EC8791 EMBEDDED AND REAL TIME SYSTEMS

UNIT I INTRODUCTION TO EMBEDDEDSYSTEM DESIGN

Complex systems and micro processors– Embedded system design process –Design example: Model train controller- Design methodologies- Design flows - Requirement Analysis – Specifications-System analysis and architecture design – Quality Assurance techniques - Designing with computing platforms – consumer electronics architecture – platform-level performance analysis.

PART A

1. Differentiate top down and bottom up design.[APRIL 2014]

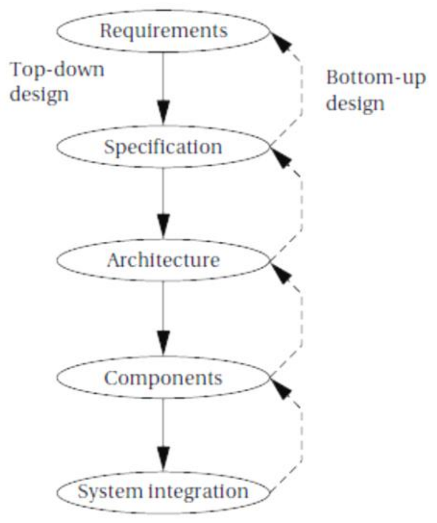


FIGURE 1.1
Major levels of abstraction in the design process.

Top–down view	Bottom–up view
In this top–down view, we start with the system *requirements*. In the next step, *specification*, we create a more detailed description of what we want. But 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	The alternative is a bottom–up view in which we start with components to build a system Decisions at one stage of design are based upon estimates of what will happen later: How fast can we make a particular function run? How much memory will we need? How much system bus capacity do we need? If our estimates are inadequate, we may have to backtrack

components. Once we know the components we need, we can design those components, including both software and hardware	and amend our original decisions to take the new facts into account. In general, the less experience we have with the design of similar systems, the more we will have to rely on bottom-up design information to help us refine the system.

2. Define embedded computer system and what are the challenges in embedded computing system design.(Or)Why embedded computing is more suitable for real time systems?
(Nov /Dec 2015) (Apr/May 16)

Embedded computer system

❖ It is any device that includes a **programmable computer** but is not itself intended to be a general-purpose computer.

❖ Thus, a PC is not itself an embedded computing system, although PCs are often used to build embedded computing systems.

❖ But a **fax machine or a clock** built from a microprocessor is an embedded computing system.

Challenges in embedded computing system design

- Hardware constraints
- Meeting a deadline
- Minimizing power consumption
- Design for upgradability
- Really working or not.

3. **What are all the application areas of the embedded systems?**

- Telecom Smart Cards,
- Missiles and Satellites,
- Computer Networking,
- Digital Consumer Electronics, and Automotive

4. Define UML with an example

Creating requirements and specifications, architecting the system, designing code, and designing tests. It is often helpful to conceptualize these tasks in diagrams; there is a *visual language that can be used to capture all these design tasks: the **Unified Modeling Language(UML).***

UML is an ***object-oriented*** modeling language but object-oriented design emphasizes two concepts of importance:

■ It encourages the design to be described as a number of interacting objects,

rather than a few large monolithic blocks of code.
■ At least some of those objects will correspond to real pieces of software or hardware in the system.

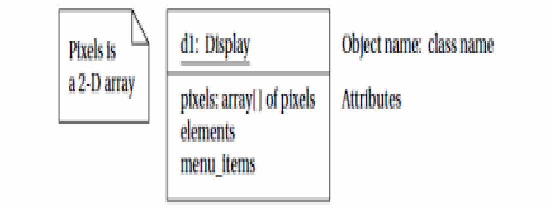


FIGURE 1.5
An object in UML notation.

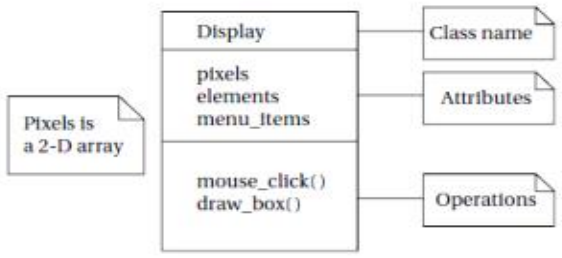


FIGURE 1.6
A class in UML notation.

5. What are the relationship that exists between objects and classes?

There are several types of ***relationships*** that can exist between objects and classes:

- ***Association*** occurs between objects that communicate with each other but have no ownership relationship between them.
- ***Aggregation*** describes a complex object made of smaller objects.
- ***Composition*** is a type of aggregation in which the owner does not allow access to the component objects.
- ***Generalization*** allows us to define one class in terms of another

6. Define Digital Command Control (DCC)and Memory Mapping Unit(MMU)

Digital Command Control (DCC)

❖ The **Digital Command Control (DCC)** standard was created by the National ModelRailroadAssociation to support interoperable digitally-controlled model trains.
The DCC standard is given in two 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 Standard, **defines the packets** that carry information.

PSA(sD) + E

MMU Memory Mapping Unit(MMU)

❖ A MMU translates *addresses between the CPU and physical memory*.
❖ This translation process is often known as ***memory mapping*** since addresses are mapped from a logical space into a physical space.
❖ MMUs are used toprovide ***virtual addressing***.

7. Define compiler, linker and loader?

Compiler

❖ A **compiler** is a computer program (or set of programs) that **transforms source code** written in a programming language (the source language) **into another computer language** (the target language, often having a binary form known as object code).
❖ The most common reason for converting a source code is to create an executable program.

Linker

❖ A **linker** or link editor is a computer program that takes one or more **object files** generated by a compiler
❖ It combines them into a single executable file, library file, or another object file.

Loader

❖ In computing, a **loader** is the part of an operating system that is responsible for **loading programs and libraries**. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution.

8.What are the types of power management features in CPU?

There are two types of power management features provided by CPUs.

- **Static power management**
- **Dynamic power management**

❖ A ***static power management*** mechanism is invoked by the user but does not Otherwise depend on CPU activities. An example of a static mechanism is a ***power down mode*** intended to save energy. This mode provides a high-level way to reduce unnecessary power consumption.
❖ A ***dynamic power management*** mechanism takes actions to control power based upon the dynamic activity in the CPU. For example, the CPU may turn off certain sections of the CPU when the instructions being executed do not need them

9.What is Super Harvard Architecture (SHARC)?

Differentiate Von Neumann Architecture and Super Harvard Architecture (SHARC)?

Von Neumann Architecture	**Harvard Architecture**
The memory holds both data and instructions, and can be read or written when given an address. A computer whose memory holds both data and instructions is known as a ***von Neumann*** machine	Harvard machine has **separate memories** for data and program.
The CPU has several internal ***registers***	The program counter points to program

that store values used internally. One of those registers is the ***program counter (PC)***,which holds the address in memory of an instruction.The CPU fetches the instruction frommemory,decodes the instruction,and executes it. The program counter does not directly determine what themachine does next, but only indirectly by pointing to an instruction in memory	memory, not data memory. As a result, it isharder to write self-modifying programs (programs that write data values, then use those values as instructions) on Harvard machines

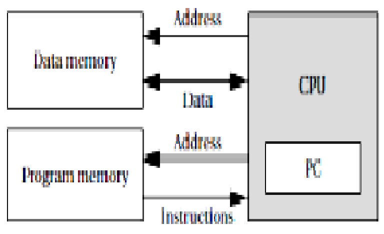


FIGURE 2.2
A Harvard architecture.

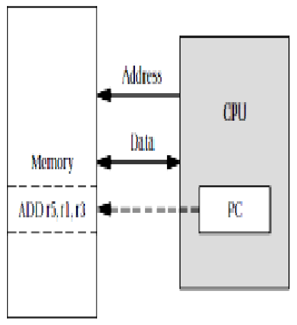


FIGURE 2.1
A von Neumann architecture computer.

10.What is an embedded system?

An embedded system is one that has **computer-hardware with software embedded in it as one of its most important component**. It is a computing device that does a specific focused job.

11.What is watch dog timer?

❖ A timer with timeout from which resets the processor in case the program gets stuck for an unexpected time.
❖ An important timing device in a system that **resets the system after a pre-defined timeout**. This time may be definable within the first few clock cycles after reset.

4. What is kernel?

A program with functions for memory allocation and deallocation,
tasks scheduling, inter-Process communication, effective management of shared memory access by using the Signals ,exception handling signals,

semaphores,queues,mailboxes,pipes,i/management, interrupt controls, device drivers and device management.

12. **Enumerate some embedded computers that are exists from origin of embedded Systems. (Nov/Dec 16)**

- Apollo Guidance Computer,
- Autonetics D-17 guidance computer

13. **Mention the various methods for reading from or writing to an I/O port (Apr/May 17).**

Microprocessors can provide programming support for input and output in two ways: ***I/O Instructions*** and ***memory-mapped I/O***.

14. **What are the roles of microprocessor in Embedded system? (Nov 17)**

- First, microprocessors execute programs very efficiently.
- Second, microprocessor manufacturers spend a great deal of money to make their CPUs run very fast.

15. Compare the functions of CPU and Co-processor (Apr/May 19)

CPU	CO-PROCESSOR
CPU is the main processing unit of the computer that performs arithmetic, logic and control operations according to the instructions	**Co-processors** are attached to the CPU and implement some of the instructions.
Maintains proper functioning of the entire computer	It helps the processor to increase the system performance

16. **Define assembler (Apr/May 19)**

- ***Assembler*** splits the program line by line.
- Assemblers must also provide some ***pseudo-ops*** to help programmers create complete assembly language programs.

17. **Determine the average memory access time of machine whose hit rate is 90% with a cache access time of 3ns and main memory access time of 70ns. (Nov/Dec 18)**

Solution:

Given

h= hit rate=90%=0.9

t_{cache}=cache access time=3ns=$3x10^{-9}$

t_{main}=main memory access time = 70ns=$70x10^{-9}$

<u>**Formula**</u>

$$t_{av} = ht_{cache} + (1 - h)t_{main},$$

Answer

t_{av}=0.9x3x10^{-9} + ((1-0.9)*70x10^{-9}) = 3.4x10^{-9}=3.4ns

18. Why design methodologies are used in embedded system?

Process is important because without it, we can't reliably deliver the products we want to create. The obvious goal of a design process is to create a product that does something useful.

19.List the product metrics and goals used in design process (or) Mention the goals of design process in embedded computing systems Apr 18

- ✓ *Functionality*
- ✓ *manufacturing cost*
- ✓ *power consumption*
- ✓ *Time-to-market*
- ✓ *Design cost*
- ✓ *Quality*

20. **What is meant by design flows?**

A design flow is a sequence of steps to be followed during a design. Some of the steps can be performed by tools, such as compilers or computer-aided design (CAD) systems; other steps can be performed by hand.

21. List the characteristics of design flows used in embedded system. Nov 17

- ✓ *Waterfall model*
- ✓ *Spiral model*
- ✓ *Successive refinement*
- ✓ *Hierarchical design flows*
- ✓ *Concurrent engineering*

22. Define TCP?

- ✓ TCP guarantees the reliable, in order delivery of a stream of bytes.
- ✓ It is a full-duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction.

23. List out the phases of waterfall model (Nov/Dec 16)

- ✓ *Requirements*
- ✓ *Architecture*
- ✓ *Coding*
- ✓ *Testing*
- ✓ *Maintenance*

24. What do you meant by spiral model? (Nov/Dec 16)

The spiral model assumes **that several versions of the system will be built.** Early systems will be simple mock-ups constructed to aid designers' intuition and to build experience with the system. As design progresses, more complex systems will be constructed. At each level of design, the designers go through requirements, construction, and testing phases. At later stages when more complete versions of the system are constructed, each phase requires more work, widening the design spiral.

25. Differ waterfall model Vs Spiral model

The waterfall model assumes that **the system is built once in its entirety**, the spiral model assumes that several versions of the system will be built

26. What is meant by successive refinement?

In this approach, **the system is built several times.** A first system is used as a rough prototype, and successive models of the system are further refined. Refining the system by building several increasingly complex systems allows you to test out architecture and design techniques.

27. What do you meant by concurrent engineering?

- ✓ Concurrent engineering **attempts to take a broader approach and optimize the total flow.**
- ✓ Reduced design time is an important goal for concurrent engineering.
- ✓ It tries to eliminate "over-the-wall" design steps, in which one designer performs an isolated task and then throws the result over the wall to the next designer, with little interaction between the two.
- ✓ In particular, concurrent engineering usually **requires eliminating the wall between design and manufacturing.**

28. Define requirement analysis

- Requirements are informal descriptions of **what the customer wants**, while specifications are more detailed, precise, and consistent descriptions of the system that can be used to create the architecture.
- Both requirements and specifications are, however, directed to the outward behavior of the system, not its internal structure.

29. List the types of requirements used in embedded design (or) List out some of the verification and specification related to the design flow.(APR/MAY 2017)

There are two types of requirements

- ✓ *Functional*
- ✓ *Non-functional*

A functional requirement states what the system must do, such as compute an FFT. A non-functional requirement can be any number of other attributes, including physical size, cost, power consumption, design time, reliability, and so on.

30. List the functional and non functional requirements

- ✓ *Correctness*
- ✓ *Unambiguousness*
- ✓ *Completeness*
- ✓ *Verifiability*
- ✓ *Consistency*
- ✓ *Modifiability*
- ✓ *Traceability*

31. What do you meant by SDL language?

- ❖ An example of a widely used state machine specification language is the **SDL language,** which was developed by the communications industry for specifying communication protocols, telephone systems, and so forth.
- ❖ SDL specifications **include states, actions, and both conditional and unconditional transitions between states**.
- ❖ SDL is an **event-oriented state** machine model because transitions between states are caused by internal and external events.

32. Define statechart

- ❖ The State chart is one well-known technique for state-based specification that introduced some important concepts. The State chart notation uses an event-driven model. State charts allow states to be grouped together to show common functionality.
- ❖ **There are two basic groupings: OR and AND**

33. What do you meant by CRC cards? (Or) Write the special Characteristics of a CRC cards.(Nov/Dec2018)

The CRC card methodology is a well-known and useful way to help **analyze a system's structure**. It is particularly well suited to object-oriented design because it encourages the encapsulation of data and functions.

- ❖ The acronym *CRC* stands for the following three major items that the methodology tries to identify:
  - • ***Classes*** define the logical groupings of data and functionality.
  - • ***Responsibilities*** describe what the classes do.
  - • ***Collaborators*** are the other classes with which a given class works.

34. How to analyze the system using CRC card methodology?

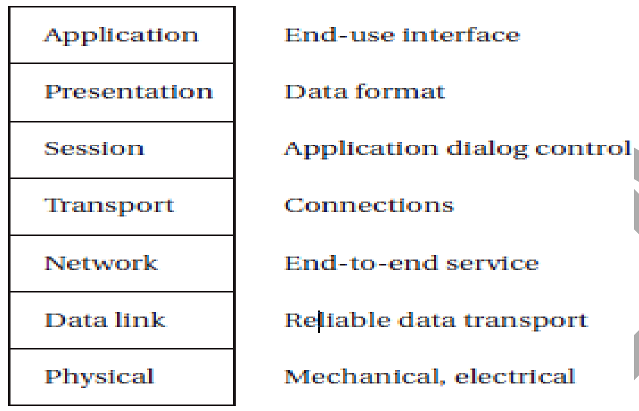
- ✓ *Develop an initial list of classes*
- ✓ *Write an initial list of responsibilities and collaborators*
- ✓ *Create some usage scenarios*
- ✓ *Walk through the scenarios*
- ✓ *Refine the classes, responsibilities, and collaborators*
- ✓ *Add class relationships*

35. Define Capability Maturity Model (CMM)

One well-known way of measuring the quality of an organization's software development process is the **Capability Maturity Model (CMM)** developed by Carnegie Mellon University's Software Engineering Institute [SEI99]. The CMM provides a model for judging an organization. It defines the following five levels of maturity:

- ✓ *Initial*
- ✓ *Repeatable*
- ✓ *Defined*
- ✓ *Managed*
- ✓ *Optimizing*

36. List the seven layers in OSI model



Application	End-use interface
Presentation	Data format
Session	Application dialog control
Transport	Connections
Network	End-to-end service
Data link	Reliable data transport
Physical	Mechanical, electrical

37. State the features of FSK Detection Scheme in MODEM.(NOV/DEC 2018)

A modem converts the binary data from a computer to FSK for transmission over telephone lines, cables, optical fiber, or wireless media. The modem also converts incoming FSK signals to **digital** low and high states, which the computer can "understand.

. ***Direct memory access (DMA)*** is a bus operation that allows reads and writes **not controlled by the CPU**. A DMA transfer is controlled by a ***DMA controller***, which requests control of the bus from the CPU.After gaining control, the DMA controller performs read and write operations directly between devices and memory.

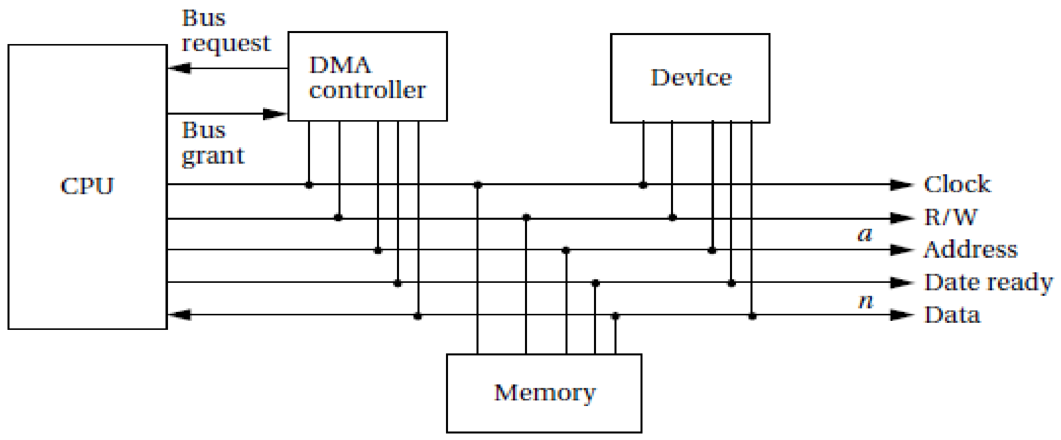


FIGURE 4.9

A bus with a DMA controller.

38. Define Peripheral Component Interconnect (PCI)

PCI (*Peripheral Component Interconnect*) is the dominant high-performance system bus today. PCI uses high-speed data transmission techniques and efficient protocols to achieve high throughput.

The original PCI standard allowed operation up to 33 MHz; at that rate, it could achieve a maximum transfer rate of 264 MB/s using 64-bit transfers. The revised PCI standard allows the bus to run up to 66 MHz, giving a maximum transfer rate of 524 MB/s with 64-bit wide transfers.

39. What is a data flow graph? [APRIL 2014](Apr /may 17)

A ***data flow graph*** is a model of a **program with no conditionals**. In a high-level programming language, a code segment with no conditionals—more precisely, withonly one entry and exit point—is known as a basic block. Figure 5.2 shows a simple basic block. As the C code is executed, we would enter this basic block at the beginning and execute all the statements.

```
w = a + b;
x = a − c;
y = x + d;
x = a + c;
z = y + e;
```

FIGURE 5.2

A basic block in C.

40. **Show the structure of typical CPU bus which supports read/Write? Apr/May**

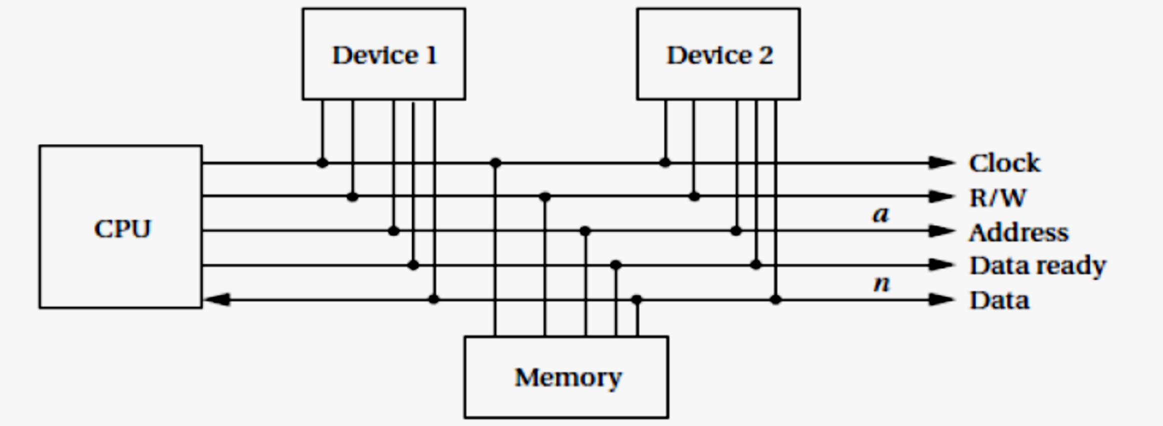


FIGURE 4.2

A typical microprocessor bus.

<u>PART-B</u>

1. EXPLAIN COMPLEX SYSTEMS AND MICROPROCESSORS

<u>Sub-Topics:</u>

- ***A. Embedding Computers***
- ***B. Characteristics of Embedded Computing Applications***
- ***C. Why Use Microprocessors?***
- ***D. The Physics of Software***
- ***E. Challenges in Embedded Computing System Design***
- *F.* ***Performance in Embedded Computing***
- ***G. Different levels of abstraction***

A. Embedding Computers

<u>An Embedded system</u>

An embedded system is one that has **computer-hardware with software embedded in it as one of its most important component.** It is a computing device that does a specific focused job.

B) Characteristics of Embedded Computing Applications

Complex algorithms:

- The operations performed by the microprocessor may be very sophisticated.

User interface:

- Microprocessors are frequently used to control complex user interfaces **that may include multiple menus and many options.**
- The moving maps in Global Positioning System (GPS) navigation are good examples of sophisticated user interfaces.

Real time:

- Many embedded computing systems have to performing real time— **if the data is not ready by a certain deadline, the system breaks.**
- In some cases, failure to meet a deadline is unsafe and can even endanger lives.

Multirate:

- Not only must operations be completed by deadlines, but many embedded computing systems have **several real-time activities going on at the same time.**

Manufacturing cost:

- The total cost of building the system is very important in many cases.
- Manufacturing cost is determined by many factors, including the type of microprocessor used, the amount of memory required, and the types of I/O devices.

Power and energy:

- Power consumption directly affects the cost of the hardware, since a larger power supply may be necessary.
- **Energy consumption affects battery life, which is important in many applications**, as well as heat consumption, which can be important even in desktop applications.

C) Why Use Microprocessors?

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

■ Microprocessors make it easier to design families of products that can be built to provide various feature sets at different price points and can be extended to provide new features to keep up with rapidly changing markets.

There are two factors that work together to make microprocessor-based designs fast.

➢ First, microprocessors **execute programs very efficiently.**

➢ Second, microprocessor manufacturers spend a great deal of money to make their **CPUs run very fast**.

D) The Physics of Software

- **Computing** is a physical act.
- **Software performance** and energy consumption are very important properties in connecting our embedded computers to the real world.
- **Understand the sources** of performance and power consumption.
- As much as possible, we want to make computing abstractions work for us as we work on the physics of our software systems.

E) Challenges in Embedded Computing System Design

- Hardware constraints
- Meeting a deadline
- Minimizing power consumption
- Design for upgradability
- Really working or not

F) Performance in Embedded Computing

- Embedded system designers, in contrast, have a very clear performance goal in mind—their program must meet its ***deadline***.
- At the heart of embedded computing is ***real-time computing***.The program receives its input data; the deadline is the time at which a computation must be finished.
- If the program does not produce the required output by the deadline, then the program does not work, even if the output that it eventually produces is functionally correct.

G.Different levels of abstraction

i.CPU

ii.Platform

iii.Program.

iv.Task

v.Multiprocessor

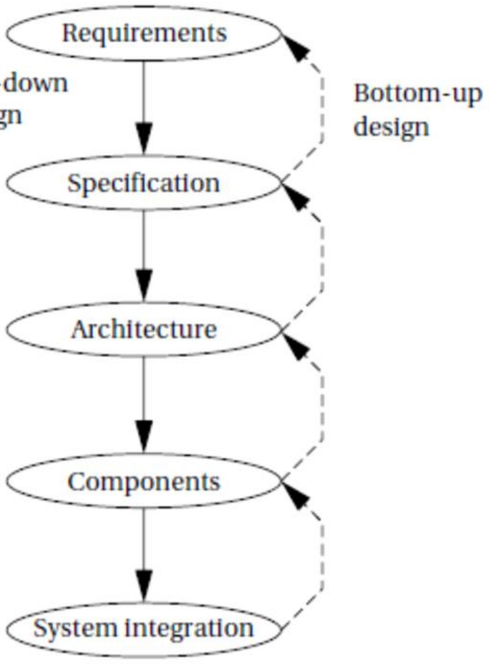
2. Explain embedded system design process.(OR) What are the parameters to be considered while designing an embedded system design process.(OR) Write in detail about the steps or factors involved in Embedded system design process.[MAY/JUNE 2013,APR 2015,2016](or) analyse the requirements for designing the GPS moving map in embedded systems design process (Nov/Dec 16)(Apr/May 18) Demonstrate the challenges and performance of embedded processes for the system design (Nov/Dec 18).

Sub-Topics

A. ***Top down and Bottom up approach***
B. ***Major goals to be considered for design of an embedded system***
C. ***Tasks we need to perform at every step in the design***
   i. ***Requirements (Table)***
   ii. ***Specification***
   iii. ***Architecture***
   iv. ***components***
   v. ***System integration***

A. Top down and Bottom up approach

The embedded system design process id described in both top down and bottom up approach as in below figure:



Top–down view	**Bottom–up view**
In this top–down view, we start with the system *requirements*. In the next step, *specification*, we create a more detailed description of what we want. But the specification states only how	The alternative is a bottom–up view in which we start with components to build a system Decisions at one stage of design are based upon estimates of what will happen later: How fast can

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 components we need, we can design those components, including both software and hardware	we make a particular function run? How much memory will we need? How much system bus capacity do we need? If our estimates are inadequate, we may have to backtrack and amend our original decisions to take the new facts into account. In general, the less experience we have with the design of similar systems, the more we will have to rely on bottom-up design information to help us refine the system.

B. Major goals to be considered for design of an embedded system:

■ Manufacturing cost;
■ Performance (both overall speed and deadlines); and
■ Power consumption.

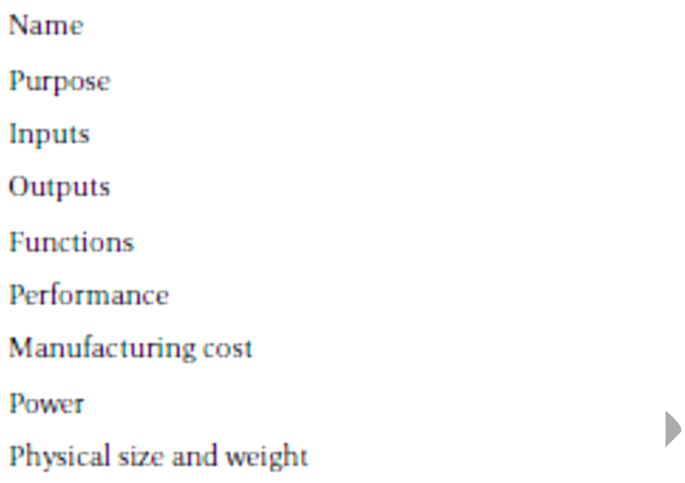
C. Tasks we need to perform at every step in the design process

■ We must ***analyze*** the design at each step to determine how we can meet the specifications.
■ We must then ***refine*** the design to add detail.
■ And we must ***verify*** the design to ensure that it still meets all system goals, such as cost, speed, and so on.

A) Requirements

❖ First, we gather an informal description from the customers known as requirements,
❖ Requirements may be ***functional*** or ***nonfunctional***. We must of course capture the basic functions of the embedded system, but functional description is often not sufficient.

Typical non functional requirements include:
■ Performance
■ Cost
■ Physical size and weight
■ Power consumption



Name

Purpose

Inputs

Outputs

Functions

Performance

Manufacturing cost

Power

Physical size and weight

FIGURE 1.2

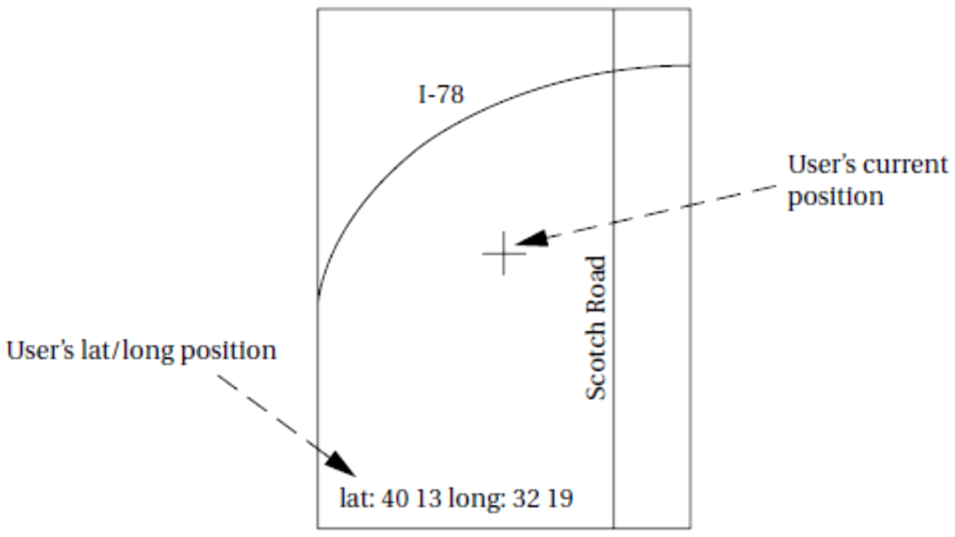
Sample requirements form.

Example

Requirements analysis of a GPS moving map

The moving map is a handheld device that displays for the user a map of the terrain around the user's current position; the map display changes as the user and the map device change position. The moving map obtains its position from the GPS, a satellite-based navigation system. The moving map display might look something like the following figure.

Name	**GPS moving map**
Purpose	**Consumer grade moving map for driving use**
Inputs	**Power button, two control buttons**
Outputs	**Back-light LCD display 400 x600**
Functions	**Uses 5-receiver GPS system; three user-selectable resolutions;always displays current latitude and longitude**
Performance	**Updates screen within 0.25 seconds upon movement**
Manufacturing cost	**$30**
Power	**100mW**
Physical size and weight	**No more than 2" to 6" , 12 ounces**



Requirement(Table)

B) Specification

- ❖ The specification is more precise—it serves as the contract between the customer and the architects. The specification should be understandable enough so that someone can verify that it meets system requirements and overall expectations of the customer.

A specification of the GPS system would include several components:

■ Data received from the GPS satellite constellation.

■ Map data.

■ User interface.

■ Operations that must be performed to satisfy customer requests.

■ Background actions required to keep the system running, such as operating the GPS receiver.

C)Architecture Design

The specification does not say how the system does things, only what the system does. Describing how the system implements those functions is the purpose of the architecture. 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.

Block diagram for GPS moving Map

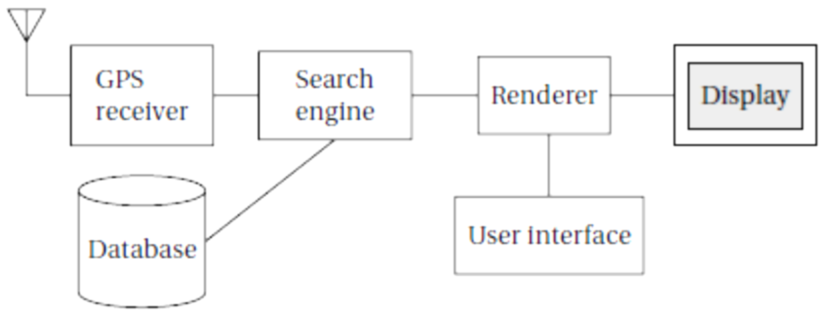


FIGURE 1.3

Block diagram for the moving map.

❖ The hardware block diagram clearly shows that we have one central CPU surrounded by memory and I/O devices.

D. Hardware and Software Components

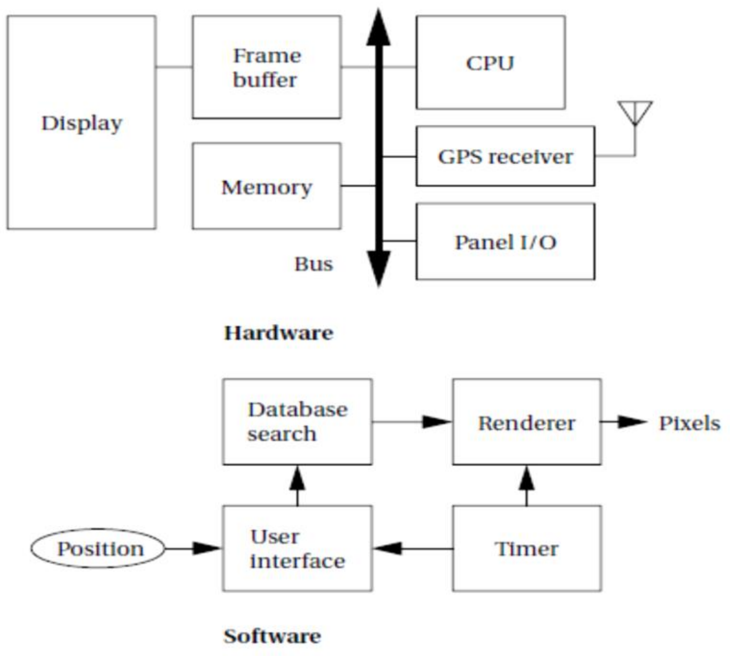


FIGURE 1.4

Hardware and software architectures for the moving map.

E) System Integration

❖ Only after the components are built do we have the satisfaction **of putting them together and seeing a working system**. Of course, this phase usually consists of a lot more than just plugging everything together and standing back. **Bugs are typically found during system integration, and good planning can help us find the bugs quickly.**

❖ By building up the system in phases and running properly chosen tests, we can often find bugs more easily.

- ❖ System integration is difficult because it usually uncovers problems.

3. Discuss about Formalisms for system design. (APR/MAY 2015)

- ➢ A number of different design tasks at different levels of abstraction: creating requirements and specifications, architecting the system, designing code, and designing tests.
- ➢ Using visual language all these design tasks can be captured: the ***Unified Modeling Language(UML)***.
- ➢ UML was designed to be useful at many levels of abstraction in the design process.
- ➢ UML is useful because it encourages design by successive refinement and progressively adding detail to the design, rather than rethinking the design at each new level of abstraction.
- ➢ UML is an ***object-oriented*** modeling language.

Object-oriented design emphasizes two concepts of importance:

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

■ At least some of those objects will correspond to real pieces of software or hardware in the system.

Object-oriented (often abbreviated OO) specification can be seen in two complementary ways:

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

■ 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' components to real-world objects.

Structural Description

- ❖ The principal component of an object-oriented design is, naturally enough, 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.
- ❖ **The text in the folded-corner page icon is a *note*;** it does not correspond to an object in the system and only serves as a comment. The attribute is, in this case, an array of pixels that holds the contents of the display. The object is identified in two ways: It has a unique name, and it is a member of a ***class***.
- ❖ **A class is a form of type definition**—all objects derived from the same class have the same characteristics, although 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 interactswith the rest of the world.

❖ **The *Display* class defines the *pixels* attribute seen in the object**

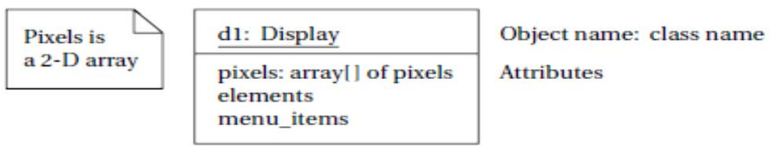


FIGURE 1.5
An object in UML notation.

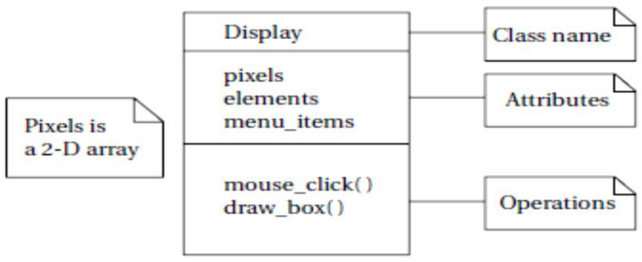


FIGURE 1.6
A class in UML notation.

- A class defines both the ***interface*** for a particular type of object and that object's ***implementation***.
- There are several types of ***relationships*** that can exist between objects and classes:

■ ***Association*** occurs between objects that communicate with each other but have no ownership relationship between them.
■ ***Aggregation*** describes a complex object made of smaller objects.
■ ***Composition*** is a type of aggregation in which the owner does not allow access to the component objects.
■ ***Generalization*** allows us to define one class in terms of another.

- *Unified Modeling Language*, like most object-oriented languages, allows us to **define one class in terms of another**. An example is shown in Figure, where we ***derive*** two particular types of displays.
- **The first, *BW_display*, describes a black and- white display**. This does not require us to add new attributes or operations, but we can specialize both to work on one-bit pixels.
- **The second, *Color_map_display*, uses a graphic device known as a color map** to allow the user to select large number of available colors even with a small number of bits per pixel. This class defines a *color_map*attribute that determines how pixel values are mapped onto display colors.
- A ***derived class*** inherits all the attributes and operations from its ***base class***. In this class, *Display* is the base class for the two derived classes. A derived class is defined to include all the attributes of its base class

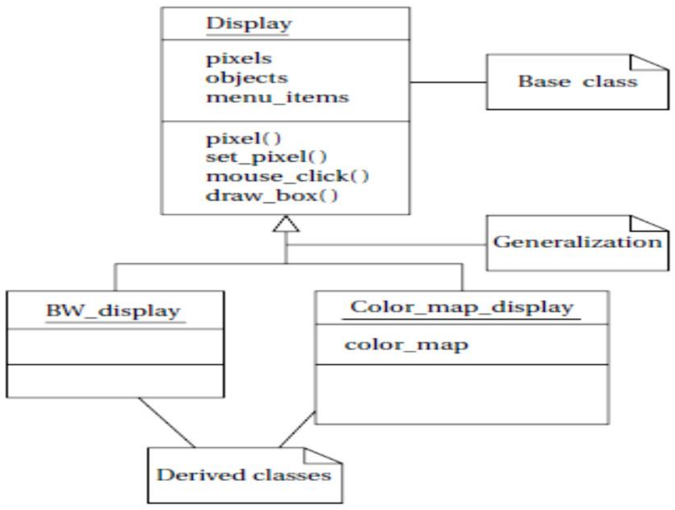


FIGURE 1.7
Derived classes as a form of generalization in UML.

- UML also allows us to define *multiple inheritance*, in which a **class is derived from more than one base class**.
- **A*Multimedia_display*class by combining the *Display* class with a *Speaker* class for sound.** The derived class inherits all the attributes and operations of both its base classes, *Display* and *Speaker*. Because multiple inheritance causes the sizes of the attribute set and operations to expand so quickly, it should be used with care.
- **A *link* describes a relationship between objects**; association is to link as class is to object.We need links because objects often do not stand alone; associations let us capture type information about these links.
- When generalized into classes,we define an association between the message set class and the message class. The association is drawn as a line between the two labeled with the name of the association, namely, *contains*. The ball and the number at the message class end indicate that the message set may include zero or more message objects

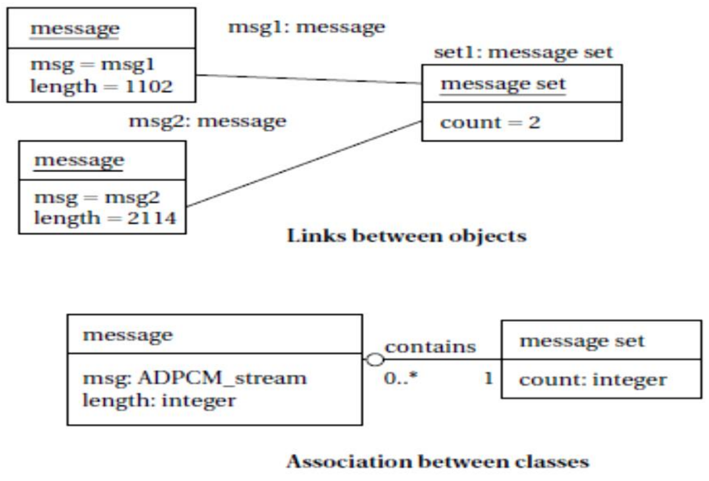


FIGURE 1.9
Links and association.

Behavioral Description

- One way to **specify the behavior of an operation is a*state machine***. Figure 1.10 shows UMLstates; the transition between two states is shown by a skeleton arrow.
- These state machines will not rely on the operation of a clock, as in hardware; rather,**changes from one state to another are triggered by the occurrence of*events.***

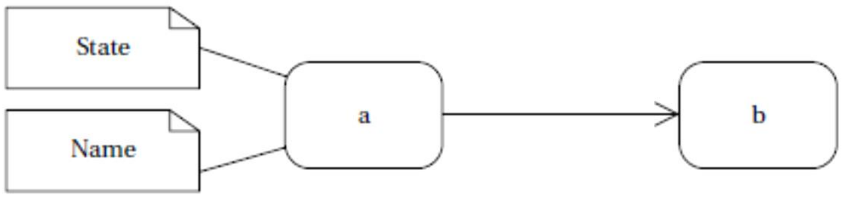


FIGURE 1.10
A state and transition in UML.

- A ***signal*** is an asynchronous occurrence. It is defined in UML by an object that is labeled as a <<*signal*>>. The object in the diagram serves as a declaration of the event's existence. Because it is an object, a signal may have parameters that are passed to the signal's receiver.
- A ***call event*** follows the model of a **procedure call in a programming language**.
- A ***time-out event*** causes the machine to **leave a state after a certain amountof time.**
- The label *tm(time-value)* on the edge gives the **amount of time after which the transition occurs**. A time-out is generally implemented with an external timer.This notation simplifies the specification and allows us to defer implementation details about the time-out mechanism.
- A ***sequence diagram*** is somewhat similar to a **hardware timing diagram**, although the time flows verticallyin a sequence diagram, whereas time typically flows horizontally in a timingdiagram.The sequence diagram is designed to show a particular scenario or choiceof events—it is not convenient for showing a number of mutually exclusive possibilities.

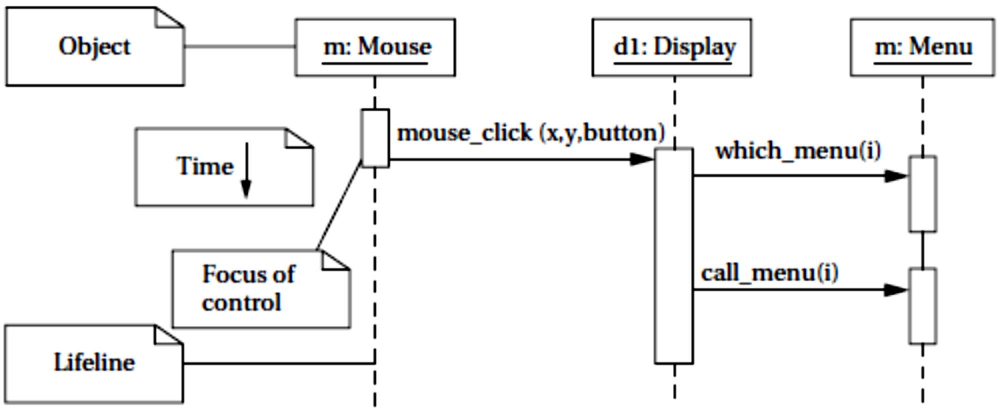


FIGURE 1.13

A sequence diagram in UML.

4. With a simple system namely a model Train controller, how will you use the UML to the model systems.(or) Design a model train controller with suitable diagrams and explain. [MAY/JUNE 2014,APR/MAY 2015]Nov 16, Nov17,Apr 18

Nov/Dec 18

Subtopics

- ✓ ***Diagram for Modern train control system***
- ✓ ***Digital Command Control (DCC)***
- ✓ ***Requirements (Table)***
- ✓ ***Specification***

i. Conceptual Specification-Class and UML diagram for Train controller system

ii. Detailed Specification- Sequence diagram for transmitter

Sequence diagram for Receiver

- ❖ A model train controller, which is illustrated in 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.

- ❖ Since 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.

Modern train control system

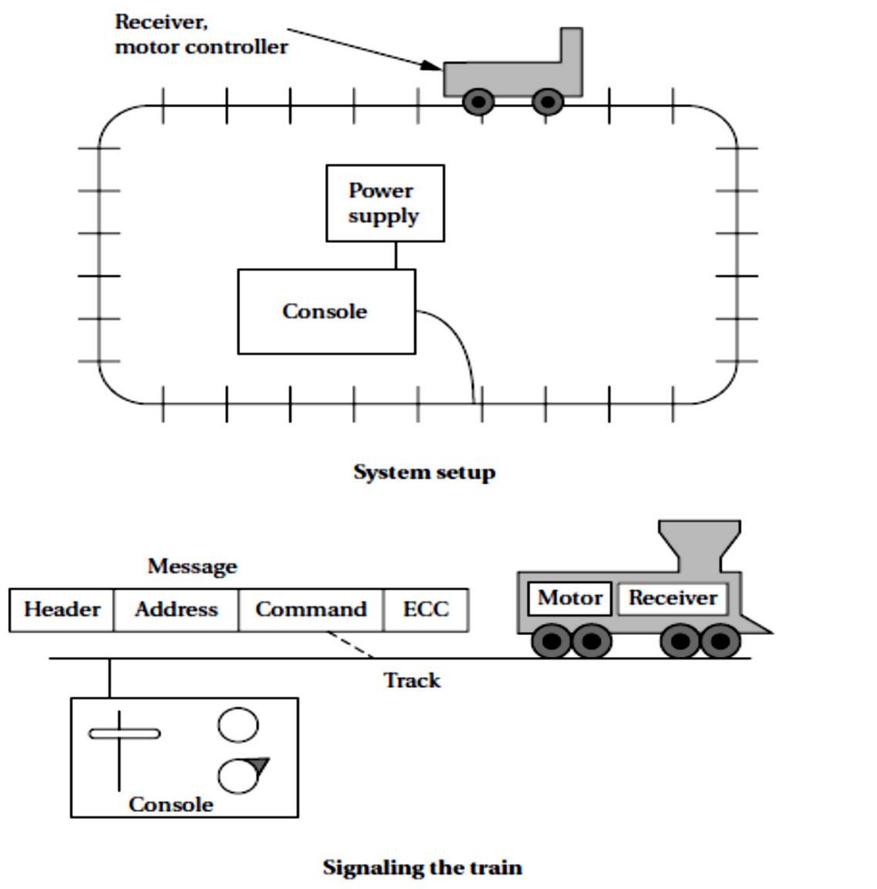


FIGURE 1.14
A model train control system.

A) Requirements

Name	**Model train controller**
Purpose	**Control speed of up to eight 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	**10W (plugs into wall)**

B) DCC

- ✓ The **Digital Command Control (DCC)** standard was created by the **National Model Railroad Association** to support interoperable digitally-controlled model trains. DCC was created to provide a standard that could be built by any manufacturer so that hobbyists could mix and match components from multiple vendors.

The DCC standard is given in two 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 Standard, **defines the packets that carry information.**

We can write the basic packet format as a regular expression:

PSA*(*sD*)* + E

In this regular expression:

■ *P* is the preamble

■ *A* is an address data byte

■ *s*is the data byte start bit, which, like the packet start bit, is a 0.

■ *D* is the data byte, which includes eight bits. A data byte may contain an address, instruction, data, or error correction information.

■ *E* is a packet end bit, which is a 1 bit.

A ***baseline packet*** is the minimum packet that must be accepted by all DCC implementations. More complex packets are given in a Recommended Practice document.

C) Conceptual Specification

- ❖ A ***conceptual specification*** allows us to understand the system a little better.A train control system turns ***commands*** into ***packets***. Commands and packets may not be generated in a 1-to-1 ratio.
- ❖ There are clearly two major subsystems: the command unit and the train-board component as shown in Figure.

Class and UML diagram for Train controller system

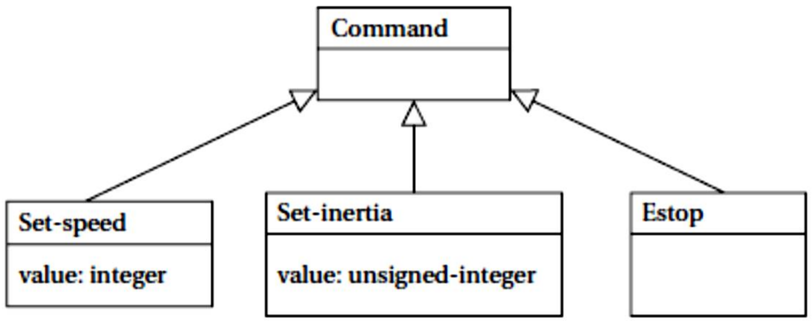


FIGURE 1.16

❖ Class diagram for the train controller messages.

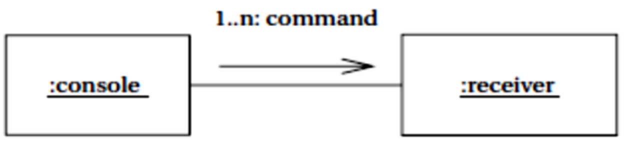


FIGURE 1.17

UML collaboration diagram for major subsystems of the train controller system.

- ❖ A UML class diagram for the train controller showing the composition of the subsystems. Modeling the tracks would help us identify failure modes and possible recovery mechanisms..
  - ❖ The **train receiver must also perform three major functions**:
    - ✓ Receive the message,
    - ✓ Interpret the message (taking into account the current speed, inertia setting, etc.),
    - ✓ Actually control the motor.

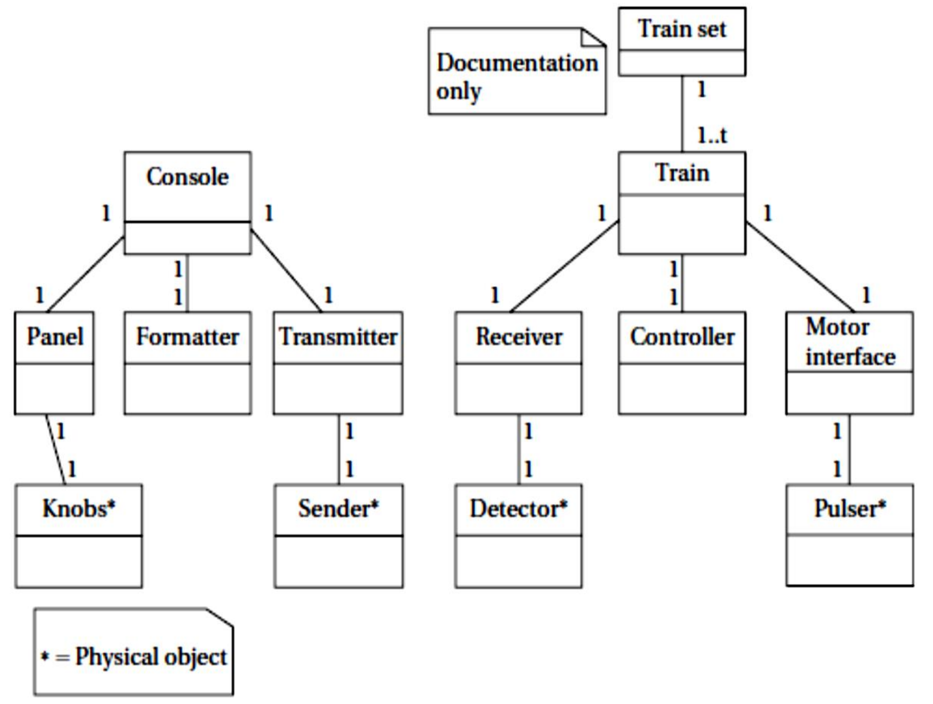


FIGURE 1.18

A UML class diagram for the train controller showing the composition of the subsystems.

■ The ***Console*** class describes the command unit's front panel, which **contains the analog knobs and hardware** to interface to the digital parts of the system.
■ The ***Formatter*** class includes behaviours that know **how to read the panel knobs** and creates a bit stream for the required message.
■ The ***Transmitter*** class interfaces to analog electronics to **send the message** along the track.
■ ***Knobs*** describes the actual analog knobs, buttons, and levers on the control panel.
■ ***Sender*** describes the analog electronics that **send bits along the track**. Likewise, the Train makes use of three other classes that define its components:
■ The ***Receiver*** class knows how to **turn the analog signals on the track into digital form**.
■ The ***Controller*** class includes behaviors that interpret the commands and figures out how to **control the motor**.
■ The ***Motor interface*** class defines how to generate the analog signals required to control the motor. We define two classes to represent analog components:
■ ***Detector* detects analog signals on the track** and converts them into digital form.
■ ***Pulser*** turns **digital commands into the analog signals** required to control the motor speed.

D) Detailed Specification

The basic classes, let's refine it to create a more detailed specification.

- At this point, we need to define the analog components in a little more detail because their characteristics will strongly influence the *Formatter* and *Controller*.
- Figure 1.22 shows a class diagram for these classes; this diagram shows a little more detail than Figure since it includes attributes and behaviours of these classes.
- The *Panel* has three knobs: *train* number (which train is currently being controlled), *speed* (which can be positive or negative), and *inertia*. It also has one button for *emergency-stop*.
- **When we change the train number setting, we also want to reset the other controls to the proper values for that train so that the previous train's control settings are not used to change the current train's settings**.
- To do this, ***Knobs* must provide a *set-knobs*** behaviour that allows the rest of the system to modify the knob settings.
- The motor system takes its motor commands in two parts. The *Sender* and *Detector* classes are relatively simple: They simply put out and pick up a bit, respectively.
- Power is applied in a pulse for a fraction of some fixed interval, with the fraction of the time that power is applied determining the speed.

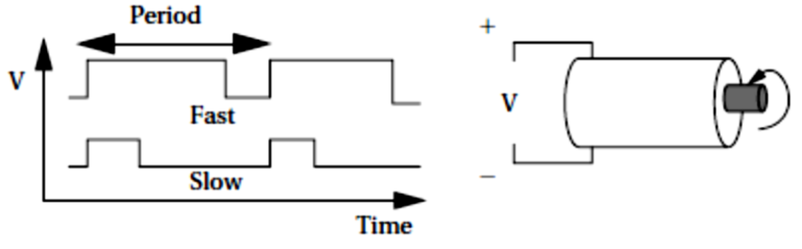


FIGURE 1.20

Controlling motor speed by pulse-width modulation.

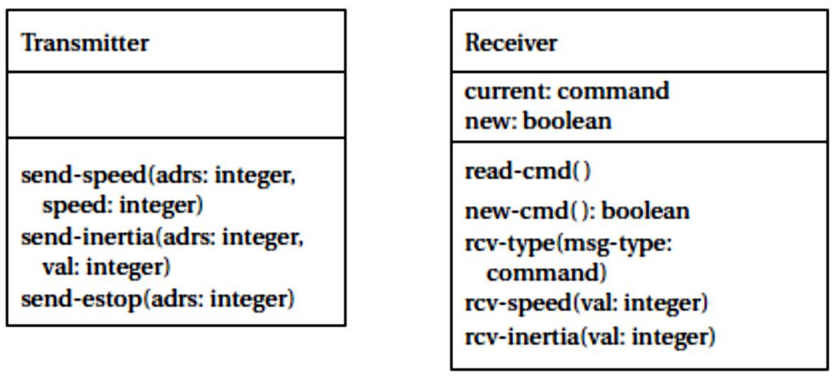

Transmitter
send-speed(adrs: integer, speed: integer)
send-inertia(adrs: integer, val: integer)
send-estop(adrs: integer)
Receiver
current: command
new: boolean
read-cmd( )
new-cmd( ): boolean
rcv-type(msg-type: command)
rcv-speed(val: integer)
rcv-inertia(val: integer)

FIGURE 1.22

Class diagram for the Transmitter and Receiver.

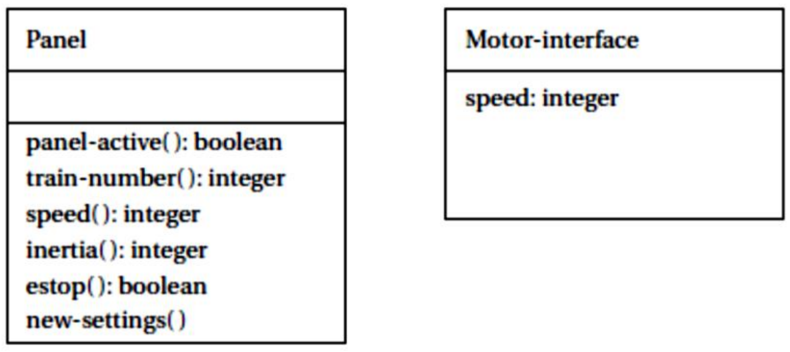


FIGURE 1.21

Class diagram for the Panel and Motor interface.

- ✓ The ***Panel* class defines a behavior for each of the controls on the panel**; we have chosen not to define an internal variable for each control since their values can be read directly from the physical device, but a given implementation may choose to use internal variables.
- ✓ **The *new-settings* behavior uses the *set-knobs* behavior of the *Knobs* class to change the knobs settings** whenever the train number setting is changed.
- ✓ **The *Motor-interface* defines an attribute for speed** that can be set by other classes.
- ✓ The *Transmitter* and *Receiver* classes are shown in Figure 1.22 **they provide the software interface to the physical devices** that send and receive bits along the track.

<u>Sequence diagram for transmitter:</u>

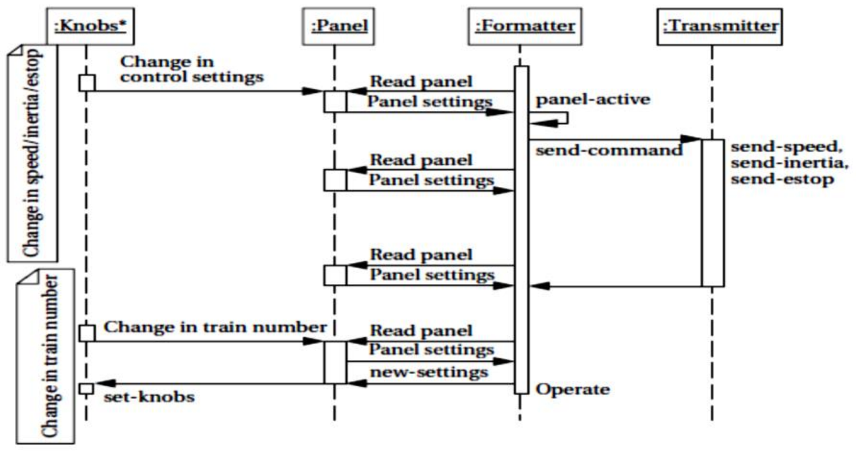


FIGURE 1.24

Sequences diagram for transmitting a control input.

- ✓ The ***Transmitter*** provides a distinct behavior for each type of message that can be sent; it internally takes care of **formatting the message**.
- ✓ The ***Receiver*** class provides a *read-cmd* behavior **to read a message off the tracks**.
- ✓ We do not need a separate behavior for an *Estop* message since it has no parameters—knowing the type of message is sufficient.
- ✓ The role of the formatter during the panel's operation is illustrated by the sequence diagram of Figure 1.24.
- ✓ The figure shows two changes to the knob settings:
- ✓ First to the throttle, inertia, or emergency stop; then to the train number.
- ✓ The panel is called periodically by the formatter to determine if any control settings have changed. If a setting has changed for the current train, the formatter decides to send a command, **issuing a *send-command* behavior to cause the transmitter to send the bits.**
- ✓ Because transmission is serial, it takes a noticeable amount of time for the transmitter to finish a command; in the meantime, the formatter continues to check the panel's control settings. If the train number has changed, the formatter must cause the knob settings to be reset to the proper values for the new train.

<u>Sequence diagram for Receiver</u>

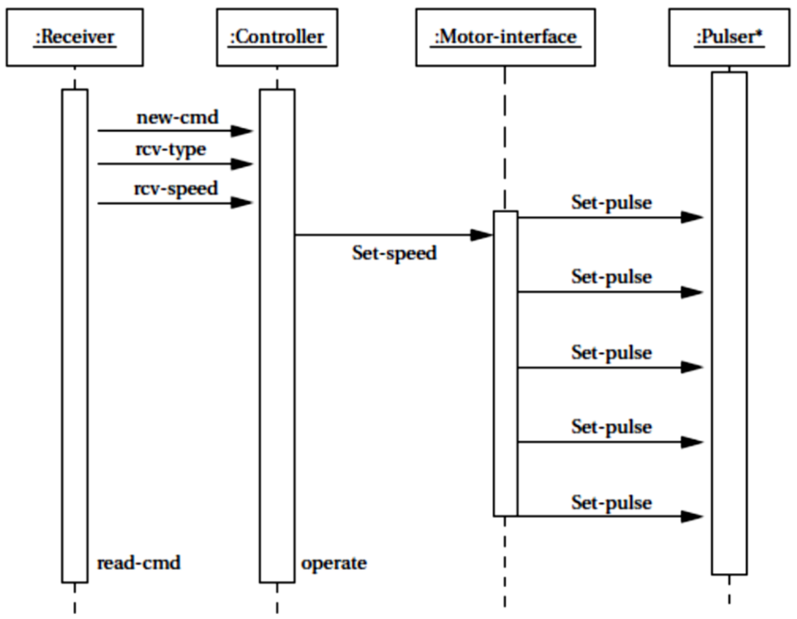


FIGURE 1.29
Sequence diagram for a set-speed command received by the train.

The operation of the *Controller* class during the reception of a *set-speed* command is illustrated in Figure 1.29. The *Controller's operate* behavior must execute several behaviors to determine the nature of the message. Once the speed command has been parsed, it must send a sequence of commands to the motor to smoothly change the train's speed.

5. Discuss the concepts of design methodologies and design flow.

Nov 17(OR)Explain the various design methodologies and design flow in system .(Nov/ Dec 2018)

Design methodologies

Process is important because without it, we can't reliably deliver the products wewant to create. Thinking about the sequence of steps necessary to build something.

Product metrics

A design process has severalimportant goals beyond function, performance, and power:

- *Time-to-market.*
- *Design cost.*
- *Quality.*

Processes evolve over time. They change due to external and internal forces. Customersmay change, requirements change, products change, and available componentschange.

Design flows

A **design flow** is a sequence of steps to be followed during a design. Some of thesteps can be performed by tools, such as compilers or computer-aided design (CAD) systems; other steps can be performed by hand. In this section we look at the basiccharacteristics of design flows.

Waterfall model

Figure 7.1shows the **waterfall model** introduced by Royce [Dav90], the first modelproposed for the software development process. The waterfall development model consistsof five major phases: *requirements* analysis determines the basic characteristics ofthe system; *architecture*design decomposes the functionality into major components;*coding* implements the pieces and integrates them; *testing* uncovers bugs; and *maintenance*entails deployment in the field, bug fixes, and upgrades.

.

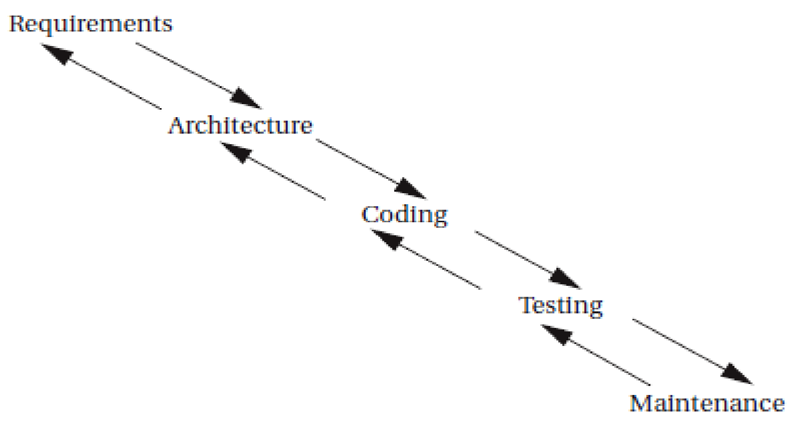


FIGURE 7.1

The waterfall model of software development.

Spiral model

- ✓ Figure 7.2illustrates an alternative model of software development, called the **spiralmodel.**
- ✓ While the waterfall model assumes that the system is builtonce in its entirety, the spiral model assumes that several versions of the system will be built.
- ✓ Early systems will be simple mock-ups constructed to aid designers 'instuition and to build experience with the system. As design progresses, more complexsystems will be constructed.
- ✓ At each level of design, the designers go throughrequirements, construction, and testing phases. At later stages when more completeversions of the system are constructed, each phase requires more work, widening thedesign spiral. This successive refinement approach helps the designers understandthe system they are working on through a series of design cycles.

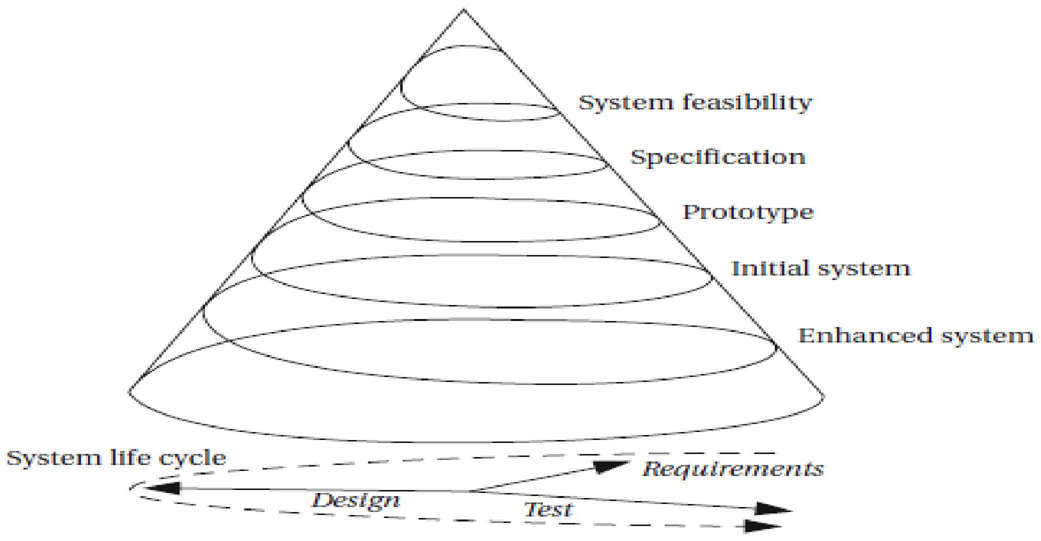


FIGURE 7.2

The spiral model of software design.

Successive refinement

- ✓ Figure 7.3shows a **successive refinement** design methodology. In this approach,the system is built several times. A first system is used as a rough prototype, andsuccessive models of the system are furtherrefined.
- ✓ Refining the system by building several increasingly complexsystems allows you to test out architecture and design techniques.
- ✓ The variousiterations may also be only partially completed; for example, continuing an initialsystem only through the detailed design phase may teach you enough to helpyou avoid many mistakes in a second design iteration that is carried through tocompletion.

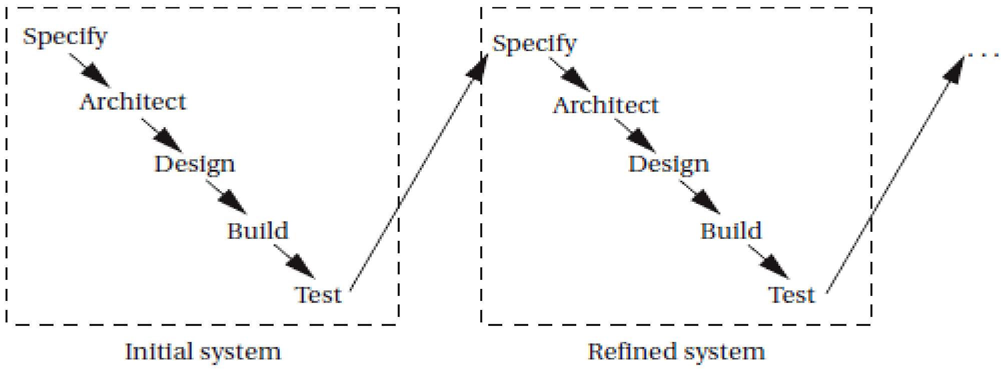


FIGURE 7.3

A successive refinement development model.

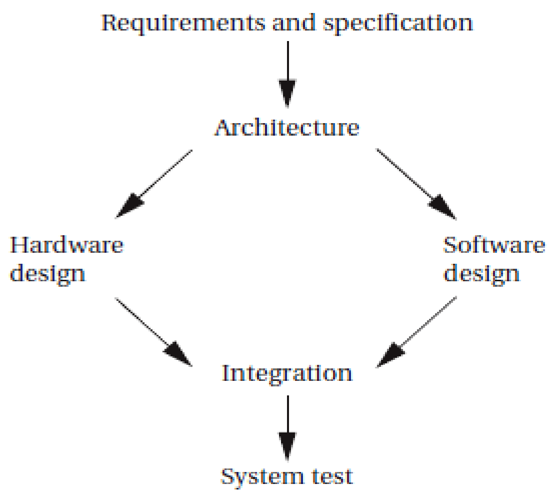


FIGURE 7.4

A simple hardware/software design methodology.

Hierarchical design flows

The design flow follows thelevels of abstraction in the system, from complete system design flows at the mostabstract to design flows for individual components. The design flow for these complexsystemsresembles the flow shown in Figure 7.5.

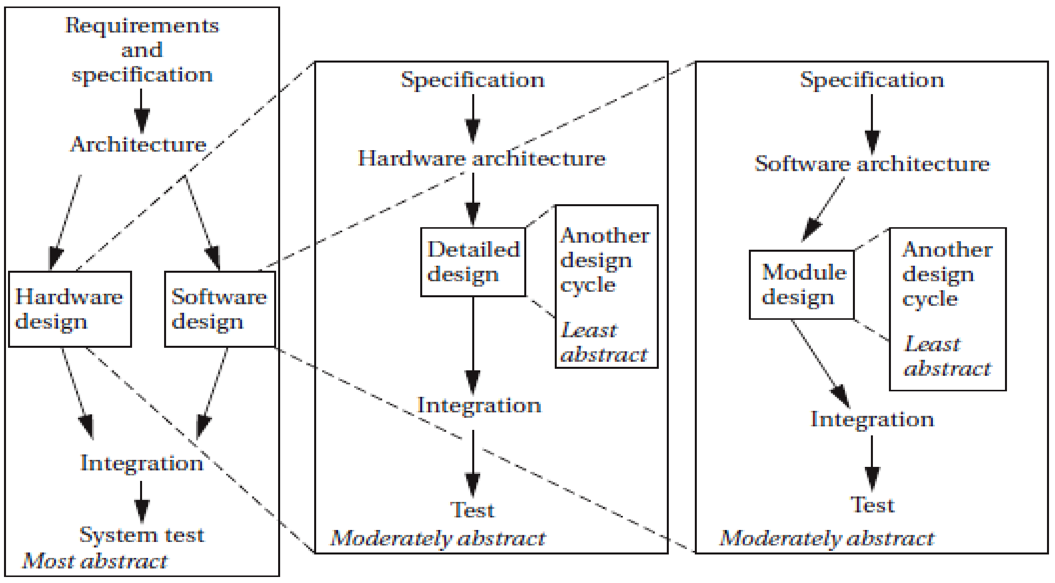


FIGURE 7.5

A hierarchical design flow for an embedded system.

Concurrent engineering

Concurrent engineering attempts to take a broader approach and optimize the total flow.

6.Discuss the importance of requirement analysis for system design

Nov 17

Requirements are informal descriptions of what the customer wants, while **specifications** are more detailed, precise, and consistent descriptions of the system that can be used to create the architecture.

Functional and nonfunctional requirements

We have two types of requirements: **functional** and **nonfunctional.**

A functionalrequirement states what the system must do, such as compute an FFT. A nonfunctionalrequirement can be any number of other attributes, including physicalsize, cost, power consumption, design time, reliability, and so on.A good set of requirements should meet several tests

A good set of requirements should meet several tests:

- **Correctness**:
- **Unambiguousness**
- **Completeness**
- **Verifiability**
- **Consistency**
- **Modifiability**
- **Traceability**:

Each requirement should be traceable in the following ways:

- We should be able to **trace backward** from the requirements to know why eachrequirement exists.
- We should also be able to **trace forward** from documents created before the requirements (e.g., marketing memos) to understand how they relate to thefinal requirements.

Direct customer contact gives the designer an unfiltered sample of what the customer says.

7. Describe in detail about the quality assurance techniques (Apr/May 16) (or) observe in detail about quality assurance techniques and verifying its specifications. Apr/MAY 18, NOV/DEC 17

Quality assurance

❖ The quality of a product or service can be judged by how well it satisfies its intendedfunction.

- ❖ A product can be of low quality for several reasons, such as it was shoddilymanufactured, its components were improperly designed, its architecture was poorlyconceived, and the product's requirements were poorly understood. Quality must bedesigned in.
- ❖ You can't test out enough bugs to deliver a high-quality product. The**quality assurance (QA)** process is vital for the delivery of a satisfactory system

Quality assurance techniques

- o The International Standards Organization (ISO) has created a set of quality standardsknown as **ISO 9000.**
- o ISO 9000 was created to apply to a broad range of industries,including but not limited to embeddedhardware and software. .
- o Consequently, ISO 9000 concentrates on processes used to create theproduct or service.
- o The processes used to satisfy ISO 9000 affect the entire organizationas well as the individual steps taken during design and manufacturing.

A detailed description of ISO 9000 is beyond the scope of this book; severalbooks describe ISO 9000's applicability to software development. We can, however, make the following observations about quality management based onISO 9000:

- o *Process is crucial.*
- o *Documentation is important.*
- o *Communication is important.*
- o Many types of techniques can be used to verify system designs and ensure quality.Techniques can be either *manual* or *tool based*. Manual techniques are surprisinglyeffective in practice.

Tool-basedverification helps considerably in managing large quantities of information that maybe generated in a complex design.

Test generation programs can automate much of thedrudgery of creating test sets for programs. Tracking tools can help ensure that varioussteps have been performed. Design flow tools automate the process of running designdata through other tools.

Metrics are important to the quality control process.

To know whether we haveachieved high levels of quality, we must be able to measure aspects of the system andour design process

Tool and manual techniques must fit into an overall process. The details of thatprocess will be determined by several factors, including the type of product beingdesigned (e.g., video game, laser printer, air traffic control system), the number ofunits to be manufactured and the time allowed for design, the existing practices in thecompany into which any new processes must be integrated, and many other factors.

Animportant role of ISO 9000 is to help organizations study their total process, not justparticular segments that may appear to be important at a particular time.

One well-known way of measuring the quality of an organization's software developmentprocessis the **Capability Maturity Model (CMM)** developed by Carnegie Mellon University's Software Engineering Institute [SEI99]. The CMM provides amodel for judging an organization. It defines the following five levels of maturity:

1. Initial.
2. *Repeatable.*
3. *Defined.*
4. *Managed.*
5. *Optimizing.*

Verifying the specification

The requirements and specification are generated very early in the design process.

Verifying the requirementsand specification is very important for the simple reasonthat bugs in the requirements or specification can be extremely expensive to fix lateron.

Figure 7.11shows how the cost of fixing bugs grows over the course of the designprocess. The longer a bug survives in the system, the more expensive it will beto fix.

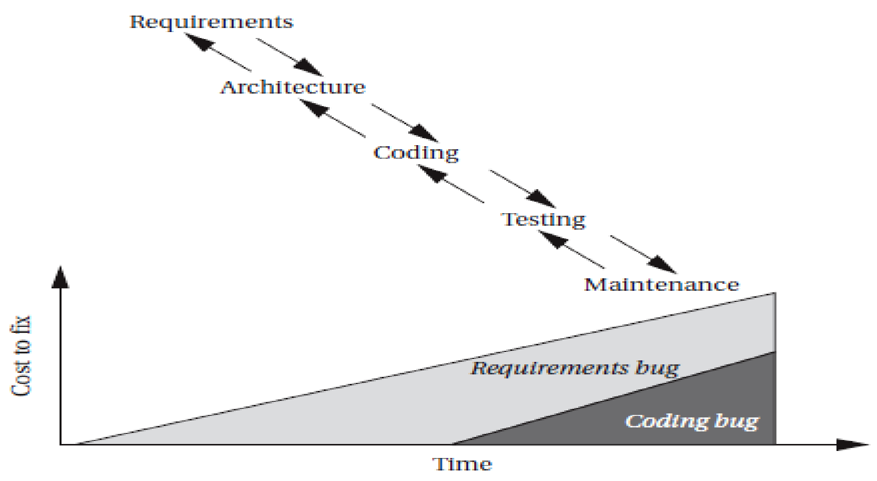


FIGURE 7.11
Long-lived bugs are more expensive to fix.

.

Requirements validation

- **Prototypes** are a very useful tool when dealing withend users—rather than simply describe the system to them in broad, technical terms, aprototype can let them see, hear, and touch at least some of the important aspects of thesystem. Of course, the prototype will not be fully functional because the design workhas not yet been done.

- Certain programming languages, sometimes called**prototyping languages** or **specification languages,** are especially well suited to prototyping.
- **Preexistingsystems** can also be used to help the end user articulate his or her needs.

Validation of specifications

The techniques used to validate requirements are also useful in verifying that the specifications are correct. Building prototypes, specification languages, and comparisons to preexisting systems are as useful to system analysis and designers as they are to end users. Auditing tools may be useful in verifying consistency, completeness, and so forth.

Design reviews

The **design review** is a critical component of any quality assurance process. The design review is a simple, low-cost way to catch bugs early in the design process.

Design review format

A design review is held to review a particular component of the system. A design review team has several types of members:

• The *designers*
• The *review leader*
• The *review scribe*
• The *review audience*

• *Cross-functional teams*
• *Concurrent product realization*
• *Incremental information sharing*
• *Integrated project management*
• *Early and continual supplier involvement*
• *Early and continual customer focus*

8. Explain the Memory Devices? (or) describe the basic types of memory components commonly used in embedded systems? [MAY/JUNE 2014] Apr/May 16, Apr18, Nov 18

Subtopics

- ✓ **Memory Device Organization**
- ✓ **Random-Access Memories (RAM)**
- ✓ **Read-Only Memories**

a) Memory Device Organization

- ✓ The most basic way to characterize a memory is by its capacity, such as 256 MB.
- ✓ However, manufacturers usually make several versions of a memory of a given size, each with a different data width. For example, a 256-MB memory may be available in two versions:

■ **As a 64M _ 4-bit array**, a single memory access obtains an 8-bit data item, with a maximum of 226 different addresses.

■ **As a 32 M_ 8-bit array**, a single memory access obtains a 1-bit data item, with a maximum of 223 different addresses.

The height/width ratio of a memory is known as its ***aspect ratio***.

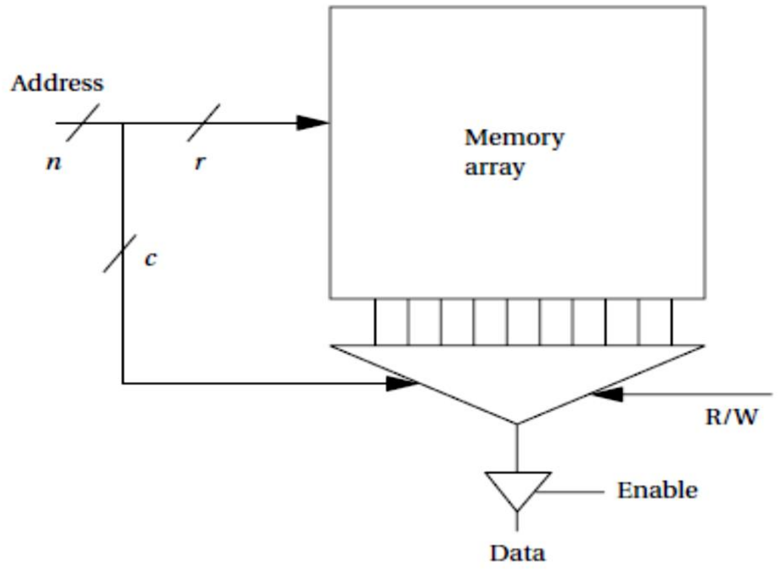


FIGURE 4.15

Internal organization of a memory device.

b) Random-Access Memories (RAM)

- *Random-access memories can be both read and written*. T
- hey are called random access because, unlike magnetic disks, **addresses can be read in any order.** Most bulk memory in modern systems is ***dynamic RAM (DRAM)***.
- DRAM is very dense; it does, however, require that its values be **refreshed** periodically since the values inside *the memory cells decay over time.*
- The dominant form of dynamic RAM today is the ***synchronous DRAMs(SDRAMs)***, which uses clocks to improve DRAM performance.
- SDRAMs use **Row Address Select (RAS) and Column Address Select (CAS) signals** to break the address into two parts, which **select the proper row and column** in the RAM array. Signal transitions are relative to the SDRAM clock, which allows the internal SDRAM operations to be pipelined.

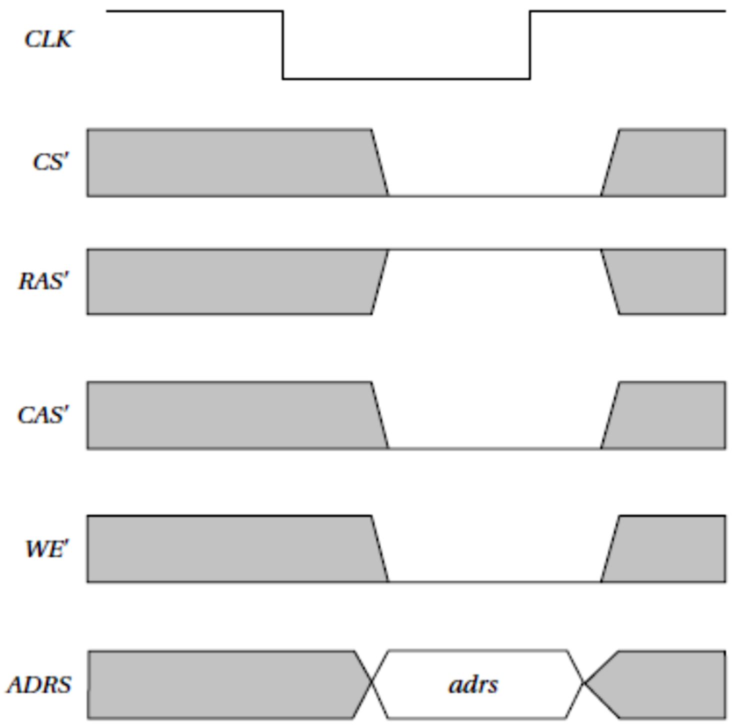


FIGURE 4.16

Timing diagram for a read on a synchronous DRAM.

- SDRAMs generally also support an interleaved mode that exchanges pairs of bytes. Even faster synchronous DRAMs, known as ***double-data rate (DDR)*** SDRAMs or **DDR2** and **DDR3** SDRAMs, are now in use.
- Memory for PCs is generally purchased as ***single in-line memory modules (SIMMs)*** or ***double in-line memory modules (DIMMs)***.

c) Read-Only Memories

- ***Read-only memories (ROMs)*** are pre programmed with fixed data.
- They are very useful in embedded systems since a great deal of the code, and perhaps some data, ***does not change over time.***
- Read-only memories are also less sensitive to radiation induced errors.
- There are several varieties of ROM available. The first-level distinction to be made is between ***factory-programmed ROM*** (sometimes called ***mask-programmedROM***) and ***field-programmable ROM***. Factory-programmed ROMs are ordered from the factory with particular programming.

- ❖ ROMs can typically be ordered in lots of a few thousand, but clearly factory programming is useful only when the ROMs are to be installed in some quantity. Field-programmable ROMs, on the other hand, can be programmed in the lab. ***Flash memory*** is the dominant form of field-programmable ROM and is electrically erasable.
- ❖ Flash memory uses standard system voltage for erasing and programming, allowing it to be reprogrammed inside a typical system.
- ❖ As a result, this form of flash is commonly known as ***boot-block flash***. Similar to those of single-rate SDRAMs; DDRs simply use sophisticated circuit techniques to perform more operations per clock cycle.

9. How embedded systems are designed with computing platforms? Apr/May 16 (NoV/Dec16) Nov 17(or) Compare and contrast debugging techniques used in embedded systems. Nov 18

Subtopics

- ✓ ***Example platforms***
- ✓ ***Choosing a platform***
- ✓ ***Intellectual property***
- ✓ ***Development environments***
- ✓ ***Debugging techniques***
- ✓ ***Debugging challenges***

1. Example platforms

- ❖ The design complexity of the hardware platform can vary greatly, from a totally offthe-shelf solution to a highly customized design.
- ❖ Chip vendors often provide their **own *evaluation boards or evaluation modulesfor their chips*.** The evaluation board may be a complete solution or provide what youneed with only slight modifications.

2. Choosing a platform

- ❖ We probably will not design the platform for our embedded system from scratch. We may assemble hardware and software components from several sources; we may also acquire a complete hardware/software platform package.
- ❖ **The various components may all playa factor in the suitability of the platform**.
    - ✓ *CPU,*
    - ✓ *Bus,*
    - ✓ *Memory and*

- ✓ *Input and output devices*

❖ When we think about software components of the platform, we generally thinkabout both the run-time components and the support components.

❖ Run-time componentsbecome part of the final system:

- ✓ *The operating system,*
- ✓ *Code libraries, and so on.*

<u>3. Intellectual property</u>

❖ **Intellectual property (IP)** is something that **we can own but not touch**:

- ✓ ***software,***
- ✓ ***netlists, and so on.***

❖ Just as we need to acquire hardware components to build our system, we also need to acquire intellectual property to make that hardware useful.

Examples of the wide range of **Intellectual property** that we use in embedded system design:

- ***run-time software libraries;***
- ***software development environments;***
- ***schematics, netlists, and other hardware design information.***

<u>4. Development environments</u>

❖ Although we may use an evaluation board, much of the software development for an**embedded system is done on a PC or workstation known as a*host***as illustrated inFigure 4.23.

❖ **The hardware on which the code will finally run** is known as the **target.**

❖ The host and target are frequently **connected by a USB link**, but a higher-speed linksuch as Ethernet can also be used.

❖ The **target** must include a **small amount of software to talk to the host system**. Thatsoftware will take up some memory, interrupt vectors, and so on, but it should generallyleave the smallest possible footprint in the target to avoid interfering with the applicationsoftware. **The host should be able to do the following:**

- ***load programs into the target;***
- ***start and stop program execution on the target; and***
- ***examine memory and CPU registers.***

❖ A **cross-compiler** is a compiler that **runs on one type of machine but generatescode for another.** After compilation, the executable code is typically downloaded to theembedded system by USB.

❖ We often create a **testbench program** that can be built to help debug embeddedcode. The testbench generates inputs to stimulate a piece of code and compares theoutputs against expected values, providing valuable early debugging help.

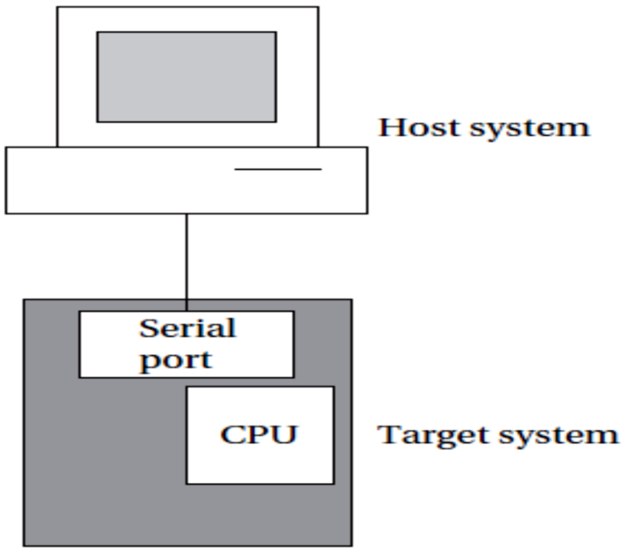


FIGURE 4.23
Connecting a host and target system.

❖

5. Debugging techniques

❖ Another very important debugging tool is the **breakpoint.** The simplest form of a break point is for the user to specify an address at which the program's execution is to break.
❖ Implementing breakpoints does not require using exceptions or external devices.
❖ When software tools are insufficient to debug the system, hardware aids can be deployed to give a clearer view of what is happening when the system is running.
Debugging techniques are:
- ✓ ***Microprocessor in-circuit emulator (ICE)***
- ✓ ***Logic analyzer***

❖ The ***microprocessor in-circuit emulator (ICE)***is a specialized hardware tool that can help **debug software in a working embedded system**. At the heart of an in-circuitemulator is a special version of the microprocessor that allows its internal registersto be read out when it is stopped.
❖ The**main drawback** to in-circuit emulation is that the machine is specific to a particularmicroprocessor, even down to the pinout. If you use several microprocessors, maintaininga fleet of in-circuit emulators to match can be **very expensive**.

- ❖ The **logic analyzer** is the other major piece of instrumentation in theembedded system designer's arsenal. Think of a logic analyzer as an array of inexpensiveoscilloscopes—the analyzer can sample many different signals simultaneously(tens to hundreds) but can display **only 0, 1, or changing values for each**.
- ❖ A typical logic analyzer can acquire data in either of two modes that are typicallycalled **state** and **timing modes.**
- ❖ The internal architecture of a logic analyzer is shown in Figure 4.24.

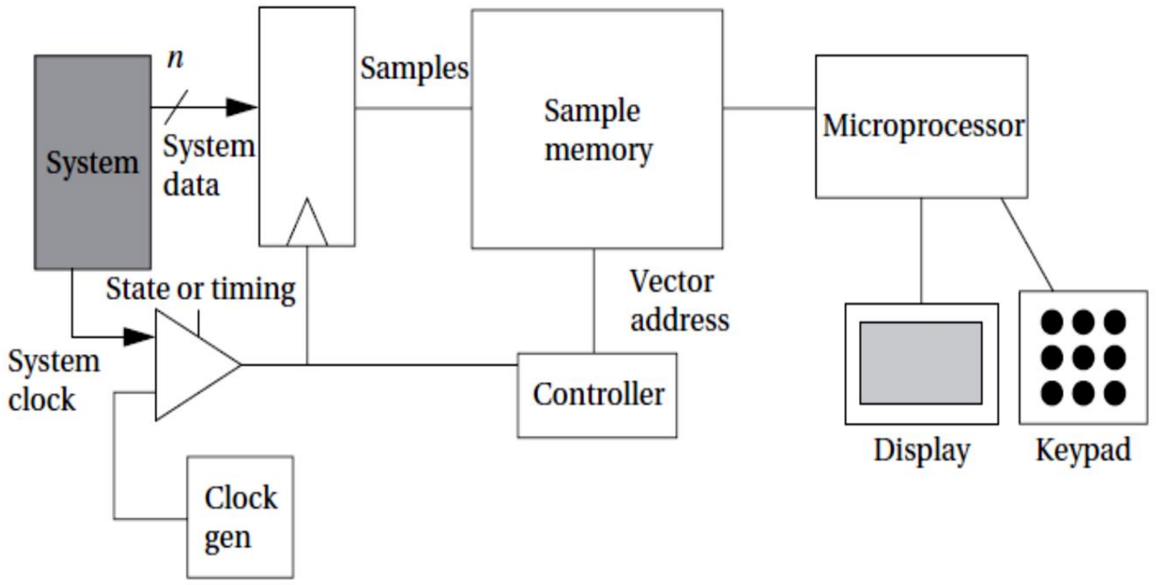


FIGURE 4.24

Architecture of a logic analyzer.

❖

6. Debugging challenges

- ❖ Logical errors in **software can be hard to track down**, but errors in real-time code cancreate problems that are even harder to diagnose.
- ❖ Real-time programs are required to **finish their work within a certain amount of time**; if they run too long, they can creat every unexpected behavior.

10. Explain how Consumer electronics architecture are helpful in designing embedded systems Nov 17

1. Consumer electronics use cases and requirements

- ❖ Consumer electronics deviceshave converged over the past decade around a set of common features that are supportedby common architectural features.
- ❖ Not all devices have all features, dependingon the way the device is to be used, but most devices select features from a commonmenu.
- ❖ Similarly, there is no single platform for consumer electronics devices, but thearchitectures in use are organized around some common themes.
- ❖ This convergence is possible because these devices implement a few basic types offunctions in variouscombinations:
  - ✓ *Multimedia and*
  - ✓ *Communications.*
- ❖ The style of multimediaor communicationsmay vary, and **different devices may use different formats,**but this causes variations in hardwareand software components within the basic architecturaltemplates. Consumer electronics devices provide several types of services in differentcombinations:
- ❖ **Multimedia:** The media may be audio, still images, or video.
- ❖ **Data storage and management:** Because people want to select what multimediaobjects they **save or play, data storage goes hand-in-hand** with multimediacapture and display.
- ❖ **Communications:** Communications may be relatively simple, such as a USBinterface to a host computer. The communications link may also be moresophisticated, **such as an Ethernet port or a cellular telephone link.**

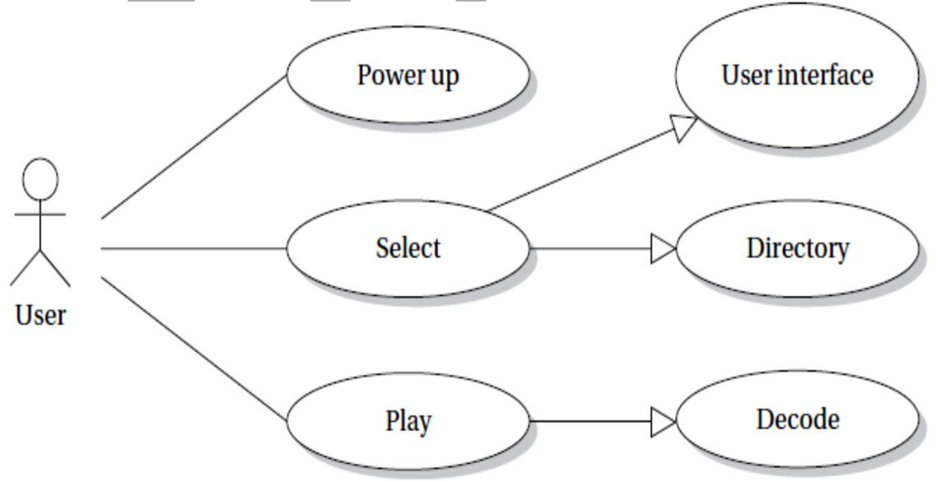


FIGURE 4.25

Use case for playing multimedia.

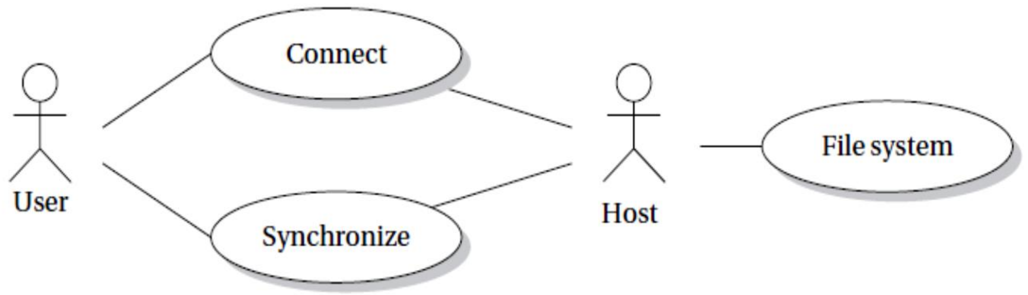


FIGURE 4.26

Use case of synchronizing with a host system.

- ❖ Figure 4.27 shows a **functional block diagram of a typical device**. The storage system provides bulk, permanent storage. The network interface may provide a simpleUSB connection or a full-blown Internet connection.
- ❖ Multiprocessor architectures are common in many consumer multimedia devices.
- ❖ Figure 4.27 shows a **two-processor architecture**; if more computation is required, more**DSPs and CPUs may be added.**

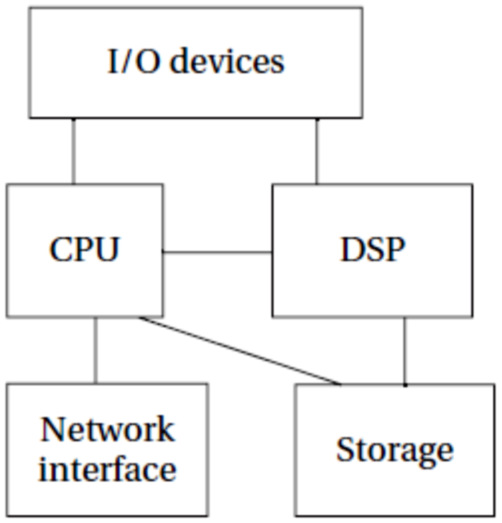


FIGURE 4.27

Hardware architecture of a generic consumer electronics device.

2. File systems

- ❖ **DOS file allocation table** (**FAT**) file systems refer to the file system developed byMicrosoft for early versions of the **DOS operating system.**

- ❖ **FAT** can be implemented on flash storage devices as well as **magnetic disks; wear-leveling algorithms for flash memory** can be implemented without disturbing the basic operation of the file system.
- ❖ Many consumer electronics devices use **flash memory** for mass storage.
- ❖ Flash memory has one important limitation that must be taken into account. Writing a flash memory cell causes mechanical stress that eventually wears out the cell.
- ❖ A wear-leveling flash file system manages the use of flash memory locations to equalize wear while maintaining compatibility with existing file systems.
- ❖ **A simple model of a standard file system has two layers:**
  - ✓ **the bottom layer handles physical reads and writes on the storage device;**
  - ✓ **the top layer provides a logical view of the file system .**

7. Explain Program-Level Energy And Power Analysis And Optimization? Apr/May16 (Apr /may 17) (NoV/Dec 16)(Nov/Dec 18)

Power consumption is a particularly important design metric for battery-powered systems because the battery has a very limited lifetime. However, power consumption is increasingly important in systems that run off the power grid.

Example

■ We may be able to replace the algorithms with others that do things in clever ways that consume less power.

■ Memory accesses are a major component of power consumption in many applications. By optimizing memory accesses we may be able to significantly reduce power.

■ We may be able to turn off parts of the system—such as subsystems of the CPU, chips in the system, and so on—when we do not need them in order to save power.

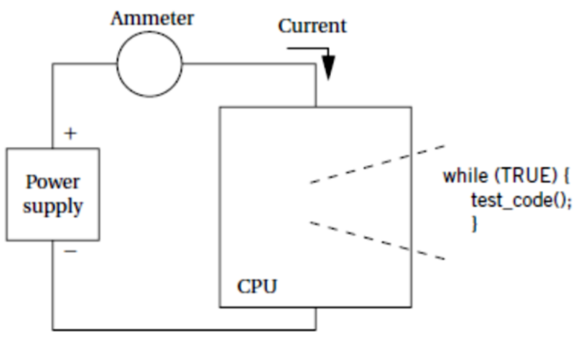


FIGURE 5.24
Measuring energy consumption for a piece of code.

Several factors contribute to the energy consumption of the program.

■ **Energy consumption varies somewhat from instruction to instruction.**

■ **The sequence of instructions has some influence.**
■ **Theopcode and the locations of the operands also matter.**

❖ Choosing which instructions to use can make some difference in a program's energy consumption, but concentrating on the instruction opcodes has limited payoffs in most CPUs. The program has to do a certain amount of computation to perform its function.
❖ While there may be some clever ways to perform that computation, the energy cost of the basic computation will change only a fairly small amount compared to the total system energy consumption, and usually only after a great deal of effort.
❖ We are further hampered in our ability to optimize instruction level energy consumption because **most manufacturers do not provide detailed**, instruction-level energy consumption figures for their processors.
❖ **In many applications, the biggest payoff in energy reduction** for a given amount of designer effort comes from concentrating on the memory system.
❖ **Accesses to registers are the most energy efficient**; cache accesses are more energy efficient than main memory accesses. Caches are an important factor in energy consumption.
❖ On the one hand, a cache hit saves a costly main memory access, and on the other, the cache itself is relatively power hungry because it is built from SRAM, not DRAM. If we can control the size of the cache, we want to choose the smallest cache that provides us with the necessary performance.
❖ **If the cache is too small, the program runs slowly** and the system consumes a lot of power due to the high cost of main memory accesses.
❖ **If the cache is too large, the power consumption is high** without a corresponding payoff in performance. At intermediate values, the execution time and power consumption are both good.?
❖ The best overall advice is that *high performance = low power*.
❖ If the program can be modified to reduce instruction or data cache conflicts, for example, the energy required by the memory system can be significantly reduced. The effectiveness of changes such as reordering instructions or selecting different instructions depends on the processor involved, but they are generally less effective than cache optimizations.
❖ A few optimizations mentioned previously for performance are also often useful for improving energy consumption:

■ ***Try to use registers efficiently.***
■ ***Analyze cache behavior to find major cache conflicts.***
■ ***Make use of page mode accesses in the memory system whenever possible***