S.Y.B.SC (I.T)

SEMESTER - IV

INTRODUCTION TO EMBEDDED SYSTEMS

SUBJECT CODE: USIT402

Programme Co-ordinator : **Shri Mandar Bhanushe**
Head, Faculty of Science and Technology IDOL
Univeristy of Mumbai – 400098

Course Co-ordinator : **Gouri S.Sawant**
Assistant Professor B.Sc.I.T, IDOL
University of Mumbai- 400098

Editor : **Mr Umesh Koyande**
Assistant Professor,
Vidyalankar School Of Information Technology,
Wadala, Mumbai.

Course Writers : **Mr Sabir Moin Shaikh**
Assistant Professor,
Vidyalankar School of Information Technology,
Wadala, Mumbai

: **Mr Mahesh Dhaygude**
Assistant Professor,
Ramsheth Thakur College of Commerce and
Science, Kharghar

February 2022, Print - 1

Published by : Director,
Institute of Distance and Open Learning,
University of Mumbai,
Vidyanagari,Mumbai - 400 098.

DTP composed and Printed by: Mumbai University Press

CONTENTS

S.Y.B.SC (I.T)

SEMESTER - IV

INTRODUCTION TO EMBEDDED SYSTEMS

Unit	Details	Lectures
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. **Characteristics and quality attributes of embedded systems:** Characteristics, operational and non-operational quality attributes.	**12**
II	**Embedded Systems – Application and Domain Specific:** Application specific – washing machine, domain specific - automotive. **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. **Peripherals:** Control and Status Registers, Device Driver, Timer Driver - Watchdog Timers.	**12**
III	**The 8051 Microcontrollers:** Microcontrollers and Embedded processors, Overview of 8051 family. 8051 Microcontroller hardware, Input/output pins, Ports, and Circuits, External Memory. **8051 Programming in C:** Data Types and time delay in 8051 C, I/O Programming, Logic operations, Data conversion Programs.	**12**
IV	**Designing Embedded System with 8051 Microcontroller:** Factors to be considered in selecting a controller, why 8051 Microcontroller, Designing with 8051. **Programming embedded systems:** structure of embedded program, infinite loop, compiling, linking and debugging.	**12**
V	**Real Time Operating System (RTOS):** Operating system basics, types of operating systems, Real-Time Characteristics, Selection Process of an RTOS. **Design and Development:** Embedded system development Environment – IDE, types of file generated on cross compilation, disassembler/ de-compiler, simulator, emulator and debugging, embedded product development life-cycle, trends in embedded industry.	**12**

Books and References:

Sr. No.	Title	Author/s	Publisher	Edition	Year
1.	Programming Embedded Systems in C and C++	Michael Barr	O'Reilly	First	1999
2.	Introduction to embedded systems	Shibu K V	Tata Mcgraw-Hill	First	2012
3.	The 8051 Microcontroller and Embedded Systems	Muhammad Ali Mazidi	Pearson	Second	2011
4.	Embedded Systems	Rajkamal	Tata Mcgraw-Hill		

Unit 1

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 recognized 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

❄❄❄❄❄❄❄

Unit 1

2

CORE OF EMBEDDED SYSTEM

Unit Structure

2.0 Objectives

- From this chapter student will be able to explain difference between microprocessor and controller
- Students will gain the knowledge different communication interfaces.
- Students will be able to explain working of various communication devices associated with embedded system

2.1 Difference between Microprocessor and Microcontroller:

Micro-processor	Micro-controller
1. In general, a microprocessor is a general-purpose device that finds its application in most of the electronic devices.	1. Microcontroller is a specific purpose device which has specific task for specific device.
2. It is a dependent unit that requires other chips for its operation.	2. It is an independent device that does not require any other specific chips.
3. Microprocessor is an I.C. which contains many useful functions.	3. It is called as microchip which contains the components of microprocessor.
4. It requires external memory device to store set of instructions to carry out user-defined task.	4. It has the ability to execute a stored set of instruction to carry out user-defined task.
5. Example :- 8085.	5. Example :- 8051.

2.2 RISC (Reduced Instruction Set Computing)

- RISC is designed to perform smaller number of types of computer instructions so that it can operate at higher speed.
- The range of instruction is 30 to 40.
- Since each instruction type that a computer must perform requires additional transistors and circuitry a large list or a set of computer instruction tends to make microprocessor more complicated and slower in operation.
- It is a type of microprocessor architecture that utilizes a small highly optimized set of instruction rather than a more specialized set of instruction

often found n other types of architecture.

- JOHN COCKE of IBM invented the RISC concept in 1974 by providing that 20% of the instruction in a computer did 80% of its work.
- MACINTOSH computer uses RISC microcomputers.

2.3 CISC (Complex Instruction Set Computing)

- A complex instruction set computer (CISC /ˈsɪsk/) is a computer in which single instructions can execute several low-level operations (such as a load from memory, an arithmetic operation, and a memory store) or are capable of multi-step operations or addressing modes within single instructions.
- The term was retroactively coined in contrast to reduced instruction set computer (RISC) and has therefore become something of an umbrella term for everything that is not RISC, from large and complex mainframe computers to simplistic microcontrollers where memory load and store operations are not separated from arithmetic instructions.
- A modern RISC processor can therefore be much more complex than, say, a modern microcontroller using a CISC-labeled instruction set, especially in the complexity of its electronic circuits, but also in the number of instructions or the complexity of their encoding patterns.
- The only typical differentiating characteristic is that most RISC designs use uniform instruction length for almost all instructions, and employ strictly separate load/store-instructions.

2.3.1 Difference between RISC and CISC

Sr.No.	RISC	CISC
1	Multiple register set often consisting of more than 256 registers.	Single register set, typically 6 to 16 registers total.
2	Three register operand allowed per instruction(eg. Add R1,R2,R3)	Only one or two register operands allowed per instruction(eg. Add R1,R2)
3	Parameter passing through efficient on-chip register windows.	Parameter passing through inefficient off-chip register windows.

4	Single cycle instruction except for load and store instruction	Multiple cycle instruction
5	Hardwired controlled	Microprogrammed controlled
6	Highly pipelined	Less pipelined
7	Fixed length of instruction (30-40 instruction)	Variable length instruction
8	Only load and store instruction can access memory	Many instructions can access memory.
9	Few addressing modes	Many addressing modes.

2.4 Big Endian And Little Endian

- BE and LE are terms that describes the order in which the sequence of bytes (group of 8 bits) are stored in computer memory.
- BE is an order in which BIG END (MSB) is stored first at the lowest storage address while LSB is stored at the higher storage address.
- For ex-If we want to store 5678H at memory locations 2002H and 2003H than the order of storage will be 2002H=56H,2003H=78H.
- LE is an order in which LITTELE END (LSB) is stored first at the lowest storage address while MSB is stored at the higher storage address.
- For ex-If we want to store 5678H at memory locations 2002H and 2003H than the order of storage will be 2002H=78H,2003H=56H.

2.5 Types of Processors

1. **GENERAL PURPOSE and DOMAIN SPECIFIC PROCESSOR**.
    - Almost 80% of embedded system is based on microprocessor or microcontroller.
    - Most of the embedded system is used in industry as well as those involving monitoring applications makes use of microprocessor and microcontroller.
    - Applications requires signal processing such as speech coding ,such as speech recognition makes use of special kind of digital signal processor (DSP) .

2. APPLICATION SPECIFIC INTEGRATED CIRCIUT (ASIC)

- ASIC is a microchip designed to perform some specific function or task.
- It is basically a microchip customized for a particular use rather than intended for general purpose use.
- For ex-A chip design solely to run a cell phone is an ASIC.
- ASIC are categorized according to the technology used for manufacturing. Hence ,are the following types of ASIC :

i. FULLY-CUSTOMIZED ASIC

- ➢ Fully-customized ASIC are those IC's which cannot be modified to suit different applications.
- ➢ Fully customized ASIC's are those that are entirely tailor filtered to a particular application.
- ➢ Since its ultimate design and functionality is pre-specified by the user.
- ➢ The use of pre-defined mask for manufacturing leaves no option for circuit modification during fabrication except for some minor fine tunings or calibration and is generally produced as a single specific product for a particular application only.

ii. SEMI-CUSTOMIZED ASIC

- ➢ These ASIC can be modified partially to serve different functions within its general area of application.
- ➢ Unlike fully-customized ASIC, semi-customized ASIC are designed to allow certain degree of modification during manufacturing process.

iii. STRUCTURED/PLATFORM ASIC

- ➢ Structured or platform ASICs belongs to relatively new ASIC classification.
- ➢ These are those ASIC which have been designed and produced from a tightly defined set of designed methodologies, intellectual properties and well characterized silicon.
- ➢ The aim to developed this type of ASIC is to shorten design cycle and minimizing development cost.
- ➢ A platform ASIC is built from group of platform slices with a platform slice being defined as a pre-manufactured device system or logic for that platform

3. PROGRAMMABLE LOGIC DEVICE (PLD)

- In digital electronic system there are only three kinds of devices that are memory, microprocessor and logic devices.
- Memory devices store random information such as database.
- Microprocessor executes software instruction to perform a wide variety of tasks such as running a word processing program or a video game.
- Logic devices provide specific function including device to device interfacing data communication. Signal processing data display timing and control operations and almost every other function assistive must perform.
- PLD is an electronic component use to build re-configurable digital circuit.
- Un-like a logic gate which has fixed function a PLD has un-defined function at the time of manufacture.
- Before a PLD can use in a circuit it must be programmed i.e. re-configured.
- PLD's are chip that can be programmed and reprogrammed to implemented different logic function.
- The main reason to produce PLD is to reduce total cost.
- Designing with PLD is faster due to which it reduces the time require to bring the product to the market.
- It also reduces the risk associated with the product development since they allow last minute changes without having to re-designed circuit boards.
- There are two types of PLD's:
  - i. Fixed logic device (FLD)
  - ii. Programmable logic device (PLD)
- ADVANTAGES OF PLD
  - ➢ Less board space is required
  - ➢ Faster in speed.
  - ➢ Lower power requirement.
  - ➢ Less costly assembly process.
  - ➢ Higher reliability (since fewer IC's) and circuit connections are there which helps in making troubleshooting easier.
  - ➢ Availability of design software.

- 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.

3.4 Review Questions

3.4.1 Explain the characteristics of an embedded system

3.4.2 Explain the OperationalQuality Attributes of an embedded system

3.4.3 Explain the non-quality attributes of an embedded system

3.5 References & Further Reading

3.5.1 Programming Embedded systems in C++ by Michael Barr

3.5.2 Introduction to Embedded systems – Shibu K. V

❄❄❄❄❄❄❄

Unit 2

4

EMBEDDED SYSTEMS- APPLICATION AND DOMAIN SPECIFIC

Unit Structure

4.0 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

4.1 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.

4.2 Elements of Embedded Systems.

As defined earlier, 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.

Diagrammatically an embedded system can be represented as follows:

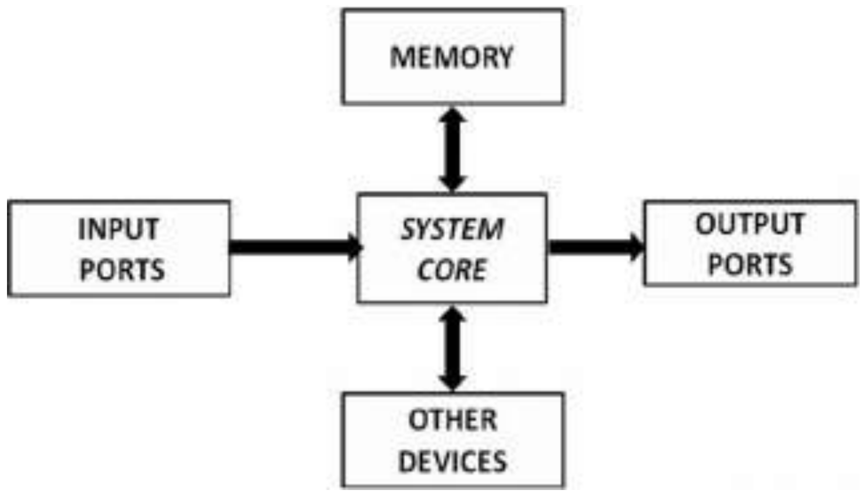


Figure 2.0 : Elements of an Embedded System

- 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.

4.3 Case Studies (Examples)

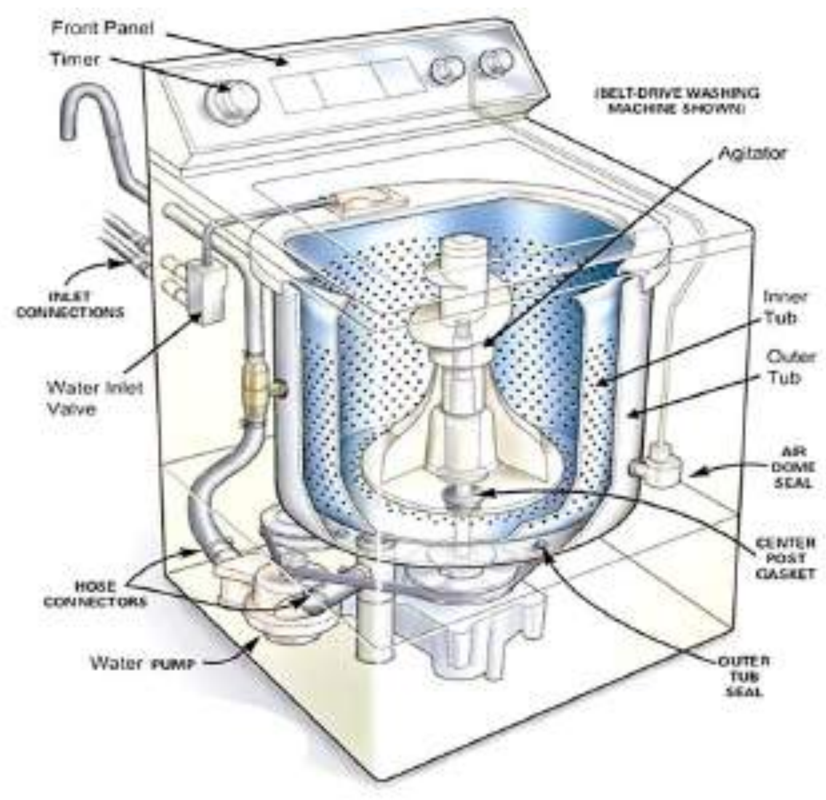
Here are some case studies on some commonly used embedded systems that will help to better understand the concept.

4.3.1 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.



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 the inner tub and supports it during various cycles of clothes washing.

- 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

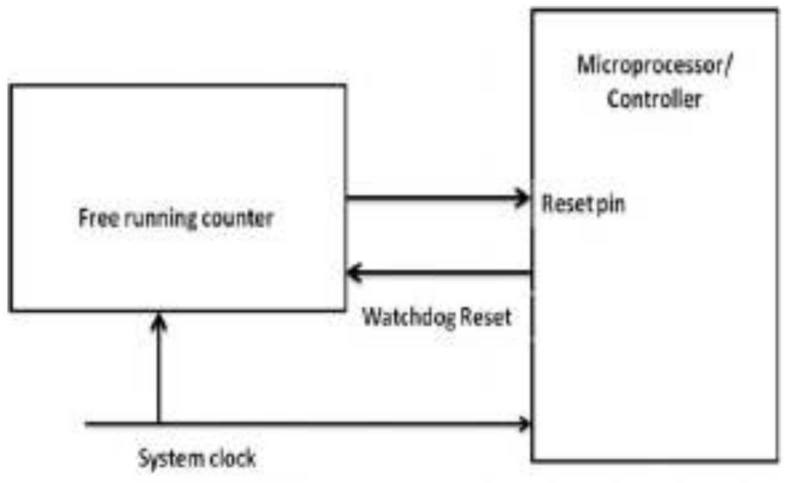


Figure: Watchdog Timer

7.7 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

7.8 References & Further Reading

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

❄❄❄❄❄❄❄

Unit 3

8

THE 8051 MICROCONTROLLERS

Unit Structure

8.0 Objectives

To know the all-Basic things about Microcontroller, Microprocessor (Embedded) & family. How we use Microcontroller & Its Applications in various fields.

To study how we select Microcontroller. To study of Memories & Data conversion useful in Microcontroller. It can be thought of as a computer hardware system having software embedded in it. An embedded system can be either an independent system or a part of a large system including mainly Microcontroller.

8.1 Introduction

As we know an embedded system as a microcontroller-based, software-driven, reliable, real-time control system, designed to perform a specific task. A **printer** is an example of embedded system since the processor inside it performs only one task namely, getting the data and printing it. An embedded product uses a microprocessor (or microcontroller) to do one task only.

PC can be used for any number of applications such as word processor, print server, bank teller terminal, video game player, network server, or internet terminal. Software for a variety of applications can be loaded and run. PC can perform myriad tasks is that it has RAM memory and an operating system that loads the application software into RAM and lets the CPU run it.

8051 is one of the first most popular microcontroller also known as MCS-51. It introduced it in the year 1981 by Intel Company. The 8051 Microcontroller is one of the most popular general purpose microcontrollers especially designed for embedded systems. It a small chip based on an architecture with support for embedded applications, such as measuring device, security systems, robotics, remoter control applications, scroll message display, etc.

8.2 An Overview

Whatever we make, using Microcontroller that is also possible with Microprocessor, but main thing is cost & Size (Microcontroller implied with compact size & low cost). A system is an arrangement in which all its unit assemble & work together according to protocols. It can also be defined as a way of working, organizing or doing one or many tasks according to a fixed plan. For example, a watch is a time displaying system. Its components follow a set of rules to show time. If one of its parts fails, the watch will stop working. Therefore, we can say in a system, all its subcomponents depend on each other.

As its name suggests, Embedded means something that is attached to another thing. An embedded system can be thought of as a computer hardware system having software embedded in it. An embedded system can be an independent system or it can be a part of a large system. An embedded system is a microcontroller or Microprocessor based system, which is designed to perform a specific task.

8.3 Microcontroller and Embedded Processors

8.3.1 Processors in a System

A processor has two essential units-

- Program Flow **Control Unit** (CU)
- **Execution Unit** (EU)

The CU includes a fetch unit for fetching instructions from the memory. The EU has circuits that implement the instructions pertaining to data transfer operation and data conversion from one form to another.

The EU includes the Arithmetic and Logical Unit (ALU) and the circuits that execute instructions for a program control task such as interrupt or jump to another set of instructions.

A processor runs the cycles of fetch and executes the instructions in the same sequence as they are fetched from memory.

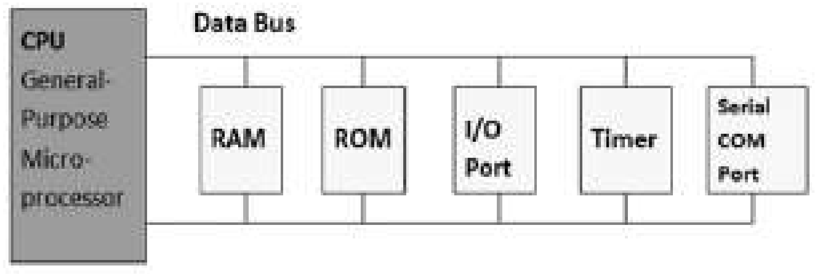
8.3.2 Types of Processors

General Purpose Processor (GPP) can be of the following categories-

- o Microprocessor
- o Microcontroller
- o Embedded Processor
- o Digital Signal Processor
- o Media Processor

Microprocessor:

A microprocessor is a single VLSI chip having a CPU. In addition, it may also have other units such as coaches, floating-point processing arithmetic unit, and pipelining units that help in faster processing of instructions. Earlier generation microprocessors' fetch-and-execute cycle was guided by a clock frequency of order of ~1 MHz Processors now operate at a clock frequency of 2GHz.



Block Diagram of General purpose Microprocessor

Microcontroller:

A microcontroller is a single-chip VLSI unit (also called microcomputer) which, although having limited computational capabilities, possesses enhanced input/output capability and a number of on-chip functional units.

Unit 3

9

C - LANGUAGE

Unit Structure

9.0 Objectives

To learn the different steps involved in the design & Development of C programming in Embedded system. Learn about the fundamentals of Programming, firmware using Embedded C. To study of Data Types and Time Delay, I/O Programming, Logic Operations & Data Conversion Programs in Embedded C & its Applications.

9.1 Introduction

Programming is the most essential part of any Embedded system. Programming is required for Computational tasks. Program in a High level language gives us benefits of short development cycle for a complex system hardware modifications. C began to grow in popularity not just for systems programming, but as a general purpose programming language.
The extension of the C language is called an Embedded C programming language. The C programming language was originally developed as a language to replace assembler in systems programming. It was very successful, making system code portable and easier to write and read.

9.2 An Overview

Each processor is associated with embedded software. Embedded C Programming plays a major role in performing specific functions by the processor. In our day-to-day life, we frequently use many electronic devices such as washing machines, mobile phones, digital camera and so on will work based on microcontrollers that are programmed by embedded C.

The C code written is more reliable, portable, scalable and much easier to understand. The first and foremost tool is the embedded software that decides the operation of an embedded system. Embedded C programming language is most frequently used for programming the microcontrollers. For writing the program the embedded designers must have sufficient knowledge on the hardware of particular processors or controllers as the embedded C programming is a full hardware related programming technique. The 8051 microcontroller is the **8-bit '**CISC' architecture. It consists of memories, serial communication, interrupts, input/output ports and timer/counters, built into a single integrated chip, which is programmed to control the peripheral devices which are interfaced with it.

An example of Embedded system is a Car. A modern day Car has several individual embedded systems that perform their specific tasks with the aim of making a smooth and safe journey. Some of the embedded systems in a Car are Anti-lock Braking System (ABS), Temperature Monitoring System, Automatic Climate Control, Tire Pressure Monitoring System, Engine Oil Level Monitor, etc.

9.3 8051 Programming In C

9.3.1 Basics of Embedded C Program

Embedded C is one of the most popular and most commonly used Programming Languages in the development of Embedded Systems. Therefore we will do some of the Basics of Embedded C Program and the Programming Structure of

Embedded C. Embedded C is perhaps the most popular languages among Embedded Programmers for programming Embedded Systems. There are many popular programming languages like Assembly, BASIC, C++, Python etc. that are often used for developing Embedded Systems but Embedded C remains popular due to its efficiency, less development time and portability.

An Embedded System can be best described as a system, which has both the hardware and software and is designed to do a specific task. A good example for an Embedded System, which many households a Washing Machine. It takes some inputs from the user like wash cycle, type of clothes, extra soaking and rinsing, spin rpm, etc., performs the necessary actions as per the instructions and finishes washing and drying the clothes. If no new instructions are given for the next wash, then the washing machines repeats the same set of tasks as the previous wash. Embedded Systems can not only be stand-alone devices like Washing Machines but also be a part of a much larger system.

9.3.2 Factors for Selecting the Programming Language

The following are few factors that are to be considered while selecting the Programming Language for the development of Embedded Systems.

- **Size**: The memory that the program occupies is very important as Embedded Processors like Microcontrollers have a very limited amount of ROM (Program Memory).
- **Speed**: The programs must be very fast i.e., they must run as fast as possible. The hardware should not be slowed down due to a slow running software.
- **Portability**: The same program can be compiled for different processors.
- Ease of Implementation
- Ease of Maintenance
- Readability

9.3.3 Difference between C and Embedded C

C Language

- C used for desktop based applications
- C code generate the compatible .exe files
- It has limitless resources like memory
- C language uses the desktop OS memory
- C don’t have registers to store data temporarily

Embedded C:

- It is used for micro controller based applications
- Embedded programs generate the .hex files
- It has limited resources like memory (RAM, ROM)
- Embedded programming uses the micro controller inbuilt memory
- It has registers

9.3.4 Keywords in Embedded C

A Keyword is a special word with a special meaning to the compiler (a C Compiler for example, is a software that is used to convert program written in C to Machine Code).

Writing code
Example 1: Blinking of LED connected to Port 1 of 8051.
Program:

```
#include<reg51.h>
//delay function declaration
void delay(void);
void main(void)
{
//an infinite loop
while(1)
{
// Turn ON all LED's connected to Port1
P1 = 0xFF;
delay();
// Turn OFF all LED's connected to Port1
P1 = 0x00;
delay();
}
}
//delay function definition
void delay(void)
{
int i,j;
for(i=0;i<0xff;i++)
for(j=0;j<0xff;j++);
}
```

Example 2: Write a program to generate a 10 KHz square wave using 8051.
Program:

```
#include<reg51.h>
void main()
{
unsigned int x;
for(;;)
{
P1=0X80;
for(x=0;x<40000;x++)
{
P1=0X00;
}
}
}
```

9.4 Data Types and Time Delay In 8051 C

Data Types:

Data Types in C Programming Language (or any programming language for that matter) help us declaring variables in the program. There are many data types in C Programming Language like signed int, unsigned int, signed char, unsigned char, float, double, etc. In addition to these there few more data types in Embedded C.

The following are the extra data types in Embedded C associated with the Keil's Cx51 Compiler.

- bit
- sbit
- sfr
- sfr16

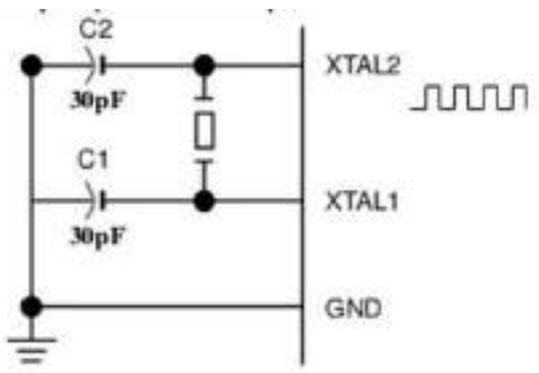
The following table shows some of the data types in Cx51 Compiler along with their ranges.

Data Type	**Bits (Bytes)**	**Range**
Bit	1	0 or 1 (bit addressable part of RAM)
signed int	16 (2)	-32768 to +32767
unsigned int	16 (2)	0 to 65535
signed char	8 (1)	-128 to +127
Unsigned	8 (1)	0 to 255
Float	32 (4)	±1.175494E-38 to ±3.402823E+38
Double	32 (4)	±1.175494E-38 to ±3.402823E+38
Sbit	1	0 or 1 (bit addressable part of RAM)
Sfr	8 (1)	RAM Addresses (80h to FFh)
sfr16	16 (2)	0 to 65535

Time Delay:

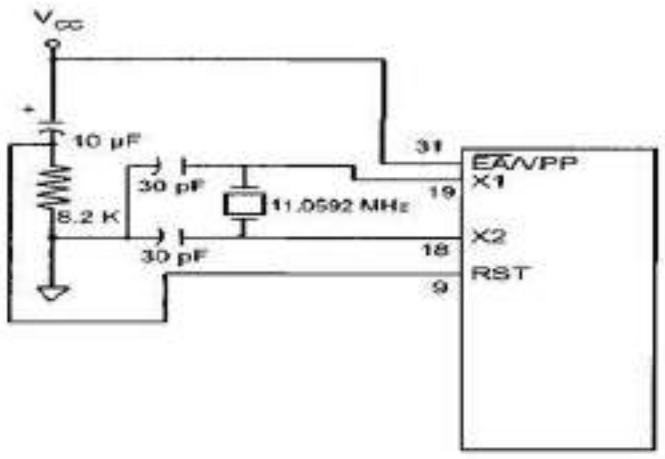
The delay length in 8051 microcontroller depends on three factors:
The crystal frequency
The number of clock per machine
The C compiler.
Code generating delay using timer register:

```
#include <REG52.h>
void T0Delay(void);
void main(void){
```



- **RST (Pin No. 9)** − It is an Input pin and active High pin. Upon applying a high pulse on this pin, that is 1, the microcontroller will reset and terminate all activities. This process is known as Power-On Reset. Activating a power-on reset will cause all values in the register to be lost. It will set a program counter to all 0's. To ensure a valid input of Reset, the high pulse must be high for a minimum of two machine cycles before it is allowed to go low, which depends on the capacitor value and the rate at which it charges. (Machine Cycle is the minimum amount of frequency a single instruction requires in execution).

- **EA or External Access (Pin No. 31)** − It is an input pin. This pin is an active low pin; upon applying a low pulse, it is activated. In case of microcontroller (8051/52) having on-chip ROM, the EA (bar) pin is connected to Vcc. However, in an 8031 microcontroller that does not have an on-chip ROM, the code is stored in an external ROM and then fetched by the microcontroller. In this case, we must connect the (pin no 31) EA to Gnd to indicate that the program code is stored externally.



- **PSEN or Program store Enable (Pin No 29)** - This is also an active low pin, i.e., it is activated after applying a low pulse. It is an output pin and used along with the EA pin in 8031 based (i.e. ROMLESS) Systems to allow storage of program code in external ROM.

- **ALE or (Address Latch Enable)** - This is an Output Pin and is active high. It is especially used for 8031 IC to connect it to the external memory. It can be used while deciding whether P0 pins will be used as Address bus or Data bus. When ALE = 1, then the P0 pins work as Data bus and when ALE = 0, then the P0 pins act as Address bus.

9.5.4 I/O Ports And Bit Addressability

It is a most widely used feature of 8051 while writing code for 8051. Sometimes we need to access only 1 or 2 bits of the port instead of the entire 8-bits. 8051 provides the capability to access individual bits of the ports.

While accessing a port in a single-bit manner, we use the syntax "SETB X. Y" where X is the port number (0 to 3), and Y is a bit number (0 to 7) for data bits D0-D7 where D0 is the LSB and D7 is the MSB. For example, "SETB P1.5" sets high bit 5 of port 1.

The following code shows how we can toggle the bit P1.2 continuously.

```
AGAIN:
SETB    P1.2
ACALL   DELAY
CLR     P1.2
ACALL   DELAY
SJMP    AGAIN
```

9.5.5 Single-Bit Instructions

Instructions	Function
SETB bit	Set the bit (bit = 1)
CLR bit	clear the bit (bit = 0)
CPL bit	complement the bit (bit = NOT bit)
JB bit, target	jump to target if bit = 1 (jump if bit)
JNB bit, target	jump to target if bit = 0 (jump if no bit)
JBC bit, target	jump to target if bit = 1, clear bit (jump if bit, then clear)

12.1 Objectives

- To learn about the characteristics of Real time Operating system (RTOS).
- To have a knowledge about embedded operating system & its building blocks.
- To understand the kernel services of an operating system.
- To learn about Operating system basics & selection of RTOS.
- To Design & Develops RTOS.
- To study Embedded IDE.
- To understand the concept of Embedded Development Life cycle (EDLC).

12.2 Introduction

We know that an embedded operating system is a small-scale computer system with a limited number of features. It is designed to carry out a function or a set of functions of an electronic product. Embedded operating system is also known as Real time operating system (RTOS). In the most embedded OSs, the application are built in to the OS or part of the OS, so they are loaded immediately when the OS starts.

For instance, all mobile phones have an integrated embedded operating system software like Android or IOS that starts up when the phone is switched on. Many modern electronic devices are based on Arduino or Raspberry PI. Raspberry PI devices often run an ARM-based Linux kernel, but there are actually a number of different operating systems that can be run on Raspberry PI devices.

In contrast to an operating system for a general-purpose computer, an embedded operating system can be quite limited in terms of function depending on the device in question; the system may only run a single application. Modern systems require better functionality, more options and opportunities. That is why the popularity of real-time operating systems is rapidly growing in the world of embedded solutions.

Guaranteeing the timely execution of high-priority tasks is extremely important for critical or lifesaving applications and real-time systems that have strict deadlines. One of the most obvious reasons is multitasking. This is unnecessary with a simple system that only has a couple of tasks running in sequence. However, if you need to perform many tasks simultaneously, RTOS is at your service. It allows the developer to run a number of tasks concurrently. In case your embedded system has strict task prioritization, the pre-emptive scheduler of an RTOS makes it possible to switch to a high-priority operation at any time, ensuring that it will be completed first. A developer has no need to control the task execution because an RTOS takes care of it.

12.3 Overview

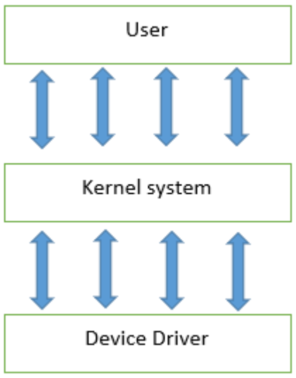
A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter increments of time. In a RTOS, Processing time requirement are calculated in tenths of second's increments of time. Time-bound system can be defined as fixed time constraints. In this type of system, processing must be done inside the specified constraints. Otherwise, the system will fail.

Real-time operating systems (RTOSs) are often used to develop applications for systems with complex time and resource constraints. This situation often specifies laboratory automation where one or more computers must synchronize the activities among one or more instruments involving time, process, or precedent constraints. RTOSs are deterministic by design, which allows them to meet deadlines associated with external events using a limited set of resources. Advances in modern development tools and frameworks have made RTOSs more accessible to developers of all levels.

Developing applications for RTOSs used to be a job for the most skillful developers not only due to the complexity of the application, but also due to the need to use in-circuit emulators or very sophisticated cross-development platforms. Advances in tools, languages and frameworks, however, have made the development of applications for real-time systems easier. In the next sections, we examine certain important aspects of RTOSs. Our intent is to provide information to allow you to understand the concepts in this area, to help you determine when a RTOS will benefit an application and to stimulate your interest to learn more.

12.4 Operating System Basics -

The Operating system is a software in between hardware device and user. Operating system is a platform from where user can easily get communication with hardware device. Operating system manages the all Input/output services.



- **Start state**

  The task has been created and memory allotted to its structure λ However, it is not ready and is not schedulable by kernel.

- **Ready (Active) State**

  The created task is ready and is schedulable by the kernel but not running at present as another higher priority task is scheduled to run and gets the system resources at this instance.

- **Running state**

  Executing the codes and getting the system resources at this instance. It will run till it needs some IPC (input) or wait for an event or till it gets preempted by another higher priority task than this one.

- **Waiting state**

  A task is pending while it waits for an input from the keyboard or a file. The scheduler then puts it in the blocked state

- **Terminated or Exit**

  Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

12.4.2 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.

MULTIPLE TASKS WAITING:

So far we have only dealt with the simple case of two task synchronisation. We now address the case where multiple tasks are waiting at the synchronisation point:

- **Long Term Scheduler:-**

  It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

  The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

- **Short Term Scheduler**

  It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

- **Medium Term Scheduler**

  Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes. A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping** and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

12.4.4 Context Switch

A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique, a context switcher enables multiple

12.5.2 Real-Time OS

Real Time system complete its work & deliver its services on time. Real-Time operating systems serve real-time systems & implies deterministic in timing behavior. RTOS are useful where many events occur in a short time or certain deadlines, such as real-time simulations. Functions of RTOS are Task management, Scheduling, Resource allocation, Interrupt handling etc.

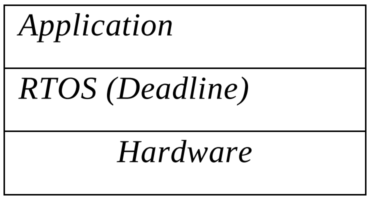


Fig - Simple RTOS

The super loop concept – Task Execution model for firmware executes the task subsequently in order which the tasks listed within the loop. Here every task is repeated at regular interval & task execution is non real time. Because there are no operating system to return to or an embedded device is running until the power supply is removed. So, to run set of statements, we need **a loop that must not be finished**, such kind of loops are known as 'Super Loop' or 'Infinite Loop'.

Advantages of Real-Time OS

- It provides more output from all the resources as there is maximum utilization of embedded systems.
- Work done within deadlines.
- It provides the best management of memory allocation.
- These systems are always error-free.
- These operating systems focus more on running applications than those in the queue.
- Shifting from one task to another takes very little time

Disadvantages of Real-Time OS

- System resources are extremely expensive and are not so good.
- The algorithms used are very complex.
- Only limited tasks can run at a single time.
- In such systems, we cannot set thread priority as these systems cannot switch tasks easily.

produce correct result within the given time interval. If the result is not obtained within the given time interval then also result is not considered correct. In real-time systems, correctness of result is to obtain correct result in time constraint.

Embedded:-
all the real-time systems are embedded now-a-days. Embedded system means that combination of hardware and software designed for a specific purpose. Real-time systems collect the data from the environment and passes to other components of the system for processing.

Safety:-
Safety is necessary for any system but real-time systems provide critical safety. Real-time systems also can perform for a long time without failures. It also recovers very soon when failure occurs in is system and it does not cause any harm to the data and information.

Concurrency:-
Real-time systems are concurrent that means it can respond to a several number of processes at a time. There are several different tasks going on within the system and it responds accordingly to every task in short intervals. This makes the real-time systems concurrent systems.

Distributed:-
In various real-time systems, all the components of the systems are connected in a distributed way. The real-time systems are connected in such a way that different components are at different geographical locations. Thus all the operations of real-time systems are operated in distributed ways.

Stability:-
Even when the load is very heavy, real-time systems respond in the time constraint i.e. real-time systems does not delay the result of tasks even when there are several task going on a same time. This brings the stability in real-time systems.

12.7 Selection process of RTOS

Colin Walls of mentor graphics discussed whether or not you need to use an OS and if so, whether it will be a free, open source version. While selecting real time operating system for an embedded system Performance is the most important factor required to be considered while selecting for a RTOS. RTOS systems are error-free. Therefore, there is no chance of getting an error while performing the task. A good RTS should be capable, and it has some extra features like how it operates to execute a command, efficient protection of the memory of the system, etc.

12.9.2.3 Microsoft Visual Studio

Visual Studio is an IDE for embedded software development that allows you to develop both console applications and applications with a graphical interface, including those with support for Windows Forms technology. It is also suitable for building websites, web applications, and web services for all supported platforms.

Advantages:

- A free version; Built-in command-line interface;
- API for connecting additional debugging tools;
- A complete set of developer tools for creating and cloning Git repositories, managing branches, and resolving merge conflicts right in the C ++ IDE;
- An extensive set of add-ons to expand the basic functionality.

Disadvantages:

- High cost of paid versions – Professional and Enterprise (from $ 45 per month);
- High requirements for “hardware”;
- Lack of any Linux versions.

12.9.2.4 CodeLite

CodeLite is a free embedded software development that runs on a variety of operating systems. The interface is intuitive and straightforward, making it a suitable choice for beginners. Note that the latest versions of this C ++ IDE support PHP and Node.js projects.

Advantages:

- Powerful code completion tool based on its parser
- Plugins for working with Git and SVN & Built-in debugger.

Disadvantages: Complicated interface

12.9.3 Software Development Steps

In any environment, to develop executable software you need to create source file(s), compile the source files to produce machine code (object files), and link the object files with each other and any libraries or other resources required to produce an executable file.

Source files contain the code statements to do the tasks your program is being created for. They contain program statements specific to the language you are using. If programming in c, the source files contain c code statements; java source files contain java statements. Usually source files names have extensions indicating the code they contain. A c source file may be named "myfile.c". Compilers translate the source files to appropriate machine level code for the target environment. Linkers take all the object files required for a program and link them together, assigning memory and registers to variables, setting up data. They also link in library files to support operating system tasks and any other files the program needs. Linkers output executable files.

12.9.4 Life Without IDEs

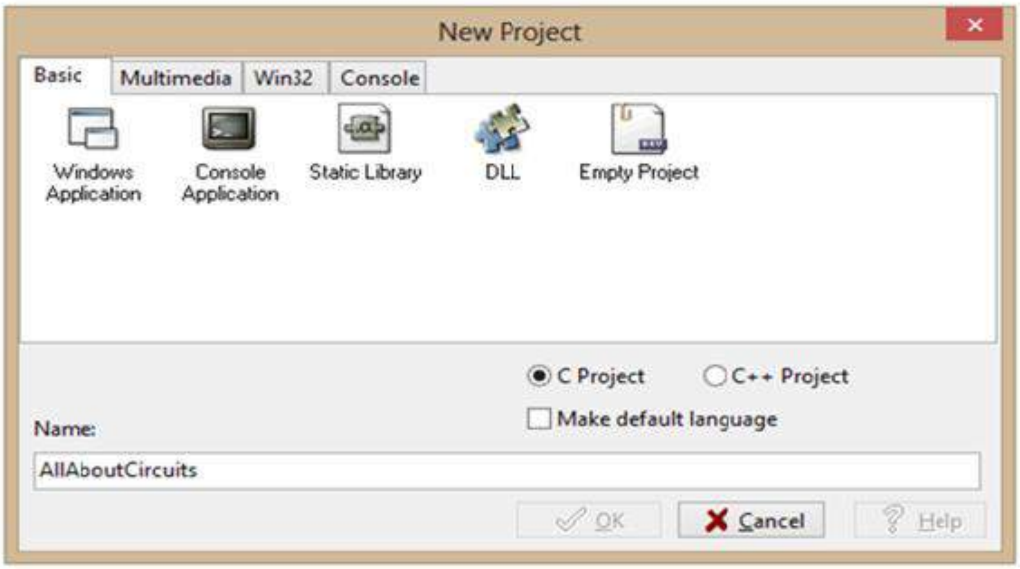
When not using an IDE, developers use an editor, compiler, and linker installed on their development machine to create code files, compile, and link them. Using the editor to create a source file, the code blocks, comments, and program statements are entered and the file saved. There are no "corrective actions," taken as the editor doesn't know this is supposed to be a "source file" as opposed to notes for class! If working in a position-dependent language like Python, the developer would have to be very careful about indenting. The file has to be saved with the correct file extension and in a directory where the compiler can find it.

Each source file has to be compiled separately; if the program has a few source files, they all have to be named separately in the compiler. When invoking the compiler, it has to be directed to look in the correct directory for the source files and where the output files should be stored. If there is an error in the source file, the compiler will output messages and fail to complete. For any errors, the developer goes back and edits the source file, working from line numbers and compiler messages to fix the problems... and these steps continue until all the source files compile without errors.

When linking, each object file is specified as being part of the build. Again, the locations for the object files and executable are given. There may be errors at this point because it isn't until the entire program is linked that some errors can be detected. Assuming the linker finds all the variables and functions, it produces a file that can be run. If the program is run and it works, all's well! If it seems to do nothing.... that means it's debugging time! Since there is no insight to what the program is doing, the developer may go back and put in some brute force methods, like print statements to print messages out at certain points in the program or blink some LEDs at strategic places, which means back to the editor, and the cycle continues.

12.9.5 Using IDEs

Bringing up an IDE presents a workspace & with an IDE, a Project provides a workspace where all the files for a program can be collected. We need to select the language and the type of program to create. This IDE supports c and C++ and various application types



[✓ C Project] C++ Project

☐ Make default language

Name: AllAboutCircuits

If set for a Windows Application in C, it brings up a template: A console C program brings up a different template: Depending on the IDE, it may set up code blocks automatically, indent as required, track variable names in colors, show comments. Compile? Just click the compile selection on the dropdown menu (or press F9). Compiler results will show in one of the windows and in the log.

Compiler options and directories are set up using the options menus. As source files are created, they are added to the project. The Rebuild selection rebuilds all the files, first checking for the latest versions, then compiles and links to produce an executable result. Errors on the compile or link? The offending code will be shown in the code window. The statement containing the error or the lines around it is known, since the compiler, linker, and editor are seamlessly connected. You can run the executable from the IDE by selecting Run: The results show in a separate window. Problems when running your new program? Usually IDEs provide an option to create a debug version.

With a debug version, the IDE controls the execution of the program, allowing insight to data variables and memory locations. Some IDEs show both the high level source statements as well as the machine code. The debugger may include options to "watch" local variables and track the contents of memory locations, offer line by line execution, provide the ability to set break points to run to a certain point in the program, and the ability to step into or over function calls. Some IDEs include emulators, allowing debugging in the IDE environment without having to export the code to the target device.

- **Cross-compiler:** A cross compiler is necessary to compile code for multiple platforms from one development host. Direct compilation on the target platform might be infeasible, for example on embedded systems with limited computing resources.
- **Editor:** A source code editor is a text editor program designed specifically for editing source code to control embedded systems. It may be a standalone application or it may be built into an integrated development environment (e.g. IDE). Source code editors may have features specifically designed to simplify and speed up input of source code, such as syntax highlighting and auto complete functionality. These features ease the development of code
- **Compiler:** A compiler is a computer program that translates the source code into computer language (object code). Commonly the output has a form suitable for processing by other programs (e.g., a linker), but it may be a human readable text file. A compiler translates source code from a high level language to a lower level language (e.g., assembly language or machine language). The most common reason for wanting to translate source code is to create a program that can be executed on a computer or on an embedded system. The compiler is called a cross compiler if the source code is compiled to run on a platform other than the one on which the cross compiler is run. For embedded systems the compiler always runs on another platform, so a cross compiler is needed.
- **Linker:** A linker or link editor is a program that takes one or more objects generated by compilers and assembles them into a single executable program or a library that can later be linked to in itself. All of the object files resulting from compiling must be combined in a special way before the program locator will produce an output file that contains a binary image that can be loaded into the target ROM. A commonly used linker/locater for embedded systems isld (GNU).

12.11 Types of File Generated on cross compilation

The various 5 types of files generated during cross compilation process are as follows –

1) **List Files (.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.

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.

5) **Hex File (.hex) -** It is a binary executable file created from the source code. The file created by linker/locater is converted into processor understandable binary code. The tool used for converting and object file into a hex file is known as object to Hex converter. Hex file have specific format and it varies for different processor and controller. Two commonly used hex file format are Intel Hex & Motorola Hex. Both Intel and Motorola hex file format represent data in the form of ASCII codes.

12.12 Disassembler / De-Compiler :

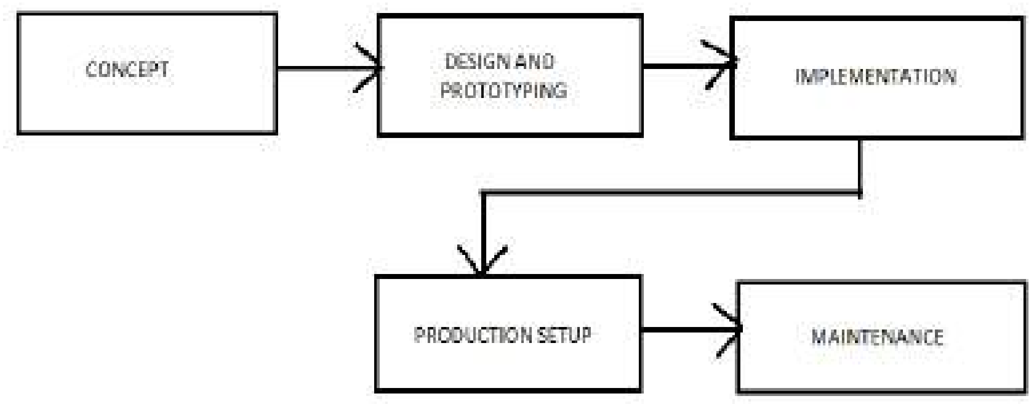
A disassembler is **a computer program that translates machine language into assembly language** the inverse operation to that of an assembler. Some disassemblers make use of the symbolic debugging information present in object files such as ELF. A disassembler differs from a decompiler, which targets a high-level language rather than an assembly language. A **decompiler** is a computer program translates an executable file in to a high-level source file that can be recompiled successfully. A disassembler is software that converts machine language instructions into assembly language instructions

12.13 Simulators & Emulator

Some simulators go even a step further and include the whole system (simulation of peripherals outside of the microcontroller). No matter how fast PC, there is no simulator on the market that can actually simulate a microcontroller's behavior in real-time. Simulating external events can become a time-consuming exercise, as you have to manually create "stimulus" files that tell the simulator what external waveforms to expect on which microcontroller pin. A simulator can also not talk to your target system, so functions that rely on external components are difficult to verify. For that reason simulators are best suited to test algorithms that run completely within the microcontroller.

An emulator is a piece of hardware that ideally behaves exactly like the real microcontroller chip with all its integrated functionality. It is the most powerful debugging tool of all. A microcontroller's functions are emulated in real-time.

Emulator:



Concept:

- Description of the original idea in a formal technical form (verbal requirements)
- Investigation of the existing prototypes and/or models that match the idea
- Comparative analysis of existing implementations
- Proposal of implementation and materials options

Design: Functional Requirements:

- Development of hardware functional specification
- Development of software and firmware functional requirements
- Analysis of the third-party requirements documentation

Architecture Design:

- Development of the system architecture concept
- Design of the mechanics parts of the system
- Development of hardware design documentation (including FPGA design)
- Development of the detailed software design specification
- Analysis of the third-party design documents

Hardware Modelling:

- Schematics design
- PCB Layout Design
- Re-engineering and repairing
- Samples & Prototypes Assembly

Prototyping:

- Product prototyping (including all types of mechanics, hardware, software and the whole system prototyping)
- Mechanical parts manufacturing (including press forms manufacturing)

- Hardware development
- Software and firmware coding
- System integration (software with hardware and mechanics)

Implementation:

Porting:

- Porting of an existing system to a new hardware platform
- Product certification (preparation of hardware and software for further certification process)

System Optimization:

- Optimization of system performance, usability, cost, time to market and more
- Analysis of the third-party implementation with suggested improvements
- System benchmarking documentation

Testing/Debugging:

- Creation of a sophisticated test system to verify a product on each life cycle stage
- Development of testing documentation
- Remote hardware test system setup to allow customer run their own applications in a sophisticated hardware/software environment
- Quality improvement by analyzing the third-party products for existing caveats and issues, and performing the corresponding debugging

Transition to Manufacturing:

- Schematics design
- PCB Layout Design
- Re-engineering and repairing
- Samples & Prototypes Assembly

Maintenance:

- Debugging of the known problems
- Development of an ECO system by developing additional demo applications that can be used as a starting base for the system development
- System upgrades (new hardware, new software, new mechanics etc).

12.16 TRENDS IN EMBEDDED INDUSTRY

The embedded systems industry was born with the invention of microcontrollers and since then it has evolved into various forms, from primarily being designed for

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.

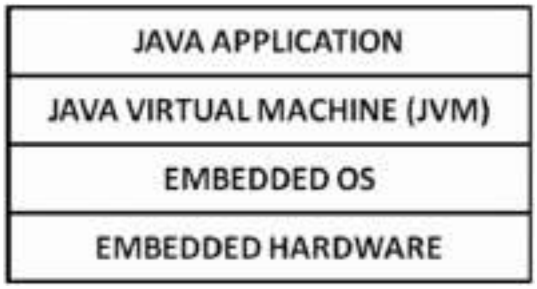


Figure: Java based Embedded Application Development

- 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:
  - o For real time applications java is slow
  - o Garbage collector of Java is non-deterministic in behavior which makes it not suitable for hard real time systems.
  - o Processors need to have a built in version of JVM
  - o Those processors that don't have JVM require it to be ported for the specific processor architecture.
  - o Java is limited in terms of low level hardware handling compared to C and C++
  - o 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.

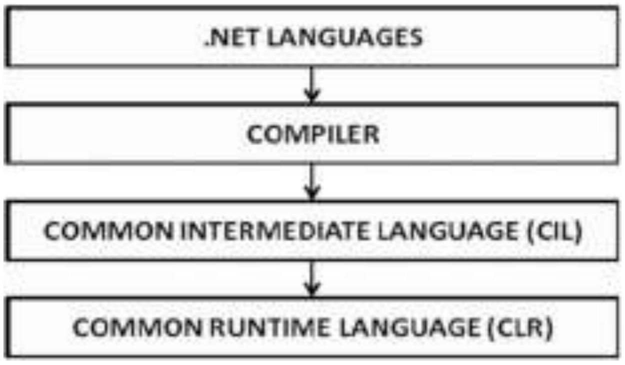

.NET LANGUAGES
COMPILER
COMMON INTERMEDIATE LANGUAGE (CIL)
COMMON RUNTIME LANGUAGE (CLR)

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

Introduction to Embedded systems – Shibu K. V

❄❄❄❄❄❄❄