MODULE - 2.

HARDWARE SOFTWARE CO-DESIGN & PROGRAM MODELLING

HARDWARE SOFTWARE CO-DESIGN

→ Traditional ES development.

- Software, hardware partitioning is done first
- S/w group take care of S/w architecture development
  h/w " " h/w building.
- Less interaction b/w 2 teams
- Once hardware & S/w are ready, the integration is performed.

→ Co-design Approach for ES development

- H/w & S/w are co-developed (or concurrently developed) instead of independently developing both
- Verification is also done concurrently.
- It tries to exploit the synergy (interaction) of h/w & S/w with the goal to optimize & satisfy design constraints such as cost, performance and power of the final product.
- Steps < sym design / Architecture design.
  → The product requirements captured from the customer are converted into system level needs or processing requirements (or functional requirements)

Sym design → These requirements are then transferred into functions which can be simulated & verified against performance & functionality

architecture design → The partitioning of sym level requirements into h/w & s/w take place during the architecture design phase. The partitioning is performed based on the hardware-s/w trade-offs.

→ The architectural design results in the detailed behavioural description of the h/w requirement & the definition of the s/w required for the hardware.

→ The processing requirement behaviour is usually captured using computational models (eg:- Data flow diagram model, state m/c model, sequential program model etc) and ultimately the models representing the s/w processing requirements are translated into firmware implementation using programy languages.

- h/w & s/w trade-offs

→ Certain sym. level processing requirements may be possible to develop in either h/w or s/w.

The actual selection is based on < the trade-offs / actual sym requirements.

eg:- Some multimedia codec requirement.

Here trade-off is the performance & Re-configurability

h/w implementation → efficient, low processing, low power, optimised

s/w " → re usable, re configurable etc.

→ Some imp. h/w s/w trade-offs in ES design are.

1) Processing speed & Performance
2) Frequency of change (Re-configurability)
3) Memory size and gate count
4) Reliability
5) Man Hours (Effort) & cost.

FUNDAMENTAL ISSUES IN HARDWARE-SOFTWARE CO-DESIGN.

1) Selecting the Model

- Model → is a formal sym consisting of objects and composition rules. eg:- DFD, state m/c model, sequential program model etc
- In h/w s/w co-design, models are used for capturing & describing the sym characteristics
- Choosing a model is a difficult job. Most often designers switch b/w a variety of models from the requirement specification to the implementation aspect of the sym design. (because the objective varies with each phase)

2) Selecting the Architecture

- Architecture specifies how a sym is going to implement in terms of the number & types of different components & the interconnection among them.

- Commonly used architectures in sys design are.

Application specific architecture
- → Controller architecture
- → Datapath "

General purpose architecture
- → CISC
- → RISC.

Parallel processing architecture
- → Very Long Instruction Word Computing (VLIW)
- → SIMD (Single Instr" Multiple Data)
- → MIMD (Multiple " " ") etc.

- **Controller Architecture**
  - → implements the FSM model using a state register & two combinational circuits. The state reg. holds the present state & the combinational circuits implement the logic for next state and o/p.
- **Datapath Architecture**
  - → implements the data flow graph model, where the o/p is generated as a result of a set of predefined computations on the i/p data.
  - → Datapath → a channel b/w the i/p & o/p and it may contain registers, counters, reg. files, memories & ports along with high speed arithmetic units.
  - → Ports connect the datapaths to multiple buses
  - → Most of the time, arithmetic units are connected in parallel with pipelining support for bringing high performance.

- CISC & RISC (Refer µp/µc - core of ES)
- VLIW
  → implements multiple functional units (ALU, multipliers etc) in the datapath.
  → The VLIW architecture packages one standard instruction per functional unit of the datapath.
- SIMD & MIMD
  → Parallel processing architecture implements multiple concurrent Processing Elements (PEs) and each processing element may associate a datapath containing registers & local memory. eg:- SIMD, MIMD
  → SIMD :- Single Instruction is executed in parallel with the help of PEs. They form the basis of Re-configurable processors.
  → MIMD :- PEs execute different instructions at a given point of time. They form the basis of multiprocessed systems.

3) Selecting the Language

- A programming lang. captures a "Computational Model" and maps it into architecture.
- A model can be captured using multiple programming languages.
  ↓
  eg:- C, C++, C#, Java, etc for S/W implementation
  VHDL, System C, Verilog for H/W "

- Also, a single language can be used for capturing a variety of models.
- Certain langs. are good for capturing certain models.
  eg:- C++ is good for capturing objected oriented model.
- Lang. selection criteria → Lang. should capture the model easily.

4) Partitioning System Requirements into hardware & Software

- It is possible to implement the system requirements in either h/w or s/w.
- The decision depends on various h/w s/w trade-offs. like.

(Refer Prev. Page)

COMPUTATIONAL MODELS IN EMBEDDED DESIGN

1) Data Flow Graph / Diagram (DFG) Model

→ It 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.

→ It focus on data & operations on the data which transforms the i/p data to o/p data.

→ It is a visual model in which

○ → process (or operation on data)

→ → dataflow. [i/p data →○ ○→ o/p data]

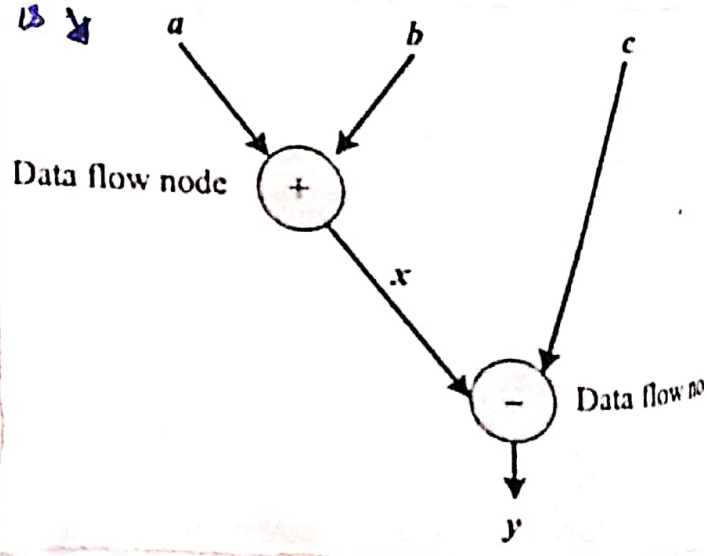
→ Embedded applications which are computational intensive & data driven are modeled using DFG.

eg :- DSP Applications.

→ eg : Suppose the computational requirements are.

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

the corresponding DFG is



→ Here datapath is the data flow path from i/p to o/p.

→ Acyclic DFG (ADFG) → It doesn't contain multiple values for i/p variable & multiple o/p values for a given set of input(s).

→ Feedback inputs, 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)

→ Used for modelling applications involving conditional program execution.

→ This model contains both data operations and control operations.

→ It uses DFG as element & conditional constructs as decision makers.

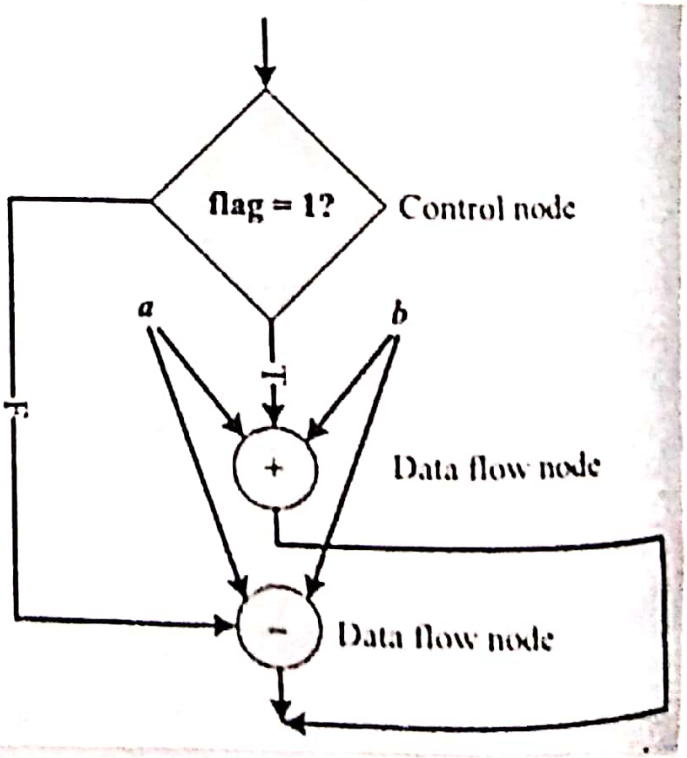
→ It contains. < data flow nodes & decision nodes

→ Consider the requirement.

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

The CDFG model for this is shown in fig.

→ Control node (decision making element).



→ The decision on which process is to be executed is determined by the control node.

→ Real World eg :- Capturing & saving of the image to a format set by the user in a digital still 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.

3) State Machine Model

→ Used for modelling Reactive or Event-driven ESs, whose processing behaviour are depended on state transitions.

eg. for event driven ESs → ESs used in control & Industrial applications.

→ This model describes the system behaviour with

- States : A representation of a current situation
- Events : An I/P to the state, which act as stimuli for state transition
- Actions : An activity to be performed by the state machine.
- Transition : Movement from one state to another.

→ FSM (Finite state Machine) model

- In FSM, the no. of states are finite. ie, the sys is described using a finite number of possible states.

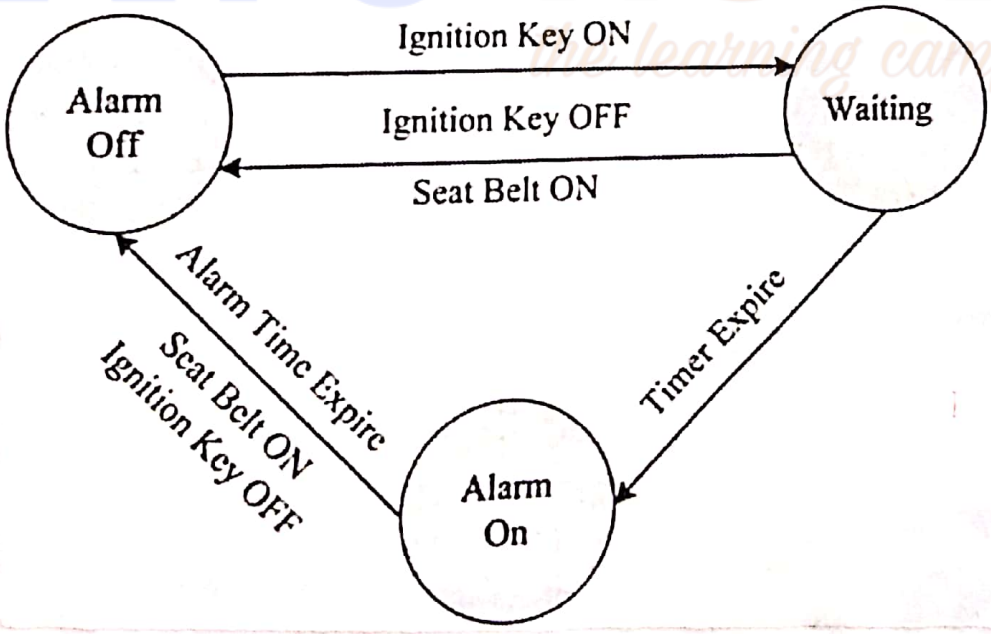
eg1) FSM model for Automatic seat belt warning system.

Suppose the system requirements are,

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 sec) expires or if the driver/passenger fastens the belt or if the ignition switch is turned off, whichever happens first.

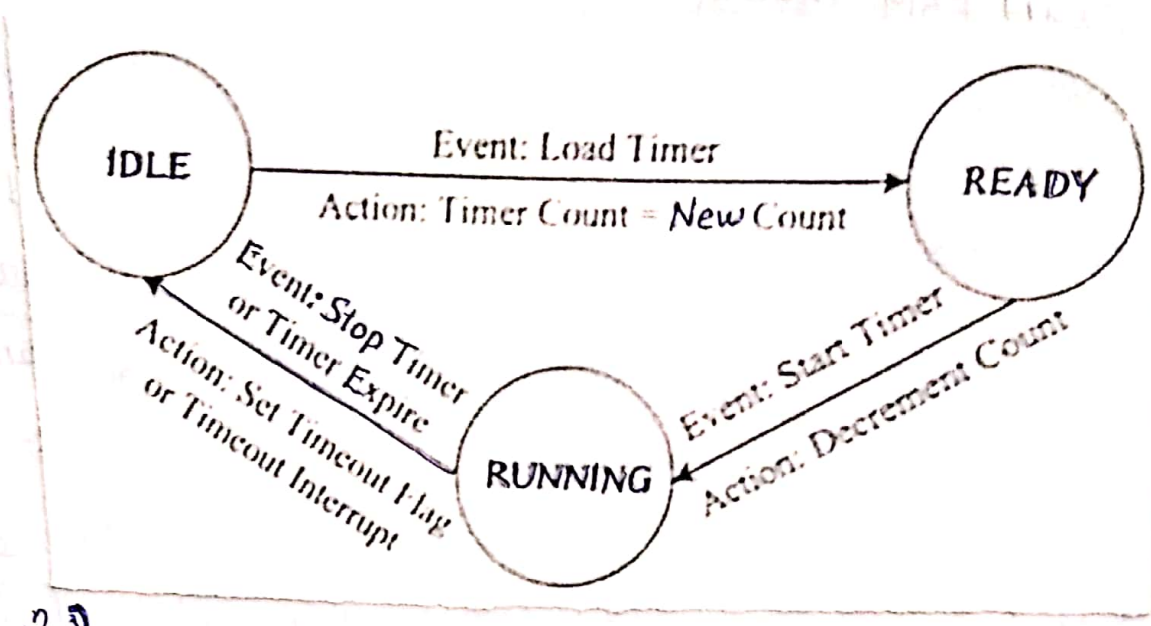
These requirements can be modelled using FSM as shown below.



(expln)

Here wait state is implemented using a timer. The timer also has certain set of states and events for state transitions.

The FSM model for timer is shown below,



(expln)

eg2 :- ~~And~~ FSM Model for Automatic Tea/Coffee Vending Machine

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 & take back the coin.

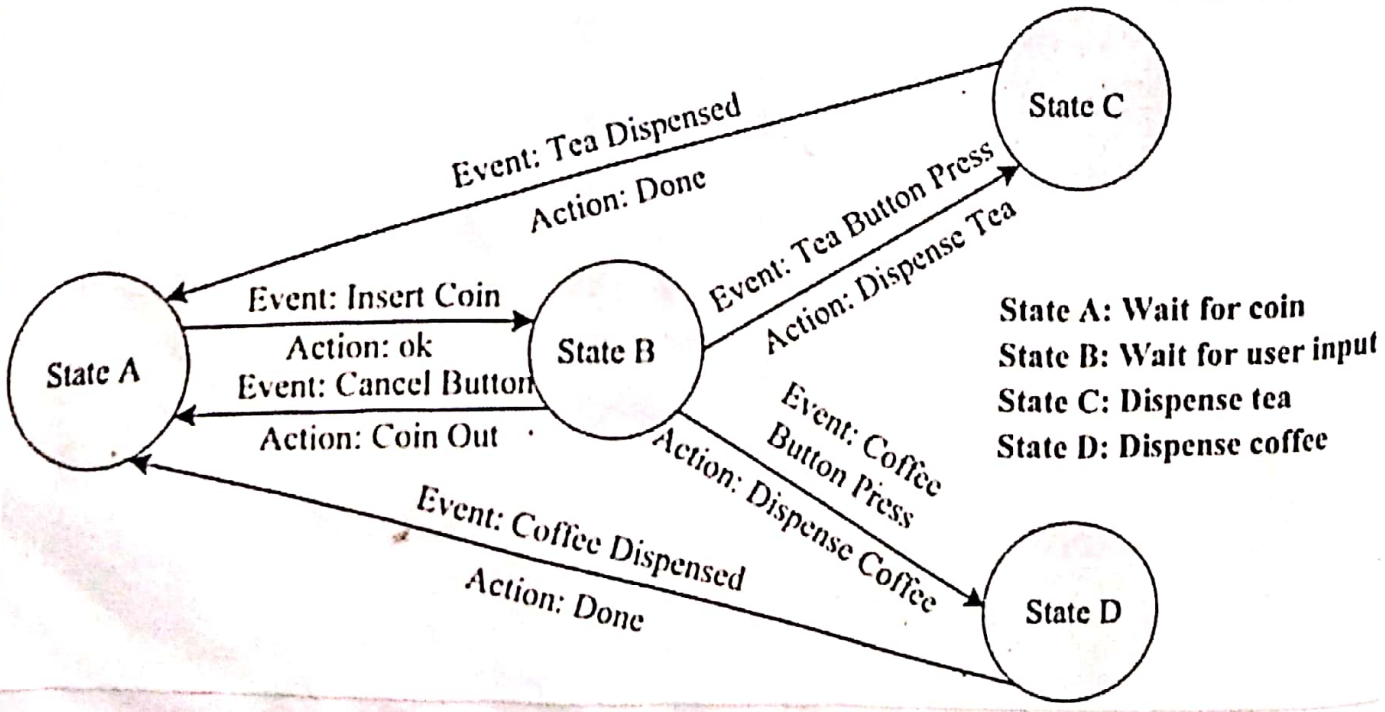
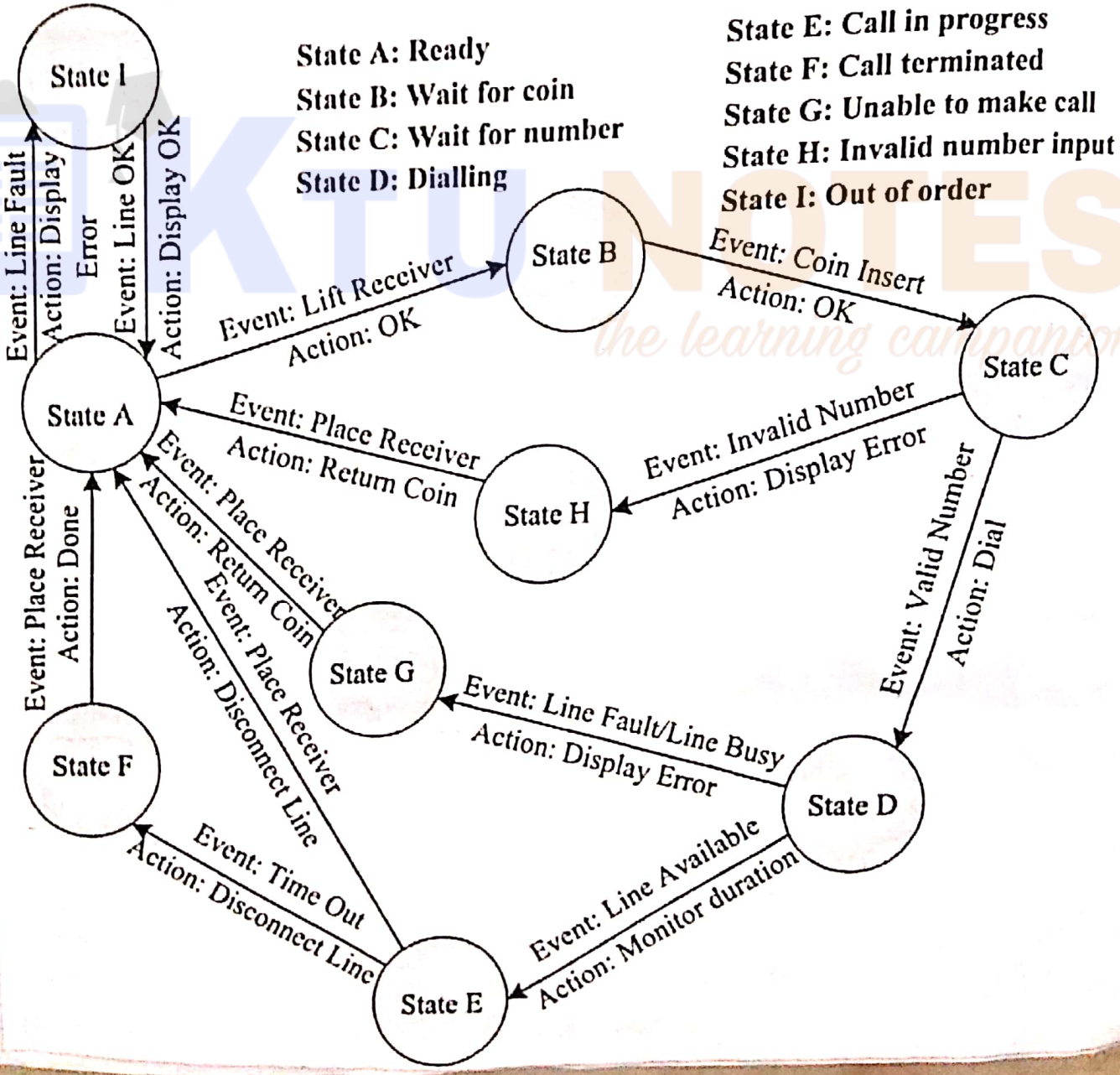


Fig 3 :- FSM Model for Coin Operated Telephone System

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 sec and at the end of 45th sec, prompt for inserting another 1 rupee coin for continuing the call is initiated.
5. If the user doesn't insert 1 rs coin, call is terminated after 60 sec
6. The sym is ready to accept new call request when the receiver is placed back on the hook.
7. The sym goes to the 'Out of order' state when there is a line fault



4) Sequential Program Model

→ Same as Conventional Procedural Programming

→ the functions or processing requirements are executed in sequence.

→ tools used for sequential peg modelling < FSM, Flowcharts

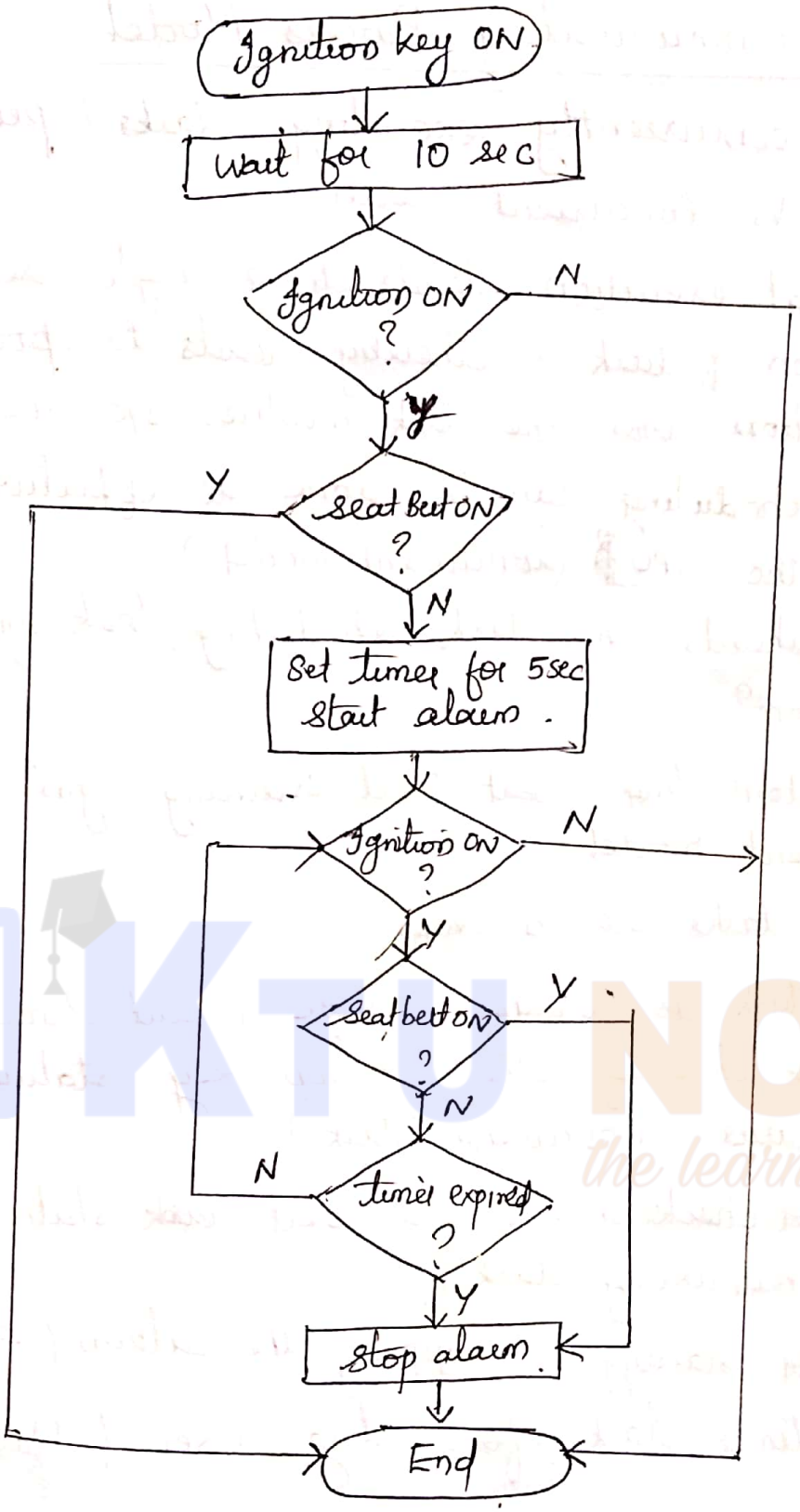
FSM represents the states, events, transitions & actions
Flowchart models the execution flow.

→ eg:- Seat Belt warning sym → The execution of functions in a sequential peg model is given below.

```
# 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) &&
                  time_expire () == NO));
          stop_alarm ();
    } } }
```

→ Fig. below shows the Flowchart approach ~~for~~ of Sequential Program model for seat belt warning system.



5) Concurrent / Communicating Process Model

→ It models concurrently executing tasks / processes

→ Sequential Vs Concurrent model

- Sequential execution leads to a single sequential execution of task & thereby leads to poor processor utilisation. when the task involves I/O waiting, sleeping etc
- Task scheduling can be done to effectively use the CPU (concurrent model)
- But overheads in task scheduling, task synchronization and comm^n.

→ eg :- Implementing 'Seat Belt Warning Sys' in concurrent model.

Split the tasks into 5.

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

Here we need to synchronize the execution of these tasks through some mechanism (We can't execute them randomly / sequentially). Some events are associated with each task to manage the scenario.

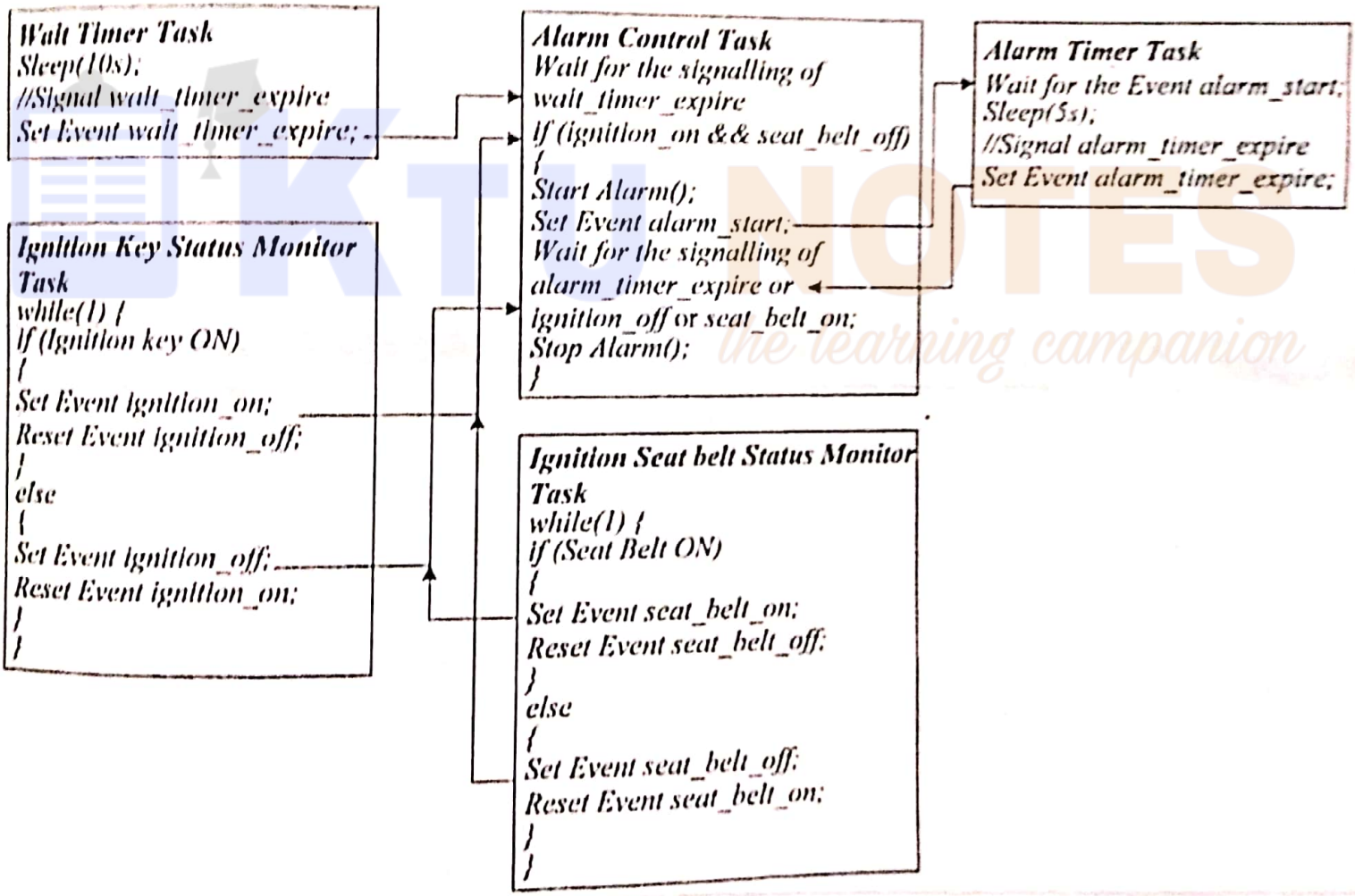
Events are ⇒ wait_timer_expire → task ①
ignition_on, ignition_off → ②
seat_belt_on, seat_belt_off → ③
alarm_start → ④ alarm_timer_start, alarm_timer_expire ⑤

Tasks & events for 'seat belt warning sym'

```
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 sym'

(a)



→ The concurrent model is commonly used for modelling of 'Real time systems'

→ Various techniques like shared memory, Message Passing, Events etc are used for comm & synchronization b/w concurrent processes

6) Object - Oriented Model

→ an object based model, where 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 & state.

→ class :- An abstract description of a set of objects (ie, a blueprint of objects)

A class represents the state of an object through member variables and object behaviour through member functions.

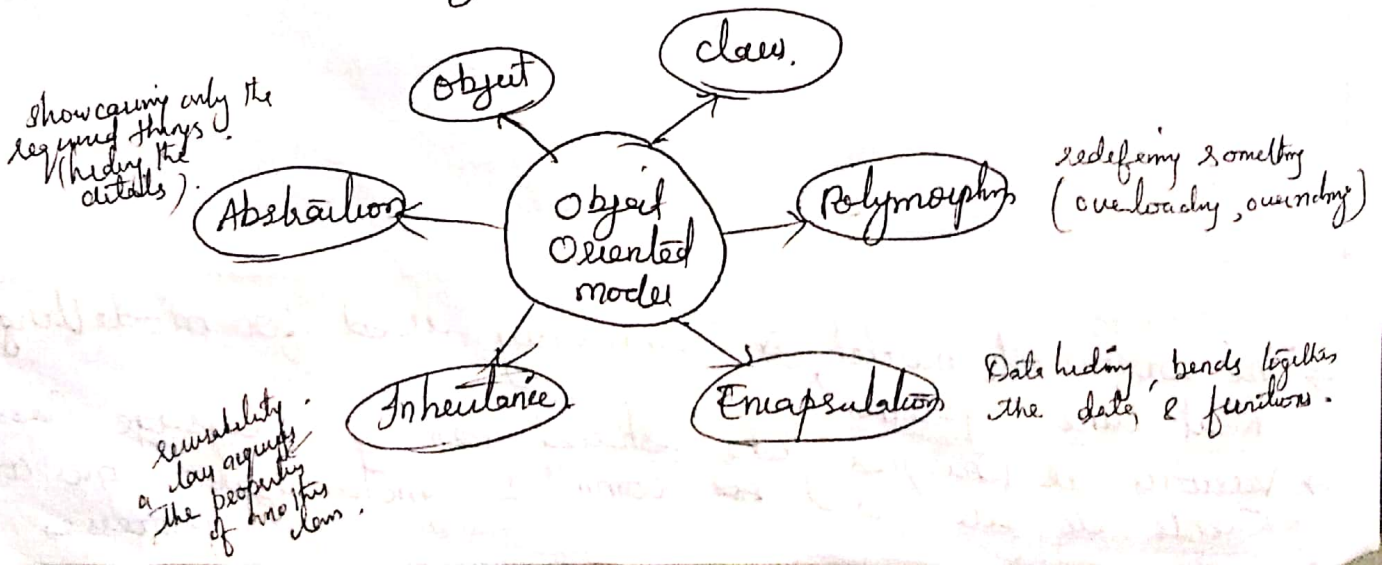
This can be private, public or protected

public : accessible within the class and outside the class

private : accessible only within in the class

protected : protected from external access, but classes derived from a parent class can also access the protected members.

→ This model also brings concepts like Abstraction, Inheritance, Polymorphism etc.



UNIFIED MODELLING LANGUAGE (UML)

→ A visual modelling language for Object Oriented Design (OOD)

→ It helps in all phases of sys design like ⇐ requirement capturing design deployment.

→ UML Building blocks

① **Things** :- A thing is an abstraction of the UML model.

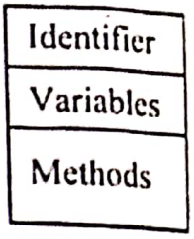
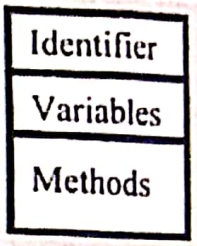
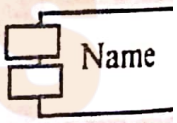
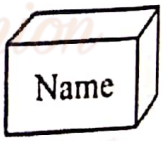
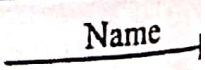
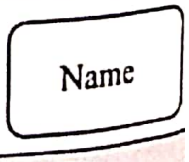
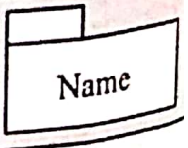
a) **Structural things** :-
- Represents mostly the static parts of a UML model
- also known as Classifiers
- eg:- class, interface, use case, use case realization (collaboration), active class, component & node

b) **Behavioural things**
- Represents the dynamic parts of a UML model
- eg:- Interaction, state machine & activity

c) **Grouping things**
- Organisational parts of a UML model.
- eg:- Package & subsystem

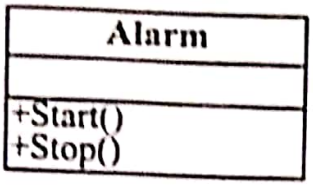
d) **Annotational things**
- Explanatory parts of a UML ~~th~~ model
- eg:- Note.

Fig. shows the things in a UML model.

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 Realisation)	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.	
Behavioural	Interaction	Behaviour comprising a set of objects exchanging messages to accomplish a specific purpose. Represented by arrow with name of operation	
	State Machine	Behaviour specifying the sequence of states in response to events, through which an object traverses during its lifetime.	
Grouping	Package	Organises 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.	

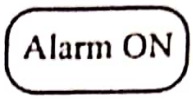
Eg:-

a) The Alarm class

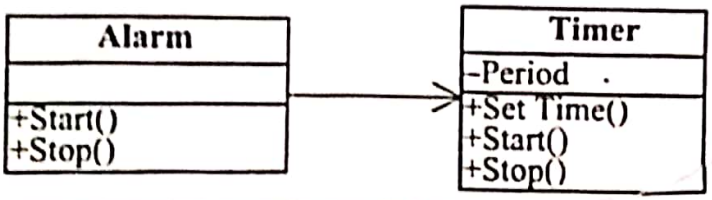


+ - public attribute
- - private "
- protected "
~ - package "

b) State representation for 'Alarm ON' state



c) Alarm - Timer class interaction for the seat-belt warning sym

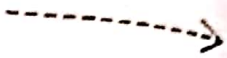
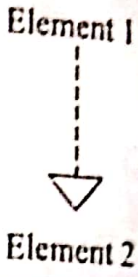


(2) Relationships

They express the type of relationship between UML elements (objects, classes etc).

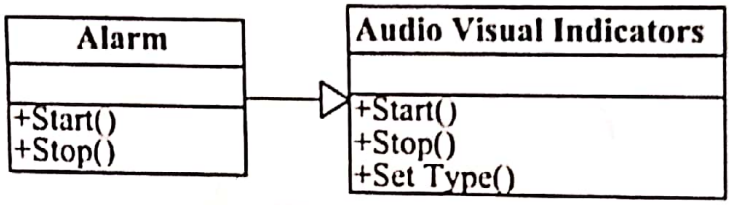
Table shows the various relationships in UML

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.	relationship
Aggregation	It represents is "a part of" relationship. Represented by a line with a hollow diamond at the end.	a part of
Composition (parts are deleted when the whole is deleted)	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.	Child / Parent

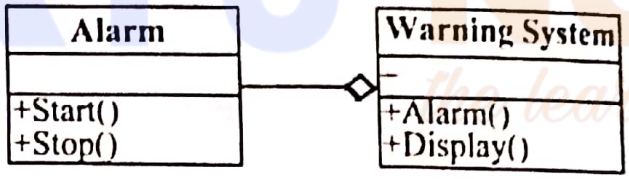
Dependency	Represents a relationship in which one element (object, class) uses or depends on another element (object, class). Represented by a dotted arrow with head pointing to the dependent element.	
Realisation	The relationship between two elements in which one element realises the behaviour specified by the other element.	

y. the supplier & the client
one represent a specification & the other represents the implementation of that

Eg :- a) Alarm is a special type of Audio Visual Indicator (Generalization)



b) Alarm is a part of Warning system. (Aggregation)



(3) UML Diagrams

give a pictorial representation of the static aspects, behavioural aspects and organisation and mgmt of diff^nt modules (classes, packages etc).

2 types < static diagrams
Behavioural "

static diagrams

- represents the structural (static) aspects of the sym
- Eg:- Object diagram, Component diagram, class diag Package diagram, deployment diagram

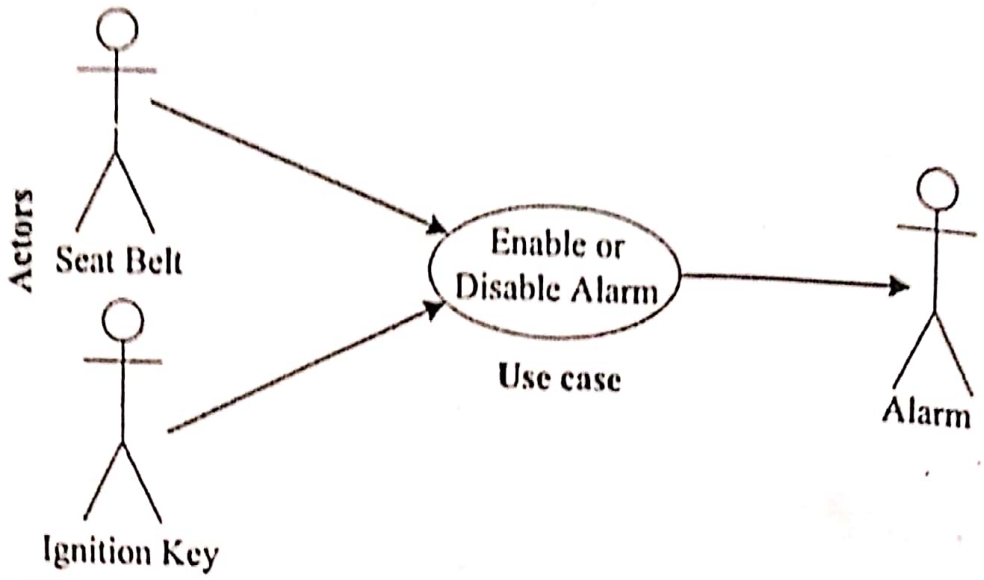
Diagram	Description
Object diagram	Gives a pictorial representation of a set of objects and their relationships. It represents the structural organisation between objects.
Class diagram	Gives a pictorial representation of the different classes in a UML model, their interfaces, the collaborations, interactions 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 components (physical packaging of classes and interfaces), relationships and associations among the components.
Package diagram	It is a representation of the organisation of packages and their elements. Package diagrams are mostly used for organising 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.

Behavioural diagrams

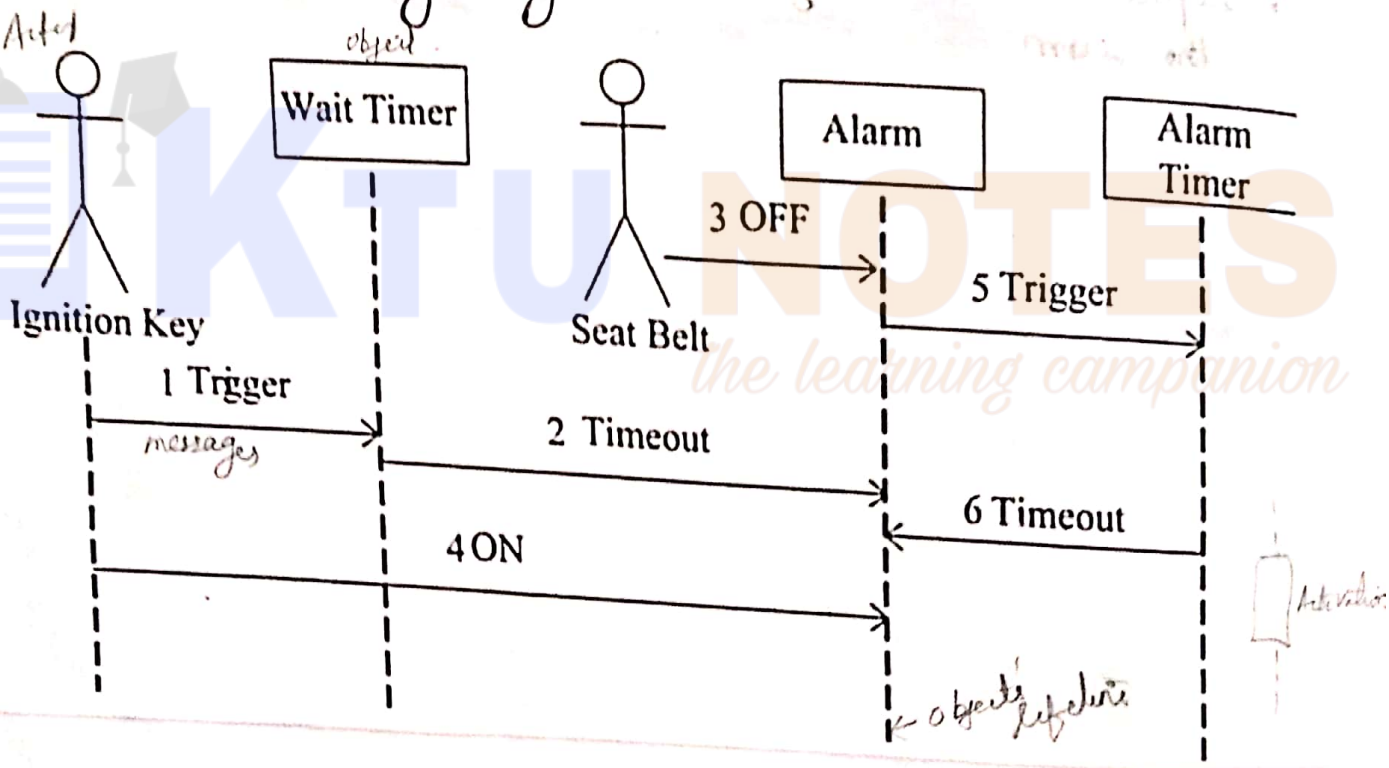
- represent the dynamic (behavioural) aspects of the sym
- eg:- Use case dia., sequence dia, state dia, Communication dia, Activity dia, Timing dia, Interaction overview dia.

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 (Comm 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.

Eg:- Use case diagram for seat belt warning sym



Sequence diagram for one possible sequence for the seat belt warning sym



→ The UML Tools

- used for building UML based models
- available from different vendors
- Can be < Commercial / free / open source
- Some of the popular UML modelling tools are.

Tool	Provider/Comments
Rational Rose Enterprise	IBM Software (http://www-03.ibm.com/software/products/en/enterprise)
Rational System Developer	Eclipse† based tool from IBM Software (http://www-03.ibm.com/software/products/en/ratisoftarch)
Telelogic Rhapsody	UML/SysML-based model-driven development tool for real-time or embedded systems from IBM Software (http://www-03.ibm.com/software/products/en/ratirhapfami)
Borland® Together®	Borland (www.borland.com)
Enterprise Architect	Sparx Systems (http://www.sparxsystems.com)
ARTiSAN Studio	Artisan Software Tools Inc (http://www.artisansoftwaretools.com/)
Microsoft® Visio	Microsoft Corporation. The Microsoft Visio (Part of Microsoft® Office product) supports UML model Diagram generation. www.microsoft.com/office/visio

other tools

Eclipse
UMLet
ArgoUML
Modelio
Dia etc

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 H/w S/w Co-design

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

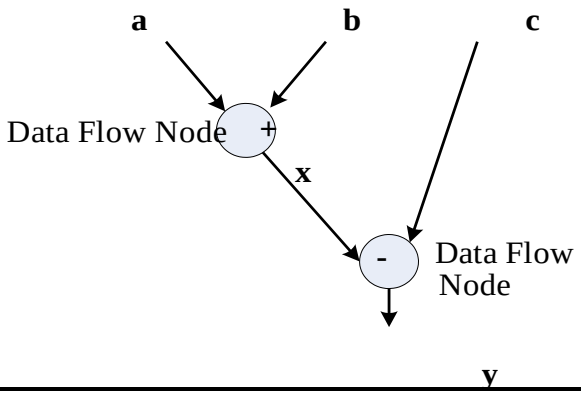
Partitioning of System Requirements into H/w and S/w

- 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

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 (like DSP applications)

E.g. Model the requirement x = a + b; and y = x - c;



Data path: The data flow path from input to output

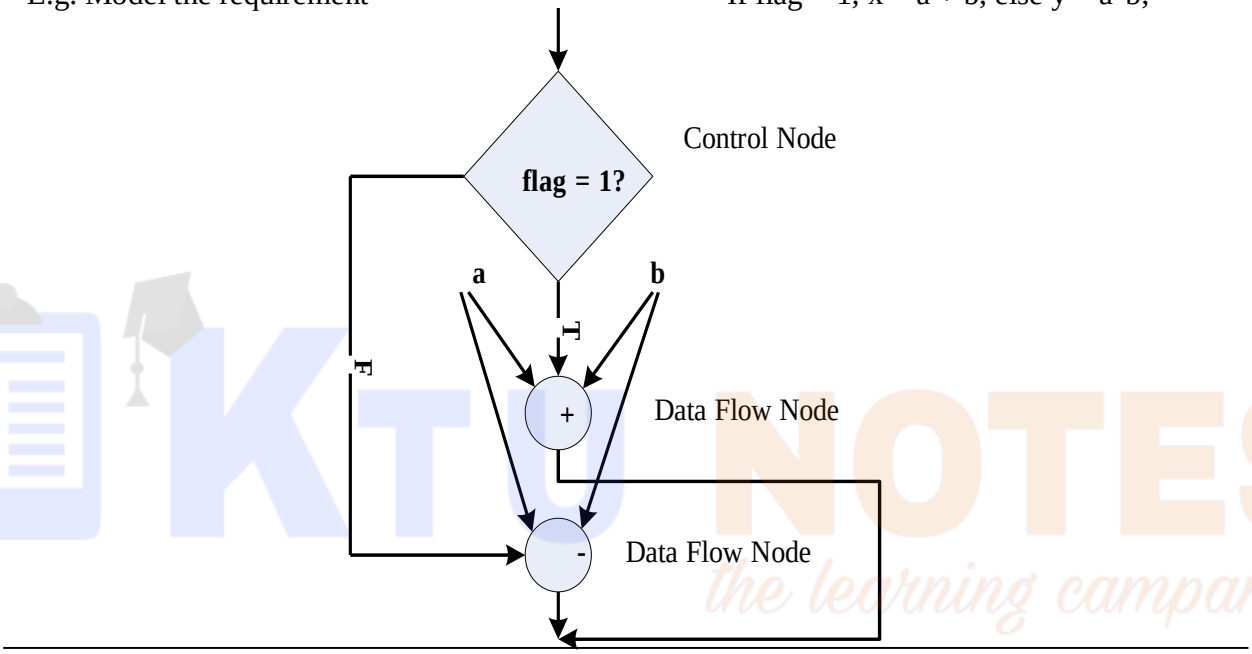
A DFG model is said to be acyclic DFG (ADFG) if it doesn't contain multiple values for the input variable and multiple output values for a given set of input(s). Feedback inputs (Output is feed back to Input), events etc are examples for non-acyclic inputs. A DFG model translates the program as a single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

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 state charts 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'.

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.

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.

E.g. Automatic 'Seat Belt Warning'

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.

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

Concurrent/Communicating Process Model

- Models concurrently executing tasks/processes. The program instructions are iterated and executed conditionally and the data gets transformed through a series of operations
- Certain processing requirements are easier to model 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.
- Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication

Introduction to 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

- 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

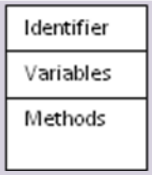
Things

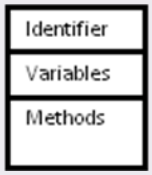
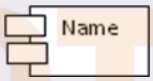
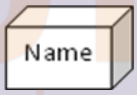
Structural things: Represents mostly the static parts of a UML model. They are also known as 'classifiers'. Class, interface, use case, use case realization (collaboration), active class, component and node are the structural things in UML.

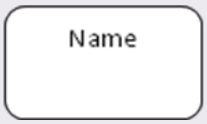
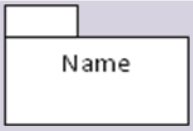
Behavioral things: Represents mostly the dynamic parts of a UML model. Interaction, state machine and activity are the behavioral things in UML.

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.	

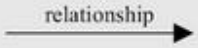
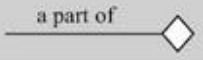
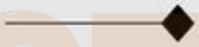
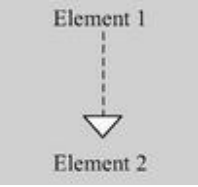
	Active Class	Class presenting a thread of control in the system. It can initiate control activity. Active class is represented in the same way as that of a class but with thick border lines.	
	Interface	A collection of externally visible operations which specify a service of a class. It is represented 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.	

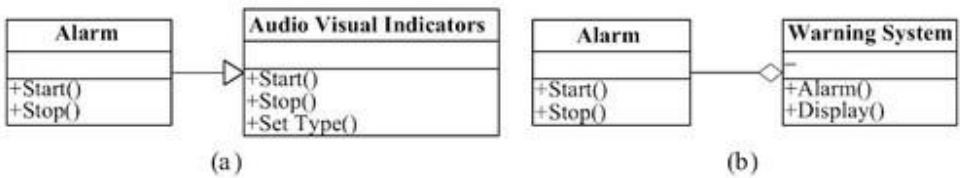
Thing	Element	Description	Representation
Behavioral	Interaction	Behavior comprising a set of objects exchanging messages to accomplish a specific purpose. Represented by arrow with name of operation	Name
	State Machine	Behavior specifying the sequence of states in response to events, through which an 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.	

Relationships in UML

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

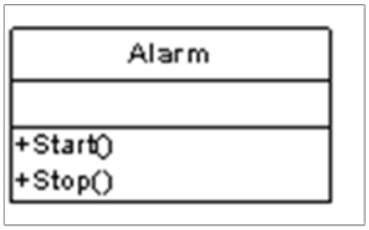
Relationship	Description	Representation
Association	It is a structural relationship describing the link between objects. The association can be one-to-one or one-to-many. Aggregation and Composition are the two variants of Association.	
Aggregation	It represents is "a part of" relationship. Represented 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.	



(a) (b)

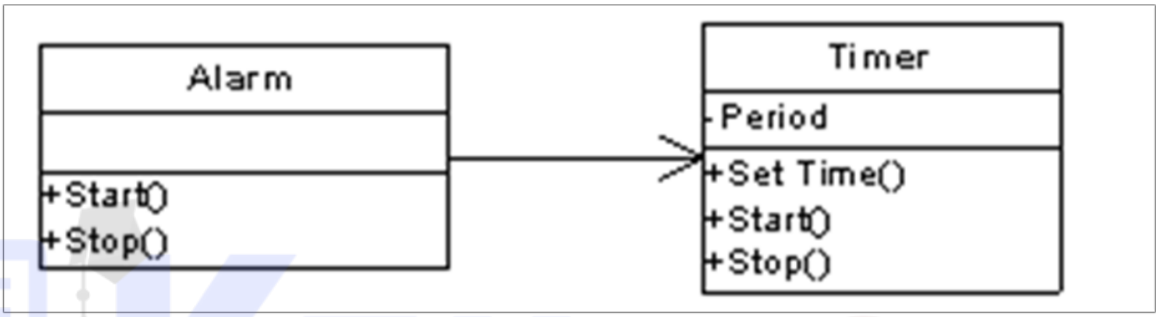
(a) Alarm is a special type of Audio Visual Indicator (Generalisation), (b) Alarm is a part of Warning System (Aggregation)

The Seat Belt Warning System – UML modeling



The Alarm Class

State Representation for 'Alarm ON' state



Alarm – Timer Class interaction for the Seat belt Warning System

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

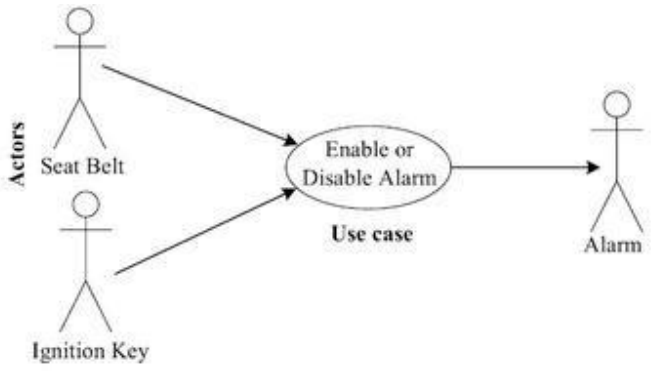
Static Diagrams: Diagram representing the static (structural) aspects of the system. Class Diagram, Object Diagram, Component Diagram, Package Diagram, Composite Structure Diagram and Deployment Diagram falls under this category

Diagram	Description
Object Diagram	Gives a pictorial representation of a set of objects and their relationships. It represents the structural organization between objects.

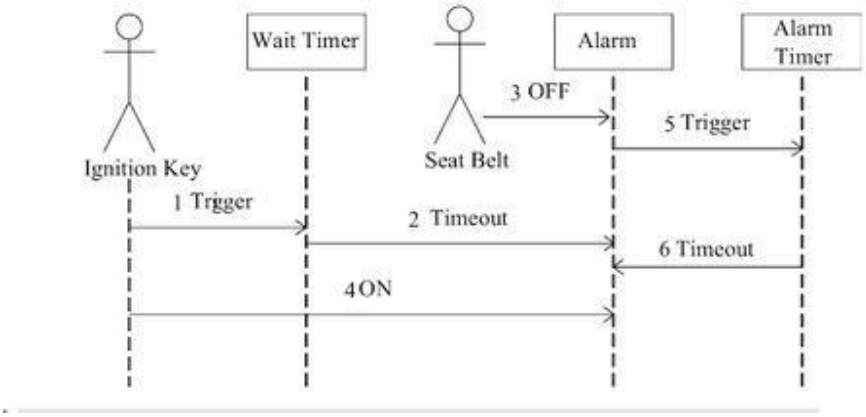
Class Diagram	Gives a pictorial representation of the different classes in a UML model, their interfaces, the collaborations, interactions and relationship between the classes etc. It captures the static design of the system.
Component Diagram	It is a pictorial representation of the implementation view 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 and interaction diagram are diagrams in UML



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.

Hardware Software Trade-offs

- Certain system level processing requirements may be possible to develop in either hardware or software
- The decision on which one to opt is based on the trade-offs and actual system requirement
- Processing speed, performance, memory requirements, effort, re-usability etc. are examples for some of the hardware software trade-offs in the partitioning of a system requirement