MODULE VI

Networks

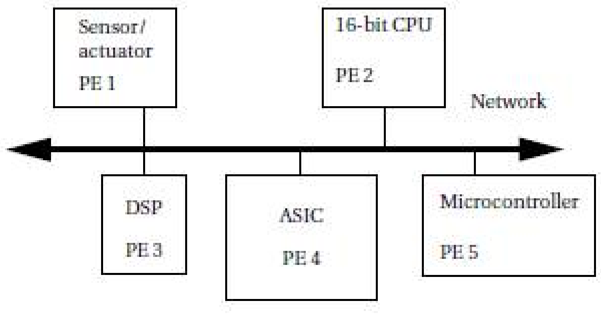
In a distributed embedded system, several processing elements (PEs) (either microprocessors or ASICs) are connected by a network that allows them to communicate. The application is distributed over the PEs, and some of the work is done at each node in the network.

Reasons to build network-based embedded systems.

- When the processing tasks are physically distributed, it may be necessary to put some of the computing power near where the events occur.
- Data reduction is another important reason for distributed processing. It may be possible to perform some initial signal processing on captured data to reduce its volume
- Modularity is another motivation for network-based design. For instance, when a large system is assembled out of existing components, those components may use a network port as a clean interface that does not interfere with the internal operation of the component in ways that using the microprocessor bus would.
- A distributed system can also be easier to debug the microprocessors in one part of the network can be used to probe components in another part of the network. Finally, in some cases, networks are used to build fault tolerance into systems.
- Distributed embedded system design is another example of hardware/software co-design, since we must design the network topology as well as the software running on the network.

DISTRIBUTED EMBEDDED ARCHITECTURE

A distributed embedded system can be organized in many different ways, but its basic units are the PE and the network as illustrated in Figure.



A PE may be an instruction set processor such as a DSP, CPU, or microcontroller, as well as a nonprogrammable unit such as the ASICs used to implement PE 4. An I/O device such as PE 1 (which we call here a sensor or actuator, depending on whether it provides input or output) may also be a PE, so long as it can speak the network protocol to communicate with other PEs. The network in this case is a bus, but other network topologies are also possible. It is also possible that the system can use more than one network, such as when relatively independent functions require relatively little communication among them. We often refer to the connection between PEs provided by the network as a **communication link**. The system of PEs and networks forms the hardware platform on which the application runs.

Why Distributed?

In some cases, distributed systems are necessary because the devices that the PEs communicate with are physically separated. If the deadlines for processing the data are short, it may be more cost-effective to put the PEs where the data are located rather than build a higher-speed network to carry the data to a distant, fast PE.

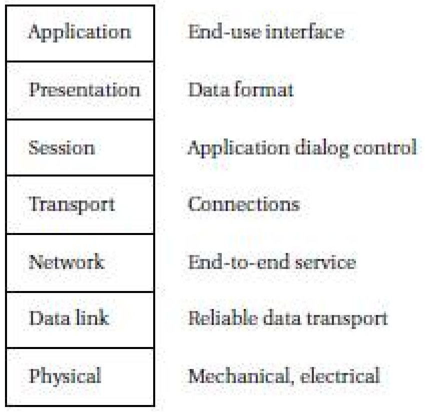
An important advantage of a distributed system with several CPUs is that one part of the system can be used to help diagnose problems in another part. Whether you are debugging a prototype or diagnosing a problem in the field, isolating the error to one part of the system can be difficult when everything is done on a single CPU. If you have several CPUs in the system, you can use one to generate inputs for another and to watch its output.

Network Abstractions

Networks are complex systems. Ideally, they provide high-level services while hiding many of the details of data transmission from the other components in the system. In order to help understand (and design) networks, the International Standards Organization has developed a seven-layer model for networks known as Open Systems Interconnection (OSI) models.

The seven layers of the OSI model, shown in Figure, are intended to cover a broad spectrum of networks and their uses. Some networks may not need the services of one or more layers because the higher layers may be totally missing or an intermediate layer may not be necessary. However, any data network should fit into the OSI model. The OSI layers from lowest to highest level of abstraction are described below.

- Physical: The physical layer defines the basic properties of the interface between systems, including the physical connections (plugs and wires), electrical properties, basic functions of the electrical and physical components, and the basic procedures for exchanging bits.



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

- Data link: The primary purpose of this layer is error detection and control across a single link. However, if the network requires multiple hops over several data links, the data link layer does not define the mechanism for data integrity between hops, but only within a single hop.
- Network: This layer defines the basic end-to-end data transmission service. The network layer is particularly important in multihop networks.
- Transport: The transport layer defines connection-oriented services that ensure that data are delivered in the proper order and without errors across multiple links. This layer may also try to optimize network resource utilization.
- Session: A session provides mechanisms for controlling the interaction of end-user services across a network, such as data grouping and checkpointing.
- Presentation: This layer defines data exchange formats and provides transformation utilities to application programs.
- Application: The application layer provides the application interface between the network and end users.

Hardware and Software Architectures

Distributed embedded systems can be organized in many different ways depending upon the needs of the application and cost constraints. One good way to understand possible architectures is to consider the different types of interconnection networks that can be used.

A point-to-point link establishes a connection between exactly two PEs. Point-to-point links are simple to design precisely because they deal with only two components. We do not have to worry about other PEs interfering with communication on the link.

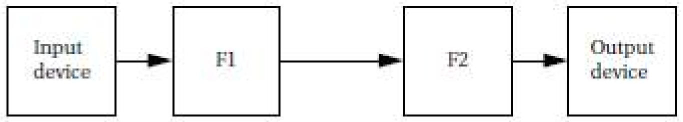
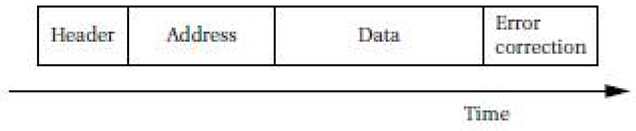


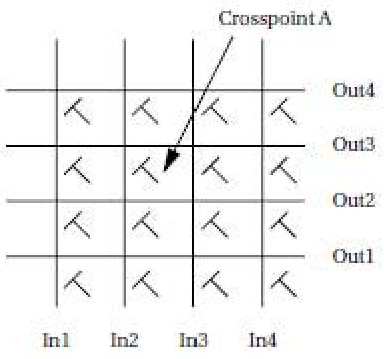
Figure shows a simple example of a distributed embedded system built from point-to-point links. The input signal is sampled by the input device and passed to the first digital filter, F1, over a point-to-point link. The results of that filter are sent through a second point-to-point link to filter F2. The results in turn are sent to the output device over a third point-to-point link. A digital filtering system requires that its outputs arrive at strict intervals, which means that the filters must process their inputs in a timely fashion. Using point-to-point connections allows both F1 and F2 to receive a new sample and send a new output at the same time without worrying about collisions on the communications network.

A bus is a more general form of network since it allows multiple devices to be connected to it. Like a microprocessor bus, PEs connected to the bus have addresses. Communications on the bus generally take the form of packets as illustrated in Figure. A packet contains an address for the destination and the data to be delivered. It frequently includes error detection/correction information such as parity. It also may include bits that serve to signal to other PEs that the bus is in use, such as the header shown in the figure. The data to be transmitted from one PE to another may not fit exactly into the size of the data payload on the packet. It is the responsibility of the transmitting PE to divide its data into packets; the receiving PE must of course reassemble the complete data message from the packets.



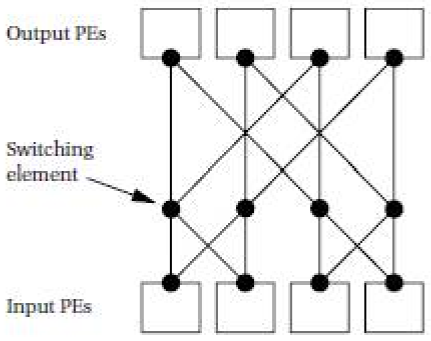
Distributed system buses must be arbitrated to control simultaneous access, just as with microprocessor buses. Arbitration scheme types are summarized below.

- Fixed-priority arbitration always gives priority to competing devices in the same way. If a high-priority and a low-priority device both have long data transmissions ready at the same time, it is quite possible that the low-priority device will not be able to transmit anything until the high-priority device has sent all its data packets.
- Fair arbitration schemes make sure that no device is starved. Round-robin arbitration is the most commonly used of the fair arbitration schemes. The PCI bus requires that the arbitration scheme used on the bus must be fair, although it does not specify a particular arbitration scheme. Most implementations of PCI use round-robin arbitration. A bus has limited available bandwidth. Since all devices connect to the bus, communications can interfere with each other. Other network topologies can be used to reduce communication conflicts. At the opposite end of the generality spectrum from the bus is the crossbar network shown in Figure.



A crossbar not only allows any input to be connected to any output, it also allows all combinations of input/output connections to be made. Thus, for example, we can simultaneously connect in1 to out4, in2 to out3, in3 to out2, and in4 to out1 or any other combinations of inputs. (Multicast connections can also be made from one input to several outputs.) A crosspoint is a switch that connects an input to an output. To connect an input to an output, we activate the crosspoint at the intersection between the corresponding input and output lines in the crossbar. For example, to connect in2 and out3 in the figure, we would activate crossbar A as shown. The major drawback of the crossbar network is expense: The size of the network grows as the square of the number of inputs (assuming the numbers of inputs and outputs are equal).

Many other networks have been designed that provide varying amounts of parallel communication at varying hardware costs. Below given figure shows an example multistage network. The crossbar of above figure is a direct network in which messages go from source to destination without going through any memory element. Multistage networks have intermediate routing nodes to guide the data packets.



Message Passing Programming

Distributed embedded systems do not have shared memory, so they must communicate by passing messages. We will refer to a message as the natural communication unit of an algorithm; in general, a message must be broken up into packets to be sent on the network. A procedural interface for sending a packet might look like the following:

send_packet(address,data);

The routine should return a value to indicate whether the message was sent successfully if the network includes a handshaking protocol. If the message to be sent is longer than a packet,it must be broken up into packet-size data segments as follows:

```
for (i = 0; i < message.length;i=i+ PACKET_SIZE)
    send_packet(address,&message.data[i]);
```

The above code uses a loop to break up an arbitrary-length message into packet- size chunks. However, clever system design may be able to recast the message to take advantage of the packet format. For example, clever encoding may reduce the length of the message enough so that it fits into a single packet. On the other hand, if the message is shorter than a packet or not an even multiple of the packet data size, some extra information may be packed into the remaining bits of a packet. Reception of a packet will probably be implemented with interrupts. The simplest procedural interface will simply check to see whether a received message is waiting in a buffer. In a more complex RTOS-based system, reception of a packet may enable a process for execution.

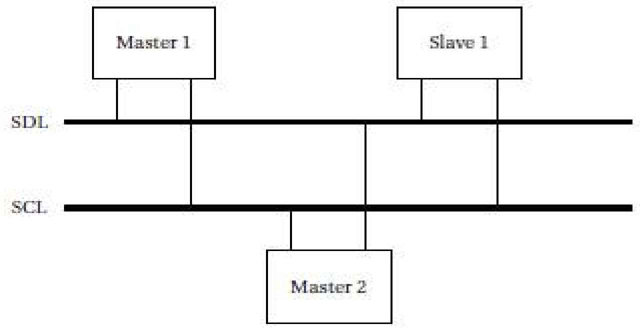
Network protocols may encourage a data-push design style for the system built around the network. In a single-CPU environment, a program typically initiates a read whenever it wants data. In many networked systems, nodes send values out without any request from the intended user of the system. Data-push programming makes sense for periodic data—if the data will always be used at regular intervals, we can reduce data traffic on the network by automatically sending it when it is needed.

NETWORKS FOR EMBEDDED SYSTEMS

Networks for embedded computing span a broad range of requirements; many of those requirements are very different from those for general-purpose networks. Some networks are used in safety-critical applications, such as automotive control. Some networks, such as those used in consumer electronics systems, must be very inexpensive. Other networks, such as industrial control networks, must be extremely rugged and reliable.

The I^{2}C Bus

The I^{2}C bus is a well-known bus commonly used to link microcontrollers into systems. I 2C is designed to be low cost, easy to implement, and of moderate speed (up to 100 KB/s for the standard bus and up to 400 KB/s for the extended bus). As a result, it uses only two lines: the serial data line (SDL) for data and the serial clock line (SCL), which indicates when valid data are on the data line. Figure shows the structure of a typical I2C bus system. Every node in the network is connected to both SCL and SDL. Some nodes may be able to act as bus masters and the bus may have more than one master. Other nodes may act as slaves that only respond to requests from masters.

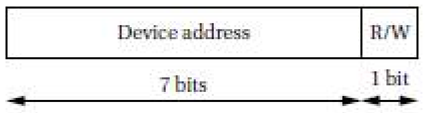


The I2C bus is designed as a multimaster bus—any one of several different devices may act as the master at various times. As a result, there is no global master to generate the clock signal on SCL. Instead, a master drives both SCL and SDL .when it is sending data. When the bus is idle, both SCL and SDL remain high. When two devices try to drive either SCL or SDL to different values, the open collector open drain circuitry prevents errors, but each master device must listen to the bus while transmitting to be sure that it is not interfering with another message—if the device receives a different value than it is trying to transmit, then it knows that it interfering with another message.

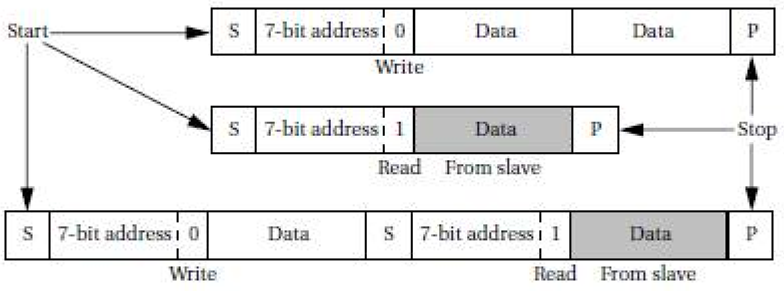
Every I2 C device has an address. The addresses of the devices are determined by the system designer, usually as part of the program for the I2 C driver. The addresses must of course be chosen so that no two devices in the system have the same address. A device address is 7 bits in the standard I2 C definition (the extended I2 C allows 10-bit addresses).

A bus transaction is initiated by a start signal and completed with an end signal as follows:

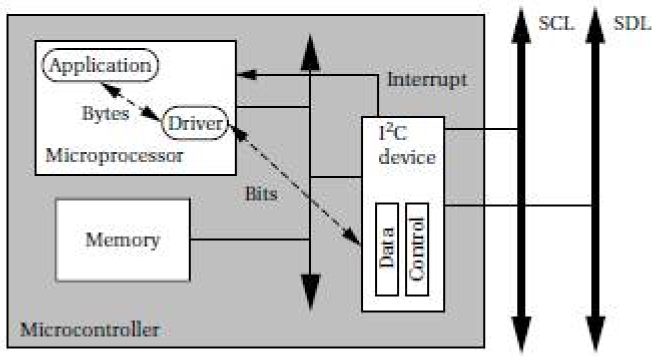
- A start is signaled by leaving the SCL high and sending a 1 to 0 transition on SDL.
- A stop is signaled by setting the SCL high and sending a 0 to 1 transition on SDL.



The formats of some typical complete bus transactions are shown in figure below. In the first example, the master writes 2 bytes to the addressed slave. In the second, the master requests a read from a slave. In the third, the master writes 1 byte to the slave, and then sends another start to initiate a read from the slave.



The I2C interface on a microcontroller can be implemented with varying percentages of the functionality in software and hardware. As illustrated in Figure below, a typical system has a 1-bit hardware interface with routines for byte level functions. The I2C device takes care of generating the clock and data. The application code calls routines to send an address, send a data byte, and so on, which then generates the SCL and SDL, acknowledges, and so forth. One of the microcontroller's timers is typically used to control the length of bits on the bus. Interrupts may be used to recognize bits. However, when used in master mode, polled I/O may be acceptable if no other pending tasks can be performed, since masters initiate their own transfers.

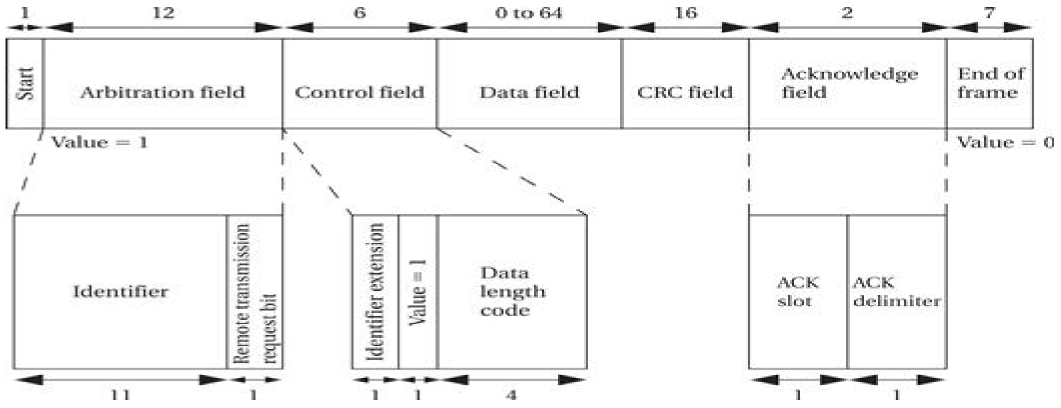


CAN BUS

The **CAN bus** was designed for automotive electronics and was first used in production cars in 1991. It uses bit-serial transmission. CAN can run at rates of 1 Mb/second over a twisted pair connection of 40 meters. An optical link can also be used. The bus protocol supports multiple masters on the bus.

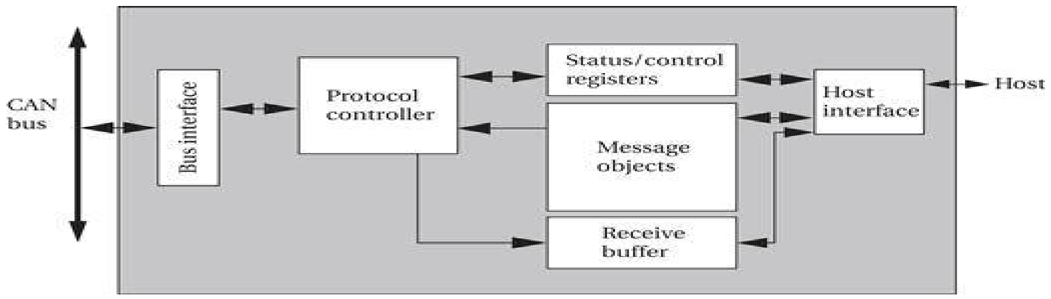
CAN is a synchronous bus—all transmitters must send at the same time for bus arbitration to work. Nodes synchronize themselves to the bus by listening to the bit transitions on the bus. The first bit of a data frame provides the first synchronization opportunity in a frame. The nodes must also continue to synchronize themselves against later transitions in each frame.

The format of a CAN data frame is shown in Figure 8.6. A data frame starts with a 1 and ends with a string of seven zeroes. (There are at least three bit fields between data frames.) The first field in the packet contains the packet's destination address and is known as the arbitration field. The destination identifier is 11 bits long. The trailing remote transmission request (RTR) bit is set to 0 if the data frame is used to request data from the device specified by the identifier. When RTR = 1, the packet is used to write data to the destination identifier. The control field provides an identifier extension and a 4-bit length for the data field with a 1 in between. The data field is from 0 to 64 bytes, depending on the value given in the control field. A cyclic redundancy check (CRC) is sent after the data field for error detection. The acknowledge field is used to let the identifier signal whether the frame was correctly received: The sender puts a recessive bit (1) in the ACK slot of the acknowledge field; if the receiver detected an error, it forces the value to a dominant (0) value. If the sender sees a 0 on the bus in the ACK slot, it knows that it must retransmit. The ACK slot is followed by a single bit delimiter followed by the end-of-frame field.



An error frame can be generated by any node that detects an error on the bus. Upon detecting an error, a node interrupts the current transmission with an error frame, which consists of an error flag field followed by an error delimiter field of 8 recessive bits. The error delimiter field allows the bus to return to the quiescent state so that data frame transmission can resume. The bus also supports an overload frame, which is a special error frame sent during the interframe quiescent period. An overload frame signals that a node is overloaded and will not be able to handle the next message. The node can delay the transmission of the next frame with up to two overload frames in a row, hopefully giving it enough time to recover from its overload. The CRC field can be used to check a message’s data field for correctness.

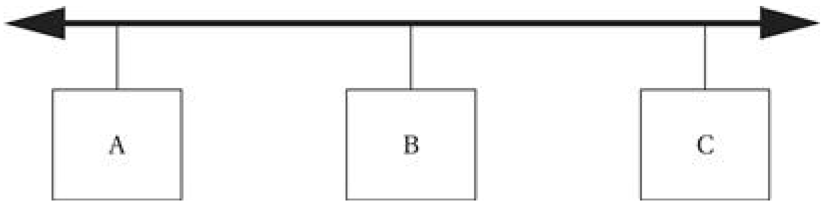
Figure shows the basic architecture of a typical CAN controller. The controller implements the physical and data link layers; because CAN is a bus, it does not need network layer services to establish end-to-end connections. The protocol control block is responsible for determining when to send messages, when a message must be resent due to arbitration losses, and when a message should be received.



ETHERNET

Ethernet is very widely used as a local area network for general-purpose computing. Because of its ubiquity and the low cost of Ethernet interfaces, it has seen significant use as a network for embedded computing. Ethernet is particularly useful when PCs are used as platforms, making it possible to use standard components, and when the network does not have to meet rigorous real-time requirements.

The physical organization of an Ethernet is very simple, as shown in Figure 8.16. The network is a bus with a single signal path; the Ethernet standard allows for several different implementations such as twisted pair and coaxial cable.



Unlike the I^{2}2C or CAN bus, nodes on the Ethernet are not synchronized—they can send their bits at any time. I^{2}2C and CAN rely on the fact that a collision can be detected and quashed within a single bit time thanks to synchronization. But because Ethernet nodes are not synchronized, if two nodes decide to transmit at the same time, the message will be ruined. The Ethernet arbitration scheme is known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD). The algorithm is outlined in Figure 8.17. A node that has a message waits for the bus to become silent and then starts transmitting. It simultaneously listens, and if it hears another transmission that interferes with its transmission, it stops transmitting and waits to retransmit. The waiting time is random, but weighted by an exponential function of the number of times the message has been aborted.

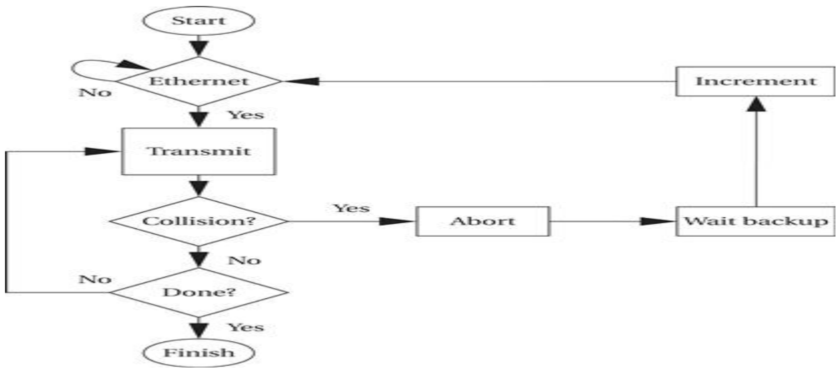
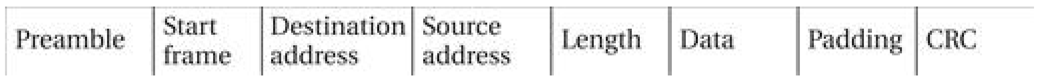


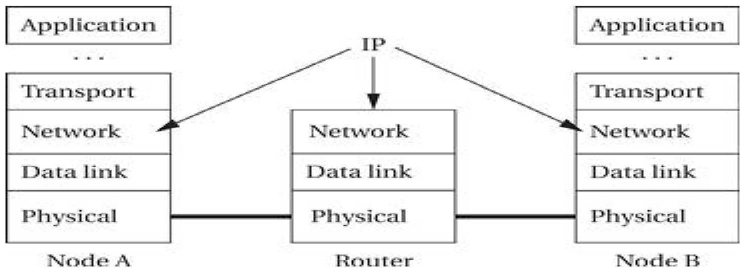
Figure shows the basic format of an Ethernet packet. It provides addresses of both the destination and the source. It also provides for a variable-length data payload.



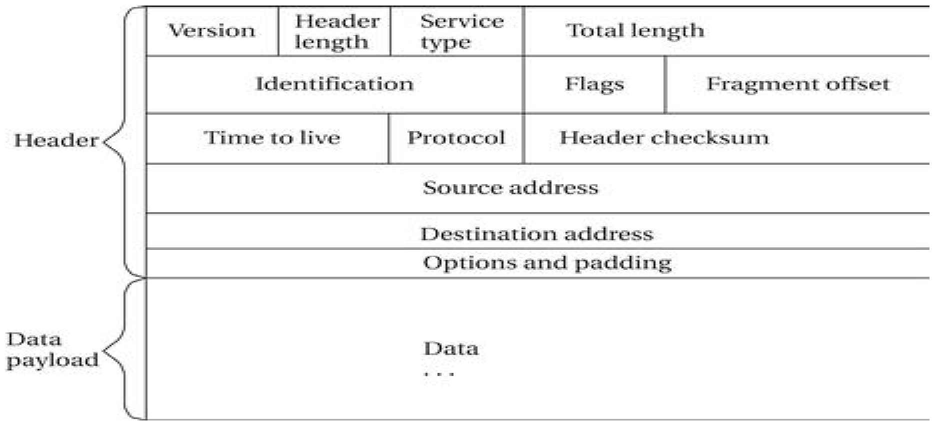
Preamble	Start frame	Destination address	Source address	Length	Data	Padding	CRC

Internet

IP is not defined over a particular physical implementation—it is an **internetworking** standard. Internet packets are assumed to be carried by some other network, such as an Ethernet. In general, an Internet packet will travel over several different networks from source to destination. The IP allows data to flow seamlessly through these networks from one end user to another. The relationship between IP and individual networks is illustrated in Figure. IP works at the network layer. When node *A* wants to send data to node *B*, the application’s data pass through several layers of the protocol stack to get to the Internet Protocol. IP creates packets for routing to the destination, which are then sent to the *data link* and *physical* layers. A node that transmits data among different types of networks is known as a **router.** The router’s functionality must go up to the IP layer, but because it is not running applications, it does not need to go to higher levels of the OSI model. In general, a packet may go through several routers to get to its destination. At the destination, the IP layer provides data to the transport layer and ultimately the receiving application. As the data pass through several layers of the protocol stack, the IP packet data are encapsulated in packet formats appropriate to each layer.



The basic format of an IP packet is shown in Figure. The header and data payload are both of variable length. The maximum total length of the header and data payload is 65,535 bytes.



Embedded Product Development Life Cycle (EDLC)

Embedded Product Development Life Cycle is an 'Analysis -Design -Implementation' based standard problem solving approach for Embedded Product Development. In any product development application, the first and foremost step is to figure out what product needs to be developed (analysis), next you need to figure out a good approach for building it (design) and last but not least you need to develop it.

EDLC is essential for understanding the scope and complexity of the work involved in any embedded product development. EDLC defines the interaction and activities among various groups of a product development sector including project management, system design and development (hardware, firmware and enclosure design and development), system testing, release management and quality assurance. The standards imposed by EDLC on a product development makes the product, developer independent in terms of standard documents and it also provides uniformity in development approaches.

OBJECTIVES OF EDLC

The ultimate aim of any embedded product in a commercial production setup is to produce marginal benefit. Marginal benefit is usually expressed in terms of Return on Investment. A product is said to be profitable only if the turnover from the selling of the product is more than that of the overall investment expenditure. For this, the product should be acceptable by the end user and it should meet the requirements of end user in terms of quality, reliability and functionality. So it is very essential to ensure that the product is meeting all these criteria, throughout the design, development implementation and support phases.

Embedded Product Development Life Cycle (EDLC) helps out in ensuring all these requirements. EDLC has three primary objectives, namely

- Ensure that high quality products are delivered to end user.
- Risk minimization and defect prevention in product development through project management.
- Maximize the productivity.

1. Ensuring High Quality for Products

The primary definition of quality in any embedded product development is the Return on Investment (ROI) achieved by the product. The expenses incurred for developing the product may fall in any of the categories; initial investment, developer recruiting, training, or any other infrastructure requirement related. The budget allocation might have done after studying the market trends and requirements of the product, competition, etc. EDLC must ensure that the development of the product has taken account of all the qualitative attributes of the embedded system.

2. Risk Minimization and Defect Prevention through Management

Whenever a product development request comes, an estimate on the duration of' the development and deployment activity should be given to the end user/client. The timeframe may be expressed in number of person days PDS (The effort in terms of single person working for this much days) or 'X person for X week (e.g. 2 person 2 week) etc. This is one aspect of predictability. The management team might have reached on this estimate based on past experience in handling similar project or on the analysis of work summary or data available for a similar project, which was logged in using an activity tool. Resource (Developer) allocation is another aspect of predictability in management.

Resource allocations like how many resources should work for this project for how many days, how many resources are critical with respect to the work handling by them and how many backups required for the resources to overcome a critical situation where a resource is not available (Risk minimization).

Project management adds an extra cost on the budget but it is essential for ensuring the development process is going in the right direction and the schedules of the development activity are meeting. Without management, the development work may go beyond the stipulated time frame (schedule slipping) and may end up in a product which is not meeting the requirements from the client side.

3. Increased Productivity

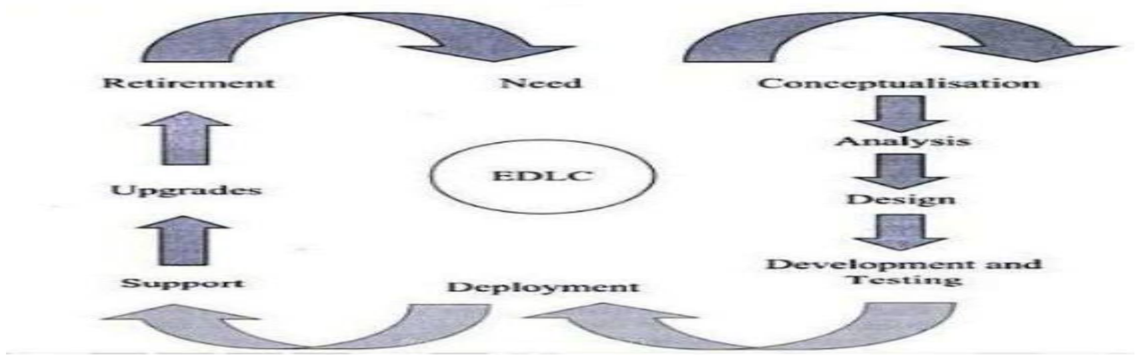
Productivity is a measure of efficiency as well as Return on Investment (ROT). One aspect of productivity covers how many resources are utilized to build the product, how much investment required, how much time is taken for developing the product, etc. For example, the productivity of a system is said to be doubled if a product developed by a team of 'X' members in a period of 'X' days is developed by another team of `X/2' members in a period of 'X' days or by a team of 'X' members in a period of 'X/2' days.

This productivity measurement is based on total manpower efficiency. Productivity in terms of Returns is said to be increased, if the product is capable of yielding maximum returns with reduced investment. Saving manpower effort will definitely result in increased productivity. Usage of automated tools, wherever possible, is recommended for this. The initial investment on tools may be an additional burden in terms of money, but it will definitely save efforts in the next project also. It is a one-time investment. "Pay once use many time". Another important factor which can help in increased productivity is "re -usable effort".

DIFFERENT PHASES OF EDLC

The life cycle of a product development is commonly referred to as models. Model defines the various phases involved in the life cycle. The number of phases involved in a model may vary according to the complexity of the product under consideration.

A typical simple product contains five minimal phases namely: requirement analysis, Design, development and test, deployment and maintenance. The classic Embedded Product Life Cycle Model contains the phases: Need, Conceptualization, Analysis, Design, Development and Testing, Deployment, Support, Upgrades and Retirement/Disposal.



Need

Any embedded product evolves as an output of a Weed'. The need may come from an individual or from the public or from a company (Generally speaking from an end user/client).'Need' should be articulated to initiate the Product Development Life Cycle and based on the need for the product, a 'Statement of Need' or 'Concept Proposal' is prepared. The product development need can be visualized in any one of the following three needs.

1. **New or Custom Developmen**t

The need for a product which does not exist in the market or a product which acts as competitor to an existing product in the current market will lead to the development of a completely new product. The product can be a commercial requirement or an individual requirement or a specific organization’s requirement.

2. **Product Re –engineering**

The embedded product market is dynamic and competitive. In order to sustain in the market, the product developer should be aware of the general market trends, technology changes and the taste of the end use. Reengineering a product is the process of making changes in an existing product design and launching it as a new version. It is generally termed as product upgrade. Re - engineering an existing product comes as a result of the following needs.

- Change in Business requirements
- User Interface Enhancements
- Technology Upgrades

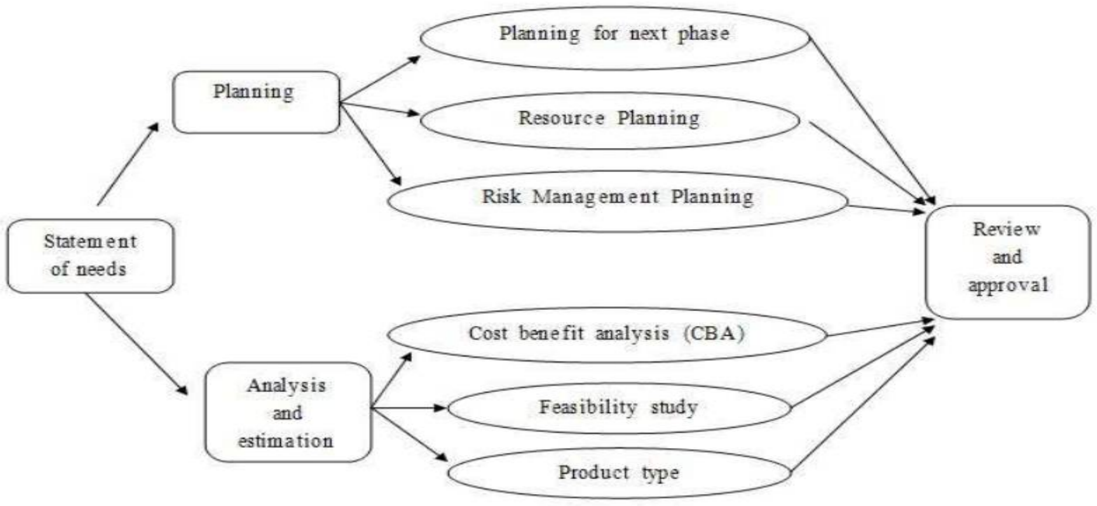
3. Product Maintenance

Product maintenance 'need' deals with providing technical support to the end user for an existing product in the market. The maintenance request may come as a result of product nonfunctioning or failure. Product maintenance is generally classified into two categories: Corrective maintenance and Preventive maintenance. Corrective maintenance deals with making corrective actions following a failure or non- functioning. Preventive maintenance is the scheduled maintenance to avoid the failure or non-functioning of the product.

Conceptualization

Conceptualization is the 'Product Concept Development Phase and it begins immediately after a Concept Proposal is formally approved. Conceptualization phase defines the scope of the concept, performs cost benefit analysis and feasibility study and prepares project management and risk management plans.

Conceptualization can be viewed as a phase shaping the “Need” of an end-user or convincing the end user, whether it is a feasible product and how this product can be realized.



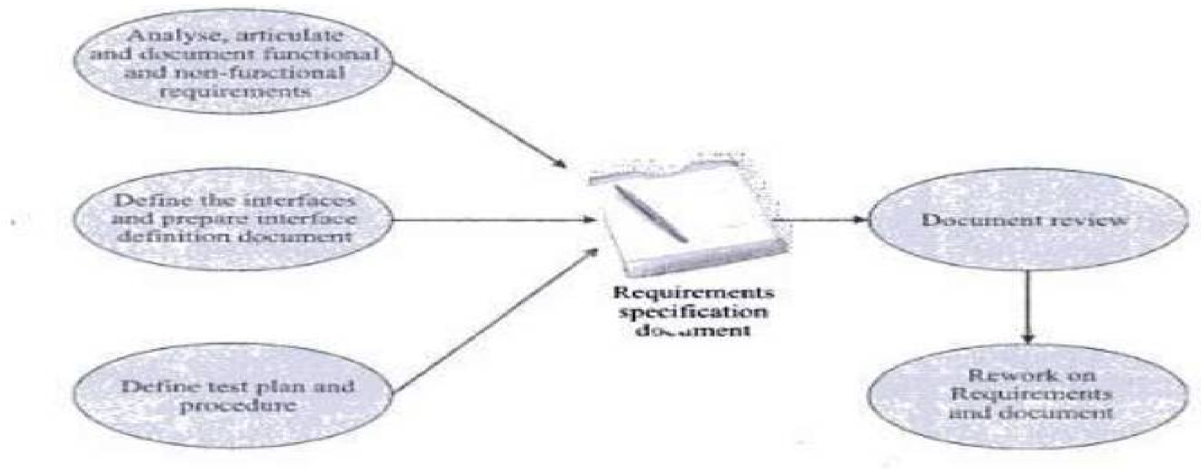
The ‘Conceptualization’ phase involves two types of activities, namely; ‘Planning Activity’ and ‘Analysis and Study Activity’. The Analysis and study activities are performed to understand the opportunity of the product in the market (for a commercial product). The following are the important Analysis and study activities performed during the conceptualization phase.

- **Feasibility Study:** Examine the need and suggest possible solutions.
- **Cost Benefit Analysis (CBA):** Revealing and assessing the total development cost and profit expected from the product. It is somewhat related to loss gain analysis in business. Some Common underlying principles are given below like Common Unit of measurement, market choice based benefit measurement, target end users etc.
- **Product Scope:** Deals with what is scope (functionalities need to be considered) and what is not in scope. It should be properly documented for future reference
- **Planning Activities:** The planning activity covers various plans required for the product development, like the plan and schedule for executing the next phases following conceptualization,
    - Resource Planning – How many resources should work on the project
    - Risk Management Plans – The technical and the other kind of risks involved in the work and what are the migration plans, etc.

At the end of the conceptualization phase, the reports on ‘Analysis and Study Activities’ and ‘Planning Activities’ are submitted to the client/project sponsor for review and approval.

Analysis

A requirement analysis phase starts immediately after the documents submitted during the ‘Conceptualization’ phase is approved by the sponsor of the project. Documentation related to user requirements from the conceptualization phase is used as the basis for the further user need analysis and the development of detailed user requirements. Requirement analysis is performed to develop a detailed functional model of the product under considerations. During the Requirement Analysis phase, the product is defined in detail with respect to the inputs, processes, outputs, and interfaces at the functional level. Requirement analysis phase gives emphasis on determining ‘what functions must be performed by the product’ rather than how to perform those functions.


Analyse, articulate and document functional and non-functional requirements
Define the interfaces and prepare interface definition document
Define test plan and procedure
Requirements specification document
Document review
Rework on Requirements and document

various activities involved in requirement analysis phase

The various activities performed during this phase..

1. Analysis and Documentations: This activity consolidates the business needs of the product under development. Requirements that need to be addressed during this phase are

1. Functional Capabilities like performance
2. Operational and non-operational quality attributes.
3. Product external interface requirements
4. User manuals and Data requirements
5. Operational requirements
6. Maintenance requirements
7. General assumptions

2. Interface definition and documentation

This activity should clearly analyze on the physical interfaces as well as data exchange through these interfaces and should document it.

3. Defining Test Plan and Procedures

The various type of testing performed in a product development are:

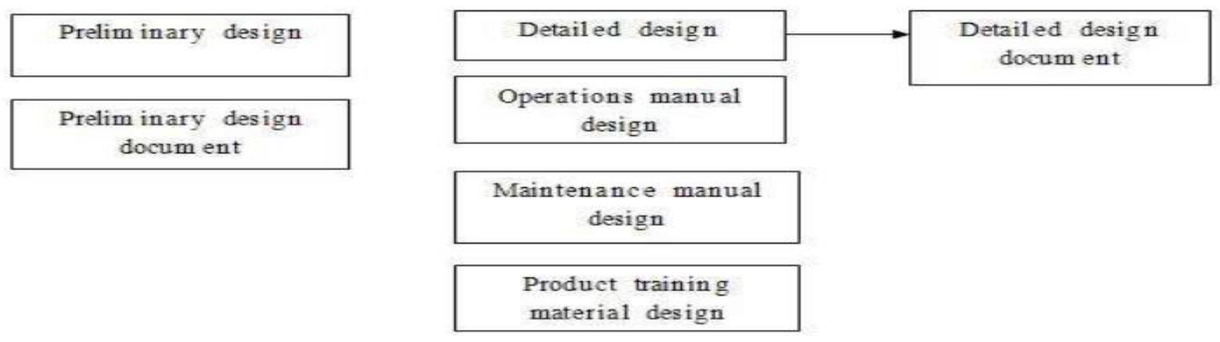
- Unit testing—Testing each unit or module of the product independently for required functionality and quality aspects

- Integration testing—Integrating each modules and testing the integrated unit for required functionality
- System testing- Testing the functional aspects/product requirements of the product after integration. System testing refers to a set of different tests and a few among them are listed below.
    - Usability testing—Teats the usability of the product
    - Load testing—Tests the behavior of the product under different loading conditions.
    - Security testing—testing the security aspects of the product
    - Scalability testing—testing the scalability aspect of the product
    - Sanity testing—superficial testing performed to ensure that the product is functioning
    - properly
    - Smoke testing—non exhaustive test to ensure that the crucial requirements for the product are functioning properly. This test is not giving importance to the finer details of different functionalities.
    - Performance testing—Testing the performance aspects of the product after integration
    - Endurance testing—Durability test of the product
- User acceptance testing- Testing the product to ensure it is meeting all requirements of the end user.

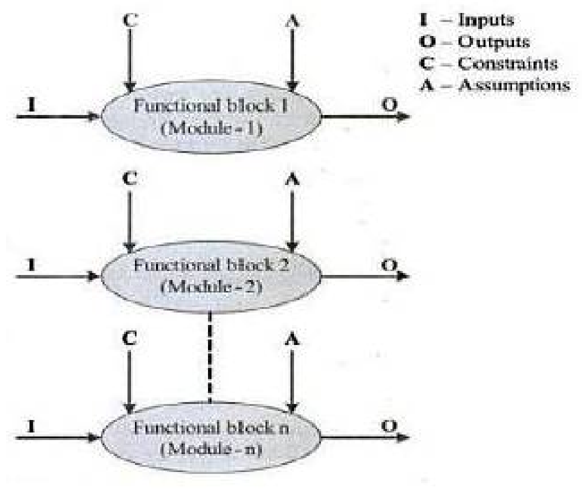
At the end, all requirements should be put in a template suggested by the standard (e.g. ISO - 900l) Process Model (e.g. CMM Level 5) followed for the Quality Assurance. This document is referred as 'Requirements Specification Document'.

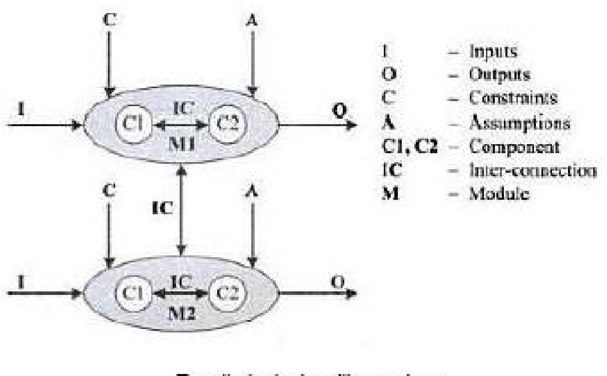
Design

The design phase identifies application environment and creates an overall architecture for the product. It starts with the Preliminary Design. It establishes the top level architecture for the product. On completion it resembles a ‘black box’ that defines only the inputs and outputs. The final product is called Preliminary Design Document (PDD). Once the PDD is accepted by the End User the next task is to create the ‘Detailed Design’. It encompasses the Operations manual design, Maintenance Manual Design and Product Training material Design and is together called the ‘Detailed Design Document’.


Preliminary design
Preliminary design document
Detailed design
Detailed design document
Operations manual design
Maintenance manual design
Product training material design

various activities involved in design phase


C
A
I
O
Functional block 1 (Module-1)
Functional block 2 (Module-2)
Functional block n (Module-n)
I – Inputs
O – Outputs
C – Constraints
A – Assumptions



Detailed design illustration

preliminary design illustration

Development and Testing

Development phase transforms the design into a realizable product. The detailed specification generated during the design phase is translated into hardware and firmware. The Testing phase can be divided into independent testing of firmware and hardware that is:

- Unit testing
- Integration testing
- System testing
- User acceptance testing

Deployment

Deployment is the process of launching the first fully functional model of the product in the market (for a commercial embedded product) or handing over the fully functional initial model to an end user/client. It is also known as First Customer Shipping (FCS).During this phase, the product modifications as per the various integration tests are implemented and the product is made operational in a production environment. The 'Deployment Phase' is initiated after the system is tested and accepted (User Acceptance Test) by the end user. The important tasks performed during the Deployment Phase arc listed below.

1. **Notification of Product Deployment:** Tasks performed here include:
    - Deployment schedule
    - Brief description about the product
    - Targeted end user
    - Extra features supported
    - Product support information
2. **Execution of training plan:** Proper training should be given to the end user top get them acquainted with the new product.
3. Product installation: Install the product as per the installation document to ensure that it is fully functional.
4. **Product post Implementation Review:** After the product launch, a post implementation review is done to test the success of the product

Support

The support phase deals with the operational and maintenance of the product in the production environment. Bugs in the product may be observed and reported. The support phase ensures that the product meets the user needs and it continues functioning in the production environment. Activities involved under support are

- **Setting up of a dedicated support wing**: Involves providing 24 x 7 supports for the product after it is launched.
- **Identify Bugs and Areas of Improvement:** Identify bugs and take measures to eliminate them.

Upgrades

The upgrade phase of product development deals with the development of upgrades (new versions) for the product which is already present in the market. Product upgrade results as an output of major bug fixes or feature enhancement requirements from the end user. During the upgrade phase the system is subject to design modification to fix the major bugs reported or to incorporate the new feature addition requirements aroused during the support phase. For embedded products, the upgrades may be for the product resident firmware or for the hardware embedding the firmware.

Some bugs may be easily fixed by modifying the firmware and it is known as firmware up - gradation. Some feature enhancements can also be performed easily by mere firmware modification. The product resident firmware will have a version number which starts with version say 1.0 and after each firmware modification or bug fix, the firmware version is changed accordingly (e.g. version 1.1). Version numbering is essential for backward traceability. Releasing of upgrades is managed through release management.

Retirement/Disposal

We are living in a dynamic world where everything is subjected to rapid changes. The technology you feel as the most advanced and best today may not be the same tomorrow. Due to the most advanced revolutionary technological changes, a product cannot sustain in the market for a long time. The disposal/retirement of the product is a gradual process. When the product manufacturer realizes that there is another powerful technology or the product available in the market which is most suitable for the production of the current product, they will release the current product as obsolete and the newer version/upgrade of the same is going to be released soon. The disposition of the product is essential due to the following reason.

1. Rapid technological advancements.
2. Increased user needs.

ELDC APPROACHES

Following are some of the different types of approaches that can be used to model embedded products.

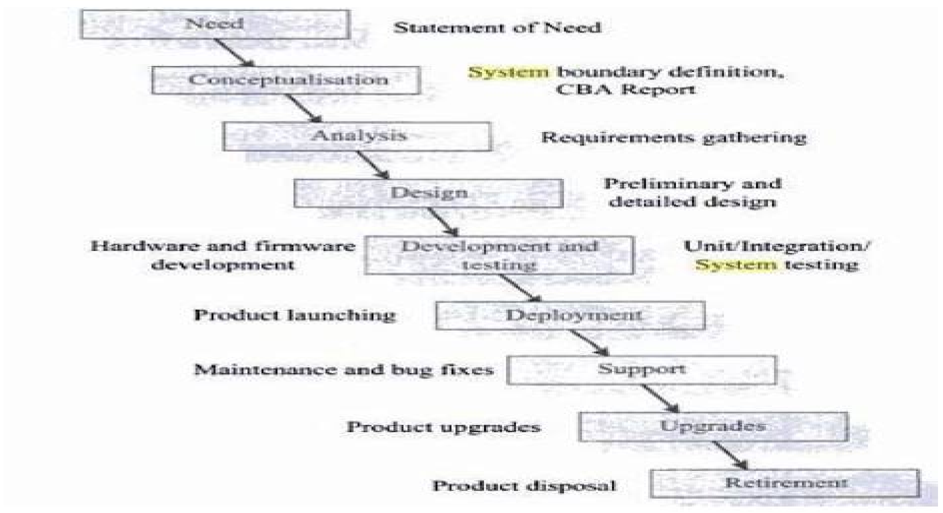
1. Waterfall or Linear Model
2. Iterative/ Incremental or Fountain Model
3. Prototyping Model/Evolutionary Model
4. Spiral Model

Waterfall or Linear Model

Linear or waterfall model approach is adopted in most of the older systems and in this approach each phase of EDLC is executed in sequence. The linear model establishes a formal analysis and design methodology with highly structured development phases. In linear model, the various phases of EDLC are executed in sequence and the flow is unidirectional with output of one phase serving as the input to the next phase. In the linear model all activities involved in each phase are well documented, giving an insight into what should be done in the next phase and how it can be done.

The feedback of each phase is available locally and only after they are executed. The linear model implements extensive review mechanisms to ensure the process flow is going in the right direction and validates the effort during a phase. One significant feature of linear model is that even if you identify bugs in the current design, the corrective actions are not implemented immediately and the development process proceeds with the current design. The fixes for the bugs are postponed till the support phase, which accompanies the deployment phase. The major advantage of 'Linear Model' is that the product development is rich in terms of documentation, easy project management and good control over cost and schedule.

The major drawback of this approach is that it assumes all the analysis can be done and everything will be in right place without doing any design or implementation. Also the risk analysis is performed only once throughout the development and risks involved in any changes are not accommodated in subsequent phases, the working product is available only at the end of the development phase and bug fixes and corrections are performed only at the maintenance/support phase of the life cycle. 'Linear Model' is best suited for product developments, where the requirements are well defined and within the scope, and no change request are expected till the completion of the cycle.



. linear (Waterfall) EDLC model

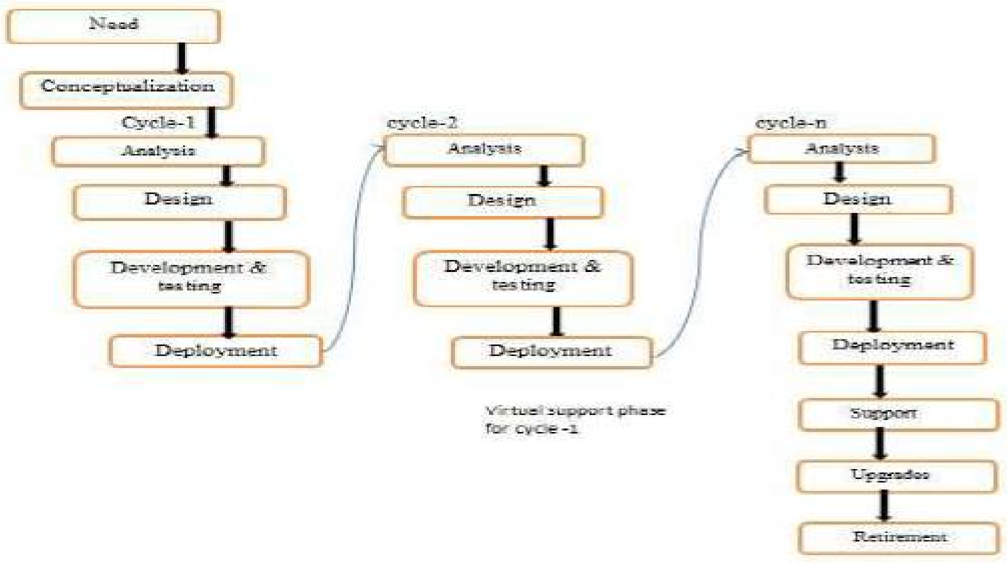
Iterative/Incremental or Fountain model

Iteration or fountain model follows the sequence – Do some analysis, follow some design, the some implementation, evaluate it and based on the shortcomings, cycle black through and conduct more analysis, opt for new design and implementation and repeat the cycle till the requirements are met completely. The iterative model can be viewed as a cascaded series of linear models. The incremental model is a superset of iterative model where the requirements are known at the beginning and they divided into different groups. If you closely observe this model you can see that each cycle is interconnected in a similar fashion of a fountain, where water first moves up and then comes down, again moves up and comes down.

The major advantage of iterative/fountain model is that it provides very good development cycle feedback at each function/feature implementation and hence the data can be used as a reference for similar product development in future. Since each cycle can act as a maintenance phase for previous cycle, changes in feature and functionalities can be easily incorporated during the development and hence more responsive to changing user needs. The iterative model provides a working product model with at least minimum features at the first cycle itself. Risk is spread across each individual cycle and can be minimized easily. Project management as well as testing is much simpler compared to the linear model.

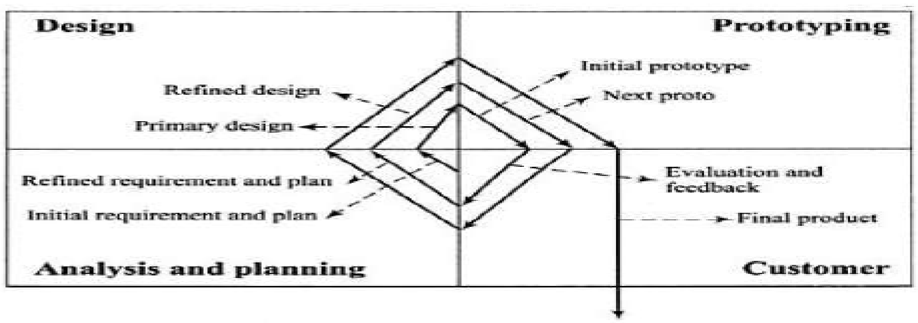
Another major advantage is that the product development can be stopped at any stage with a bare minimum working product. Though iterative model is a good solution for product development, it possess lots of drawbacks like extensive review requirement at each cycle, impact on operations due to new releases, training requirement for each new deployment at the end o f each development cycle, structured and well documented interface definition across modules to accommodate changes.

The iterative/incremental model is deployed in product developments where the risk is very high when the development is carried out by linear model. By choosing an iterative model, the risk is spread across multiple cycles. Since each cycle produces a working model, this model is best suited for product developments where the continued funding for each cycle is not assured.



Prototyping/Evolutionary Model

Prototyping/evolutionary model is similar to the iterative model and the product is developed in multiple cycles. The only difference is that this model produces a more refined prototype of the product at the end of each cycle instead of functionality/feature addition in each cycle as performed by the iterative model. There won't be any commercial deployment of the prototype of the product at each cycle's end. The shortcomings of the proto-model after each cycle are evaluated and it is fixed in the next cycle. After the initial requirement analysis, the design for the first prototype is made, the development process is started. On finishing the prototype, it is sent to the customer for evaluation. The customer evaluates the product for the set of requirements and gives his/her feedback to the developer in terms of shortcomings and improvements needed. The developer refines the product according to the customer's exact expectation and repeats the proto development process. After a finite number of iterations, the final product is delivered to the customer and launches in the market/operational environment. In this approach, the product undergoes significant evolution as a result of periodic shuttling of product information between the customer and developer. The prototyping model follows die approach — 'Requirements definition, proto-type development, proto-type evaluation and requirements refining'. Since the requirements undergo refinement after each proto model, it is easy to incorporate new requirements and technology changes at any stage and thereby the product development process can start with a bare minimum set of requirements.



The evolutionary model relies heavily on user feedback after each implementation and hence fine-tuning of final requirements is possible. Another major advantage of prototyping model is that the risk is spread across each proto development cycle and it is well under control. The major drawbacks of proto-typing model are

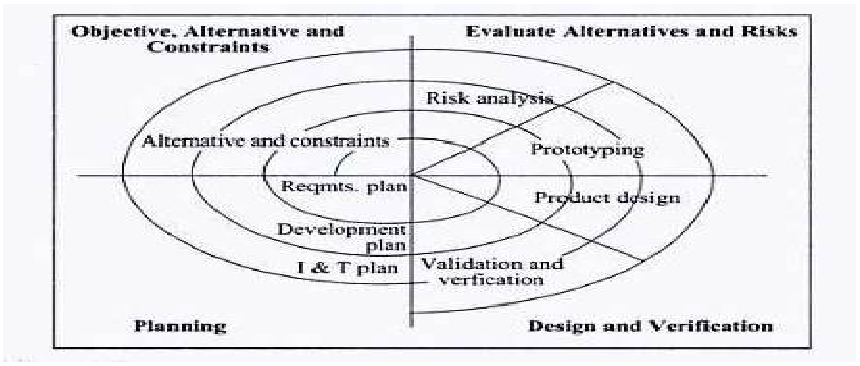
- Deviations from expected cost and schedule due to requirements refinement
- Increased project management
- Minimal documentation on each prototype may create problems in backward prototype traceability
- Increased Configuration Management activities

Prototyping model is the most popular product development model adopted in embedded product industry. This approach can be considered as the best approach for products, whose requirements are not fully available and are subject to change. This model is not recommended for projects involving the up gradation of an existing product. There can be slight variations in the base prototyping model depending on project management.

Spiral Model

Spiral model combines the elements of linear and prototyping models to give the best possible risk minimized EDLC Model. Spiral model is developed by Barry Boehm in 1988. The product development starts with project definition and traverse through all phases of EDLC through multiple phases. The activities involved in the Spiral model can be associated with the four quadrants of a spiral and are listed below.

- Determine objectives, alternatives, constraints.
- Evaluate alternatives. Identify and resolve risks.
- Develop and test.
- Plan.


Objective, Alternative and Constraints
Evaluate Alternatives and Risks
Risk analysis
Alternative and constraints
Prototyping
Reqmts. plan
Product design
Development plan
I & T plan
Validation and verfication
Planning
Design and Verification

Spiral model is best suited for the development of complex embedded products and situations where requirements arc changing from customer side. Customer evaluation of prototype at each stage allows addition of requirements and technology changes. Risk evaluation in each stage helps in risk planning and mitigation. The proto model developed at each stage is evaluated by the customer against various parameters like strength, weakness, risk, etc. and the final product is built based on the final prototype on agreement with the client.

TRENDS IN EMBEDDED SYSTEMS

9.1. PROCESSOR TRENDS

There have been tremendous advancements in the area of processor design.

Following are some of the points of difference between the first generation of processor/controller and today's processor/ controller.

- **Number of ICs per chip**: Early processors had a few number of IC/gates per chip. Today's processors with Very Large Scale Integration (VLSI) technology can pack together ten of thousands of IC/gates per processor.
- **Need for individual components:** Early processors need different components like brown out circuit, timers, DAC/ADC separately interfaced if required to be used in the circuit. Today's processors have all these components on the same chip as the processor.
- **Speed of Execution:** Early processors were slow in terms of number of instructions executed per second. Today's processor with advanced architecture support features like instruction pipeline improving the execution speed.
- **Clock frequency:** Early processors could execute at a frequency of a few MHz only. Today's processors are capable of achieving execution frequency in rage of GHz.
- **Application specific processor:** Early systems were designed using the processors available at that time. Today it is possible to custom create a processor according to a product requirement.

Following are the major trends in processor architecture in embedded development.

9.1.1 System on Chip (SoC)

This concept makes it possible to integrate almost all functional systems required to build an embedded product into a single chip. SoC are now available for a wide variety of diverse applications like Set Top boxes, Media Players, PDA, etc. SoC integrate multiple functional components on the same chip thereby saving board space which helps to miniaturize the overall design.

9.1.2 Multicore Processors/ Chiplevel Multi Processor

This concept employs multiple cores on the same processor chip operating at the same clock frequency and battery. Based on the number of cores, these processors are known as:

■ Dual Core – 2 cores

■ Tri Core – 3 cores

■ Quad Core – 4 cores

Processors implement multiprocessing concept where each core implements pipelining and multithreading.

9.1.3 Reconfigurable Processors

It is a processor with reconfigurable hardware features.

Depending on the requirement, reconfigurable processors can change their functionality to adapt to the new requirement. Example: A reconfigurable processor chip can be configured as the heart of a camera or that of media player.

These processors contain an Array of Programming Elements (PE) along with a microprocessor. The PE can be used as a computational engine like ALU or a memory element.

9.2 EMBEDDED OPERATING SYSTEM TRENDS

The advancements in processor technology have caused a major change in the Embedded Operating System Industry. There are lots of options for embedded operating system to select from which can be both commercial and proprietary or Open Source. Virtualization concept is brought in picture in the embedded OS industry which replaces the monolithic architecture with the microkernel architecture. This enables only essential services to be contained in the kernel and the rest are installed as services in the user space as is done in Mobile phones.

Off the shelf OS customized for specific device requirements are now becoming a major trend.

9.3 DEVELOPMENT LANGUAGE TRENDS

There are two aspects to Development Languages with respect to Embedded Systems Development

- Embedded Firmware

It is the application that is responsible for execution of embedded system. It is the software that performs low level hardware interaction, memory management etc on the embedded system.

- Embedded Software

It is the software that runs on the host computer and is responsible for interfacing with the embedded system. It is the user application that executes on top of the embedded system on a host computer. Early languages available for embedded systems development were limited to C & C++ only. Now languages like Microsoft C$, ASP.NET, VB, Java, etc are available.

9.3.1 Java for embedded development

Java is not a popular language for embedded systems development due to its nature of execution. Java programs are compiled by a compiler into bytecode. This bytecode is then converted by the JVM into processor specific object code. During runtime, this interpretation of the bytecode by

the JVM makes java applications slower that other cross compiled applications. This disadvantage is overcome by providing in built hardware support for java bytecode execution.
Another technique used to speed up execution of java bytecode is using Just In Time (JIT) compiler. It speeds up the program execution by caching all previously executed instruction.
Following are some of the disadvantage of Java in Embedded Systems development:

■ For real time applications java is slow

■ Garbage collector of Java is non-deterministic in behavior which makes it not suitable for hard real time systems.

■ Processors need to have a built in version of JVM

■ Those processors that don't have JVM require it to be ported for the specific processor architecture.

■ Java is limited in terms of low level hardware handling compared to C and C++

■ Runtime memory requirement of JAVA is high which is not affordable by embedded systems.

9.3.2 .NET CF for embedded development

It stands for .NET Compact Framework. NET CF is a replacement of the original .NET framework to be used on embedded systems. The CF version is customized to contain all the necessary components for application development. The Original version of .NET Framework is very large and hence not a good choice for embedded development. The .NET Framework is a collection of precompiled libraries. Common Language Runtime (CLR) is the runtime environment of .NET. It provides functions like memory management, exception handling, etc. Applications written in .NET are compiled to a platform neutral language called Common Intermediate Language (CIL). For execution, the CIL is converted to target specific machine instructions by CLR.

9.4. OPEN STANDARDS, FRAMEWORKS AND ALLIANCES

Standards are necessary for ensuring interoperability. With diverse market it is essential to have formal specifications to ensure interoperability.
Following are some of the popular strategic alliances, open source standards and frameworks specific to the mobile handset industry.

9.4.1 Open Mobile Alliance (OMA)

It is a standard body for creating open standards for mobile industry. OMA is the Leading Industry Forum for Developing Market Driven – Interoperable Mobile Service Enablers. OMA was formed in June 2002 by the world's leading mobile operators, device and network suppliers, information technology companies and content and service providers. OMA delivers open specifications for creating interoperable services that work across all geographical boundaries, on any bearer network. OMA's specifications support the billions of new and existing fixed and mobile terminals across a variety of mobile networks, including traditional cellular operator networks and emerging networks supporting machine-to-machine device communication. OMA is the focal point for the development of mobile service enabler specifications, which support the creation of interoperable end-to-end mobile services.

Goals of OMA

- Deliver high quality, open technical specifications based upon market requirements that drive modularity, extensibility, and consistency amongst enablers to reduce industry implementation efforts.
- Ensure OMA service enabler specifications provide interoperability across different devices, geographies, service providers, operators, and networks; facilitate interoperability of the resulting product implementations.
- Be the catalyst for the consolidation of standards activity within the mobile data service industry; working in conjunction with other existing standards organizations and industry fora to improve interoperability and decrease operational costs for all involved.
- Provide value and benefits to members in OMA from all parts of the value chain including content and service providers, information technology providers, mobile operators and wireless vendors such that they elect to actively participate in the organization.

9.4.2 Open Handset Alliance (OHA)

The Open Handset Alliance is a group of 84 technology and mobile companies who have come together to accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience. Together they have developed Android™, the first complete, open, and free mobile platform and are committed to commercially deploy handsets and services using the

Android Platform. Members of OHA include mobile operators, handset manufacturers, semiconductor companies, software companies, and commercialization companies.

9.4.3 Android

Android is an operating system based on the Linux kernel, and designed primarily for touchscreen mobile devices such as smartphones and tablet computers. Initially developed by Android, Inc., which Google supported financially and later bought in 2005, Android was unveiled in 2007 along with the founding of the Open Handset Alliance: a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices. The first publicly-available Smartphone to run Android, the HTC Dream, was released on October 18, 2008

9.4.4 Openmoko

Openmoko is a project to create a family of open source mobile phones, including the hardware specification and the operating system. The first sub-project is Openmoko Linux, a Linux-based operating system designed for mobile phones, built using free software. The second sub-project is developing hardware devices on which Openmoko Linux runs.

Bottlenecks faced by Embedded Industry

Following are some of the problems faced by the embedded devices industry:

1. **Memory Performance**

The rate at which processors can process may have increased considerably but rate at which memory speed is increasing is slower.

2. **Lack of Standards/ Conformance to standards**

Standards in the embedded industry are followed only in certain handful areas like Mobile handsets. There is growing trend of proprietary architecture and design in other areas.

3. **Lack of Skilled Resource**

Most important aspect in the development of embedded system is availability of skilled labor. There may be thousands of developers who know how to code in C, C++, Java or .NET but very little in embedded software