EMBEDDED AND REAL TIME SYSTEMS

UNIT 1 ⇒ Complex system and Microprocessor

* Embedded system is one that has computer hadward with software embedded in it as one of the important components.

* An embedded system is a specific computer system design to perform one or a few dedicated functions (tasks) often with real-time computing constrains.

* It is a sbopisticated system that has a Computer hardware with Application software and Real-time Operating system (RTOS) embedded in it as one of its components.

→ It executes (runs) a predefined and dedicated task for an application.

→ The embedded system may be an independent system or a part of a larger system.

Embedded systems are electronic systems that consists of microprocessor or microcontroller, memory, Input/output devices but It is not a computer.

An embedded system has three main components

1. Hardware ; Embedded system has hardware similar to a computer. As its software locates in the ROM or Flash memory, it does not need a secondary hard disk and CD/DVD memory as in a computer.

2. Application software : The application software may concurrently performs a series of tasks or processes or threads.

3. RTOS [Real Time Operating Systems]
   * It supervises the application software running on the hardware.
   * It organizes access to a resource according to the priorities of tasks in the systems.
   * It provides a mechanism to the processor run a process/task as scheduled & context-switch between the various tasks.
   * It sets the rules during the execution of the application software.
   * A small scale embedded system may not need an RTOS.

* KERNEL : The kernal is the central part (component) of most computer OS. It is responsible for task management, Inter task-communication and synchronization. It stays in RAM. & it runs until the system turned off or crashes. User Program make use of Kernal via the Sys-call interface

Application Specific System Processors (ASSP)

➡ An ASSP is dedicated to specific task and provides a faster solution

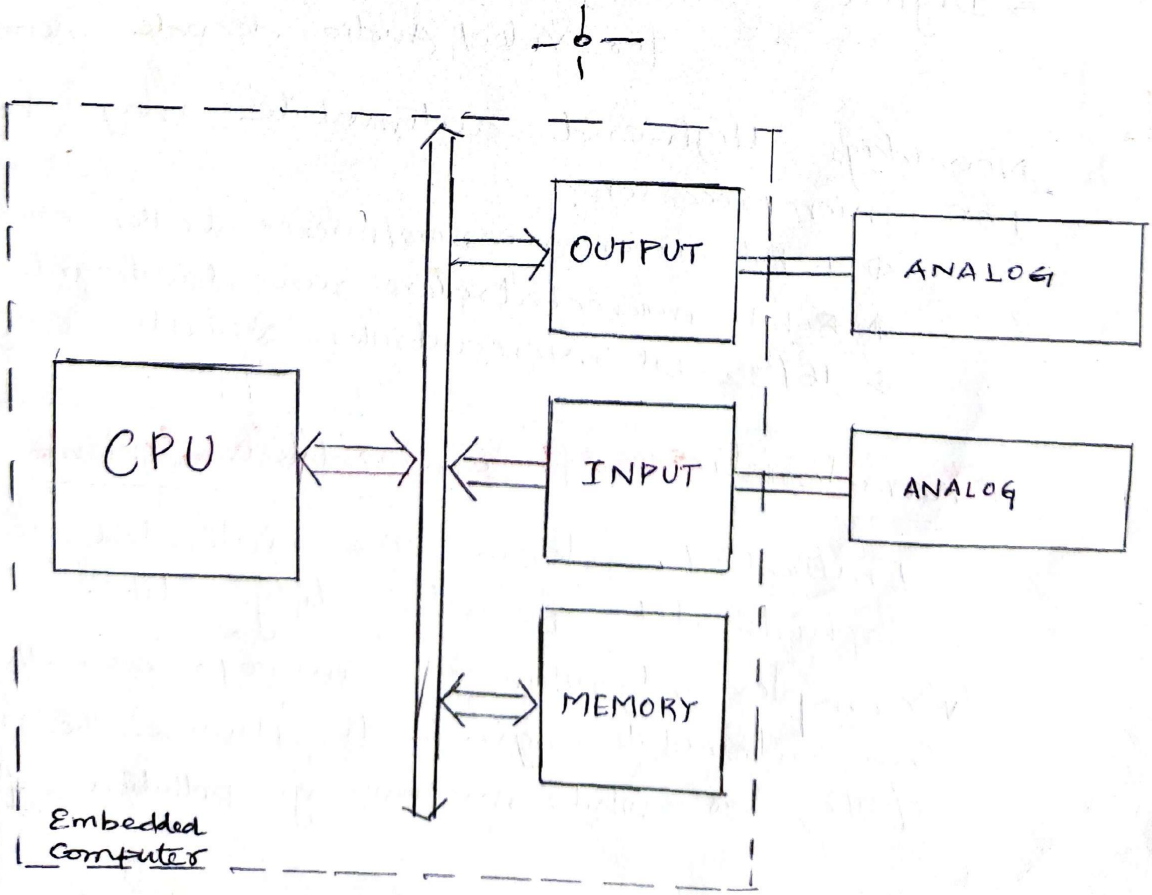
➡ An ASSP is used as an additional processing unit for running the applications in place of using embedded software.

Ex: IIM #100

Multi-processor System using General Purpose Processors

➡ Multiple processors are used when a single processor does not meet the needs of the different tasks that have to be performed concurrently.

➡ The operations of all the processors are synchronized to obtain an optimum performance.



Examples of embedded System

- Cell phone
- Printer
- Automobile : engine, brakes, dash etc.
- Airplane : engine, flight controls, nav/comm.
- Digital TV
- Household appliances.

Early history

→ Late 1940's : MIT Whirlwind computer was designed for Real-time operations.

→ HP-35 calculator used several chips to implement a microprocessor in 1972.

→ Automobiles used microprocess-based engine controllers starting in 1970's

→ Canon EOS 3 has three microprocessors.
- 32-bit RISC CPU runs autofocus & eye control system.

→ Digital TV : Programmable CPUs, Hardwired logic for Video/Audio decode, menus etc.,

Nowadays High-end automobile may have 100 microprocessors:
- 4-bit microprocessors/microcontroller checks seat belt
- 8-bit microcontrollers run dashboard devices
- 16/32-bit microcontroller controls engine.

Characteristics of Embedded Systems.

Embedded systems are intended to provide sophisticated functionality like

* Complex algorithms : the microprocessor that controls an automobile engine to optimize the performance of the car while minimizing pollution & fuel utilization.

User interface : Microprocessors are frequently used to control complex user interfaces like multiple menu & many operations

Ex : Moving map in GPS navigation.

So, to meet these kinds of embedded computing operations needs some important characteristics. like

- → Real time operation
- → Multirate
- → Cost
- → Power & energy
- → Sophisticated functionality
- → Designed to tight deadlines by small teams.

Real Time :

The tasks must be completed by deadlines. There two real time operations

1. Hard real-time : missing operation causes unsafe and can even endanger lives.
2. Soft real-time : missing deadline results in degraded performance

Multirate :

- The embedded computing system have several real-time activities performing on at the same time
- The tasks may have different rates.
- Ex. Multimedia applications, The audio & video have different rates but they must be synchronized. Otherwise it spoils the perception of the entire presentation.

Manufacturing cost: It is important in many cases. The cost is determined by many factors including the types of microprocessor used, size of memory required and the types of I/O devices.

Power and Energy:

Power consumption - affects the cost of the hardware since a larger power supply is required.

Energy consumption - affects the battery life

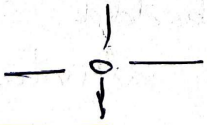
Sophisticated functionality:

→ often have to run complex or multiple algorithms. Ex: cell phone, laser printer.

→ often provide sophisticated user interface.

Design team:

→ often designed by a small team of designer.



Reason to select microprocessor for an Embedded System

→ Microprocessors are a very efficient way to implement digital systems.

→ Easier to design families of product with various features at different prices.

→ It can be extended to provide new features to keep up with rapidly changing markets.

→ It executes programs very efficiently.

→ Microprocessors are very efficient utilizers of logic.

→ Different algorithms can be used by changing the programs.

→ Several function can be implemented on a single processor.

Challenges in Embedded Computing System Design

Some important problems that makes difficult in embedded system design.

→ **Hardware**

An embedded system consist of microprocessor, amount of memory, I/O ports and application demanded peripherals.

To meet performance deadline & manufacturing cost the selection of hardware is important

too little hardware – The system fails to meet its deadlines.

too much hardware – The system becomes too expensive & consumes more power.

→ **Deadline**

- To meet the deadline, have to increase the speed of the hardware but the system will be more expensive.
- On the other way can increase the clock rate of the CPU, Program speed may be limited by the system memory.

Power consumption

→ Reduce the power consumption is a very important task in an embedded system.

→ Excessive power consumption can increase heat dissipation.

→ To reduce the power consumption, slowdown the system speed. But it lead to missed deadlines.

→ Therefore careful design is required to slow down the non-critical parts of the system for power consumption. with meeting the deadline.

Design for Upgradability

→ Same hardware may be used for the upgrade version of a product in the same generation

→ Adding or changing of features may be performed by changing the software.

→ But we could design a system that will provide the required performance for software without changing it.

The Realiability of the system is another important factor, particularly in safty-critical system

The nature of embedded computing system makes their design more difficult. They are

1) Complex testing
2) limited observability & controllability
3) Restricted development environments

Complex testing :

Testing of an embedded system is very difficult because the embedded system is an integration of software and hardware machine.

limited Observability and Controllability :

In Real-time embedded systems, It is not easy to observe the tasks runs in the embedded system and also stop it.

Restricted Development Environment :

→ The development environment (IDE) for embedded systems are very less & limited.

→ The code is compiled on one type of PC & download it onto the embedded system.

→ To debug the code, we must rely on program that runs on the PC and then look inside the embedded system.

Embedded System Design Process :

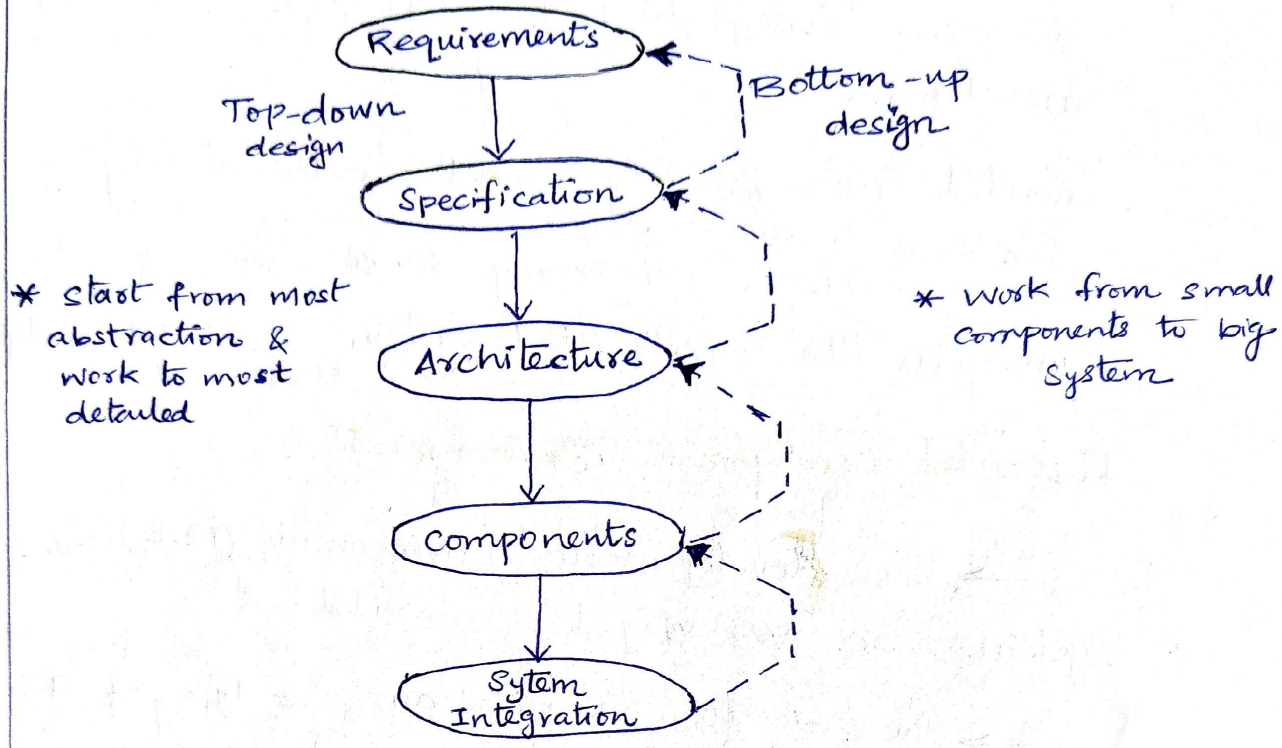
It has two objectives

1) Introduction to the various steps in embedded system design before developed in detailed design.
2) Design Methodology

The design methodology is important for three reasons

1) It allow us to test the performance for optimizing.
2) It allow us to develop computer-aided design tools.
3) Makes the design much easier for members of a design team to communicate.

They are 5 major steps in the embedded system design process. The design can approach either with Top-Down or Bottom-up design.



Major Levels of abstraction in the Design process

➪ The bottom-up design is to help us refine the system.

➪ The major goals of the design are

- manufacturing cost
- Performance (speed & deadlines)
- Power consumption
- cost

➪ **Requirements Analysis**

The designing process is generally proceed in two phases

1) we gather an informal description from the customer known as requirements.
2) we refine the requirements into a specification that contain enough information to begin designing the system architecture.

→ Requirement may be functional or nonfunctional descriptions.

→ Both descriptions should be include for a design.

Functional requirements ; Output as function of input

Non-functional requirements ; Time required to compute O/P,
Size, weight, etc.,
Power consumption,
realiability, etc.
Cost,
Performance etc.

Performance

→ System speed & cost are major considerations.

→ the performance may be a combination of soft performance such as approximate time to operate a user-level function & Hard deadlines (completion of the task with in the time)

Cost:

The purchase price of the system includes manufacturing cost (components & assembly) & nonrecurring engineering cost (Personal & other expences)

Physical Size & Weight

Industrial Instrument ‖ Size – greatly depending on the application & Weight – not no limitation on weight

Handheld device ‖ size & weight should be minimum

Power Consumption:

Power can be specified in the requirement stage in terms of battery life.

A sample requirements form of a system may be as follows

1) Name
2) Purpose
3) Inputs
4) Outputs
5) Functions
6) Performance
7) Manufacturing cost
8) Power
9) Physical Size & weight

Specifications

→ The specification is more precise it serves as the contract between the customer and the architects.

→ It should be clearly written then only system meets customer's requirements.

→ It is essential to design with minimum of designer effort.

→ It will guide the designer towards the purpose of the system & deadline.

→ It should be understandable enough so that someone can verify that it meets system requirements & overall expectations of the customer.

Architecture Design

→ The architecture is a plan for the overall structure of the system.

→ The creation of of the architecture is the first phase of the design

→ The architecture in the form of block diagram that shows major operations and data flow among them.

→ The block diagram is still quite abstract and not specified software running on the CPU & special purpose hardware.

→ The architectural description must be designed to satisfy both functional & non-functional requirements.

→ Modify and refine the block diagram to seperate hardware and software architectures for meeting all specifications.

→ When creating hardware & software architecture, should concentrate on functional elements and non-functional constrains.

Designing with computing platforms.

⇨ Designing Hardware and Software Components

→ The component design effort builds understanding of architecture & specifications.

→ The components includs both hardwares & software modules.

→ Some of the components are readly available in the market example CPU, and memory chip and many other components

System Integration:

➡ In this phase, the components are put together as per the architectural plan.

➡ More bugs are found during integration and it can easily find if a good planning is adopted.

➡ A simple bug in early stage will be a complex one at later.

➡ By building up the system in phases and running properly chosen test, then can easily find the bugs.
i.e. test the functions relatively independently.

➡ System integration is difficult & challenging because it usually uncovers problems.

Formalisms for System Design

➡ There are number of abstraction design tasks at different levels in embedded design.

➡ They are requirements, specification, architecturing the system, designing the code & designing test.

➡ There is a visual language that can be used to capture all these design tasks known as Unified Modeling Language (UML).

➡ UML was designed to be useful to many levels of abstraction in the design process.

⇒ UML is very useful because it supports design process by successive refinement & further adding details to the design rather than redesign.

⇒ UML is an object-oriented modeling language and it shows two concepts of importance

1) It supports the design to be described as a number of interacting objects rather than a large blocks of code.

2) Objects will correspond to the real pieces of software or hardware in the system.

The object oriented specification can be seen in two concepts of complementary ways:

i) Object-oriented specification allows a system to be described in a way that closely models real-world objects and their interactions.

ii) Object-Oriented specification provides a basic set of primitives that can be used to describe systems with particular attributes, irrespective of the relationships of those systems component to real-world objects.

Object Oriented Specification may not be executable when compared with Object Oriented Programming language.

But both languages provides similar basic methods for structuring large systems.

UML is a large & rich, having many graphical elements. It needs more care to use correct drawing to describe something.

⇒ Model Train Controller : A design example.

Structural Description

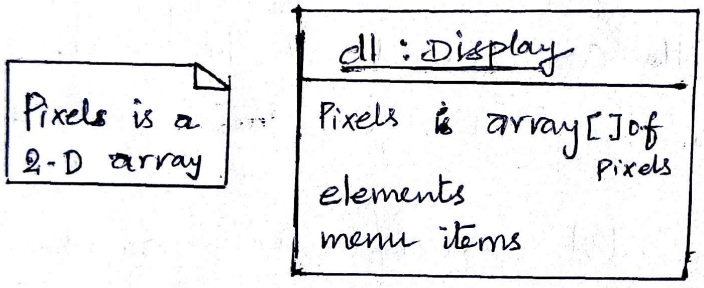
→ The structural description gives basic components of the system.

→ Designer can easily learn to describe those components.

→ The principle component of an object oriented design is the object.

→ An object includes a set of attributes that defines its internal state

→ An object describing a display is shown in UML notation as shown below



An Object in UML notation

→ The object is identified in two ways

1) It has a unique name & it is a member of a class
2) The name is underlined to show that this is a description of an object and not of a class.

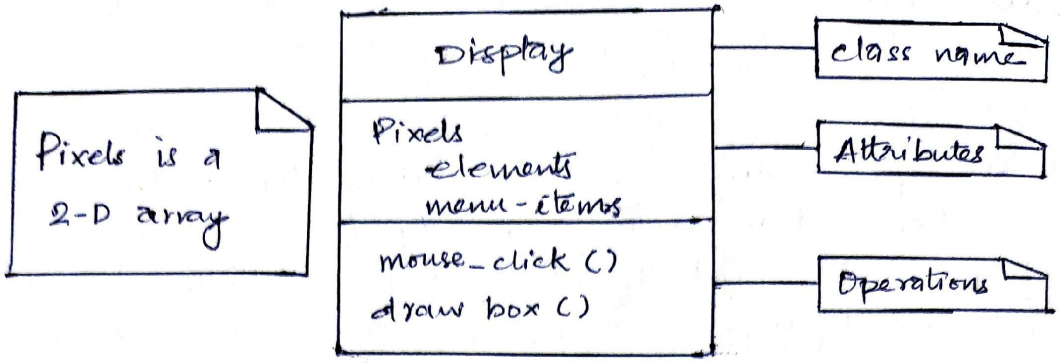
→ A class is a form of type definition all objects derived from the same class have the same characteristics, but their attributes may have different values.

→ A class defines the attributes that an object may have.

→ It also defines the operations that determine how the object interacts with the rest of the world.

The UML description of the display class is shown as follows.

A class in UML notation



→ A class defines both the interfaces for a particular type of object and that objects implementation.

→ When we use an object, we do not directly manipulate its attributes, we can only read or modify the objects state through the operations that define the interface to the object.

→ The proper interface must provide ways to access the objects state as well as ways to update the state.

→ There are several types of relationships that can exist between objects & classes.

1) **Association** It occurs between objects that communicate with each other but have no ownership relationship between them.

2) **Aggregation** It describes a complex object made of smaller objects.

3) **Composition** It is a type of aggregation in which the owner does not allow access to the component objects.

4) **Generalization** It allows us to define one class in terms of another.

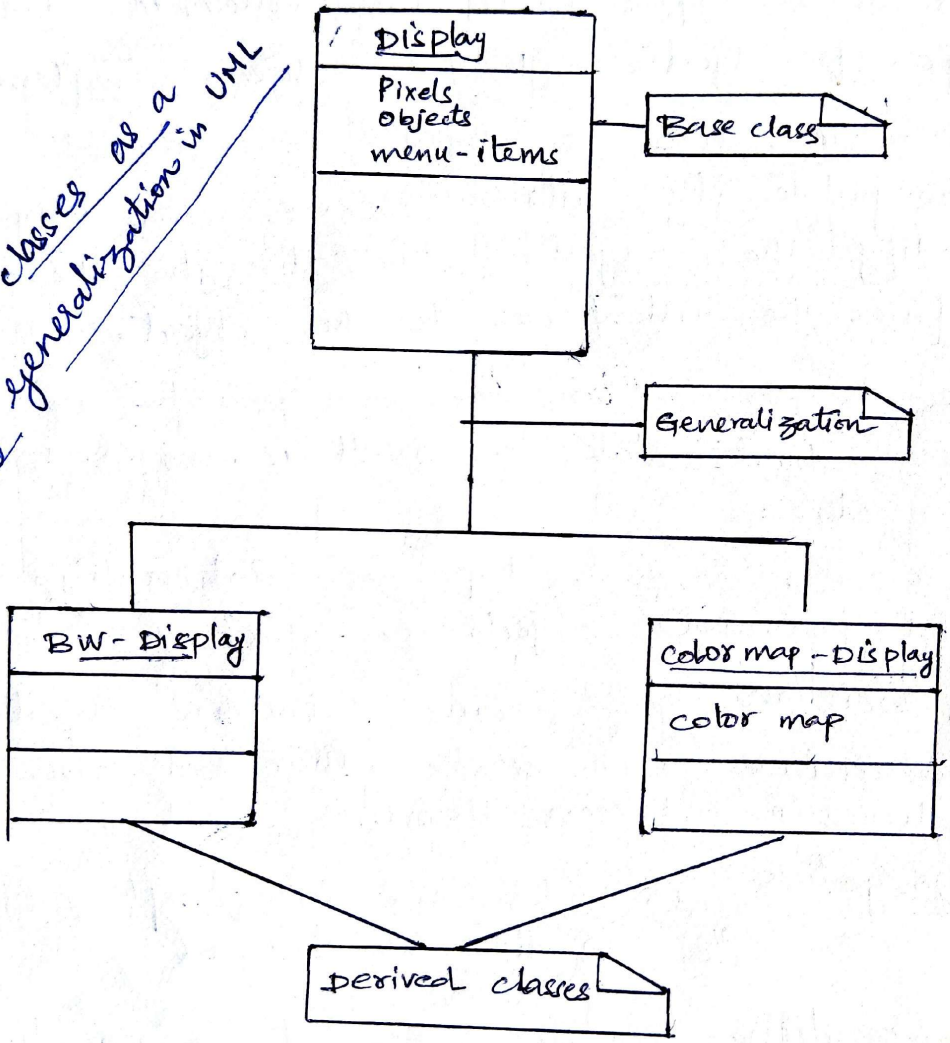
The elements of a UML class or object do not necessarily directly correspond to statements in a programming language, if the UML is intended to describe something more abstract than program, there may be a significant gap between the UML & a programming.

→ An attribute is some value that reflects the current state of the object.

→ The behaviours of the object must be in a higher level specification and contains basic things that can be done with an object.

→ like most object-oriented languages UML also allows us to define one class in terms of another.

Derived classes as a form of generalization in UML



→ A derived class inherits all the attributes & operations from its base class.

→ A derived class is defined to include all the attributes of its base class.

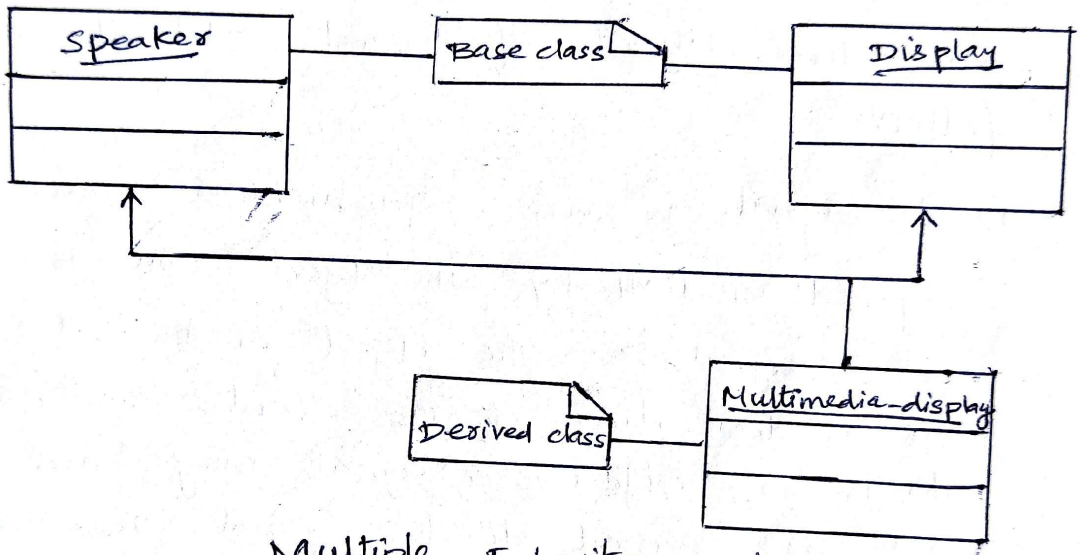
→ From our example display is the base class & BW display and color map display are the two derived classes.

Inheritance

⇨ UML considers inheritance to be one form of generalization

⇨ A generalization relationship is shown in UML diagram as an arrow with an open arrow head.

⇨ Both BW-display & color-map display are specific versions of display UML also allows us to define multiple inheritance.



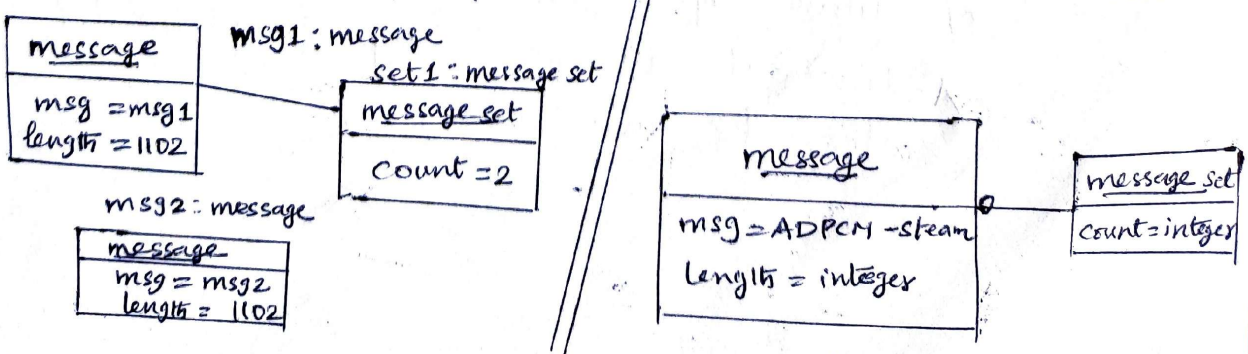
Multiple Inheritance in UML

⇨ Multiple inheritance means which a class is derived from more than one base class.

⇨ A like describes a relationship between objects.

⇨ Association is to like as class is to object.

⇨ Link used to make objects to stand and association capture type information about these links.

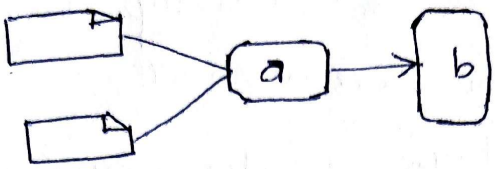


Link between Objects

Association between classes

Behavioral Description

⇒ One way to specify the behaviour of an operation is a state machine.



This state machine shows change from one state to another are triggered by the occurrence of events.

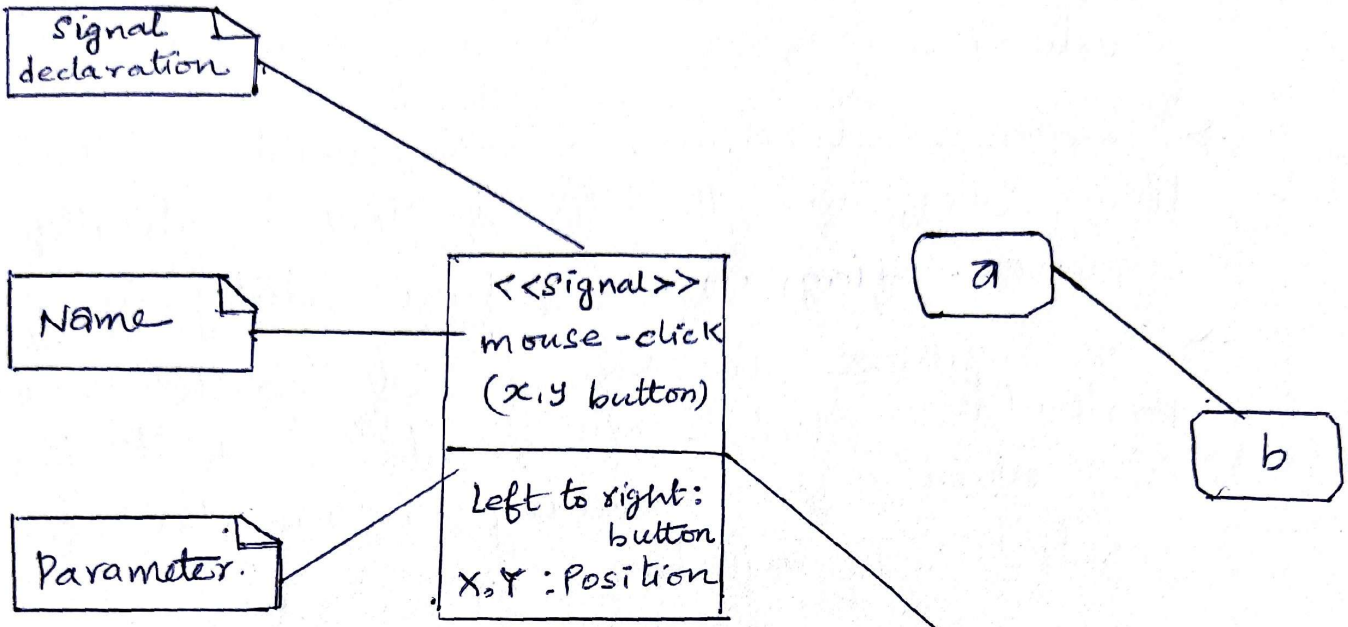
An event is some type of action & there are three types of events defined by UML as follows

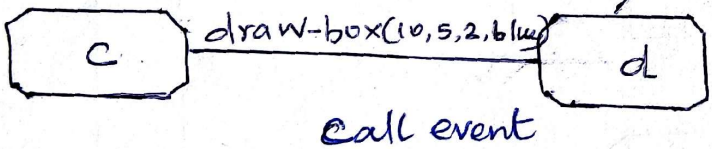
1) A **signal** is an asynchronous occurance. It is defined in UML by an object that is labeled as a <<signal>>. The object in the diagram serves as a decleration of the event's existence. Because it is an object, a signal may have parameters that are passed to the signal's receiver.

2) A **call event** follows the model of a procedure call in programming language.

3) A **time-out event** causes the machine to leave a state after a certain amount of time. The label tm(time-value) on the edge gives the amount of time after which the transition occurs. A timeout is generally implemented with an external timer.

Behavioral Description





Call event

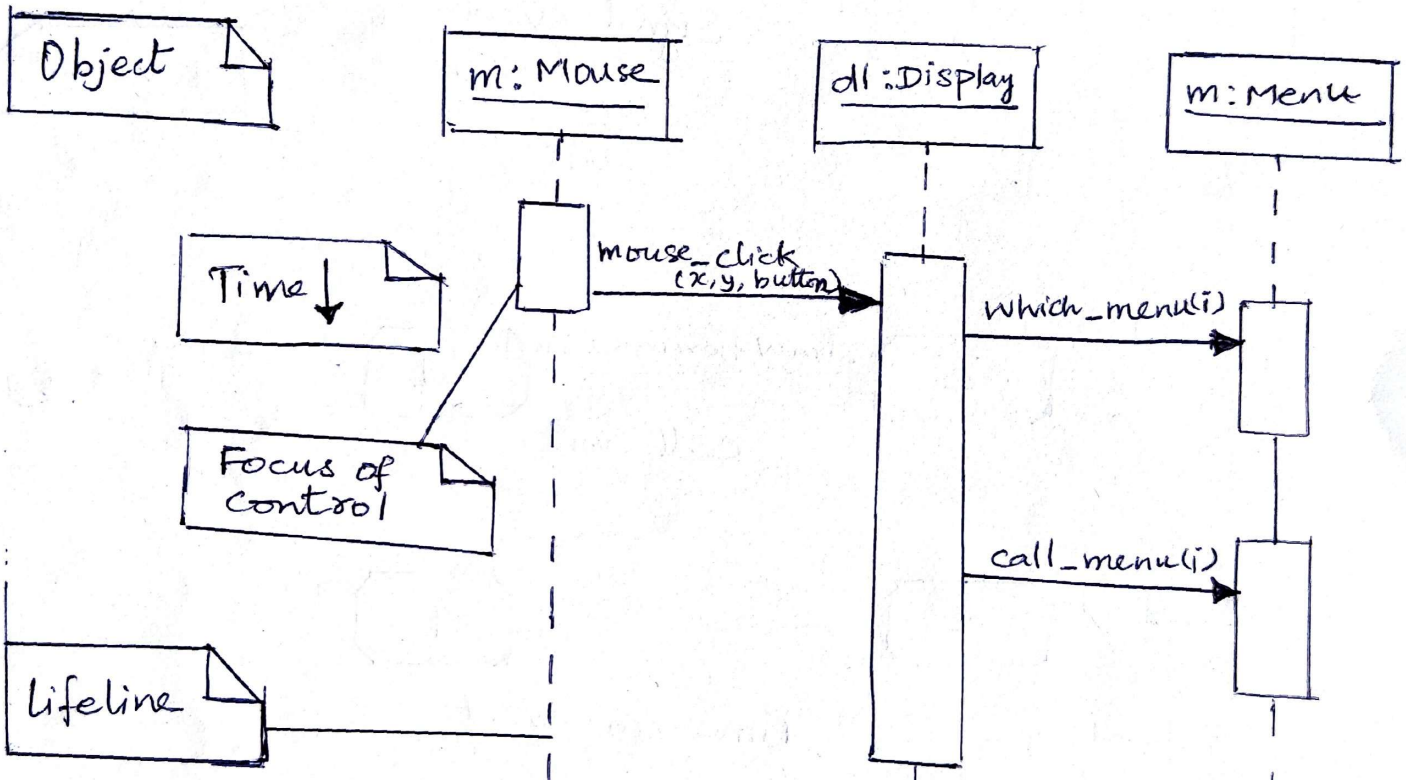
Time-out event

Sequence Diagram

→ It is useful to show the sequence of operations over time, particularly when several objects are involved.

→ A sequence diagram is somewhat similar to hardware timing diagram, the time flows vertically in a sequence diagram & horizontally in a timing diagram.

→ The sequence diagram is designed to show a particular choice of events and it is not convenient for showing a number of mutually exclusive possibilities.



A sequence diagram in UML

If messages can interfere with each other in the network, analyzing communication delay becomes difficult

The message delay $\boxed{t_y = t_d + t_x}$

Where t_d = the network availability time delay incurred (got) waiting for the network to become available.
t_d depends on the type arbitration used.

⇒ If the network uses fixed priority arbitration, the network availability delay is infinite for all but the highest-priority device.

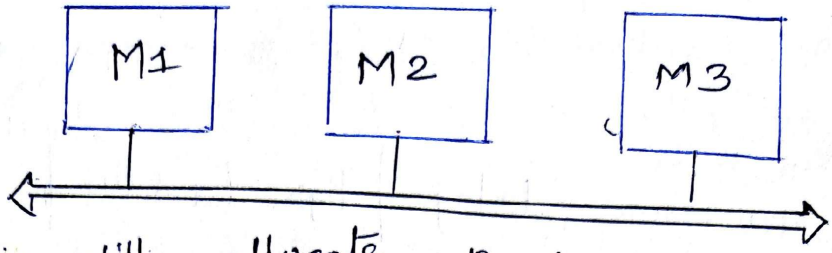
⇒ Since the highest-priority device always gets the network first, unless there is an application-specific limit on how long it will transmit before terminate the network, it keeps blocking the other devices indefinitely.

⇒ If the network uses fair arbitration. the network availability delay is finite.

⇒ In the case of round robine arbitration, if there are N devices, then the worst-case network availability delay is $N(t_x + t_{arb})$, where t_{arb} is the delay got for arbitration & it is small compared to transmission time.

⇒ Even when round-robine arbitration is used to bound the network availability delay, the waiting time can be very long

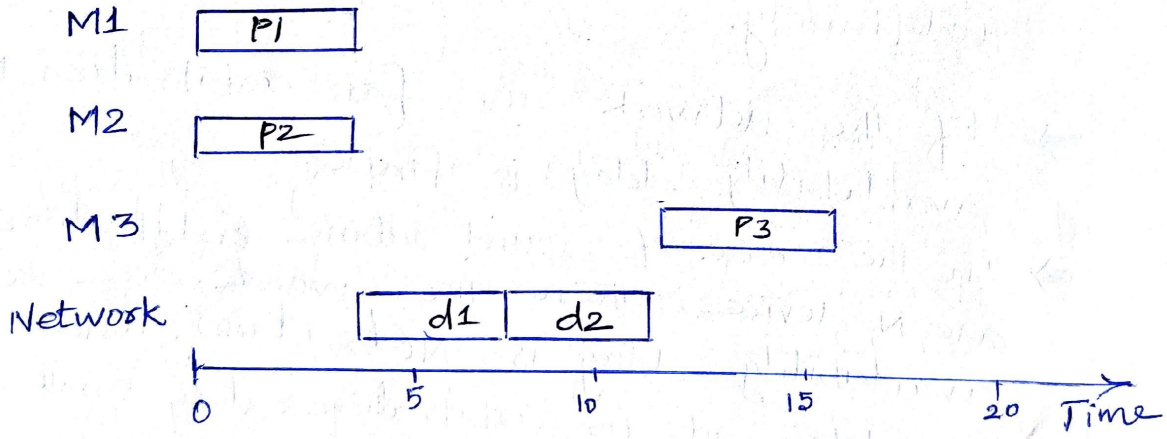
⇒ It is worthwhile to examine the application to determine whether the message structure can be readjusted to reduce t_d



⇒ We will allocate P1 to M1, P2 to M2 & P3 to M3. P1 & P2 runs for three time units while P3 runs for 4 time units.

⇒ A comple transmission of either d1 or d2 takes 4 time units. The task graph shows that P3 cannot start until it receives its data from both P1 & P2 over the bus network.

⇒ The simplest implementation transmits all the required data in one large message, which is 4 packets long in this case. Appearing below is a schedule based on that message structure



⇒ P3 does not start until time 11, when the transmission of the second message has been completed. The total schedule length is 15

⇒ Let's redesign P3 so that it doesn't require all of both messages to begin. We modify the program so that it reads one packet of data each from d1 & d2 and start computing on that. Now it picks up the packets & keeps computing.

⇒ This organization allows us to take advantages of concurrency between M_3 (PE) & the network shown by the schedule below.

⇒ Reorganizing & concurrent execution reduces the scheduling length from 15 to 12.

⇒ When a low priority message is on the network, the network is effectively allocated to that low-priority message, allowing it to block higher-priority messages, but it can slow down critical communication.

⇒ The only solution is to analyze network behavious to determine whether priority inversion cause some messages to be delayed for too long.

⇒ A round-robine arbitration networks put all communications at same priority. This does not eliminate the priority inversion problem because processess still have priorities.

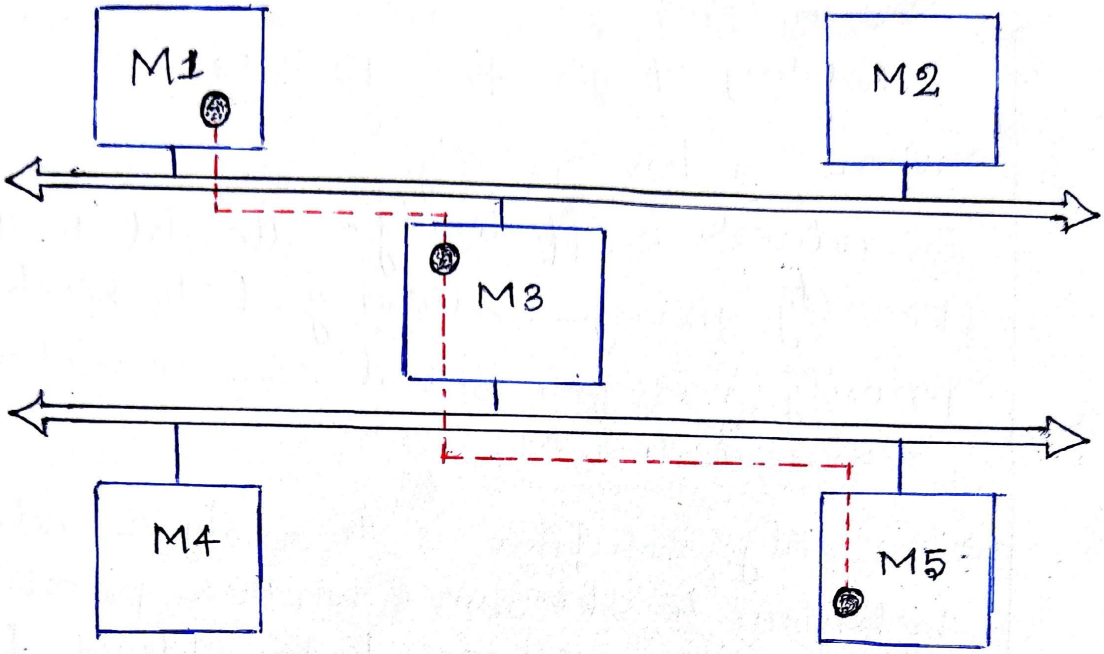
Single-hop network: A mess is received at its intended destination directly from the source, without going through any other networks node.

Multihop networks in which messages are routed through network nodes to go to their destinations.

⇒ Here the hardward platform has two seperate networks, but there is no direct path from M_1 to M_5. The message is therefore routed through M_3, which read it from one network & send it on the other one.

⇒ Analysing delay is very difficult, because the time that the message is held at M_3 depends on both computational load of M_3 & the other message that it must handle.

⇒ If there is more than one network, we must allocate communication to the networks so that it works based on priority.



A Multihop communication

⇒ Scheduling and allocation of computations and communications are clearly interrelated

System Performance Analysis

⇒ Analyzing the performance of distributed embedded system is very difficult.

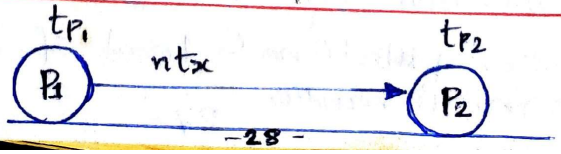
⇒ The figure below in this page shows a very simple task graph with two processes and one n-packet data communication.

⇒ The worst-case execution times of the processes are t_{p1} and t_{p2} and the communication takes nt_x time units.

⇒ There is no interference from outside element, two processes occurs seperately at a single rate.

∴ The worst-case execution time = $t_{p1} + nt_x + t_{p2}$

Delay through a simple task graph

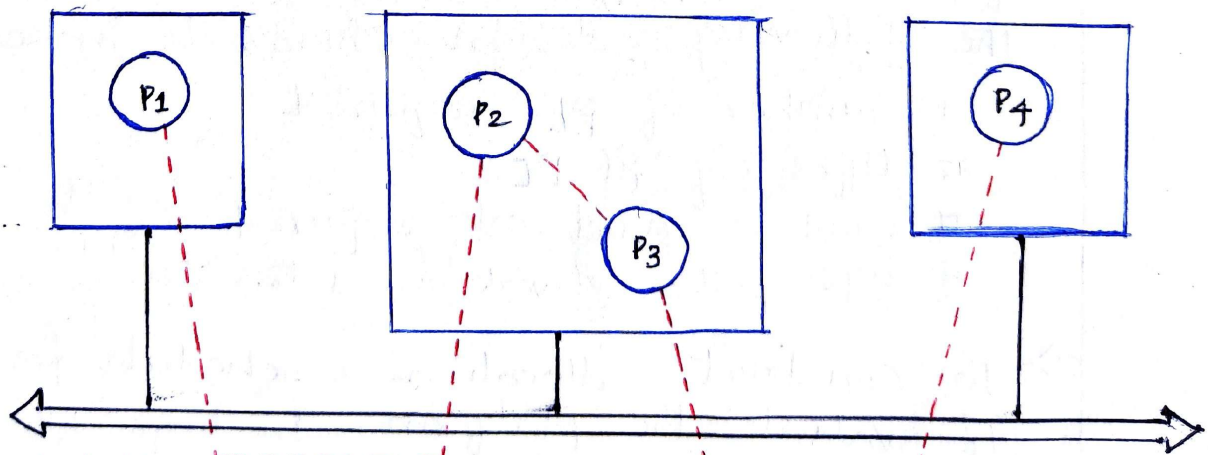


Interference between tasks

⇒ Performance analysis becomes much harder in computations and communications to interfere with each other.

⇒ we have supperimpose the task graph on the target architecture, P_2 & P_3 run on the same PE, which helps enable the following chain of events that can affect the whole system.

- The data dependency from P_1 to P_2 translates any uncertainty in the execution time of P_1 into uncertainty about the start time of P_2.
- The coallocation of P_2 & P_3 processing elements M3 means the variations in the ready time of P_2 can affect the completion time of P_3. ofcourse, variations in the execution time of P_2 also affect P_3.
- The data dependency from P_3 to P_4 translates variation in the completion time of P_3 to the start time of P_4.



A distributed system with multirate concurrency

⇒ Even through P_2 & P_3 are seperate tasks but allocated in same PE, therefor messages from the two tasks can interfere with each other on the bus, causing more variation in completion time.

⇨ complex distributed embedded system requires CAD tool to accurately analyze performance.

⇨ By using hand-desiging anylize performance, it is necessary take care about to meet hand real-time deadlines, non critical tasks can be turned off temporarily

⇨ When there are several critical tasks must occur simultaneously, hand design requires allocating them to share nothing - no PEs nor communication links.

⇨ While this is a conventional design strategy, it makes hand analysis feasible.

Designing with computing platforms:

Hardware Platform Design, Allocation and Scheduling.

⇨ Desiging the hardward platform is necessarily closely related to our choices in scheduling & allocating processes.

⇨ Creating that schedule requires an allocation of processes to PEs, which in turn requires knowing the available hardware.

⇨ When designing the hardware platform, we have the following design choices to make:

- number of PEs required
- types of all PEs
- numbers of networks required
- types of the networks & data rates.

⇨ To construct allocations & schedules for the processes to evaluate the platform. In turn, allocation & scheduling are driven by system performance analysis.

⇨ A lower bound on the computational needs of the system can be obtained by summing up the worse-case execution times of the processes of follows.

$$t_c = \sum_{\text{Process } i} \frac{T_L}{T_i} t_{pi}$$

Where t_{pi} = the execution time of process P_i &
T_L = the least-common multiple of the periods T_i

This formula computes the total execution time over the schedule unrolled to the least-common multiple of the periods.

Similarly, we can compute the communication volume over the least-common multiple of the periods

$$n_c = \sum_{\text{Process } i} l_i$$

this formula computes the total number of bytes transmitted in the unrolled schedule by counting the output bytes of all process in the system

⇒ Depending on the type of system we are designing, the following two strategies may be useful to help us quickly come up with an efficient system:

- for I/O-intensive systems we will start with the I/O devices and their associated processing
- For computation-intensive systems will start with the processes.

I/O-intensive system design.

⇒ I/O are important devices, to support the I/O devices

* Inventory the required I/O devices
* I/O devices that do not require local processing may be attached to the network with the simplest available interface.

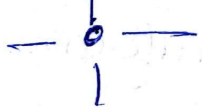
* Determine which devices can share a processing element or networ interface.
* Analyze communication time to satisfy communication deadlines.
* Allocate the minimum required PE to go with each I/O device.
* Design the rest of the system using procedure for computation intensive system.

computation - intensive system design

for this we should consider the processes & their deadlines and communication as follows:

* start with the tasks with the shortest deadlines. If a high-priority-task shares a PE with a low-priority task not only will a more expensive PE be required, but scheduling overhead will be paid for at the nonlinear rate.
* Analyze communication to determine whether critical communications may interfere with each other.
* Allocate low-priority tasks to share PEs where possible.

⇒ The designed systems should meets our performance goals, power consumption & other requirements.



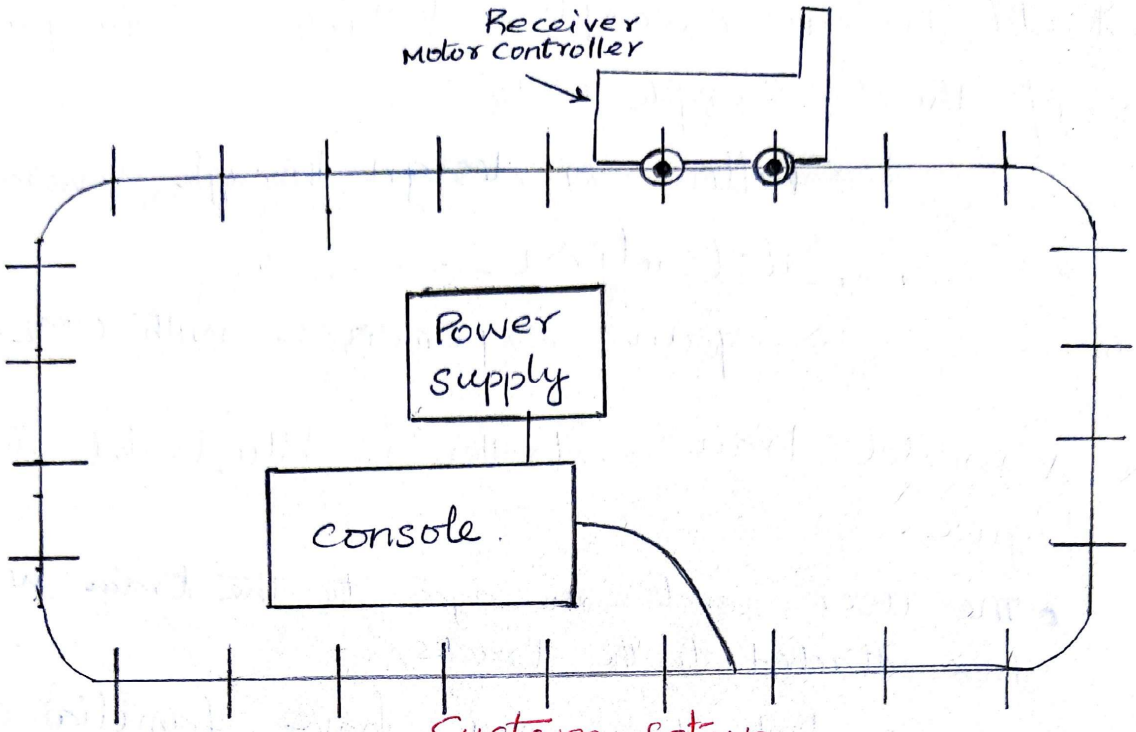
MODEL TRAIN CONTROLLER

* It is an example design and the purpose of this example is to
  - ⇒ Follow a design through several levels of abstraction.
  - ⇒ Gain experience with UML.

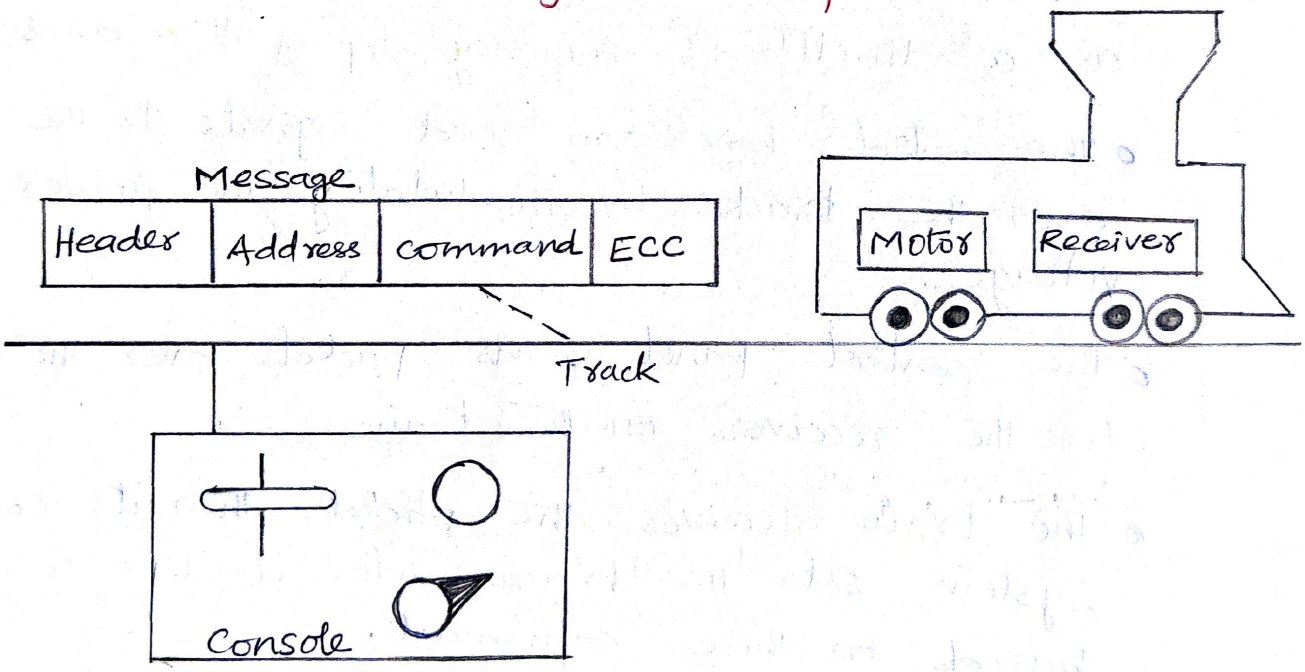
* A model train controller is illustrated in the following figure.
  - The user sends messages to the train with a control box attached to the tracks
  - The control box may have familiar controls such as a throttle, emergency stop button and so on.
  - The control box can send signals to the train over the tracks by modulating the power supply voltage.
  - The control panel sends packets over the tracks to the receiver on the train.
  - The train receives the packets then its control system sets the train motors speed & direction based on those commands.
  - The packets includes Header, address, command & ECC (Error correction code)
  - This is a one-way communication system – the model train cannot send commands back to the user.

* This model start with the **requirements** for the train control system and then **specifications**.

A model train control system



System setup



Signaling the train

Requirements

* console can controll up to 8 trains on a single track
* Throttle has 63 different levels of speed in each direction.

* Inertia control adjusts responsiveness with at least 8 levels
* Emergency stop button
* Error detection scheme on messages.

Requirement Form

Name	Model train controller
purpose	Control speed of up to 8 model trains.
Inputs	Throttle, inertia setting, emergency stop train number
Outputs	Train control signals.
Functions	Set engine speed based upon inertia settings; respond to emergency stop
Performance	Can update train speed at least 10 times per second
Manufacturing cost	$50
Power	10 watts
Physical size & weight	approximate size of standard keyboard & weight < 2 pounds.

DIGITAL COMMAND CONTROL [DCC]

⇨ DCC was created by the National Model Railroad Association to support practical digitally-controlled model trains.

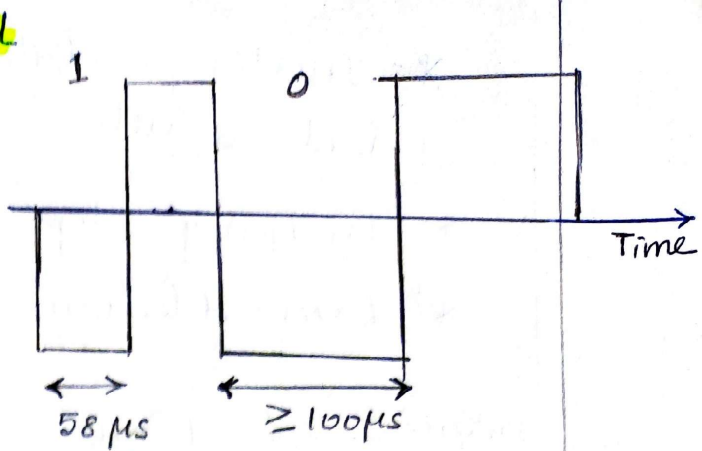
⇨ Hobbyists could mix & match components from several vendors.

⇨ DCC Documents

- Standard S-9.1, the DCC Electrical Standard, defines how bits are encoded on the rails for transmission.
- Standard S-9.2, the DCC communication standards, defines the packets that carry information.

DCC electrical Standard

* Voltage moves around the power supply voltage; adds no DC components
* 1 → 58 µs
  0 → ≥100 µs



Bit encoding in DCC

DCC Communication Standard

The basic packet format ; PSA (&D) +E

- P : Preamble which is a sequence of at least 10 one bits 1111111111
- S : Packet start bit =0
- A : Address data byte; it is an eight bits long
- & : Data byte start bit =0
- D : Data byte, which includes eight bits
  A data byte contains address, instruction, data or ECC.
- E ; Packet end bit. = 1

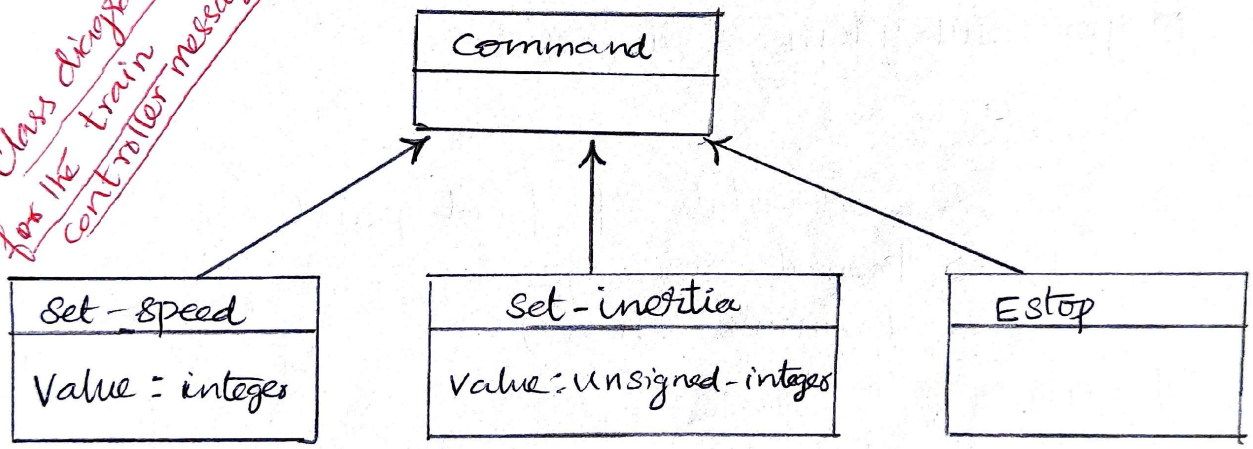
* A packet includes one or more data bytes
* A baseline packet is the minimum packet that must be accepted by all DCC implementation.
* A baseline packet has three data bytes
  (1) Address data bytes that gives the intended receiver of the packet
  (2) Instruction data byte provides a basic instruction
  (3) Error correction data byte is used to detect & correct transmission errors.
* The instruction data byte carrier several informations.
  Bits 0-3 ⇒ 4 bit speed value
  Bit 4 ⇒ additional speed bit & it is a LSB
  Bit 5 ⇒ direction 1 - forward & 0 - reverse
  Bit 7-8 ⇒ Set at 01 for provide speed & direction.
* ECC ⇒ Address bits Instruction // The packets are seperated by 5 ms.

Conceptual Specification

* Before create a detailed specification, it is an initial and simplified specification.
- It gives good practice in specification and UML
→ Good idea in general to identify potential problems before investing too much effort in detail.

* In the model train control system, there are two major subsystems
  1) Command unit
  2) Train-board component

Class diagram for the train controller message



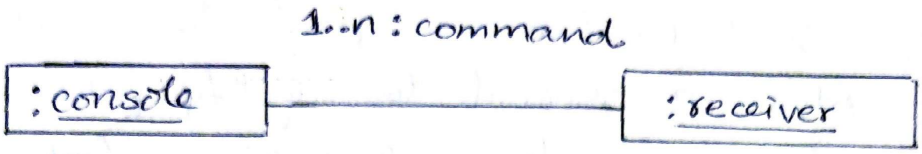
Command name	Parameter
Set-speed	Speed (+ve/-ve)
Set-inertia	inertia value (non-negative)
Estop	non.

Roles of message classes

⇨ Implemented message classes derived from message class
- Attributes & operations will be filled in for detailed specifications.

⇨ Implemented message classes specify message type by their class
- may have to add types as parameter to data structure in implementation.

Subsystem collaboration diagram shows relationship between console & receive (ignores role of track)



Subsystem structure modeling

→ Some classes define non-computer components denoted by *

→ choose important system at this point to show basic relationships.

Major subsystem & its roles

Console

→ read state of front panel
→ Format message
→ transmit messages.

Train

→ receive message
→ interpret message
→ control the train

Console class roles

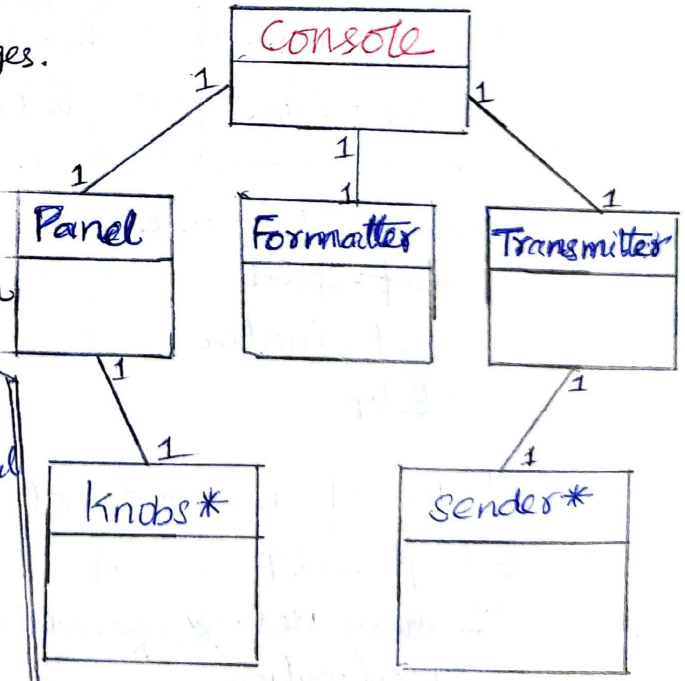
Panel : describe the analog knobs and hardware to interface the digital parts of the system

Formatter : turns knob settings into bit (format) streams.

Transmitter : this class interfaces to analog electronics to send the message along the track.

Knobs* : analog knobs, buttons & levers.

Sender : Analog electronics that send bits along the track.



*=physical Object

A UML class digram for the train controller showing the composition of the subsystem - Console.

-38-

Train Class roles

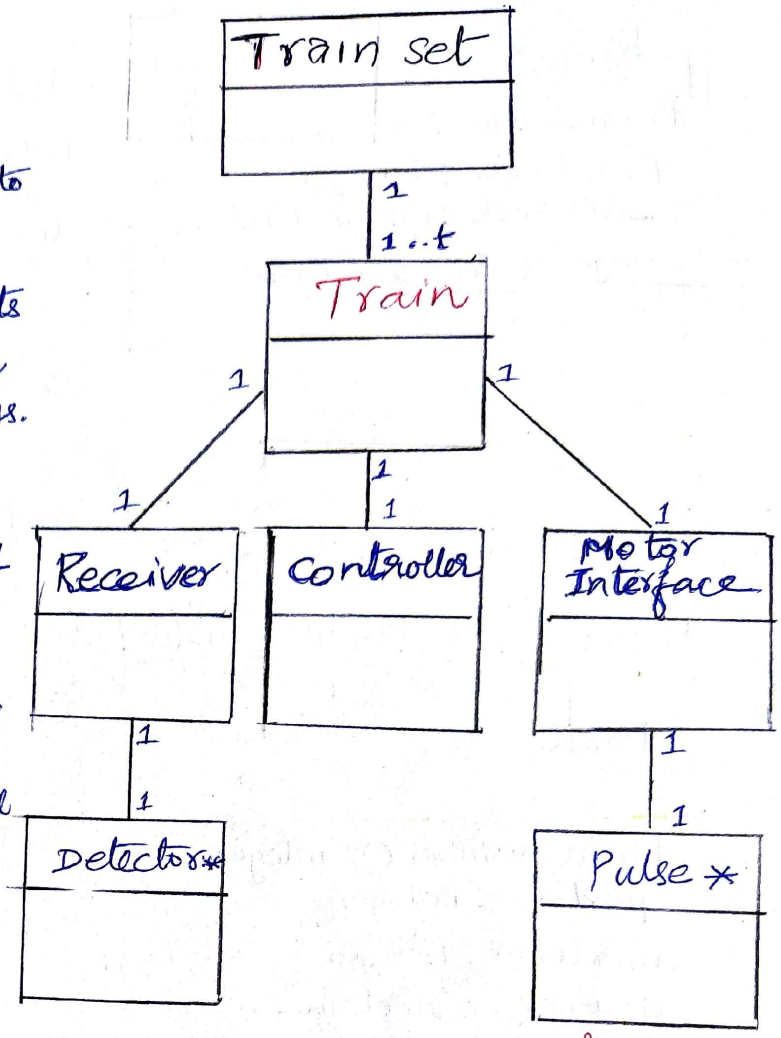
Receiver class knows how to turn the analog signals on the track into digital form.

Controller class interprets received commands & makes control decisions.

Motor Interface class generates signal required by the motor.

Detector* detects analog signals on the track & converts them into digital format

Pulser* turns digital commands into analog signals required to control the motor speed



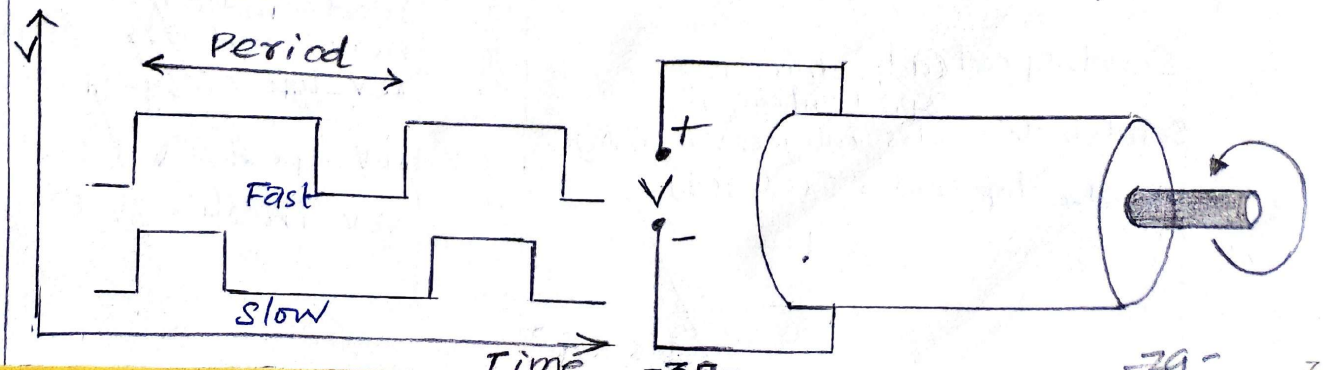
UML class diagram for the train subsystem.

DETAILED SPECIFICATION

⇒ We can now fill in the details of the conceptual specification:
- more classes
- behaviors

⇒ Sketching out the specification first help us to understand the basic relationship in the system.

Train Speed Control : Speed is controlled by PWM



-39-

Console - physical Object classes

Knobs*
train_knob : integer Speed_knob : integer inertia-knob : unsigned integer emergency_stop: boolean

Pulser*
Pulse_width : unsigned_integer direction : boolean

Sender*
Send-bit()

detector*
read-bit() : integer

Panel and motor interface classes

Panel
train_number() : integer speed() : integer inertia() : integer estop() : boolean new_setting()

motor_interface
Speed : integer

Class description

* Panel class defines the controls.
  - new_setting() behaviour reads the controls.

* Motor-interface class defines the motor speed held as state

Transmitter & Receiver Classes

Transmitter
Send-speed (adrs : integer, speed : integer) Send-inertia (adrs : integer, val : integer) send-estop (adrs : integer)

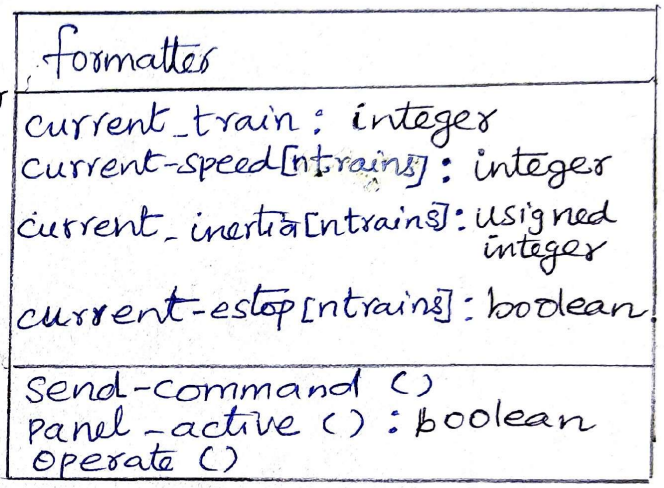
Receiver
current : command new : boolean
read-cmd() new-cmd() : boolean rcv_type (msg_type : command) rcv_speed (val : integer) rcv_inertia (val : integer

Class description

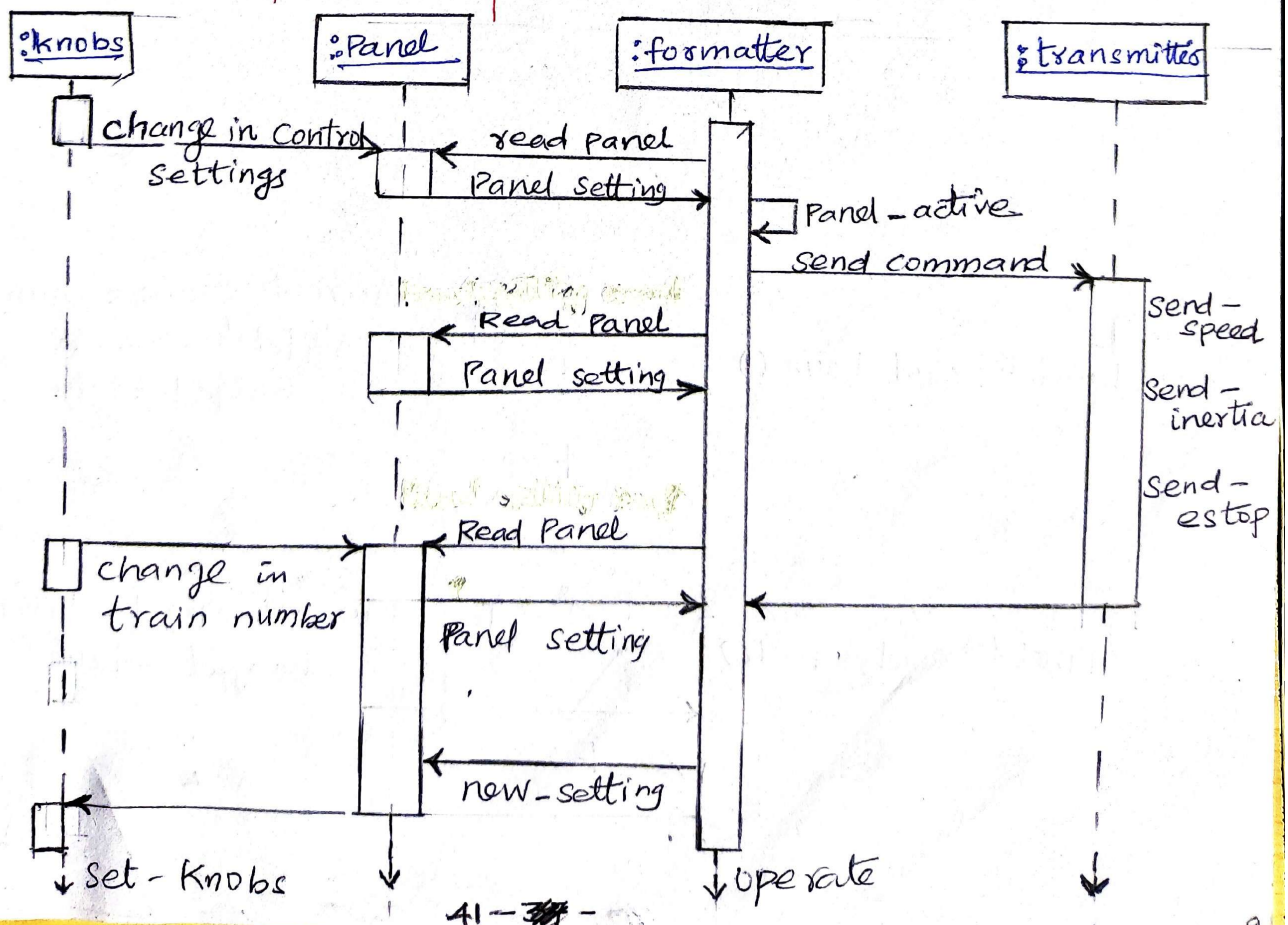
* Transmitter class has one behaviour for each type of message sent
* Receiver function provides methods to:
  - detect a new message;
  - determine its type;
  - read its parameters (estop has no parameters)

Formatter class

* Formatter class description
  - Formatter class holds state for each train setting for current train.
  - The operate () operation performs the basic formatting task.



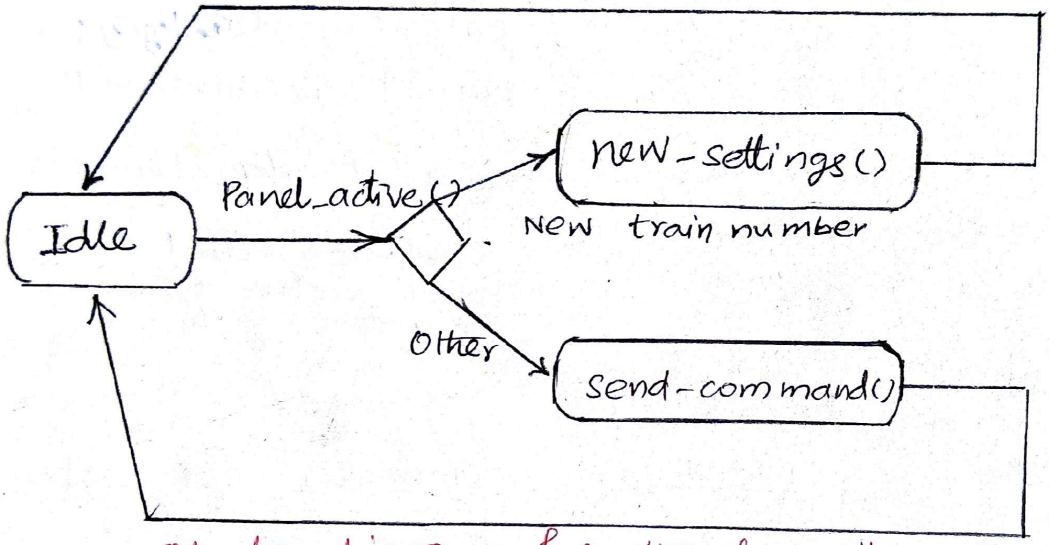
Control input sequence diagram



Control input cases

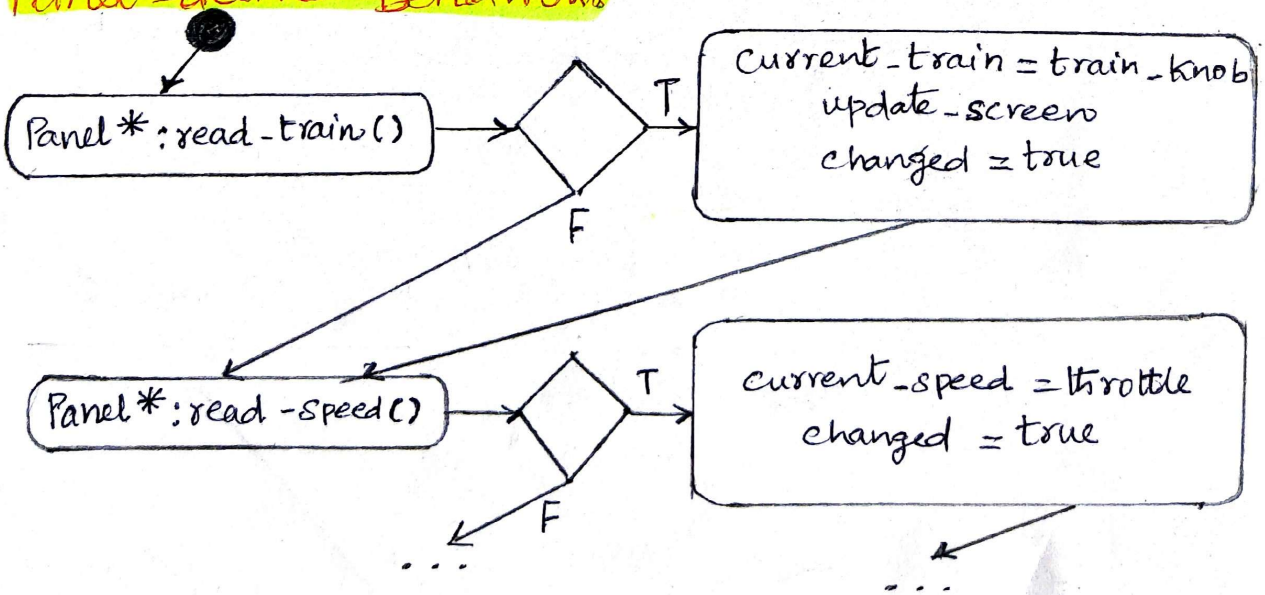
* use a soft panel to show current panel setting for each train
* Changing train number:
  - must change soft panel setting to reflect current train's speed, etc.
* controlling throttle / inertia / estop.
  - read panel, check for changes, perform command.

Formatter Operate Behaviour

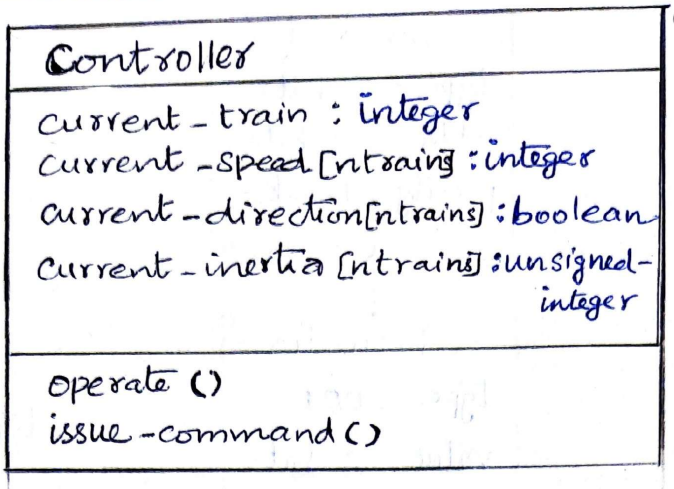


State diagram for the formatter operate behaviour

Panel-active Behaviour



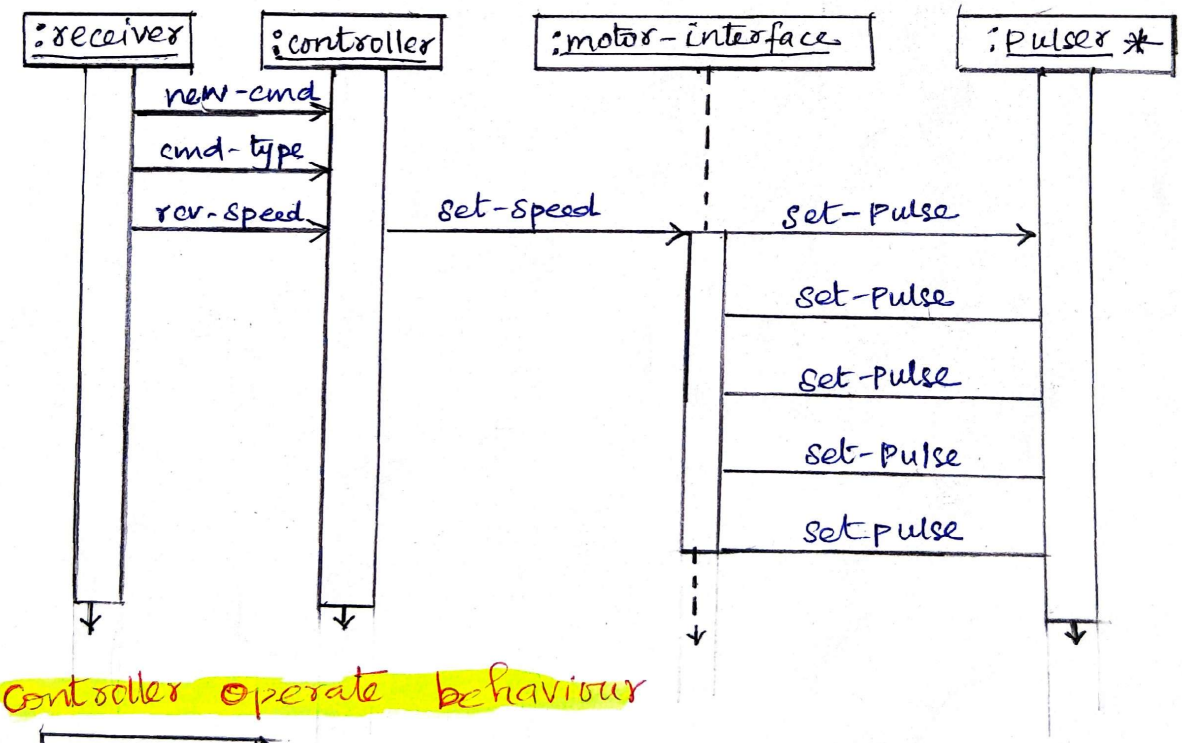
Controller class



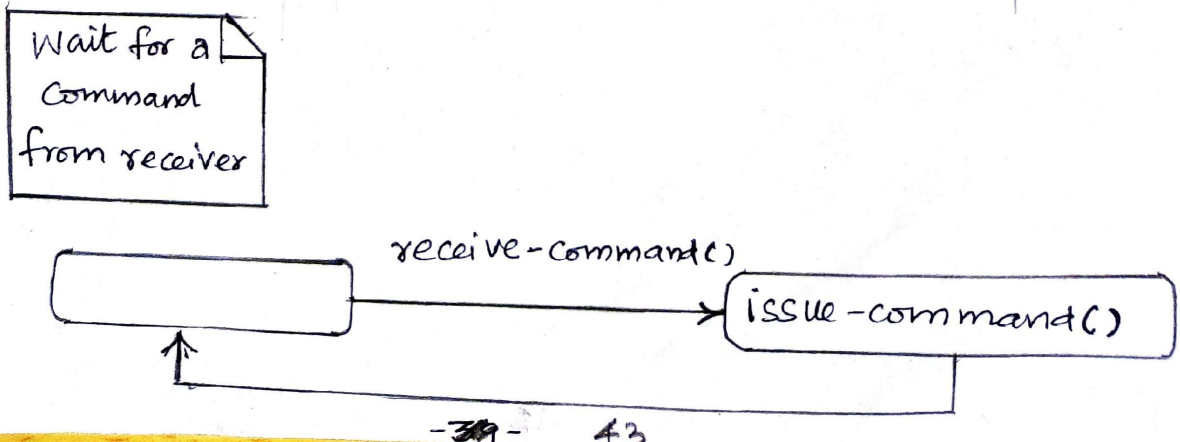
Setting the speed

* Don't want to change speed instantaneously.
* controller should change speed gradually by sending several commands.

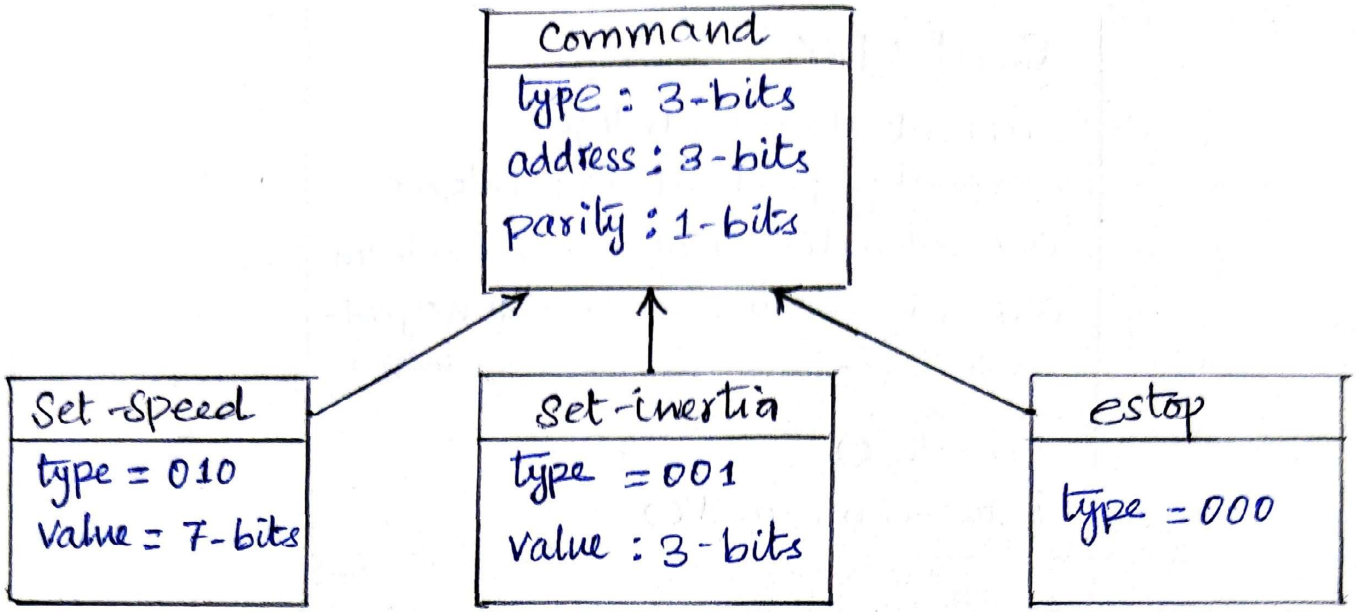
Sequence diagram for set-speed command



Controller operate behaviour



Refined command classes



⇨ Seperate specification and programming
  * Small mistakes are easier to fix in the specification
  * Big mistakes in programming cost a lot of time.

⇨ It is not possible to seperate specification & architecture.

Design Methodologies

⇨ An effective design methodology should be defined in terms of design objectives and design choice, to be applied at each consecutive design step.

⇨ The design objectives and choice determine milestones to be achieved during a certain instance of a design process.

⇨ As embedded system are designed to perform dedicated functions, like real-time computing constraints.

⇨ In most cases, they are made of components that communicate with each other and the environment via Sensors and actuators.

⇨ Challenges in the design of embedded systems

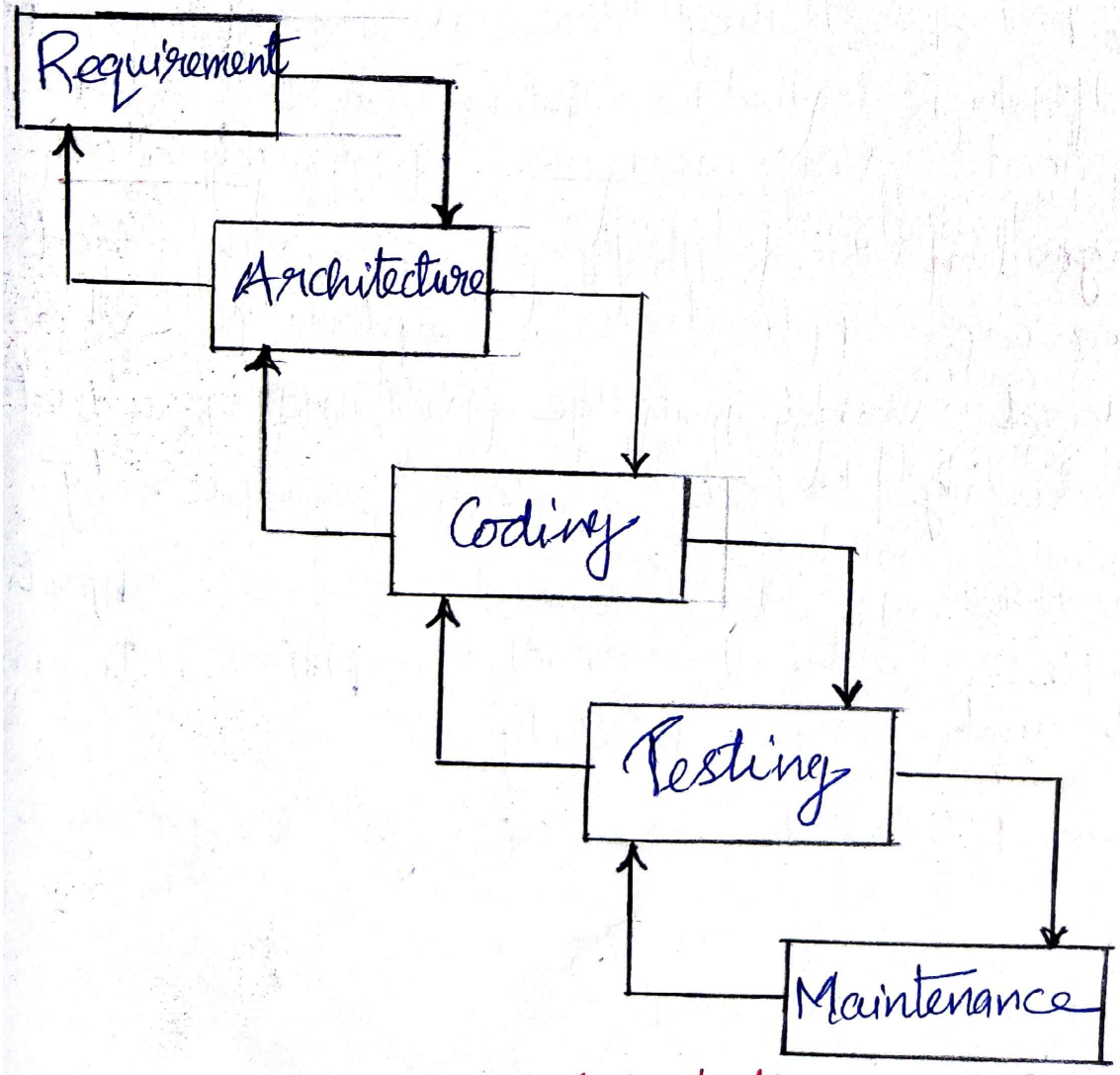
1. Increasing application complexity even in standard and large volume products.
2. Increasing target system complexity.
3. Numerous constraints and design objectives; example: cost, power consumptions, timing constraints, dependability.
4. Reduced and overlapping design cycles.

Design Flow

Design flow is a sequence of steps to be followed during a design. Some of the steps can be performed by tools and other steps can be performed by hand.

Waterfall Model:

The below figure shows the waterfall model introduced by Royce, the first model proposed for the software development process.



Waterfall Model.

In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. The outcome of one phase acts as the input for the next phase sequentially.

Requirements: Defines needed information, function, behaviour, performance and interfaces.

Architecture: Data structures, software architecture, interface representations, algorithmic details.

Implementation: Source code, database, user documentation, testing.

Testing: All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.

Maintenance: There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Advantages of Waterfall model:

1. Simple and easy to understand and use.
2. Provides structure to inexperienced staff.
3. Milestones are well understood.
4. Sets requirements stability
5. Good for management control (plan, staff, track)
6. Works well when quality is more important than cost or schedule.

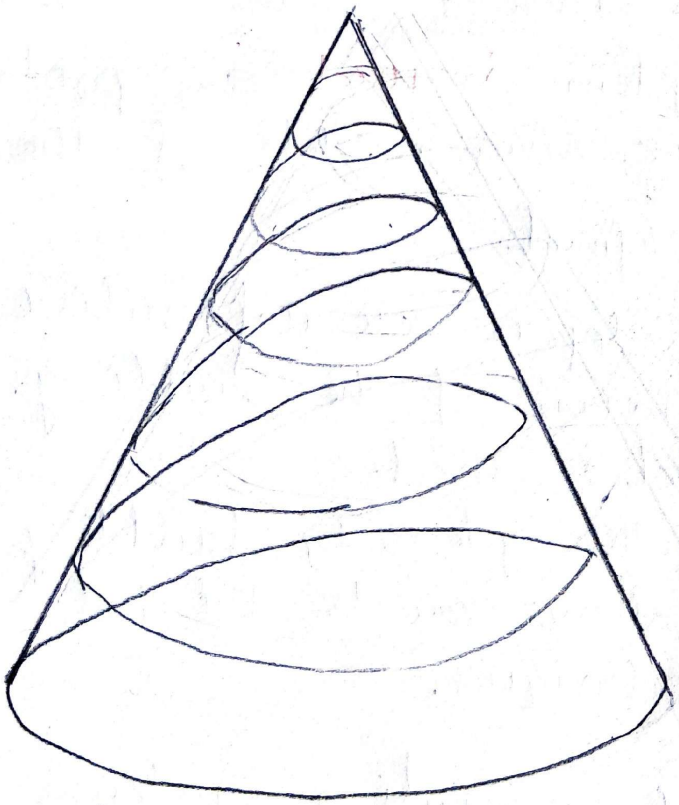
Disadvantages of waterfall model

1. Poor model for long and ongoing projects.
2. High amounts of risk and uncertainty.
3. It is difficult to measure progress within stages
4. Cannot accommodate changing requirements.

Spiral model:

* The following figure shows Spiral model of Software design. Spiral model uses a cyclic approach to incrementally develop the software product while decreasing its degree of risk.
* This model uses planning risk analysis, engineering and evaluation phases.
* A software project repeatedly passes through these in cycles.

* The spiral model uses planning, risk analysis, engineering and evaluation phases.
* Projects split into smaller sub-projects: each iteration corresponds to a smaller projects.
* In spiral model, before each cycle to involve stake holders to decide on scope of cycle to arrive at plan.
* Following this risk analysis is carried out to arrive at list of risks. stake holders performs concurrent engineering to adress these risks through pro-typing and before the completion of current cycle.



* Software is produced in the engineering phase and tested at the end of the phase. The evaluation phase allows the customer to evaluate the output of the project before the project continues to the next spiral.

* Spiral models are found very effective for large and mission-critical projects.

* In spiral model, entire project cost is very high since too many risk analysis and pro-typing are involved. Also, spiral model demands highly specific expertise to perform risk analysis.

Successive Refinement Design Methodology:

* The following figure shows successive refinement design methodology.

* In this method, system is built many times First system is used as prototype model and successive models of the system are further refined.

* When developers are unfamiliar with the application domain of the building the system, this model is used.

* Refining the system by building several complex systems allows you to test out architecture and design techniques.

* Embedded computing systems often involve the design of hardware/software project.