EMBEDDED SYSTEM DESIGN PROCESS

Methodology:

Understanding your design methodology helps you to ensure that you didn't skip anything. A design methodology is important for the following three reasons:

(i) Optimizing Performance:

It allows us to keep a scorecard on a design to ensure that everything have done whatever we need to do, such as optimizing the performance or performing functional tests.

(ii) Automated Steps:

Developing a single program which emits a completed design with the concept of an embedded system would be a risk task that can be resolved by breaking the process into manageable steps and it can work on automating the steps one at a time.

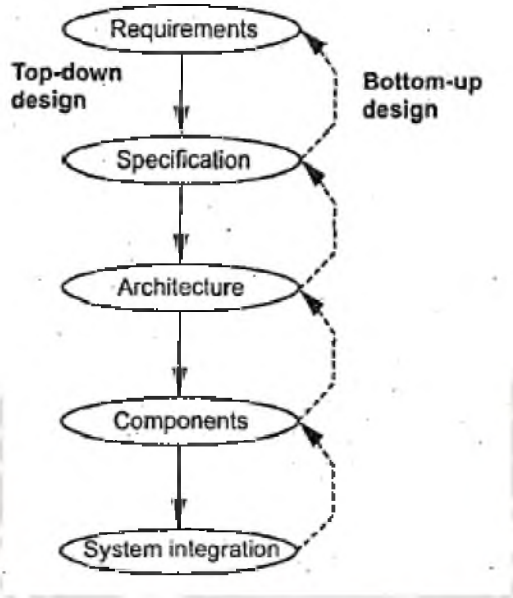
It allows us to develop Computer-Aided Design (CAD) tools that helps us for automate methodology steps and to keep track of the methodology itself.

(iii) Easy Understanding:

Design methodology makes it much easier for members of a design team to communicate. Most of the embedded systems are designed by teams, so, the coordination is the most important role of a well-defined design methodology.

Steps

summarizes the major steps involved in the embedded system design process. In top-down design, we will begin with the most abstract description of the system and it concludes with the concrete details. But in the bottom-up design, we start from small components to build a large system. The real design
often uses the both techniques.



In this top-down view, we start with the system requirements and specification that is we create a more detailed description of what we want?

The bottom-up design information to help us refine the system and its steps are dashed-line arrows.

The specification states only how the system behaves, not how it is built? The details of the system's internals begin to take shape when we develop the architecture, which gives the system structure in terms of large components.

Once we know the needed components, we can design our need using those components, including both the software modules and any specialized hardware. Based on those components, we can finally build a complete system.

Major Goals:

The major goals of the embedded system design are,

(i) Manufacturing cost,

(ii) Performance (both overall speed and deadlines), and

(iii) Power consumption.

Tasks:

The tasks which should be performed at every step in the design process:

(i) We must analyze the design at each step to determine how we can meet the specifications.

(ii) We must then refine the design to add detail.

(iii) We must verify the design to ensure that it still meets all system goals, such as cost, speed, and so on.

Requirements

At the initial stages of the design process, we must know what we are designing? This information is useful in creating the architecture and components. We generally proceed this information in two phases:

(i) We gather an informal description from the customers known as requirements, and

(ii) We refine the requirements into a specification that contains enough information to begin the designing of the system architecture.

Validating Requirements

Validating a set of requirements is a psychological task because it requires understanding both i.e., what the people want and how they communicate those needs.

One good way is an user interface portion of a system's requirements is to build a mock-up which may uses canned data to simulate the functionality in a restricted demonstration, and it may be executed on a PC or a workstation.

It should give the customer a good idea that how the system will be used and how the user can react to it. Generally, the nonfunctional models of devices can also give customers a better idea of characteristics such as size and weight.

Simple Requirements Form

Requirements analysis for a big system can be complex and a time consuming I one. Capturing a relatively small amount of information in a clear and simple format is a good start toward understanding the system requirements.

Sample requirements form that can be filled out at the start of the project. We can use this form as a checklist and it is considered as the basic characteristics of the system.

(i) Name:

This is simple and helpful. The name of the project can also be useful to develop the purpose of the design.

(ii} Purpose:

It is a one or two-line description that what the system is supposed to do.

(iii) inputs and outputs:

The inputs and outputs to the system gives an idea about the following de Types of data, Data characteristics, and Types of I/O devices.

(iv) Functions:

The function is a more detailed description of what the system does. A good . way to approach this is to work from the inputs to the outputs.

(v) Performance:

Many embedded computing systems spend at least some time to control the physical devices. These computations must be performed within a certain time frame.

It is essential that the performance requirements be identified early because they must be carefully measured during implementation to ensure that the system works properly.

(vi) Manufacturing Cost:

This includes primarily the cost of the hardware components.

(vii) Power:

We may have only a rough idea of how much power the system can consume. Typically, the most important decision is whether the machine will be battery powered or plugged into the wall.

E Battery-powered machines must be much more careful about how they spend energy.

Specification

A The specification serves as the contract between the customer and the architects. It must be carefully written so that it can accurately reflects the customer's requirements that can be clearly followed during the design.

Specification is essential in creating the working systems with a minimum of designer effort.

The specification should be understandable because someone can verify that whether it meets system requirements and overall expectations of the customer.

A Designers can run into several different types of problems caused by unclear specifications. So, the designers must know what they need to build.

Architecture Design

Definition:

The architecture is a plan for the overall structure of the system that will be used later to design the components that make up the architecture. The creation of the architecture is the first phase of what many designers think of as design.

Designing Hardware and Software Components

The designer must spend time for architecting the system before start coding or designing the circuits. Some components are ready-made and some can be modified from existing designs, others must be designed from the scratch.

In the moving map, the GPS receiver is a good example of a specialized component which will be a predesigned, standard component. We can also make use of standard software modules.

System Integration

After all the components are built, we can putting them together and check the whether the system is working or not. Bugs are typically found during this system integration, and good planning can help us to find the bugs quickly.

Formalisms for System Design

A Unified Modeling Language (UML) is an object-oriented modeling language I which is designed to be useful at many levels of abstraction in the design process.

A Object-oriented design emphasizes two important concepts:

(i) It encourages the design to be described as a number of interacting objects, rather than a few large monolithic blocks of code.

(ii) We can also use UML to model the outside world that interacts with our system such objects may be a people or other machines. Thinking of the design in terms of actual objects helps us understand the natural structure of the system.

A Object-Oriented (OO) specification can be viewed in two complementary ways:

(i) It allows a system to be described in a way that closely models the real-world objects and their interactions.

(ii) It provides a basic set of primitives that can be used to describe systems with a particular attributes, irrespective of the relationships of those systems' components to the real-world objects.

A Both object-oriented specification and object-oriented programming languages provide a basic methods that are similar for structuring large systems.

Structural Description

A Structural description means the basic components of the system. The principal component of an object-oriented design is the object. An object includes a set of attributes that define its internal state.

When implemented in a programming language, these attributes usually become variables or constants held in a data structure.

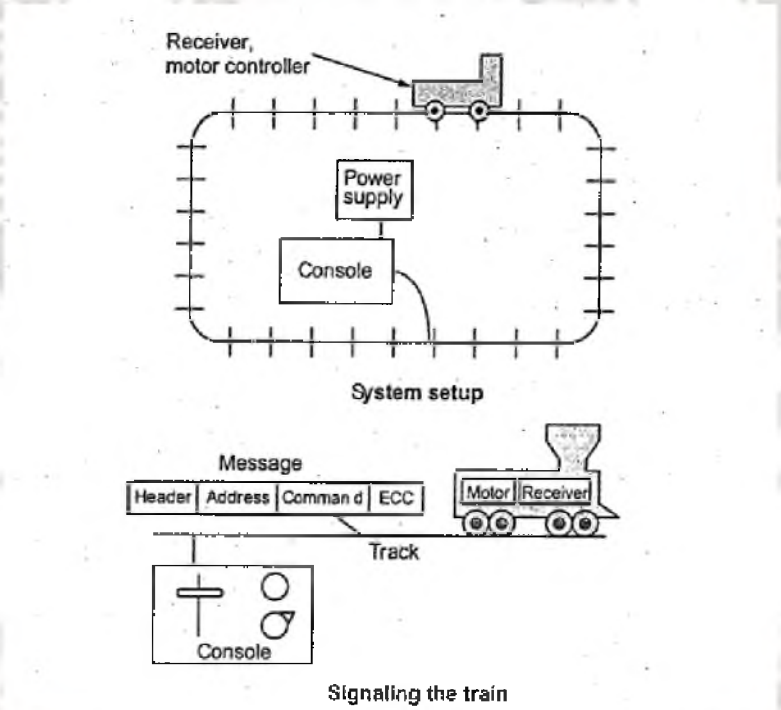
An object describing a display such as a CRT screen in UML. The text in the folded-corner-page icon is a note which does not correspond to an object in the system and only serves as a comment.

DESIGN EXAMPLE: MODEL TRAIN CONTROLLER

Introduction :

Let us consider a simple system that is, a model train controller which is illustrated in order to learn how to use UML to model the systems.

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 train receives its electrical power from the two rails of the track 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 includes analog electronics to sense the bits being transmitted and a control system to set the train motor's speed and direction based on those commands.

Each packet includes an address so that the console can control several trains on the same track and the packet also includes an Error Correction Code (ECC) to guard against transmission errors.

This is a one-way communication system. So, the model train cannot send commands back to the user.

Requirements

The basic set of requirements for the system are,

(i) The console shall be able to control up to eight trains on a single track.

(ii) The speed of each train shall be controllable by a throttle to at least 63 different levels in each direction i.e., forward and reverse.

(iii) There shall be an inertia control that should allow the user to adjust the responsiveness of the train to commanded changes in speed. The inertia control will provide atleast eight different levels.

(iv) There shall be an emergency stop button.

(v) An error detection scheme will be used to transmit messages.

Digital Command Control (DCC)

The DCC standard was created by the National Model Railroad Association tc support an interoperable digitally controlled model trains.

The DCC was created to provide a standard that could be built by any manufacturer.. Therefore, we can use mix and match components from multiple vendors.

(1) DCC Standards

(i) Standard S-9.1

This is the DCC electrical standard that defines how bits are encoded on the rails for transmission.

(ii) Standard S-9.2

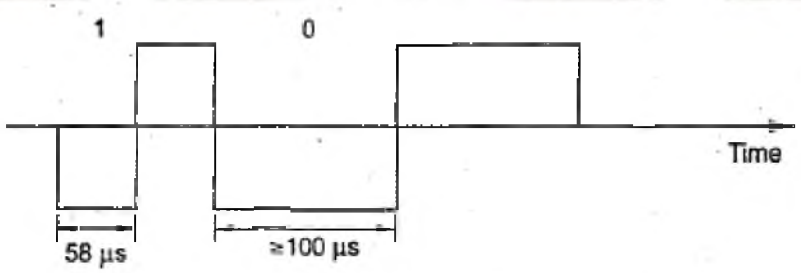
This is the DCC communication standard that defines the packets that carry information.

(2) DCC Electrical Standard

The DCC electrical standard deals with voltages and currents on the track. This standard must be carefully designed because the main function of the track is to carry power to the locomotives.

The signal encoding system should not interfere with power transmission either to DCC or non-DCC locomotives. A key requirement is that the data signal should not change the DC value of the rails.

The data signal swings between two voltages around the power supply voltage. Bits are encoded in the time between transitions, not only by voltage levels. The bit time of '0' is atleast 1 OOjis while a 1 is nominally 58us.



The specification also gives the allowable variations in bit times that a conforming DCC receiver must be able to tolerate.

This standard also describes1 other electrical properties of the system, such as allowable transition times for signals.

(3) DCC Communication Standard

The DCC communication standard describes how bits are combined into packets. The basic packet format is expressed in a regular expression as,

PSA(sD) + E where,

P - Preamble, which is a sequence of atleast 101 bits.

S - Packet start bit which is a 0 bit.

A - Address data byte that gives the address of the unit which is 8 bits long.

s - Data byte start bit, which is a 0 bit.

D - Data byte, which is 8 bits long and it may contain an address, instruction, data, or error correction information.

E - Packet end bit, which is a 1 bit.