Syllabus
S.Y.B.Sc (Information Technology)
Sem - IV, Paper - V
Embedded Systems

Unit - I **Introduction :** Embedded Systems and general purpose computer systems, history, classifications, applications and purpose of embedded systems.

Core of Embedded Systems : Microprocessors and microcontrollers, RISC and CISC controllers, Big endian and Little endian processors, Application specific ICs, Programmable logic devices, COTS, sensors and actuators, communication interface, embedded firmware, other system components, PCB and passive components.

Unit - II **Characteristics and quality attributes of embedded systems :** Characteristics, Operational and non-operational quality attributes, application specific embedded system - washing machine, domain specific - automotive.

Unit - III **Programming Embedded Systems :** Structure of embedded program, infinite loop, compiling, linking and locating, downloading and debugging.

Unit - IV **Embedded hardware :** Memory map, i/o map, interrupt map, processor family, external peripherals, memory - RAM, ROM, types of RAM and ROM, memory testing, CRC, Flash memory.

Unit - V **Peripherals :** Control and Status Registers, Device Driver, Timer Driver-Watchdog Timers, Embedded Operating System, Real-Time Characteristics, Selection Process.

Unit - VI **Design and Development :** Embedded System development environment - IDE, Types of file generated on cross compilation, disassembler / decompiler, simulator, emulator and debugging, embedded product development life-cycle, trends in embedded industry.

Books :

Programming Embedded Systems in C and C++, First Edition January, Michael Barr, O' Reilly Introduction to embedded systems, Shibu K V Tata McGraw-Hill.

References :

Embedded Systems, Rajkamal, TataMcGraw-Hill

Term Work :
Assignments : Should contain at least 6 assignments (one per unit) covering the Syllabus.

Tutorial : At least three tutorials based on above syllabus must be conducted.

Practical List :

1) Configure timer control registers of 8051 and develop a program to generate given time delay.
2) Port I / O : Use one of the four ports of 8051 for O / P interfaced to eight LED's. Simulate binary counter (8 bit) on LED's.
3) Serial I / O : Configure 8051 serial port for asynchronous serial communication with serial port of PC exchange text messages to PC and display on PC screen. Signify end of message by carriage return.
4) Interface 8051 with D/A converter and generate square wave of given frequency on oscilloscope.
5) Interface 8051 with D/A converter and generate triangular wave of given frequency on oscilloscope.
6) Using D/A converter generate sine wave on oscilloscope with the help of lookup table stored in data area of 8051.
7) Interface Stepper motor with 8051 and write a program to move the motor through a given angle in clock wise or counter clock wise direction.
8) Generate traffic signal.
9) Temperature controller.
10) Elevator control.

1

EMBEDDED SYSTEM : AN INTRODUCTION

Unit Structure

1.0 OBJECTIVES

- ✓ To understand what is an Embedded System and then define it
- ✓ Look at embedded systems from a historical point of view
- ✓ Classify embedded systems
- ✓ Look at certain applications & purposes of embedded systems

1.1 INTRODUCTION

This chapter introduces the reader to the world of embedded systems. Everything that we look around us today is electronic. The days are gone where almost everything was manual. Now even the food that we eat is cooked with the assistance of a microchip (oven) and the ease at which we wash our clothes is due to the washing machine. This world of electronic items is made up of embedded system. In this chapter we will understand the basics of embedded system right from its definition.

1.2 DEFINITION OF AN EMBEDDED SYSTEM

- An embedded system is a combination of 3 things:
    a. Hardware
    b. Software
    c. Mechanical Components

  And it is supposed to do one specific task only.

- **Example 1: Washing Machine**

  A washing machine from an embedded systems point of view has:

  a. Hardware: Buttons, Display & buzzer, electronic circuitry.
  b. Software: It has a chip on the circuit that holds the software which drives controls & monitors the various operations possible.
  c. Mechanical Components: the internals of a washing machine which actually wash the clothes control the input and output of water, the chassis itself.

- **Example 2: Air Conditioner**

  An Air Conditioner from an embedded systems point of view has:

  a. Hardware: Remote, Display & buzzer, Infrared Sensors, electronic circuitry.
  b. Software: It has a chip on the circuit that holds the software which drives controls & monitors the various operations possible. The software monitors the external temperature through the sensors and then releases the coolant or suppresses it.
  c. Mechanical Components: the internals of an air conditioner the motor, the chassis, the outlet, etc

- An embedded system is designed to do a specific job only. Example: a washing machine can only wash clothes, an air conditioner can control the temperature in the room in which it is placed.

- The hardware & mechanical components will consist all the physically visible things that are used for input, output, etc.

- An embedded system will always have a chip (either microprocessor or microcontroller) that has the code or software which drives the system.

1.3 HISTORY OF EMBEDDED SYSTEM

- The first recognised embedded system is the Apollo Guidance Computer(AGC) developed by MIT lab.
- AGC was designed on 4K words of ROM & 256 words of RAM.
- The clock frequency of first microchip used in AGC was 1.024 MHz.
- The computing unit of AGC consists of 11 instructions and 16 bit word logic.

- It used 5000 ICs.
- The UI of AGC is known DSKY(display/keyboard) which resembles a calculator type keypad with array of numerals.
- The first mass-produced embedded system was guidance computer for the Minuteman-I missile in 1961.
- In the year 1971 Intel introduced the world's first microprocessor chip called the 4004, was designed for use in business calculators. It was produced by the Japanese company Busicom.

1.4 EMBEDDED SYSTEM & GENERAL PURPOSE COMPUTER

The Embedded System and the General purpose computer are at two extremes. The embedded system is designed to perform a specific task whereas as per definition the general purpose computer is meant for general use. It can be used for playing games, watching movies, creating software, work on documents or spreadsheets etc.

Following are certain specific points of difference between embedded systems and general purpose computers:

Criteria	**General Purpose Computer**	**Embedded system**
Contents	It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Operating System	It contains general purpose operating system	It may or may not contain operating system.
Alterations	Applications are alterable by the user.	Applications are non-alterable by the user.
Key factor	Performance" is key factor.	Application specific requirements are key factors.
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications

1.5 CLASSIFICATION OF EMBEDDED SYSTEM

The classification of embedded system is based on following criteria's:

- On generation
- On complexity & performance
- On deterministic behaviour
- On triggering

1.5.1 On generation

1. **First generation(1G):**
   - Built around 8bit microprocessor & microcontroller.
   - Simple in hardware circuit & firmware developed.
   - Examples: Digital telephone keypads.

2. **Second generation(2G):**
   - Built around 16-bit µp & 8-bit µc.
   - They are more complex & powerful than 1G µp & µc.
   - Examples: SCADA systems

3. **Third generation(3G):**
   - Built around 32-bit µp & 16-bit µc.
   - Concepts like Digital Signal Processors(DSPs), Application Specific Integrated Circuits(ASICs) evolved.
   - Examples: Robotics, Media, etc.

4. **Fourth generation**:
   - Built around 64-bit µp & 32-bit µc.
   - The concept of System on Chips (SoC), Multicore Processors evolved.
   - Highly complex & very powerful.
   - Examples: Smart Phones.

1.5.2 On complexity & performance

1. **Small-scale:**
   - Simple in application need
   - Performance not time-critical.
   - Built around low performance & low cost 8 or 16 bit µp/µc.
   - Example: an electronic toy

2. **Medium-scale:**
   - Slightly complex in hardware & firmware requirement.
   - Built around medium performance & low cost 16 or 32 bit µp/µc.
   - Usually contain operating system.
   - Examples: Industrial machines.

3. Large-scale:

- Highly complex hardware & firmware.
- Built around 32 or 64 bit RISC µp/µc or PLDs or Multicore Processors.
- Response is time-critical.
- Examples: Mission critical applications.

1.5.3 On deterministic behaviour

- This classification is applicable for “Real Time” systems.
- The task execution behaviour for an embedded system may be deterministic or non-deterministic.
- Based on execution behaviour Real Time embedded systems are divided into Hard and Soft.

1.5.4 On triggering

- Embedded systems which are “Reactive” in nature can be based on triggering.
- Reactive systems can be:
  - ✓ Event triggered
  - ✓ Time triggered

1.6 APPLICATION OF EMBEDDED SYSTEM

The application areas and the products in the embedded domain are countless.

1. Consumer Electronics: Camcorders, Cameras.
2. Household appliances: Washing machine, Refrigerator.
3. Automotive industry: Anti-lock breaking system(ABS), engine control.
4. Home automation & security systems: Air conditioners, sprinklers, fire alarms.
5. Telecom: Cellular phones, telephone switches.
6. Computer peripherals: Printers, scanners.
7. Computer networking systems: Network routers and switches.
8. Healthcare: EEG, ECG machines.
9. Banking & Retail: Automatic teller machines, point of sales.
10. Card Readers: Barcode, smart card readers.

1.7 PURPOSE OF EMBEDDED SYSTEM

1. Data Collection/Storage/Representation

- Embedded system designed for the purpose of data collection performs acquisition of data from the external world.
- Data collection is usually done for storage,analysis, manipulation and transmission.
- Data can be analog or digital.

- Embedded systems with analog data capturing techniques collect data directly in the form of analog signal whereas embedded systems with digital data collection mechanism converts the analog signal to the digital signal using analog to digital converters.
- If the data is digital it can be directly captured by digital embedded system.
- A digital camera is a typical example of an embedded
- System with data collection/storage/representation of data.
- Images are captured and the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.

2. Data communication

- Embedded data communication systems are deployed in applications from complex satellite communication to simple home networking systems.
- The transmission of data is achieved either by a wire-line medium or by a wire-less medium.
- Data can either be transmitted by analog means or by digital means.
- Wireless modules-Bluetooth, Wi-Fi.
- Wire-line modules-USB, TCP/IP.
- Network hubs, routers, switches are examples of dedicated data transmission embedded systems.

3. Data signal processing

- Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, audio video codec, transmission applications etc.
- A digital hearing aid is a typical example of an embedded system employing data processing.
- Digital hearing aid improves the hearing capacity of hearing impaired person

4. Monitoring

- All embedded products coming under the medical domain are with monitoring functions.
- Electro cardiogram machine is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.
- Other examples with monitoring function are digital CRO, digital multi-meters, and logic analyzers.

5. Control

- A system with control functionality contains both sensors and actuators.

- Sensors are connected to the input port for capturing the changes in environmental variable and the actuators connected to the output port are controlled according to the changes in the input variable.
- Air conditioner system used to control the room temperature to a specified limit is a typical example for CONTROL purpose.

6. Application specific user interface

- Buttons, switches, keypad, lights, bells, display units etc are application specific user interfaces.
- Mobile phone is an example of application specific user interface.
- In mobile phone the user interface is provided through the keypad, system speaker, vibration alert etc.

1.8 REVIEW QUESTIONS

1. Define Embedded System with the help of Microwave Owen as an example
2. Differentiate between general purpose computers & embedded systems
3. Give a classification of embedded systems
4. List some applications of embedded systems
5. Explain the various possible purposes of using and embedded system.

1.9 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

2

ELEMENTS OF EMBEDDED SYSTEMS

Unit Structure

3.0 OBJECTIVES

After learning this chapter you will be able to:

1. Define and describe the elements of an embedded system
2. Understand how embedded system works with the help of two case studies:
    i. Washing Machine
    ii. Microwave Owen

3.1 INTRODUCTION

The previous chapter was an introduction to the world of embedded systems and helped us define what is an embedded system.

This chapter introduces us to the elements of an embedded system and explains how embedded system works with the help of two case studies.

3.2 ELEMENTS OF EMBEDDED SYSTEMS.

- As defined earlier, an embedded system is a combination of 3 things:
    d. Hardware
    e. Software
    f. Mechanical Components

  And it is supposed to do one specific task only.

Diagrammatically an embedded system can be represented as follows:

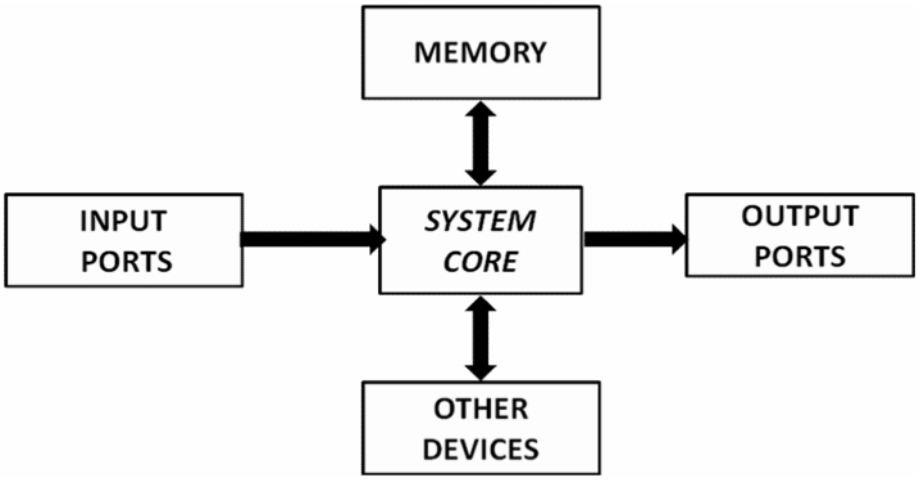


Figure 2.0 : Elements of an Embedded System

- Embedded systems are basically designed to regulate a physical variable (such Microwave Oven) or to manipulate the state of some devices by sending some signals to the actuators or devices connected to the output port system (such as temperature in Air Conditioner), in response to the input signal provided by the end users or sensors which are connected to the input ports.
- Hence the embedded systems can be viewed as a reactive system.
- Examples of common user interface input devices are keyboards, push button, switches, etc.
- The memory of the system is responsible for holding the code (control algorithm and other important configuration details).
- An embedded system without code (i.e. the control algorithm) implemented memory has all the peripherals but is not capable of making decisions depending on the situational as well as real world changes.
- Memory for implementing the code may be present on the processor or may be implemented as a separate chip interfacing the processor In a controller based embedded system, the controller may contain internal memory for storing code
- Such controllers are called Micro-controllers with on-chip ROM, eg. Atmel AT89C51.

2.3 CASE STUDIES (EXAMPLES)

Here are some case studies on some commonly used embedded systems that will help to better understand the concept.

2.3.1 Washing Machine

Let us see the important parts of the washing machine; this will also help us understand the working of the washing machine:

1) **Water inlet control valve**: Near the water inlet point of the washing there is water inlet control valve. When you load the clothes in washing machine, this valve gets opened automatically and it closes automatically depending on the total quantity of the water required. The water control valve is actually the solenoid valve.

2) **Water pump**: The water pump circulates water through the washing machine. It works in two directions, re-circulating the water during wash cycle and draining the water during the spin cycle.

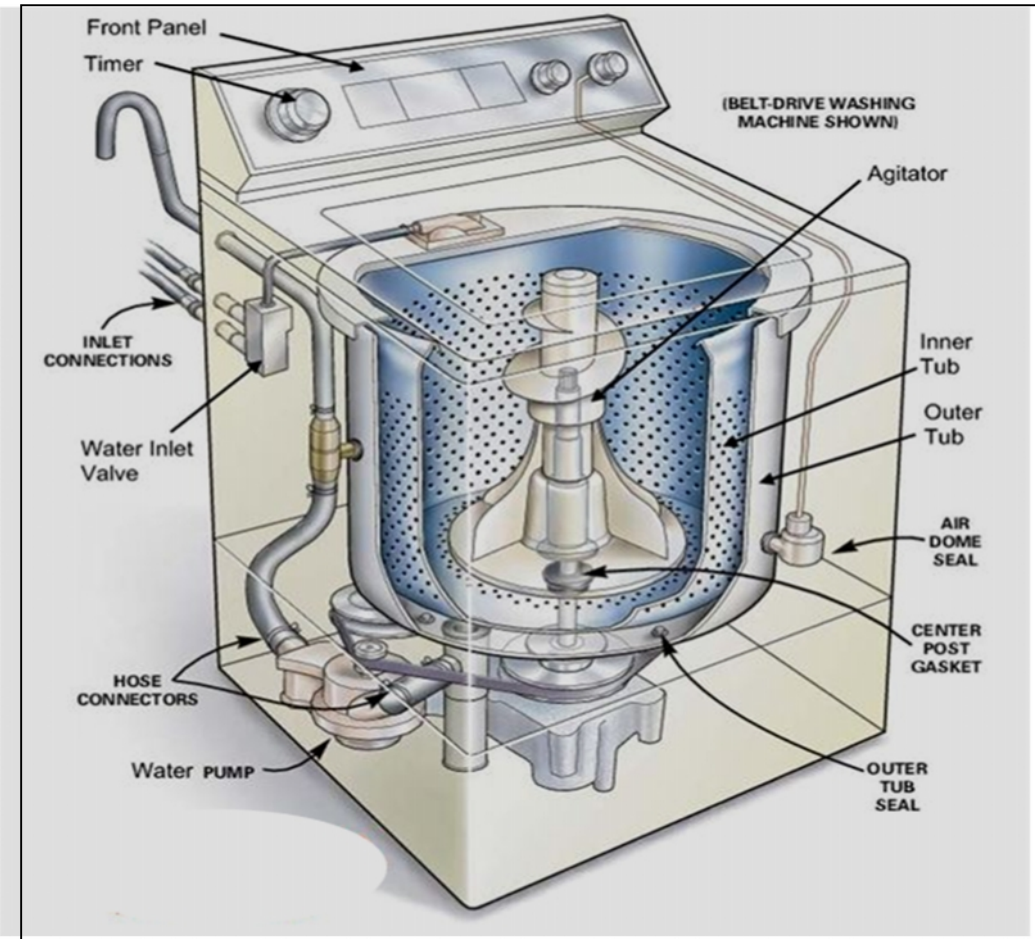


Figure 2.1 : Parts of a Washing Machine

3) **Tub**: There are two types of tubs in the washing washing machine: inner and outer. The clothes are loaded in the inner tub, where the clothes are washed, rinsed and dried. The inner tub has small holes for draining the water. The external tub covers theinner tub and supports it during various cycles of clothes washing.

4) **Agitator or rotating disc**: The agitator is located inside the tub of the washing machine. It is the important part of the washing machine that actually performs the cleaning operation of the clothes. During the wash cycle the agitator rotates continuously and produces strong rotating currents within the water due to which the clothes also rotate inside the tub. The rotation of the clothes within water containing the detergent enables the removal of the dirt particles from the fabric of the clothes. Thus the agitator produces most important function of rubbing the clothes with each other as well as with water.

   In some washing machines, instead of the long agitator, there is a disc that contains blades on its upper side. The rotation of the disc and the blades produce strong currents within the water and the rubbing of clothes that helps in removing the dirt from clothes.

5) **Motor of the washing machine**: The motor is coupled to the agitator or the disc and produces it rotator motion. These are multispeed motors, whose speed can be changed as per the requirement. In the fully automatic washing machine the speed of the motor i.e. the agitator changes automatically as per the load on the washing machine.

6) **Timer**: The timer helps setting the wash time for the clothes manually. In the automatic mode the time is set automatically depending upon the number of clothes inside the washing machine.

7) **Printed circuit board (PCB)**: The PCB comprises of the various electronic components and circuits, which are programmed to perform in unique ways depending on the load conditions (the condition and the amount of clothes loaded in the washing machine). They are sort of artificial intelligence devices that sense the various external conditions and take the decisions accordingly. These are also called as fuzzy logic systems. Thus the PCB will calculate the total weight of the clothes, and find out the quantity of water and detergent required, and the total time required for washing the clothes. Then they will decide the time required for washing and rinsing. The entire processing is done on a kind of processor which may be a microprocessor or microcontroller.

8) Drain pipe: The drain pipe enables removing the dirty water from the washing that has been used for the washing purpose.

2.3.2 Microwave Owen

Let us see the important parts of the microwave oven; this will also help us understand the working of the washing machine:

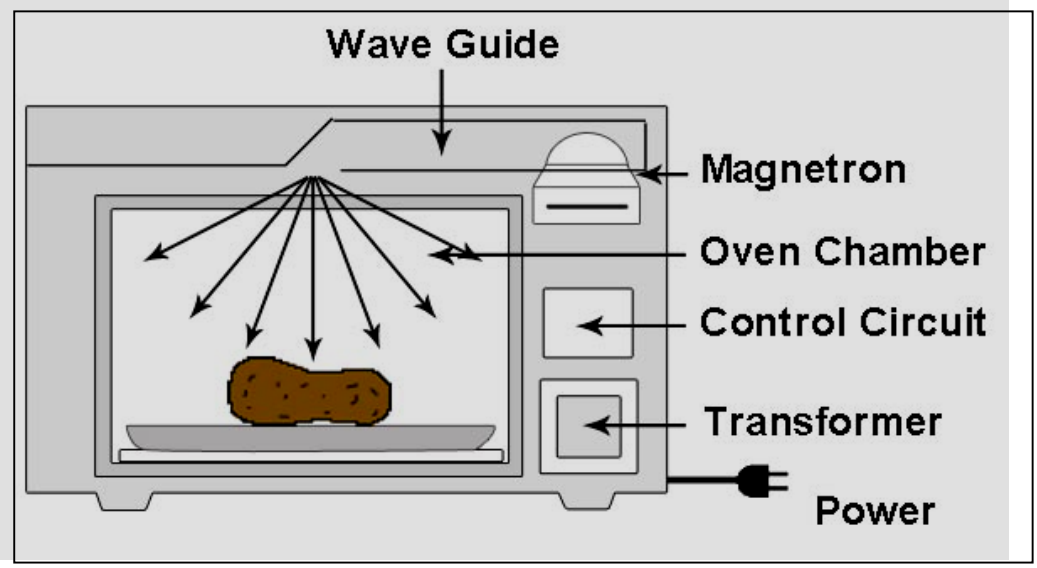


Figure 2.3 : Parts of a Microwave Owen

A microwave oven consists of:

1. A high voltage transformer, which passes energy to the magnetron
2. A cavity magnetron,
3. A Control circuit with a microcontroller,
4. A waveguide, and
5. A cooking chamber

1. A **Transformer** transfers electrical energy through a circuit by magnetic coupling without using motion between parts. These are used for supplying power to the magnetron.

2. A **Cavity magnetron** is a microwave antenna placed in a vacuum tube and oscillated in an electromagnetic field in order to produce high GHz microwaves. Magnetrons are used in microwave ovens and radar systems.

3. A **control circuit** with a microcontroller is integrated on a circuit board. The microcontroller controls the waveguide and the entire unit so the microwaves are emitted at a constant rate.

4. A **Waveguide** is any linear structure that guides electromagnetic waves for the purpose of transmitting power or

signals. Generally constructed of a hollow metal pipe. Placing a waveguide into a vacuum causes radio waves to scatter.

5. A **Cooking Chamber** is a microwave safe container the prevents microwaves from escaping. The door has a microwave proof mesh with holes that are just small enough that microwaves can't pass through but lightwaves can. The cooking chamber itself is a Faraday cage enclosure which prevents the microwaves from escaping into the environment. The oven door is usually a glass panel for easy viewing, but has a layer of conductive mesh to maintain the shielding.

2.3.3 Automotive Embedded System (AES)

- The Automotive industry is one of the major application domains of embedded systems.
- Automotive embedded systems are the one where electronics take control over the mechanical system. Ex. Simple viper control.
- The number of embedded controllers in a normal vehicle varies somewhere between 20 to 40 and can easily be between 75 to 100 for more sophisticated vehicles.
- One of the first and very popular use of embedded system in automotive industry was microprocessor based fuel injection.
- Some of the other uses of embedded controllers in a vehicle are listed below:
    a. Air Conditioner
    b. Engine Control
    c. Fan Control
    d. Headlamp Control
    e. Automatic break system control
    f. Wiper control
    g. Air bag control
    h. Power Windows
- AES are normally built around microcontrollers or DSPs or a hybrid of the two and are generally known as Electronic Control Units (ECUs).

- **Types Of Electronic Control Units(ECU)**

1. **<u>High-speed Electronic Control Units (HECUs)</u>:**
    a. HECUs are deployed in critical control units requiring fast response.
    b. They Include fuel injection systems, antilock brake systems, engine control, electronic throttle, steering controls, transmission control and central control units.
2. **<u>Low Speed Electronic Control Units (LECUs):-</u>**
    a. They are deployed in applications where response time is not so critical.
    b. They are built around low cost microprocessors and microcontrollers and digital signal processors.
    c. Audio controller, passenger and driver door locks, door glass control etc.

- **Automotive Communication Buses**

Embedded system used inside an automobile communicate with each other using serial buses. This reduces the wiring required.

Following are the different types of serial Interfaces used in automotive embedded applications:

a. <u>Controller Area Network (CAN):-</u>

- CAN bus was originally proposed by Robert Bosch.
- It supports medium speed and high speed data transfer
- CAN is an event driven protocol interface with support for error handling in data transmission.

b. <u>Local Interconnect Network (LIN):-</u>

- LIN bus is single master multiple slave communication interface with support for data rates up to 20 Kbps and is used for sensor/actuator interfacing

- LIN bus follows the master communication triggering to eliminate the bus arbitration problem
- LIN bus applications are mirror controls , fan controls , seat positioning controls

c. Media-Oriented System Transport(MOST):-

- MOST is targeted for automotive audio/video equipment interfacing
- A MOST bus is a multimedia fiber optics point–to-point network implemented in a star , ring or daisy chained topology over optical fiber cables.
- MOST bus specifications define the physical as well as application layer , network layer and media access control.

2.4 REVIEW QUESTIONS

1. What is an embedded system? What are the working elements of an embedded system?
2. Explain the working of embedded system with respect to:
   B. Washing Machine
   C. MICROWAVE Owen
3. Conduct case studies for working of embedded systems for the following topics:
   A. Air Conditioner
   B. Automobile

2.5 REFERENCES & FURTHER READING

Books:

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

Websites:

1. **Washing Machine:** http://www.brighthubengineering.com
2. **Microwave Owen :** http://globalmicrowave.org/microwaves.php

❖❖❖❖

3

CORE THE OF EMBEDDED SYSTEM

Unit Structure

3.0 OBJECTIVES

After reading this chapter you will be able to:

- ✓ Understand the different types of core i.e processor
- ✓ Understand difference between microprocessor & microcontroller
- ✓ Understand the classification of processors based on Bus Architecture, Instruction set Architecture and Endianness.
- ✓ Have an overview of processors from most simple and cheap to most expensive and complex, powerful
- ✓ Understand what are sensors and actuators, communication interfaces

3.1 INTRODUCTION

The first two chapters attempted on explain what an embedded system is about and what the working parts are. This chapter attempts to go deeper and explain the core of embedded system along with other related topics.

3.2 CORE OF EMBEDDED SYSTEMS

Embedded systems are domain and application specific and are built around a central core. The core of the embedded system falls into any of the following categories:

1. General purpose and Domain Specific Processors
   1.1. Microprocessors
   1.2. Microcontrollers
   1.3. Digital Signal Processors
2. Application Specific Integrated Circuits. (ASIC)
3. Programmable logic devices(PLD's)
4. Commercial off-the-shelf components (COTs)

3.2.1 GENERAL PURPOSE AND DOMAIN SPECIFIC PROCESSOR.

- Almost 80% of the embedded systems are processor/ controller based.

- The processor may be microprocessor or a microcontroller or digital signal processor, depending on the domain and application.

3.2.1.1 MICROPROCESSORS

- A microprocessor is a silicon chip representing a central processing unit.

- A microprocessor is a dependent unit and it requires the combination of other hardware like memory, timer unit, and *interrupt* controller, etc. for proper functioning.

- Developers of microprocessors.
  - Intel – *Intel 4004* – November 1971(4-bit).
  - Intel – *Intel 4040.*
  - Intel – *Intel 8008* – April 1972.
  - Intel – *Intel 8080* – April 1974(8-bit).
  - Motorola – *Motorola 6800.*
  - Intel – *Intel 8085* – 1976.
  - Zilog - *Z80* – July 1976.

- Architectures used for processor design are Harvard or Von-Neumann.

Harvard architecture	Von-Neumann architecture
⊡ It has separate buses for instruction as well as data fetching.	⊡ It shares single common bus for instruction and data fetching.
⊡ Easier to pipeline, so high performance can be achieve.	⊡ Low performance as compared to Harvard architecture.
⊡ Comparatively high cost.	⊡ It is cheaper.
⊡ Since data memory and program memory are stored physically in different locations, no chances exist for accidental corruption of program memory.	⊡ Accidental corruption of program memory may occur if data memory and program memory are stored physically in the same chip,

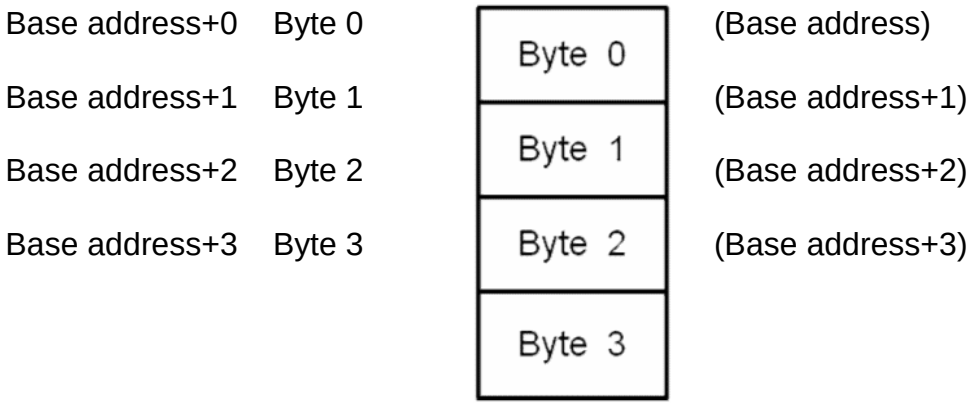
- RISC and CISC are the two common Instruction Set Architectures (ISA) available for processor design.

RISC	CISC
⊡ Reduced Instruction Set Computing	⊡ Complex Instruction Set Computing
⊡ It contains lesser number of instructions.	⊡ It contains greater number of instructions.
⊡ Instruction pipelining and increased execution speed.	⊡ Instruction pipelining feature does not exist.
⊡ Orthogonal instruction set(allows each instruction to operate on any register and use any addressing mode.	⊡ Non-orthogonal set(all instructions are not allowed to operate on any register and use any addressing mode.
⊡ Operations are performed on registers only, only memory operations are load and store.	⊡ Operations are performed either on registers or memory depending on instruction.
⊡ A larger number of registers are available.	⊡ The number of general purpose registers are very limited.
⊡ Programmer needs to write more code to execute a task since instructions are simpler ones.	⊡ Instructions are like macros in C language. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instruction in RISC.
⊡ It is single, fixed length instruction.	⊡ It is variable length instruction.

⊡ Less silicon usage and pin count.	⊡ More silicon usage since more additional decoder logic is required to implement the complex instruction decoding.
⊡ With Harvard Architecture.	⊡ Can be Harvard or Von-Neumann Architecture.

- **Endiannes**
  - Endianness specifies the order which the data is stored in the memory by processor operations in a multi byte system.
  - Based on Endiannes processors can be of two types:
    1. Little Endian Processors
    2. Big Endian Processors

1. Little-endian means lower order data byte is stored in memory at the lowest address and the higher order data byte at the highest address. For e.g, 4 byte long integer Byte3, Byte2, Byte1, Byte0 will be store in the memory as follows:



2. Big-endian means the higher order data byte is stored in memory at the lowest and the lower order data byte at the highest address. For e.g. a 4 byte integer Byte3, Byte2, Byte1, Byte0 will be stored in the memory as follows:

Base address+0	Byte 3	Byte 3	(Base address)
Base address+1	Byte 2	Byte 2	(Base address+1)
Base address+2	Byte 1	Byte 1	(Base address+2)
Base address+3	Byte 0	Byte 0	(Base address+3)

3.2.1.2 MICROCONTROLLERS.

- A microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, special and general purpose register arrays,on chip ROM/FLASH memory for program storage , timer and interrupt control units and dedicated I/O ports.

- Texas Instrument's TMS 1000 Is considered as the world's first microcontroller.

- Some embedded system application require only 8 bit controllers whereas some requiring superior performance and computational needs demand 16/32 bit controllers.

- The instruction set of a microcontroller can be RISC or CISC.

- Microcontrollers are designed for either general purpose application requirement or domain specific application requirement.

3.2.1.3 Digital Signal Processors

- DSP are powerful special purpose 8/16/32 bit microprocessor designed to meet the computational demands and power constraints of today's embedded audio, video and communication applications.

- DSP are 2 to 3 times faster than general purpose microprocessors in signal processing applications. This is because of the architectural difference between DSP and general purpose microprocessors.

- DSPs implement algorithms in hardware which speeds up the execution whereas general purpose processor implement the algorithm in software and the speed of execution depends primarily on the clock for the processors.

- DSP includes following key units:

i. **Program memory:** It is a memory for storing the program required by DSP to process the data.

ii. **Data memory:** It is a working memory for storing temporary variables and data/signal to be processed.

iii. **Computational engine:** It performs the signal processing in accordance with the stored program memory computational engine incorporated many specialized arithmetic units and each of them operates simultaneously to increase the execution speed. It also includes multiple hardware shifters for shifting operands and saves execution time.

iv. **I/O unit:** It acts as an interface between the outside world and DSP. It is responsible for capturing signals to be processed and delivering the processed signals.

- Examples: Audio video signal processing, telecommunication and multimedia applications.

- SOP(Sum of Products) calculation, convolution, FFT(Fast Fourier Transform), DFT(Discrete Fourier Transform), etc are some of the operation performed by DSP.

3.2.2 Application Specific Integrated Circuits. (ASIC)

- ASICs is a microchip design to perform a specific and unique applications.

- Because of using single chip for integrates several functions there by reduces the system development cost.

- Most of the ASICs are proprietary (which having some trade name) products, it is referred as Application Specific Standard Products(ASSP).

- As a single chip ASIC consumes a very small area in the total system. Thereby helps in the design of smaller system with high capabilities or functionalities.

- The developers of such chips may not be interested in revealing the internal detail of it .

3.2.3 Programmable logic devices(PLD's)

- A PLD is an electronic component. It used to build digital circuits which are reconfigurable.

- A logic gate has a fixed function but a PLD does not have a defined function at the time of manufacture.

- PLDs offer customers a wide range of logic capacity, features, speed, voltage characteristics.

- PLDs can be reconfigured to perform any number of functions at any time.

- A variety of tools are available for the designers of PLDs which are inexpensive and help to develop, simulate and test the designs.

- PLDs having following two major types.

1) CPLD(Complex Programmable Logic Device):

CPLDs offer much smaller amount of logic up to 1000 gates.

2) FPGAs(Field Programmable Gate Arrays):

It offers highest amount of performance as well as highest logic density, the most features.

- **Advantages of PLDs :-**

1) PLDs offer customer much more flexibility during the design cycle.

2) PLDs do not require long lead times for prototypes or production parts because PLDs are already on a distributors shelf and ready for shipment.

3) PLDs can be reprogrammed even after a piece of equipment is shipped to a customer

3.2.4 Commercial off-the-shelf components(COTs)

1) A Commercial off the Shelf product is one which is used 'as-is'.

2) The COTS components itself may be develop around a general purpose or domain specific processor or an ASICs or a PLDs.

3) The major advantage of using COTS is that they are readily available in the market, are chip and a developer can cut down his/her development time to a great extent

4) The major drawback of using COTS components in embedded design is that the manufacturer of the COTS component may withdraw the product or discontinue the production of the COTS at any time if rapid change in technology occurs.

5) **Advantages of COTS:**
    1) Ready to use
    2) Easy to integrate
    3) Reduces development time

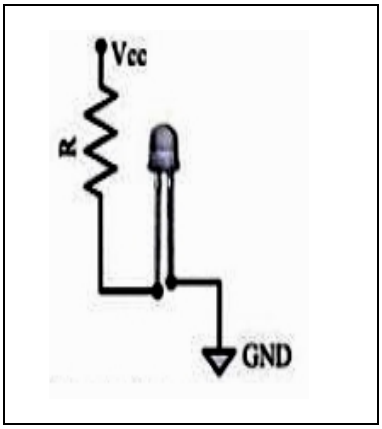
6) **Disadvantages of COTS:**
    1) No operational or manufacturing standard (all proprietary)

    2) Vendor or manufacturer may discontinue production of a particular COTS product

3.3 SENSORS & ACTUATORS

- **Sensor**
    - A Sensor is used for taking Input
    - It is a transducer that converts energy from one form to another for any measurement or control purpose
    - Ex. A Temperature sensor

- **Actuator**
    - Actuator is used for output.
    - It is a transducer that may be either mechanical or electrical which converts signals to corresponding physical actions.
    - Ex. LED (Light Emitting Diode)
    - LED is a p-n junction diode and contains a **CATHODE** and **ANODE**
    - For functioning the anode is connected to +ve end of power supply and cathode is connected to –ve end of power supply.
    - The maximum current flowing through the LED is limited by connecting a RESISTOR in series between the power supply and LED as shown in the figure below



- There are two ways to interface an LED to a microprocessor/microcontroller:

1. **The Anode of LED is connected to the port pin and cathode to Ground :** In this approach the port pin sources the current to the LED when it is at logic high(ie. 1).

2. **The Cathode of LED is connected to the port pin and Anode to Vcc :** In this approach the port pin sources the current to the LED when it is at logic high (ie. 1). Here the port pin sinks the current and the LED is turned ON when the port pin is at Logic low (ie. 0)

3.4 COMMUNICATION INTERFACES

For any embedded system, the communication interfaces can broadly classified into:

1. **Onboard Communication Interfaces**
   - These are used for internal communication of the embedded system i.e: communication between different components present on the system.

   - Common examples of onboard interfaces are:
     - Inter Integrated Circuit (I2C)
     - Serial Peripheral Interface (SPI)
     - Universal Asynchronous Receiver Transmitter (UART)
     - 1-Wire Interface
     - Parallel Interface

   - Example :**Inter Integrated Circuit (I2C)**
     - It is synchronous
     - Bi-directional, half duplex , two wire serial interface bus
     - Developed by Phillips semiconductors in 1980

- It comprises of two buses :
  1. Serial clock –SCL
  2. Serial Data – SDA
- SCL generates synchronization clock pulses
- SDA transmits data serially across devices
- I2C is a shared bus system to which many devices can be connected
- Devices connected by I2C can act as either master or slave
- The master device is responsible for controlling communication by initiating/ terminating data transfer.
- Devices acting as slave wait for commands from the master and respond to those commands.

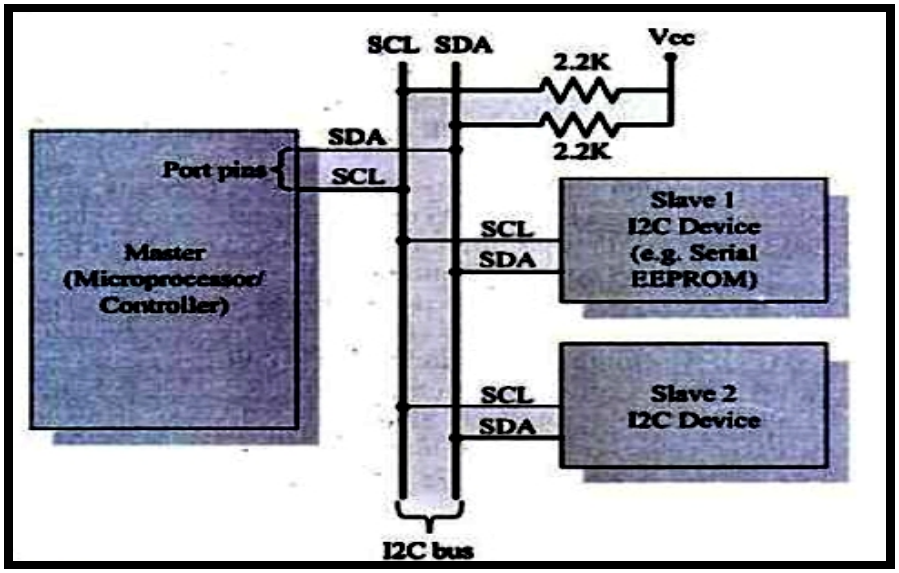


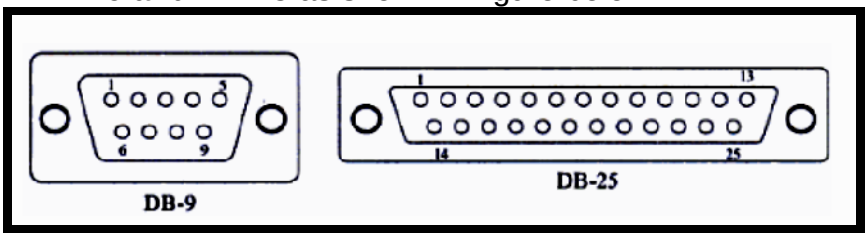
Figure: I2C Bus Interfacing

2. External or Peripheral Communication Interfaces

- These are used for external communication of the embedded system i.e: communication of different components present on the system with external or peripheral components/devices.

- Common examples of external interfaces are:
  - RS-232 C & RS-485
  - Universal Serial Bus (USB)
  - IEEE 1394 (Firewire)
  - Infrared (IrDA)
  - Bluetooth
  - Wi-Fi
  - Zig Bee
  - General Packet Radio Service (GPRS)
  - **Example: RS-232 C & RS-485**

- It is wired, asynchronous, serial, full duplex communication
- RS 232 interface was developed by EIA (Electronic Industries Associates) In early 1960s
- RS 232 is the extension to UART for external communications
- RS-232 logic levels use:
- +3 to +25 volts to signify a "Space" (Logic 0) and
- -3 to -25 volts to signify a "Mark" (logic 1).
- RS 232 supports two different types of connectors :
- **DB 9 and DB 25** as shown in figure below



- RS 232 interface is a point to point communication interface and the devices involved are called as Data Terminating Equipment (DTE) And Data Communications Terminating Equipment (DCE)
- Embedded devices contain UART for serial transmission and generate signal levels as per TTL/CMOS logic.
- A level translator IC (like Max 232) is used for converting the signal lines from UART to RS 232 signal lines for communication.
- The vice versa is performed on the receiving side.
- Converter chips contain converters for both transmitters and receivers
- RS 232 is used only for point to point connections
- It is susceptible to noise and hence is limited to short distances only
- RS 422 is another serial interface from EIA.
- It supports multipoint connections with 1 transmitter and 10 receivers.
- It supports data rates up to 100Kbps and distance up to 400 ft
- RS 485 is enhanced version of RS 422 and supports up to 32 transmitters and 32 receivers

3.5 REVIEW QUESTIONS

1. What do you mean by core of the embedded system? What is its significance? What are the possible options that can be used as a core?
2. Distinguish between Microprocessor & Microcontroller

3. Explain the different types of processors according to their system bus architecture
4. Explain the different types of processors according to Instruction set Architecture
5. Explain the different types of processors according to Endianness
6. Write short note on :
   i. DSP
   ii. PLD
   iii. ASIC
   iv. COTS
7. Explain Communication Interfaces with respect to embedded system
8. Explain the following with example:
   1. Onboard communication interface
   2. Peripheral communication interface
9. Find out information and write case studies on the following communication interfaces:
   i. Infrared
   ii. WiFi
   iii. Zigbee
   iv. UART

3.6 REFERENCES & FURTHER READING

2. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

4

CHARACTERISTICS & QUALITY ATTRIBUTES OF EMBEDDED SYSTEMS

Unit Structure

4.0 OBJECTIVES

After reading this chapter you will:

1. Understand the characteristics of Embedded system
2. Understand the attributes related to quality of embedded system.

4.1 INTRODUCTION

The characteristics of embedded system are different from those of a general purpose computer and so are its Quality metrics. This chapter gives a brief introduction on the characteristics of an embedded system and the attributes that are associated with its quality.

4.2 CHARACTERISTICS OF EMBEDDED SYSTEM

Following are some of the characteristics of an embedded system that make it different from a general purpose computer:

1. Application and Domain specific

- An embedded system is designed for a specific purpose only. It will not do any other task.
- Ex. A washing machine can only wash, it cannot cook
- Certain embedded systems are specific to a domain: ex. A hearing aid is an application that belongs to the domain of signal processing.

2. **Reactive and Real time**
   - Certain Embedded systems are designed to react to the events that occur in the nearby environment. These events also occur real-time.
   - Ex. An air conditioner adjusts its mechanical parts as soon as it gets a signal from its sensors to increase or decrease the temperature when the user operates it using a remote control.
   - An embedded system uses Sensors to take inputs and has actuators to bring out the required functionality.

3. **Operation in harsh environment**
   - Certain embedded systems are designed to operate in harsh environments like very high temperature of the deserts or very low temperature of the mountains or extreme rains.
   - These embedded systems have to be capable of sustaining the environmental conditions it is designed to operate in.

4. **Distributed**
   - Certain embedded systems are part of a larger system and thus form components of a distributed system.
   - These components are independent of each other but have to work together for the larger system to function properly.
   - Ex. A car has many embedded systems controlled to its dash board. Each one is an independent embedded system yet the entire car can be said to function properly only if all the systems work together.

5. **Small size and weight**
   - An embedded system that is compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.
   - Ex. Currently available cell phones. The cell phones that have the maximum features are popular but also their size and weight is an important characteristic.
   - For convenience users prefer mobile phones than phablets. (phone + tablet pc)

6. **Power concerns**
   - It is desirable that the power utilization and heat dissipation of any embedded system be low.

- If more heat is dissipated then additional units like heat sinks or cooling fans need to be added to the circuit.

- If more power is required then a battery of higher power or more batteries need to be accommodated in the embedded system.

4.3 QUALITY ATTRIBUTES OF EMBEDDED SYSTEM

These are the attributes that together form the deciding factor about the quality of an embedded system.

There are two types of quality attributes are:-

1. Operational Quality Attributes.

- These are attributes related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.

2. Non-Operational Quality Attributes.

- These are attributes **not** related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.

- These are the attributes that are associated with the embedded system before it can be put in operation.

4.3.1 Operational Attributes

a) Response

- Response is a measure of quickness of the system.
- It gives you an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.

b) Throughput

- Throughput deals with the efficiency of system.
- It can be defined as rate of production or process of a defined process over a stated period of time.
- In case of card reader like the ones used in buses, throughput means how much transaction the reader can perform in a minute or hour or day.

c) Reliability

- Reliability is a measure of how much percentage you rely upon the proper functioning of the system .

- Mean Time between failures and Mean Time To Repair are terms used in defining system reliability.
- Mean Time between failures can be defined as the average time the system is functioning before a failure occurs.
- Mean time to repair can be defined as the average time the system has spent in repairs.

d) Maintainability

- Maintainability deals with support and maintenance to the end user or a client in case of technical issues and product failures or on the basis of a routine system checkup
- It can be classified into two types :-

1. Scheduled or Periodic Maintenance

- o This is the maintenance that is required regularly after a periodic time interval.
- o Example :
  - Periodic Cleaning of Air Conditioners
  - Refilling of printer cartridges.

2. Maintenance to unexpected failure

- This involves the maintenance due to a sudden breakdown in the functioning of the system.
- Example:
  1. Air conditioner not powering on
  2. Printer not taking paper in spite of a full paper stack

e) Security

- Confidentiality, Integrity and Availability are three corner stones of information security.
- Confidentiality deals with protection data from unauthorized disclosure.
- Integrity gives protection from unauthorized modification.
- Availability gives protection from unauthorized user
- Certain Embedded systems have to make sure they conform to the security measures. Ex. An Electronic Safety Deposit Locker can be used only with a pin number like a password.

f) Safety

- Safety deals with the possible damage that can happen to the operating person and environment due to the breakdown of an embedded system or due to the emission of hazardous materials from the embedded products.
- A safety analysis is a must in product engineering to evaluate the anticipated damage and determine the best

course of action to bring down the consequence of damages to an acceptable level.

4.3.2 Non Operational Attributes

a) Testability and Debug-ability

- It deals with how easily one can test his/her design, application and by which mean he/she can test it.
- In hardware testing the peripherals and total hardware function in designed manner
- Firmware testing is functioning in expected way
- Debug-ability is means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system

b) Evolvability

- For embedded system, the qualitative attribute “Evolvability” refer to ease with which the embedded product can be modified to take advantage of new firmware or hardware technology.

c) Portability

- Portability is measured of “system Independence”.
- An embedded product can be called portable if it is capable of performing its operation as it is intended to do in various environments irrespective of different processor and or controller and embedded operating systems.

d) Time to prototype and market

- Time to Market is the time elapsed between the conceptualization of a product and time at which the product is ready for selling or use
- Product prototyping help in reducing time to market.
- Prototyping is an informal kind of rapid product development in which important feature of the under consider are develop.
- In order to shorten the time to prototype, make use of all possible option like use of reuse, off the self component etc**.**

e) Per unit and total cost

- Cost is an important factor which needs to be carefully monitored. Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.
- When the product is introduced in the market, for the initial period the sales and revenue will be low
- There won’t be much competition when the product sales and revenue increase.

- During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.

4.4 REVIEW QUESTIONS

1. Explain the characteristics of an embedded system
2. Explain the Operational Quality Attributes of an embedded system
3. Explain the non quality attributes of an embedded system

4.5 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

5

PROGRAM FOR EMBEDDED SYSTEMS AND BUILD PROCESS

Unit Structure

5.0 OBJECTIVES

After reading this chapter you will be able to:

- ✓ Know the difficulties involved in programming embedded systems
- ✓ Reasons for not implementing Hello World as a first program
- ✓ Write a generic code for Blinking LED
- ✓ Infinite loop
- ✓ Build Process in embedded system

5.1 INTRODUCTION

This chapter gives a head start into programming the embedded system. The embedded code is just like any other code but there are several restrictions. For Example: the code that runs on a computer makes certain assumptions about the available memory while in embedded system there is no scope for assumptions. The Embedded System Programmer has to know the hardware before he can even attempt to write any code.

This chapter introduces to the readers the Blinking LED program as a first program in embedded systems and explains why the hello world would be rather a difficult program.

Then an essential necessity of the code is explained: the infinite loop.

5.2 STARTING WITH EMBEDDED PROGRAMMING

- Programmers for Embedded systems must be self-reliant. The usual built in functions (i.e. standard library routines like cin and cout) may not be available to them.

- Further, programmers for embedded systems have to know beforehand what hardware is involved.

  Example: They should know which processor will be used, how much memory is available and what the memory offsets are. All this information will change if the underlying hardware changes.

- Embedded system programming has no scope for assumption.

  Example: Memory is a very limited hence precious resource in an embedded system. A large amount of memory for code as well as processing data would mean adding more memory to the circuit in the form of additional memory. A simple variable declaration like an integer (usually 2 bytes or 4 bytes) should be thought carefully before declaring it. Declaring the same variable as unsigned integer (1 byte) will suffice the purpose as well as save memory location.

- Not many programmers are available for programming of embedded systems.

5.3 “HELLO WORLD” FOR EMBEDDED SYSTEMS

- Every book on programming language begins with the example that prints "Hello, World!" on the output screen. The hello world is a simple program that does not involve any logic and can be implemented by a no-brainer (beginner).

- However, in embedded systems, the hello world program would be a bad choice to be implemented as a first program. In case of any programming language the hello world program is executed on a general purpose computer where we have requirements like CPU, input and output devices and interaction between them taken care by the software and the operating system.

- In case of embedded system there may be no operating system at all. Certain embedded systems may not even have an output device like a monitor. Embedded systems usually have LED Displays. To incorporate a display device in an embedded system the programmer would involve writing a piece of code

called a device driver which otherwise is taken care by the operating system in case of a general purpose computer.

- Writing a code for a device driver is not an easy job at beginner's level. Hence hello world is a difficult program to begin with while learning to program embedded system.

5.4 BLINKING LED PROGRAM AND INFINITE LOOP

- The substitute for hello world program could be a program that blinks an LED. LEDs are used in almost every embedded system. Also the code used to program an LED would be very small.

- To blink an LED we would require the following hardware:
  - LED
  - A microcontroller or microprocessor

- The LED can be connected to any available port i.e P1, P2, P3, P4 on the microprocessor. Assuming the LED is connected to port 2, i.e. P2 its state is controlled by a bit in a register called the Port 2 I/O Latch Register, also known the P2LTCH.

- The structure of the program would be like this:

```
void main()
{
    while (1)
    {
        toggle(LED_1);      /* Change the
state of the LED.    */
        delay(500);         /* Pause for 500
milliseconds.     */
    }
}
```

- The above piece of code is hardware independent hence can be implemented for any circuit.
- It contains two functions namely : toggle() & delay()
  - **toggle():** This function is used to toggle the state of the LED.
  - **delay():** This function is used to introduce a delay of 500 ms every time the LED is toggled
- The implementation of toggle() and delay() is hardware specific.
- **Infinite Loop**

- The code for every embedded program is written in an infinite loop. This is because the embedded system is supposed to run every time it is turned on till the time its power goes off or it stops functioning.

- The code for blinking LED is also enclosed in an infinite loop. The functions toggle() and delay() run infinite number of times.

- An application of an embedded system has an infinite loop around its code. It's just like the program you did to implement switch case where the program has to run continuously until the user selects to exit.

5.5 BUILD PROCESS IN EMBEDDED SYSTEM

- **Definition:** The process which converts source code to executable code is called as the build process.

- The build process for embedded systems is different. This is because the code to be run on an embedded system is written one platform i.e. general purpose computer and executed on another platform i.e. the target hardware.

- An Embedded system would also use tools such as a Compiler, Linker, Locater and Debugger to perform the entire build process. These tools would be a part of a larger IDE.

- A compiler which produces the executable code to be run on a different platform is called a cross-compiler; else it is called a native compiler.

- Ex. Turbo C++ is a native compiler. The compiler in case of embedded systems development is a cross compiler.

- The build process involves three steps:
    1. Compiling
    2. Linking
    3. Locating

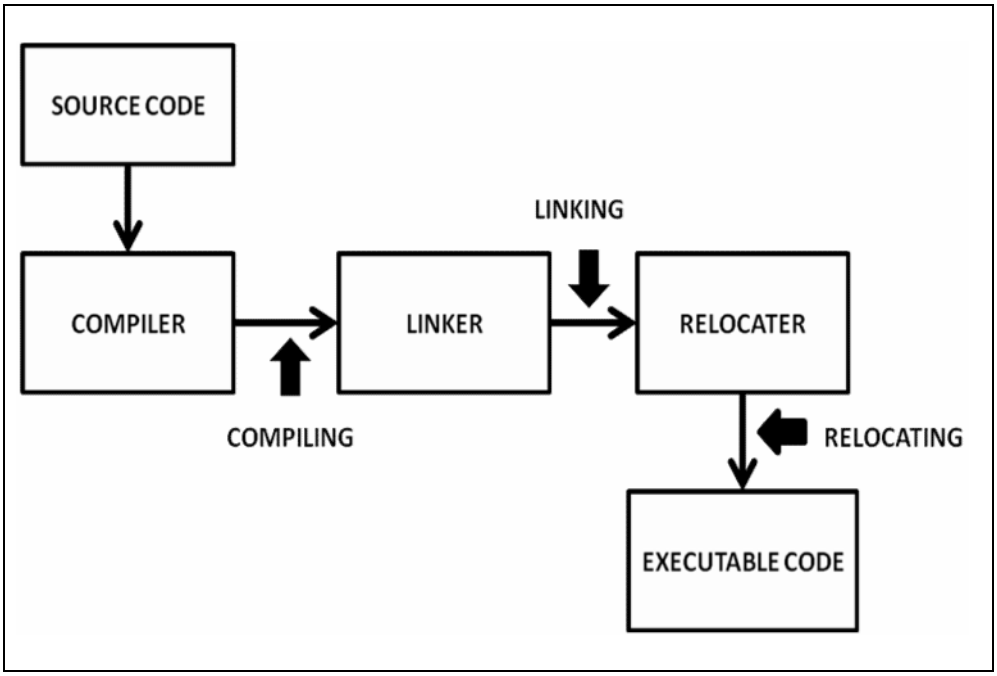


Figure: Build Process in Embedded System

- **Compiling**
  - The process of compiling is done by the compiler.
  - The compiler takes input as source code files and gives output as multiple object files.
  - Compilers for embedded systems are essentially cross-compilers. For example while compiling the programmer has to select the target processor for which the code has to be generated.
  - The contents of the object files depend on its format. Two commonly used formats are: 1. Common Object file format (COFF)
    2. Extended file format (ELF)
  - Object files generally have the following structure:

header	It describes the sections that will be contained in the object file.
text	It contains all code blocks
data	It contains all initialized global variables and their values
bss	It contains all initialized global variables.

- **Linking**
  - The process of linking is carried out by the linker
  - The linker takes input as multiple object files and gives output as a single object file which is also called as the relocatable code.

- The output of compiler is multiple object files. These files are incomplete in the sense that they may contain reference to variables and functions across multiple object files which need to be resolved.
- The job of the linker is to combine these multiple object files and resolve the unresolved symbols.
- The Linker does this by merging the various sections like text, data, and bss of the individual object files. The output of the linker will be a single file which contains all of the machine language code from all of the input object files that will be in the text section of this new file, and all of the initialized and uninitialized variables will reside in the new data section and bss section respectively.

- **Locating**
  - The process of relocating is carried out by the relocater.
  - The relocater takes input as the relocatable code produced by the linker and gives output as the final executable code.
  - This output is a binary executable file which is called hex code.
  - The locator needs to be given information about the memory available on the target processor.
  - The locator will use this information to assign physical memory addresses to each of the code and data sections within the relocatable program code. Finally it produces an output file that contains a binary memory image that can be loaded into the target processors ROM.

5.6 REVIEW QUESTIONS

1. What are the difficulties involved in programming embedded systems?
2. Why can't a simple program like printing "hello world" on the screen be used as a starting program in embedded system?
3. Explain the Blinking LED program structure for embedded system
4. Explain the build process for embedded system
5. Write a short note on Infinite Loop.

5.7 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

6

DEBUGGING ON EMBEDDED SYSTEMS

Unit Structure

6.0 OBJECTIVES

- After reading this chapter you will understand:
- Concept of downloading the embedded code
- Debugging the embedded software
- Different possible tools available for debugging
- Difference between Remote Debugger, Emulator & Simulator

6.1 INTRODUCTION

In the previous chapter we saw how the code or software to be executed on the embedded system (target board) is written on a computer. The resulting code created after subjecting it to be build process is called the *binary executable image* or simply *hex code*.

This chapter explains how the hex code is put on the target board which is referred as downloading and what are the various possible ways of debugging a code meant to run on a embedded system.

6.2 DOWNLOADING THE EMBEDDED CODE

The code to be run on the target embedded system is always developed on the host computer. This code is called the *binary executable image* or simply *hex code*.

The process of putting this code in the memory chip of the target embedded system is called Downloading.

There are two ways of downloading the binary image on the embedded system:

1. Using a Device Programmer

A device programmer is a piece of hardware that works in two steps.

Step 1 Once the binary image is ready on the computer, the device programmer is connected to the computer and the binary image is transferred to the device programmer.

Step 2 The microcontroller/microprocessor or memory chip, usually the ROM which is supposed to contain the binary image is placed on the proper socket on the device programmer. The device programmer contains a software interface through which the user selects the target microprocessor for which the binary image has to be downloaded. The Device programmer then transfers the binary image bit by bit to the chip.

2. Using In System Programmer(ISP)

Certain Target embedded platforms contain a piece of hardware called ISP that have a hardware interface to both the computer as well the chip where the code is to be downloaded.

The user through the ISP's software interface sends the binary image to the target board.

This avoids the requirement of frequently removing the microprocessor / microcontroller or ROM for downloading the code if a device programmer had to be used.

6.3 DEBUGGING THE EMBEDDED SOFTWARE

- Debugging is the process of eliminating the bugs/errors in software.
- The software written to run on embedded systems may contain errors and hence needs debugging.
- However, the difficulty in case of embedded systems is to find out the bug/ error itself. This is because the binary image you downloaded on the target board was free of syntax errors but

still if the embedded system does not function the way it was supposed to be then it can be either because of a hardware problem or a software problem. Assuming that the hardware is perfect all that remains to check is the software.

- The difficult part here is that once the embedded system starts functioning there is no way for the user or programmer to know the internal state of the components on the target board.

- The most primitive method of debugging is using LEDs. This is similar to using a printf or a cout statement in c/c++ programs to test if the control enters the loop or not. Similarly an LED blind or a pattern of LED blinks can be used to check if the control enters a particular piece of code.

- There are other advanced debugging tools like;
    a. Remote debugger
    b. Emulator
    c. Simulator

6.3.1 Remote Debuggers

- Remote Debugger is a tool that can be commonly used for:
    - Downloading
    - Executing and
    - Debugging embedded software
- A Remote Debugger contains a hardware interface between the host computer and the target embedded system.

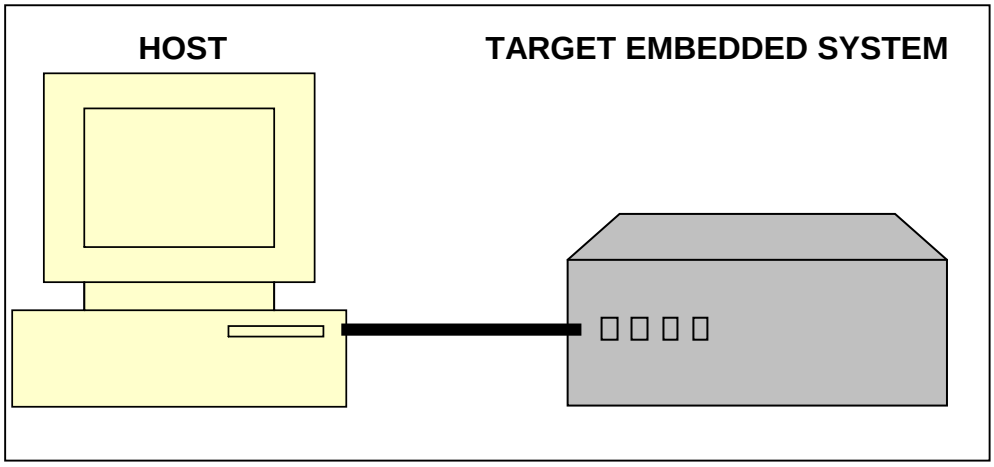


Figure: Remote Debugger

- The Software interface of the remote debugger has GUI-based main window and several smaller windows for the source code, register contents and other information about the executing program.

- It contains two pieces of software :
  - **Frontend remote debugger**
    - It runs on the host computer.
    - It provides the human interface.

  - **Backend remote debugger**
    - Backend remote debugger runs on the target processor.
    - It communicates with the frontend over a communications link of some sort.
    - It provides for low-level control of the target processor and is usually called the debug monitor.

    - **Debug monitor** is a piece of software that has been designed specifically for use as a debugging tool for processors and chips.
    - It is automatically started whenever the processor is reset.
    - It monitors the communication link to the host computer and responds to requests from the remote debugger running there.
    - One such debugger is the GNU.
    - It was originally designed for native debugger.
    - It performs cross-debugging.
    - Communication between the GDB frontend and debug monitor is byte-oriented and designed for transmission over a serial connection.

6.3.2 Emulators

- A Remote debugger is helpful for monitoring and controlling the state of embedded software prior to downloading it only.

- An Emulator allows you to examine the state of the processor on which that program is actually running. It is itself an embedded system, with its own copy of the target processor, RAM, ROM, and its own embedded software

- An Emulator takes the place of-or emulates-the processor on the target board.

- Emulator uses a remote debugger for its human interface.

- Emulator supports such powerful debugging features such as hardware breakpoints and real-time tracing. Hardware breakpoints allow you to stop execution in response to a wide

variety of events. These events include instruction fetches, memory and I/O reads and writes and interrupts. Real Time tracing allows you to see the exact order in which events occurred, so it can help you answer questions related to specific errors.

- **ROM Emulator**
  - It is a device that emulates a read only memory device like ICE (in-circuit emulator).
  - It connects to the target embedded system and communicates with the host.
  - When a target connection is via a ROM socket to embedded system it looks like any other read only memory. But when it is to the remote debugger it looks like a debug monitor.

- **Advantages:**
  - There is no need to port the debug monitor code to particular target hardware.
  - The ROM emulator supplies its own serial or network connection to the host
  - The ROM emulator is a true replacement for the original ROM, so none of the target's memory is used up by the debug monitor code

6.3.3 Simulators

- A simulator is a completely host-based program that simulates the functionality and instructions set of the target processor.
- Advantage: A Simulator can be quite valuable in the earlier stage of a project when there has not yet been any actual hardware implementation for the programmers to experiment with.
- Disadvantage: One of the disadvantages of simulator is that it only simulates the processors.

6.4 OTHER TOOLS

Logic Analyzers and Oscilloscopes are very important debugging tools.

- **Logic Analyzers**
  - It is a piece of laboratory equipment that is designed especially for troubleshooting digital hardware.

  - It can have multiple inputs (up to 100 even), each capable of detecting whether the electrical signal it is attached to is currently at logic level 1 or 0

- An oscilloscope is another pieces of laboratory equipment of hardware debugging. But this one is used to examine any electrical signal, analog or digital, on any piece of hardware

❑ **Oscilloscopes**

- An oscilloscope is another pieces of laboratory equipment of hardware debugging. But this one is used to examine any electrical signal, analog or digital, on any piece of hardware

- Oscilloscope are sometimes useful for quickly observing the voltage on the particular pin or, in the absence of a logic analyzer, for something ,more complex

6.5 REVIEW QUESTIONS

1. Explain the process of Downloading embedded software code
2. Write short note on debugging the embedded software code
3. Explain the working of Remote Debugger
4. Explain the working of Emulator
5. Explain the use of Simulators, Logic analyzers and Oscilloscopes in debugging embedded systems.

6.6 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. 2. Introduction to Embedded systems – Shibu K. V

❖❖❖❖

7

EMBEDDED HARDWARE FROM SOFTWARE PROGRAMMERS PERSPECTIVE

Unit Structure

7.0 OBJECTIVES

After reading this chapter you will be able to:

- Understand in general the difference in programming software for general purpose computers and embedded systems
- The way in which processor communicates with components of embedded system
- Memory Map, I/O Map & Interrupt Map

7.1 INTRODUCTION

The software programmer must know the hardware involved in an embedded system before he can even attempt to write code for its functioning.

Programming for embedded systems is different than programming on computers. Embedded systems have certain strict assumptions to be followed. Until the programmer does not know what hardware components are involved and what are the assumptions and rules related to those components, the program or code cannot be written.

This chapter introduces the reader with the hardware of embedded system from a software perspective. It is this where the

reader shall understand where the code fits in an embedded system.

7.2 COMPONENTS ON AN EMBEDDED SYSTEM

- Before the programmer can start to code anything, he has to invest some time in understand the functioning of the embedded system.

- He is expected to understand the following things:
  a. Functioning or purpose of the embedded system
  b. Individual components involved
  c. The way data flows through the components of an embedded system.

- Consider an example of an embedded system intended to be used as a printer-sharing device. This device is attached to a printer and allows access to two computers through serial interface and one printer through a parallel interface.

- The diagram below describes the way the devices are connected to each other. Data to be printed is accepted from either serial port, held in RAM until the printer is ready for more data, and delivered to the printer via the parallel port. The software that makes all of this happen is stored in ROM.

- The working or execution of the code is brought about by the processor. The processor knows two types of components: memory and peripherals.

- Memories are for data and code storage and retrieval. Ex. RAM & ROM

- Peripherals are specialized hardware devices that either coordinate interaction with the outside world (I/O) or perform a specific hardware function. Ex. Serial Port

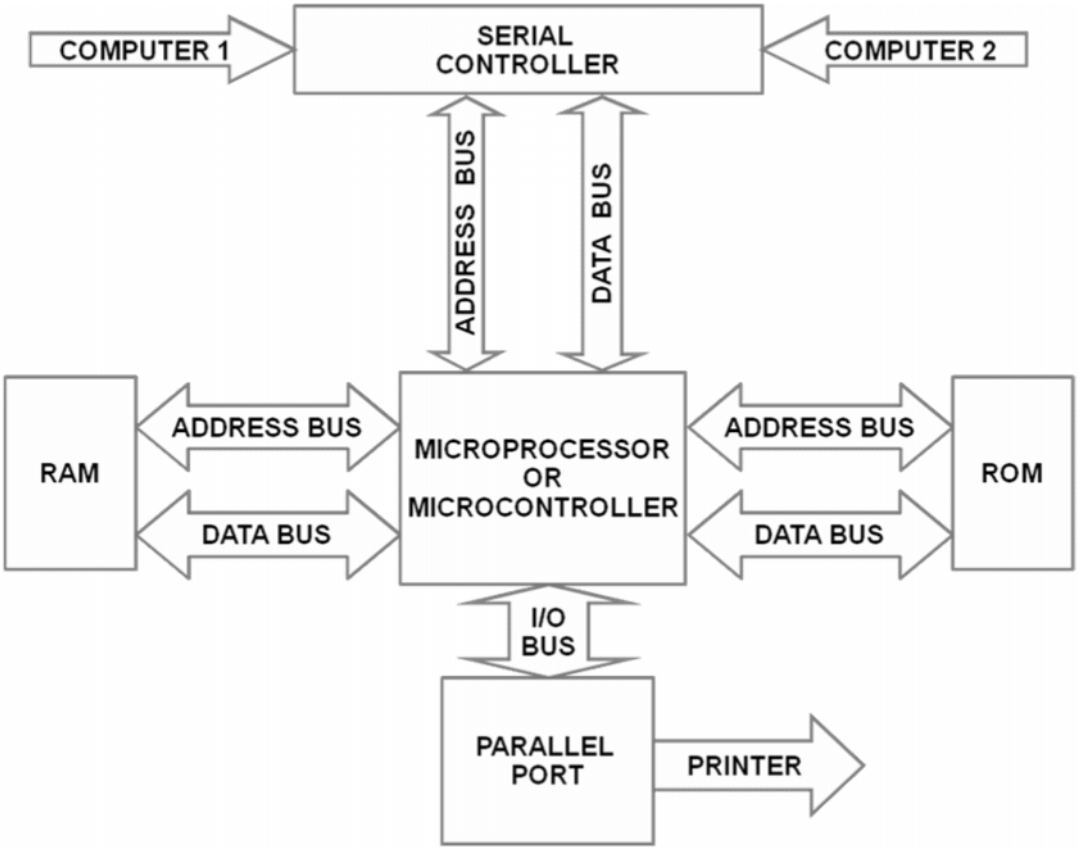


Figure: Components involved in an printer sharing device

- Certain processors like intel communicate with these memories and peripherals with two distinct address spaces.

- The first address space is called the memory space and is intended mainly for memory devices; the second is reserved exclusively for peripherals and is called the I/O space.

- When peripherals are located in I/O space they are called I/O Mapped peripheral else when peripherals are located in memory space they are called Memory Mapped peripherals or memory mapped I/O.

- If given a choice, Memory mapped peripherals are better because it has advantages for both the hardware and software developers. It is attractive to the hardware developer because he might be able to eliminate the I/O space, and some of its associated wires, altogether. It is attractive to the software developer who is able to use pointers, data structures, and unions to interact with the peripherals more easily and efficiently.

7.3 MEMORY MAP

- A Memory Map is the processor's "address book." It shows what these devices look like to the processor. The memory map contains one entry for each of the memories and peripherals that are accessible from the processor's memory space.

- All processors store their programs and data in memory.

- These chips are located in the processor's memory space, and the processor communicates with them by way of two sets of electrical wires called the address bus and the data bus. To read or write a particular location in memory, the processor first writes the desired address onto the address bus. The data is then transferred over the data bus.

- A memory map is a table that shows the name and address range of each memory device and peripheral that is located in the memory space.

- Organize the table such that the lowest address is at the bottom and the highest address is at the top. Each time a new device is added, add it to the memory map, place it in its approximate location in memory and label the starting and ending addresses, in hexadecimal. After inserting all of the devices into the memory map, be sure to label any unused memory regions as such.

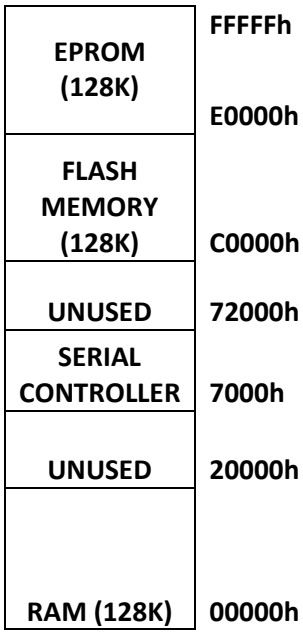
- The block diagram of the Printer sharing device shown above contains three devices attached to the address and data buses. These devices are the RAM and ROM and a Serial Controller.

- Let us assume that the RAM is located at the bottom of memory and extends upward for the first 128 KB of the memory space.

- The ROM is located at the top of memory and extends downward for 256 KB. But considering the ROM contains two ROMs-an EPROM and a Flash memory device-each of size 128 KB.

- The third device, the Serial Controller, is a memory-mapped peripheral whose registers are accessible between the addresses say 70000h and 72000h.

- The diagram below shows the memory map for the printer sharing device.



- For every embedded system, a header file should be created that describes these important features and provides an abstract interface to the hardware. It allows the programmer to refer to the various devices on the board by name, rather than by address.

- The part of the header file below describes the memory map

```
#define RAM_BASE      (void *) 0x00000000
#define SC_BASE       (void *) 0x70000000
#define SC_INTACK     (void *) 0x70001000
#define FLASH_BASE    (void *) 0xC0000000
#define EPROM_BASE    (void *) 0xE0000000
```

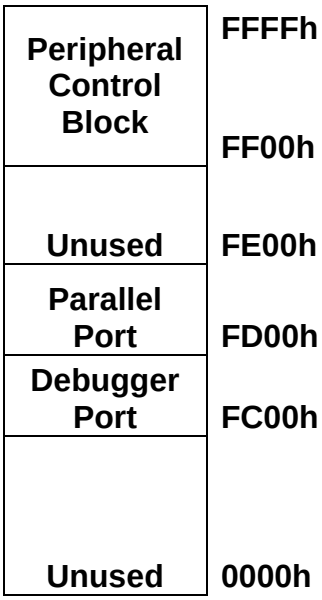
7.4 I/O MAP

- The I/O map contains one entry for each of the peripheral.

- An I/O map has to be created if a separate I/O space is present. It is done by repeating the steps performed to create memory map.

- To create an I/O map, simply create a table of peripheral names and address ranges, organized in such a way that the lowest addresses are at the bottom.

- The diagram below shows the I/O map for the printer sharing device



- It includes three devices: the peripheral control block (PCB), parallel port, and debugger port. The PCB is a set of registers within the processor that are used to control the on-chip peripherals. The chips that control the parallel port and debugger port reside outside of the processor. These ports are used to communicate with the printer and a host-based debugger, respectively.

- The part of the header file below describes the I/O map

```
#define SVIEW_BASE 0xFC00
#define PIO_BASE 0xFD00
#define PCB_BASE 0xFF00
```

7.5 INTERRUPT MAP

- There are two techniques which can be used by the processor to communicate with memories or peripheral devices. These are:

a. Polling: In this technique the processor polls the device (asks question) repeatedly at regular intervals to check if the device has completed the given task or has any new task to execute.

b. Interrupt:

- An interrupt is a signal sent from a peripheral to the processor. A peripheral may send an interrupt signal to a processor when it has some job to perform which requires the processors intervention.

- Upon receiving an interrupt signal the Processor does the job by issuing certain commands and waits for another interrupt to signal the completion of the job.

- While the processor is waiting for the interrupt to arrive, it is free to continue working on other things.

- When a fresh interrupt signal is received, the processor temporarily sets aside its current work and executes a small piece of software called the interrupt service routine (ISR). When the ISR completes, the processor returns to the work that was interrupted.

- The programmer must write the ISR himself and enable it so that it will be executed when the relevant interrupt occurs.

- **Interrupt Map**
- Embedded systems usually have only a handful of interrupts. Associated with each of these are an interrupt pin which is present on the outside of the processor chip and an ISR.

- In order for the processor to execute the correct ISR, a mapping must exist between interrupt pins and ISRs. This mapping usually takes the form of an interrupt vector table.

- The vector table is usually just an array of pointers to functions, located at some known memory address. The processor uses the interrupt type (a unique number associated with each interrupt pin) as its index into this array. The value stored at that location in the vector table is usually just the address of the ISR to be executed.

- An Interrupt Map is a step taken in this process. The Interrupt Map is a table that contains a list of interrupt types and the devices to which they refer.

- The diagram below shows the Interrupt map for the printer sharing device

Interrupt Type	Generating Device
8	Timer/Counter #0
17	Serial Controller
18	Timer/Counter #1
19	Timer/Counter #2
20	Serial Port Receive
21	Serial Port Transmit

- Once the I/O map is created the header file should be appended with the following information:

```
#define SCC_INT 17                  /*Serial Controller*/

#define TIMER0_INT 8                /* On-Chip Timer/Counters*/
#define TIMER1_INT 18
#define TIMER2_INT 19

#define RX_INT 20                   /* On-Chip Serial Ports */
#define TX_INT 21
```

7.6 REVIEW QUESTIONS

1. Explain the Components involved in a printer sharing device
2. Explain Memory Map for a printer sharing device
3. Explain I/O Map for a printer sharing device
4. Explain interrupt Map for a printer sharing device

7.7 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

8

EMBEDDED SYSTEMS: MEMORY

Unit Structure

37.0 OBJECTIVES

After reading this chapter you will understand:

- ✓ Different types of memory available
- ✓ Types of RAM
- ✓ Types of ROM
- ✓ Types of Hybrid Memory

8.1 INTRODUCTION

There are different types of memories available to be used in computers as well as embedded system.

This chapter guides the reader through the different types of memories that are available and can be used and tries to explain their differences in simple words.

8.2 TYPES OF MEMORY

- There are three main types of memories, they are

a) RAM (Random Access Memory)

- It is read write memory.
- Data at any memory location can be read or written.
- It is volatile memory, i.e. retains the contents as long as electricity is supplied.
- Data access to RAM is very fast

b) ROM (Read Only Memory)

- It is read only memory.
- Data at any memory location can be only read.
- It is non-volatile memory, i.e. the contents are retained even after electricity is switched off and available after it is switched on.
- Data access to ROM is slow compared to RAM

c) HYBRID

- It is combination of RAM as well as ROM
- It has certain features of RAM and some of ROM
- Like RAM the contents to hybrid memory can be read and written
- Like ROM the contents of hybrid memory are non volatile

- The following figure gives a classification of different types of memory

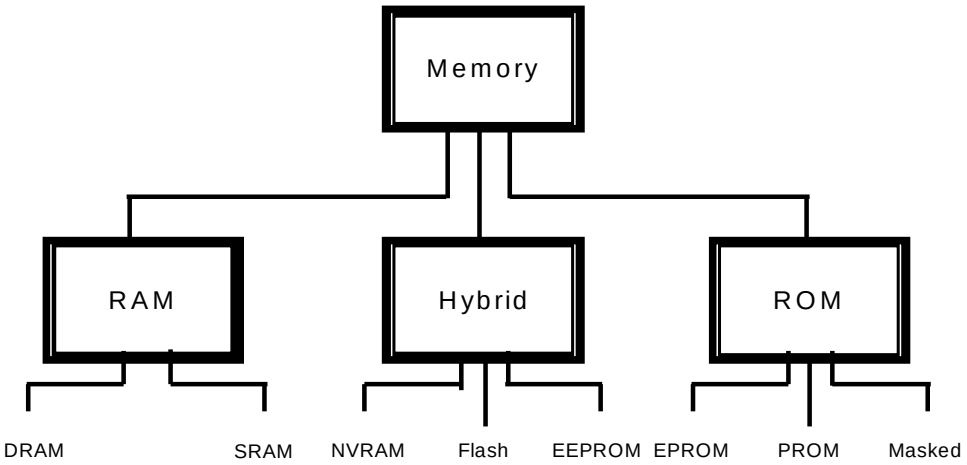


Figure: Types of Memory

8.3 TYPES OF RAM

- There are 2 important memory device in the RAM family.
  a) SRAM (Static RAM)
  b) DRAM (Dynamic RAM)

8.3.1 SRAM (Static RAM)

c) It retains the content as long as the power is applied to the chip.
d) If the power is turned off then its contents will be lost forever.

8.3.2 DRAM (Dynamic RAM)

a) DRAM has extremely short Data lifetime(usually less than a quarter of second). This is true even when power is applied constantly.
b) A DRAM controller is used to make DRAM behave more like SRAM.
c) The DRAM controller periodically refreshes the data stored in the DRAM. By refreshing the data several times a second, the DRAM controller keeps the contents of memory alive for a long time.

8.4 TYPES OF ROM

There are three types of ROM described as follows:

8.4.1 Masked ROM

a. These are hardwired memory devices found on system.
b. It contains pre-programmed set of instruction and data and it cannot be modified or appended in any way. (it is just like an Audio CD that contains songs pre-written on it and does not allow to write any other data)
c. The main advantage of masked ROM is low cost of production**.**

8.4.2 PROM (PROGRAMMABLE ROM)

a) This memory device comes in an un-programmed state i.e. at the time of purchased it is in an un-programmed state and it allows the user to write his/her own program or code into this ROM.
b) In the un-programmed state the data is entirely made up of 1’s.
c) PROMs are also known as one-time-programmable (OTP) device because any data can be written on it only once. If the data on the chip has some error and needs to be modified this memory chip has to be discarded and the modified data has to be written to another new PROM.

8.4.3 EPROM (ERASABLE-AND-PROGRAMABLE ROM)

a) It is same as PROM and is programmed in same manner as a PROM.
b) It can be erased and reprogrammed repeatedly as the name suggests.
c) The erase operation in case of an EPROM is performed by exposing the chip to a source of ultraviolet light.
d) The reprogramming ability makes EPROM as essential part of software development and testing process.

8.5 TYPES OF HYBRID MEMORY

There are three types of Hybrid memory devices:

8.5.1 EEPROMs

a. EEPROMs stand for Electrically Erasable and Programmable ROM.
b. It is same as EPROM, but the erase operation is performed electrically.
c. Any byte in EEPROM can be erased and rewritten as desired

8.5.2 Flash

a. Flash memory is the most recent advancement in memory technology.
b. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable.
c. Flash is much more popular than EEPROM and is rapidly displacing many of the ROM devices.
d. Flash devices can be erased only one sector at a time, not byte by byte.

8.5.3 NVRAM

a. NVRAM is usually just a SRAM with battery backup.
b. When power is turned on, the NVRAM operates just like any other SRAM but when power is off, the NVRAM draws enough electrical power from the battery to retain its content.
c. NVRAM is fairly common in embedded systems.
d. It is more expensive than SRAM.

8.6 DIRECT MEMORY ACCESS (DMA)

- DMA is a technique for transferring blocks of data directly between two hardware devices.

- In the absence of DMA the processor must read the data from one device and write it to the other one byte or word at a time.
- DMA Absence Disadvantage: If the amount of data to be transferred is large or frequency of transfer is high the rest of the software might never get a chance to run.
- DMA Presence Advantage: The DMA Controller performs entire transfer with little help from the Processor.
- Working of DMA
    - The Processor provides the DMA Controller with source and destination address & total number of bytes of the block of data which needs transfer.
    - After copying each byte each address is incremented & remaining bytes are reduced by one.
    - When number of bytes reaches zeros the block transfer ends & DMA Controller sends an Interrupt to Processor.

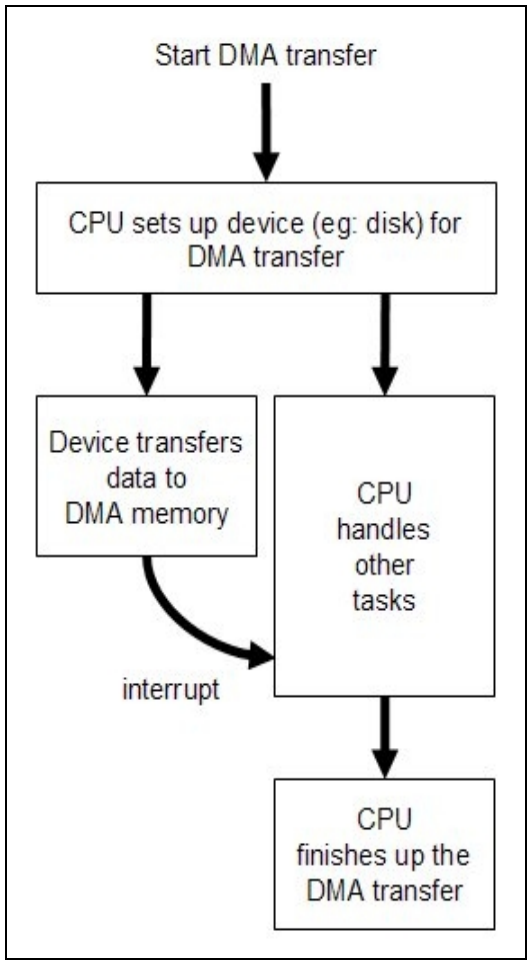


Figure: Direct Memory Access

8.7 REVIEW QUESTIONS

1. What are the different types of Memory?
2. What are the different types of RAM?
3. What are the different types of ROM?
4. What are the different types of Hybrid Memory?

8.8 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

9

EMBEDDED SYSTEMS: MEMORY TESTING

Unit Structure

9.0 OBJECTIVES

After reading this chapter you will be able to understand:

- ✓ What is memory testing?
- ✓ What are the common memory related problems?
- ✓ What are the different types of test to detect memory related problems and a general idea about the working of these tests

9.1 INTRODUCTION

The previous chapter dealt with the different types of memory. This chapter will focus on the concept of testing memory devices, its purpose and different methods available.

9.2 MEMORY TESTING AND ITS PURPOSE

- The purpose of a memory test is to confirm that each storage location in a memory device is working.
- Memory Testing is performed when prototype hardware is ready and the designer needs to verify that address and data lines are correctly wired and memory chips are working properly.
- Basic idea implement in testing can be understood by this simple task:

  - Write some set of Data values to each Address in Memory and Read it back to verify.
  - Ex. If number '50' is stored at a particular Address it is expected to be there unless rewritten or erased.
  - If all values are verified by reading back then Memory device passes the test.
- Only through careful selection of data values can make sure passing result to be meaningful.
- Difficulties involved in memory testing:
  - It can be difficult to detect all memory problems with a simple test.
  - Many Embedded Systems include Memory Tests only to detect catastrophic memory failures which might not even notice memory chips removal.

9.3 COMMON MEMORY PROBLEMS

- Memory Problems rarely occur with the chip itself, but due to a variety of post production tests to check quality this possibility is ruled out.

- Catastrophic Failure is a memory problem that occurs due to physical and electrical damage, it is uncommon and easily detectable.

- A common source of memory problems is associated with the circuit board. Typical circuit board problems are:
  1. Circuit board wiring between Processor & Memory device.
  2. Missing Memory chip.
  3. Improperly inserted Memory chip**.**

1. Circuit board wiring between Processor & Memory device.

- These are usually caused by,
  i. An error in design
  ii. An error in production of the board
  iii. Any damage after manufacture

- Wires that connect the memory are:-

i. Address line :- select the memory location
ii. Data line :- transfer the data
iii. Control line :- read or write operation

- Two wiring problems are shown below

1. Connected to another wire on the board

- May be caused by a bit of solder splash

2. **Not connected to anything**

- Caused by broken trace

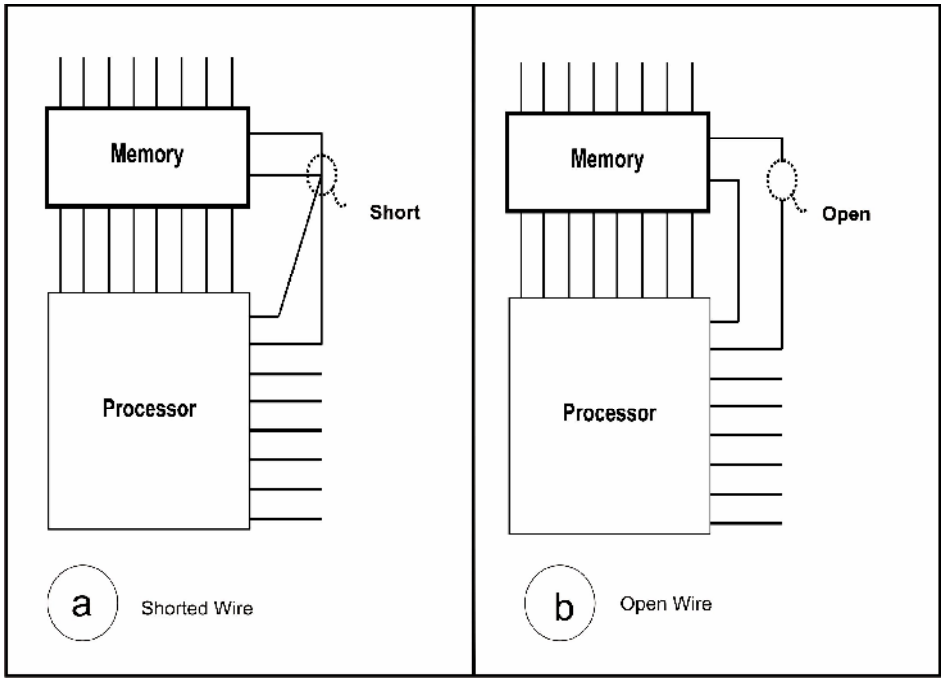


Figure: a. wiring problems: two wires shorted
b. wiring problems: one wire open

- When **Address line** has a wiring problem
  - memory locations overlap
  - i.e. memory device to see an address different from the one selected by the processor.
  - Problem is with a **data line**
  - several data bits “stuck together”
  - i.e. two or more bits always contains same value

- When the problem is with a **Data line**
  - several data bits “stuck together”
  - i.e. two or more bits always contains same value
  - When Control lines is shorted or open

- When **Control lines** is shorted or open
  - The operation of many control lines is specific to the processor or memory architecture.
  - the memory will probably not work at all.

2. Missing Memory chip.

- A missing memory chip is clearly a problem that should be detected
- Unfortunately, because of the capacitive nature of unconnected electrical wires, some memory tests will not detect.

- For e.g. suppose you decided to use the following test algorithm
  - ✓ write the value 1 to the first location in memory, verify the value by reading it back
  - ✓ write 2 to the second location, verify the value
  - ✓ write 3 to the third location, verify, etc.
- Because each read occurs immediately after the corresponding write, it is possible that the data read back represents nothing more than the **voltage remaining** on the data bus from the previous write.
- If the data is read back too quickly, it will appear that the data has been correctly stored in memory-even though there is no memory chip at the other end of the bus!
- To detect a missing memory chip the previous algorithm for test must be altered.
- For example,
  - ✓ write the value 1 to the first location,
  - ✓ 2 to the second location,
  - ✓ And 3 to the third location,
- Then verify the data at the first location, the second location, etc. If the data values are unique (as they are in the test just described), the missing chip will be detected

3. Improperly inserted Memory chip.

- Caused by pins on the memory chip
  - Will either not be connected to the socket at all
  - Will be connected at the wrong place
- Symptoms :-
  - System behaves same as though there is a wiring problem or a missing chip.
- How to detect :-
  - Detected by any test

9.4 A STRATEGY FOR MEMORY TESTING

- For memory testing the strategy adopted should be effective and efficient. Ideally there should be multiple small tests instead of one large test.
- It would be best to have three individual memory tests:
  1. **A data bus test:** Checks electrical wiring problems

2. **An address bus test:** Checks improperly inserted chips
3. **A device test:** Checks to detect missing chips and catastrophic failures and problems with the control bus wiring

- These tests have to be executed in a proper order which is: data bus test first, followed by the address bus test, and then the device test. That's because the address bus test assumes a working data bus, and the device test results are meaningless unless both the address and data buses are known to be good.

9.4.1 Data Bus Test

- It is used to check data bus wiring.
- In this test we need to confirm that the received data is same as the data sent by processor

- **Implementation:**
  - Here we write all possible data values and verify that the memory device stores each one successfully.
  - In short to test the bus one bit at a time.

- **Walking 1's test**
  - This test is used to independently test every bit.
  - A single data bit is set to 1 and "walked" through the entire data word.
  - If the data bus is working properly, the function will return 0.
  - Otherwise it will return the data value for which the test failed.
  - Because we are testing only the data bus at this point, all of the data values can be written to the same address. Any address within the memory device will do

00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000

Figure: Consecutive data values for walking 1's test

9.4.2 Address Bus Test

- Address bus problems lead to overlapping memory locations.
- In the Address Bus test we need to confirm that each of the address pins can be set to 0 and 1 without affecting any of the others.
- The smallest set of address that will cover all possible combinations is the set of “power of two” addresses.
- After writing one of the addresses, we must check none of the others has been overwritten.

9.4.3 Device Test

- It is used to test if the memory device is working properly. It is necessary to test the integrity of the memory device itself.
- The thing to test is that every bit in the device is capable of holding both 0 and 1.
- For a thorough and complete device test every memory location has to be visited twice.
- A simple test implemented is the **Increment test** as shown in the table below
  - The first column represents the memory location
  - The second column represents the data that is written at the memory location indicated in column 1 in incremental fashion.
  - The third column represents the data of column 2 in inverted format.

000h	00000001	11111110
001h	00000010	11111101
002h	00000011	11111100
003h	00000100	11111011
… … …	… … …	… … …
… … …	… … …	… … …
… … …	… … …	… … …
0FEh	11111111	00000000
0FFh	00000000	11111111

Figure: Data Bus Test – Increment Test

- During the first pass the data in column 1 is verified and during second pass the data in column 2 is verified.

9.5 REVIEW QUESTIONS

1. What is Memory Testing? Why is it required?
2. What are common memory problems in embedded system?
3. Describe a test strategy for performing memory testing on embedded system. Is there a specific order to perform these tests? if yes, why?
4. Describe the different types of memory testing techniques available.

9.6 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

10

EMBEDDED SYSTEMS: PERIPHERALS

Chapter Structure

10.0 OBJECTIVES

After reading this chapter you will learn:

- ✓ Concept of testing non –volatile memory devices using Checksum and CRC
- ✓ Control and Status Registers
- ✓ Device Driver
- ✓ Watch Dog Timer

10.1 INTRODUCTION

This chapter initially continues the part of memory testing from last chapter. Here testing of Non Volatile memory devices is studied.

Then we study how peripheral devices are incorporated in Embedded System. Control and Status Registers, Device Drivers and Watch Dog Timers are explained in the subsequent sections.

10.2 TESTING NON VOLATILE (ROM AND HYBRID) MEMORY DEVICES

- The testing techniques described previously cannot help to test ROM and hybrid devices since ROM devices cannot be written at all, and hybrid devices usually contain data or programs that cannot be overwritten.

- However ROM or hybrid memory device face the same problems as missing memory chip, improperly inserted memory chip, damaged memory chip or wiring problem with the memory chip.

- Two Techniques Checksums and CRC can be used to test non volatile memory devices.

- **Checksum**
  - Checksums basically deals with the question whether the data stored in a memory device is valid or not?
  - To do this the checksum of the data in the memory device is computed and stored along with the data. The moment when we have to confirm the validity of the data, we just have to recalculate the checksum and compare it with previous checksum. If the two checksums match, the data is assumed to be valid.
  - The simplest checksum algorithm is to add up all the data bytes discarding carries.
  - A Checksum is usually stored at some fixed location in memory. This makes it easy to compute and store the check sum for the very first time and later on to compare the recomputed checksum with the original one.
  - Disadvantage: A simple sum-of-data checksum cannot detect many of the most common data errors.

- **CRC – Cyclic Redundancy Check**
  - A Cyclic Redundancy Check is a specific checksum algorithm designed to detect the most common data errors.
  - CRC's are frequently used in Embedded Applications that requires the storage or transmission of large blocks of data.
  - The CRC works as follows:
    - The message is composed of a long string of 0's and 1's
    - A division operation occurs between the message at numerator and the generator polynomial at denominator. The generator polynomial is a fixed smaller length binary string.
    - The remainder of the division operation is the CRC Checksum

10.3 CONTROL AND STATUS REGISTERS

- Control and status registers are the basic interface between and embedded processor and peripheral device.
- These registers are a part of peripheral hardware and their location size and individual meanings are feature of the peripheral.
- For example, The registers vary from device to device: example the registers within a serial controller are very different from those in a timer.
- Depending upon the design of the processor and target board , peripheral devices are located either in the processor's memory space or within the I/O space.
- It is common for Embedded Systems to include some peripherals of each type. These are called Memory-Mapped and I/O-mapped peripherals.
- Of the two types, memory-mapped peripherals are generally easier to work with and are increasingly popular.
- Memory-mapped control and status registers can be used just like ordinary variables.

10.4 DEVICE DRIVER

- The goal of designing a device driver is to hide the hardware completely.
- Attempts to hide the hardware completely are difficult.
- For example all Flash memory devices share the concept of sectors. An erase operation can be performed only on an entire sector. Once erased individual bites or words can be rewritten.
- Device drivers for embedded systems are quite different from the workstation counter parts. In modern computers workstation device drivers are most often concerned with satisfying the requirement of the operating system.
- There are three benefits of good device driver:

  i. Modularization, it makes the structure of the overall software is easier to understand.

  ii. There exists only one module that interacts directly with the peripheral's registers making communication easier.

  iii. Software changes that result from hardware changes are localized to the device driver.

- **Components of a Device Driver**

A device driver can be implemented (as components) in the following steps:

1. **A data structure that overlays the memory-mapped control and status registers of the device:**
   - This basic step involves creating a C style structure that is actually a map of the registers present in the device. These registers can be found out by referring to the data sheet for the device.
   - A table is created which maps the control register to their relative offsets.
   - An example is shown below for a timer counter data structure.

```
struct TimerCounter
                {
unsigned short count; // Current Count, offset 0x00
unsigned short maxCountA;// Maximum Count, offset 0x02
unsigned short _reserved;  // Unused Space, offset 0x04
unsigned short control; // Control Bits, offset 0x06
                };
```

   - To make the bits within the control register easier to read and write individually, we define the following bitmasks:

```
#define TIMER_ENABLE 0xC000 // Enable the timer.
#define TIMER_DISABLE 0x4000 // Disable the timer.
#define TIMER_INTERRUPT 0x2000 // Enable timer interrupts.
#define TIMER_MAXCOUNT 0x0020 // Timer complete?
#define TIMER_PERIODIC 0x0001 // Periodic timer?
```

2. **A set of variables to track the current state of the hardware and device driver:** It involves listing out the required variables needed to keep track of the state of the hardware and device driver

3. **Initialize the hardware:** Once the variables to be used are known the next step in device driver programming is to initialize the hardware. Next functions can be written to control the device.

4. **A set of routines that provide an API for users of the device driver**
   This involves writing different functions that will implement the various tasks listed to be performed by the device.

5. **Interrupt service routines**
   Once the required functions and routines are coded the thing remaining to be done is to identify and write routines for servicing the interrupts.

10.5 WATCHDOG TIMER

- It is hardware equipment.
- It is special purpose hardware that protects the system from software hangs.
- Watchdog timer always counts down from some large number to zero
- This process takes a few seconds to reset, in the meantime, it is possible for embedded software to “kick” the watchdog timer, to reset its counter to the original large number.
- If the timer expires i.e. counter reaches zero, the watchdog timer will assume that the system has entered a state of software hang, then resets the embedded processor and restarts the software
- It is a common way to recover from unexpected software hangs
- The figure below diagrammatically represents the working of the watchdog timer

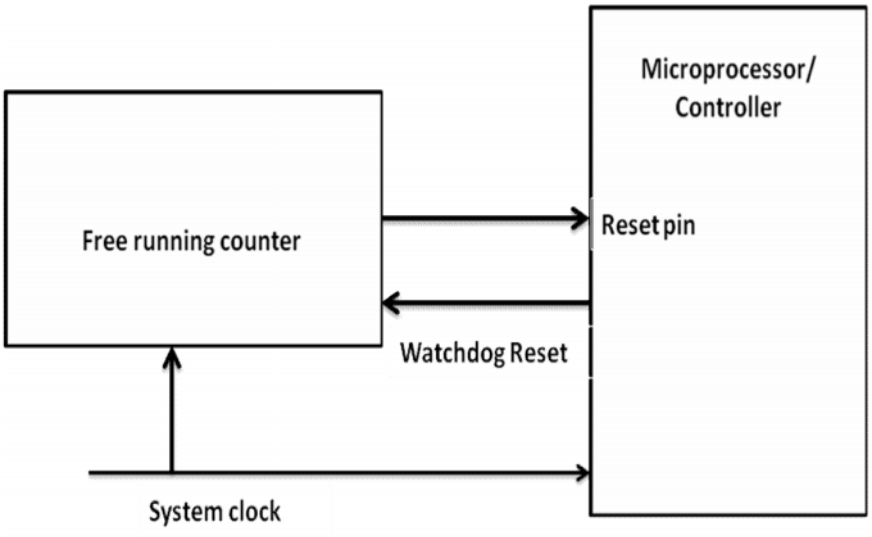


Figure: Watchdog Timer

10.6 REVIEW QUESTIONS

1. Explain testing for non-volatile memory devices
2. Write short note on Control and status registers
3. What is a device driver?
4. What are the components of a device driver?
5. Write short note on Watch Dog Timer

10.7 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

11

EMBEDDED OPERATING SYSTEM

Chapter Structure

11.0 OBJECTIVES

After reading this chapter you will learn:

- ✓ The basics of embedded Operating system with respect to
    1. Tasks
    2. Task States
- ✓ Scheduler with respect to:
    1. Scheduling Points
    2. Ready List
    3. Idle task
- ✓ Concept of Context Switch and Task Synchronization
- ✓ Real Time Characteristic of embedded operating system.

11.1 INTRODUCTION

This chapter introduces the readers to the embedded operating system. Any operating system has a set of programs which are implemented through a set of tasks.

Every embedded system may not require an operating system. The requirement and complexity on an operating system depends on the functionality to be implemented by the embedded system.

11.2 BASICS

11.2.1 Tasks

- Task is a piece of code or program that is separate from another task and can be executed independently of the other tasks.

- In embedded systems, the operating system has to deal with a limited number of tasks depending on the functionality to be implemented in the embedded system.

- Multiple tasks are not executed at the same time instead they are executed in pseudo parallel i.e. the tasks execute in turns as the use the processor.

- From a multitasking point of view, executing multiple tasks is like a single book being read by multiple people, at a time only one person can read it and then take turns to read it. Different bookmarks may be used to help a reader identify where to resume reading next time.

- An Operating System decides which task to execute in case there are multiple tasks to be executed. The operating system maintains information about every task and information about the state of each task.

- The information about a task is recorded in a data structure called the ***task context.*** When a task is executing, it uses the processor and the registers available for all sorts of processing. When a task leaves the processor for another task to execute before it has finished its own, it should resume at a later time from where it stopped and not from the first instruction. This requires the information about the task with respect to the registers of the processor to be stored somewhere. This information is recorded in the task context.

- A C++ version of a Task that holds all information needed by operating system is as follows:

```
class Task
{
        public:
                Task(void (*function)(), Priority p, int
                stackSize);

                TaskId id;
                Context context;
                TaskState state;
                Priority priority;
```

```
        int * pStack;
        Task * pNext;

        void (*entryPoint)();

    private:
    static TaskId nextId;
};
```

11.2.2 Task States

- In an operation system there are always multiple tasks. At a time only one task can be executed. This means that there are other tasks which are waiting their turn to be executed.

- Depending upon execution or not a task may be classified into the following three states:

  - **Running state** - Only one task can actually be using the processor at a given time that task is said to be the "running" task and its state is "running state". No other task can be in that same state at the same time

  - **Ready state** - Tasks that are are not currently using the processor but are ready to run are in the "ready" state. There may be a queue of tasks in the ready state.

  - **Waiting state -** Tasks that are neither in running nor ready state but that are waiting for some event external to themselves to occur before the can go for execution on are in the "waiting" state**.**

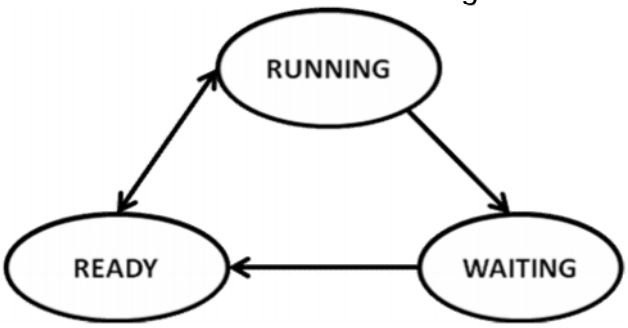


Figure: Task States

- A transition of state between the ready and running state occurs whenever the operating system selects a new task to run.

- The task that was previously in running state becomes ready and the new task is promoted to running state.

- A task will leave running state only if it needs to wait for some event external to itself to occur before continuing.

- A task's state can be defined as follows:
  enum TaskState { Ready, Running, Waiting };

11.3 SCHEDULER

- The heart and soul of any operating system is its scheduler.
- This is the piece of the operating system that decides which of the ready tasks has the right to use the processor at a given time.
- It simple checks to see if the running task is the highest priority ready task.
- Some of the more common scheduling algorithms:

1. First-in-first-out

- First-in-first-out (FIFO) scheduling describes an operating system which is not a multitasking operating system.

- Each task runs until it is finished, and only after that is the next task started on a first come first served basis.

2. Shortest job first

- Shortest job first scheduling uses algorithms that will select always select a task that will require the least amount of processor time to complete.

3. Round robin.

- Round robin scheduling uses algorithms that allow every task to execute for a fixed amount to time.

- A running task is interrupted an put to a waiting state if its execution time expires.

11.3.1 Scheduling Points

- The scheduling points are the set of operating system events that result in an invocation of the scheduler.

- There are three such events: **task creation** and **task deletion.** During each of these events a method is called to select the next task to be run.

- A third scheduling point called the **clock tick** is a periodic event that is triggered by a timer interrupt. When a timer expires, all of the tasks that are waiting for it to complete are changed from the waiting state to the ready state.

11.3.2 Ready List

- The scheduler uses a data structure called the **ready list** to track the tasks that are in the ready state.
- The ready list is implemented as an ordinary linked list, ordered by priority.
- So the head of this list is always the highest priority task that is ready to run.

11.3.3 Idle task

- If there are no tasks in the ready state when the scheduler is called, the idle task will be executed.
- The idle task looks the same in every operating system.
- The idle task is always considered to be in the ready state.

11.4 CONTEXT SWITCH

- The actual process of changing from one task to another is called **Context Switch**.
- Since **contexts are processor-specific**, so is the code that implements the context switches, hence, it must always be **written in assembly language**.

11.5 TASK SYNCHRONIZATION

- All the tasks in the multitasking operating systems work together to solve a larger problem and to synchronize their activities, they occasionally communicate with one another.
- For example, in the printer sharing device the printer task doesn't have any work to do until new data is supplied to it by one of the computer tasks.
- So the printer and the computer tasks must communicate with one another to coordinate their access to common data buffers.
- One way to do this is **to use a data structure called a mutex**.
- Mutexes are mechanisms provided by many operating systems **to assist with task synchronization**.

- A mutex is **a multitasking-aware binary flag**. It is because the processes of setting and clearing the binary flag are **atomic** (i.e. these operations cannot be interrupted).

- When this binary flag is set, the shared data buffer is assumed to be in use by one of the tasks. All other tasks must wait until that flag is cleared before reading or writing any of the data within that buffer.

- The atomicity of the mutex set and clear operations is enforced by the operating system, which disables interrupts before reading or modifying the state of the binary flag.

11.6 REAL TIME CHARACTERISTIC

An Operating system is called "Real-Time Operating System" (RTOS) only if it has following characteristics:

i. Deterministic

- An OS is said to be deterministic if the worst case execution time of each of the system calls is calculable.

- The data sheet of an OS should publish the real-time behavior of its RTOS provides average, minimum and maximum number of clock cycles required by each system call.

ii. Interrupt Latency

- Interrupt Latency is the total length of time from an interrupt signal's arrival at the processor to the start of the associated interrupt service routine.

iii.Context Switch

- Context Switch is important because it represents overhead across your entire system.

11.7 SELECTION PROCESS

The process of selecting the best commercial operating system that best fits the needs of one's project depends on various factors.

- Commercial operating systems form a continuum of functionality, performance and price.

- Operating Systems that offer only a basic scheduler and a few other system calls are inexpensive and come with the source code that one can modify and do not require payment of royalties.

- While on the other hand operating systems that include a lot of useful functionality beyond just the scheduler are quite expensive and royalties due on every copy shipped in ROM and they might also make a stronger guarantees about real-time performance.

Two important points to be considered while selecting an operating system :-

- Put your processor, real time performance and budgetary requirements first.
- Contact all of the vendors of the remaining operating systems for more detailed technical information.

11.8 REVIEW QUESTIONS

1. Explain the embedded Operating system with respect to
   i) Tasks
   ii) Task States
2. Explain Scheduler with respect to:
   i) Scheduling Points
   ii) Ready List
   iii) Idle task
3. Write a short note on Context Switch and Task Synchronization
4. Explain the Real Time Characteristic of embedded operating system.

11.9 REFERENCES & FURTHER READING

1. Programming Embedded systems in C++ by Michael Barr
2. Introduction to Embedded systems – Shibu K. V

12

EMBEDDED SYSTEMS: INTEGRATED DEVELOPMENT ENVIRONMENT

Chapter Structure

12.0 OBJECTIVES

After reading this chapter you will understand:

- Embedded IDE
- Types of file involved
- Disassembler/ Decomipiler
- Simulator
- Firmware Debugging and Emulator

12.1 INTRODUCTION

This chapter explains the IDE used for embedded systems. It then explains the different types of files that are generated on cross compilation. Then it gives an account of utility tools like Disassembler/ Decomipiler, Simulator and then FirmWare Debugging.

12.2 EMBEDDED IDE

- Integrated Development Environment with respect to embedded system IDE stands for an Integrated Environment for developing and debugging the target processor specific embedded software.

- IDE is a software package which contains:
    1. Text Editor(Source Code Editor)

2. Cross Compiler(For Cross platform development and complier for the same platform development)
3. Linker and debugger.

- Some IDEs may provide an interface to an emulator or device programmer.
- IDEs are used in embedded firmware development.

➢ IDEs may be of two types:

1. **Command Line Base**
   - Turbo C++ IDE is an example for a generic IDE with a Command Line Interface.
2. **GUI Base**
   - Microsoft Visual Studio is an example of GUI base IDE.
   - Others examples are NetBeans, Eclipse.

12.3 TYPES OF FILE GENERATED ON CROSS COMPILATION

Following are some of the files generated upon cross compilation:

1. List file .lst
2. Hex file .hex
3. Preprocessor output file
4. Map file .map
5. Obj file .obj

1. **List File(.lst):-**
   - Listing file is generated during the cross-compilation process.
   - It contains an information about the cross compilation process like cross compiler details, formatted source text('C' code), assembly code generated from the source file, symbol tables, errors and warnings detected during the cross-compilation process.
   - The list file contain the following sections:

1. Page Header
   - It indicates the compiler version name, source file name, Date, Page No.
   - Example: C51 COMPILER V8.02 SAMPLE 05/23/2006 11:12:58 PAGE 1

2.Command Line
   - It represents the entire command line that was used for invoking the compiler.
   - C51 COMPILER V8.02, COMPILATION OF MODULE SAMPLE OBJECT MODULE PLACED IN sample.obj

- COMPILER INVOKED BY: C:\Keil\C51\BIN\C51.EXE sample.c BROWSE DEBUG OBJECTTEXTEND CODE LISTINCLUDE SYMBOLS

3.Source Code

- It contains source code along with line numbers
- Line level Source

```
1       //Sample.c for printing Hello World!
2       //Written by xyz
3       #include<stdio.h>
1       //Body part starts
2
3
4
5
6       //Body part end
4       void main()
5       {
6       printf("Hello World");
7       }
8       //Header part ends
```

4. Assembly listing

- It contains the asembly code generated by compiler for even given 'C' code.
- ASSEMBLY LISTING OF GENERATED OBJECT CODE;
- FUNCTION main(BEGIN)

```
;SOURCE LINE #5
;SOURCE LINE #6
    0000  7BFF              MOV  R3,#0FFH
    0002  7A00      R       MOV
    R2,#HIGH?SC_0
```

5. Symbol listing

- It contains symbolic information about the various symbols present in the cross compiled source file.
- Eg: NAME, TYPE, SFR, SIZE.
-

6. Module Information

- The module information provides the size of initialized and un-initialized memory areas defined by the source file.

Module Information	Static	Overlayable
Code Size	9	----------
Constant size	14	----------
Bit size	-------	----------

END OF MODULE INFORMATION

7. Warnings and Errors

- Warnings and Errors section of list file records the errors encountered or any statement that may create issues in application(Warnings), during cross compilation.

- ie:- C51 COMPILATION COMPLETE, 0WARNING(S), 0 ERROR(S).

2. Preprocessor Output File

- It contains preprocessor output for preprocessor instructions used in the source file.

- This file is used for verifying the operation of Macros and preprocessor directive.

3. Object File(.OBJ File)

- Cross-compiling each source module converts the Embedded C/Assembly instructions and other directives present in the module to an object(.OBJ file)

4. Map File(.MAP)

- Also called as Linker List file. Map file contains information about the link/locate process and is composed of a number of sections described below:

I. Page Header

Each MAP file contains a header which indicates the linker version number, date, time and page number.

II. Command Line

Represents the entire command line that was used for invoking the linker.

III. CPU Details

It contains details about the target CPU and its memory model which includes information on internal data memory, external data memory, paged data memory, etc.

IV. Input Modules

It includes the names of all the object files, library files and other files that are included in the linking process.

V. Memory Map

It lists the starting address, length, relocation type and name of each segment in the program

VI. Symbol Table

It contains the name, value and type for all symbols from different input modules.

VII. Inter Module Cross Reference

It includes the section name, memory type and module names in which it is defined and all modules where it is accessed.

Ex.

NAME……..USAGE……..

MODULE NAMES

?CCCASE…….CODE;……?C?CCASE PRINTF

?C?CLDOPTR……CODE;……..?C?CLDOPTR PRINTF

?C?CSTPTR……CODE;……..?C?CSTPTR PRINTF

VIII. Program Size

It contains the size of various memory areas, constants and code space for the entire application

Ex. Program Size: data=80.1 xdata=0 code 2000

IX. Warnings and Errors

It contains the warnings and errors that are generated while linking a program. It is used in debugging link errors

5. HEX FILE (.hex file)

1. It is a binary executable file created from the source code.
2. The file created by linker/locater is converted into processor understandable binary code.
3. The tool used for converting and object file into a hex file is known as object to Hex converter.
4. Hex file have specific format and it varies for different processor and controller. Two commonly used hex file format are:

   A. Intel Hex

   B. Motorola Hex.

5. Both Intel and Motorola hex file format represent data in the form of ASCII codes.

12.4 DISASSEMBLER/ DECOMIPILER

- A **Disassembler/ Decomipiler** is a reverse engineering tool.
- Reverse Engineering is used in embedded system to find out the secret behind the working of a proprietary product.
- A **DISASSEMBLER** is a utility program which converts machine codes into target processor specific assembly code/instruction.
- The process of converting machine codes to assembly code is called **disassembling.**
- A **DECOMIPILER** is a utility program for translating machine codes into corresponding high level language instruction.
- A decompiler performs the reverse operation of a compiler/cross-compiler.

12.5 SIMULATOR

- Simulators are used for embedded firmware debugging.
- Simulator simulates the target hardware, while the code execution can be inspected.
- Simulators have the following characteristics which make them very much favorable:
  - ✓ Purely software based
  - ✓ No need of target system (hardware)
  - ✓ Support only for basic operations
  - ✓ Cannot Support or lack real time behavior

- Advantages

1. Simple and straight forward.

- Simulators are a software utility with assumptions about the underlying hardware. So it only requires concentrating on debugging of the code, hence straight forward.

2. No Hardware

- Simulators are purely software oriented.
- The IDE simulates the target CPU. The user needs to know only about the target specific details like memory map of various devices.
- Since no hardware is required the code can be written and tested even before the hardware prototype is ready thus saving development time

3. Simulation options

- Simulators provide various simulation options like I/O peripherals or CRO or Logic analyzers.

- Simulators I/O support can be used to edit values for I/O registers.

4. Simulation of abnormal conditions

- Using simulator the code can be tested for any desired value.
- This helps to study the code behavior in abnormal conditions without actually testing it on the hardware.

- **Disadvantages**
  1. **Lack of real time behavior**
     - A simulator assumes the ideal condition for code execution.
     - Hence the developer may not be able to debug the code under all possible combinations of input.
     - The results obtained in simulation may deviate from actual results on target hardware.

  2. **Lack of real timeliness**
     - The I/O condition in hardware is unpredictable. So the output of simulation is usually under ideal condition and hence lacks timeliness.

12.6 FIRMWARE DEBUGGING

- Debugging in embedded application is the process of diagnosing the firmware execution, monitoring the target processor's registers and memory while the firmware is running and checking the signals on various buses of hardware.
- Debugging is classified into Hardware Debugging and Firmware Debugging.
- Hardware Debugging deals with debugging the various aspects of hardware involved in the embedded system.
- The various tools used for hard ware debugging are Multimeter, CRO, Logic Analyzers and Function Generators.
- Firmware Debugging involves inspecting the code, its execution flow, changes to different registers on code execution.
- It is done to find out the bugs or errors in code which produces unexpected behavior in the system.
- There is a wide variety of firmware debugging techniques available that have advanced from basic to advanced.
- Some to the tools used are Simulators and Emulators.

- **Emulators**
  - The terms simulators and emulators are very confusing but their basic functionality is the same i.e. to debug the code. There is a difference in which this is achieved by both the tools.

  - A simulator is a utility program that duplicates the target CPU and simulates the features and instructions supported by target CPU whereas an Emulator is a self contained hardware device which emulates the target CPU.

  - The Emulator hardware contains the necessary emulation logic and is connected to the debugging application that runs on the host PC.

  - The Simulator *'simulates'* while the Emulator *'emulates'*

12.7 REVIEW QUESTIONS

1. Write a Short note on Embedded IDE
2. What is Cross- Compilation? List the files that are generated upon cross compilation
3. Explain the contents of .MAP file.
4. Explain the contents of .LST file.
5. Write short notes on :
   - I. .OBJ File
   - II. .HEX File
   - III. Preprocessor Output File

12.8 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

13

EMBEDDED DEVELOPMENT LIFE CYCLE

Chapter Structure

13.0 OBJECTIVES

After Reading this chapter you will understand

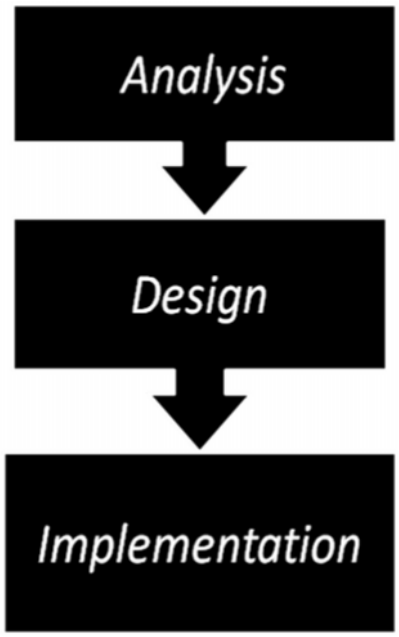
- ✓ The Embedded Development Life Cycle
- ✓ Phases Involved in the EDLC

13.1 INTRODUCTION

Just like the SDLC used in Software Development, there is EDLC used in Embedded product development. This chapter explains what is the EDLC, its objectives, the phases that are involved in the EDLC.

13.2 EMBEDDED PRODUCT DEVELOPMENT LIFE CYCLE (EDLC)

- ➢ EDLC is Embedded Product Development Life Cycle
- ➢ It is an Analysis – Design – Implementation based problem solving approach for embedded systems development.
- ➢ There are three phases to Product development:



- Analysis involves understanding what product needs to be developed
- Design involves what approach to be used to build the product
- Implementation is developing the product by realizing the design.

13.2.1 Need for EDLC

- EDLC is essential for understanding the scope and complexity of the work involved in embedded systems development
- It can be used in any developing any embedded product
- EDLC defines the interaction and activities among various groups of a product development phase.
  Example:-project management, system design

13.2.2 Objectives of EDLC

- The ultimate aim of any embedded product in a commercial production setup is to produce Marginal benefit
- Marginal is usually expressed in terms of Return On Investment
- The investment for product development includes initial investment, manpower, infrastructure investment etc.
- EDLC has three primary objectives are:

i. Ensure that high quality products are delivered to user

- Quality in any product development is Return On Investment achieved by the product
- The expenses incurred for developing the product the product are:-

- Initial investment
- Developer recruiting
- Training
- Infrastructure requirement related

ii. **Risk minimization defect prevention in product development through project management**

- In which required for product development 'loose' or 'tight' project management
- 'project management is essential for ' predictability co-ordination and risk minimization
- Resource allocation is critical and it is having a direct impact on investment
- Example:- Microsoft @ Project Tool

iii. **Maximize the productivity**

- Productivity is a measure of efficiency as well as Return On Investment
- This productivity measurement is based on total manpower efficiency
- Productivity in which when product is increased then investment is fall down
- Saving manpower

13.3 DIFFERENT PHASES OF EDLC

The following figure depicts the different phases in EDLC:

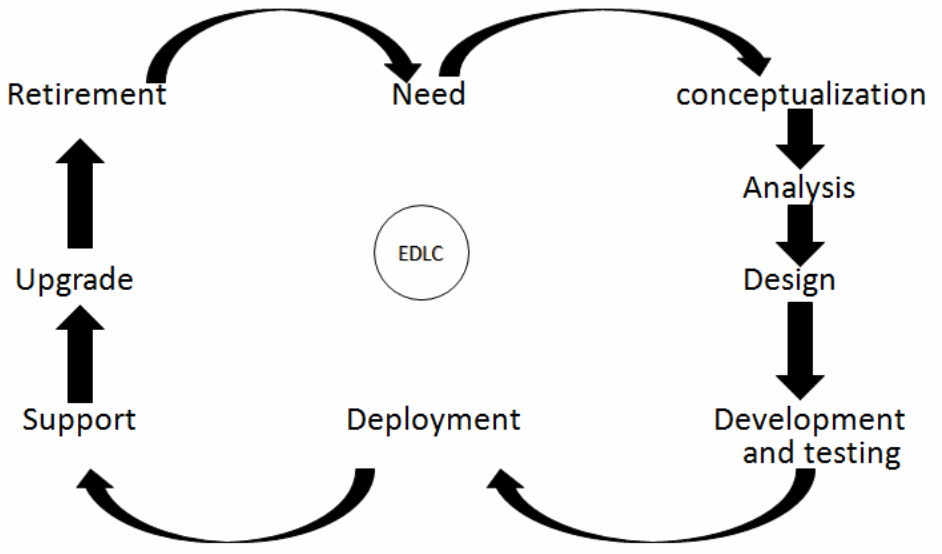


Figure : Phases of EDLC

1. Need

- The need may come from an individual or from the public or from a company.
- 'Need' should be articulated to initiate the Development Life Cycle; a 'Concept Proposal' is prepared which is reviewed by the senior management for approval.

- **Need can be visualized in any one of the following three needs:**
  1. New or Custom Product Development.
  2. Product Re-engineering.
  3. Product Maintenance.

2. Conceptualization

- Defines the scope of concept, performs cost benefit analysis and feasibility study and prepare project management and risk management plans.

- **The following activities performed during this phase:**

  1. **Feasibility Study :** Examine the need and suggest possible solutions.
  2. **Cost Benefit Analysis (CBA):** Revealing and assessing the total development cost and profit expected from the product.
  3. **Product Scope:** Deals with the activities involved in the product to be made.
  4. **Planning Activities:** Requires various plans to be developed first before development like Resource Planning & Risk management Plans.

3. Analysis

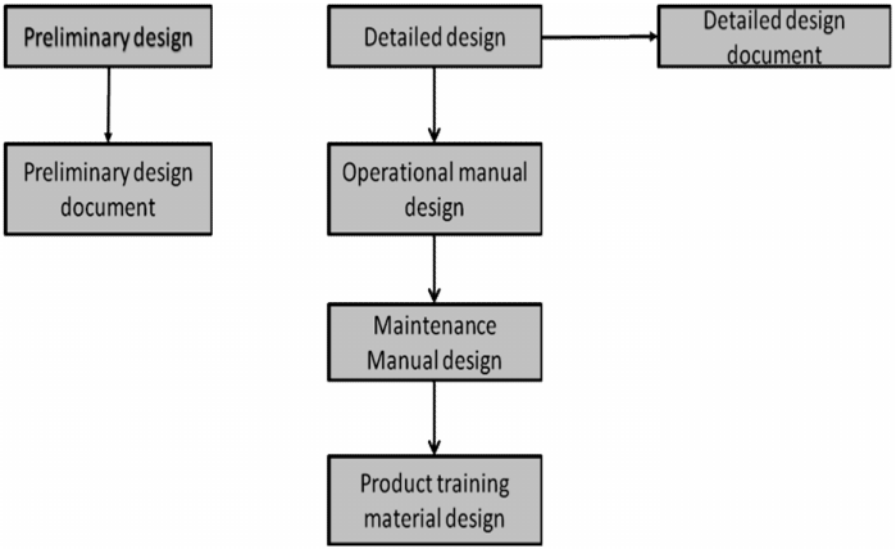
- The product is defined in detail with respect to the inputs, processes, outputs, and interfaces at a functional level.

- **The various activities performed during this phase..**
  - **Analysis and Documentations:** This activity consolidates the business needs of the product under development.
  - **Requirements that need to be addressed..**
    - ❑ Functional Capabilities like performance
    - ❑ Operational and non-operational quality attribute
    - ❑ Product external interface requirements
    - ❑ Data requirements
    - ❑ User manuals
    - ❑ Operational requirements
    - ❑ Maintenance requirements
    - ❑ General assumptions

- **Defining Test Plan and Procedures**: The various type of testing performed in a product development are:
  - ❑ **Unit testing –** Testing Individual modules
  - ❑ **Integration testing –** Testing a group of modules for required functionality
  - ❑ **System testing-** Testing functional aspects or functional requirements of the product after integration
  - ❑ **User acceptance testing-** Testing the product to meet the end user requirements.

4. 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'.



5. 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

6. Deployment

- Deployment is the process of launching the first fully functional model of the product in the market.
- It is also known as First Customer Shipping (FCS).
- **Tasks performed during this phase are:**
  - **Notification of Product Deployment:** Tasks performed here include:
    - ❑ Deployment schedule
    - ❑ Brief description about the product
    - ❑ Targeted end user
    - ❑ Extra features supported
    - ❑ Product support information
  - **Execution of training plan**

    Proper training should be given to the end user top get them acquainted with the new product.
  - **Product installation**

    Install the product as per the installation document to ensure that it is fully functional.
  - **Product post Implementation Review**

    After the product launch, a post implementation review is done to test the success of the product.

7. 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.

8. Upgrades

- 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.
- During the upgrade phase the system is subject to design modification to fix the major bugs reported.

9. Retirement/Disposal

- The retirement/disposal of the product is a gradual process.
- This phase is the final phase in a product development life cycle where the product is declared as discontinued from the market.
- The disposal of a product is essential due to the following reasons
  - Rapid technology advancement
  - Increased user needs

13.4 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
4. Spiral Model

13.5 REVIEW QUESTIONS

1. What is EDLC? Why is it needed? What are its objectives?
2. Draw an neat labeled diagram of the phases of the EDLC and explain any two phases in detail.

13.6 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

❖❖❖❖

14

EDLC MODELS

Chapter Structure

14.0 OBJECTIVES

After reading this chapter you will understand:

- ✓ Some EDLC Models like:
  - ➢ Waterfall or Linear Model
  - ➢ Iterative/ Incremental or Fountain Model
  - ➢ Prototyping Model
  - ➢ Spiral Model

14.1 INTRODUCTION

The previous chapters introduced the readers to what is meant by EDLC. This chapter is meant to explain the various models available under the EDLC.

14.2 WATERFALL MODEL

- ➢ Linear or waterfall model is the one adopted in most of the olden systems.
- ➢ In this approach each phase of EDLC (Embedded Development Product Lifecycle) is executed in sequence.
- ➢ It establishes analysis and design with highly structured development phases.
- ➢ The execution flow is unidirectional.

- The output of one phase serves as the input of the next phase
- All activities involved in each phase are well planned so that what should be done in the next phase and how it can be done.
- The feedback of each phase is available only after they are executed.
- It implements extensive review systems To ensure the process flow is going in the right direction.
- One significant feature of this model is that even if you identify bugs in the current design the development process proceeds with the design.
- The fixes for the bug are postponed till the support phase.
- Advantages
    - Product development is rich in terms of:
        - Documentation
        - Easy project management
        - Good control over cost & Schedule
- Drawbacks
    - It assumes all the analysis can be done without doing any design or implementation
    - The risk analysis is performed only once.
    - The working product is available only at the end of the development phase
- Bug fixes and correction are performed only at the maintenance/support phase of the life cycle.

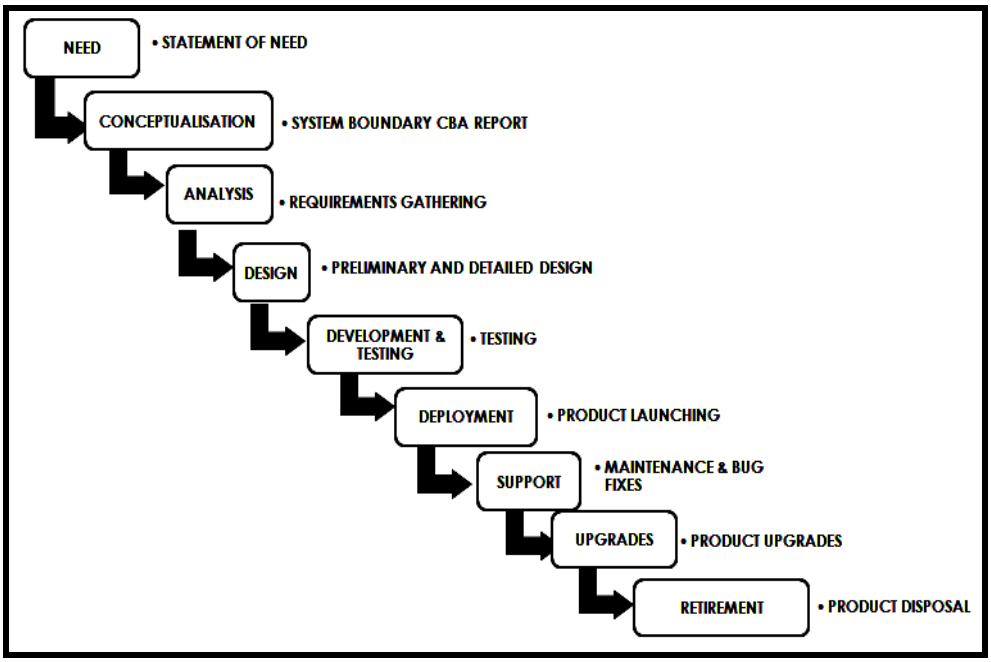


Figure: Waterfall Model

14.3 ITERATIVE/ INCREMENTAL OR FOUNTAIN MODEL

- Iterative and Incremental development is at the heart of a cyclic software development process developed in response to the weaknesses of the waterfall model.
- The iterative model is the repetitive process in which the Waterfall model is repeated over and over to correct the ambiguities observed in each iteration.

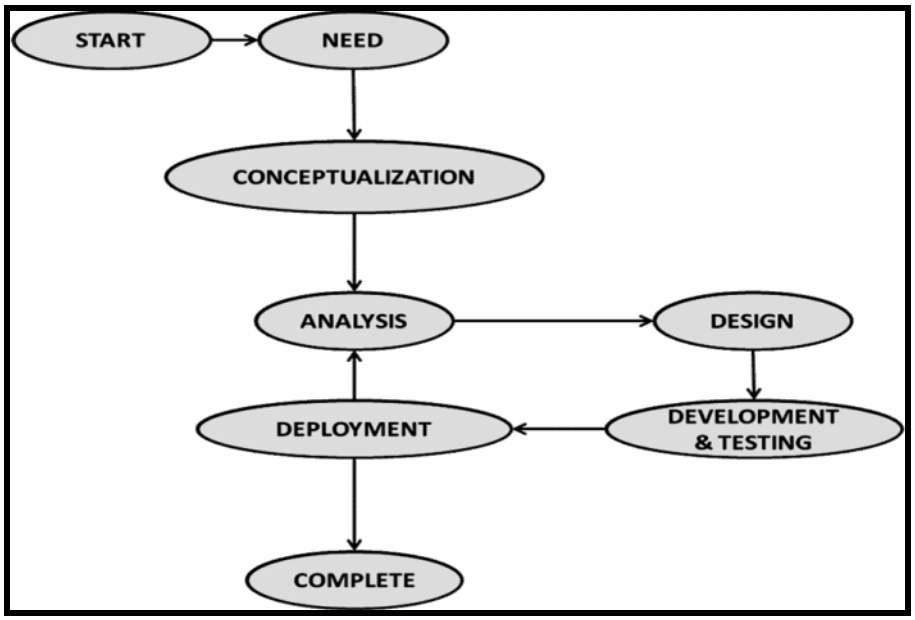


Figure: Iterative model

- The above figure illustrates the repetitive nature of the Iterative model.
- The core set of functions for each group is identified in the first cycle, it is then built, deployed and release. This release is called as the first release.
- Bug fixes and modification for first cycle carried out in second cycle.
- Process is repeated until all functionalities are implemented meeting the requirements.

- **Advantages**
  - Good development cycle feedback at each function/feature implementation
  - Data can be used as reference for similar product development in future.
  - More responsive to changing user needs.
  - Provides working product model with at least minimum features at the first cycle.
  - Minimized Risk
  - Project management and testing is much simpler compared to linear model.
  - Product development can be stopped at any stage with a bare minimum working product.

- **Disadvantages**
  - Extensive review requirement each cycle.
  - Impact on operations due to new releases.
  - Training requirement for each new deployment at the end of each development cycle.
  - Structured and well documented interface definition across modules to accommodate changes

14.4 PROTOTYPING MODEL

- It is similar to iterative model and the product is developed in multiple cycles.
- The only difference is that, Prototyping model produces a 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 the approach-
    - Requirement definition
    - Proto-type development
    - Proto-type evaluation
    - Requirements refining

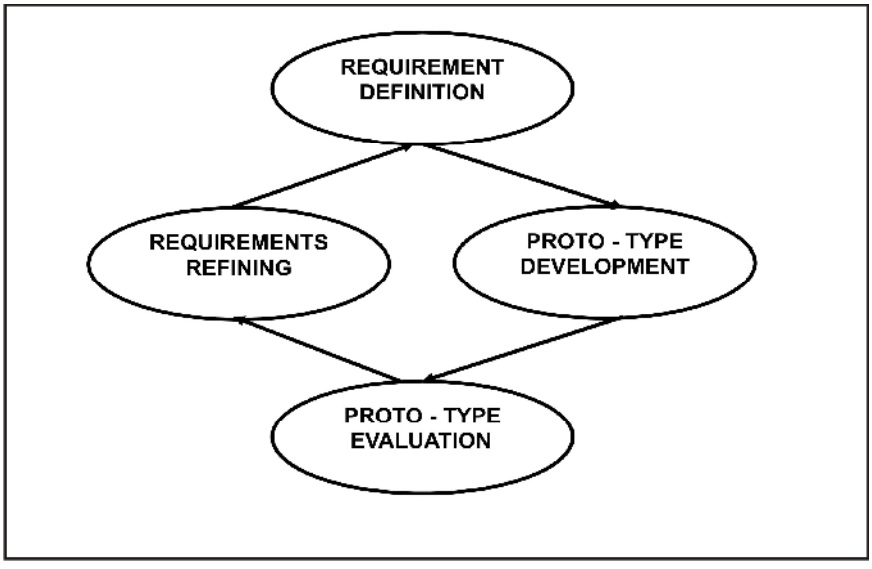


Figure: Prototyping Model

14.5 SPIRAL MODEL

- Spiral model is developed by Barry Boehm in 1988.
- The Product development starts with project definition and traverse through all phases of EDLC(Embedded Product Development Life Cycle).
- The activities involved are:
    I. Determine objectives, alternatives, constraints
    II. Evaluate alternatives, identify and resolve risks
    III. Develop and test
    IV. Plan
- It is a combines the concept of Linear Model and iterative nature of Prototyping Model.

- **Prototyping Model**
    - In prototyping after the requirement analysis the design for the prototype is made and development process is started.
    - On finishing the prototype it is send to the customer for evaluation ie. Judgment.
    - After customer evaluation for the product the feedback is taken from the customer in term of what improvement is needed.
    - Then developer refines the product according to the customer expectation.

- **Linear Model**
- Spiral Model contains the concept of linear model, having following type.
    - Requirement
    - Analysis
    - Design
    - Implementation

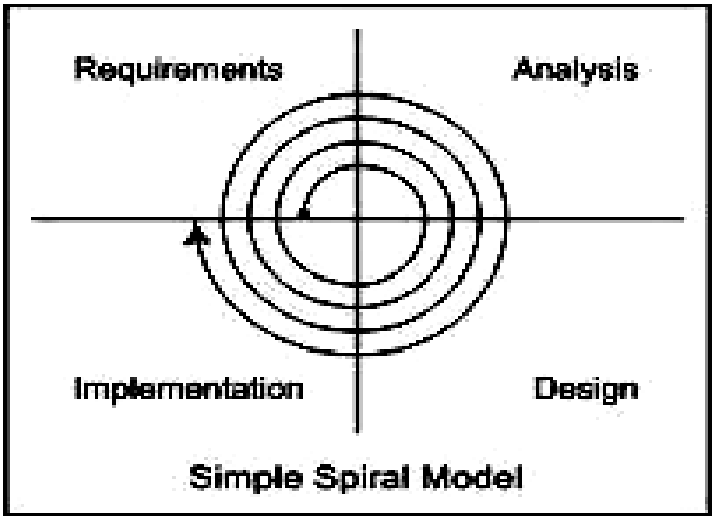


Figure: Spiral Model

1. Requirement:

- This process is focused specifically on embedded software, to understand the nature of the software to be build and what are the requirement for the software.

- And the requirement for both the system & the software is documented & viewed to customer.

2. Analysis:

- Analysis is performed to develop a detailed functional module under consideration.
- The product is defined in detailed with respect to the input, processing & output.
- This phase emphasis on determining 'what function must be performed by the product' & how to perform those function.

3. Design:

- Product design deals with the entire design of the product taking the requirement into consideration.
- The design phase translates requirement into representation**.**

4. Implementation:

- In this process the launching of first fully functional model of the product in the market is done or handing over the model to an end user/client
- In this product modifications are implemented & product is made operational in production environment.

14.6 REVIEW QUESTIONS

1. Explain in detail the Waterfall or Linear Model
2. Explain in detail the Iterative/ Incremental or Fountain Model
3. Explain in detail the Prototyping Model
4. Explain in detail the Spiral Model

14.7 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

15

TRENDS IN EMBEDDED SYSTEMS

Chapter Structure

15.0 OBJECTIVES

After reading this chapter you will understand:

- ✓ Different trends in the embedded industry related to:
  - Processor Trends
  - Operating System Trends
  - Development Language Trends
  - Open Standards, Frameworks and alliances
  - Bottlenecks faced by Embedded Industry

15.1 INTRODUCTION

This concluding chapter describes the trends in the embedded systems industry.

15.2 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.**

A. 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.

B. 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

- These processors implement multiprocessing concept where each core implements pipelining and multithreading.

C. 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 a 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.

15.3 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.

15.4 DEVELOPMENT LANGUAGE TRENDS

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

A. 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.

B. 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.

A. Java

- 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.

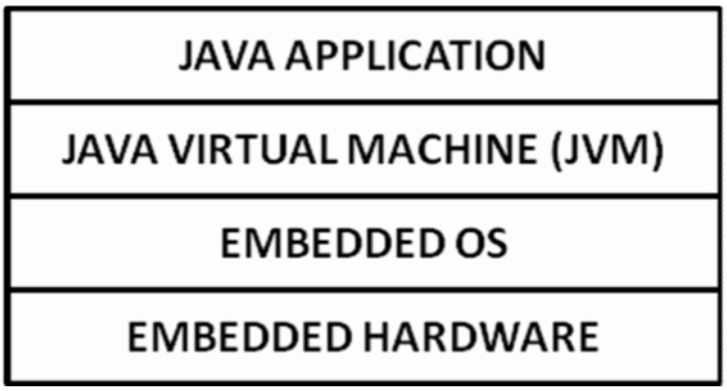


Figure: Java based Embedded Application Development

- 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.

B. .NET CF

- 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.

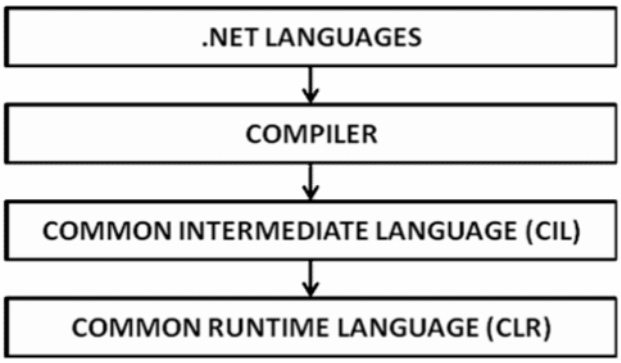


Figure: .NET based Embedded Application Development

15.5 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.

A. 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.

(Source : http://www.openmobilealliance.org)

B. 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.
  (Source : http://www.openhandsetalliance.com/)

C. 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

Source:
http://en.wikipedia.org/wiki/Android_(28operating_system)

D. 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.

(Source: http://en.wikipedia.org/wiki/Openmoko)

15.6 Bottlenecks faced by Embedded Industry

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

A. Memory Performance

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

B. 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.

C. 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 few in embedded software.

15.7 REVIEW QUESTIONS

1. Write a short note on Processor Trends in Embedded Systems
2. Explain the Embedded Operating System Trends
3. Write Short notes on Embedded Development Language Trends
4. Explain Open Standards, Frameworks and alliances
5. Write short note on Bottlenecks faced by Embedded Industry

15.8 REFERENCES & FURTHER READING