ATME College of Engineering

Embedded Systems-BEC601

Module-2

Embedded System Design Concepts

Pradeep Kumar Y
Assistant Professor
Dept.of ECE

Topics

1. Characteristics of Embedded Systems
2. Quality Attributes of Embedded Systems
   i. Operational quality attributes
   ii. Non-operational quality attributes
3. Application specific Embedded System
4. Domain specific Embedded System

Topics

5. Hardware Software Co-Design and Program Modeling
   i. Fundamental Issues in Hw-Sw Co-Design
   ii. Computational Models in Embedded Design
6. Embedded Firmware Design Approaches
7. Embedded Firmware Development Languages

Text Book

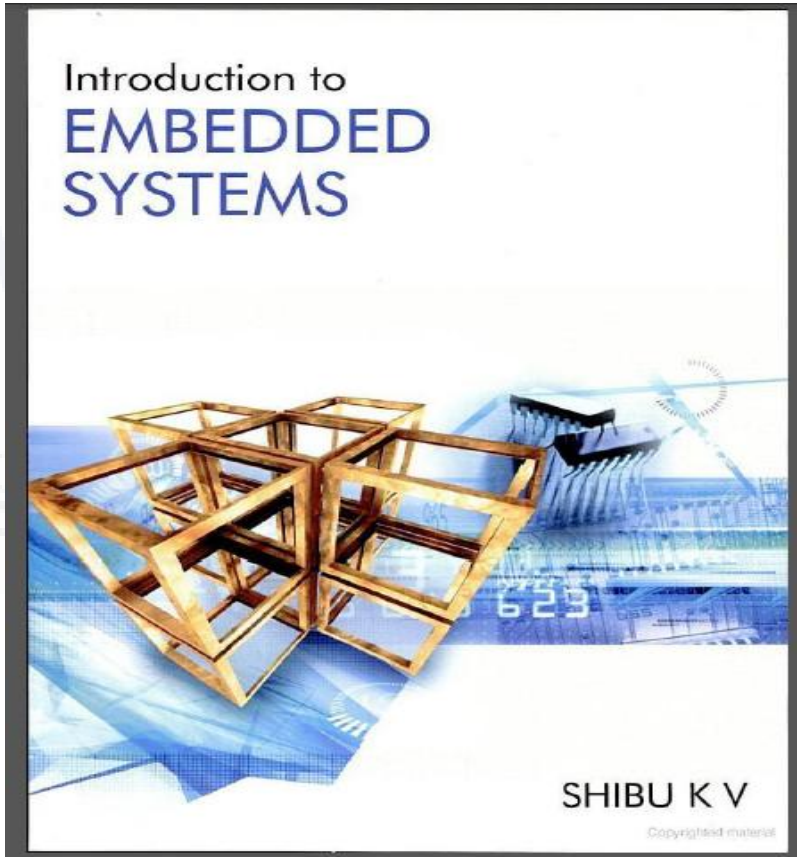
Shibu K V, "**Introduction to Embedded Systems**"
Tata McGraw Hill Education Private Limited, 2nd Edition

Ch-3

Ch-4 (4.1, 4.2.1 and 4.2.2 only)

Ch-7 (Sections 7.1, 7.2 only)

Ch-9 (Sections 9.1, 9.2, 9.3.1, 9.3.2 only)



Characteristics of Embedded System

Each Embedded System possess a set of characteristics which are unique to it.

Some important characteristics of embedded systems are:

- ❑Application & Domain Specific
- ❑Reactive & Real Time
- ❑Operates in ‘harsh’ environment
- ❑Distributed
- ❑Small size and Weight
- ❑Power Concerns

Application & Domain Specific- Characteristics of Embedded System

- An embedded system is designed to perform **specific function** only.

Ex. *Air conditioner's* embedded control unit, cannot replace *microwave oven*

- Certain embedded systems are **specific to a domain.**

Ex. A hearing aid is an application that belongs to the domain of signal processing another control unit designed to serve another domain like consumer electronics

Reactive and Real Time - Characteristics of Embedded System

- Embedded Systems are in constant interaction with the real world through sensors and user defined input devices
- Event captured by sensors and input devices

  Ex. An air conditioner

- Real time System operation-Timing behavior of the system should be deterministic

  Ex. Flight control systems, Antilock Brake Systems (ABS)

Operation in Harsh Environment - Characteristics of Embedded System

- Embedded systems may deployed in harsh environments like a dusty one or a high temperature zone or an area subject to vibrations and shock.
- High temperature grade systems
- Using of shock absorption technique
- *Power supply fluctuations*
- *Corrosion*
- *Component aging*

Distributed- Characteristics of Embedded System

- 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: **Automatic Teller Machine (ATM)**

  Card reader embedded unit + *Transaction unit + a currency counter + printer unit*

Small size and Weight- Characteristics of Embedded System

- An embedded system that is small in size and has light weight will be more popular.
- Convenient to handle a compact devices than a bulky product.

Power Concerns - Characteristics of Embedded System

- It is desirable that the power utilization and heat dissipation of any embedded system be low
- Production of high amount of power needs cooling requirements

  Cooling fans, Heat Sinks
- Select the design according to the low power components like low dropout regulators, and controllers/processors with ***power saving*** modes
- The more the power consumption the less is the battery life

Quality Attributes of Embedded Systems

- Non-functional requirements that needs to be addressed and documented in the design of an embedded system

❑Operational Quality Attributes

Refers to the relevant quality attributes related to an embedded system when it is in the operational mode or '***online***' mode

❑Non-Operational Quality Attributes

The Quality attributes that needs to be addressed for the product 'not' on the basis of operational aspects are grouped under this category

Operational Quality Attributes

- Response
- Throughput
- Reliability
- Maintainability
- Security
- Safety

Operational Quality Attribute- Response

- Response is a ***measure of quickness*** of the system.
- It gives an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.
- Ex. 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.
- It is not necessary that all embedded systems should be Real Time in response.
- For example, the response time requirement for an electronic toy is not at all time-critical.

Operational Quality Attribute- Throughput

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

Operational Quality Attribute- Reliability

- Reliability is a measure of how much percentage you rely upon the proper functioning of the system or what is the % susceptibility of the system to failure.
- Mean Time Between Failures (**MTBF**) and Mean Time To Repair (**MTTR**) are the terms used in defining system reliability.
- **MTBF** gives the **frequency of failures** in hours/weeks/months.
- **MTTR** specifies **how long the system is allowed to be out of order** following a failure.
- For an embedded system with critical application need, it should be of the order of minutes.

Operational Quality Attribute- 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.

Operational Quality Attribute- Maintainability

- Maintainability can be classified into two types:

  1. Scheduled or Periodic Maintenance (Preventive Maintenance)
  2. Maintenance to Unexpected Failures (Corrective Maintenance)

  Ex: Inkjet Printer

- In any embedded system design, the ideal value for availability is expressed as

$$Ai = MTBF/(MTBF+MTTR)$$

Operational Quality Attribute- Security

- ***'Confidentially', 'Integrity',*** and ***'Availability'*** are three major measures of information security.
- **Confidentially** deals with the protection of data and application from unauthorized disclosure.
- **Integrity** deals with the protection of data and application from unauthorized modification.
- **Availability** deals with protection of data and application from unauthorized users.

Operational Quality Attribute- Safety

- Safety deals with the possible damages that can happen to the operators, public and the environment due to the breakdown of an embedded system or due to the emission of radioactive or hazardous materials from the embedded products.
- The breakdown of an embedded system may occur due to a hardware failure or a firmware failure.
- Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of the damages to an acceptable level.

Non-Operational Quality Attributes

The quality attributes that needs to be addressed for the product ‘not’ on the basic of operational aspects.

- ❑Testability & Debug-ability
- ❑Evolvability
- ❑Portability
- ❑Time to Prototype and Market
- ❑Per Unit and Total Cost

Non-Operational Quality Attributes- Testability & Debug-ability

- Testability deals with how easily one can test his/her design, application and by which means he/she can test it.
- For an embedded product, testability is applicable to both the embedded hardware and firmware.
- Debug-ability is a means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system.
- Debug-ability has two aspects in the embedded system development context, namely, hardware level debugging and firmware level debugging.

Non-Operational Quality Attributes- Evolvability

- Evolvability is a term which is closely related to Biology.
- Evolvability is referred as the non-heritable variation.
- For an embedded system, the quality attribute ‘Evolvability’ refers to the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.

Non-Operational Quality Attributes- Portability

- Portability is a measure of '*system independence*'.
- An embedded product can be called portable if it is capable of functioning in various environments, target processors/controllers and embedded operating systems.
- A standard embedded product should always be flexible and portable.

Non-Operational Quality Attributes- Time to Prototype and Market

- Time-to-market is the time elapsed between the conceptualization of a product and the time at which the product is ready for selling (for commercial product) or use (for non-commercial products).
- The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product.
- Product prototyping helps a lot in reducing time-to-market

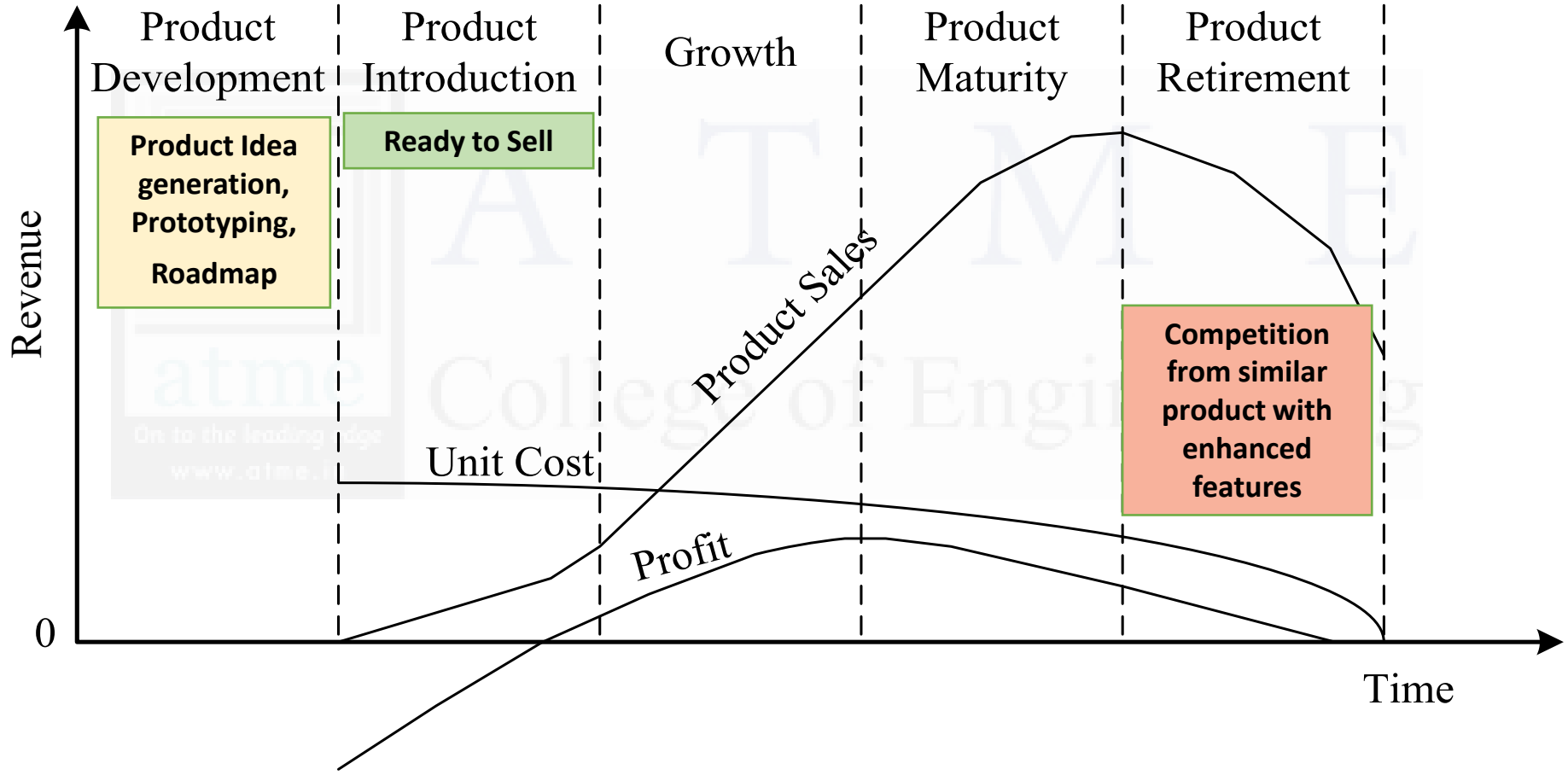
Non-Operational Quality Attributes- Per Unit and Total Cost

- Time-to-market is the time elapsed between the conceptualization of a product and the time at which the product is ready for selling (for commercial product) or use (for non-commercial products).
- The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product.
- Product prototyping helps a lot in reducing time-to-market

Non-Operational Quality Attributes- Per Unit and Total Cost

- 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.
- When the product is introduced in the market, for the initial period the sales and revenue will be low.

The Product Life Cycle (PLC) Curve

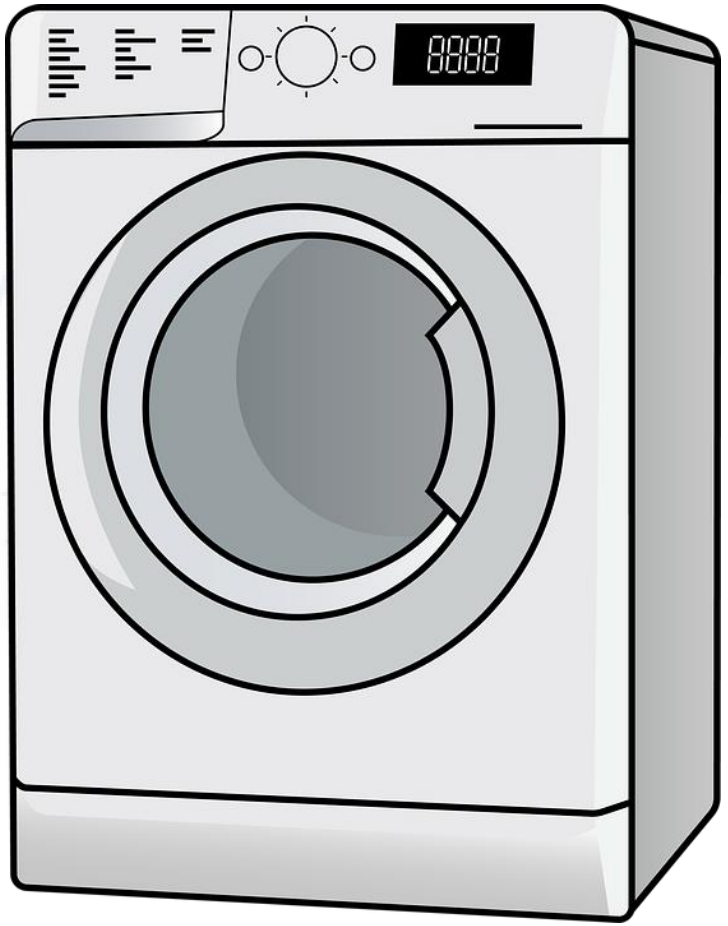


Embedded System

- Embedded systems are application specific : Washing Machine
- Embedded systems are domain specific : automotive

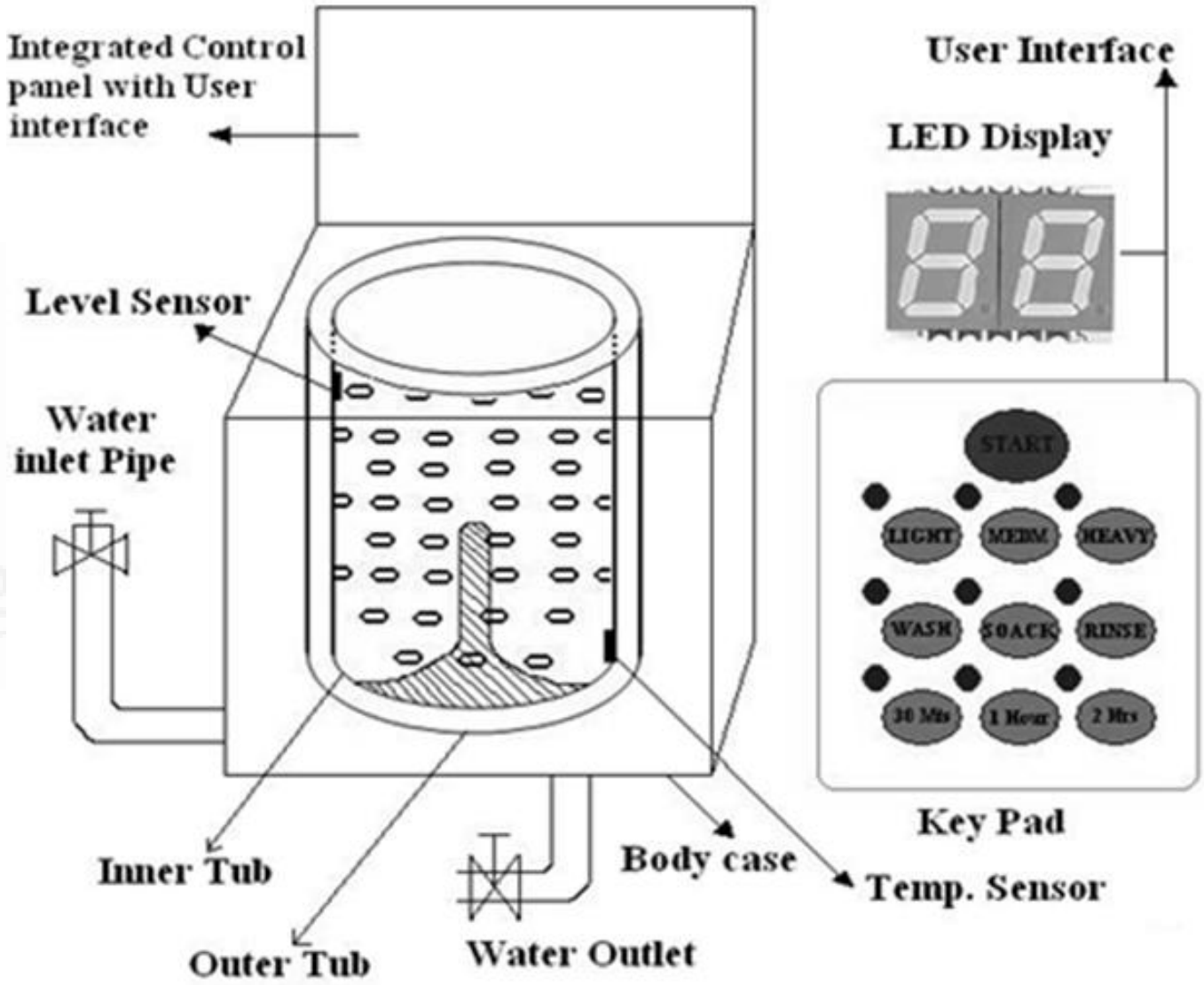
Application Specific Embedded System- Washing Machine

- Washing machine is a typical example of an embedded system providing extensive support in home automation applications.
- An embedded system contains sensors, actuators, control unit and application- specific user interfaces like keyboards, display units, etc.



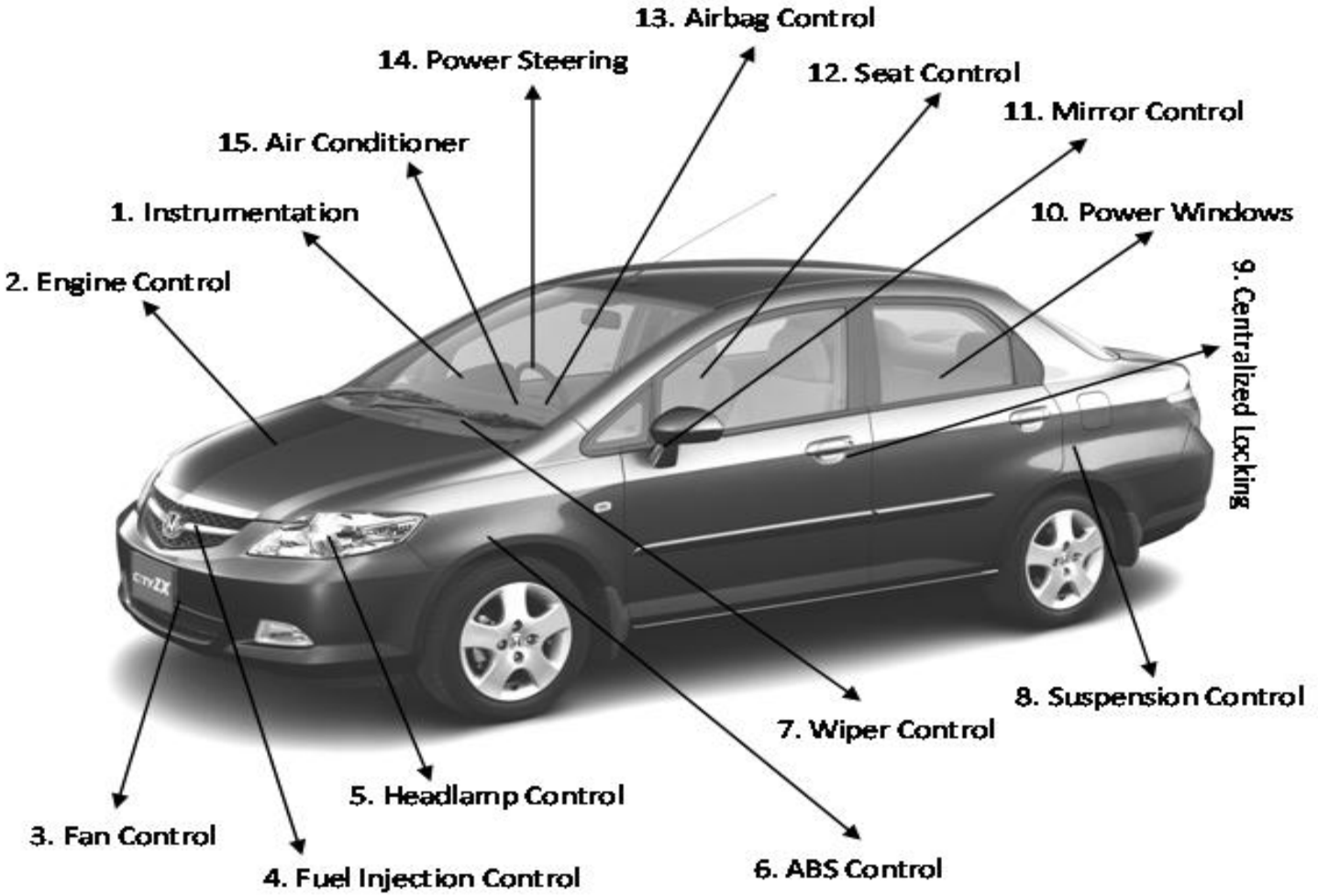
Application Specific Embedded System- Washing Machine

- ✓ Extensively used in Home Automation for washing and drying clothes
- ✓ Contains User Interface units (I/O) like Keypads, Display unit, LEDs for accepting user inputs and providing visual indications
- ✓ Contains sensors like, water level sensor, temperature sensor etc.
- ✓ Contains actuators like spin and agitation control motor units
- ✓ Contains an integrates embedded controller for controlling the washing operations
- ✓ Sensors, actuators and I/O devices are interfaced to the I/O subsystem of the embedded control unit
- ✓ Specifically designed for serving the application 'Wash & Rinse' of clothes and cannot be used for any other applications



Application Specific Embedded System- Washing Machine

Domain Specific Embedded System- Automotive



Generally built around

- ✓ Digital Signal Processors
- ✓ Application Specific Instruction Set Processors (like Atmel Automotive AVR)
- ✓ General purpose processors/Controllers
- ✓ System on Chips
- ✓ Programmable Logic Devices or
- ✓ Application Specific Integrated/Standard Products (ASIC/ASSP) or
- ✓ A combination of these

Automotive Embedded Control units are generally known as ***Electronic Control Units*** (ECUs)

Domain Specific Embedded System- Automotive

❑ High speed Electronic Control Units (HECUs):

- ✓ Deployed in critical control units ***requiring fast response***.
- ✓ They include Fuel Injection Systems, Antilock Brake Systems, Engine Control, Electronic throttle, Steering Controls, Transmission Control unit and Central Control unit

❑ Low speed Electronic Control Units (LECUs)

- ✓ Deployed in applications where ***response time is not so critical***.
- ✓ They generally are built around low cost microprocessors/microcontrollers and Digital Signal Processors.
- ✓ Audio controllers, Passenger and driver door locks, Door glass controls (Power Windows), Wiper control, Mirror Control, Seat control systems, Head lamp and tail lamp controls, Sun roof control unit etc. are examples of LECUs

Domain Specific Embedded System- Automotive

Automotive Communication Buses

Controller Area Network (CAN)

- ✓Originally proposed by Robert Bosch, pioneers in the Automotive Embedded Solution providers.
- ✓Supports medium speed (ISO11519-Class B with data rates up to 125 Kbps) and high speed (ISO11898 Class C with data rates up to 1Mbps) data transfer.
- ✓An event driven protocol interface with support for error handling in data transmission.
- ✓Generally employed in safety system like Airbag control; power train systems like Engine control and Antilock Brake Systems (ABS) and navigation systems like GPS.

Domain Specific Embedded System- Automotive

Automotive Communication Buses

Local Interconnect Network (LIN)

- ✓A single master multiple slave (up to 16 independent slave nodes) communication interface
- ✓Low speed, single wire communication interface with support for data rates up to 20Kbps
- ✓Mainly used for sensor/actuator interfacing
- ✓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
- ✓Employed in applications like mirror controls, fan controls, seat positioning controls, window controls, and position controls where response time is not a critical issue

Domain Specific Embedded System- Automotive

Automotive Communication Buses

Media Oriented System Transport (MOST) bus

- ✓Targeted for automotive Audio Video equipment interfacing
- ✓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 fiber cable connected between the Electrical Optical Converter (EOC) and Optical Electrical Converter (OEC), which would translate into the optical cable MOST bus

Hardware Software Co-Design and Program Modeling

- In the traditional embedded system development approach, the hardware software partitioning is done at an early stage.
- 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 Software Co-Design and Program Modeling

- 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.
- System level processing requirements are then transferred into functions which can be simulated and verified against performance and functionality.

Hardware Software Co-Design and Program Modeling

- 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.

Hardware Software Co-Design and Program Modeling

- 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 behavior is usually captured using computational models.
- The models representing the software processing requirements are translated into firmware implementation using programming languages.

Fundamental issues in hardware software co-design

- Selecting the Model
- Selecting the Architecture
- Selecting the Language
- Partitioning System Requirements into Hardware and Software

Fundamental issues in hardware software co-design

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.
- Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design.

Fundamental issues in hardware software co-design

Selecting the Architecture

A model only captures the system characteristics and does not provide information on 'how the system can be manufactured?'.

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.

Fundamental issues in hardware software co-design

Selecting the Architecture

The commonly used architectures in system design are:

- 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

Fundamental issues in hardware software co-design

Selecting the Architecture- *Controller Architecture*

- The 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.

Fundamental issues in hardware software co-design

Selecting the Architecture- *Datapath Architecture*

- 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.
- The datapath may contain registers, counters, register files, memories and ports along with high speed arithmetic units.
- Ports connect the datapath to multiple buses.
- Most of the time the arithmetic units are connected in parallel with pipelining support for bringing high performance

Fundamental issues in hardware software co-design

Selecting the Architecture- *Finite State Machine Datapath (FSMD)*

- It 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.
- Normally the datapath 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.

Fundamental issues in hardware software co-design

Selecting the Architecture- *Complex Instruction Set Computing (CISC) architecture*

- It 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.
- Reading a register value and comparing it with a given value and then transfer the program execution to a new address location is done using the CJNE instruction for 8051 ISA.
- The use of a single complex instruction in place of multiple simple instructions greatly reduces the program memory access and program memory size requirement.
- However it requires additional silicon for implementing microcode decoder for decoding the CISC instruction. The datapath for the CISC processor is complex

Fundamental issues in hardware software co-design

Selecting the Architecture- *Reduced Instruction Set Computing (RISC) architecture*

- It uses instruction set representing simple operations
- It requires the execution of multiple RISC instructions to perform a complex operation.
- The datapath 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.

Fundamental issues in hardware software co-design

Selecting the Architecture- *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.

Fundamental issues in hardware software co-design

Selecting the Architecture- *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.

Fundamental issues in hardware software co-design

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 model.
- 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.

Fundamental issues in hardware software co-design

Selecting the Language

- 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.
- The only pre-requisite in selecting a programming language for capturing a model is that the language should capture the model easily.

Fundamental issues in hardware software co-design

Partitioning System Requirements into Hardware and Software

- 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 Design

The commonly used computational models in embedded system design are:

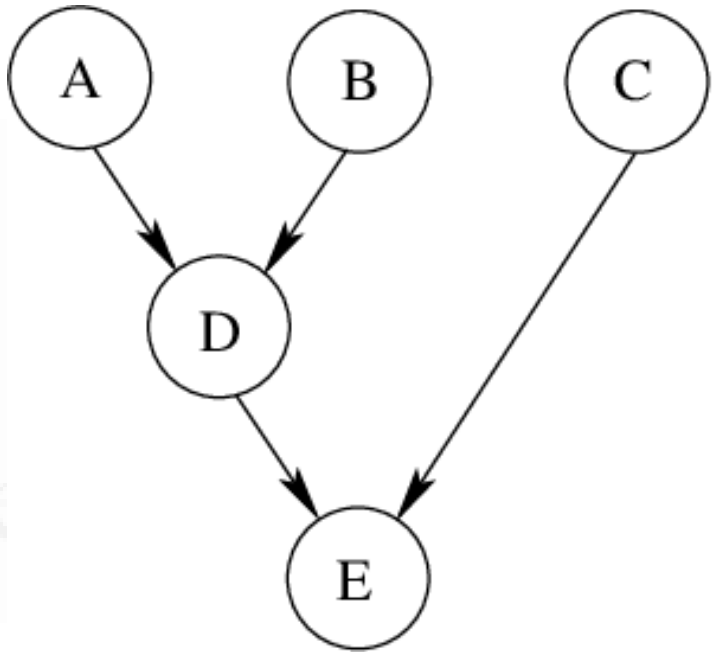
- Data Flow Graph Model
- Control Data Flow Graph Model
- State Machine Model
- Sequential Program Model
- Concurrent/Communicating Process Model
- Object-Oriented Model

Data Flow Graph/Diagram (DFG) model

- The Data Flow Graph (DFG) model translates the data processing requirements into a data flow graph.
- It is a data driven model in which the program execution is determined by data.
- This model emphasises on the data and operations on the data which transforms the input data to output data.
- Embedded applications which are computational intensive and data driven are modeled using the DFG model.
- DSP applications are typical examples for it.

Data Flow Graph/Diagram (DFG) model

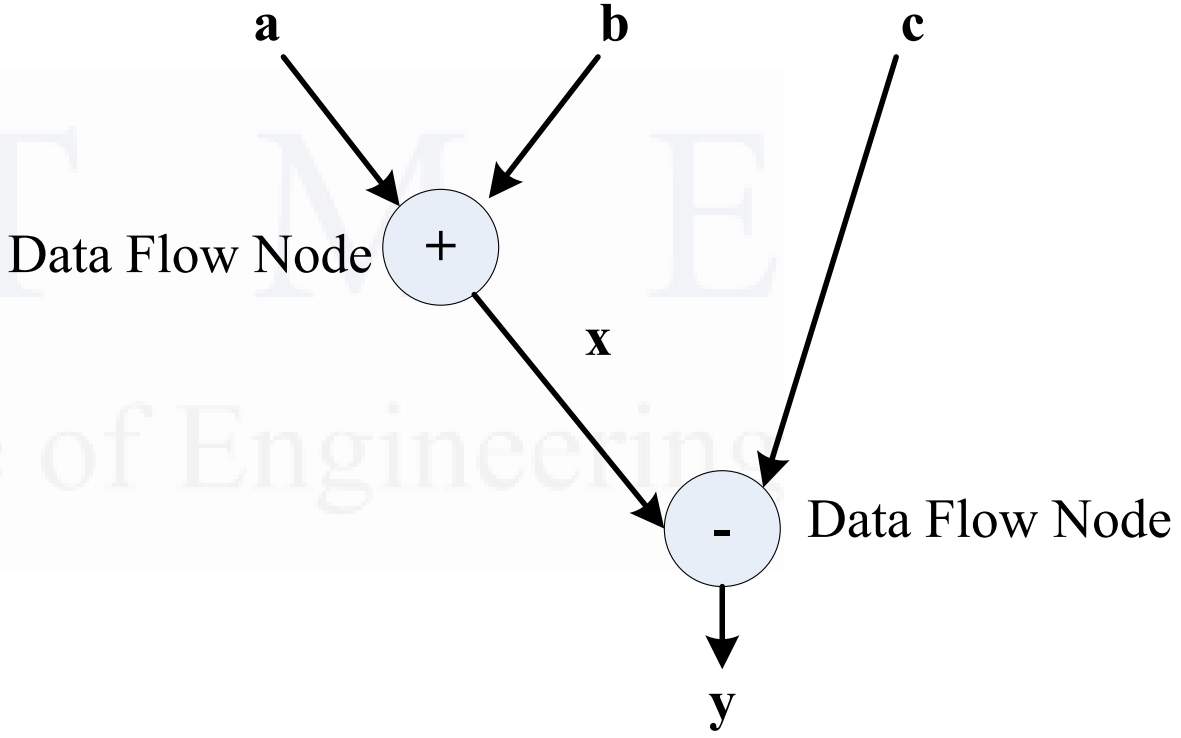
- Data Flow Graph (DFG) is a visual model in which the operation on the data (process) is represented using a block (circle) and data flow is represented using arrows.
- An inward arrow to the process (circle) represents input data and an outward arrow from the process (circle) represents output data in DFG notation.



Data Flow Graph/Diagram (DFG) model

$x = a + b$ and $y = x - c$

- 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



Control Data Flow Graph/Diagram (CDFG) model

- 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 modelling 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.

Control Data Flow Graph/Diagram (CDFG) model

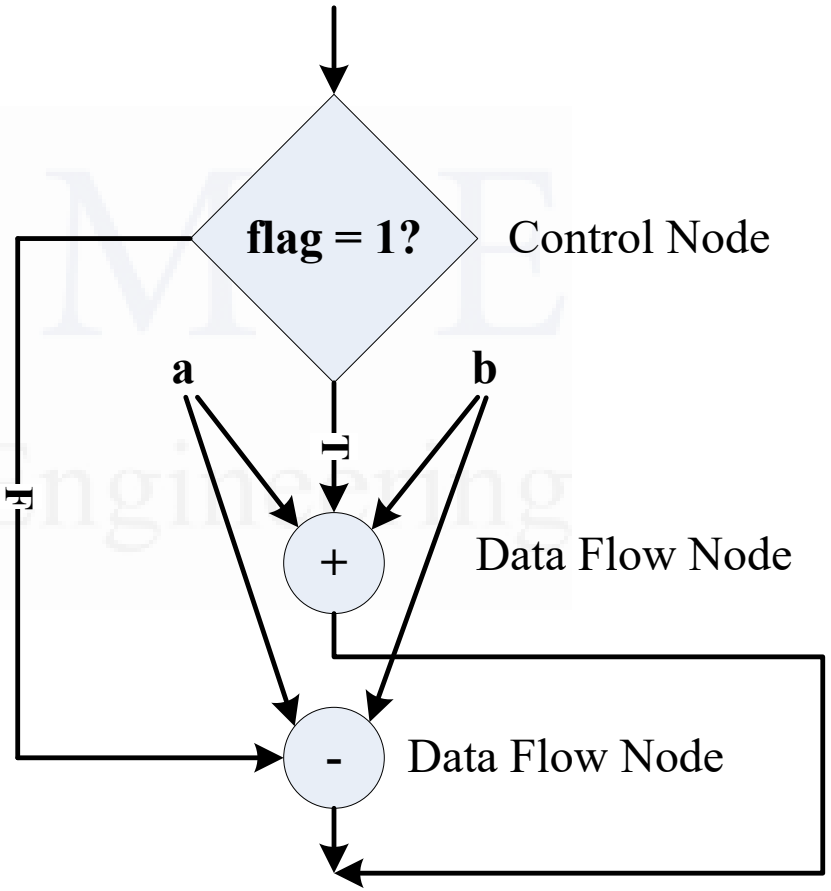
if flag = 1

$x = a + b;$

else

$x = a - b;$

- CDFG translates the requirement, which is modelled to a concurrent process model.
- The decision on which process is to be executed is determined by the control node.



Control Data Flow Graph/Diagram (CDFG) model

- Capturing of image and storing it in the format selected (bmp, jpg, tiff, etc.) in a digital camera is a typical example of an application that can be modeled with CDFG

Computational Models in Embedded Design

The commonly used computational models in embedded system design are:

- Data Flow Graph Model
- Control Data Flow Graph Model
- State Machine Model
- Sequential Program Model
- Concurrent/Communicating Process Model
- Object-Oriented Model

State Machine Model

- The State Machine Model is used for modelling *reactive or event-driven embedded systems* whose processing behaviour are 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 behaviour 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.

State Machine Model

- A Finite State Machine (FSM) model is one in which the number of states are finite.
- The system is described using a finite number of possible states.

Design of an embedded system for driver/passenger

'Seat Belt Warning' in an automotive using the FSM model.

The system requirements are captured as.

1. **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.**
2. **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..**

State Machine Model

States

'Alarm Off'

'Waiting'

'Alarm On'

The events are

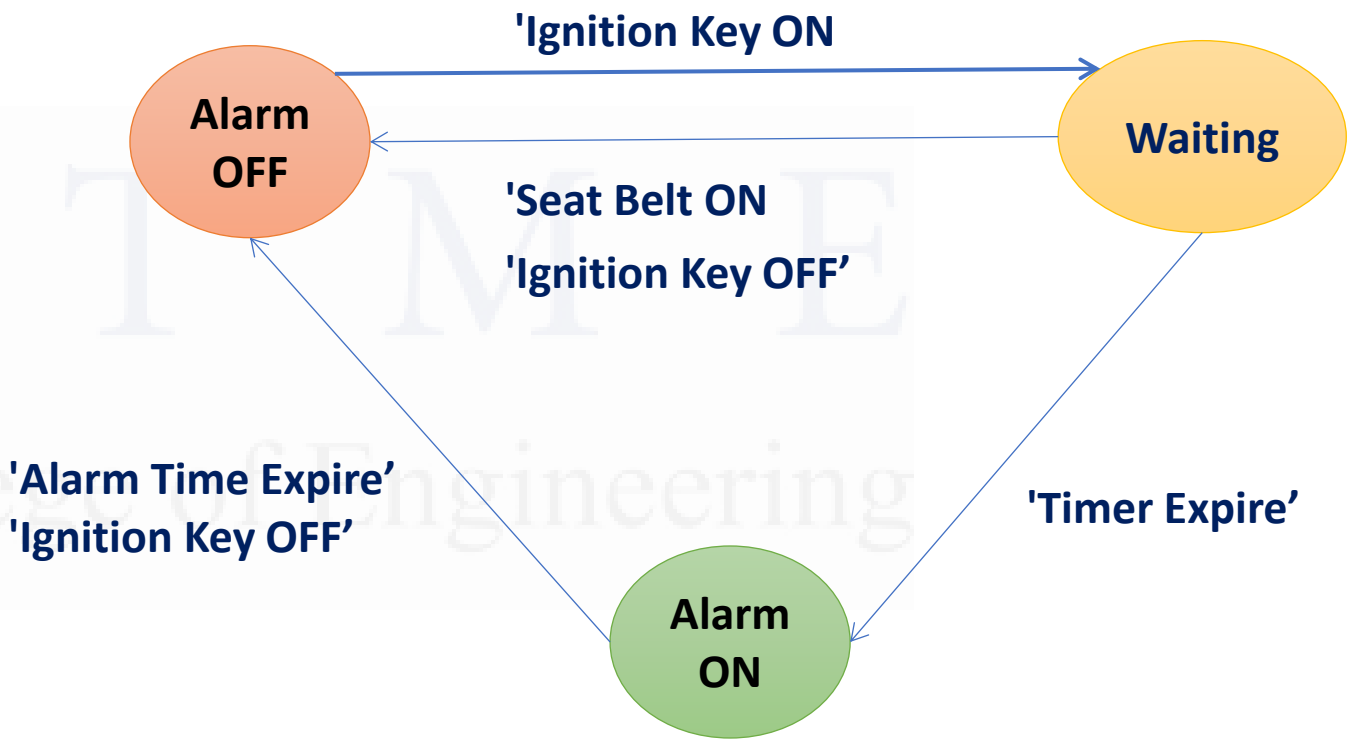
'Ignition Key ON'

'Ignition Key OFF'

'Timer Expire'

'Alarm Time Expire'

'Seat Belt ON



FSM Model for Automatic Seat Belt Warning System

State Machine Model

States

'Alarm Off'

'Waiting'

'Alarm On'

The events are

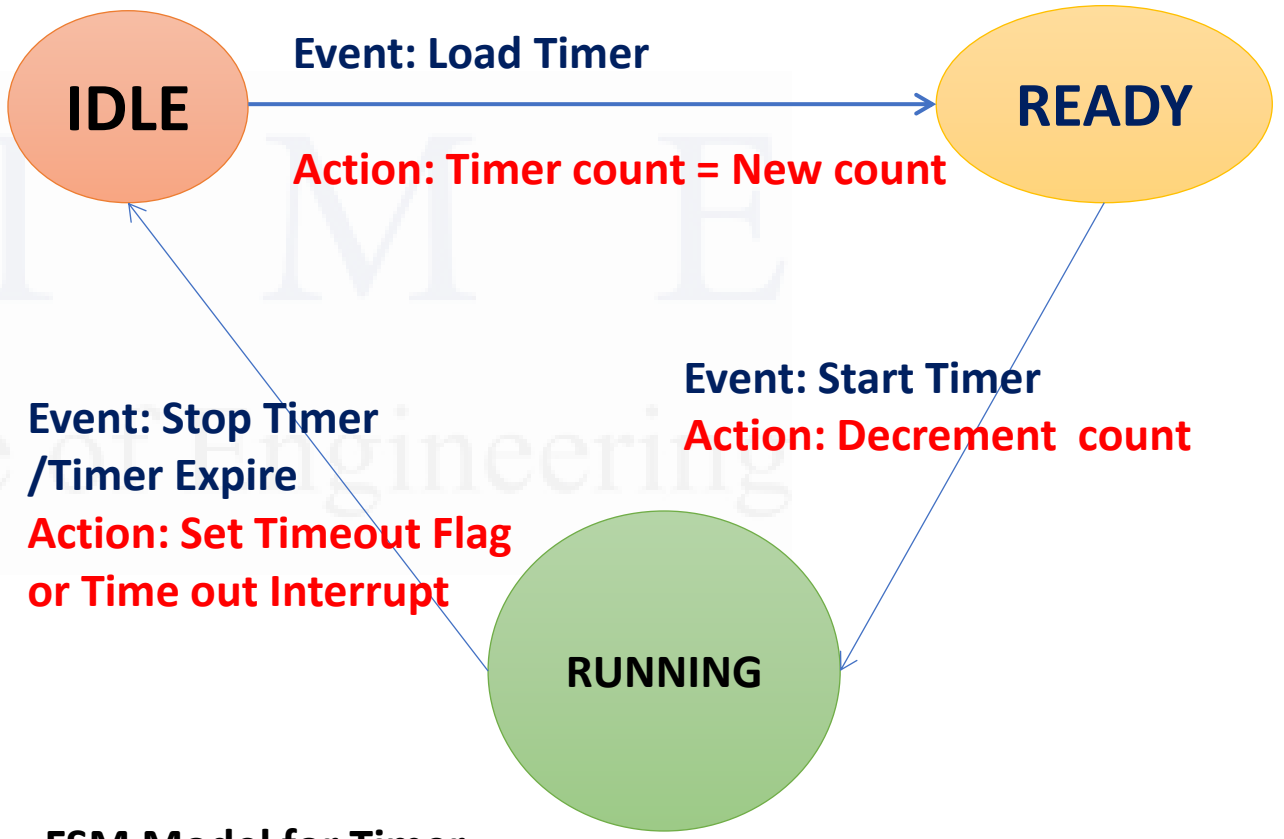
'Ignition Key ON'

'Ignition Key OFF'

'Timer Expire'

'Alarm Time Expire'

'Seat Belt ON



FSM Model for Timer

State Machine Model-FSM Model 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.

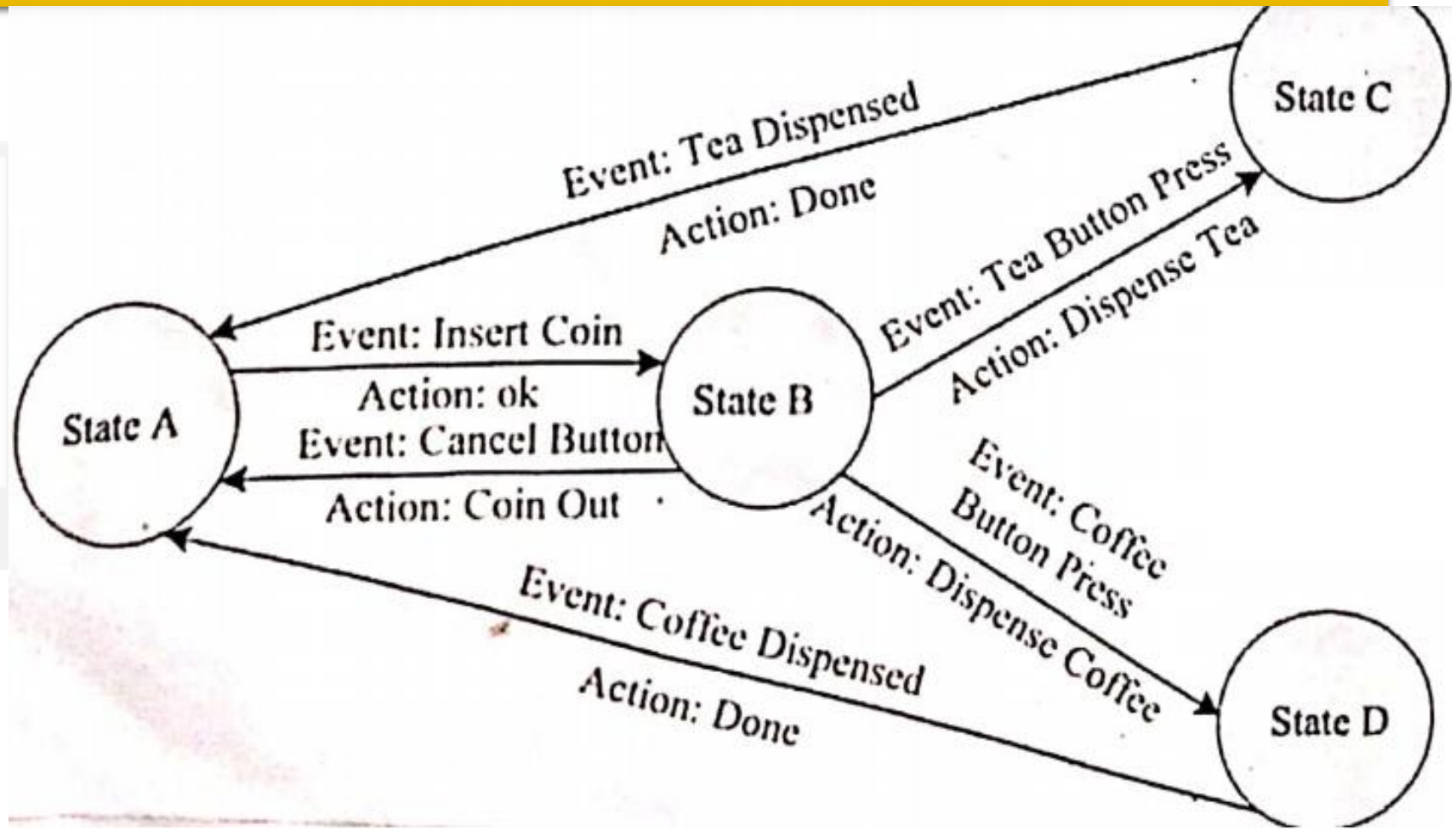
Solution

The FSM Model contains four states

1. 'Wait for coin'
2. 'Wait for User Input'
3. 'Dispense Tea'
4. 'Dispense Coffee'

State Machine Model-FSM Model Example 1 : Automatic tea/coffee vending machine

A. 'Wait for coin'
B. 'Wait for User Input'
C. 'Dispense Tea'
D. 'Dispense Coffee'



Computational Models in Embedded Design

The commonly used computational models in embedded system design are:

- Data Flow Graph Model
- Control Data Flow Graph Model
- State Machine Model
- Sequential Program Model
- Concurrent/Communicating Process Model
- Object-Oriented Model

State Machine Model-FSM Model 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) 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 60 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 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.

State Machine Model-FSM Model Example 2

State A: Ready

State B : Wait for coin

State C : Waiting for number

State D : Dialing

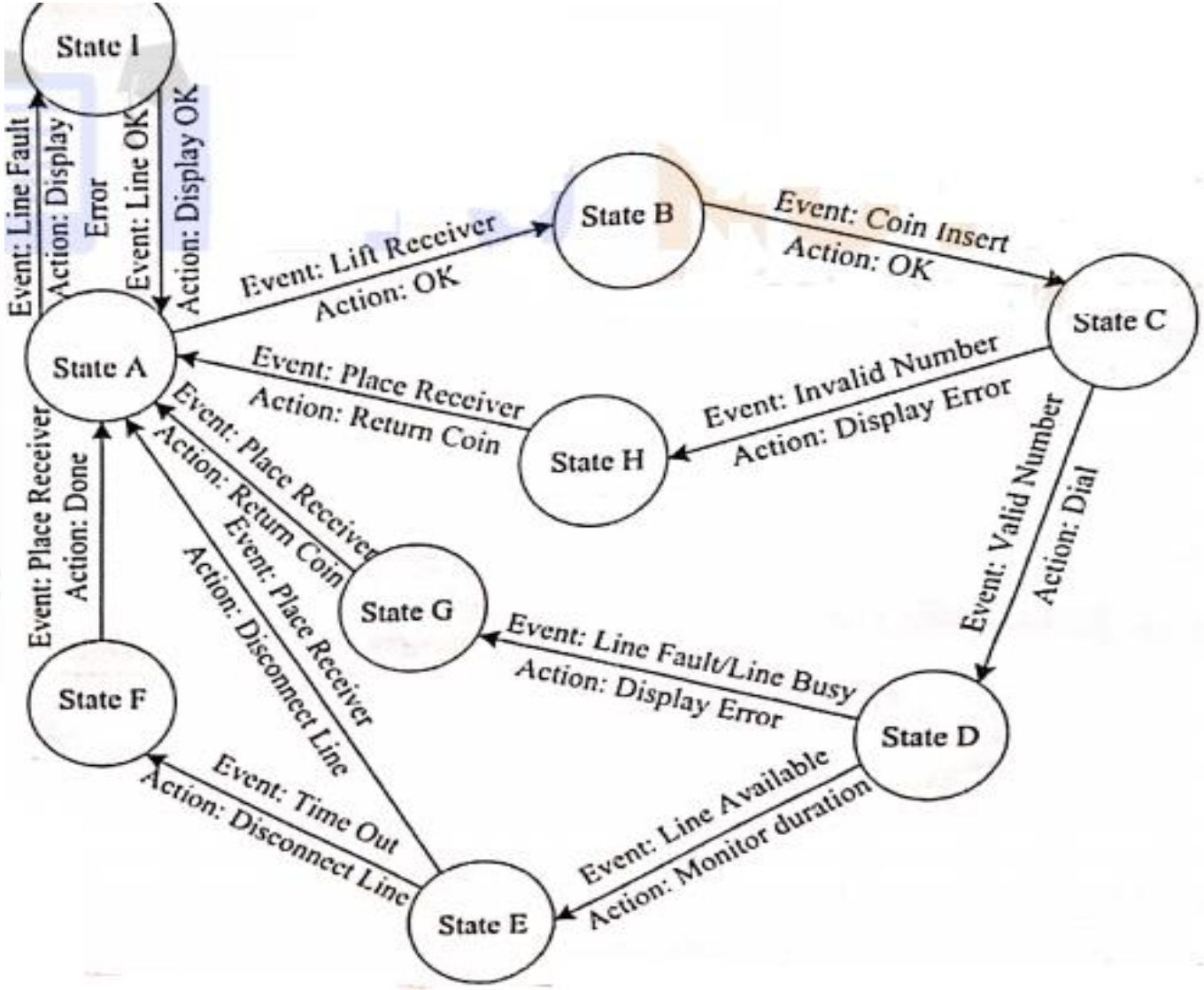
State E : Call in progress

State F : Call Terminated

State G : Unable to make a call

State H : Invalid number input

State I : Out of order

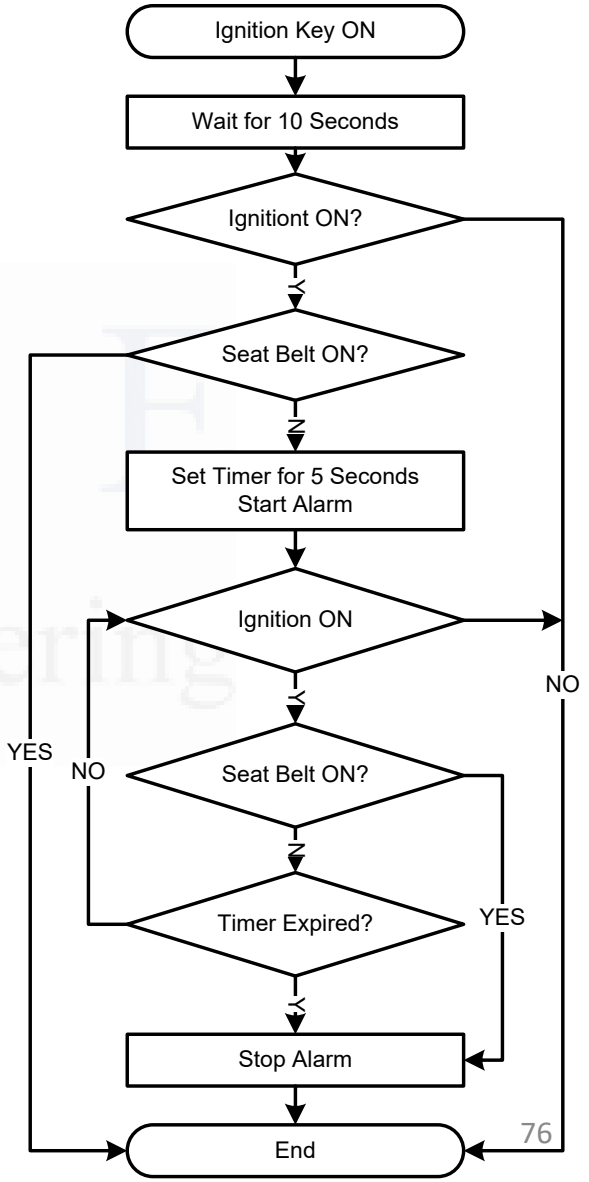


Sequential Program Model

- In the Sequential Program 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.
- Finite State Machines (FSMs) and Flow Charts are used for modelling sequential program.

Sequential Program Model

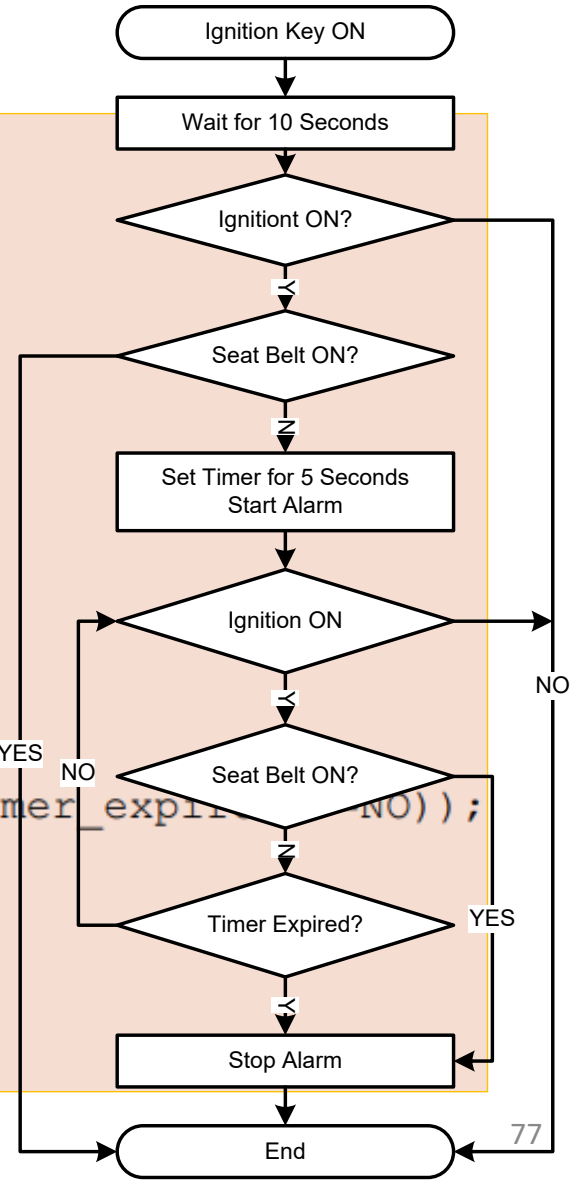
The FSM approach represents the states, events, transitions and actions, whereas the Flow Chart models the execution flow



Flow chart approach for modelling the 'Seat Belt Warning' system

Sequential Program Model

```
#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()==ON)&&(timer_expi    NO));
   stop_alarm();
   }
  }
 }
}
```



ATME College of Engineering

Embedded Systems-18EC62

Module-4

Topic: Embedded Firmware Design Approaches

Pradeep Kumar Y
Assistant Professor
Dept.of ECE

Concurrent/Communicating Process Model

- The concurrent or communicating process model models concurrently executing tasks/processes.
- It is easier to implement certain requirements in concurrent processing model than the conventional sequential execution.
- Sequential execution leads to a single sequential execution of task and thereby leads to poor processor utilization, when the task involves I/O waiting, sleeping for specified duration etc.
- If the task is split into multiple subtasks, it is possible to tackle the CPU usage effectively by switching the task execution, when the subtask under execution goes to a wait or sleep mode.

Concurrent/Communicating Process Model

- However, concurrent processing model requires additional overheads in task scheduling, task synchronization and communication.
- Example: consider the implementation of the **'Seat Belt Warning System'** using concurrent processing model.

The tasks can split into:

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

Concurrent/Communicating Process Model

- The tasks cannot be executed them randomly or sequentially.
- Need to synchronize their execution through some mechanism.
- Example, the alarm control task is executed only when the wait timer is expired and if the ignition key is in the ON state and seat belt is in the OFF state.

Events to indicate these scenarios.

The ***wait_timer_expire*** event is associated with the timer task event and it will be in the reset state initially and it is set when the timer expires.

Similarly, events ***ignition_on*** and ***ignition_off*** are associated with the task ignition key status monitoring

The events ***seat_belt_on*** and ***seat_belt_off*** are associated with the task seat belt status monitoring.

Concurrent/Communicating Process Model

Tasks for Seat Belt Warning System

Create and initialize events

wait_timer_expire, ignition_on, ignition_off,

seat_belt_on, seat_belt_off,

alarm_timer_start, alarm_timer_expire

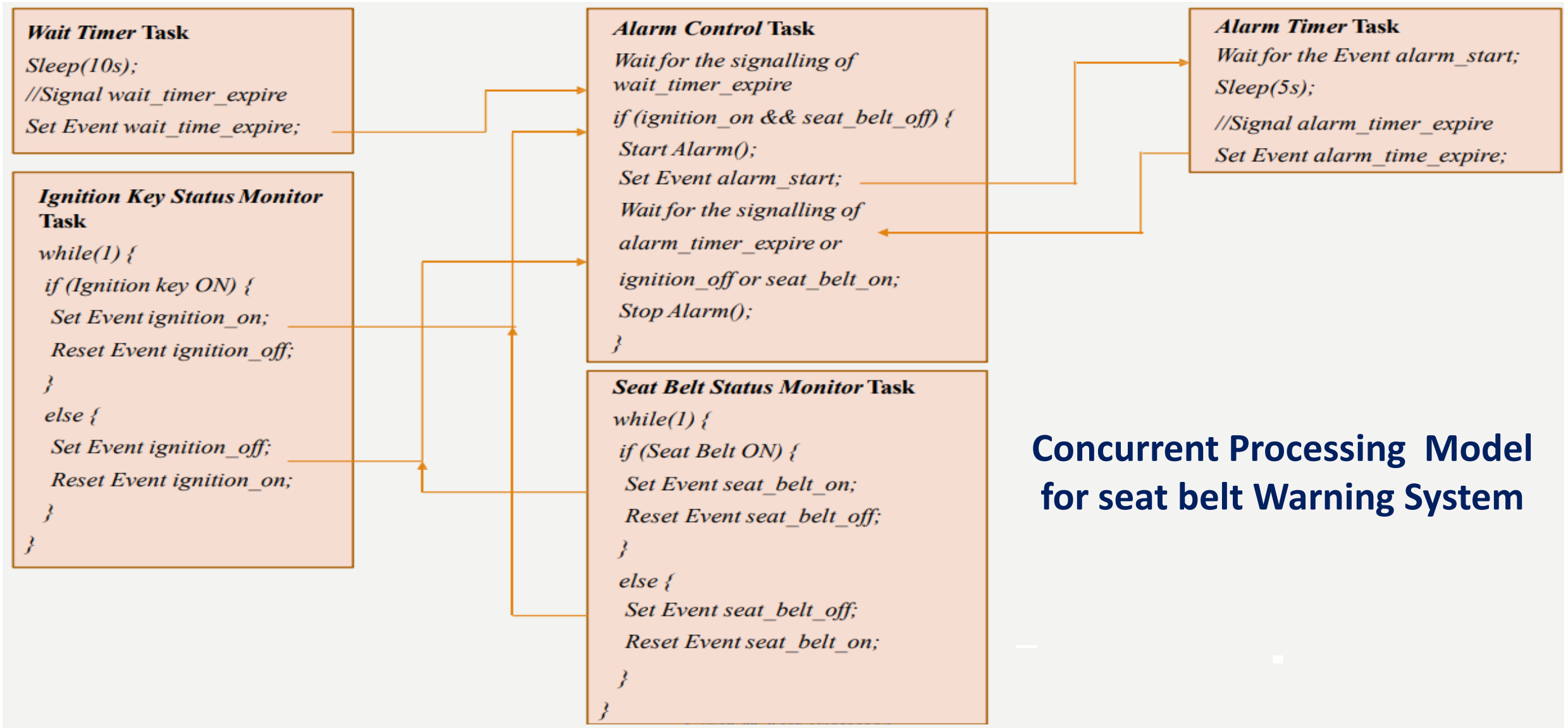
Create task *Wait Timer*

Create task *Ignition Key Status Monitor*

Create task *Seat Belt Status Monitor*

Create task *Alarm Control*

Create task *Alarm Timer*



Concurrent Processing Model for seat belt Warning System

ATME College of Engineering

Embedded Systems-18EC62

Module-4

Topic: Embedded Firmware Design & Development

Pradeep Kumar Y
Assistant Professor
Dept.of ECE

Object-Oriented Model

- It is an object based model for modeling system requirements.
- It disseminates a complex software requirement into simple well defined pieces called objects.
- It 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.

Object-Oriented Model

- 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

Protected

Object-Oriented Model

- Private member variables and functions are accessible only within the class, whereas 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.
- However classes derived from a 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 & Development

- 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)

Embedded Firmware Design & Development

- 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:
  - 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

Embedded firmware Design Approaches – The Super loop

✓A typical super loop implementation will look like:

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

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

Embedded firmware Design Approaches – The Super loop

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:
  - 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

Embedded firmware Design Approaches – Embedded OS based Approach

- Microsoft® Windows XP Embedded is an example of GPOS for embedded devices
- Point of Sale (PoS) terminals, Gaming Stations, Tablet PCs etc are examples of embedded devices running on embedded GPOSs
- '*Windows CE*', '*Windows Mobile*','*QNX*', '*VxWorks*', '*ThreadX*', '*MicroC/OS-II*', '*Embedded Linux*', '*Symbian*' etc are examples of RTOSs employed in Embedded Product development
- Mobile Phones, PDAs, Flight Control Systems etc are examples of embedded devices that runs on RTOSs

Embedded firmware Development Languages

- 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**

Assembly Language

- 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
- 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

Assembly Language

- 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 *MOV A* and 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)

Assembly Language

- 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).

Assembly Language

```
;#########################################################################
;           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
```

- ✓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; LMP DELAY

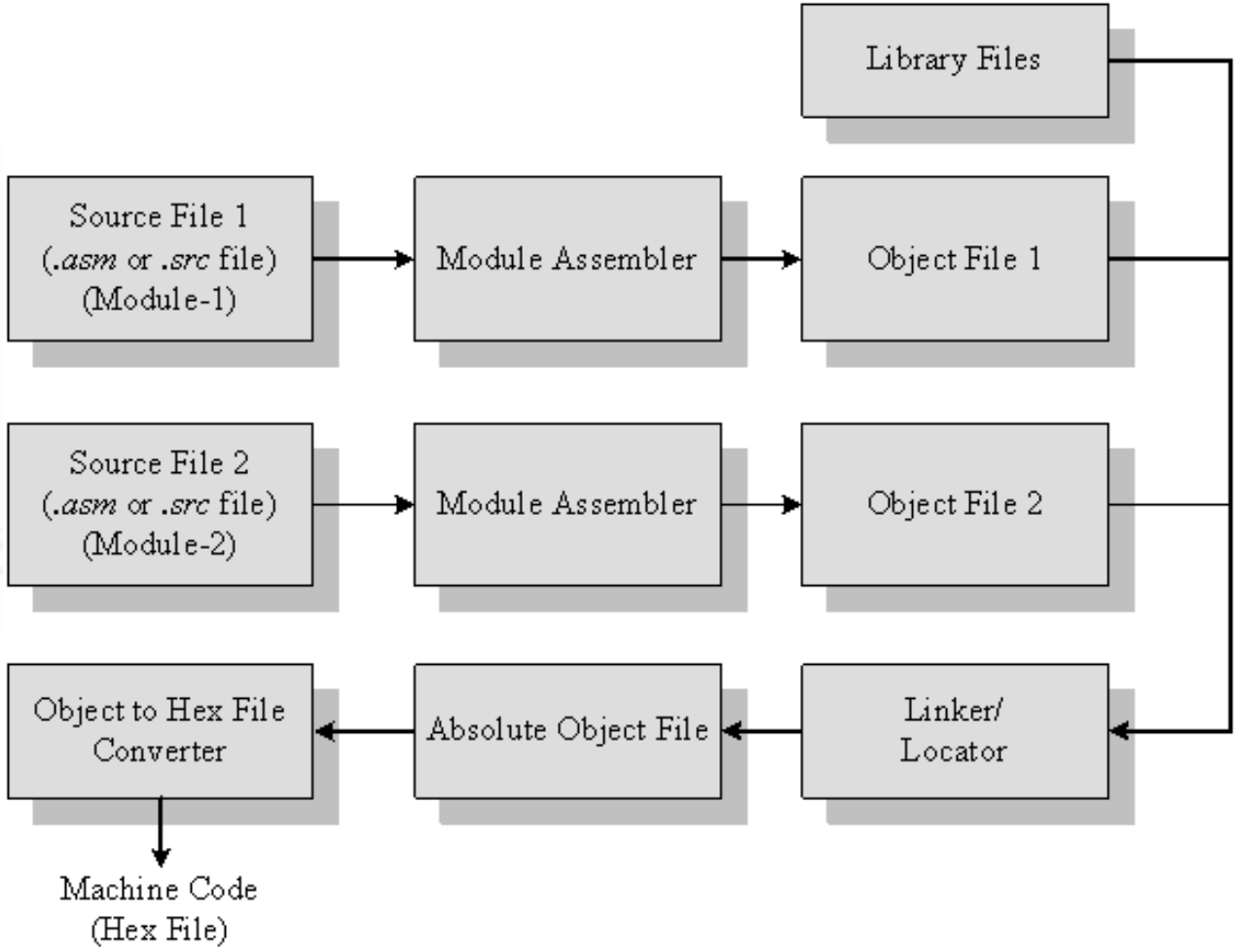
Assembly Language- Source File to Hex File Translation

Advantages:

- ✓Efficient Code Memory & Data Memory Usage (Memory Optimization)
- ✓High Performance
- ✓Low level Hardware Access
- ✓Code Reverse Engineering

Drawbacks:

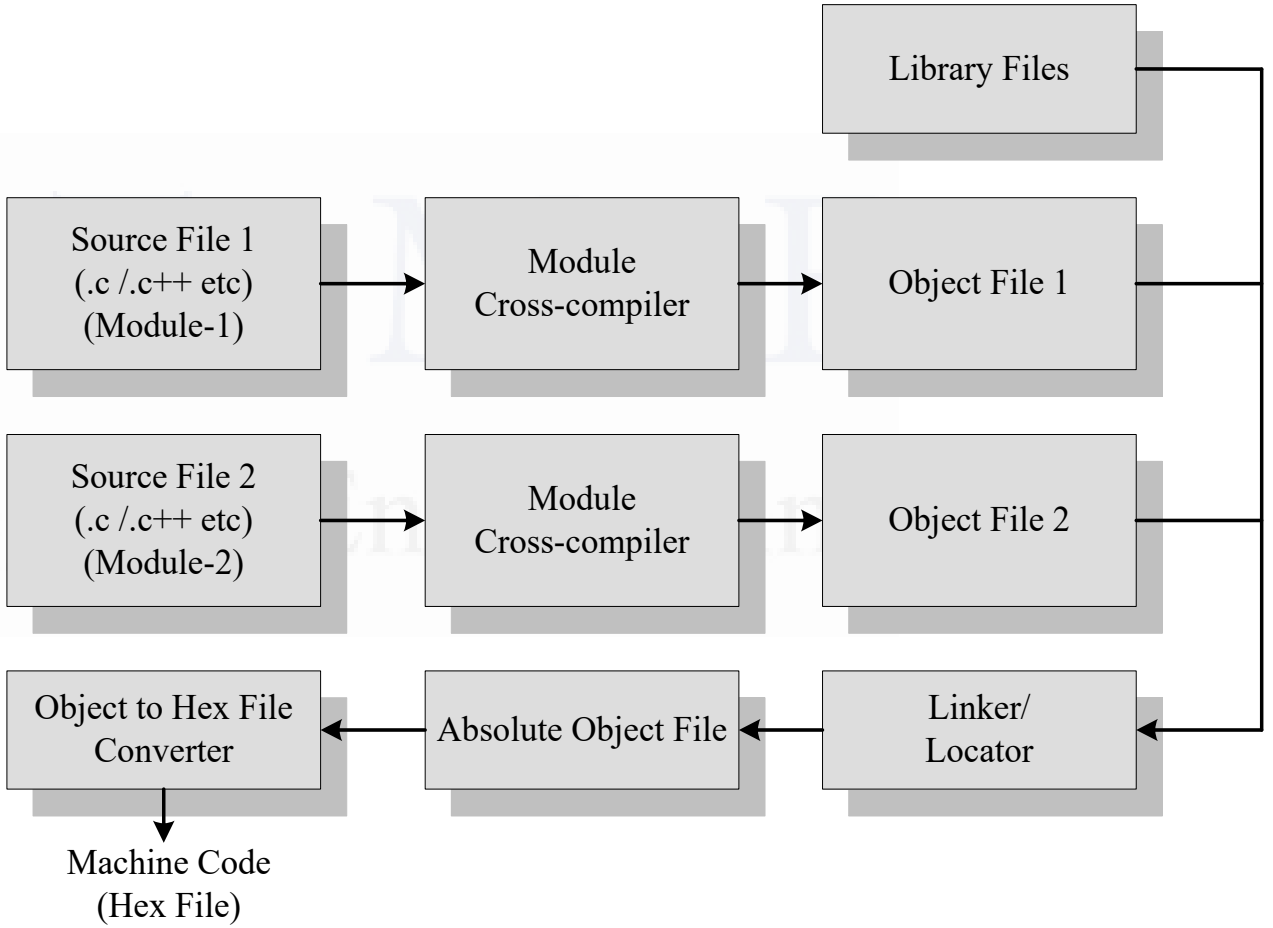
- ✓High Development time
- ✓Developer dependency
- ✓Non portable



High Level Language

Advantages:

- ✓Reduced Development time
- ✓Developer independency
- ✓Portability



Mixing of Assembly Language with High Level Language

- Certain situations in Embedded firmware development may require the mixing of Assembly Language with high level language or vice versa. Interrupt handling, Source code is already available in high level language\Assembly Language etc are examples.
- High Level language and low level language can be mixed in three different ways
  - 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 of Assembly Language with High Level Language

Assembly routines are mixed with 'C' in situations where

- The entire program is written in 'C' and the cross compiler in use do not have a built in support for implementing certain features like Interrupt Service Routine functions (ISR) or
- If the programmer wants to take advantage of the speed and optimized code offered by machine code generated by hand written assembly rather than cross compiler generated machine code.
- When accessing certain low level hardware, the timing specifications may be very critical and a cross compiler generated binary may not be able to offer the required time specifications accurately.
- Writing the hardware/peripheral access routine in processor/controller specific Assembly language and invoking it from 'C' is the most advised method to handle such situations.

Mixing of Assembly Language with High Level Language

- Mixing 'C' and Assembly is little complicated.
- The programmer must be aware of how parameters are passed from the 'C' routine to Assembly and values are returned from assembly routine to 'C' and how 'Assembly routine' is invoked from the 'C' code.
- Passing parameter to the assembly routine and returning values from the assembly routine to the caller 'C' function and the method of invoking the assembly routine from 'C' code is cross-compiler dependent.

Mixing of High Level Language with Assembly Language

Mixing the code written in a high level language like 'C' and Assembly language is useful in the following scenarios:

1. The source code is already available in Assembly language and a routine written in a high level language like 'C' needs to be included to the existing code.
2. The entire source code is planned in Assembly code for various reasons like optimized code, optimal performance, efficient code memory utilization and proven expertise in handling the Assembly, etc. But some portions of the code may be very difficult and tedious to code in Assembly.

   For example, 16-bit multiplication and division in 8051 Assembly Language.

3. To include built in library functions written in 'C' language provided by the cross compiler.
   For example, Built in Graphics library functions and String operations supported by 'C'.

Mixing of High Level Language with Assembly Language

- Most often the functions written in 'C' use parameter passing to the function and returns value/s to the calling functions.
- Parameters are passed to the function and values are returned from the function using CPU registers, stack memory and fixed memory.
- Its implementation is cross compiler dependent and it varies across cross compilers.

Mixing of High Level Language with Assembly Language

Example for the Keil C51 cross compiler.

- ✓C51 allows passing of a maximum of three arguments through general purpose registers R2 to R7.
- ✓If the three arguments are char variables, they are passed to the function using registers R7, R6 and R5, respectively.
- ✓If the parameters are int values, they are passed using register pairs (R7, R6), (R5, R4) and (R3, R2).
- ✓If the number of arguments is greater than three, the first three arguments are passed through registers and rest is passed through fixed memory locations.

Mixing of High Level Language with Assembly Language

- Return values are usually passed through general purpose registers.
- R7 is used for returning char value
- Register pair (R7, R6) is used for returning int value.
- The 'C' subroutine can be invoked from the assembly program using the subroutine call Assembly instruction.

For example

```
LCALL _Cfunction
```

where C function is a function written in 'C'

- The prefix _ informs the cross compiler that the parameters to the function are passed through registers.
- If the function is invoked without the _ prefix, it is understood that the parameters are passed through fixed memory locations.

Inline assembly Programming

- Inline assembly is a technique for inserting target processor/controller specific Assembly instructions at any location of a source code written in high level language 'C'.
- This avoids the delay in calling an assembly routine from a 'C' code.
- Special keywords are used to indicate that the start and end of Assembly instructions. \
- The keywords are cross-compiler specific.
- C51 uses the keywords #pragma asm and #pragma endasm to indicate a block of code written in assembly.

For example:

```
#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.
- 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 range of Giga bytes and storage memory in the range of several Giga bytes.
- Almost all embedded systems are limited in both storage and working memory resources
- The hands of an embedded application developer are always tied up in the memory usage context.

High Level Language – '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

High Level Language – ‘C’ V/s Embedded C

- ✓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

High Level Language Based Development – '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).
- 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.

High Level Language Based Development – 'Compiler' V/s 'Cross-Compiler'

- 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 C51compiler from Keil software is an example for cross-compiler for 8051 family architecture

Summary

1. Characteristics of Embedded Systems
2. Quality Attributes of Embedded Systems
   i. Operational quality attributes
   ii. Non-operational quality attributes
3. Application specific Embedded System
4. Domain specific Embedded System
5. Hardware Software Co-Design and Program Modeling
   i. Fundamental Issues in Hw-Sw Co-Design
   ii. Computational Models in Embedded Design
6. Embedded Firmware Design Approaches
7. Embedded Firmware Development Languages

VTU- July-2018

Module-4

7 a. Explain the term quality attributes in an embedded system development context. What are the different quality attributes to be considered in an embedded system design. **(08 Marks)**

b. Explain Data flow graph and control data flow graph models in the embedded design. **(08 Marks)**

OR

8 a. Explain the different 'Embedded firmware design' approach in detail. **(08 Marks)**

b. Explain the characteristics of an Embedded system. **(08 Marks)**

VTU- Jan-2020

Module-4

7 a. Explain the different characteristics of embedded system in detail. **(08 Marks)**

b. With a block diagram, mention the components and in the design of a washing machine and also explain its working. **(08 Marks)**

OR

8 a. What is hardware and software co-design? Explain the fundamental design approaches in detail. **(10 Marks)**

b. With FSM model, explain the design and operation of automatic tea/coffee lending machine. **(06 Marks)**

VTU- July-2019

7	a.	What are the operational and nonoperational attributes of an embedded systems.	(10 Marks)
	b.	Explain different types of serial interface bus used in automotive communication.	(06 Marks)

OR

8	a.	Explain fundamental issues in hardware software co-design.	(06 Marks)
	b.	Explain with a neat block diagram how source file to object file translation take place.	(06 marks)
	c.	Explain super loop based approach of embedded firmware design.	(04 Marks)

VTU- July-2019

Module-4

7	a.	Discuss the 6 operation quality attributes of an embedded system.	(06 Marks)
	b.	With FSM model, explain the design and operation of automatic seat belt monitoring system.	(06 Marks)
	c.	Compare CDFG and DFG with an example.	(04 Marks)

OR

8	a.	With a neat flow diagram, explain high level language to machine language conversion process.	(05 Marks)
	b.	With a block diagram, mention the components used in the design of a washing machine and also explain its working.	(06 Marks)
	c.	Describe in brief the typical characteristics of an embedded system.	(05 Marks)