MODULE 4

EMBEDDED SYSTEM DESIGN CONCEPTS

Characteristics of Embedded Systems

- Some of **the important characteristics of an embedded system are:**
  1. Application and domain specific
  2. Reactive and Real Time
  3. Operates in harsh environments
  4. Distributed
  5. Small size and weight
  6. Power concerns

1.Application and domain specific

- 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 **will not do any other task.**

  Ex. Air conditioner's embedded control unit cannot be replaced with microwave oven control unit .

  Ex. A washing machine can only wash, it cannot cook.

- Certain embedded systems are **specific to a domain: Ex. A hearing aid is an** application that belongs to **the domain of signal processing and telecom with** another control unit designed to serve another domain **like consumer electronics.**

2. Reactive and Real Time

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

3. Operation in Harsh Environment

- Certain embedded systems **are designed to operate in harsh environments like a dusty one or a high temperature zone or an area** subject to vibrations and shock or very high temperature of the deserts or very low temperature of the mountains or extreme rains.
- These embedded systems have to be **capable of sustaining the environmental conditions it is designed to operate in.**

4. Distributed

- The term distributed means that embedded systems may be a part of a larger system. These components are independent of each other but have to work together for the larger system to function properly.

  Ex. An automatic vending machine is a typical example for this. The vending machine contains a card reader (for pre-paid vending systems), a vending unit, etc. Each of them are independent embedded units but they work together to perform the overall vending function.

  Ex. Automatic Teller Machine (ATM) contains a card reader embedded unit, responsible for reading and validating the user's ATM card, transaction unit for performing transactions, a currency counter for dispatching/vending currency to the authorized person and a printer unit for printing the transaction details.

- This can visualize these as independent embedded systems. But they work together to achieve a common goal.

5. Small Size and Weight

- An embedded system that is compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.

  Ex. Currently available cell phones. The cell phones that have the maximum features are popular but also their size and weight is an important characteristic.

6. Power Concerns

- It is desirable that the power utilization and heat dissipation of any embedded system be low.
- If more heat is dissipated then additional units like heat sinks or cooling fans need to be added to the circuit.

  Ex. The production of high amount of heat demands cooling requirements like cooling fans which in turn occupies additional space and make the system bulky.
- Nowadays ultra low power components are available in the market. Select the design according to the low power components like low dropout regulators, and controllers/processors with power saving modes.
- Also power management is a critical constraint in battery operated application.
- The more the power consumption the less is the battery life.

Quality Attributes of Embedded Systems

- The various quality attributes that needs to be addressed in any embedded system development are broadly **classified into two, namely**
  - Operational Quality Attributes
  - Non-Operational Quality Attributes

- **Operational Quality Attributes**

Operational Quality Attributes
Response
Throughput
Reliability
Safety
Security
Maintainability

1.Response

- Response is a measure of quickness of the system.
- It gives you an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.
- 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.

2. Throughput

- **Throughput deals with the efficiency of system.**
- It can be **defined as rate of production or process of a defined** process over a stated period of time.
- In case of **card reader like the one used in buses, throughput** means **how much transactions the Reader can perform in a minute or hour or day.**
- Throughput is generally measured in terms of **'Benchmark'.** A 'Benchmark' is a reference point by which something can be measured.

3.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.**

4. 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.**

5.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.
- Certain embedded systems have to make sure that they conform to the security measures.
- Ex. An electronic safety Deposit Locker can be used only with a pin number like a password.

6. 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.
- Maintainability can be classified into **two types:**

1. Scheduled or Periodic Maintenance (Preventive Maintenance)

An inkjet printer uses ink cartridges, which are consumable components and as per the printer manufacturer the end use **should replace the cartridge after each 'n' number of printouts to get** quality prints.

2. Maintenance to Unexpected Failures (Corrective Maintenance)

If the paper feeding part of the printer fails the printer fails to print and it requires **immediate repairs to rectify this problem.**

- The ideal value for availability is expressed as

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

- Where Ai=Availability in the ideal condition, MTBF=Mean Time Between Failures, and MTTR=Mean Time To Repair.

❑ Non-Operational Quality Attributes

1.Testability & Debug-ability

Non-Operational Quality Attributes
Testability & Debug-ability
Evolvability
Portability
Time to prototype and market
Per unit and total cost

- ❑ Testability deals with how easily one can test his/her design, application.
- ❑ For an embedded product, testability is applicable to both the embedded hardware and firmware.
- ❑ Debug-ability means debugging the product to figure 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.

2. 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.**

3.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.**

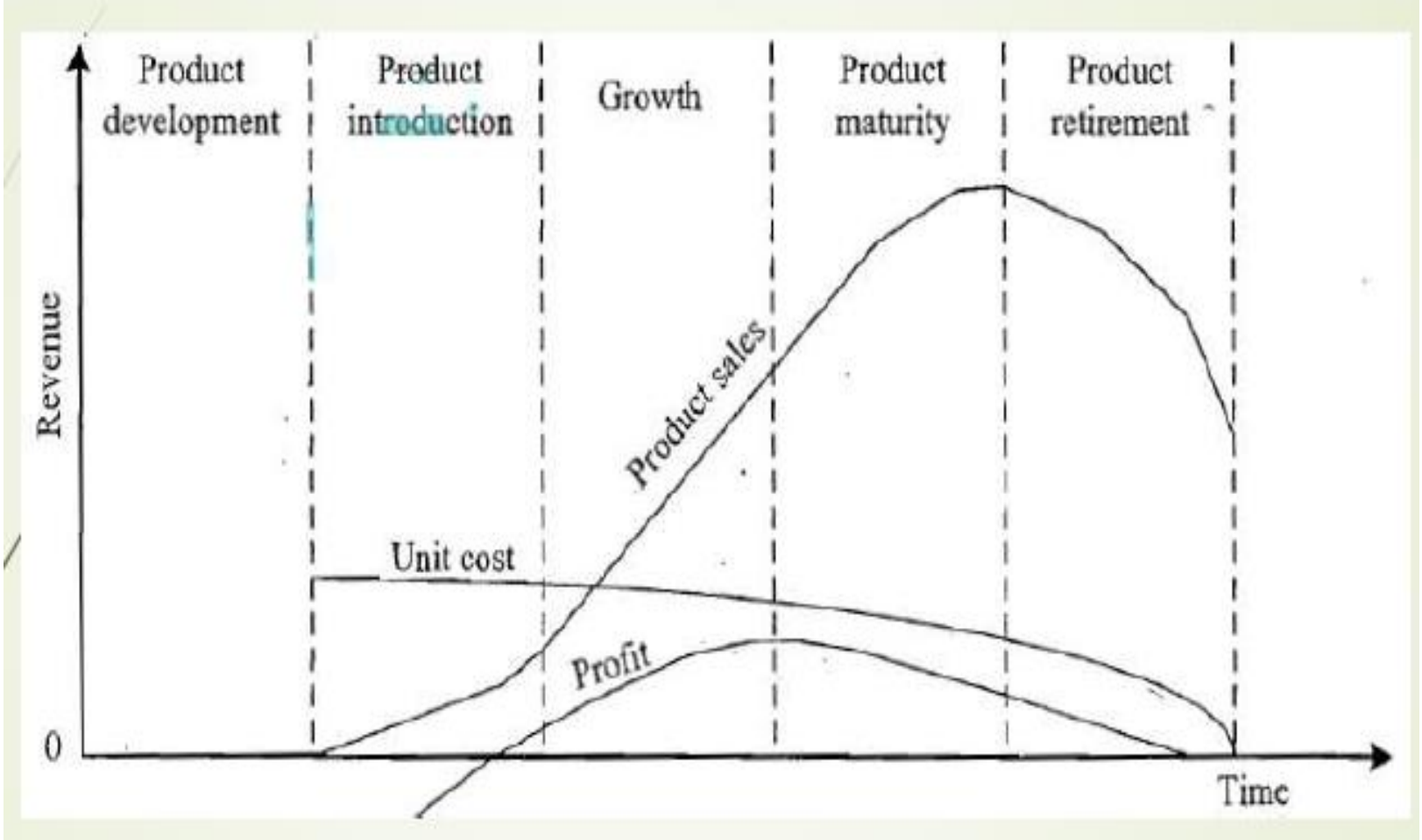
4.Time-to-Prototype and Market

- Time-to-market is **the time elapsed between the conceptualization of a** product and the time at which **the product is ready for selling.**
- **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.**

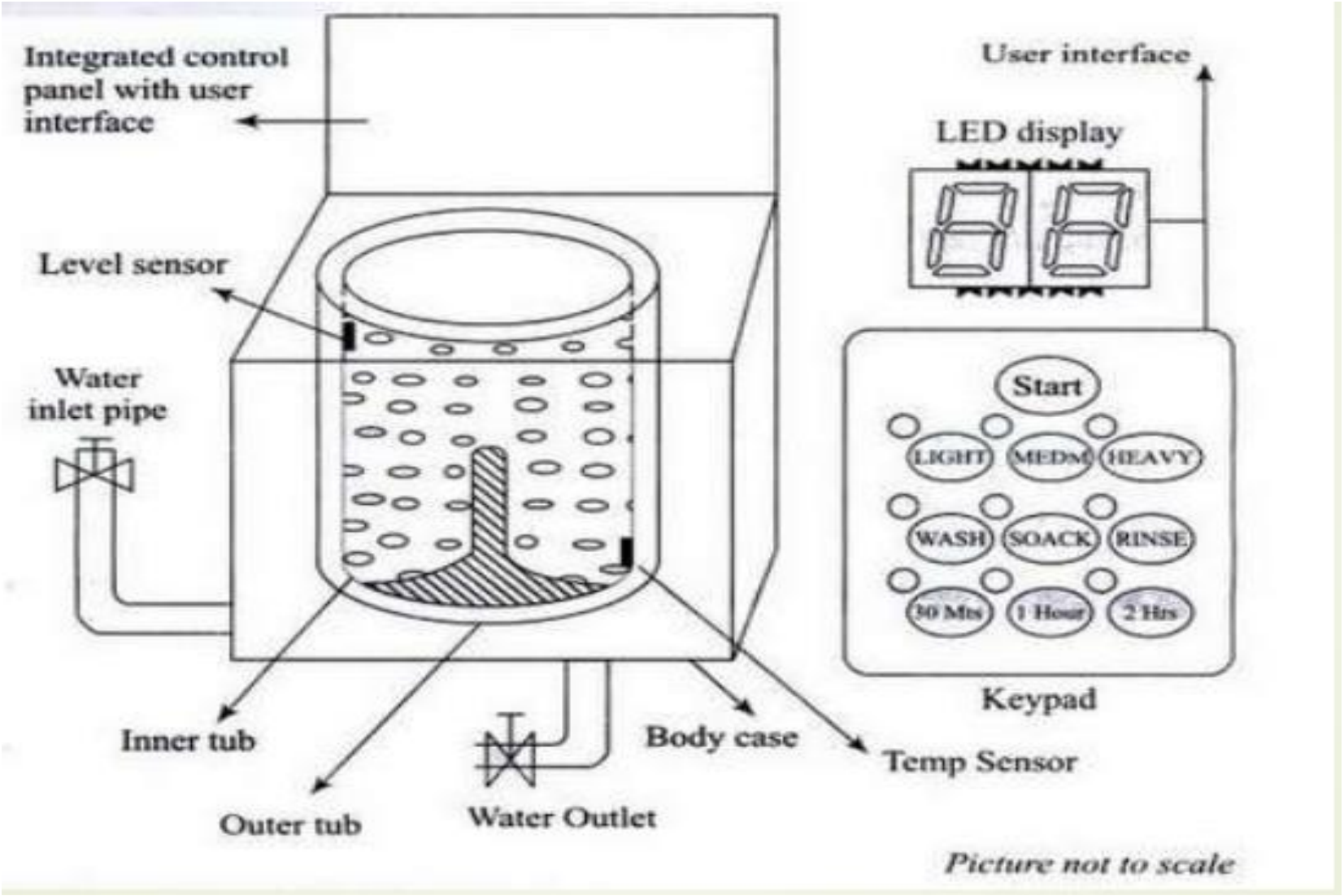
5.Per Unit Cost and Revenue

- Cost is a factor which is closely monitored by both end user (those who buy the product) and product manufacturer.
- 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.
- 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.

Product life cycle (PLC) curve



Washing Machine-Application-Specific Embedded System



- A washing Machine (Embedded system) contains **sensors, actuators, control unit and application-specific user interfaces like keyboards, display units, etc.**
- The **actuator part of washing machine consists of a motorized agitator, tumble tub, water** drawing pump and inlet valve to control the flow of water into the unit.
- The **sensor part consists of the water temperature sensor, level sensor, etc.**
- The **control part contains a microprocessor/controller based board with interfaces** to the sensors and actuators.
- The **sensor data is fed back to the control unit and the control unit generates the** necessary actuator outputs.
- The **control unit also provides connectivity to user interfaces like keypad for** setting the washing time, selecting the type of material to be washed like light, medium, heavy duty, etc.
- **User feedback is** reflected through the display unit and LEDs connected to the control board.

- In the first phase of washing, the clothes are tumbled and plunged into the water over and over again.
- In the second phase of washing, water is pumped out from the tub and the inner tub uses centrifugal force to wring out more water from the clothes by spinning at several hundred RPM called as spin phase.
- The integrated control panel consists of a microprocessor/controller based board with I/O interfaces and a control algorithm running in it.
- Input interface includes the keyboard which consists of wash type selector namely Wash, Spin and Rinse, cloth type selector namely Light, Medium, Heavy duty and washing time setting, etc.
- The output interface consists of LED/LCD displays, status indication LEDs, etc. connected to the I/O bus of the controller.
- It is to be noted that this interface may vary from manufacturer to manufacturer and model to model.

Automotive-Domain-Specific Examples of Embedded System

- Automotive embedded systems are the ones where electronics take control over the mechanical systems.
- Automotive embedded systems are normally built around microcontrollers or DSPs or a hybrid of the two and are generally known as Electronic Control Units (ECUs).
- The various types of electronic control units (ECUs) used in the automotive embedded industry can be broadly classified into two
- High speed embedded control units and Low speed embedded control units.
- High speed Electronic Control Units (HECUs) : High speed electronic control units (HECUs) are deployed in critical control units requiring fast response, like fuel injection systems, antilock brake systems, etc.
- Low speed Electronic Control Units (LECUs) : Low speed electronic control units are deployed in applications where response time is not so critical.
- They are generally built around low cost microprocessors/microcontrollers and digital signal processors.
- Audio controllers, passenger and driver door locks, door glass controls, etc., are examples for LECUs.

- **Automotive communication buses**
- Automotive applications use serial buses for communication.
- Controller Area Network (CAN), Local Interconnect Network (LIN), Media Oriented System Transport (MOST) bus, etc. are the important automotive communication buses.
- **CAN** is an event driven serial protocol interface with support for error handling in data transmission. It is generally employed in safety system like airbag control, power train systems like engine control and Antilock Brake Systems.
- **LIN** bus is a single master multiple slave (up to 16 independent slave nodes) communication interface. LIN is a low speed, single wire communication interface with support for data rates up to 20 kbps and is used for sensor/actuator interfacing.
- **MOST** bus is targeted for automotive audio video equipment interfacing.
- MOST bus is a multimedia fiber-optic point-to point network implemented in a star, ring or daisy chained topology over optical fibers cables.

Fundamental Issues In Hardware Software Co-Design

Following are the Fundamental issues in hardware software co-design

- **Selecting the model**
- It is always difficult to make a decision on which model should be used for system design.
- **Selecting the architecture**
- Architecture specifies how the system is going to behave in terms of different components and interconnectivity among them. Following are the different architecture normally used in system design.
- Controller architecture
- Datapath architecture
- Finite state machine datapath architecture(FSMD)
- CISC architecture
- Very long instruction word architecture(VLIW)
- Parallel processing architecture such as Single instruction multiple data architecture(SIMD) and Multiple instruction multiple data architecture(MIMD)

Fundamental Issues In Hardware Software Co-Design(Cont…)

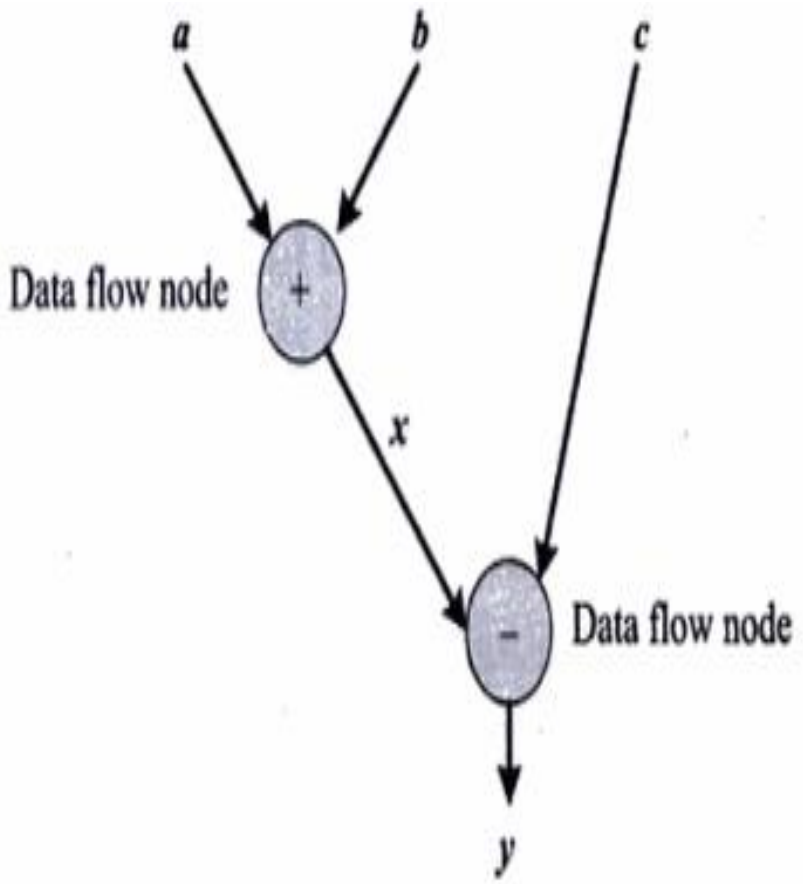
- **Selecting the language**
- A programming language captures computational model and maps it into architecture.
- Only constraint here is that a programming language must capture the model easily.
- Examples are C,C++,C# and JAVA etc.
- **Partitioning system requirements into hardware and software**
- It is tough decision making task to implement system requirements in either hardware or software.
- Various hardware software trade off can be used for decision making for hardware software partitioning.

Computational Models In Embedded System

1. Data Flow Graph Model(DFG)
2. Control Data Flow Graph Model(CDFG)
3. State Machine Model
4. Sequential Program Model
5. Concurrent process Model
6. Object Oriented Model

1.Data Flow Graph Model(DFG)

- DFG model translates the data processing requirements into a data flow graph.
- It is a Data driven model in which program execution is determined by data.
- Operation on data is represented by a circle and data flow represented by 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
- As shown in the figure first operation is represented by **x = a + b** and final is represented by **y = x - c**



2.Control Data Flow Graph Model(CDFG)

- CDFG model used for modelling applications involving conditional program execution.
- CDFG model contains both data and control operations.
- It uses DFG as element and conditions as decision makers. So It has both data flow node and control node (decision node) which is represented by diamond
- As shown in the figure if flag=1,operation is represented as **x=a+b** otherwise it is represented as **y=a-b**
- 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.

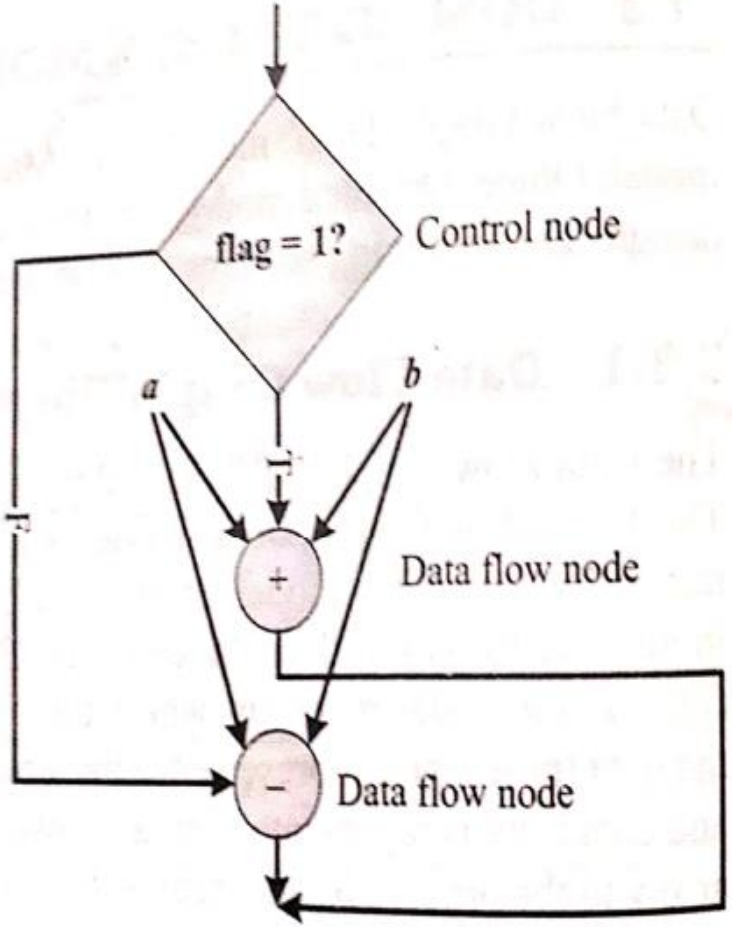


Fig. 7.2 **Control Data Flow Graph (CDFG) Model**

3.State Machine Model

- It is used for modelling reactive or event driven embedded systems whose processing behaviour are dependent on state transitions.
- This model describes system behaviour with ‘States’,’Events’,’Actions’,and ‘Transitions’.
- **A state** is a representation of the current situation.
- **An event** is the input to the state.
- **Action** is the activity to be performed by the state machine.
- **Transition** is the movement from one state to another state.
- Finite state machine (**FSM**) is a term used by programmers, mathematicians, engineers and other professionals to describe a mathematical **model** for any system that has a limited number of conditional states.
- **Examples**
  - **FSM Model for automatic seat belt warning system.**
  - **FSM Model for automatic Tea/Coffee vending machine.**
  - **FSM Model for coin operated Telephone System.**

- **FSM Model for automatic seat belt warning system**
- States are ‘alarm off', 'waiting’ and ‘alarm on’
- Events are ‘Ignition Key On’, ’Ignition Key Off’ ,’Seat Belt On’, ‘Alarm Time Expire’ and ‘Timer expire’.
- ‘Ignition Key On’ event triggers 10 second timer transits the state to 'waiting’ .
- If a ’Seat Belt On’ and ’Ignition Key Off’ event occurs during wait state, state transits to ‘alarm off‘.
- When the wait timer expires in the waiting state, event ‘Timer expire’ is generated and state transits to ‘alarm on’ from the waiting state.
- Timer state can be either ‘IDLE’ or ‘READY’ or ‘RUNNING’.
- Initially timer is in its normal state then it is said to be in ‘IDLE’ state.
- When it is loaded with count then it is said to be in ‘READY’ state. It remains in ‘READY’ state until ‘Start Timer’ event occurs.
- On receiving ‘Start Timer’ event timer changes from ‘READY’ to ‘RUNNING’ state. It remains in ‘RUNNING’ state till ‘Stop Timer’ event occurs.
- On receiving ‘Stop Timer’ event timer changes from ‘RUNNING’ to ‘IDLE’ state.

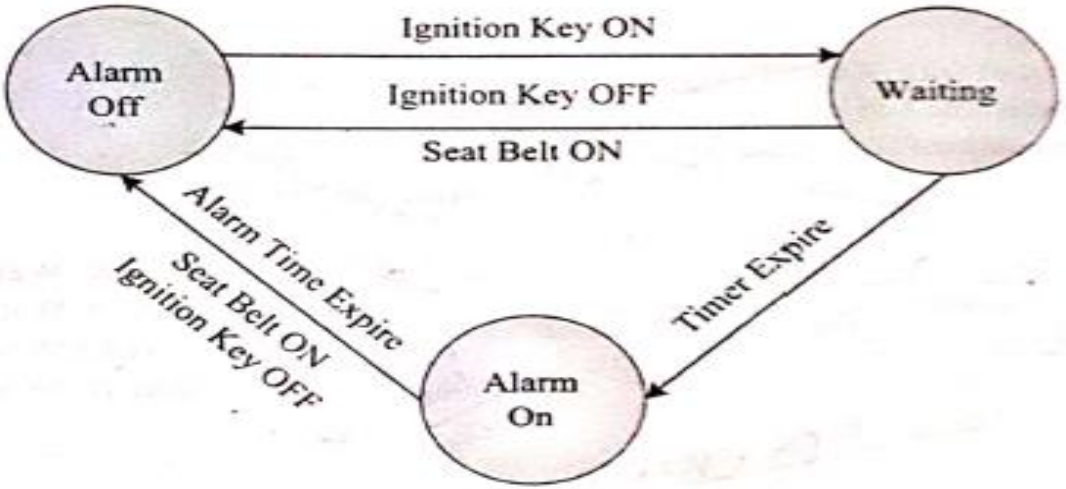


Fig. 7.3 FSM Model for Automatic seat belt warning system

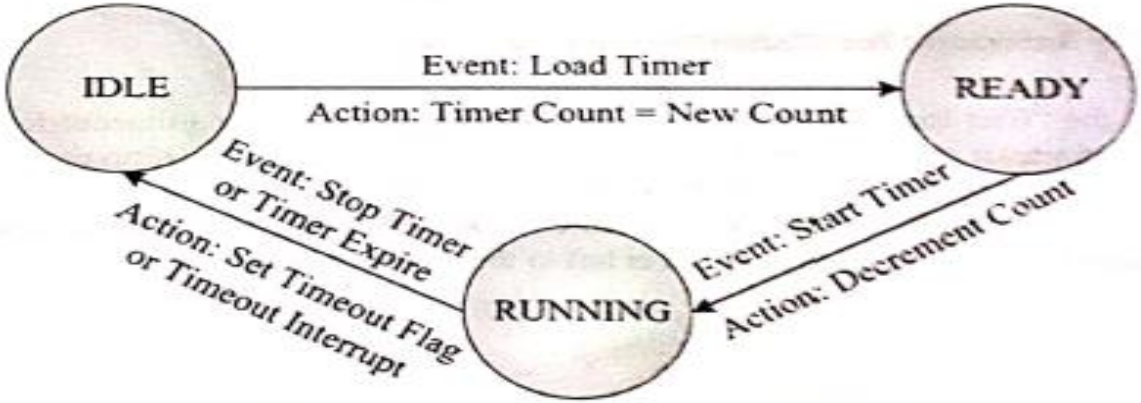


Fig. 7.4 FSM Model for timer

❑FSM Model For Automatic Tea/Coffee Vending Machine.

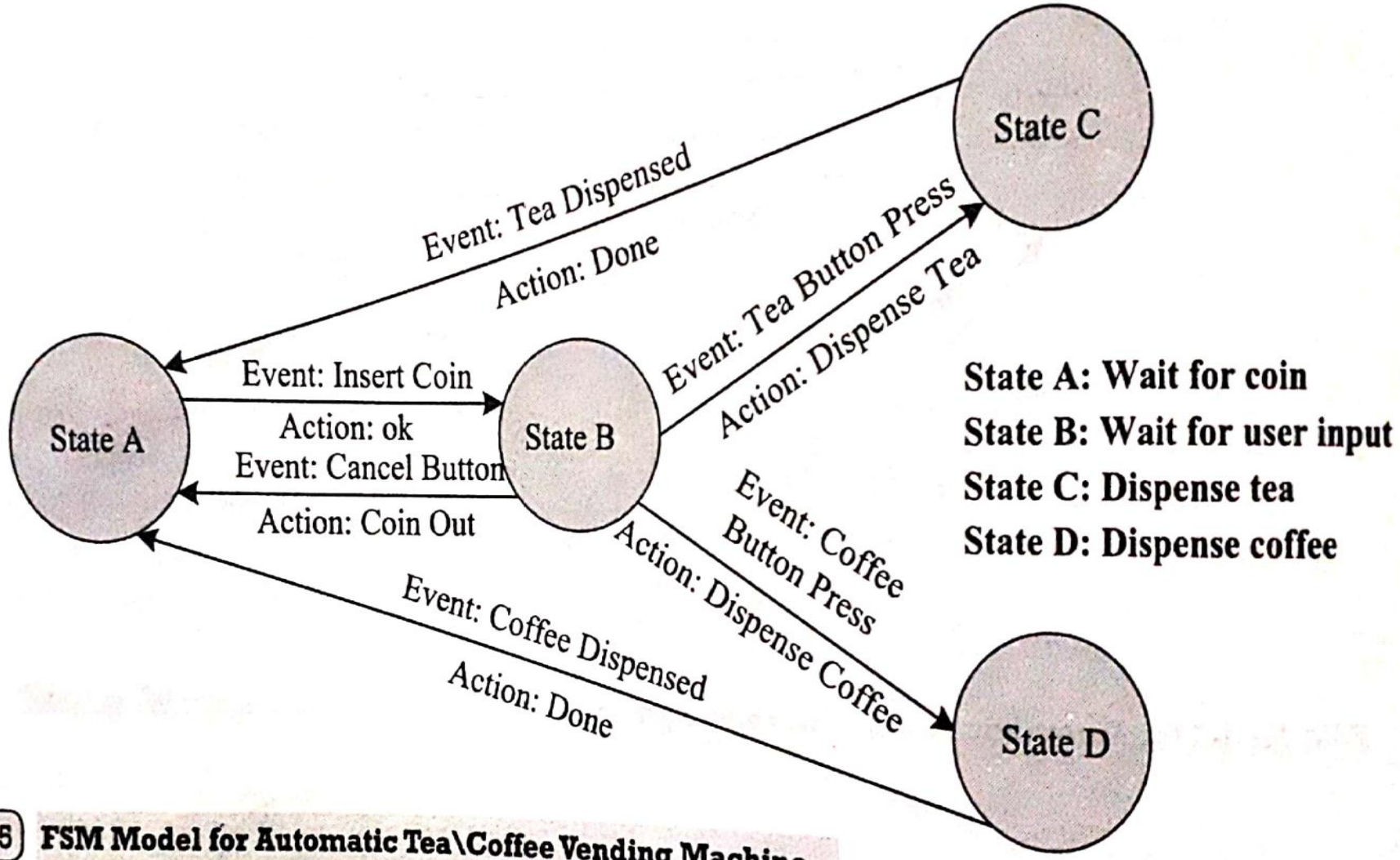


Fig. 7.5 **FSM Model for Automatic Tea\Coffee Vending Machine**

❑FSM Model For Coin Operated Telephone System

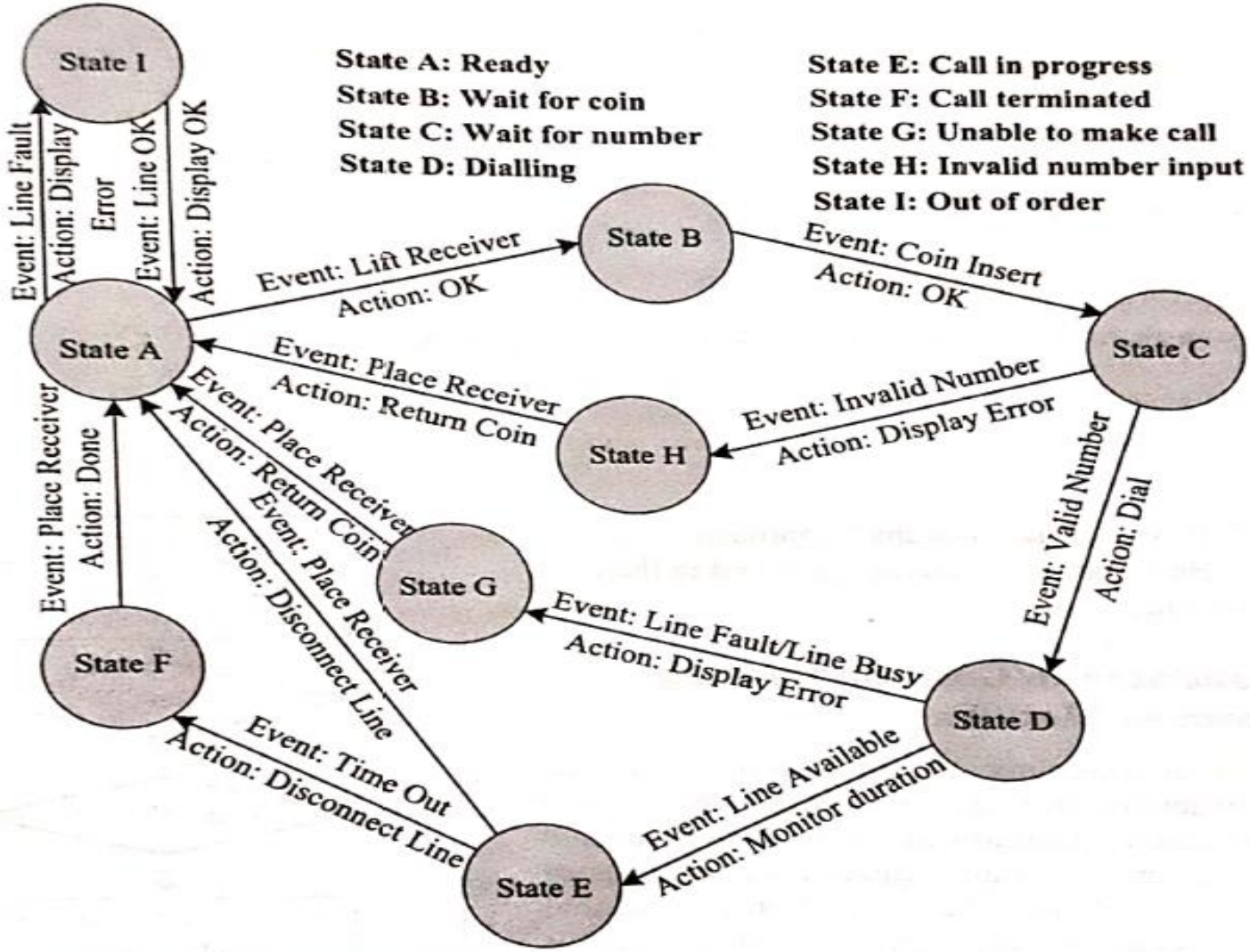


Fig. 7.6 **FSM Model for Coin Operated Telephone System**

4.Sequential Program Model

- In the sequential program model, processing requirements and functions are executed in sequence and this is same as conventional procedural programming.
- Here program instructions are iterated and executed conditionally.
- Data gets transformed through a series of operations.
- Flow Chart is another important tool used for modeling sequential program.
- Figure depicts sequential program model for seat belt warning system.

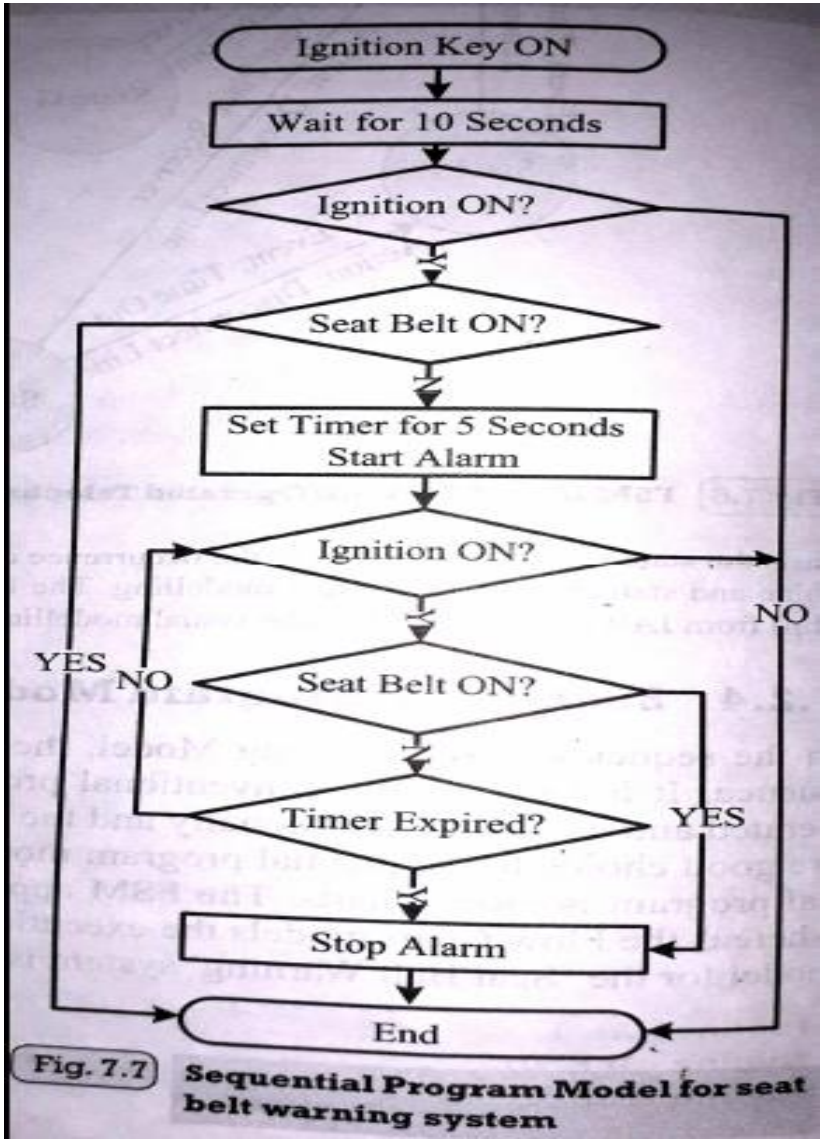


Fig. 7.7 **Sequential Program Model for seat belt warning system**

5.Concurrent Process Model

- Concurrent Process Model, concurrently executes tasks /processes.
- If the tasks/processes split into multiple sub tasks/processes it is possible to tackle the CPU usage effectively.
- Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication.
- As an example for the concurrent processing model, concurrent tasks for seat belt warning system is as shown.

- **Task 1:** Timer Task for waiting 10 seconds.
- **Task 2:**Task for checking the ignition key status.
- **Task 3:**Task for checking the seat belt status.
- **Task 4:**Task for Starting and Stopping the Alarm.
- **Task 5:**Alarm timer Task for waiting 5 seconds.

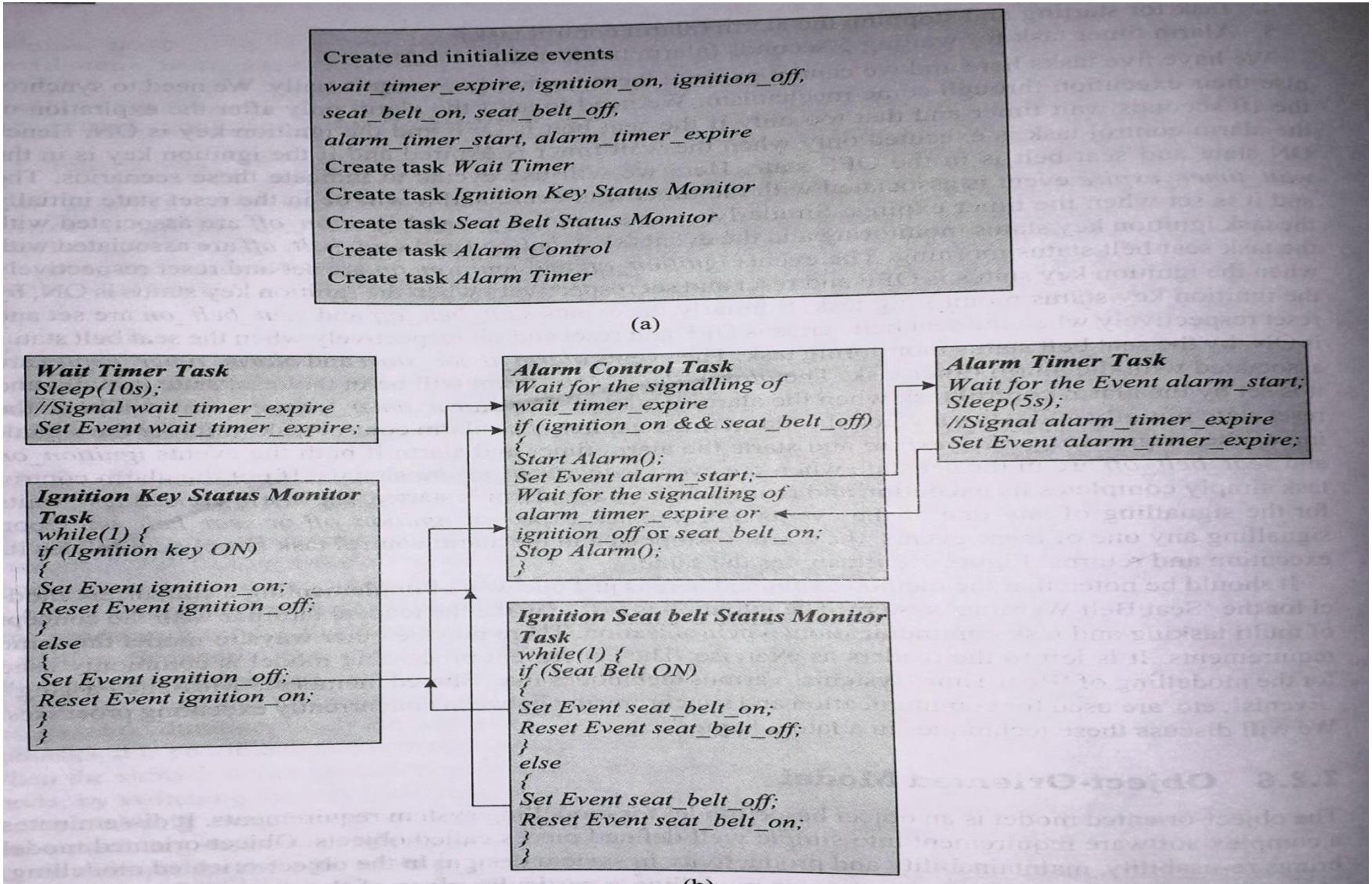


Fig. 7.8 **(a) Tasks for 'Seat Belt Warning System' (b) Concurrent processing Program model for 'Seat Belt Warning System'**

6.Object Oriented Model

- It is an object based model.
- It disseminates complex software requirements into simple well defined pieces called objects.
- It brings Reusability, Maintainability and Productivity in System design.
- In the object oriented modelling, object is an entity used for representing a particular piece of the system.
- A class is an abstract description of set of objects.
- A class represents the state of an object through member variables and behaviour through member functions.
- A member variables can be Public, Private and Protected.

The Embedded Firmware Design Approaches

- 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 Super loop based approach**
- Suitable for applications that are not time critical and where the response time is not so important (Embedded systems where missing deadlines are acceptable).
- Very similar to a conventional procedural programming where the code is executed task by task.
- The tasks are executed in a never ending loop.
- The task listed on top on the program code is executed first and the tasks just below the top are executed after completing the first task.

- **A typical super loop implementation will look like:**

  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

- **Advantages**
  - 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
- **Disadvantages**
  - Non Real time in execution behavior
  - Any issues in any task execution may affect the functioning of the product
- **Example**: Electronic Video game Toy

- **The Embedded Operating System based approach**
- The embedded device contains an Embedded Operating System which can be of
- A Real Time Operating System (RTOS) and
- 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.
- Microsoft® Windows XP Embedded is an example of GPOS for embedded devices.
- '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

The Embedded Firmware Development Languages

- In Embedded Firmware Development there are Target Processor/Controller Specific Languages, Target Processor/Controller Independent languages available for Embedded Firmware development
  - Assembly Language Based Development
  - High Level Language Based Development
  - Mixing assembly and High level Language
- **Assembly Language Based Development**
- 'Assembly Language' is the human readable notation of 'machine 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.
- Each line of an assembly language program is split into four fields as: **LABEL OPCODE OPERAND COMMENTS**

Assembly Language To Machine Language Conversion Process

- The Assembly language program written in assembly code is saved as .asm (Assembly file) file or a .src (source) file or a format supported by the assembler.
- The software utility called ‘Assembler’ performs the translation of assembly code to machine code.
- Each source file can be assembled separately to examine the syntax errors and incorrect assembly instructions.
- Assembling of each source file generates a corresponding object file. The object file does not contain the absolute address of where the generated code needs to be placed (a re-locatable code) on the program memory.
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process.
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory.
- A software utility called ‘Object to Hex file converter’ translates the absolute object file to corresponding hex file (binary file).

- **Advantages**
- **Efficient Code Memory & Data Memory Usage (Memory Optimization):**

  Leads to less utilization of Code memory and efficient utilization of Data memory.
- **High Performance:** Optimized code increases the system performance.
- **Low level Hardware Access:** Most of the code for low level programming make use of direct assembly code
- **Code Reverse Engineering:** It helps in understanding the technology by extracting the information.
- **Disadvantages**
- **High Development time:** Assembly language is much harder to code.
- **Developer dependency:** Since these are dependent on developers, modifying and upgrading on a later stage is extremely difficult.
- **Non portable:** This is the major drawback as target applications written in assembly instructions cannot be reused for another target applications hence not portable.

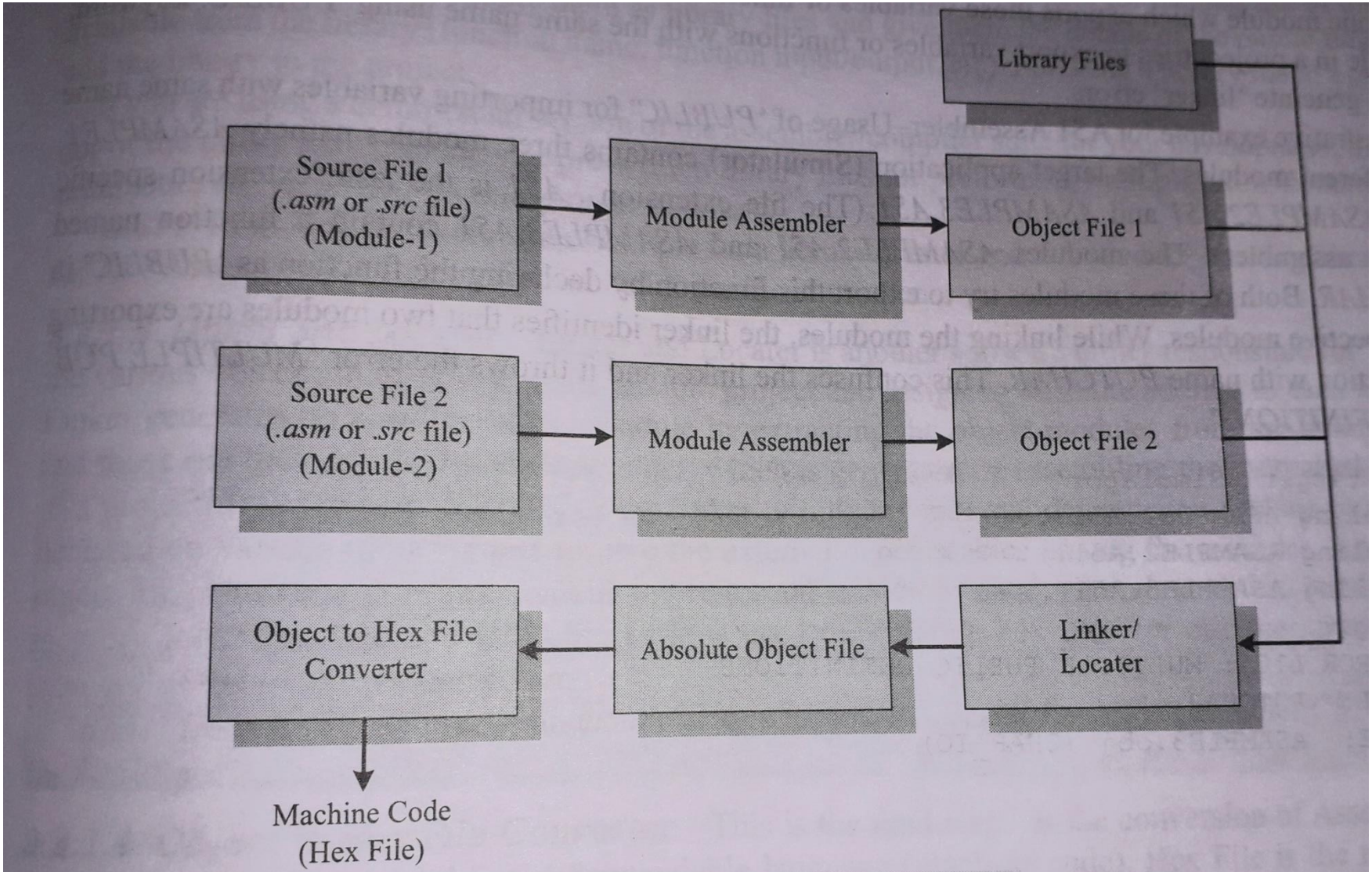


Fig. 9.1 **Assembly language to machine language conversion process**

- **High Level Language Based Development**
- Process almost similar to the assembly language based development.
- The embedded firmware is written in any high level language like C, C++.
- A software utility called ‘cross-compiler’ converts the high level language to target processor specific machine code.
- The cross-compilation of each module generates a corresponding object file.
- The software program called linker/locater is responsible for assigning absolute address to object files during the linking process.
- The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory.
- A software utility called ‘Object to Hex file converter’ translates the absolute object file to corresponding hex file (binary file).

- **Advantages**
- **Reduced Development time:** Major advantage here is developer needs little knowledge on the internal hardware details and architecture of the target processor/controller.
- **Developer independency:** Syntax used by most of the high level languages are universal. A program can be easily understood by anyone who knows the syntax of the language.
- **Portability:** An application written in high level language for a particular target processor can easily be converted to another target processor with little or less effort.
- **Disadvantages**
- Some of the Cross compiler available may not be so efficient.
- High level code snippets may not be so efficient in accessing low level hardware architecture.
- Investment required for high level language development is high.

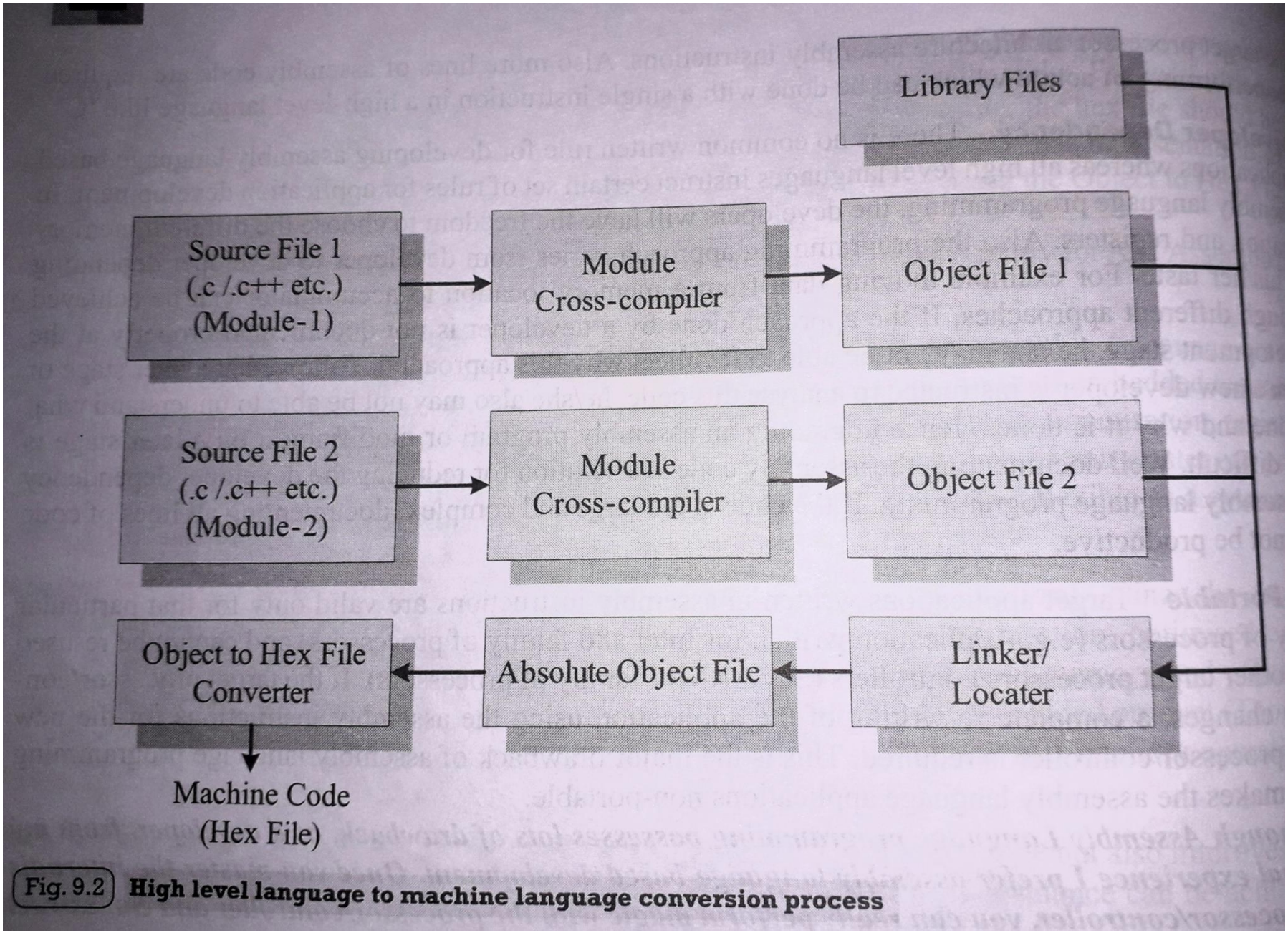


Fig. 9.2 **High level language to machine language conversion process**

- **Mixing assembly and High level Language**
- 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
- **Mixing Assembly Language with High level language like ‘C’**
- 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 built in support for implementing certain features like ISR.
- Mixing ‘C’ and assembly is little complicated.
- The programmer must be aware of how to pass parameters from the ‘C’ routine to assembly and values 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 High level language like ‘C’ with Assembly Language**
- The source code is already available in assembly language and routine written in a high level language needs to be included to the existing code.
- The programmer must be aware of how parameters are passed to the function and how values returned from the function and how function is invoked from the assembly language environment.
- Its implementation is cross compiler dependent and varies across compilers.
- **In line Assembly**
- Inline Assembly avoids the delay in calling an assembly routine from a ‘C’ code.
- Special keywords are used to indicate the start and end of Assembly instructions.
- E.g #pragma asm

  Mov A,#13H

  #pragma ensasm
- Keil C51 uses the keywords #pragma asm and #pragma endasm to indicate a block of code written in assembly.

Programming in Embedded C

- **'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.
- 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 specific binary files.
- Embedded C can be considered as a subset of conventional 'C' language.
- Embedded C supports all 'C' instructions and incorporates a few target processor specific functions/instructions.
- The standard ANSI 'C' library implementation is always tailored to the target processor/controller library files in Embedded C.
- A software program called 'Cross-compiler' is used for the conversion of programs written in Embedded C to target processor/controller specific instructions.

Compiler v/s Cross-Compiler

- Compiler is a software tool that converts a source code written in a high level language on top of a particular operating system running on a specific target processor architecture (E.g. Intel x86/Pentium).
- The source code is converted to the target processor specific machine instructions.
- A native compiler generates machine code for the same machine (processor) on which it is running.
- Cross compiler is the software tool 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.
- Keil C51compiler from Keil software is an example for cross-compiler for 8051 family architecture.