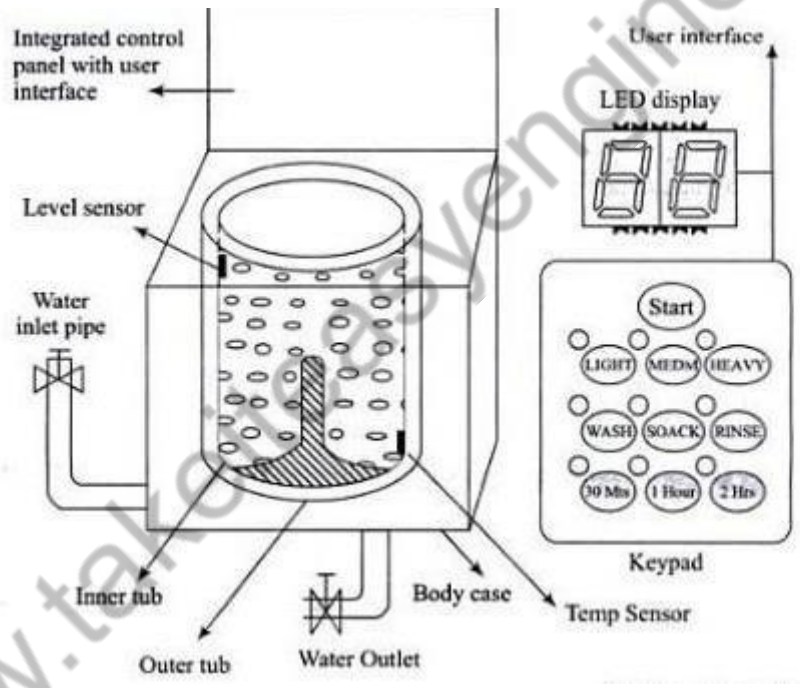
Embedded Systems-Application and Domain specific

Application specific systems: Washing Machine

Let us see the important parts of the washing machine; this will also help us understand the working of the 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 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 theinner 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 it 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.

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. The entire processing is done on a kind of processor which may be a microprocessor or microcontroller.

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

Automotive Embedded System (AES)

- The Automotive industry is one of the major application domains of embedded systems.
- Automotive embedded systems are the one where electronics take control over the mechanical system. Ex. Simple viper control.
- The number of embedded controllers in a normal vehicle varies somewhere between 20 to 40 and can easily be between 75 to 100 for more sophisticated vehicles.
- One of the first and very popular uses of embedded system in automotive industry was microprocessor based fuel injection.

Some of the other uses of embedded controllers in a vehicle are listed below:

a. Air Conditioner

b. Engine Control

c. Fan Control

d. Headlamp Control

e. Automatic break system control

f. Wiper control

g. Air bag control

h. Power Windows

Types of Electronic Control Units (ECU)

1. High-speed Electronic Control Units (HECUs):

a. HECUs are deployed in critical control units requiring fast response.

b. They Include fuel injection systems, antilock brake systems, engine control, electronic throttle, steering controls, transmission control and central control units.

2. Low Speed Electronic Control Units (LECUs):-

a. They are deployed in applications where response time is not so critical.

b. They are built around low cost microprocessors and microcontrollers and digital signal processors.

c. Audio controller, passenger and driver door locks, door glass control etc.

- **Automotive Communication Buses**

Embedded system used inside an automobile communicates with each other using serial buses. This reduces the wiring required.

Following are the different types of serial Interfaces used in automotive embedded applications:

a. Controller Area Network (CAN):-

- CAN bus was originally proposed by Robert Bosch.
- It supports medium speed and high speed data transfer
- CAN is an event driven protocol interface with support for error handling in data transmission.

b. Local Interconnect Network (LIN):-

- LIN bus is single master multiple slave 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 to eliminate the bus arbitration problem
- LIN bus applications are mirror controls , fan controls , seat positioning controls.

c. Media-Oriented System Transport (MOST):-

- MOST is targeted for automotive audio/video equipment interfacing
- A MOST bus is a multimedia fiber optics point–to- point network implemented in a star , ring or daisy chained topology over optical fiber cables.
- MOST bus specifications define the physical as well as application layer , network layer and media access control.

Hardware and Software Co-Design

The fundamental issues in hardware software co-design are:

1. **Selecting the model:** In hardware software co-design, models are used for capturing and describing the system characteristics. A 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.
2. **Selecting the Architecture:**

   The architecture specifies how a system is going to implement in terms of the number and types of different components and the interconnection among them. Controller architecture, Datapath Architecture, Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), Very Long Instruction Word Computing (VLIW), Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD), etc. are the commonly used architectures in system design.

   - The controller architecture implements the finite state machine model (which we will discuss in a later section) using a state register and two combinational circuits (we will discuss about combinational circuits in a later chapter). The state register holds the present state and the combinational circuits implement the logic for next state and output.
   - The datapath 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 datapath represents a channel between the input and output and in datapath architecture the datapath may contain registers, counters, register files, memories and ports along with high speed arithmetic units. Ports connect the datapath to multiple buses.
   - The Finite State Machine Datapath (FSMD) architecture combines the controller architecture with datapath architecture. It implements a controller with datapath. The controller generates the control input whereas the datapath processes the data. The datapath contains two types of I/O ports, out of which one acts as the control port for receiving/sending the control signals from/to the controller unit and the

second I/O port interfaces the datapath with' external world for data input and data output.

- 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 (e.g. Reading a register value and comparing it with a given value and then transfer the program execution to a new address location (The CJNE instruction for 8051 ISA)) with a single instruction. The use of a single complex instruction in place of multiple simple instructions greatly reduces the program memory access and program memory size requirement.
- The Very Long Instruction Word (VLIW) architecture implements multiple functional units (ALUs, multipliers, etc.) in the datapath. The VLIW instruction packages one standard instruction per functional unit of the datapath.
- Parallel processing architecture implements multiple concurrent Processing Elements (PEs) and each processing element may associate a datapath containing register and local memory. Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) architectures are examples for parallel processing architecture. In 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. On the other hand, the processing elements of the MIMD architecture execute different instructions at a given point off-time. The MIMD architecture forms the basis of multiprocessor systems. The PEs in a multiprocessor system communicates through mechanisms like shared memory and message passing.

3. **Selecting the language:** A programming language captures a 'Computational Model' and maps it into architecture. A model can be captured using multiple programming languages like C, C++, C#, Java, etc. for software implementations and languages like VHDL, System C, Verilog, etc. for hardware implementations. On the other hand, a single language can be used for capturing a variety of models. Certain languages are good in capturing certain computational model. For example, C++ is a good candidate for capturing an object oriented model.

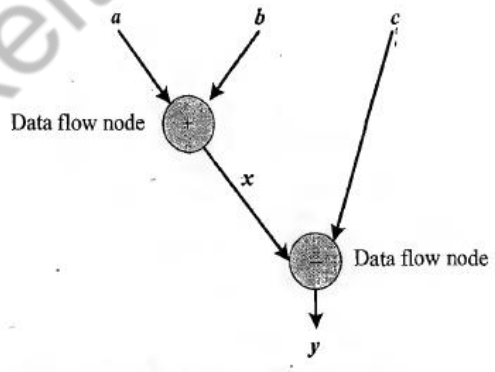
4. **Partitioning System Requirements into hardware and software:** So far we discussed about the models for capturing the system requirements and the architecture for implementing the system. From an implementation perspective, it may be possible to implement the system requirements in either hardware 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 Systems:

1. Data Flow Graph/Diagram (DFG) Model

The Data Flow Graph (DFG) model translates the data processing requirements into a data flow graph. The Data Flow Graph (DFG) model is a data driven model in which the program execution is determined by data. This model emphasizes on the data and operations on the data which transforms the input data to output data. Indeed 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. An inward arrow to the process (circle) represents input data and an outward arrow from the process (circle) represents output data in DFG notation.

Figure illustrates the implementation of a DFG model for implementing these requirements.



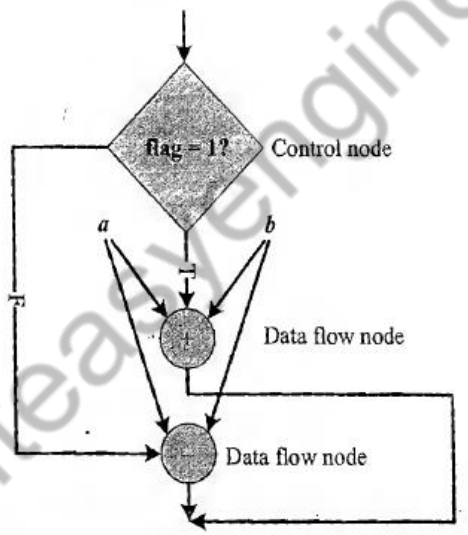
In a DFG model, a data path is the data flow path from input to output. A DFG model is said to be acyclic DFG (ADFG) if it doesn't contain multiple values for the input variable and multiple output values for a given set of input(s). Feedback inputs (Output is fed back to Input), events, etc. are examples for non-acyclic inputs. A DFG model translates the program as a single sequential process execution.

2. Control Data Flow Graph/Diagram (CDFG)

The Control DFG (CDFG) model is used for modeling applications involving conditional program execution. CDFG models contains both data operations 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.

If flag = 1, x = a + b else y = a-b;

This requirement contains a decision making process. The CDFG model for the same is given in below Fig,



The control node is represented by a 'Diamond' block which is the decision making element in a normal 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.

A real world example 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 the decision on, in which format the image is stored (formats like JPEG, TIFF, BMP, etc.) is controlled by the camera settings, con¬ figured by the user.

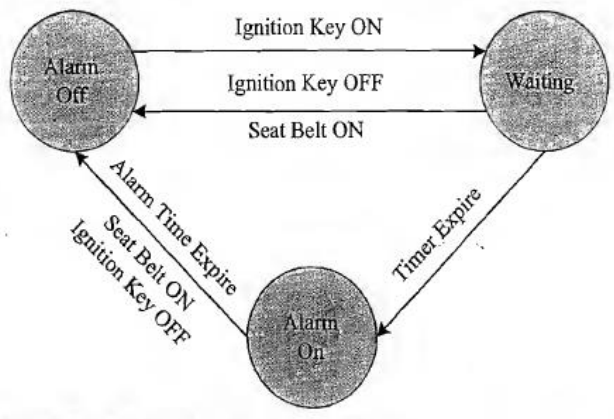
3. State Machine Model

The State Machine model is used for modeling reactive or event-driven embedded systems whose processing behavior is dependent on state transitions. Embedded systems used in the control and industrial applications are typical examples for event driven systems. The State Machine model describes the system behavior with 'States', 'Events', 'Actions' and 'Transitions'. State is a representation of a current situation. An event is an input to the state. The event 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.

A Finite State Machine (FSM) model is one in which the number of states are finite. In other words the system is described using a finite number of possible states. As an example let us consider the de¬ sign of an embedded system for driver/passenger 'Seat Belt Warning' in an automotive using the FSM model. The system requirements are captured as.

- When the vehicle ignition is turned on and the seat belt is not fastened within 10 seconds of ignition ON, the system generates an alarm signal for 5 seconds.
- The Alarm is turned off when the alarm time (5 seconds) expires or if the driver/passenger fastens the belt or if the ignition switch is turned off, whichever happens first.

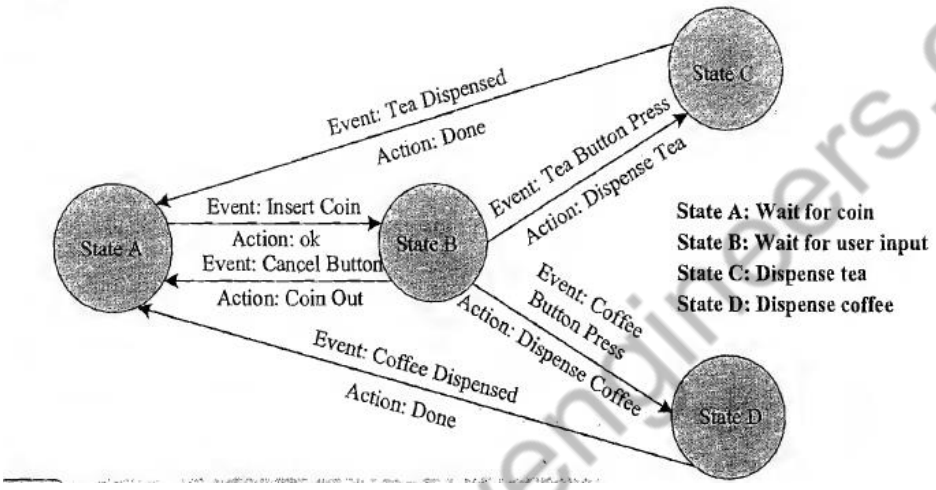
Here the states are 'Alarm Off', 'Waiting' and 'Alarm On' and the events are 'Ignition Key ON', 'Ignition Key OFF', 'Timer Expire', 'Alarm Time Expire' and 'Seat Belt ON'. Using the FSM, the system requirements can be modeled as given in Fig.



Example 1:

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

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.

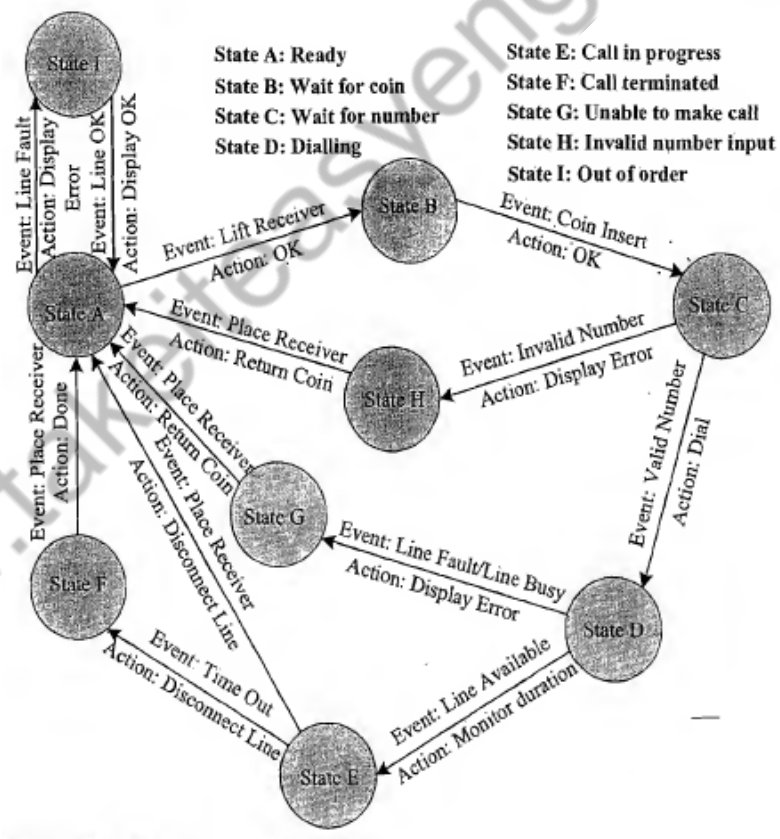


In its simplest representation, it contains four states namely; 'Wait for coin' 'Wait for User Input', 'Dispense Tea' and 'Dispense Coffee'. The event 'Insert Coin' (5 rupee coin insertion), transitions the state to 'Wait for User Input'. The system stays in this state until a user input is received from the buttons 'Cancel', 'Tea' or 'Coffee' (Tea and Coffee are the drink select button). 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 the 'Wait State' is either 'Tea' button press, or 'Coffee' button press, the state changes to 'Dispense Tea' and 'Dispense Coffee' respectively. Once the coffee/tea vending is over, the respective states transitions back to the 'Wait for Coin' state. A few modifications like adding a timeout for the 'Wait State' (Currently the 'Wait State' is infinite; it can be re-designed to a timeout based 'Wait State'. If no user input is received within the timeout period, the coin is returned back and the state automatically transitions to 'Wait for Coin' on the timeout event) 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:

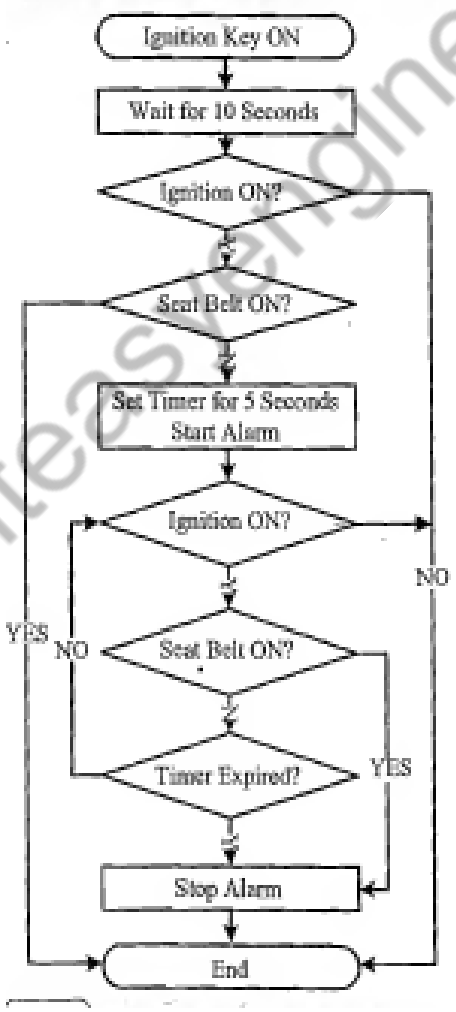
Design a coin operated public telephone unit based on FSM model for the following requirements.

1. The calling process is initiated by lifting the receiver (off-hook) ofthe 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 60 seconds and at the end of45th second, prompt for inserting another 1 rupee coin for continuing the call is initiated
5. If the user doesn't 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.



4. Sequential Program Model:

In the sequential programming Model, the functions or processing requirements are executed in sequence. It is same as the conventional procedural programming. Here the program instructions are iterated and executed conditionally and the data gets transformed through a series of operations. FSMs are good choice for sequential program modeling. Another important tool used for modeling sequential program is Flow Charts. The FSM approach represents the states, events, transitions and actions, whereas the Flow Chart models the execution flow.



5. **Object-Oriented Model**

   The object-oriented model is an object based model for modeling system requirements. It disseminates a complex software requirement into simple well defined pieces called objects. Object-oriented model brings re-usability, maintainability and productivity in system design. In the object-oriented modeling, 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. A class is an abstract description of a set of objects and it can be considered as a 'blueprint' of an object. A 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 or protected.

Embedded Firmware Design and Development

The embedded firmware is responsible for controlling the various peripherals of the embedded hardware and generating response in accordance with the functional requirements mentioned in the requirements for the particular embedded product. Firmware is considered as the master brain of the embedded system.

EMBEDDED FIRMWARE DESIGN APPROACHES:

The firmware design approaches for embedded product is purely dependent on the complexity of the functions to be performed, the speed of operation required, etc. Two basic approaches are used for embedded firmware design. They are 'Conventional Procedural Based Firmware Design' and 'Embedded Operating System (OS) Based Design'. The conventional procedural based design is also known as 'Super Loop Model'.

1. **Super Loop Model:**

The Super Loop based firmware development approach is adopted for applications that are not time critical and where the response time is not so important (embedded systems where missing deadlines are acceptable). It is very similar to a conventional procedural programming where the code is executed task by task. The task listed at the top of the program code is executed first and the tasks just below the top are executed after completing the first task. This is a true procedural one.-In a multiple task based system, each task is executed in serial in this approach. The firmware execution flow for this will be

1. Configure the common parameters and perform initialization for various hardware components memory, registers, etc.
2. Start the first task and execute it
3. Execute the second task
4. Execute the next task
5. :
6. :
7. Execute the last defined task
8. Jump back to the first task and follow the same flow

From the firmware execution sequence, it is obvious that the order in which the tasks to be executed are fixed and they are hard coded in the code itself. Also the operation is an infinite loop based approach.

Since the tasks are running inside an infinite loop, the only way to come out of the loop is either a hardware reset or an interrupt assertion.

The ‘Super loop based design' doesn’t require an operating system, since there is no need for scheduling which task is to be executed and assigning priority to each task. In a super loop based design, the priorities are fixed and the order in which the tasks to be executed are also fixed. Hence the code for performing these tasks will be residing in the code memory without an operating system image.

A typical example of a 'Super loop based' product is an electronic video game toy containing keypad and display unit. The program running inside the product may be designed in such a way that it reads-the keys to detect whether the user has given any input and if any key press is detected the graphic display is updated.

The 'Super loop based design' is simple and straight forward without any OS related overheads.

- The major drawback of this approach is that any failure in any part of a single task will affect the total system. If the program hangs up at some point while executing a task, it will remain there forever and ultimately the product stops functioning.
- Another major drawback of the 'Super loop' design approach is the lack of real timeliness. If the number of tasks to be executed within an application increases, the time at which each task is repeated also increases.

2. **The Embedded Operating System (OS) Based Approach**

The Operating System (OS) based approach contains operating systems, which can be either a General Purpose Operating System (GPOS) or a Real Time Operating System (RTOS) to host the user written application firmware. The General Purpose OS (GPOS) based design is very similar to a conventional PC based application development where the device contains an operating system (Windows/Unix/ Linux, etc. for Desktop PCs) and you will be creating and running user applications on top of it.

Real Time Operating System (RTOS) based design approach is employed in embedded products demanding Real-time response. RTOS respond in a timely and predictable manner to events. Real Time operating system contains a Real Time kernel responsible for performing pre-emptive multitasking, scheduler for scheduling tasks, multiple threads, etc. A Real Time Operating System (RTOS) allows flexible scheduling of system resources like the CPU and memory and offers some way to communicate between tasks.

Embedded Firmware Development Languages

1. **Assembly Language based Development:**

Assembly language programming is the task of writing processor specific machine code in mne¬ monic 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 and the Operands provide the data and information required to perform the action specified by the Opcode.

Assembly language instructions are written one per line. A machine code program thus 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 given below

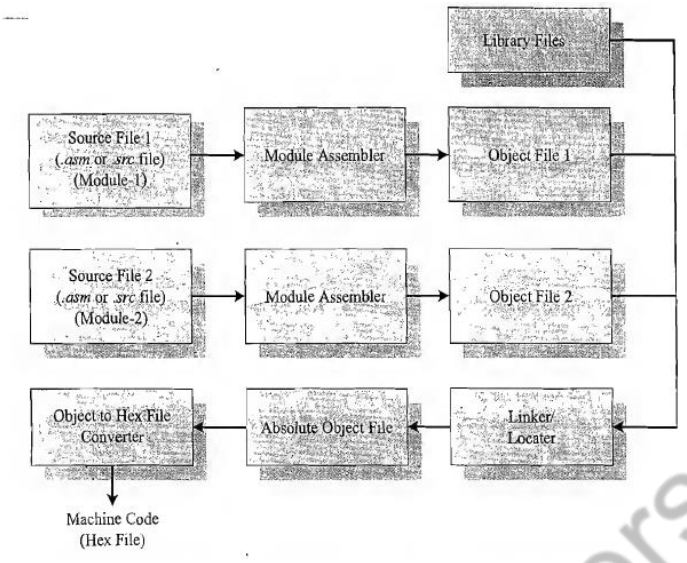
LABEL OPCODE OPERAND COMMENTS

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.

Source File to Object File Translation:

Translation of assembly code to machine code is performed by assembler.

The various steps involved in the conversion of a program written in assembly language to corresponding binary file/machine language is illustrated in Fig



Each source module is written in Assembly and is stored as .src file or .asm file. Each file can be assembled separately to examine the syntax errors and incorrect assembly instructions. On successful assembling of each .src/.asm file a corresponding object file is created with extension '.obj. The object file does not contain the absolute address of where the generated code needs to be placed on the program memory and hence it is called a re-locatable segment. It can be placed at any code memory location and it is the responsibility of the linker/locater to assign absolute address for this module.

- **Library File Creation and Usage:**

Libraries are specially formatted, ordered program col¬ lections of object modules that may be used by the linker at a later time. When the linker processes a li¬ brary, 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.

- **Linker and Locater:**

Linker and Locater 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.

- **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.

HEX files are ASCII files that contain a hexadecimal representation of target application. Hex file is created from the final 'Absolute Object File' using the Object to Hex File Converter utility.

Advantages of Assembly Language Based Development

a) Efficient Code Memory and Data Memory Usage (Memory Optimisation)

b) High Performance

c) Low Level Hardware Access

d) Code Reverse Engineering

Disadvantages of Assembly Language Based Development

a) High Development Time

b) Developer Dependency

c) Non-Portable

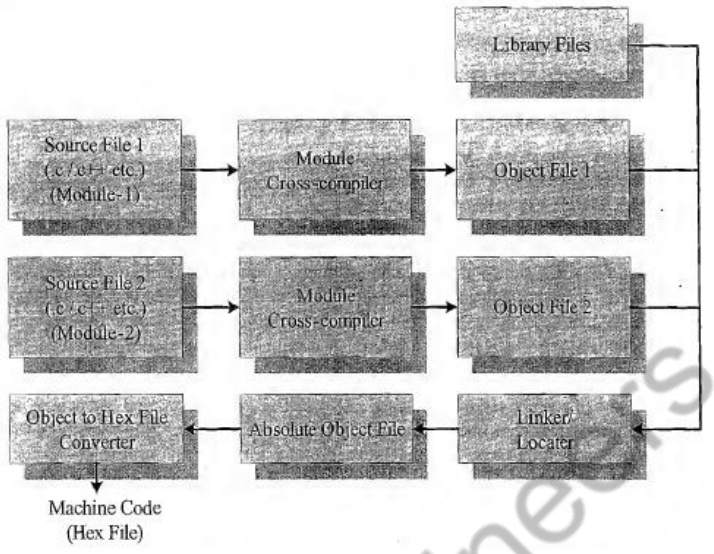
2. High Level Language Based Development

Assembly language based programming is highly time consuming, tedious and requires skilled programmers with sound knowledge of the target processor architecture. Also applications developed in Assembly language are non-portable.

Any high level language (like C, C++ or Java) with a supported cross compiler for the target processor can be used for embedded firmware development. The most commonly used high level language for embedded firmware application development is 'C'.

The various steps involved in high level language based embedded firmware development is same as that of assembly language based development except that the conversion of source file written in high level language to object file is done by a cross-compiler, whereas in Assembly language based development it is carried out by an assembler. The various

steps involved in the conversion of a program written in high level language to corresponding binary file/machine language is illustrated in Fig.



Advantages of High Level Language based Development

a) Reduced Development Time
b) Developer Independency
c) Portability

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. Programming in 'Embedded C' is quite different from conventional Desktop application development using 'C' language for a particular OS platform. Desktop computers contain working memory in the range of Megabytes (Nowadays Giga bytes) and storage memory in the range of Giga bytes. 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. This is not the case for embedded application developers. Almost all embedded systems are limited in both storage and working memory resources.

- **'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).
- 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.
- A software program called ‘Cross-compiler’ is used for the conver¬ sion of programs written in Embedded ‘C’ to target processor/controller specific instructions (machine language).

- **Compiler vs. 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.

- Cross-compilers are 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, ARM etc). Keil is an example for cross-compiler.