MODULE 3: Hardware Software Co design and Program Modelling: Fundamental issues in Hardware Software Co-design, Computational models in Embedded System Design.
Embedded Hardware Design and Development: Analog Electronic Components, Digital Electronic Components, VLSI & Integrated Circuit Design, Electronic Design Automation Tools.

HARDWARE SOFTWARE CO-DESIGN AND PROGRAM MODELING

Hardware Software Co-Design

In the traditional embedded system development approach, the hardware software partitioning is done at an early stage and engineers from the software group take care of the software architecture development and implementation, whereas engineers from the hardware group are responsible for building the hardware required for the product. There is less interaction between the two teams and the development happens either serially or in parallel. Once the hardware and software are ready, the integration is performed.

Fundamental Issues in Hardware Software Co-Design

The fundamental issues in hardware software co-design are:

1. Selecting the Model
2. Selecting the Architecture
3. Selecting the Language
4. Partitioning System Requirements into Hardware and Software

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 varieties of models from the requirements specification to the implementation aspect of the system design.

For example, at the specification stage, only the functionality of the system is in focus and not the implementation information. When the design moves to the implementation aspect, the information about the system component is revealed and the designer has to switch to a model capable of capturing the system's structure.

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. Commonly used architectures are:

- Controller architecture: Implements the finite state machine model (FSM) using a state register and two combinational circuits.
- Datapath architecture: Best suited for implementing the data flow graph model where the output is generated as a result of a set of predefined computations on input data.
- Complex Instruction Set Computing (CISC) architecture: Uses an instruction set representing complex operations and can perform a large complex operation with a single instruction.
- Reduced Instruction Set Computing (RISC) architecture: Reuses instruction set representing simple operations and it requires the execution of multiple RISC instructions to perform a complex operation.
- Very Long Instruction Word Computing (VLIW) architecture: Implements multiple functional units (ALUs, multipliers, etc.) in the datapath.
- Single Instruction Multiple Data (SIMD) architecture: A single instruction is executed in parallel with the help of the Processing Element. The SIMD architecture forms the basis of reconfigurable processor.
- Multiple Instruction Multiple Data (MIMD) architecture: Executes different instructions at a given point of time. The MIMD architecture forms the basis of multiprocessor systems.

Selecting the Language: A programming language captures a 'Computational Model' and maps it into architecture. There is no hard and fast rule to specify which 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. 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.

Partitioning System Requirements into Hardware and Software: 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:

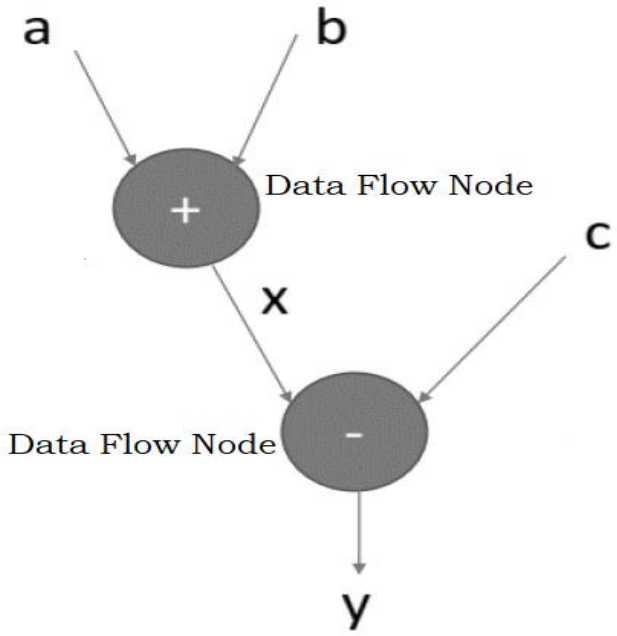
1. Data Flow Graph Model
2. Control Data Flow Graph Model
3. State Machine Model
4. Sequential Program Model
5. Concurrent/Communicating Process Model
6. Object-Oriented Model

Data Flow Graph Model

DFG model translates data processing requirements into data flow graph. It is a data driven model in which the program execution is determined by data. This model emphasizes on the data and operations on the data which transforms the input data to output data. Embedded applications which are computational intensive and data driven are modelled using the DFG model. DSP applications are typical examples for it.

DFG is a visual model in which the operation on the data (process) is represented using a block (circle) & data flow is represented using arrows. Inward arrow to the process (circle) represents input data & an outward arrow from the process (circle) represents output data in DFG notation.

Suppose one of the functions in an application contains the computational requirement $x = a + b$; and $y = x - c$. Figure illustrates the implementation.



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

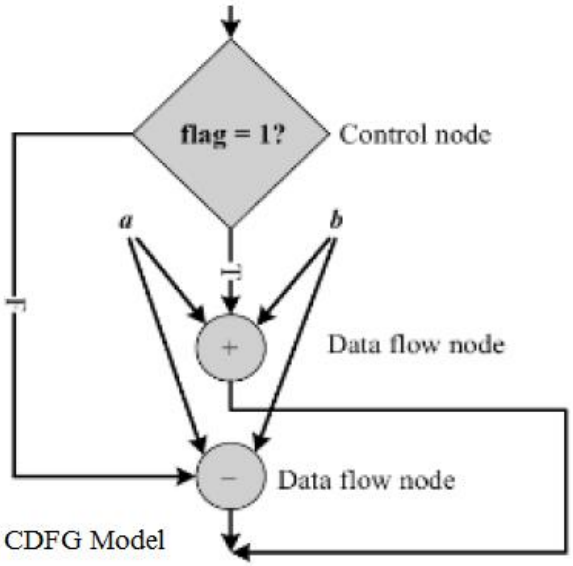
Control Data Flow Graph/Diagram (CDFG) Model

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.

Consider the implementation of the CDFG for following requirement: *If flag* = 1, $x = a + b$; *else* $y = a - b$; The CDFG model for the same is given in the figure. The control node is represented by a 'Diamond' block which is the decision making element in a normal flow chart based design. Decision on which process is to be executed is determined by control node.

Real world example for modelling the embedded application using CDFG is capturing & saving of the image to a format set by the user in a digital camera.

The decision on, in which format the image is stored (formats like JPEG, TIFF, BMP, etc.) is controlled by the camera settings, configured by the user.



State Machine Model

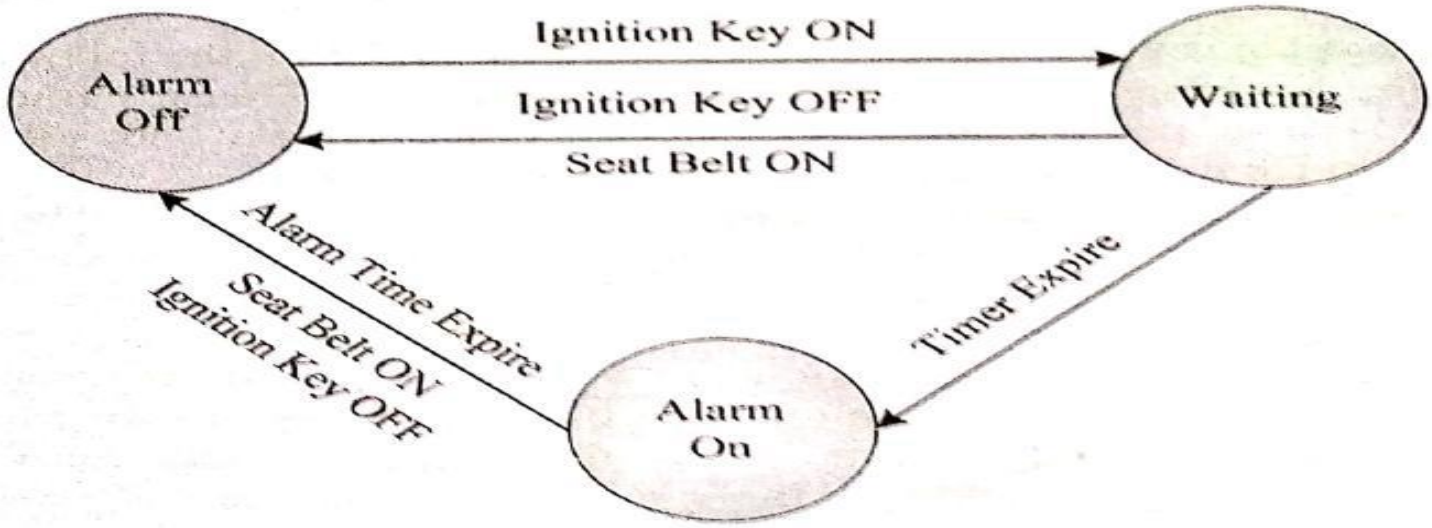
The State Machine model is used for modelling reactive or event-driven embedded systems whose processing behaviour is dependent on state transitions. Embedded systems used in the control and industrial applications are typical examples for event driven systems.

The State Machine model describes the system behaviour with 'States', 'Events', 'Actions' and 'Transitions'.

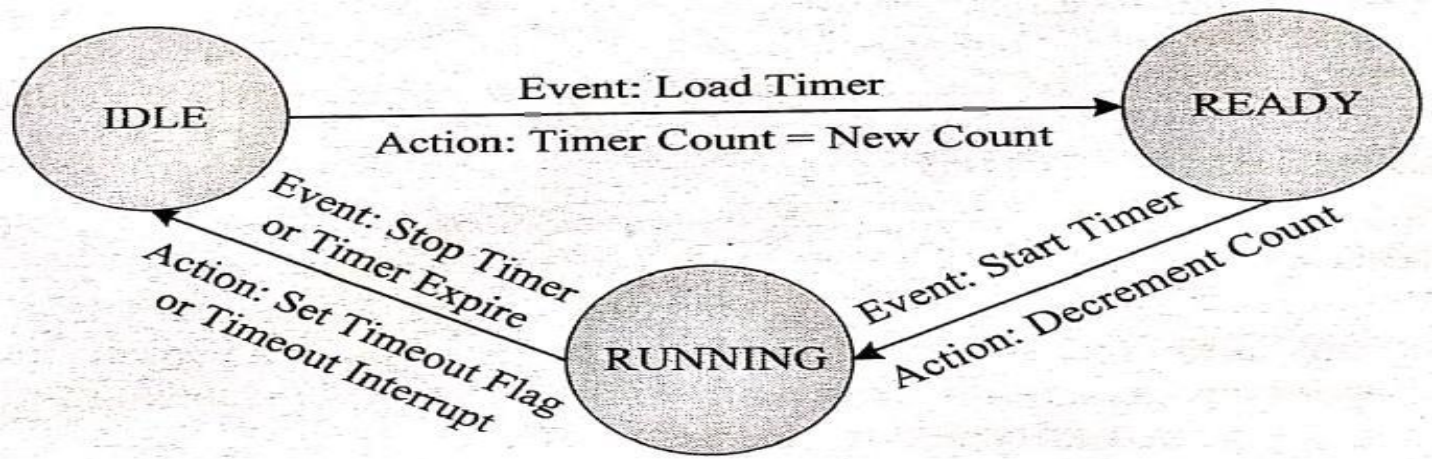
- State is a representation of a current situation.
- An event is an input to the state. The event acts as stimuli for state transition.
- Transition is the movement from one state to another.
- Action is an activity to be performed by the state machine.

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

- When the vehicle ignition is turned on and the seat belt is not fastened within 10 seconds of ignition ON, the system generates an alarm signal for 5 seconds.
- The Alarm is turned off when the alarm time (5 seconds) expires or if the driver/ passenger fasten the belt or if the ignition switch is turned off, whichever happens first.
- Here the states are 'Alarm Off', 'Waiting' and 'Alarm On' and the events are 'Ignition Key ON', 'Ignition Key OFF', 'Timer Expire', 'Alarm Time Expire' and 'Seat Belt ON'.
- Using the FSM, the system requirements can be modelled as given in following Figure.



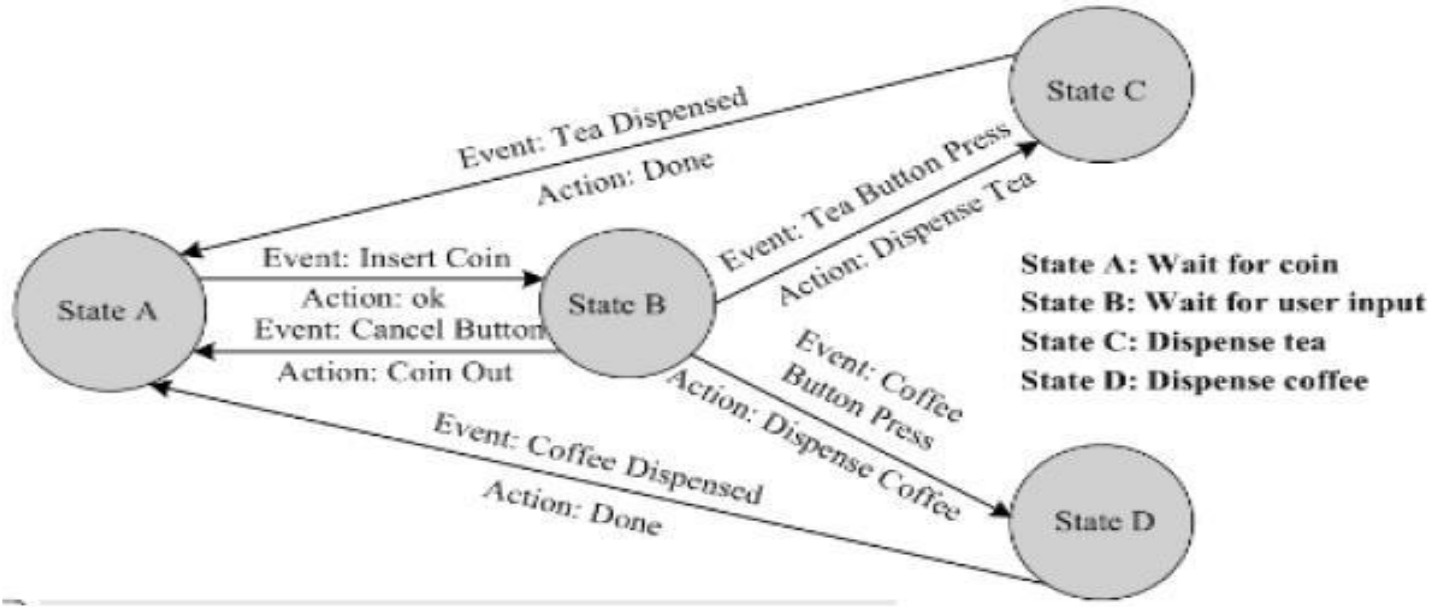
The wait state is implemented using a timer. The timer also has certain set of states and events for state transitions. Using the FSM model, the timer can be modelled as shown in the below figure.



Problem 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.

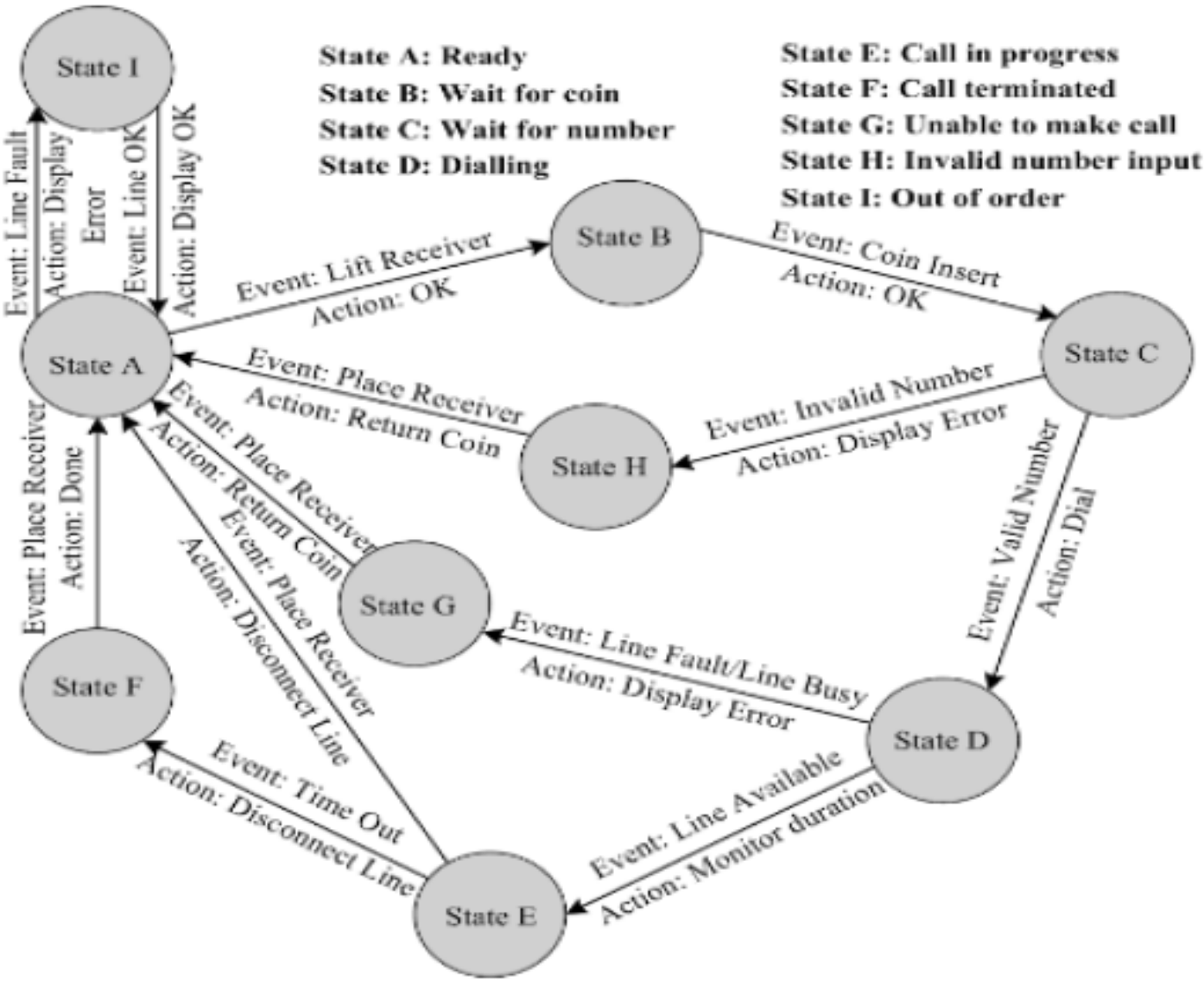
Solution: The FSM representation contains four states namely; 'Wait for coin' 'Wait for User Input', 'Dispense Tea' and 'Dispense Coffee'. The FSM representation for the above requirement is given in the below figure.



Problem 2: Design a coin operated public telephone unit based on FSM model for the following requirements.

- The calling process is initiated by lifting the receiver (off-hook) of the telephone unit.
- After lifting the phone the user needs to insert a 1 rupee coin to make the call.
- If the line is busy, the coin is returned on placing the receiver back on the hook (on-hook).
- 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.
- If the user doesn't insert another 1 rupee coin, the call is terminated on completing the 60 seconds time slot.
- The system is ready to accept new call request when the receiver is placed back on the hook (on-hook).
- The system goes to the 'Out of Order' state when there is a line fault.

Solution: The FSM representation for the above requirement is given in the below figure.



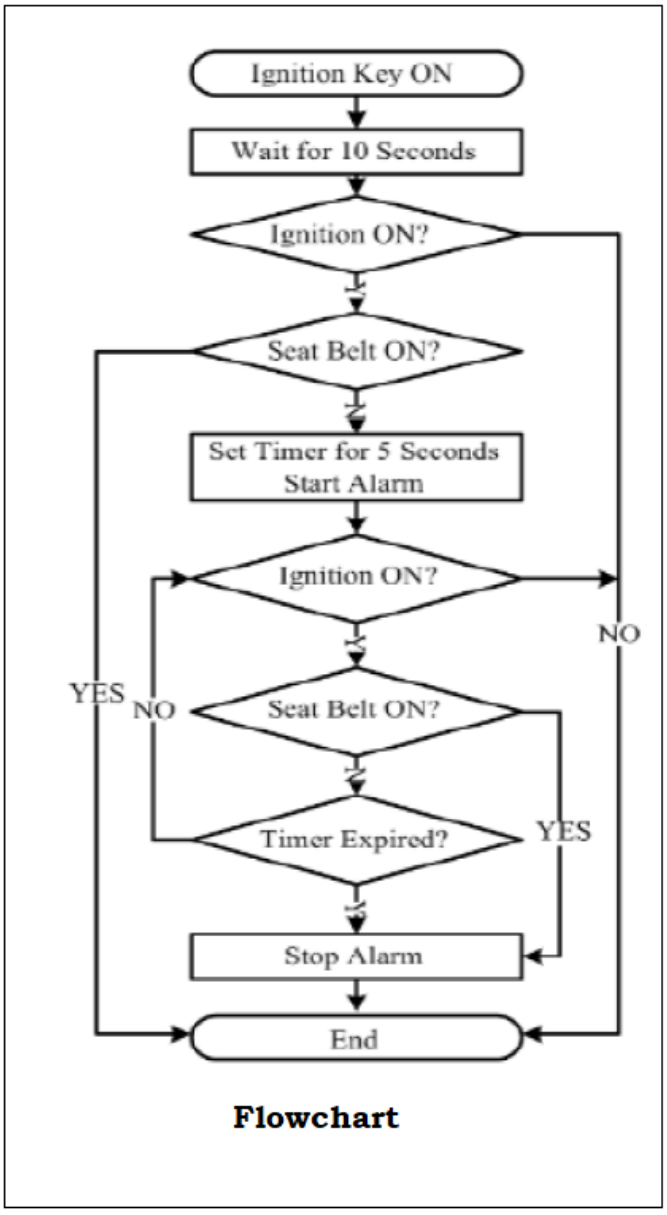
Sequential Program Model

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

The execution of functions in a sequential program model for the 'Seat Belt Warning' system is illustrated below and the Sequential Program Model (flowchart) for the same is also given in below figure:

```
#define ON 1
#define OFF 0
#define YES 1
#define NO 0
void seat_belt_warn ()
{   wait_10sec ();
    if (check_ignition_key () == ON)
    {
        if (check_seat_belt () == OFF)
        {
            set_timer (5);
            start_alarm ();
            while ((check_seat_belt ()
            == OFF) &&
            (check_ignition_key ()
             == OFF) &&
            (timer_expire () == ON));
            stop_alarm ();
        }
    }
}
```

Execution Flow



Flowchart

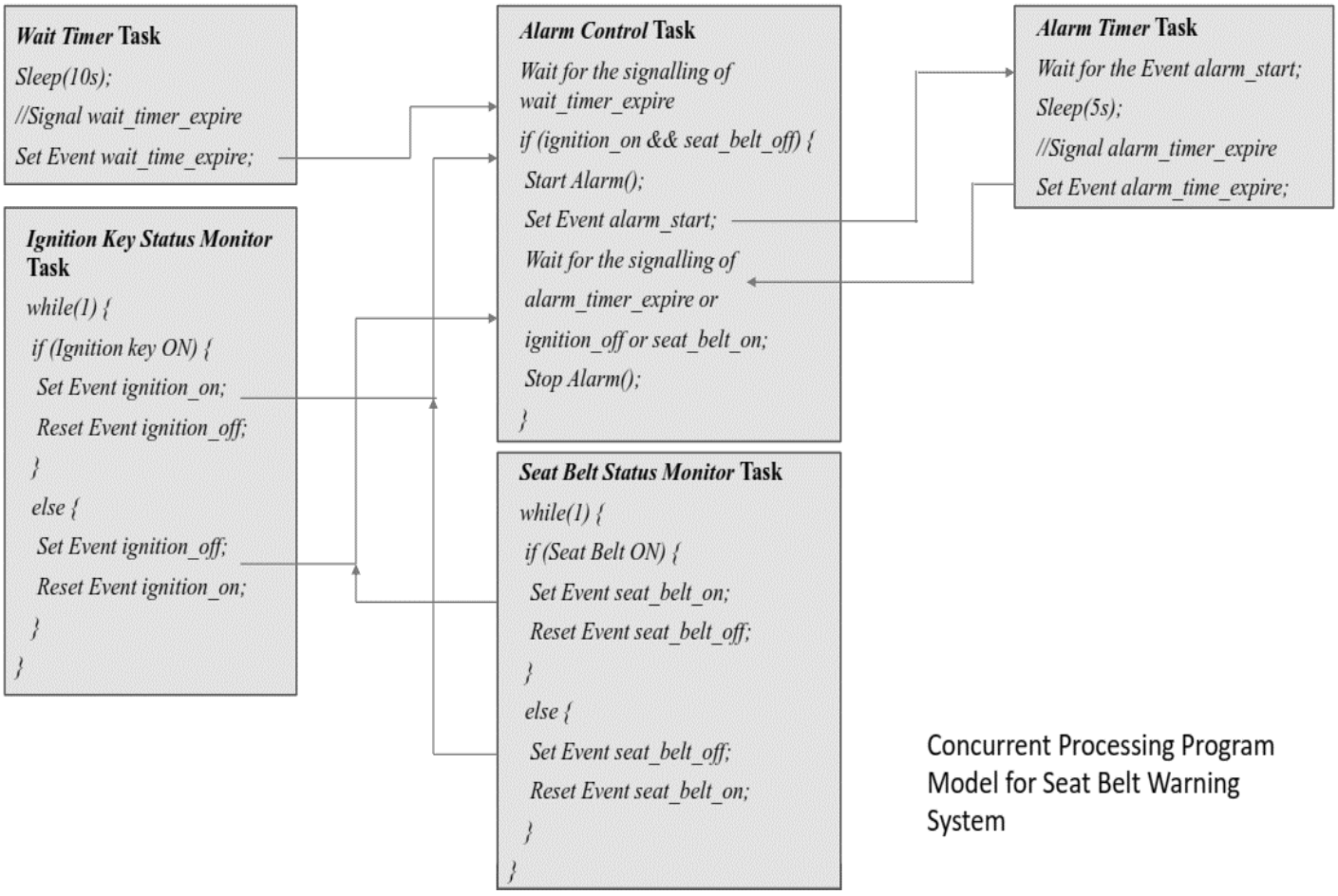
Concurrent/ Communicating Process Model

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, when the subtask under execution goes to a wait or sleep mode, by switching the task execution. However, concurrent processing model requires additional overheads in task scheduling, task synchronization and communication.

For example, consider the implementation of the 'Seat Belt Warning' system in concurrent processing model. We can split the tasks 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)

The tasks cannot be executed them randomly or sequentially. We need to synchronize their execution through some mechanism One way of implementing a concurrent model for the 'Seat Belt Warning' system is illustrated in the below figure:



Concurrent Processing Program Model for Seat Belt Warning System

Object-Oriented Model

The object-oriented model is an object based model for modelling system requirements. It disseminates a complex software requirement into simple well defined pieces called *objects*. Object-oriented model brings re-usability, maintainability and productivity in system design. In the object-oriented modelling, object is an entity used for representing or

modelling a particular piece of the system. Each object is characterized by a set of unique behaviour and state.

A class is an abstract description of a set of objects and it can be considered as a '*blueprint*' of an object. A class represents the state of an object through member variables and object behaviour through member functions. The member variables and member functions of a class can be private, public or protected. 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.