Module II
Hardware Software Co-Design and Program Modelling – Fundamental Issues, Computational Models- Data Flow Graph, Control Data Flow Graph, State Machine, Sequential Model, Concurrent Model, Object oriented model, UML

Traditional Embedded System Development Approach

- The hardware software partitioning is done at an early stage.
- Engineers from the software group take care of the software architecture development and implementation, and 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 and once the hardware and software are ready, the integration is performed.

Fundamental issues in Hardware Software Co-Design

- Model Selection
- Architecture Selection
- Language Selection
- Partitioning of System Requirements into Hardware and Software

Model Selection

- A Model captures and describes 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.
- The objectives vary with each phase.

Architecture Selection

- 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.
- Controller architecture, Datapath Architecture, Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), Very long Instruction Word Computing (VLIW), Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD) etc are the commonly used architectures in system design

Language Selection

- A programming Language captures a ‘Computational Model’ and maps it into architecture
- A model can be captured using multiple programming languages like C, C++, C#, Java etc for software implementations and languages like VHDL, System C, Verilog etc for hardware implementations
- 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 of System Requirements into H/w and S/w

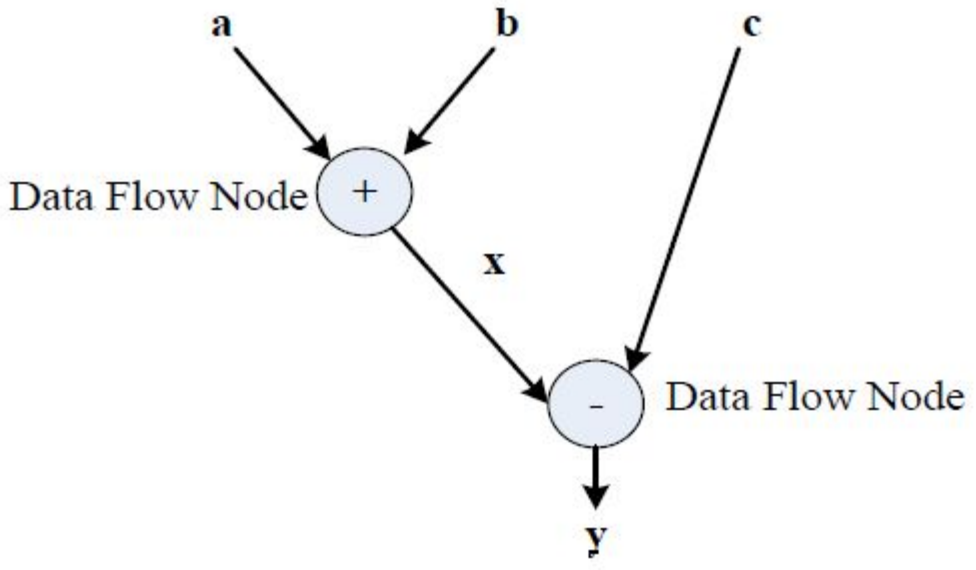
- Implementation aspect of a System level Requirement
- It may be possible to implement the system requirements in either hardware or software (firmware)
- Various hardware software trade-offs like performance, re-usability, effort etc are used for making a decision on the hardware-software partitioning.

Computational Models in Embedded Design

1. **Data Flow Graph/Diagram (DFG) Model**

- Translates the data processing requirements into a data flow graph.
- A data driven model in which the program execution is determined by data.
- Emphasizes on the data and operations on the data which transforms the input data to output data.
- 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.

- Best suited for modeling Embedded systems which are computational intensive.
- Eg. Model the requirement x = a+b; and y = x-c;



- ***Data path:*** *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 feed back to Input), events etc are examples for non-acyclic inputs.
- A DFG model translates the program as a single sequential process execution.

2. **Control Data Flow Graph/Diagram (CDFG) Model**

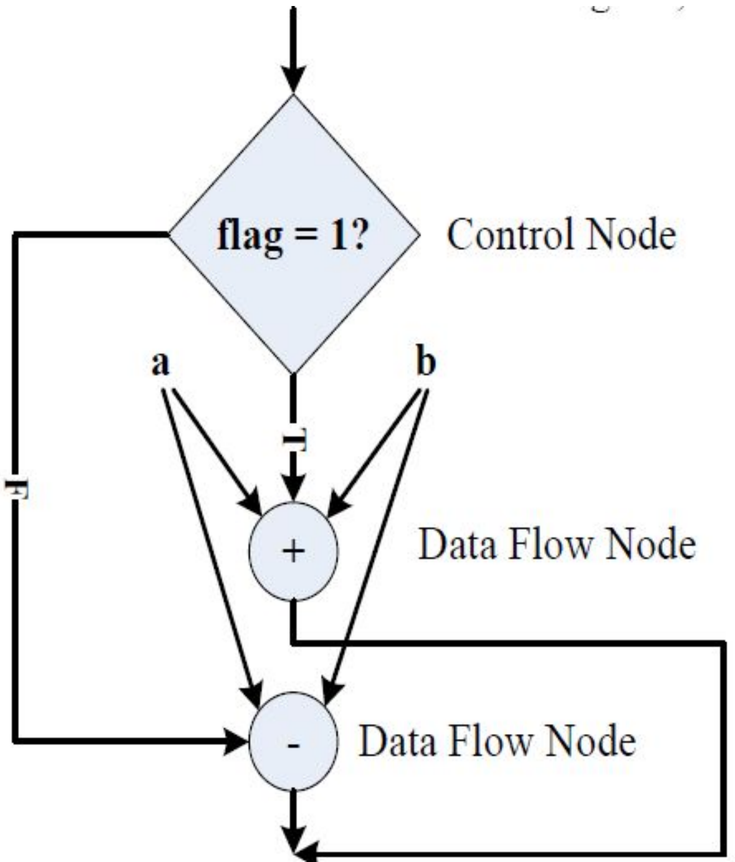
- Translates the data processing requirements into a data flow graph.
- Model applications involving conditional program execution.
- Contains both data operations and control operations
- Uses Data Flow Graph (DFG) as element and conditional (constructs) as decision makers.
- ***CDFG contains both data flow nodes and decision nodes***, whereas DFG contains only data flow nodes.

- 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.
- The **control node** is represented by a ‘**Diamond**’ block which is the decision making element in a normal flow chart based design
- Translates the requirement, which is modeled to a concurrent process model.
- The decision on which process is to be executed is determined by the control node.

- 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
- E.g. Model the requirement -

  If flag = 1, x = a + b;

  else y = a-b;



State Machine Model

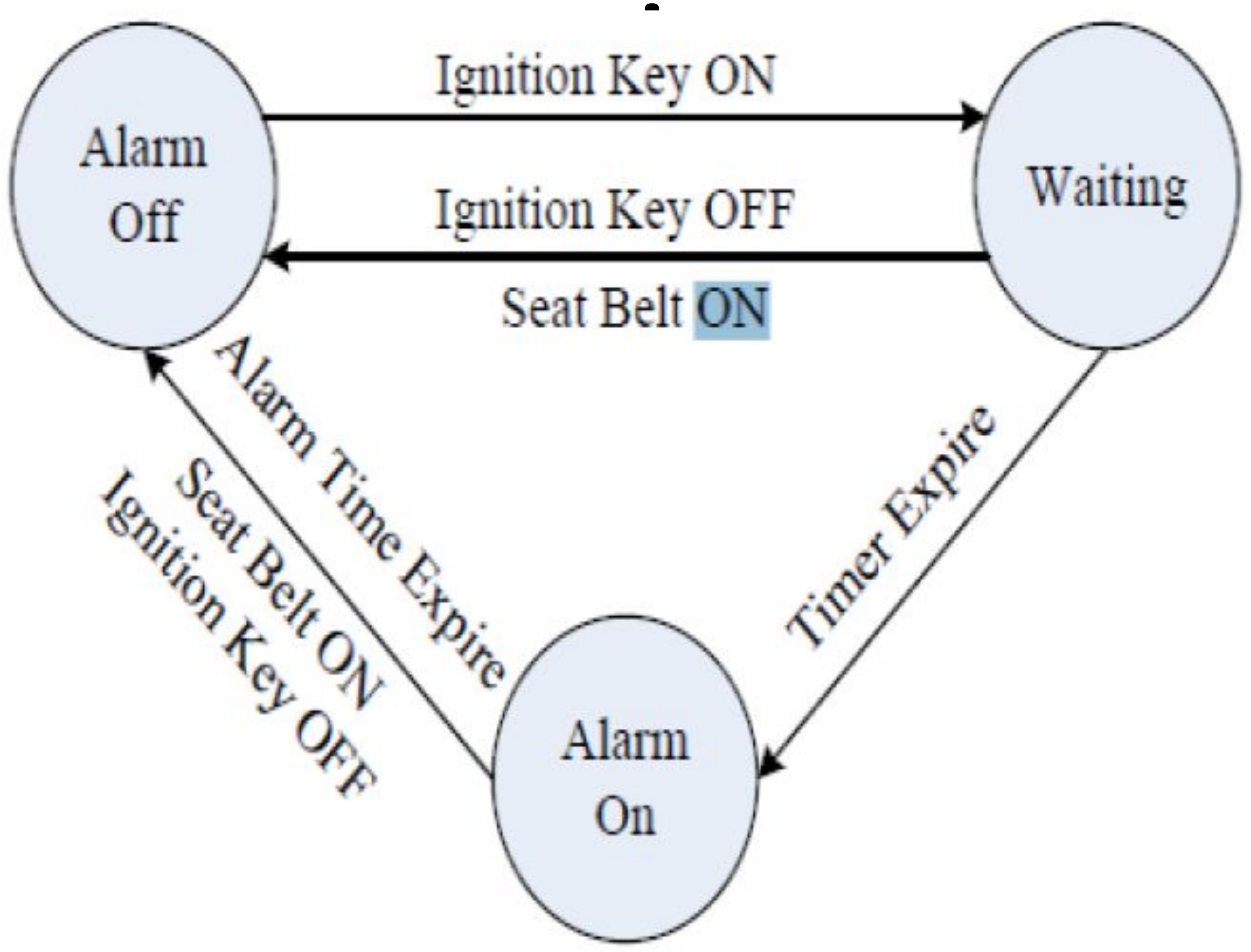
- Based on ‘States’ and ‘State Transition’
- Describes the system behavior with ‘States’, ‘Events’, ‘Actions’ and ‘Transitions’
- ***State*** *is a representation of a current situation.*
- An ***event*** *is an input to the state. The event acts as stimuli for state transition.*
- ***Transition*** *is the movement from one state to another.*
- ***Action*** *is an activity to be performed by the state machine.*

- **A Finite State Machine (FSM)** Model is one in which the number of states are finite. In other words the system is described using a finite number of possible states.
- Most of the time State Machine model translates the requirements into sequence driven program.
- The **Hierarchical/Concurrent Finite State Machine Model** (**HCFSM**) is an extension of the FSM for supporting concurrency and hierarchy.
- HCFSM extends the conventional state diagrams by the **AND, OR decomposition of States** together with inter level transitions and a **broadcast mechanism** for communicating between concurrent processes.
- HCFSM uses statecharts for capturing the states, transitions, events and actions. The Harel Statechart, UML State diagram etc are examples for popular statecharts used for the HCFSM modeling of embedded systems
- E.g. Automatic 'Seat Belt Warning' in an automotive

Requirements for automatic seat belt

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

Requirements for automatic seat belt

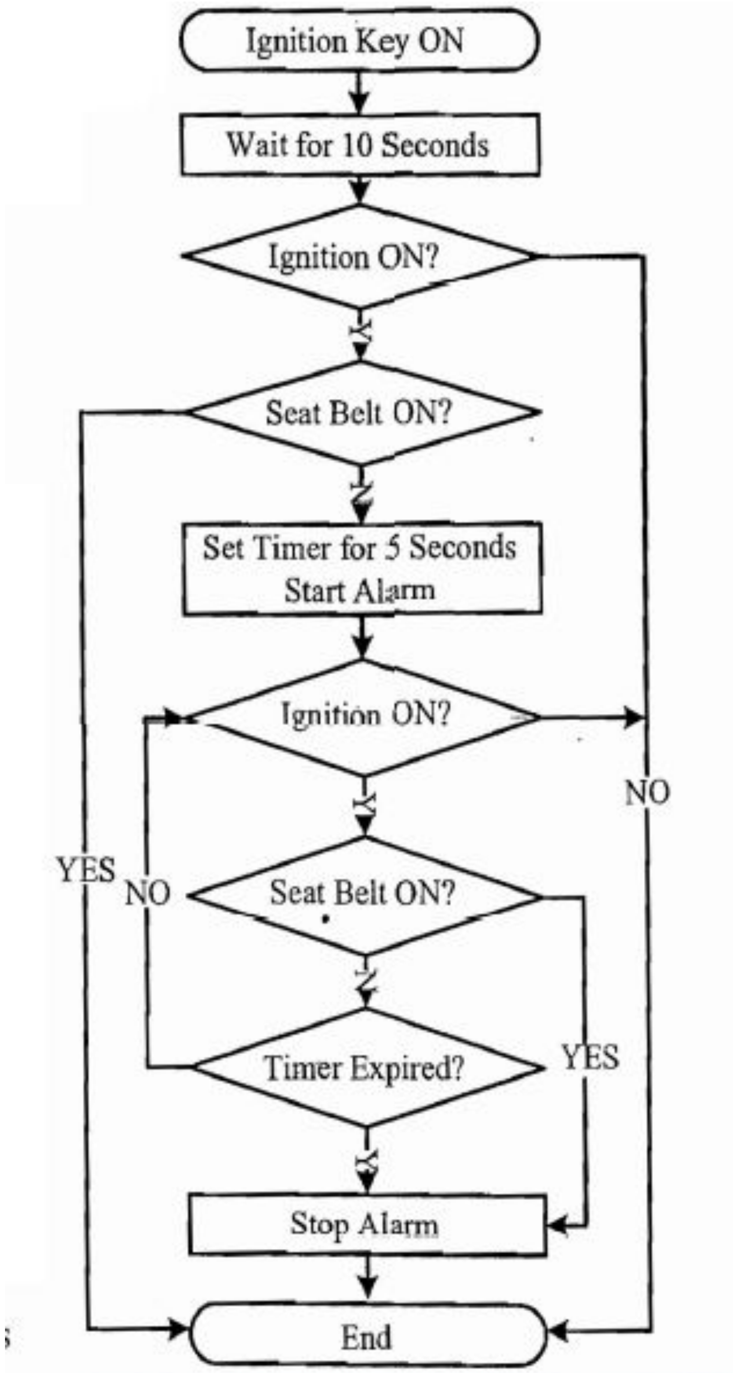


Sequential Program Model

- The functions or processing requirements are executed in sequence.
- The program instructions are iterated and executed conditionally and the data gets transformed through a series of operations.
- **FSMs** are good choice for sequential Program modeling.
- **Flow Charts** is another important tool used for modeling sequential program.
- The FSM approach represents the states, events, transitions and actions, whereas the Flow Chart models the execution flow.
- E.g. Automatic 'Seat Belt Warning' in an automotive.

Requirement:

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


Ignition Key ON
Wait for 10 Seconds
Ignition ON?
Y
NO
Seat Belt ON?
YES
N
Set Timer for 5 Seconds
Start Alarm
Ignition ON?
Y
Seat Belt ON?
NO
YES
N
Timer Expired?
Y
Stop Alarm
End

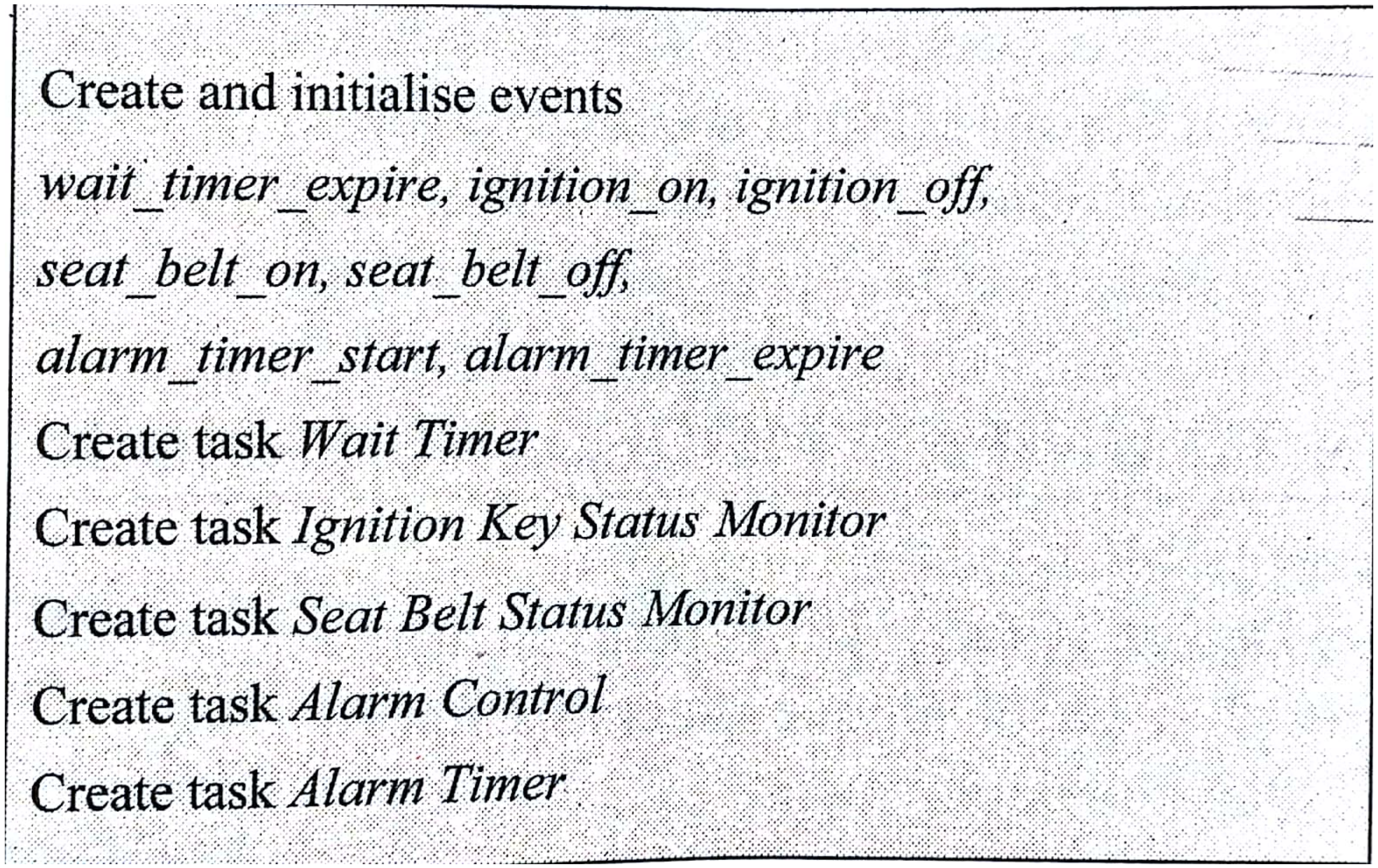
- The execution of functions in a sequential program model for the seat belt warning system :

```
#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_expire()==NO));
                        stop_alarm();
                }
        }
}
```

Concurrent/ Communicating Process Model

- Models **concurrently executing tasks/processes.**
- Certain processing requirements are easier to model in concurrent processing model than the conventional sequential execution.
- Sequential modeling is poor in processor utilization, when the task involves I/O waiting, sleeping for specific 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.
- Includes additional overhead like task scheduling, synchronization and communication.

Tasks for ‘Seat Belt Warning System’



Create and initialise events

wait_timer_expire, ignition_on, ignition_off,

seat_belt_on, seat_belt_off,

alarm_timer_start, alarm_timer_expire

Create task *Wait Timer*

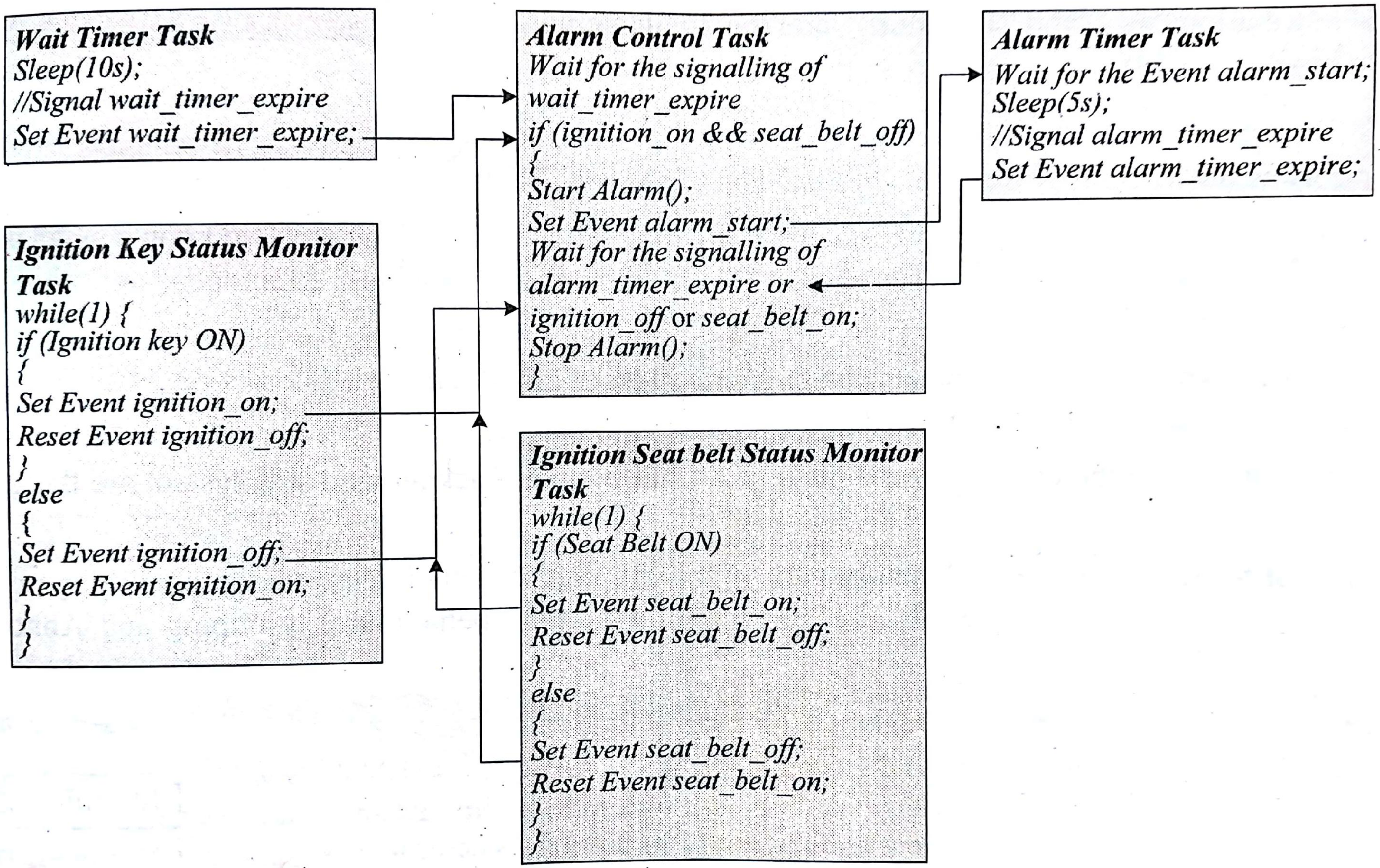
Create task *Ignition Key Status Monitor*

Create task *Seat Belt Status Monitor*

Create task *Alarm Control*

Create task *Alarm Timer*

Concurrent processing Program model for 'Seat belt Warning System'



```
Wait Timer Task
Sleep(10s);
//Signal wait_timer_expire
Set Event wait_timer_expire;
```

```
Alarm Control Task
Wait for the signalling of
wait_timer_expire
if (ignition_on && seat_belt_off)
{
Start Alarm();
Set Event alarm_start;
Wait for the signalling of
alarm_timer_expire or
ignition_off or seat_belt_on;
Stop Alarm();
}
```

```
Alarm Timer Task
Wait for the Event alarm_start;
Sleep(5s);
//Signal alarm_timer_expire
Set Event alarm_timer_expire;
```

```
Ignition Key Status Monitor
Task
while(1) {
if (Ignition key ON)
{
Set Event ignition_on;
Reset Event ignition_off;
}
else
{
Set Event ignition_off;
Reset Event ignition_on;
}
}
```

```
Ignition Seat belt Status Monitor
Task
while(1) {
if (Seat Belt ON)
{
Set Event seat_belt_on;
Reset Event seat_belt_off;
}
else
{
Set Event seat_belt_off;
Reset Event seat_belt_on;
}
}
```

- The tasks in a concurrent program cannot execute randomly or sequentially.
- The concurrent processing model is **used to model 'Real Time' Systems.**
- Various techniques like 'Shared Memory', 'Message Passing', 'Events' etc are used for communication and synchronizing between concurrently executing processes.

OBJECT ORIENTED MODEL

- Object based model.
- Disseminates complex software requirements into simple well defined pieces called **objects**.
- OO models brings
  - Reusability
  - Maintainability
  - Productivity

  in software design

Object & Class

- ***Object*** is an entity used for representing or modeling 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
  - Blueprint of an object
  - Represents state of an object through member variable
  - Represents object behaviour through member functions.

- The member variables and member functions of a class can be private, public or protected.
  - ***Private:*** accessible only within the class
  - ***Public:*** Accessible within the class as well as outside the class
  - ***Protected:*** Protected from external access. Classes derived from a parent class can access protected functions and variables.
- The concept of objects and classes brings abstraction, hiding and protection.

Unified Modeling Language (UML)

- **Unified Modeling Language (UML)** is a visual modeling language for Object Oriented Design (OOD). UML helps in all phases of system design through a set of unique diagrams for requirements capturing, designing and deployment.

UML Building Blocks

The following are the main building block of UML diagrams.

▯ **Things**

- An abstraction of the UML Model

▯ **Relationships**

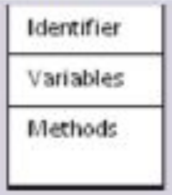
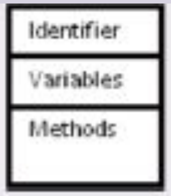
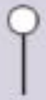
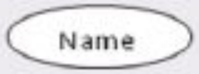
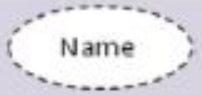
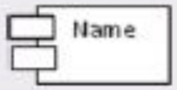
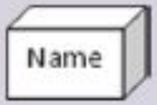
- An entity which express the type of relationship between UML elements (objects, classes etc)

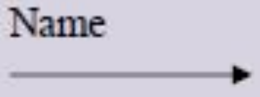
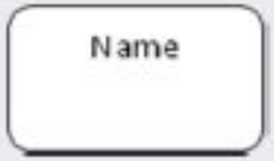
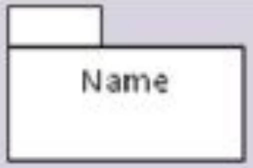
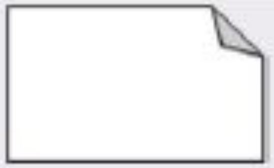
▯ **Diagrams**

- UML Diagrams give a pictorial representation of the static aspects, behavioral aspects and organization and management of different modules (classes, packages etc) of the system

1. Things

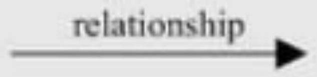
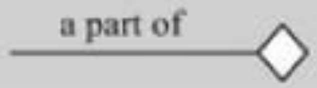
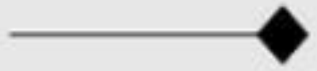
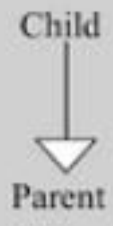
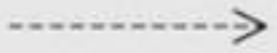
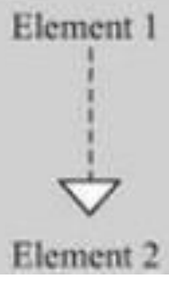
- **Structural things:** Represents mostly the **static parts** of a UML model. They are also known as '**classifiers**'. Class, interface, use case, use case realization (collaboration), active class, component and node are the structural things in UML.
- **Behavioral things:** Represents mostly the **dynamic parts** of a UML model. Interaction, state machine and activity are the behavioral things in UML.
- **Grouping things:** are the organizational parts of a UML model. Package and sub-system are the grouping things in UML.
- **Annotational things:** Are the explanatory parts of a UML model. *Note is the Annotational thing in UML.*

Thing	Element	Description	Representation
Structural	Class	A template describing a set of objects which share the same attributes, relationships, operations and semantics. It can be considered as a blueprint of object.	
	Active Class	Class presenting a thread of control in the system. It can initiate control activity. Active class is represented in the same way as that of a class but with thick border lines.	
	Interface	A collection of externally visible operations which specify a service of a class. It is represented as a circle attached to the class	
	Use case	Defines a set of sequence of actions. It is normally represented with an ellipse indicating the name.	
	Collaboration (Use case Realization)	Interaction diagram specifying the collaboration of different use cases. It is normally represented with a dotted ellipse indicating the name.	
	Component	Physical packaging of classes and interfaces.	
	Node	A computational resource existing at run time. Represented using a cube with name.	

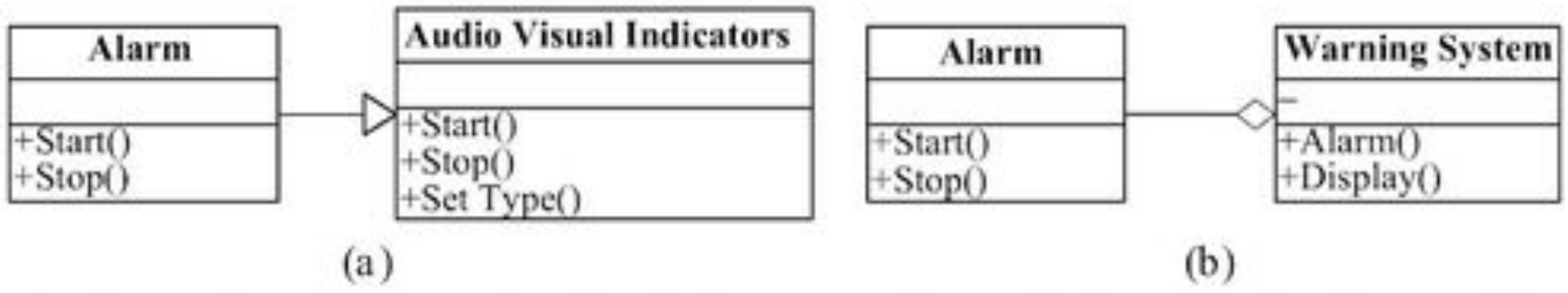
Behavioral	Interaction	Behavior comprising a set of objects exchanging messages to accomplish a specific purpose. Represented by arrow with name of operation	
	State Machine	Behavior specifying the sequence of states in response to events, through which an object traverses during its lifetime.	
Grouping	Package	Organizes elements into packages. It is only a conceptual thing. Represented as a tabbed folder with name.	
Annotational	Note	Explanatory element in UML models. Contains formal informal explanatory text. May also contain embedded image.	

2. Relationships in UML

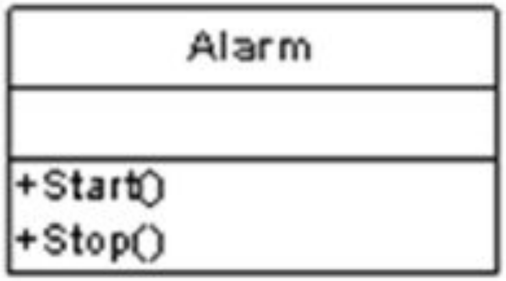
- Expresses the types of relationship between UML objects (objects, classes etc)

Relationship	Description	Representation
Association	It is a structural relationship describing the link between objects. The association can be one-to-one or one-to-many. Aggregation and Composition are the two variants of Association.	
Aggregation	It represents is "a part of" relationship. Represented by a line with a hollow diamond at the end.	
Composition	Aggregation with strong ownership relation to represent the component of a complex object. Represented by a line with a solid diamond at the end.	
Generalisation	Represents a parent-child relationship. The parent may be more generalised and child being specialised version of the parent object.	
Dependency	Represents a relationship in which one element (object, class) uses or depends on another element (object, class). Represented by a dotted arrow with head pointing to the dependent element.	
Realisation	The relationship between two elements in which one element realises the behaviour specified by the other element.	

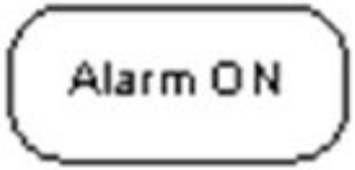
The Seat Belt Warning System – UML modeling



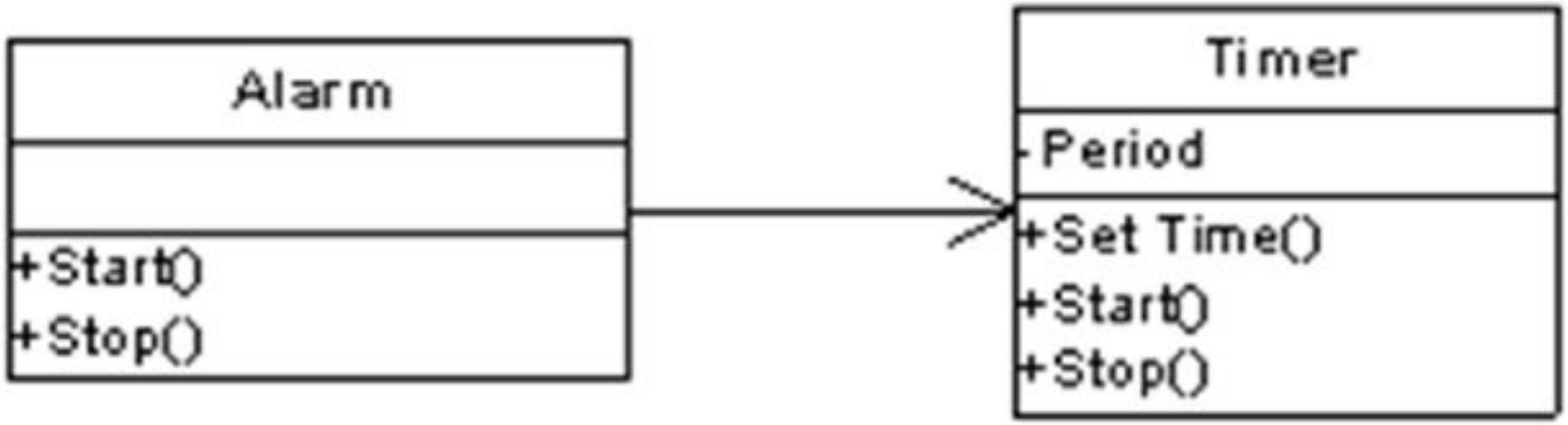
(a) Alarm is a special type of Audio Visual Indicator (Generalisation), (b) Alarm is a part of Warning System (Aggregation)



The Alarm Class



State Representation for 'Alarm ON' state



Alarm – Timer Class interaction for the Seat belt Warning System

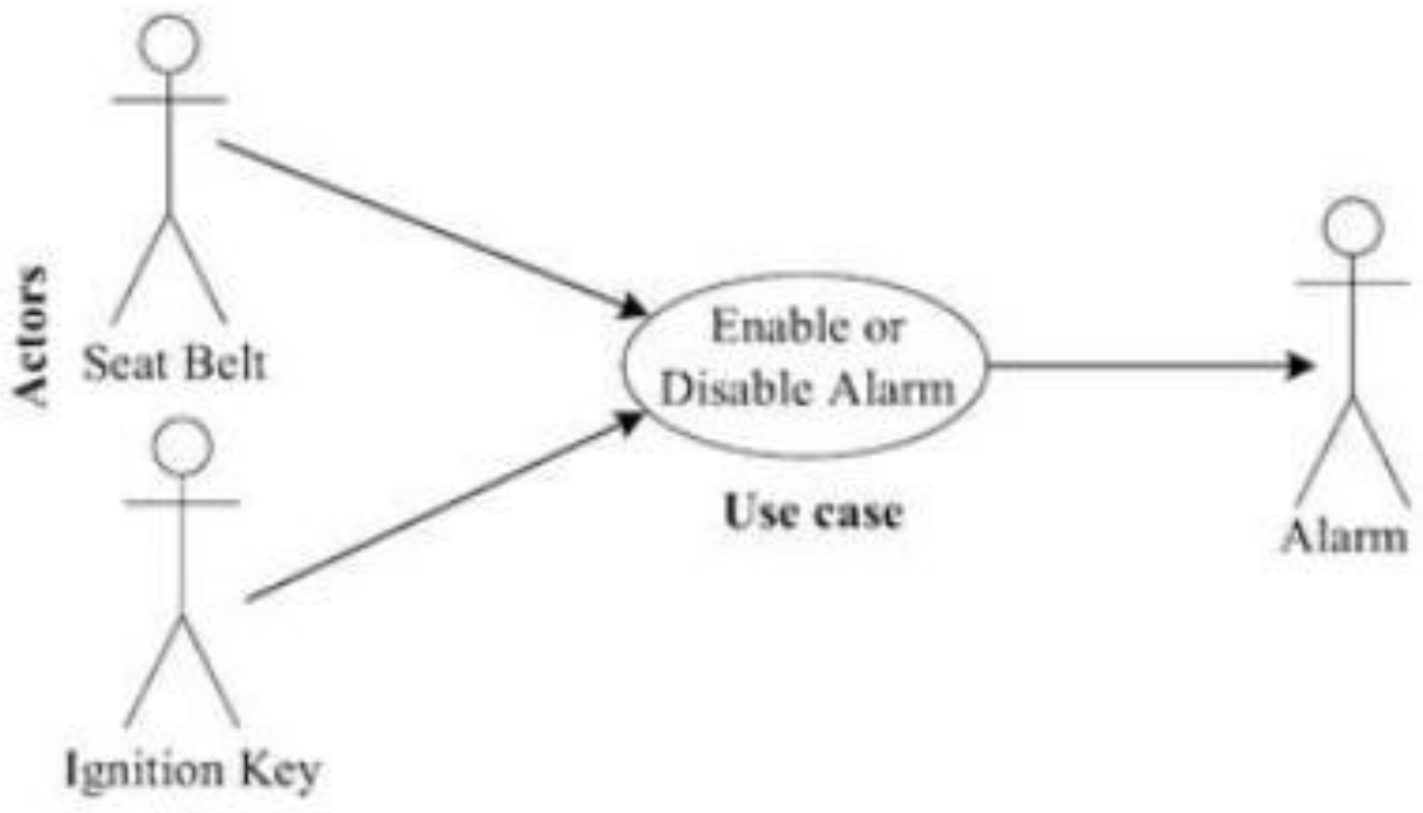
3. UML diagrams

- UML diagrams give pictorial representation of static aspects, behavioral aspects and organization and management of different modules (into classes, packages) of the systems.
- **Static Diagrams:** Diagram representing the static (structural) aspects of the system. Class Diagram, Object Diagram, Component Diagram, Package Diagram, Composite Structure Diagram and Deployment Diagram falls under this category.

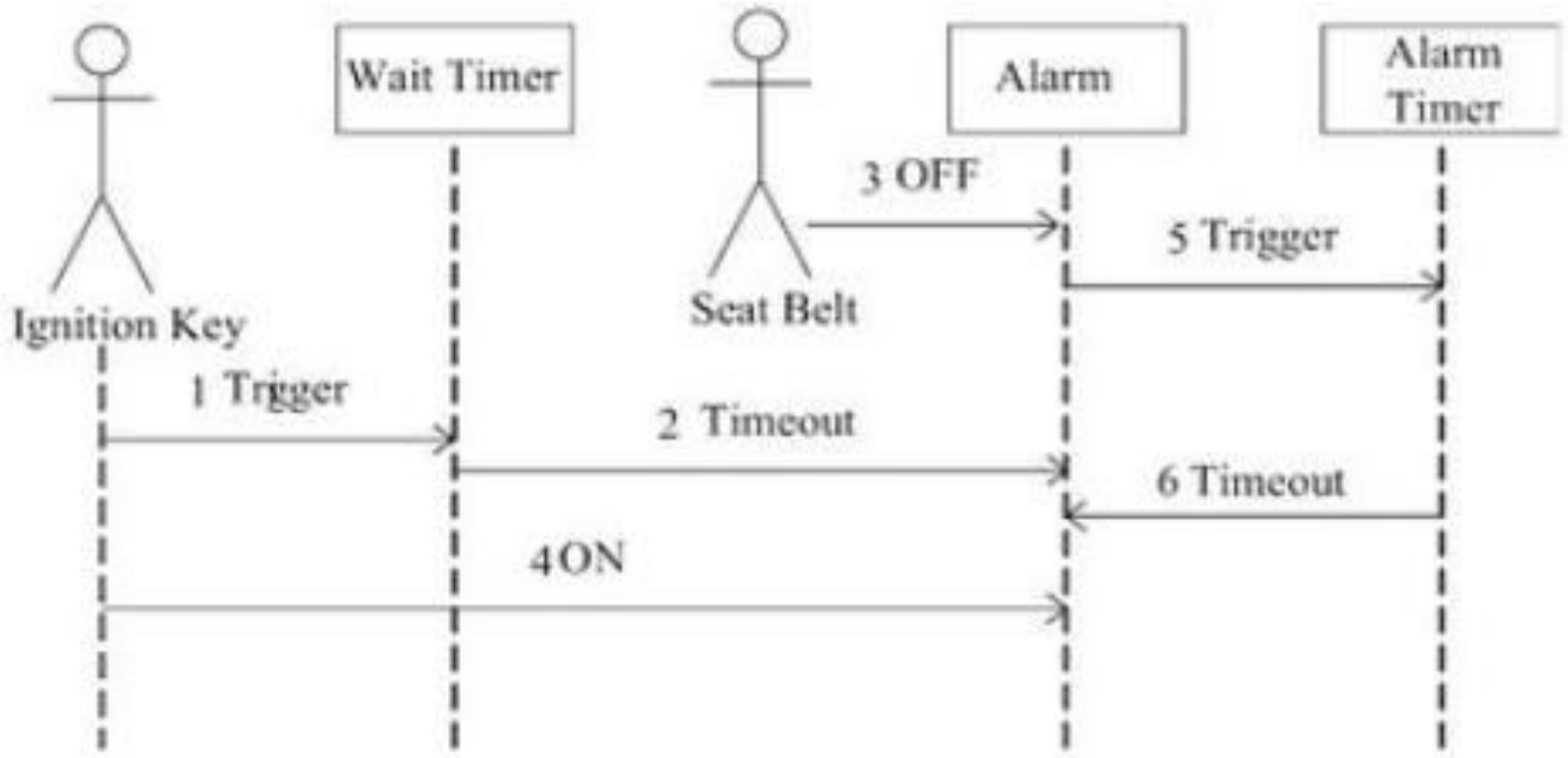
Diagram	Description
Object Diagram	Gives a pictorial representation of a set of objects and their relationships. It represents the structural organization between objects.
Class Diagram	Gives a pictorial representation of the different classes in a UML model, their interfaces, the collaborations, interactions and relationship between the classes etc. It captures the static design of the system.
Component Diagram	It is a pictorial representation of the implementation view of a system. It comprises of components (Physical packaging of classes and interfaces), relationships and associations among the components.
Package Diagram	It is a representation of the organization of packages and their elements. Package diagrams are mostly used for organizing use case diagrams and class diagrams.
Deployment Diagram	It is a pictorial representation of the configuration of run time processing nodes and the components associated with them.

Behavioral Diagram

- It represents the dynamic(behavioral) aspects of the system.
  - Use case diagram
  - sequence diagram
  - state diagram
  - communication diagram
  - activity diagram
  - timing diagram
  - interaction diagram



Use case diagram for seat belt warning system



Sequence diagram for seat belt warning system

Diagram	Description
Use Case diagram	Use Case diagrams are used for capturing system functionality as seen by users. It is very useful in system requirements capturing. Use case diagram comprise use cases, *actors* (users) and the relationship between them. In use case diagram, an *actor* is one (or something) who (or which) interacts with the system and use case is the sequence of interaction between the actor and system.
Sequence diagram	Sequence diagram is a type of interaction diagram representing object interactions with respect to time. It emphasises on the time ordering of messages. Best suited for the interaction modelling of real-time systems.
Collaboration (Communication) diagram	Collaboration or Communication diagram is a type of interaction diagram representing the object interaction and 'how they are linked together'. It gives emphasis to the structural organisation of objects that send and receive messages. In short, it represents the collaboration of objects using messages.
State Chart diagram	A diagram showing the states, transitions, events and activities similar to a state machine representation. Best suited for modelling reactive systems.
Activity diagram	It is a special type of state chart diagram showing activity to activity transition in place of state transition. It emphasises on the flow control among objects.