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 bitµp/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.

- DISADVANTAGES OF PLD
  - PLD's required additional development software and hardware which is often very expensive.
  - Design staff often needs to be trained to use new design tools. In addition, parts must be programmed before they can be assembling into a final product.

2.6 Commercial Off The Shelf Components (COTS)

- COTS describes software and hardware product that are ready made and available for sale to the general public.
- For ex-Microsoft office is a COTS product i.e. packaged software solution for businesses.
- COTS products are designed to be implemented easily into existing system without need of customization.
- Re-using components made for earlier product as an approach to new system development is a promising way of achieving the mention development and system improvement.
- There is also possibility to buy software components from component vendors.
- The use of COTS, software component is increasing in today's development of new system.
- Shorter systems life cycle and decreased development budget has made COTS necessary.
- COTS component can also provide an increased reliability compared to customize made components since they are redefined by substantial field testing.
- Although using COTS component can save valuable development time, insight in the COTS component functionality and properties must be evaluated for its intended use.
- In order to integrate COTS component in a system the developer must consider relevant properties of the component such as operational limitations, temporarily behaviour, pre-conditions etc.

2.7 Sensors

- Sensors are also called as detectors.
- It is a device that measures a physical quantity and converts It into a signal which can be read by an observer or by an instrument.
- In other words, we can say that sensors are electronic devices made to sense physical quantities.
- The sensor can be defined as a transducer that converts energy from one type to another type for any particular purpose.
- Example: -
  i. A mercury in a glass thermometer converts the measured temperature into expansion and contraction of a liquid which can be read on calibrated glass tube.
  ii. A thermocouple is a heat sensor that converts temperature to an output voltage which can be read by voltmeter.
- For accuracy, most sensors are calibrated against known standards.
- The sensors are widely used embedded systems so that the system can sense the external physical behaviour and the system can take action accordingly.
- Example: - Optical light, pressure sensor, proximity sensor, gyroscope sensor, ambient-light sensor, etc.

2.8 Actuators

- An actuator is a device that converts energy into motion.
- It can also be used to apply force.
- An actuator is a mechanical device that takes energy, that is created by air, electricity or liquid and convert it into some kind of motion. This motion can be virtually in any form such as blocking, clamping(cutting), ejecting, or many other types of motion.
- Actuators are used in manufacturing or industrial application and might be used in devices such as motors, pumps, switches and valves.
- An actuator can be defined as mechanical device used for moving or controlling a mechanism or a system.

- It is operated by a source of energy usually in form of electric current, hydraulic fluid or pneumatic pressure and converts that energy into some kind of motion.

2.9 Communication Interface :

- Communication Interface is necessary for communicating with various sub-systems of the embedded systems and with external world.
- The communication system can be classified into 2 types:-

  (i) On-board communication interface

  (ii) External communication interface
- Examples of on-board communication interface are :-

  I2C - Inter Integrated Circuit

  SPI – Serial Peripheral Interface

  UART – Universal Asynchronous Receiver Transmitter,

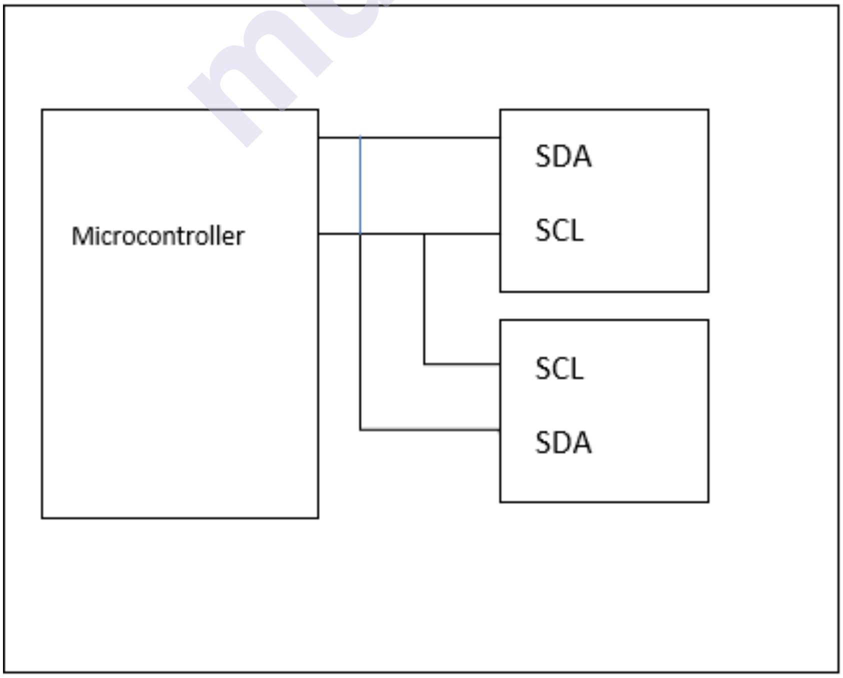
  One Wire Interface, etc.
- Example of external communication interface are :-

  RS232 C,

  USB – Universal Serial Bus,

  Ethernet, etc.

2.9.1 I2C BUS



- I2C BUS is a bi-directional 2-wire serial BUS that provides communication link between integrated circuits.
- It was designed by Phillips in the early 80's to allow communication between components which are on the same circuit board.
- The two bi-directional lines used for communication are Serial Data Line (SDA) and Serial Clockwise (SCL) as shown in the above diagram.
- These are 3 data-transfer speed for I2C BUS

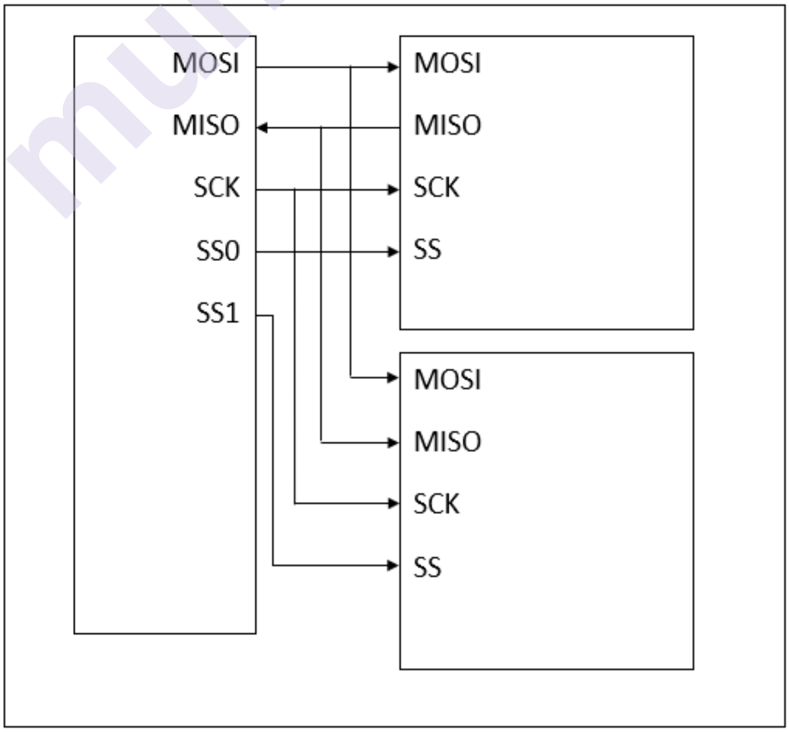
  i.e. Standard (100 kbps)

  Fast mode (400 kbps)

  High Speed (3.4 mbps)
- Features of I2C :-

1. Only 2 BUS lines are required.
2. Supports various data speed rates.
3. It provides a simple master-slave relationship.
4. There is a provision for collision detection.
5. It supports data broadcast.

2.9.2 Serial Peripheral Interface(SPI) BUS

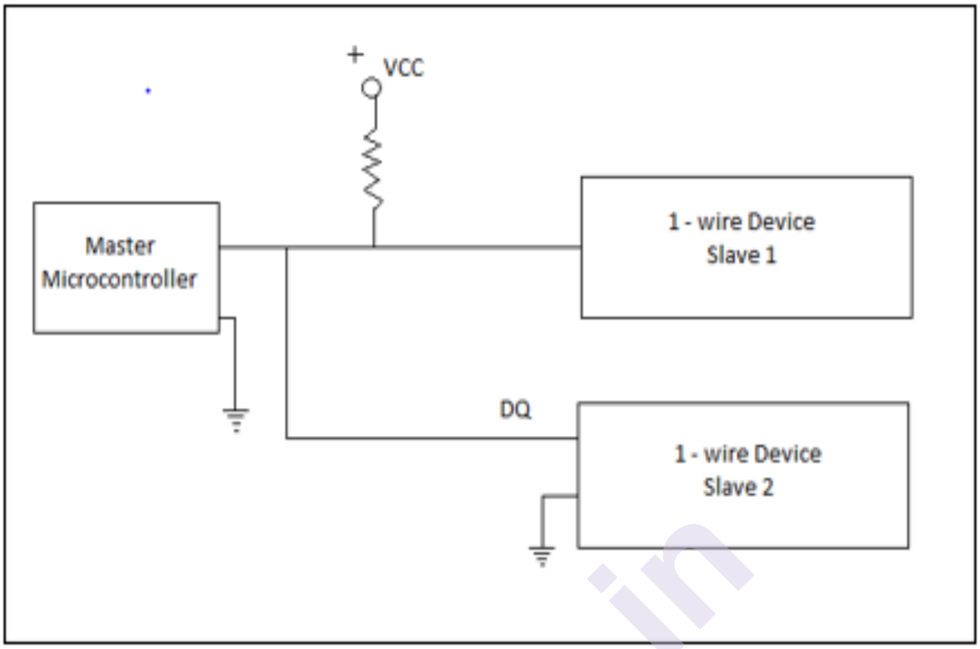


- SPI is a synchronous serial data link which works in full duplex mode.
- It is a serial interface in which 4 wires are required for communication.
- It works on the Master-slave principle where it can have multiple slave.
- The four wires used for communication are as follows :-
  - (i) SCK – Serial Clock Line
  - (ii) MOSI/SDI (Master Output Slave Input / Slave Data Input)
  - (iii) MISO/SDO (Master Input Slave Output/ Slave Data Output)
  - (iv) SS / CS (Chip Select)
- Disadvantages of SPI BUS :-
  1) It requires more number of pins.
  2) During data transfer, there is no acknowledgement from the slave.
  3) There is no provision for error checking.
  4) It supports only 1 Master.
  5) Noise may affect or corrupt the data.

2.9.3 Universal Asynchronous Receiver Transmitter (UART)

- It is a computer hardware that translates data between parallel & serial forms.
- UART is used in conjunction with communication standards such as RS-232, RS-422, etc.
- It converts the bytes received from computer along parallel circuits into a single serial bit stream for outgoing transmissions.
- For incoming transmissions, it converts the serial bit stream into the bytes that computer handles.
- It adds a parity bit on the outgoing transmission & checks the parity of incoming bytes and then discard the parity bit.
- It adds start and stop bits on outgoing signals & separate them from incoming signal.
- It handles interrupt from keyboard and mouse.
- It may handle other kinds of interrupt & device management that requires coordinating the computer speed of operation with device speed.

2.9.4 1-Wire Interface



- 1 – wire interface is an asynchronous half duplex communication protocol.
- It makes use of a single wire or a line called as "DQ" for communications and follows master-slave model.
- This type of interface supports a single master and 1 or more slaves.
- The network design for 1-wire interface can be linear-topology, star topology and stubbed topology.
- The sequence of operation for communicating with a 1-wire slave is as follows:

  (i) The master device sends a reset pulse on 1-wire bus.

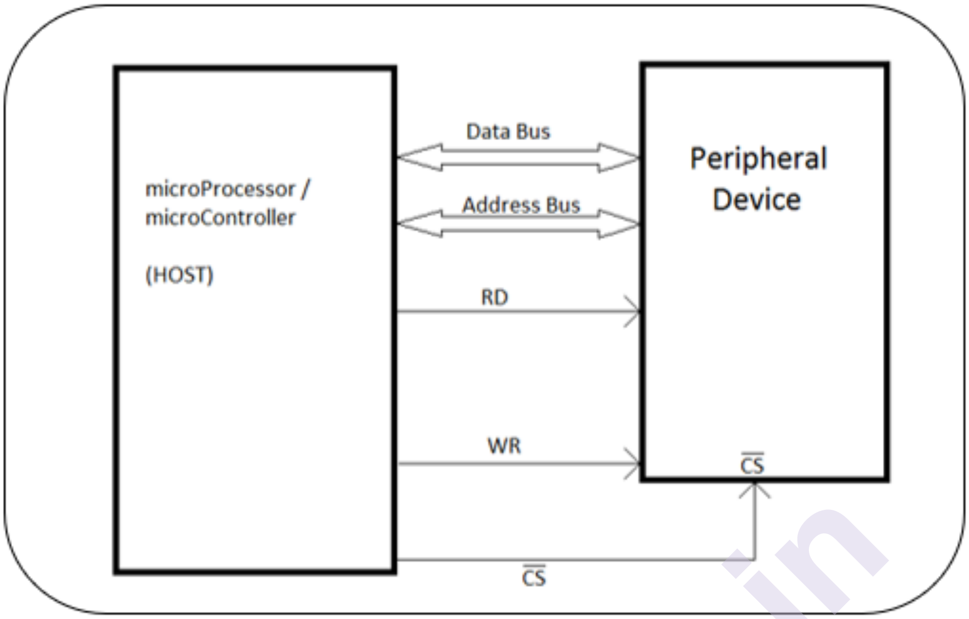
  (ii) The slave device responds with a present pulse.

  (iii) The master device sends a command and 64-bit address of the slave device to which master wants to communicate.

  (iv) The master device sends a read/write function command to read/write the internal memory or register of the slave device.

  (v) After sending read/write function, the master initiates the given operation.

2.9.5 Parallel – Interface:



- In this type of interface, the host microcontroller/microprocessor contains a parallel BUS & the devices which supports such an interface.
- The communication through the parallel BUS is controlled by the control signals such as Read, Write, and Chip Select.
- When the address selected by the processor is within the range, the decoder circuit activates the Chip Select line & the device becomes active.
- While applying proper control signal, the processor can either write into the device or read from the device.
- The width of the BUS can be 4-bit, 8-bit, 16-bit, 32-bit or 64-bit.

2.9.6 RS – 232

- RS-232 was introduced in the year 1962, where the signals were represented by voltage-level with respect to a system.
- It is a single-ended (1-wire) independent channel used for full duplex communication.
- This interface was developed by EIA(Electronic Industry Association).
- As per EIA standards, any voltage between +3V to +12V is logic 0, while any voltage between -3V to -12V is logic 1.
- RS-232 C standard defines many hand-shaking and control signal.
- It supports 2 different types of connectors, i.e. DB 9 & DB 25.

- RS-232 C is a point to point communication interface and devices involved for communication are called as Data Terminal Equipment (DTE) & Data Communication Equipment (DCE).
- This interface gives a speed of about 300 bps (bits per second) to about 20 kbps (kilobits per second).
- The maximum operating distance supported by RS-232 C is about 50 feet.
- Disadvantages :
  1) Suitable for system-to-system communication & NOT for chip to chip or chip to sensor.
  2) Low speed for long distance.
  3) Requires transceiver (transmitter + receiver) chips which add to system cost.
  4) Supports only single master-single slave interface.

2.9.7 USB (Universal Serial Bus) :

- USB is wired high speed serial bus for data communication.
- USB was originally developed in the year 1995 by its core-group members consisting of Microsoft, Intel, IBM, Compaq and Northern Telecom.
- USB communication system follows a Star-topology with the USB Host at the centre and one or more USB peripheral devices connected to it.
- A USB host can support a connection of upto 127 slave devices.
- USB transmits data in packet format where each packet data has a standard format.
- The physical connection between a USB device and a master device is done by using a USB port / USB cable.
- The USB cable supports communication upto a distance of 5 meters.
- There are 2 types of connectors : Type-A and Type-B for USB connections.
- The USB connection present in PCs & laptops are example of type-A connectors, and type-B connectors contains 4 pins for communication.
- Differential signals are used for data transmissions; hence we get good noise immunity.
- USB interface has the ability to supply power to the connecting devices.
- USB devices supports different data rates such as low speed (1.5 mbps), full speed (12 mbps), high speed (480 mbps) and super speed (4.8 Gbps).

2.10 Other Types of Communication Interfaces

2.10.1 Infrared

- Infrared is a serial half-duplex line-of-sight base wireless technology used for data communication between devices.
- For infrared communication, a transceiver must be present in both the devices.
- In this technique, infrared waves of the electromagnetic spectrum are used for transmitting the data.
- It supports point to point and point to multi-point communication, provided all the devices in the communication are within line-of-sight.
- The typical range for communication is from 10cm to 3m, depending upon transmitting power.
- IR supports data rates from 9.6 kbps to 16 mbps.
- Depending upon the speed of transmission, IR is classified into serial IR, medium IR, fast IR, very fast IR, and ultra-fast IR.
- IR communication has an L.E.D. at the transmitter-end and a photodiode at the receiver-end.

2.10.2 Bluetooth

- Bluetooth is a wireless protocol utilizing short-range communication technology felicitating data transmission over short distance from fixed and / or mobile devices creating wireless Personal Area Network (PAN).
- The objective behind development of Bluetooth was the creation of a single-digit wireless protocol capable of connecting multiple devices and overcoming issues arising from synchronization of these devices.
- Bluetooth operates on 2.4 GHz of radio-frequency spectrum, and uses technique called as Frequency Hopping Spread Spectrum (FHSS).
- It provides a data rate of upto 1 mbps and range of approximately 30 feet.
- Bluetooth supports point to point wireless communication.
- The point-to-point communication follows the master-slave relationship.
- When a network is formed in such a way that it contains one master and more than one slave, then it is called as piconet.
- A piconet supports maximum of 7 slave devices.

- Advantages :
  1) Bluetooth devices are wireless.
  2) It is inexpensive.
  3) Bluetooth is automatic.
  4) Inter-operability.
  5) Low interference.
  6) Low energy consumption.
  7) Shares voice and data.
  8) Makes use of instant PAN (Personal Area Network).
  9) Upgradable.

2.10.3 Wi-Fi (Wireless Fidelity)

- Wi-fi is a popular wireless communication technique for devices involved in network communication.
- Wi-fi supports Internet Protocol (IP) , based communication.
- In an IP-based communication, each device is identified by an IP-address which is unique for each device of the network.
- Wi-fi based communication requires Wi-fi router to manage communication.
- This router is responsible for providing access to the network, assigning IP address to the device of the network & routing data packets to the devices.
- Wi-fi operates at a frequency of 2.4 GHz or 5 GHz of the radio-spectrum.
- Wi-fi supports data rates ranging from 1 mbps to 150 mbps and it offers the range of about 300 feet.

2.11 Embedded-Firmware

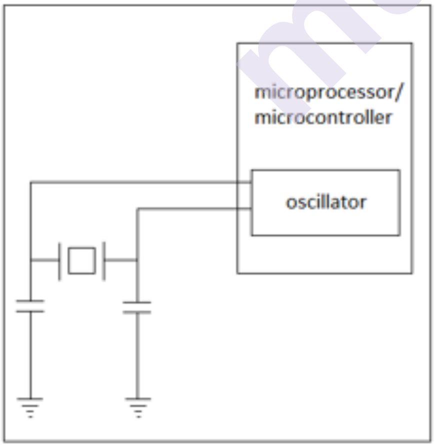
- Embedded firmware refers to control algorithm or the program instruction written by the programmer & dumped (stored) into the code memory of the embedded system.
- There are various methods for developing embedded firmware which are as follows:
  (i) Writing the program in high-level languages like C, C++ & using Integrated Development Environment (IDE).
  (ii) The IDE contains editor, compiler, debugger, etc.
  (iii) Writing the program in assembly language.

- The process of converting the program written in either high-level language or assembly language to machine language is called hex-file creation.
- For writing embedded firmware, higher-level language is preferred over assembly language because of following reasons :-
  - (i) Writing codes in high-level language is easy.
  - (ii) The code written in high-level language is easily portable.
  - (iii) It is easy to debug a high-level language program.
  - (iv) Developing assembly level language program is very tedious & time consuming.

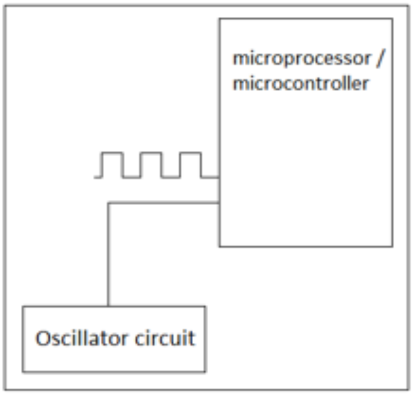
2.12 Other System Components

- While designing an embedded system, we also need supporting circuits or components for proper functioning of embedded system.
- Following are some of the important components:
  1) Oscillator circuit
  2) Reset circuit
  3) Watchdog circuit
  4) Brownout protection
  5) PCB (Printed Circuit Board)

2.12.1 Oscillator circuit



uP/uC having inbuilt oscillator circuit



uP/uC having external oscillator circuit

- Oscillator circuit is used for providing clock frequency to microprocessor or microcontroller so that they can execute program or program instructions.

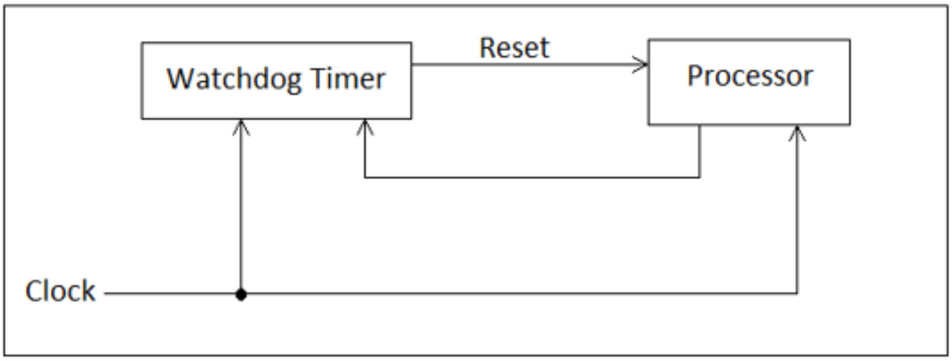
- To generate a clock frequency, a quartz crystal oscillator is connected to microprocessor or microcontroller.
- The instruction execution of a microprocessor/microcontroller occurs in synchronization with a clock signal.
- In embedded system, the oscillator circuit is responsible for generating the exact clock for microprocessor/microcontroller.
- There are certain processors or controllers that have built-in oscillator and simply requires an external resonator or quartz crystal for producing necessary clock signal.

2.12.2 Reset Circuit

- Reset means that the processor starts the processing of the instruction from a starting address.
- The address is one that is set by default in the processor program counter.
- The reset circuit is essential to ensure that the device is not operating at the voltage level where device is not guaranteed to operate when the system is just started.
- The reset signal brings the external register and the different hardware system of the microprocessor/microcontroller to a known state and starts the firmware execution from the reset address.

2.12.3 Watchdog timer

- Most of the embedded systems need to be self-relying. It is not usually possible to wait for someone to reboot them if the system hangs.
- Some embedded designs such as space probes are simply not accessible to human operator.
- If their software ever hangs, such systems are permanently disabled.
- In other cases, the speed with which a human operator might reset the system would be too slow to meet the uptime requirements of the product.
- A watchdog timer is a piece of hardware that can be used to automatically detect software abnormalities and reset the processor if required.
- The following figure shows a typical arrangement of watchdog timer circuit



- In the diagram it is seen that watchdog timer is a chip external to the processor however it could also be included within the same chip as the CPU.

  This is done in many microcontrollers.

  In either cases , the output from watchdog timer is tied/connected directly to the processor's reset signal.

- Kicking the dog is nothing but the process of restarting the watchdog timer counter by software.
- The software must restart the watchdog timer at regular rate otherwise, there is a risk if the system being restarted.
- A watchdog timer is a useful tool in helping the system to recover from failures.

2.12.4 Brownout Protection circuit

- Brownout protection circuit is used to protect microprocessor/microcontroller from unexpected program execution when the input voltage to processor or controller falls down to a specified voltage.
- Mostly all microcontrollers have built in brown-out detection (BOD) circuit which monitors supply voltage level during operation.
- BOD circuit is a comparator which compares supply voltage t a fixed trigger level.
- There are 2 types of Brownout protection circuits : -

  (a) On-chip

  (b) External / off-chip

2.12.5 PCB (Printed Circuit Board)

- PCB is used to mechanically support and electrically connect electronic components using conduction parts, tracks or signal traces etched from copper sheets laminated on a non-conductive surface.
- PCB is the backbone of every embedded system.
- After finalizing the components and inter-connections between them, a schematic diagram is created and according to the diagram, PCB is fabricated.
- PCB acts as a platform for placing all the necessary components as per the design requirement.

2.13 Chapter End Questions

Answer the following questions.

1. Differentiate between RISC and CISC
2. Differentiate between Microprocessor and Microcontroller
3. Explain Big Endian and Little Endian
4. Explain ASIC in details.
5. Explain PLD
6. Explain sensors and actuators.
7. Explain I^2C BUS and SPI BUS
8. Explain the function of watchdog timer in details.
9. Explain Bluetooth and WIFI
10. Explain RS 232 C

2.14 Summary

- From the chapter we have explained in detail working of various communication interface devices.
- Differentiated microprocessor and microcontroller
- Understood various types of processors
- Understood the concept of embedded firmware.

2.15 Reference for further reading

- Introduction to embedded systems by Shibu K V
- Embedded Systems by Rajkamal
- **Bluetooth - Wikipedia**
- Firmware vs Embedded Software - What's the difference? (andplus.com)

❄❄❄❄❄❄❄

Unit 1

3

CHARACTERISTICS & QUALITY ATTRIBUTES OF EMBEDDED SYSTEMS

Unit Structure

3.0 Objectives

After reading this chapter you will:

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

3.1 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.

3.2 Characteristics of Embedded Systems

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.

3.3 Quality Attributes of Embedded Systems

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.

3.3.1 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.

3.3.2 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 shelf 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.

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:

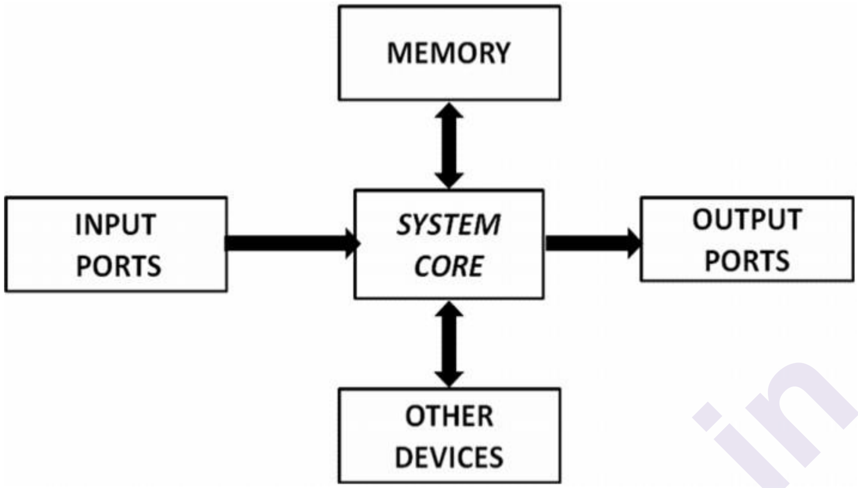


Figure 2.0 : Elements of an Embedded System

- 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)

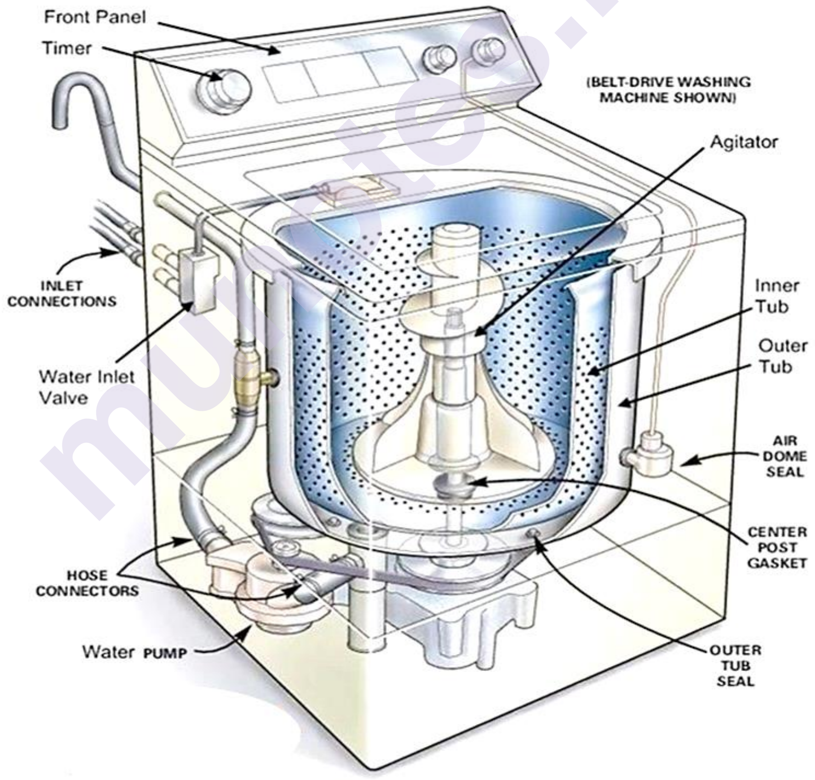
Here are some case studies on some commonly used embedded systems that will help to better understand the concept.

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.

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.

4.3.2 Microwave Owen

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

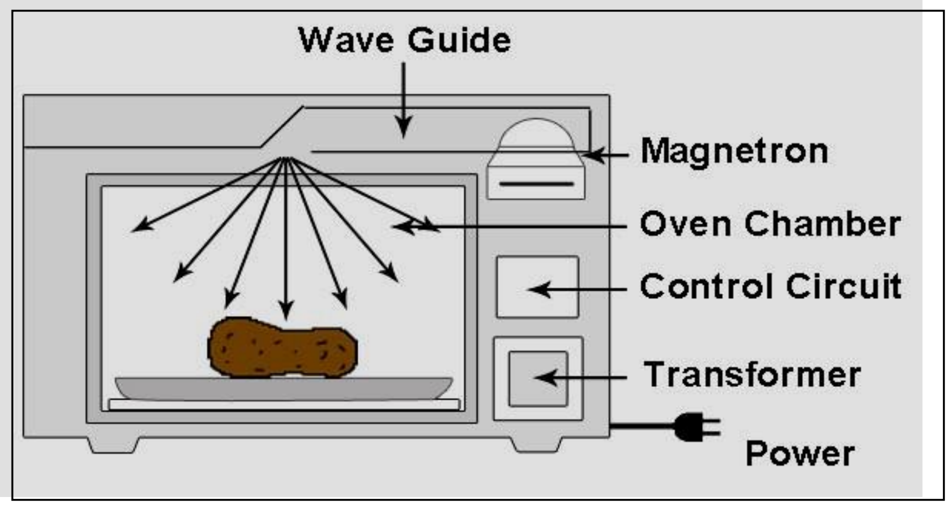


Figure 2.3 : Parts of a Microwave Owen

A microwave oven consists of:

4.3.2.1 A high voltage transformer, which passes energy to the magnetron

4.3.2.2 A cavity magnetron,

4.3.2.3 A Control circuit with a microcontroller,

4.3.2.4 A waveguide, and

4.3.2.5 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.

4.3.3 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 brake 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)

✓ **High-speed Electronic Control Units (HECUs):**

- HECUs are deployed in critical control units requiring fast response.
- They Include fuel injection systems, antilock brake systems, engine control, electronic throttle, steering controls, transmission control and central control units.

✓ **Low Speed Electronic Control Units (LECUs):-**

- They are deployed in applications where response time is not so critical.
- They are built around low cost microprocessors and microcontrollers and digital signal processors.
- 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. Controller Area Network (CAN):-

- 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. Local Interconnect Network (LIN):-

- 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.

4.4 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:
   A. Washing Machine
   B. MICROWAVE Owen
3. Conduct case studies for working of embedded systems for the following topics:
   A. Air Conditioner
   B. Automobile

4.5 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 Oven:** http://globalmicrowave.org/microwaves.php

❄❄❄❄❄❄❄

Unit 2

5

EMBEDDED HARDWARE

Unit Structure

5.0 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

5.1 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.

5.2 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:

5.2.1 Functioning or purpose of the embedded system

5.2.2 Individual components involved

5.2.3 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

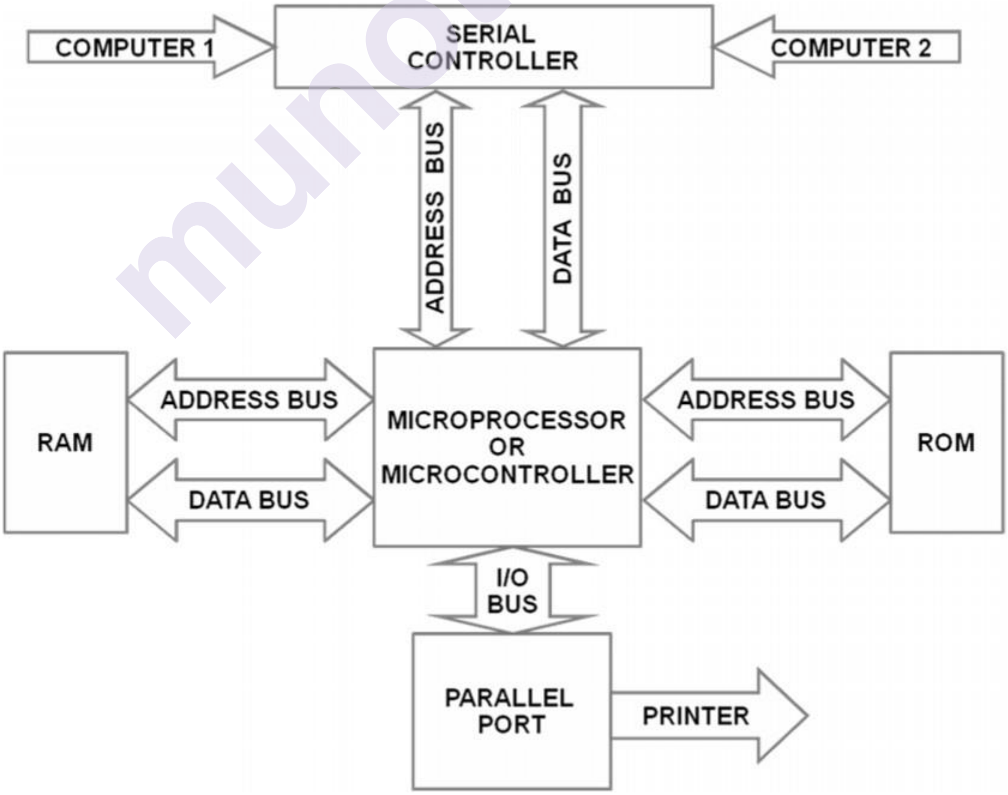


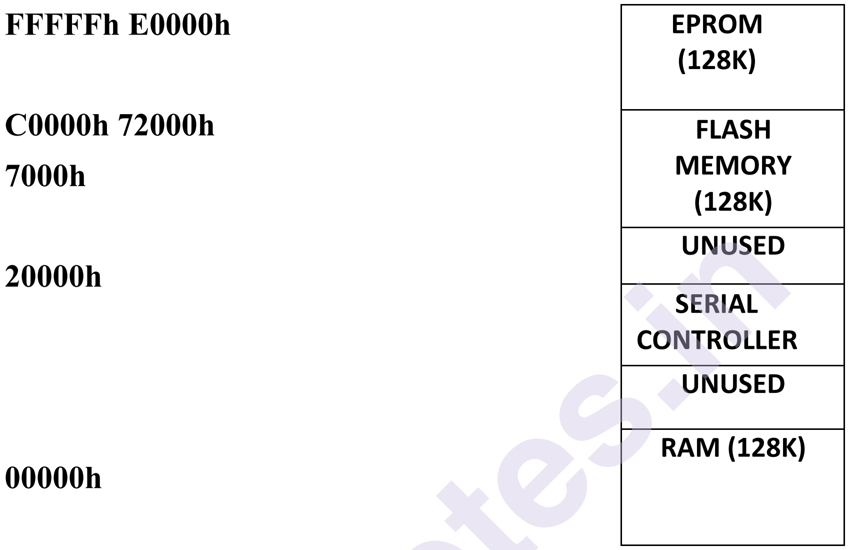
Figure: Components involved in an printer sharing device

- 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.

5.3 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.



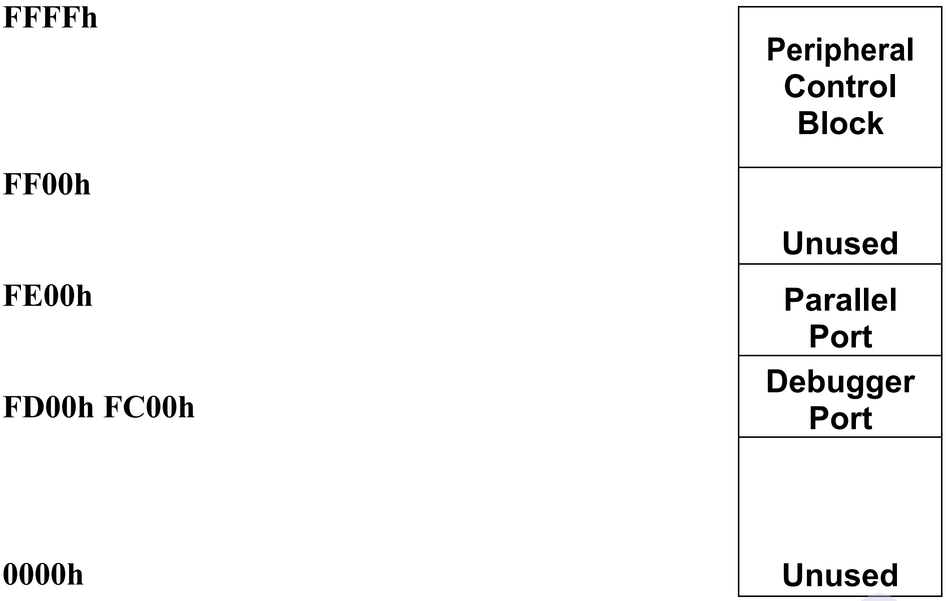
- For every embedded system, a header file should be created

  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

5.4 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

5.5 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:** In interrupt, the device notices the CPU that it requires its attention.
- 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
```

5.6 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

5.7 References & Further Reading

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

❄❄❄❄❄❄❄

Unit 2

6A

EMBEDDED SYSTEMS: MEMORY

Unit Structure

6A.1 Objectives

After reading this chapter you will understand:

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

6A.2 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.

6A.3 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

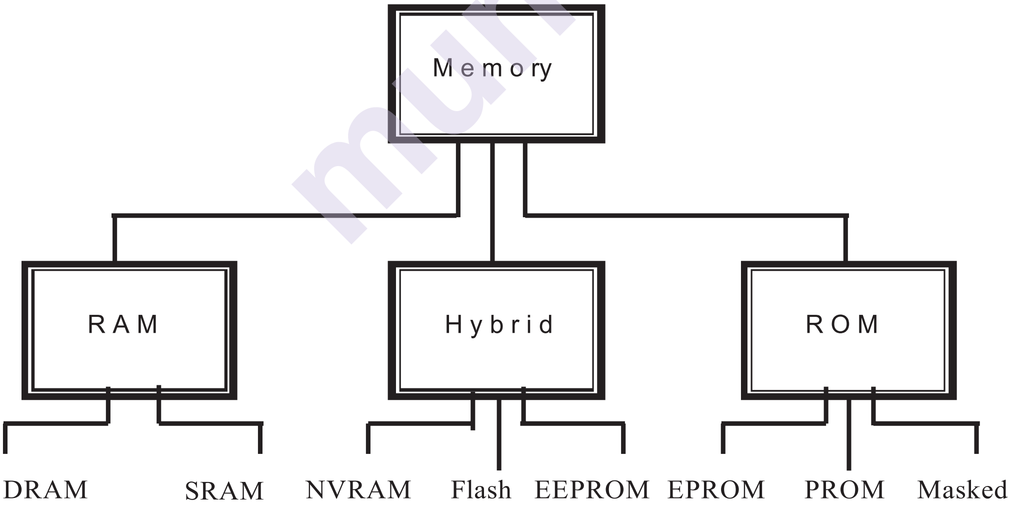


Figure: Types of Memory

6A.4 Types Of RAM

- There are 2 important memory device in the RAM family.

    d) SRAM (Static RAM)

    e) DRAM (Dynamic RAM)

6A.4.1 SRAM (Static RAM)

f) It retains the content as long as the power is applied to the chip.

g) If the power is turned off then its contents will be lost forever.

6A.4.2 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.

6A.5 Types Of Rom

There are three types of ROM described as follows:

6A.5.1 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**.**

6A.5.2 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.

6A.5.3 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.

6A.6 Types of Hybrid Memory

There are three types of Hybrid memory devices:

6A.6.1 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

6A.6.2 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.

6A.6.3 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.

6A.7 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.

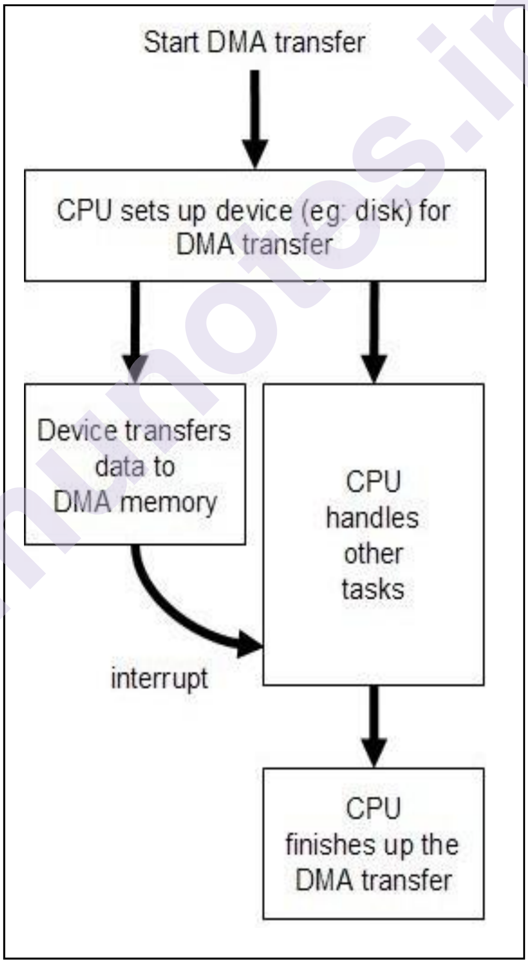

Start DMA transfer
CPU sets up device (eg: disk) for DMA transfer
Device transfers data to DMA memory
CPU handles other tasks
interrupt
CPU finishes up the DMA transfer

Figure: Direct Memory Access

6A.8 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?

6A.9 References & Further Reading

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

❄❄❄❄❄❄❄

-

Unit 2

6B

EMBEDDED SYSTEMS: MEMORY TESTING

Unit Structure

6B.1 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

6B.2 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.

6B.3 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.

6B.4 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

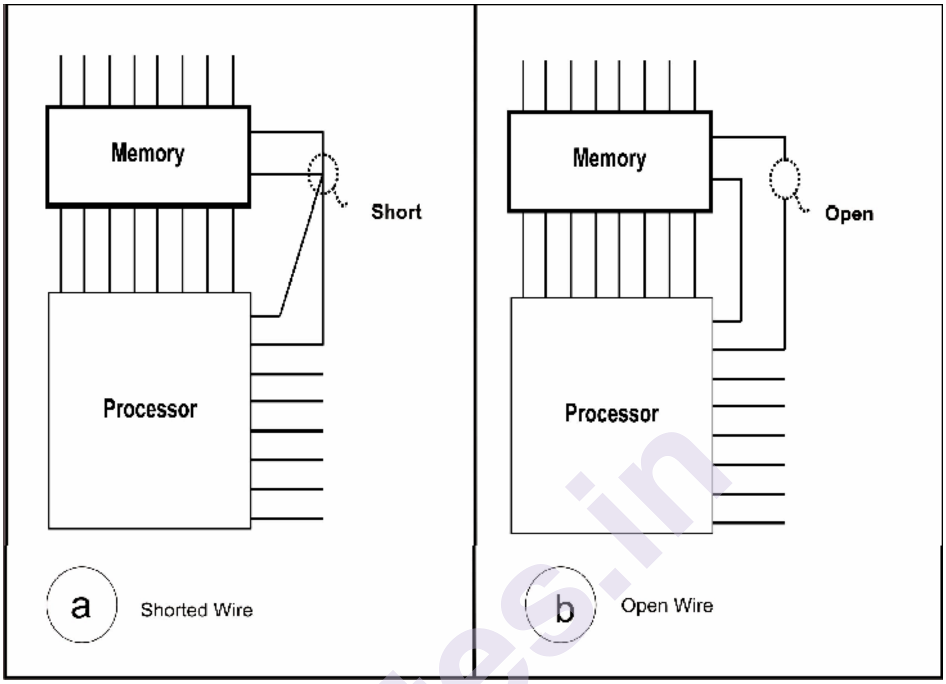


Figure: a. wiring problems: two wires shorted

b. wiring problems: one wire open

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

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

2. **Missing Memory chip.**
    - o A missing memory chip is clearly a problem that should be detected
    - o Unfortunately, because of the capacitive nature of unconnected electrical wires, some memory tests will not detect.
    - o 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.
    - o 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.
    - o 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!
    - o To detect a missing memory chip the previous algorithm for test must be altered.
    - o For example,
        - ✓ write the value 1 to the first location,
        - ✓ 2 to the second location,
        - ✓ And 3 to the third location,
    - o 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
        - o Will either not be connected to the socket at all
        - o Will be connected at the wrong place
    - ▪ Symptoms:-
        - o System behaves same as though there is a wiring problem or a missing chip.
    - ▪ How to detect :-
        - o Detected by any test

6B.5 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.

6B.5.1 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

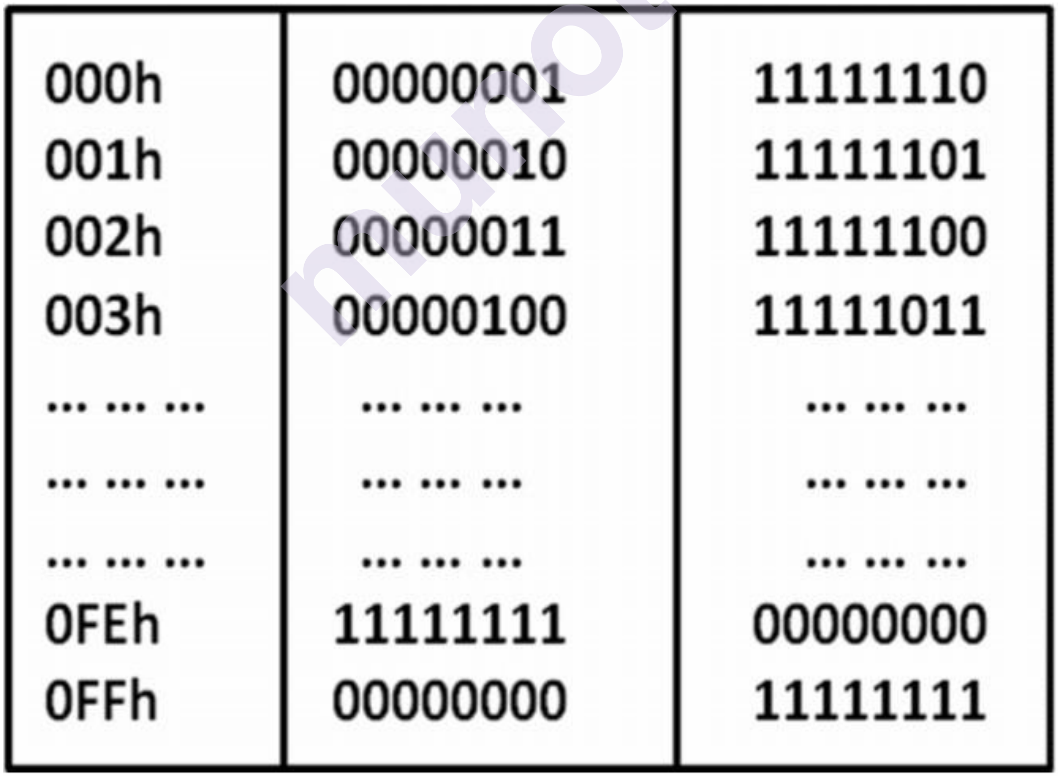
6B.5.2 Address Bus Test

- 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.

6B.5.3 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.

6B.6 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.

6B.7 References & Further Reading

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

❄❄❄❄❄❄❄

Unit 2

7

EMBEDDED SYSTEMS: PERIPHERALS

Unit Structure

7.1 Objectives

After reading this chapter you will learn:

5.7.1 Concept of testing non –volatile memory devices using Checksum and CRC

5.7.2 Control and Status Registers

5.7.3 Device Driver

5.7.4 Watch Dog Timer

7.2 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.

7.3 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

7.4 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.

7.5 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:
  1. Modularization, it makes the structure of the overall software is easier to understand.
  2. There exists only one module that interacts directly with the peripheral's registers making communication easier.
  3. 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.

7.6 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

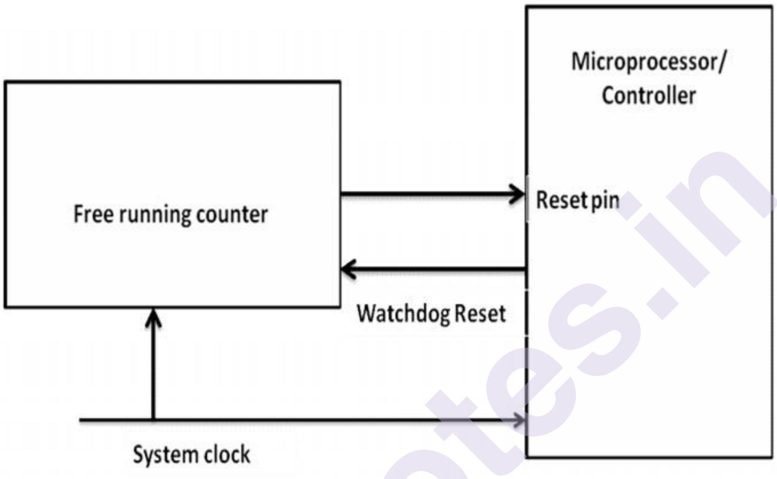


Figure: Watchdog Timer

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.

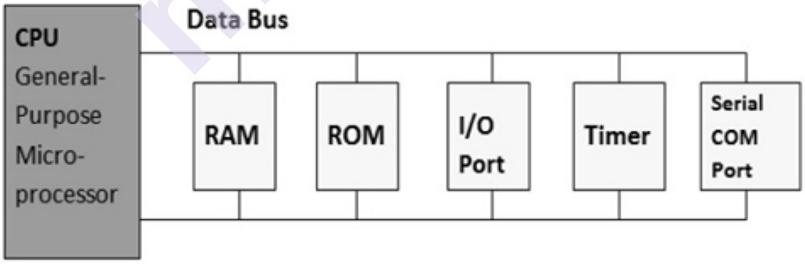
8.3.2 Types of Processors

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

- Microprocessor
- Microcontroller
- Embedded Processor
- Digital Signal Processor
- 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.

CPU	RAM	ROM
I/O Port	Timer	Serial COM Port

Block Diagram of Basic Microcontroller

Microcontrollers are particularly used in embedded systems for real-time control applications with on-chip program memory and devices.

Embedded Processor

Embedded processor is a type of microprocessor designed into a system to control electrical and mechanical functions. Embedded processors are usually simple in design, limited in computational power, I/O capabilities, and have minimal power requirements. At a basic level, embedded processors are a CPU chip placed in a system that it helps control.

Digital Signal Processor

Digital signal processing (**DSP**) is the use of digital processing, such as by computers or more specialized digital signal processors, to perform a wide variety of signal processing operations.

Media processor

A media processor, mostly used as an image/video processor, is a **microprocessor-based system-on-a-chip** which is designed to deal with digital streaming data in real-time (e.g. display refresh) rates.

8.3.3 Microprocessor Vs Microcontroller

Let us now take a look at the most notable difference between a microprocessor and a microcontroller.

Microprocessor	Microcontroller
Microprocessors are multitasking in nature. Can perform **multiple tasks** at a time. For example, on computer we can play music while writing text in text editor.	**Special purpose & Single task** oriented. For example, a washing machine is designed for washing clothes only.

Designer must add RAM, ROM, I/O Ports, and Timers **externally**.	RAM, ROM, I/O Ports, and Timers components embedded together OR **inbuilt** on a chip.
Designers can decide the number of memory or I/O ports needed.	Fixed number for memory or I/O makes a microcontroller ideal for a limited but specific task.
External support of external memory and I/O ports makes a microprocessor-based system heavier and costlier.	Microcontrollers are lightweight and **cheaper** than a microprocessor.
External devices require more space and their power consumption is higher.	A microcontroller-based system consumes **less power** and takes **less space / compact size**.

8.4 Overview of 8051 Family

8.4.1 Brief History Of 8051

In 1981, Intel Corporation introduced an **8-bit microcontroller** called the 8051. This microcontroller had 128 bytes of RAM, 4K bytes of on-chip ROM, two timers, one serial port, and four ports (each 8-bits wide) all on a single chip. At the time, it was also referred to as a “system on a chip.” The 8051 is an 8-bit processor, meaning that the CPU can work on only 8 bits of data at a time. Data larger than 8 bits has to be broken into 8-bit pieces to be processed by the CPU. The 8051 has four I/O ports, each 8 bits wide. See Figure 1-2. Although the 8051 can have a maximum of 64K bytes of on-chip ROM, many manufacturers have put only 4K bytes on the chip. This will be discussed in more detail later.

8.4.2 8051 Members

- **8052 microcontroller-**8052 has all the standard features of the 8051 microcontroller as well as an extra 128 bytes of RAM and an extra timer. It also has 8K bytes of on-chip program ROM instead of 4K bytes.
- **8031 microcontroller**-It is another member of the 8051 family. This chip is often referred to as a ROM-less 8051, since it has 0K byte of on-chip ROM. You must add external ROM to it in order to use it, which contains the program to be fetched and execute. This program can be as large as 64K

bytes. However, in the process of adding external ROM to the 8031, it lost two ports out of 4 ports. To solve this problem, we can add an external I/O to the 8031.

8.4.3 Comparison between 8051 Family Members

The following table compares the features available in 8051, 8052, and 8031.

Feature	8051	8052	8031
ROM(bytes)	4K	8K	0K
RAM(bytes)	128	256	128
Timers	2	3	2
I/O pins	32	32	32
Serial port	1	1	1
Interrupt sources	6	8	6

8.4.4 Features of 8051 Microcontroller

An 8051 microcontroller comes bundled with the following features-

- 8 Bit microcontroller
- 40 Pins DIP (Dual In package)
- 4KB bytes on-chip program memory (ROM)
- 128 bytes on-chip data memory (RAM)
- Four register banks
- 128 user defined software flags
- 8-bit bidirectional data bus
- 16-bit unidirectional address bus
- 32 general purpose registers each of 8-bit
- 16 bit Timers (usually 2, but may have more or less)
- Three internal and two external Interrupts
- Four 8-bit ports, (short model have two 8-bit ports)
- 16-bit program counter and data pointer
- 8051 may also have a number of special features such as UARTs, ADC, Op-amp, etc.

8.5 The 8051 Microcontroller Hardware

8.5.1 Block Diagram of 8051 Microcontroller

The following illustration shows the block diagram of an 8051 microcontroller-

Central Processor Unit (CPU)

As we know that, the CPU is the brain of any processing device of the microcontroller. It monitors and controls all operations that are performed on the Microcontroller units. The user has no control over the work of the CPU directly. It reads program written in ROM memory, executes them, and do the expected task of that application.

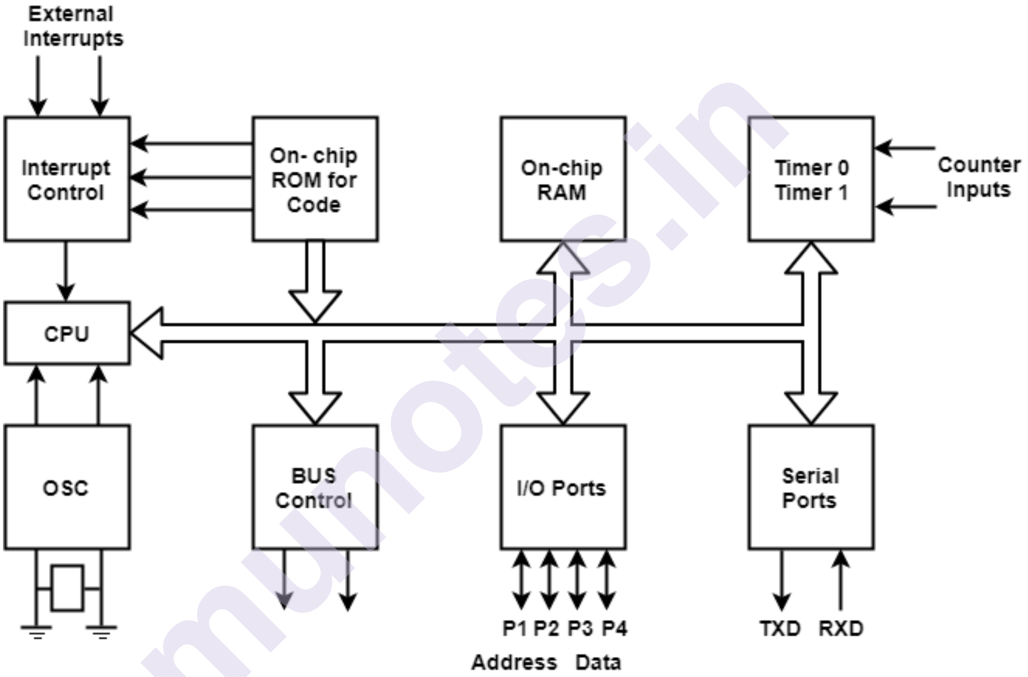


Fig. 8051 Microcontroller Architecture

Interrupts

As its name suggests, Interrupt is a subroutine call that interrupts of the microcontrollers main operations or work and causes it to execute any other program, which is more important at the time of operation. The feature of Interrupt is very useful as it helps in case of emergency operations. An Interrupts gives us a mechanism to put on hold the ongoing operations, execute a subroutine and then again resumes to another type of operations.

The Microcontroller 8051 can be configured in such a way that it temporarily terminates or pause the main program at the occurrence of interrupts. When a subroutine is completed, then the execution of main program starts. Generally five interrupt sources are there in 8051 Microcontroller.

There are 5 vectored interrupts are shown in below

- INTO
- TFO
- INT1
- TF1
- R1/T1

Memory

Microcontroller requires a program, which is a collection of instructions. This program tells microcontroller to do specific tasks. These programs require a memory on which these can be saved and read by Microcontroller to perform specific operations of a particular task. The memory which is used to store the program of the microcontroller is known as code memory or Program memory of applications. It is known, as ROM memory of microcontroller also requires a memory to store data or operands temporarily of the micro controller. The data memory of the 8051 is used to store data temporarily for operation is known RAM memory. 8051 microcontrollers has 4K of code memory or program memory that has 4KB ROM and 128 bytes of data memory of RAM.

Bus

Bus is a collection of wires, which work as a communication channel or medium for transfer of Data. These buses consist of 8, 16 or more wires of the microcontroller. Thus, these can carry 8 bits, 16 bits simultaneously. Hire two types of buses that are as follows:

- Address Bus
- Data Bus

Address Bus: Microcontroller 8051 has a 16-bit address bus for transferring the data. It is used to address memory locations and to transfer the address from CPU to Memory of the microcontroller. It has four addressing modes that are

Data Bus: Microcontroller 8051 has 8 bits of the data bus, which is used to carry data of particular applications.

Oscillator

Generally, we know that the microcontroller is a device; therefore, it requires clock pulses for its operation of microcontroller applications. For this purpose, microcontroller 8051 has an on-chip oscillator that works as a clock source for Central Processing Unit of the microcontroller. The output pulses of oscillator are stable. Therefore, it enables synchronized work of all parts of the 8051 Microcontroller.

Input / Output Port

Normally microcontroller is used in embedded systems to control the operation of machines in the microcontroller. Therefore, to connect it to other machines, devices or peripherals we require I/O interfacing ports in the microcontroller interface. For this purpose microcontroller 8051 has four input, output ports to connect it to the other peripherals.

Timers/Counters

The 8051 microcontroller has two 16-bit timers and counters. These counters are again divided into an 8-bit register. The timers are used for measurement of intervals to determine the pulse width of pulses.

8.5.2 Applications of 8051 Microcontroller

Some of the applications of 8051 is mainly used in daily life & industrial applications some of those applications are shown below

- Light sensing and controlling devices
- Temperature sensing and controlling devices
- Fire detections and safety devices
- Automobile applications
- Defence applications

Some industrial applications of micro controller and its applications

- Industrial instrumentation devices
- Process control devices

Some of 8051 microcontroller devices are used in measurement applications

- Voltmeter applications
- Measuring and revolving objects
- Current meter objects & Hand held metering system

8.6 Input/output Pins, Ports & Circuits

I/O Pins –

Pin number	Name of Pin	Function
1 to 8	Port 1	Internally pulled up, Bidirectional I/O port
9	RESET pin	To Reset Microcontroller to its initial values
10 to 17	Port 3	Interrupts, timer & control, serial communication
18, 19	Crystal Clock	Interfacing external crystal to get system clock
20	Supply	Power supply to the circuit
21 to 28	Port 2	I/O Port
29	PSEN	Program Store Enable to read signal from External memory
30	EA	External Access Input for External Memory interface
31	ALE	Address Latch Enable to DE multiplex Address Data Bus / signal of port
32 to 39	Port 0	I/O port / Lower order address & data bus multiplexed
40	GND	Complete the Circuit (To provide supply)

I/O Ports and Circuits:

Each port of 8051 has bidirectional capability. Port 0 is called 'true bidirectional port' as it floats when configured as input. Port-1, 2, 3 are called 'quasi bidirectional port'. Port-0 Pin Structure Port -0 has 8 pins (P0.0-P0.7).

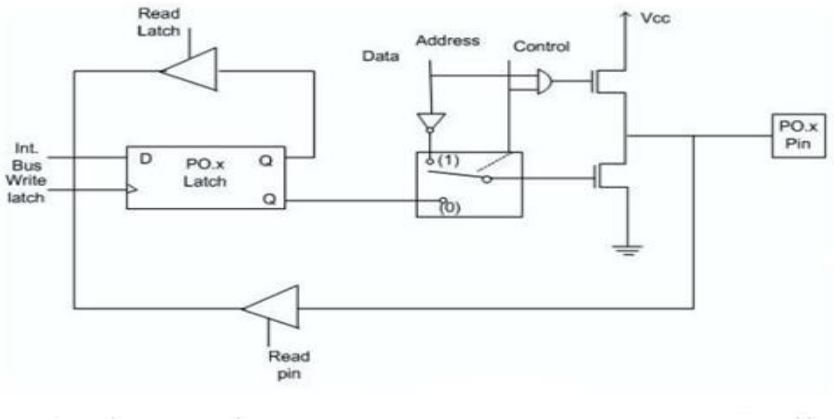


Fig. Port-0 Structure

Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory. When control is '1', the port is used for address/data interfacing. When the control is '0', the port can be used as a normal bidirectional I/O port.

Let us assume that control is '0'. When the port is used as an input port, '1' is written to the latch. In this situation both the output MOSFETs are 'off'. Hence the output pin floats. This high impedance pin can be pulled up or low by an external source. When the port is used as an output port, a '1' written to the latch again turns 'off' both the output MOSFETs and causes the output pin to float. An external pull-up is required to output a '1'. But when '0' is written to the latch, the pin is pulled down by the lower MOSFET. Hence the output becomes zero.

When the control is '1', address/data bus controls the output driver MOSFETs. If the address/data bus (internal) is '0', the upper MOSFET is 'off' and the lower MOSFET is 'on'. The output becomes '0'. If the address/data bus is '1', the upper transistor is 'on' and the lower transistor is 'off'. Hence the output is '1'. Hence for normal address/data interfacing (for external memory access) no pull-up resistors are required.

Port-0 latch is written to with 1's when used for external memory access.

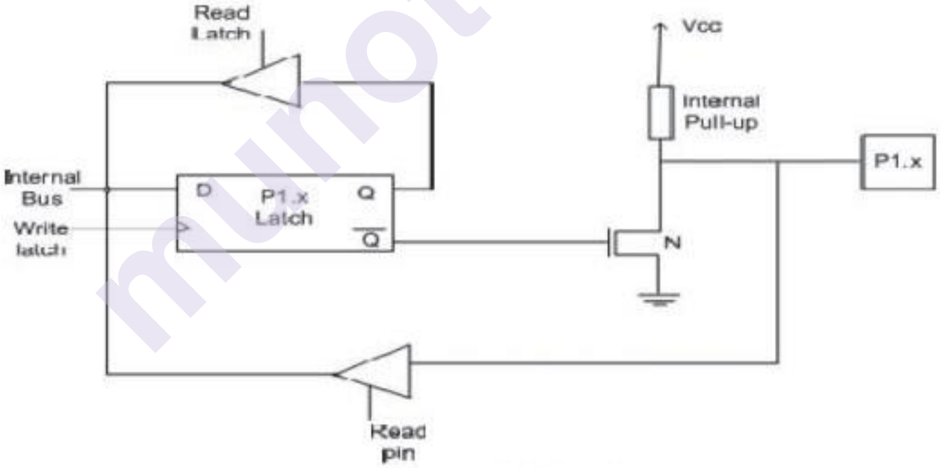


Fig. Port-1 Structure

Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up. To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up. If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.

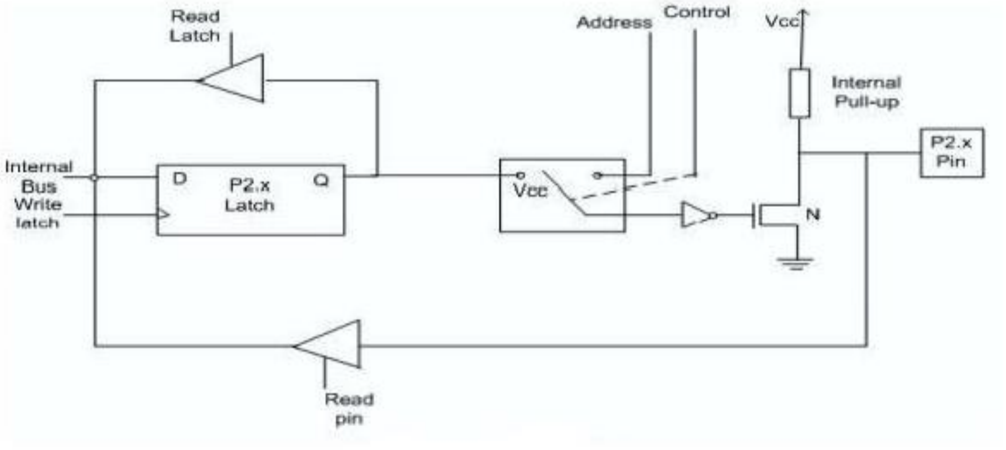


Fig. Port-2 Structure

Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1. Port-2 latch remains stable when Port-2 pin are used for external memory access. Here again due to internal pull-up there is limited current driving capability.

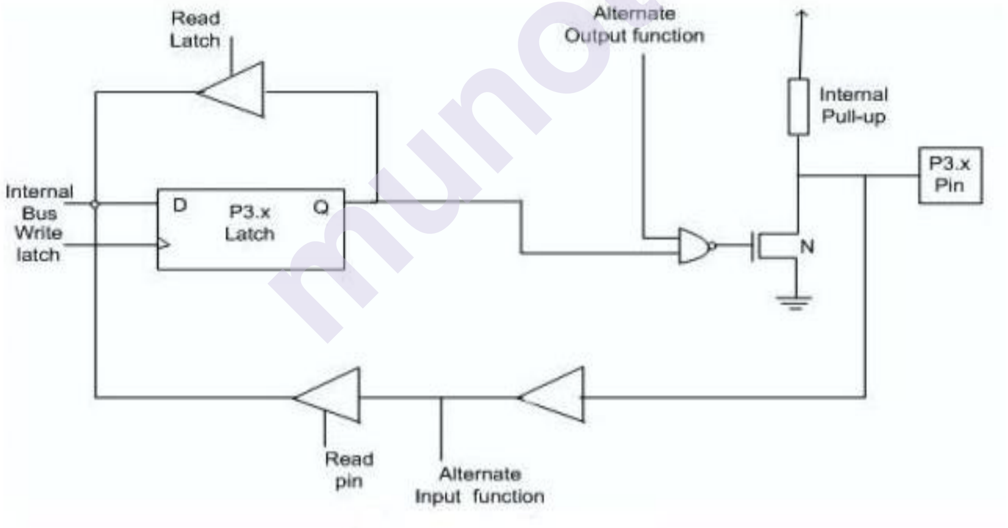


Fig. Port-3 Structure

Each pin of Port-3 can be individually programmed for I/O operation or for alternate function. The alternate function can be activated only if the corresponding latch has been written to '1'. To use the port as input port, '1' should be written to the latch. This port also has internal pull-up and limited current driving capability.

8.7 External Memory

The 8051 Microcontroller based on Harvard Architecture & Instructions / program to be executed are stored in Internal Memory (4 KB ROM). Here Address space available is 0000H to 0FFFH. If we want to connect External memory, address range available from 1000H to FFFFH. (H indicates Hexadecimal address)

Interfacing External Memory:

If external program/data memory are to be interfaced, they are interfaced in the following way.

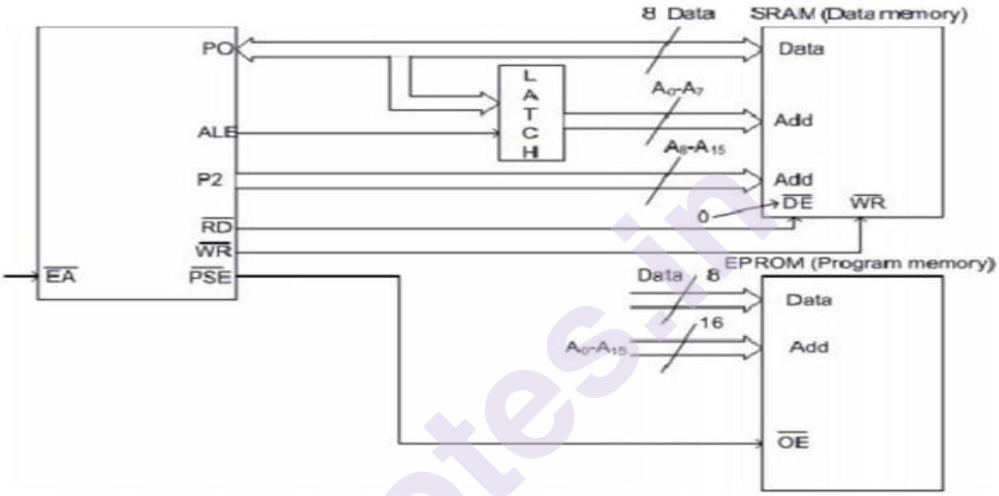


Fig. Interfacing External Memory with 8051

External program memory is fetched if either of the following two conditions are satisfied.

1. (Enable Address) is low. The microcontroller by default starts searching for program from external program memory.
2. PC is higher than FFFH for 8051.

PSEN tells the outside world whether the external memory fetched is program memory or data memory.

8.8 Summary

Embedded system as a microcontroller-based, software-driven, reliable, real-time control system, designed to perform a specific task. In Microcontroller inbuilt available RAM, ROM, I/O Ports and Timers etc. In many Applications Space, Power, cost are much consideration so we prefer Microcontroller.

The **8051** Microcontroller is 8 Bit, 40 Pins DIP. It is most popular so used in daily life & industrial applications. In family of 8051 Microcontroller 2 more Microcontroller compatible as 8052 OR 8031 depends on Application as their features are different. To Interface 2 or more devices for communication as Input/output Pins, Ports, Circuits, External Memory can add as Hardware of Embedded that depends on Applications.

8.9 Review Questions

1) Give Overview of 8051 family
2) What are the requirements of 8051 Microcontroller Hardware?
3) Explain Microcontroller & Embedded processors.
4) Draw Block diagram of 8051 with important I/O Pins
5) How we interface External Memory to 8051
6) Give Ports & circuits of 8051.
7) Differentiate Microprocessor & Microcontroller
8) Give Applications of 8051

8.10 References & Further reading

1) Book *Embedded Systems Architecture, Programming & Design* (Second Edition / The McGraw Hill publications) by **Raj kamal**.
2) Introduction to *Embedded system* By **Shibu K V** (Tata McGraw Hill Publications)
3) Book *Embedded System* by Ashwini Somnathe & Abhijit Somnathe for SYBSc IT (Sheth Publications)

❄❄❄❄❄❄❄

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:

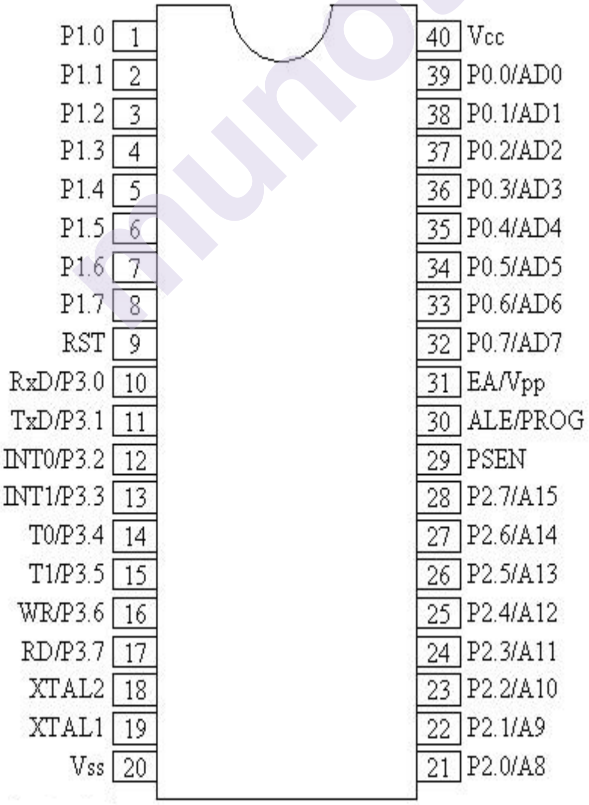
The delay length in 8051 microcontroller depends on three factors:
The crystal frequency
The number of clock per machine
The C compiler.
Code generating delay using timer register:

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

```
while (1)
{
P1=0x55;
T0Delay();
P1=0xAA;
T0Delay();
}
}
void T0Delay(){
TMOD=0x01;     // timer 0, mode 1
TL0=0x00;      // load TL0
TH0=0x35;      // load TH0
TR0=1;         // turn on Timer0
while (TF0==0);          // wait for TF0 to roll over
TR0=0;      // turn off timer
TF0=0;      // clear TF0
}
```

9.5 I/O Programming

9.5.1 Pin Diagram



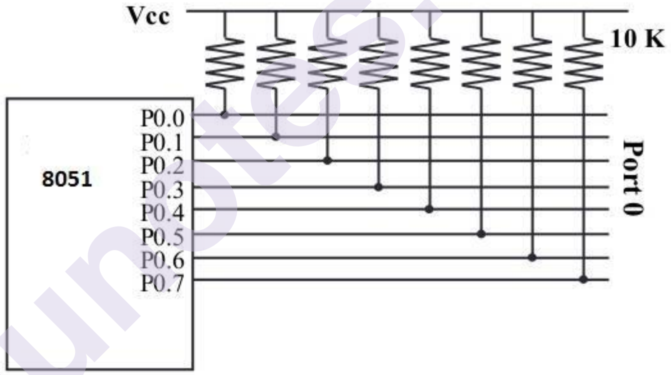
9.5.2 I/O Ports and Their Functions

The four ports P0, P1, P2, and P3, each use 8 pins, making them 8-bit ports. Upon RESET, all the ports are configured as inputs, ready to be used as input ports. When the first 0 is written to a port, it becomes an output. To reconfigure it as an input, a 1 must be sent to a port.

Port 0 (Pin No 32 – Pin No 39)

It has 8 pins (32 to 39). It can be used for input or output. Unlike P1, P2, and P3 ports, we normally connect P0 to 10K-ohm pull-up resistors to use it as an input or output port being an open drain.

It is also designated as AD0-AD7, allowing it to be used as both address and data. In case of 8031 (i.e. ROM less Chip), when we need to access the external ROM, then P0 will be used for both Address and Data Bus. ALE (Pin no 31) indicates if P0 has address or data. When ALE = 0, it provides data D0-D7, but when ALE = 1, it has address A0-A7. In case no external memory connection is available, P0 must be connected externally to a 10K-ohm pull-up resistor.



```
MOV A,#0FFH  ;(comments: A=FFH(Hexadecimal  i.e. A=1111 1111)
MOV P0,A     ;(Port0 have 1's on every pin so that it works as Input)
```

Port 1 (Pin 1 through 8)

It is an 8-bit port (pin 1 through 8) and can be used as either input or output. It does not require pull-up resistors because they are already connected internally. Upon reset, Port 1 is configured as an input port. The following code can be used to send alternating values of 55H and AAH to Port 1.

```
; Toggle all bits of continuously
MOV     A,#55
BACK:
MOV     P2,A
ACALL   DELAY
```

```
CPL     A       ; complement (invert) reg. A
SJMP    BACK
```

If Port 1 is configured to be used as an output port, then to use it as an input port again, program it by writing 1 to all of its bits as in the following code.

```
; Toggle all bits of continuously
MOV     A ,#0FFH    ;A = FF hex
MOV     P1,A        ;Make P1 an input port
MOV     A,P1        ;get data from P1
MOV     R7,A        ;save it in Reg R7
ACALL   DELAY       ;wait
MOV     A,P1        ;get another data from P1
MOV     R6,A        ;save it in R6
ACALL   DELAY       ;wait
MOV     A,P1        ;get another data from P1
MOV     R5,A        ;save it in R5
```

Port 2 (Pins 21 through 28)

Port 2 occupies 8 pins (pins 21 through 28) and can be used for both input and output operations. Just as P1 (Port 1), P2 also does not require external Pull-up resistors because they are already connected internally. It must be used along with P0 to provide the 16-bit address for the external memory. So it is also designated as (A0–A7), as shown in the pin diagram. When the 8051 is connected to an external memory, it provides path for upper 8-bits of 16-bits address, and it cannot be used as I/O. Upon reset, Port 2 is configured as an input port. The following code can be used to send alternating values of 55H and AAH to port 2.

```
; Toggle all bits of continuously
MOV     A,#55
BACK:
MOV     P2,A
ACALL   DELAY
CPL     A         ; complement (invert) reg. A
SJMP    BACK
```

If Port 2 is configured to be used as an output port, then to use it as an input port again, program it by writing 1 to all of its bits as in the following code.

```
; Get a byte from P2 and send it to P1
MOV    A, #0FFH    ;A = FF hex
MOV    P2,A       ;make P2 an input port
BACK:
MOV    A,P2       ;get data from P2
MOV    P1,A       ;send it to Port 1
SJMP   BACK       ; keep doing that
```

Port 3 (Pins 10 through 17)

It is also of 8 bits and can be used as Input/Output. This port provides some extremely important signals. P3.0 and P3.1 are RxD (Receiver) and TxD (Transmitter) respectively and are collectively used for Serial Communication. P3.2 and P3.3 pins are used for external interrupts. P3.4 and P3.5 are used for timers T0 and T1 respectively. P3.6 and P3.7 are Write (WR) and Read (RD) pins. These are active low pins, means they will be active when 0 is given to them and these are used to provide Read and Write operations to External ROM in 8031-based systems.

Dual Role of Port 0 and Port 2

- **Dual role of Port 0** − Port 0 is also designated as AD0–AD7, as it can be used for both data and address handling. While connecting an 8051 to external memory, Port 0 can provide both address and data. The 8051 microcontrollers then multiplexes the input as address or data in order to save pins.

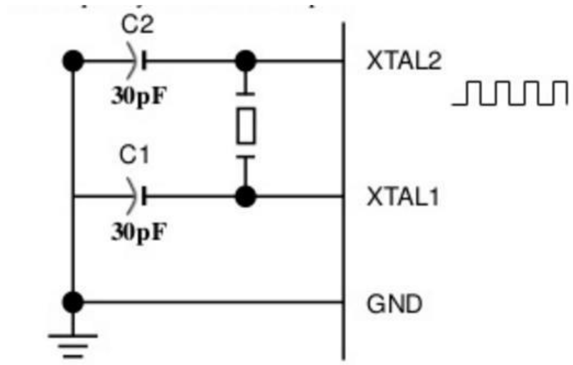
- **Dual role of Port 2** - Besides working as I/O, Port P2 is also used to provide 16-bit address bus for external memory along with Port 0. Port P2 is also designated as (A8– A15), while Port 0 provides the lower 8-bits via A0–A7. In other words, we can say that when an 8051 is connected to an external memory (ROM) which can be maximum up to 64KB and this is possible by 16-bit address bus because we know 216 = 64KB. Port2 is used for the upper 8-bit of the 16 bits address, and it cannot be used for I/O and this is the way any Program code of external ROM is addressed.

9.5.3 Hardware Connection of Pins

- **Vcc** − Pin 40 provides supply to the Chip and it is +5 V.

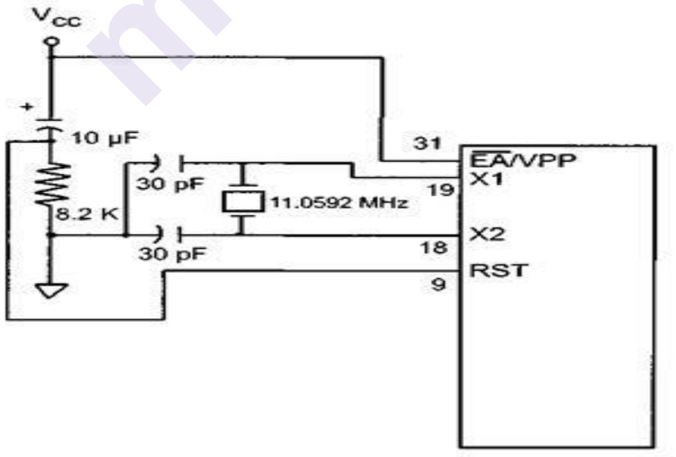
- **Gnd** − Pin 20 provides ground for the Reference.

- **XTAL1, XTAL2 (Pin no 18 & Pin no 19)** − 8051 has on-chip oscillator but requires external clock to run it. A quartz crystal is connected between the XTAL1 & XTAL2 pin of the chip. This crystal also needs two capacitors of 30pF for generating a signal of desired frequency. One side of each capacitor is connected to ground. 8051 IC is available in various speeds, and it all depends on this Quartz crystal, for example, a 20 MHz microcontroller requires a crystal with a frequency no more than 20 MHz.



- **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)

9.6 Logic Operations

9.6.1 Basics of Bitwise Operations

Now let us concentrate only on bitwise operations. We'll learn how these bitwise operations allows us for Setting, Inverting, Toggling, Clearing, Extracting and Inserting bits in embedded programming. Here is a table which summarizes operations with 2-operands.

A	B	A | B	A & B	A ^ B
0	0	0	0	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	0

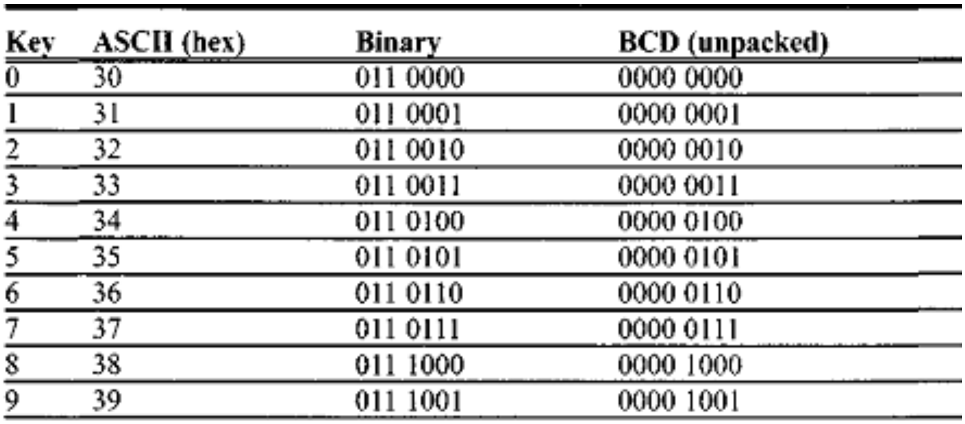
9.7 Data Conversion Programs

Many newer microcontrollers have a real-time clock (RTC) where the time and date are kept even when the power is off. Very often, the RTC provides the time and date in packed BCD. However, to display them they must be converted to ASCII. In this section we show the application of logic and rotate instructions in the conversion of BCD and ASCII

ASCII numbers

On ASCII keyboards, when the key "0″ is activated, "Oil 0000″ (30H) is provided to the computer. Similarly, 31H (Oil 0001) is provided for the key "1″, and so on, as shown in Table

ASCII Code for Digits 0 – 9



Key	ASCII (hex)	Binary	BCD (unpacked)
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001
2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	011 0110	0000 0110
7	37	011 0111	0000 0111
8	38	011 1000	0000 1000
9	39	011 1001	0000 1001

Packed BCD to ASCII conversion

The RTC provides the time of day (hour, minute, second) and the date (year, month, day) continuously, regardless of whether the power is on or off. However, this data is provided in packed BCD. To convert packed BCD to ASCII, it must first be converted to unpacked BCD. Then the unpacked BCD is tagged with Oil 0000 (30H). The following demonstrates converting from packed BCD to ASCII.

ASCII to packed BCD conversion

To convert ASCII to packed BCD, it is first converted to unpacked BCD (to get rid of the 3), and then combined to make packed BCD. For example, 4 and 7 on the keyboard give 34H and 37H, respectively. The goal is to produce 47H or “0100 0111″, which is packed BCD.

9.8 Summary

In 8051, I/O operations are done using four ports and 40 pins. I/O interface is required whenever a processor drives the I/O device. The interface must have the necessary logic to interpret the device address generated by the processor.

C language support to in line assembly gives many benefits. The C Program uses various instructions elements, Pre-processor Directives, Functions, Macro & constants, Including of the source & Header files. Basic C programming elements are Data Types, Data Structure, Modifiers, conditional statements & Loops, Function calls, multiple functions; Queue & Routines. Structure is collection of all Data types. Array of structure are helpful in configure data in Embedded Applications. Compiler is tool for native platform Application Development. Whereas Cross-Compiler is a tool for cross platform Application development.

Embedded C can be considered as subset of conventional C Language. Embedded C supports almost all C instructions & incorporates a few target processor specific functions. Embedded C also supports all Keywords, Identifiers & Data types, Storage classes, Arithmetic & Logic operations, Array & Branching instructions.

Infinite looping is greatly used features in Embedded systems. The C function arguments pass the variable values as well as pass reference to the functions, Pointers, Null & Function Pointers. A data type specifies the type of data that a variable can store such as integer, floating, character etc. When the delay subroutine is executed, the microprocessor does not execute other tasks. Logical instructions are the instructions which perform basic **logical operations** such as AND, OR, etc. The bitwise or bit-level operator lay’s foundation for bitwise operations in embedded programming. 3 factors that affect accuracy of Time Delay as of Crystal Frequency, Machine Cycle & Compiler used in 8051.

9.9 Questions

Q1) Give Basics of Embedded C Program

Q2) Differentiate C & Embedded C language

Q3) Give Advantages of Embedded C

Q4) Explain Keywords used in Embedded C?

Q5) What do you mean by I/O Programming?

Q6) Write a C Program to Blink LED connected to Port 1 of 8051.

Q7) Give the Importance of Logic Operations & Their Use

Q8) Explain Data Types & Time Delay uses in 8051?

9.10 References & Future Reading

1) Book Programming in ANSI C by E Balgurusamy.

2) Book *Embedded Systems Architecture, Programming & Design* (Second Edition / The McGraw Hill publications) by Raj kamal.

3) Introduction to *Embedded system* By Shibu K V (Tata McGraw Hill Publications)

4) Book *Embedded System* by Ashwini Somnathe & Abhijit Somnathe for SYBSc IT (Sheth Publications)

5) www.google.com

6) www.youtube.com

❄❄❄❄❄❄❄

Unit 4

10

DESIGNING EMBEDDED SYSTEM WITH 8051 MICROCONTROLLER

Unit Structure

10.0 Objectives

- In this chapter one will understand what are the factors that need to be considered while selecting a microcontroller.
- Understanding the importance of 8051 microcontroller
- Understanding how to design 8051

10.1 Factors To Be Considered In Selecting A Controller:

1. Speed of Operation :

- It is one of the most important factor that needs to be considered while selecting microcontroller.
- The number of clocks required per instruction cycle and maximum operating clock frequency supported by processor greatly affects the speed of operation of the controller.

2. **Code Memory Space:**
   - The target processor/controller application is written in C or any other high level language.
   - Does the controller support sufficient code memory space to hold the compiled hex code?
3. **Data Memory Space:**
   - Does the controller support sufficient internal data memory(ON chip RAM) to hold run time variables and data structures?
4. **Development Support**
   - In this it is essential to check that the controller manufacture provides cost effective development tools.
   - It also needs to be checked that manufacturer provide sample product for prototyping and sample development stuffs to alleviate the development pains.
   - Does the controller support third party development tools?
   - Does the manufacturer provide technical support if necessary?
5. **Availability:**
   - This is another important factor that should be considered for selecting the process.
   - Since the product is entirely dependent on the controller, the product development time and time to market the product solely depends on its availability.
   - Technically it is referred as Lead Time.
   - Lead time is the time elapsed between the purchase order approval and the supply of the product.
6. **Power Consumption:**
   - Power consumption of the controller should be minimum.
   - It is critical factor since high power requirement leads to bulky power supply design.
   - The high power dissipation also demands for cooling fans and it will make the overall system messy and expensive.
   - Controller should support idle and power down modes of operation to reduce power consumption.
7. **Cost:**
   - It is the biggest deciding factor in selecting a controller.
   - The cost should be within the reachable limit of the end user and the targeted user should not be high tech.

10.2 Why 8051 Microcontroller:

- It is very versatile microcontroller featuring powerful Boolean processor which supports bit manipulation instruction for Real time industrial control applications.
- 8051 supports 6 interrupts (2 external interrupts , 2 timer interrupts and 2 serial interrupts), two 16 bit timers/counters, 32 I/O lines and programmable full duplex serial interface.
- The interrupts of 8051 has two priority levels and each interrupt is allocated fixed 8 bytes of code memory, this approach is very efficient in real time application.
- Apart from Intel which is the original manufacturer of 8051 it is also available through more than 20 vendors with more than 100 variants of 8051.
- It supports CAN, USB, SPI and TCP/IP interfaces, it also has integrated ADC/DAC
- Apart from all the above features the most striking feature is its cost, which is very low.

10.3 Architecture diagram of 8051:

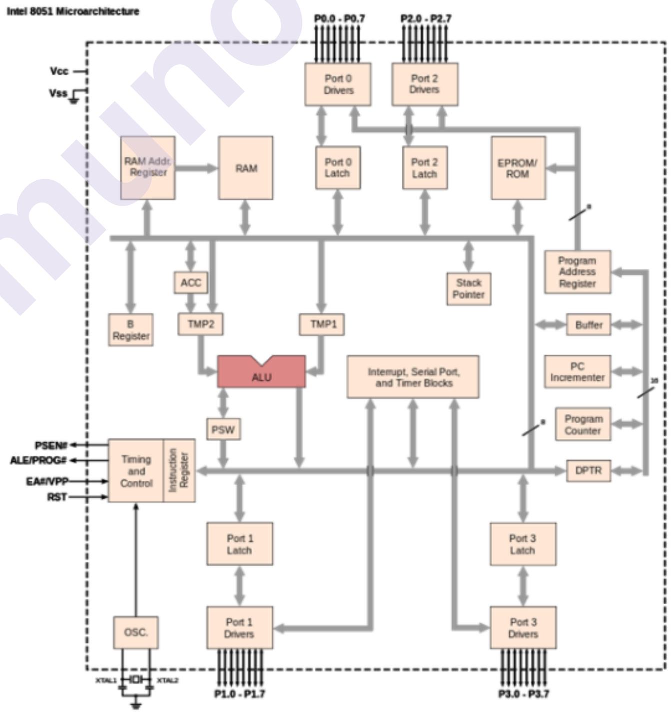
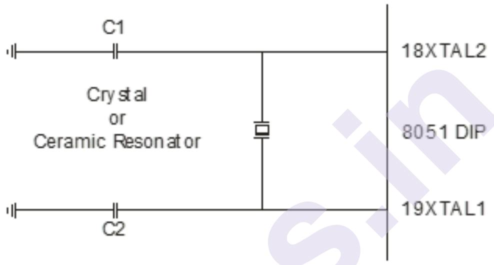


Figure 10.1 Architecture of 8051 Microcontroller Functional Description of Each Block

- Accumulator: (Acc) Acc is an 8 bit special function register. It acts an operand register. Result is temporarily stored in this register. It is used in parallel I/O transfer.
- B Register : B register is 8 bit SFR. It is used during multiply and divide operations. For other operations , it can be used as a scratchpad register.
- Program Status Word (PSW) : PSW register is 8 bit SFR. It contains program status information. It is also used to select any one of the required register bank.
- Stack Pointer (SP): It is 8 bit register. It is used to point the stack memory. The stack may reside in anywhere in on-chip memory. It is incremented before data is stored during PUSH & CALL instructions. After reset SP is initialized to 07h. This causes the stack begin at location 08h.
- Data Pointer (DPTR) : It is 16 bit register. It may be manipulated as a 16 bit register or as two independent 8 bit registers. Its function is to hold a 16 bit address. This register is used for external reference.
- Port 0 to Port 3 : Each port contains separate address. Using this address, User can communicate with these ports. Each port contains latch,output driver & input buffer.
- Serial Data Buffer : Serial data buffer contains two independent registers of a transmit buffer register and a receiver buffer register.
    - Transmit buffer is a parallel in and Serial out register.
    - Receiver buffer is a Serial in and parallel out register.
    - When data is moved to SBUF, it goes to transmit buffer .
    - When data is moved from SBUF, it comes from the receive buffer.
- Timer Registers : Register pairs (TH0,TL0) & (TH1,TL1) are the two 16 bit counting registers for Timer/Counter 0 and 1 respectively.
- Control Registers : The special function registers IP,IE , TMOD, TCON SCON and PCON contain control and status information for interrupts, timer/counters and serial port.
- Timing & Control Unit : This unit derives an necessary timing and control signals required for the internal operations of the circuit. It derives control signals required for controlling the external system bus. The interrupt, serial port and timer circuits are controlled by the control signals generated by timing & control unit.
- Oscillator : This circuit generates the basic timing clock signal for the operation of the circuit using crystal oscillator.
- Instruction Register : This register decodes the opcode of an instruction to be executed and gives information to the timing & control unit, and to generate necessary signals for the execution of the instruction.

- EPROM & Program Address Register : These blocks provide an onchip EPROM and a Mechanism to internally address it.
- RAM & RAM address register : These blocks provide an onchip RAM and a mechanism to internally address it.
- ALU : The Arithmetic And Logic Unit performs 8 bit arithmetic and logical operations over the operands held by the temporary registers TMP 1 and TMP 2. User cannot access these temporary registers.
- SFR Register Banks : It is a set of registers, which can be addressed using their respective addresses which lie in the range 80h to FFh.

10.4a. The 8051 oscillator and Clock :



Crystal or Ceramic Resonator Oscillator Circuit

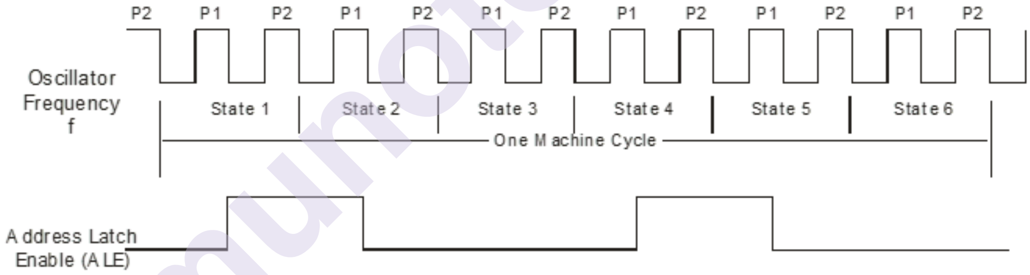


Figure 10.2:Oscillator and Clock Signal

- 8051 chips are designed to run at specific minimum and maximum frequencies. (Typically 1 MHz to 16 MHz).
- Minimum frequency implies that some internal memories are dynamic and must always operate above a minimum frequency or data will be lost.
- Clock frequency is the smallest interval of time within the microcontroller called the pulse, P, time.
- State is the basic time for discrete operations of a microcontroller such as opcode fetching, opcode decoding, opcode execution, or writing a data byte. Two oscillator pulses form each state.
- Machine cycle is the smallest interval of time required to execute a simple instruction, or part of a complex instruction. The machine cycle is made up

of 6 states or 12 clock pulses. Program instructions may require one, two or four machine cycles.

- Therefore the formula to calculate time taken for each instruction is given by :

$$T_{inst} = \frac{Machine\ cycle\ * 12}{crystal\ frequency}$$

e.g. ADD A,R1 : 1 Machine Cycle,. Consider Crystal frequency: 12 MHz

$$T_{inst} = \frac{1\ * 12}{12\ MHz} = 1\mu sec$$

10.4.b. Why is 11.0592 MHz crystal considered over 12 MHz ?

- For serial communication in 8051, it is required that the internal counters must divide the basic clock rate to achieve standard communication bits per second (baud) rates.
- The standard baud rates are 19200, 9600, 4800, 2400, 1200, and 300 Hz.
- If the basic clock frequency is not divisible without a remainder, then the resulting communication frequency is not standard.
- e.g. if crystal frequency is 11.0592 MHz then;
- $T_{inst} = \frac{1 * 12}{11.0592\ MHz} = 1.085\mu sec = 921,600Hz$
- It is noted that 921,600Hz can be divided evenly by standard communication baud rates, Hence 11.0592 MHz crystal is considered.

10.5 Memory Organisation:

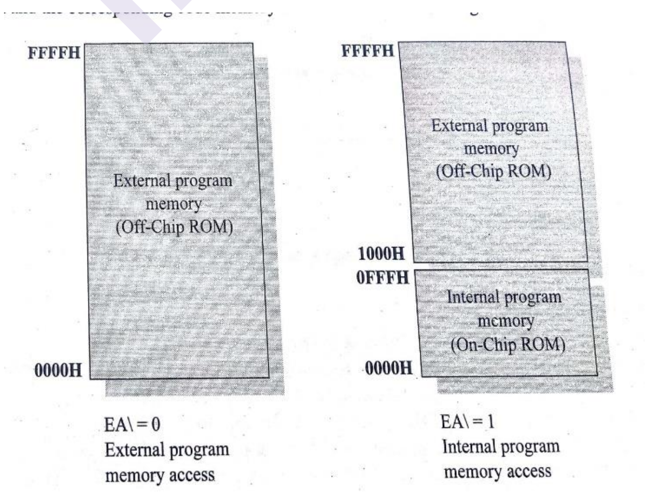

FFFFH
External program memory (Off-Chip ROM)
0000H
EA\ = 0
External program memory access
FFFFH
External program memory (Off-Chip ROM)
1000H
0FFFH
Internal program memory (On-Chip ROM)
0000H
EA\ = 1
Internal program memory access

Figure 10.3:Memory Organisation

- 8051architecture provide lowest 4K bytes of program memory as On-Chip memory.
- Switching between internal program memory and external program memory is accomplished by changing the logic level of the pin External Access (EA\----read as EA bar). EA\ pin is an active low pin.
- Connect EA\ pin to logic 1(Vcc) connects the chip to execute instruction from program memory upto 4K (program memory location upto 0FFFFH) from internal memory and 4K(program memory location 1000H)onwards from external memory, while connecting EA\ pin o logic 0(GND) configures chip to external program execution mode, where then entire code memory is executed from the external memory.
- The control signal for external program memory execution is PSEN\(Program Strobe Enable----PSEN\ is read as PSEN bar)
- For internal program memory fetches PSEN\ is not activated.

10.5.a External Program Memory Chip(ROM) interfacing:

- If the program memory is external , 16 I/O lines are used for accessing the external memory.
- Port 0 and Port 2 are used for external memory accessing, where initially Port 0 send lower order address when ALE signal of 8051 is high and then it function as input port for data transfer from the corresponding memory location while Port 2 sends higher order address.

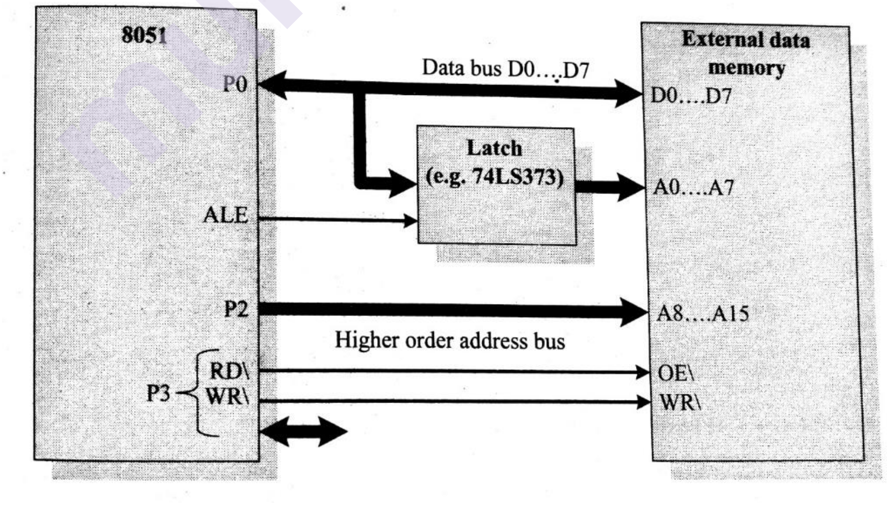


Figure 10.4:External Program Memory Organisation

10.5.b Data Memory interfacing:

- The basic 8051 architecture supports 128 bytes of internal data memory and 128 bytes of Special Function Register(SFR)
- The address range for internal user data memory is 00H to 7FH while SFR are residing at memory area 80H to FFH.
- 8051 supports interface for 64Kbytes of external data memory.
- The control signals used for external data memory accessed is Data Pointer(DPTR).
- The internal(ON chip) and external(OFF chip) data memory model of 8051 is given below

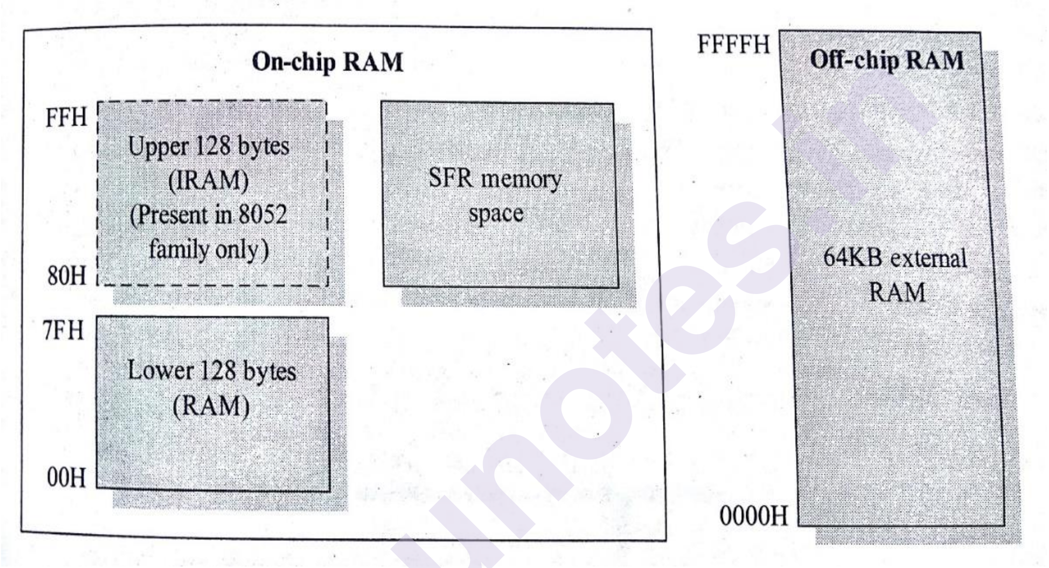


Figure 10.5: Data Memory Interfacing

10.5.c Interfacing External Data Memory access:

- External data memory address can be either one or two bytes long.
- Port 0 send lower order address when ALE signal of 8051 is high and if the memory address is two bytes and if it ranges upto 64K, the entire bits of Port 2 is used for holding the higher order value of the data memory address.
- If the memory address is 32K only 7 bits of Port 2 is required for addressing the memory.
- For 16K, only 6 lines of Port 2 are required for interfacing and so on.
- The diagram below show interfacing external data memory access

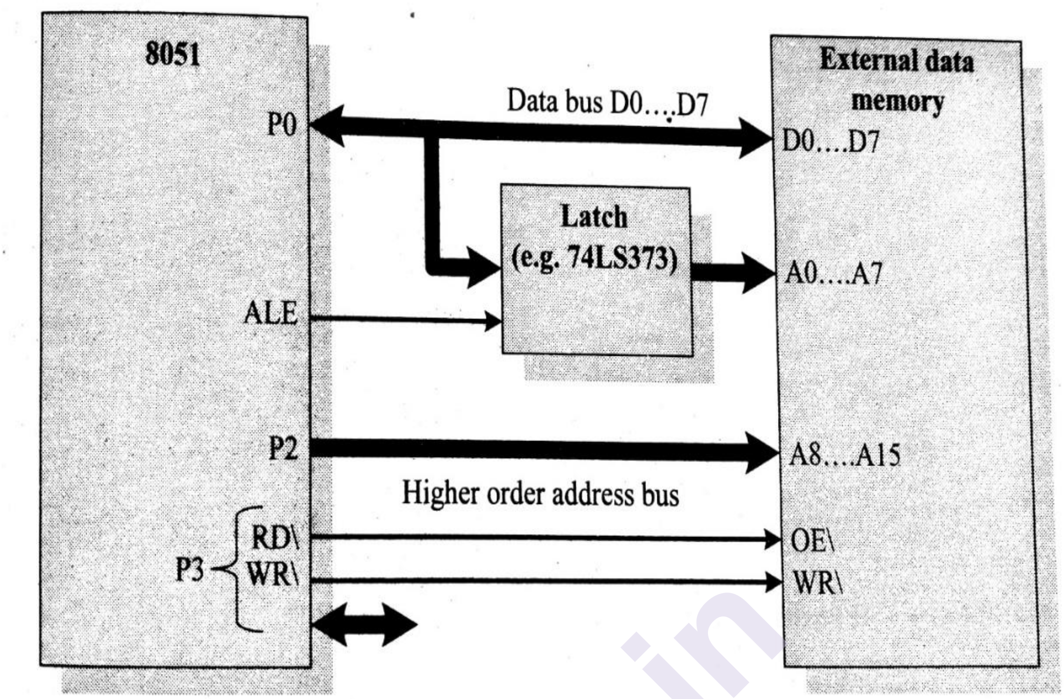


Figure 10.6: External Data Memory Interfacing

10.5.d Internal Memory:

- The 8051's on-chip memory consists of 256 memory bytes organized as follows:

First 128 bytes	00H to 1FH	Register Banks
	20H to 2FH	Bit Addressable RAM
	30H to 7FH	General Purpose RAM
Next 128 bytes	80H to FF H	Special Function Registers

□ Below figure represents internal memory organisation

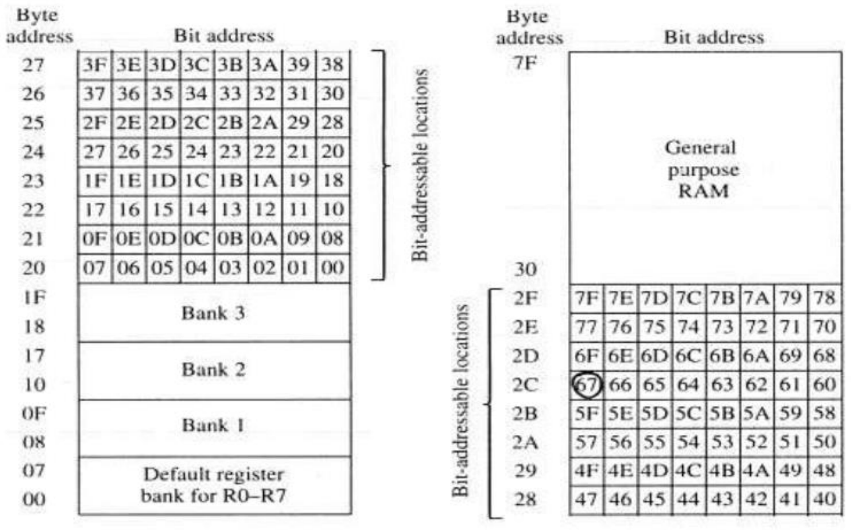

Byte address
Bit address
27 3F 3E 3D 3C 3B 3A 39 38
26 37 36 35 34 33 32 31 30
25 2F 2E 2D 2C 2B 2A 29 28
24 27 26 25 24 23 22 21 20
23 1F 1E 1D 1C 1B 1A 19 18
22 17 16 15 14 13 12 11 10
21 0F 0E 0D 0C 0B 0A 09 08
20 07 06 05 04 03 02 01 00
Bit-addressable locations
1F
Bank 3
18
17
Bank 2
10
0F
Bank 1
08
07
Default register bank for R0–R7
00
Byte address
Bit address
7F
General purpose RAM
30
Bit-addressable locations
2F 7F 7E 7D 7C 7B 7A 79 78
2E 77 76 75 74 73 72 71 70
2D 6F 6E 6D 6C 6B 6A 69 68
2C 67 66 65 64 63 62 61 60
2B 5F 5E 5D 5C 5B 5A 59 58
2A 57 56 55 54 53 52 51 50
29 4F 4E 4D 4C 4B 4A 49 48
28 47 46 45 44 43 42 41 40

Figure 10.7: Internal Memory Organization

10.6 Von-Neumann Memory Model for 8051:

- The code memory and data memory of 8051 can be combined together to get benefits of Von-Neumann model.
- A single memory chip with read/write option can be used for this purpose.
- The program memory can be allocated to the lower memory space starting from 0000H and data memory cane be assigned to some other specific area after the code memory.
- For program memory fetching and data memory read operations combine the PSEN\ and RD\ signals using AND gate and connect it to the Output enable(OE\) signal of the memory chip.
- The diagram below shows Von Neumann Memory Model

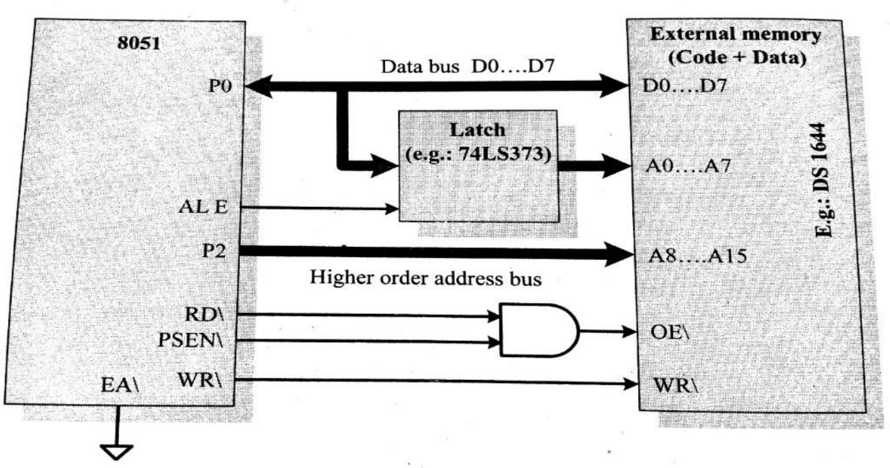


Figure 10.8: Von-Neumann model

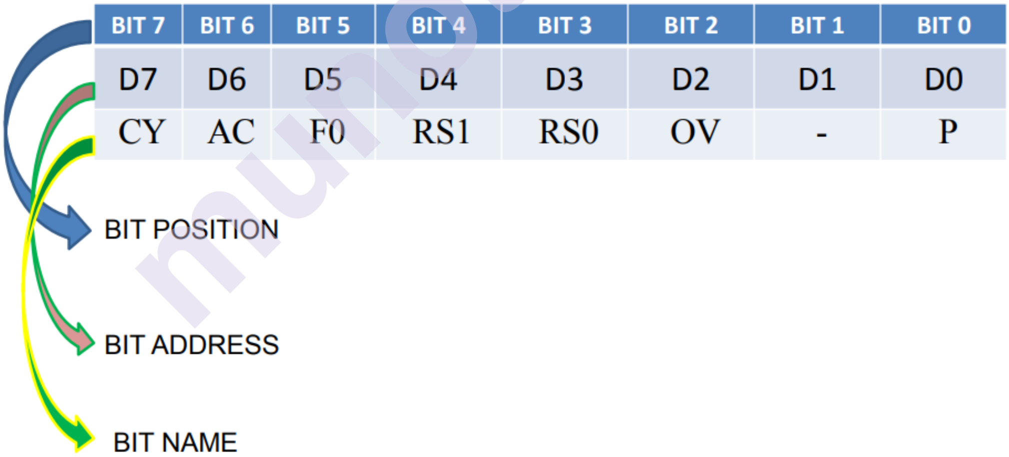
Drawback:

- Accidental corruption of program memory
- Reduction in total memory space. In separate program and data memory model the total available memory is 128KB(64KB program memory + 64KB data memory) whereas in combine only 64KB memory is available

10.7 Register Organisation:

10.7.a Special Function Register(SFR):

- The SFR registers are located within the Internal Memory in the address range 80h to FFh. Not all locations within this range are defined.
- Each SFR has a very specific function. Each SFR has an address (within the range 80h to FFh) and a name which reflects the purpose of the SFR.
- Although 128 byes of the SFR address space is defined only 21 SFR registers are defined in the standard 8051.
- Undefined SFR addresses should not be accessed as this might lead to some unpredictable results. Note some of the SFR registers are bit addressable. SFRs are accessed just like normal Internal RAM locations.
- PSW Program Status Word (flag register):



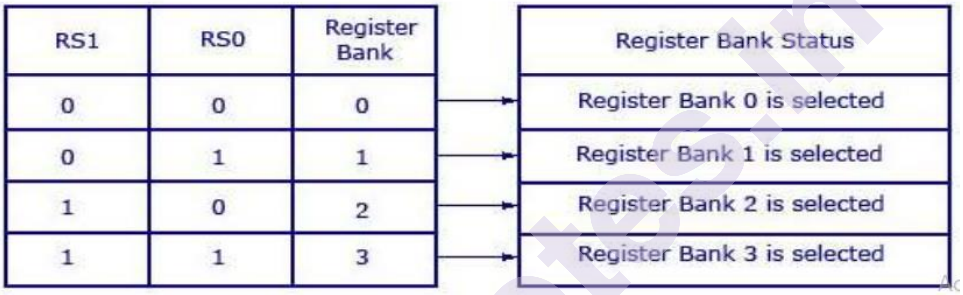
BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
D7	D6	D5	D4	D3	D2	D1	D0
CY	AC	F0	RS1	RS0	OV	-	P

Figure 10.9: PSW

- Carry Flag(CY):
- This is a conventional carry, or borrow, flag used in arithmetic operations. The carry flag is also used as the 'Boolean accumulator' for Boolean instruction operating at the bit level. This flag is sometimes referenced as the CY flag.
- Auxiliary Flag(AC):

- This is a conventional auxiliary carry (half carry) for use in BCD arithmetic.
- Flag 0.(F0):
- This is a general-purpose flag for user programming.
- Overflow(OV):
- This is a conventional overflow bit for signed arithmetic to determine if the result of a signed arithmetic operation is out of range.
- Even Parity flag(P):
- The parity flag is the accumulator parity flag, set to a value, 1 or 0, such that the number of '1' bits in the accumulator plus the parity bit add up to an even number.
- Register bank select 0 and register bank select 1. RS0 and RS1:

These bits define the active register bank (bank 0 is the default register bank).



RS1	RS0	Register Bank	Register Bank Status
0	0	0	Register Bank 0 is selected
0	1	1	Register Bank 1 is selected
1	0	2	Register Bank 2 is selected
1	1	3	Register Bank 3 is selected

Figure 10.10: Register Bank

- Stack Pointer :
- The Stack Pointer, SP, is an 8-bit SFR register at address 81h. The small address field (8 bits) and the limited space available in the Internal RAM confines the stack size, and this is sometimes a limitation for 8051 programmers.
- The SP contains the address of the data byte currently on the top of the stack. The SP pointer is initialised to a defined address.
- A new data item is 'pushed' on to the stack using a PUSH instruction which will cause the data item to be written to address SP + 1.
- Typical instructions, which cause modification to the stack are: PUSH, POP, LCALL, RET, RETI etc.
- The SP SFR, on start-up, is initialised to 07h so this means the stack will start at 08h and expand upwards in Internal RAM.
- If register banks 1 to 3 are to be used the SP SFR should be initialised to start higher up in Internal RAM.

- The following instruction is often used to initialise the stack:
- MOV SP, #2Fh
- Data pointer
- The Data Pointer, DPTR, is a special 16-bit register used to address the external code or external data memory.
- Since the SFR registers are just 8-bits wide the DPTR is stored in two SFR registers, where DPL (82h) holds the low byte of the DPTR and DPH (83h) holds the high byte of the DPTR.
- For example, if you wanted to write the value 46h to external data memory location 2500h, you might use the following instructions:
- MOV A, #46h ; Move immediate 8 bit data 46h to A (accumulator) MOV DPTR, #2504h ; Move immediate 16 bit address value 2504h to A.
- ; Now DPL holds 04h and DPH holds25h.
- MOVX @DPTR, A ; Move the value in A to external RAM location 2500h. uses indirect addressing.
- Note the MOVX (Move X) instruction is used to access external memory.

10.7.b SFR Registers for the Internal Timer:

- TCON :
- The Timer Control register is an SFR at address 88h, which is bit-addressable. TCON is used to configure and monitor the 8051 timers.
- The TCON SFR also contains some interrupt control bits, described later.
- TMOD:
  - The Timer Mode register is an SFR at address 89h and is used to define the operational modes for the timers.
  - TL0 (Timer 0 Low) and TH0 (Timer 0 High) are two SFR registers addressed at 8Ah and 8Bh respectively. The two registers are associated with Timer 0.
  - TL1 (Timer 1 Low) and TH1 (Timer 1 High) are two SFR registers addressed at 8Ch and 8Dh respectively. These two registers are associated with Timer 1.
- Power Control Register
  - PCON (Power Control) register is an SFR at address 87h. It contains various control bits including a control bit, which allows the 8051 to go to ‘sleep’ so as to save power when not in immediate use.
- Serial Port Registers

- □ SCON (Serial Control) is an SFR register located at addresses 98h, and it is bit addressable. SCON configures the behaviour of the on-chip serial port, setting up parameters such as the baud rate of the serial port, activating send and/or receive data, and setting up some specific control flags.
- □ SBUF (Serial Buffer) is an SFR register located at address 99h. SBUF is just a

- single byte deep buffer used for sending and receiving data via the on-chip serial port.
- Interrupt Registers :
  - □ IE (Interrupt Enable) is an SFR register at addresses A8h and is used to enable and disable specific interrupts. The MSB bit (bit 7) is used to disable all interrupts.
  - □ IP (Interrupt Priority) is an SFR register at addresses B8h and it is bit addressable. The IP register specifies the relative priority (high or low priority) of each interrupt. On the 8051, an interrupt may either be of low (0) priority or high (1) priority.

10.8 Input/Output (I/O)Ports:

- Port 0(P0):

	(MSB) P0.7	P0.6	P0.5	P0.4	P0.3	P0.2	P0.1	(LSB) P0.0
Direct Address 80H								
Bit Address	87	86	85	84	83	82	81	80

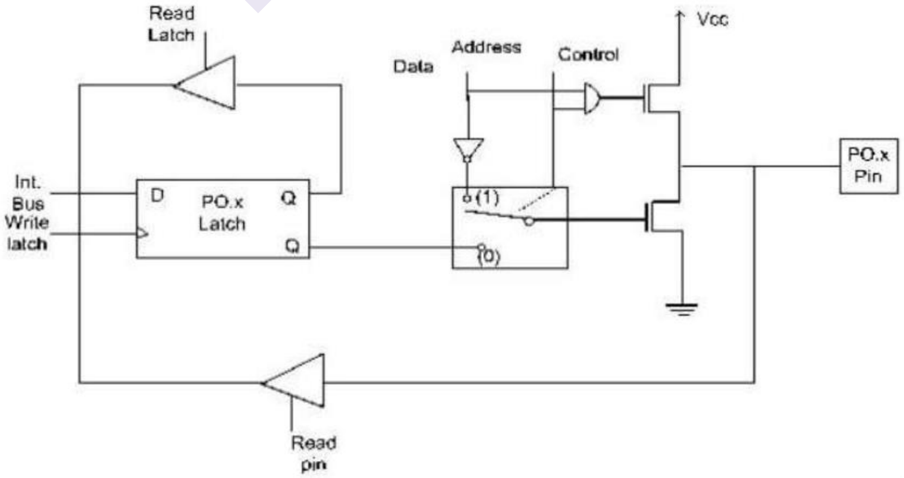
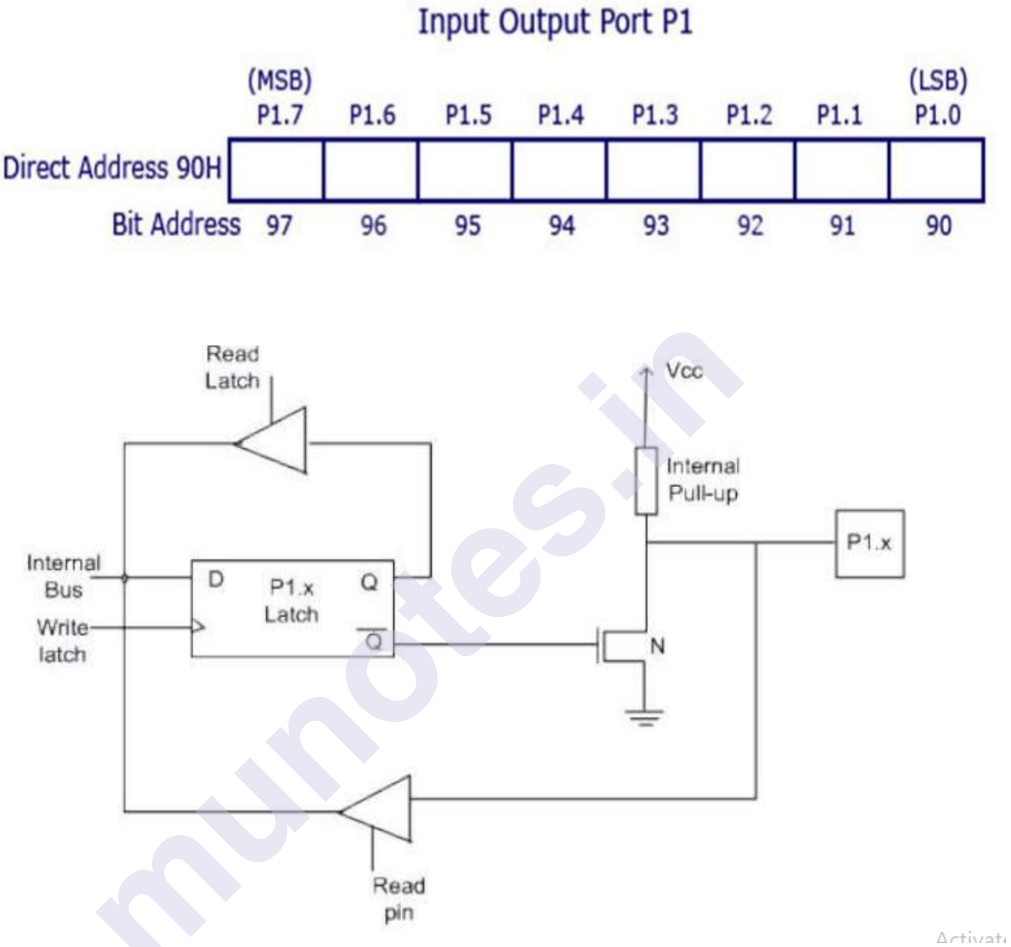


Figure 10.11: I/O Port P0 Structure

- Port-0 can be configured as a normal bidirectional I/O port or it can be used for address/data interfacing for accessing external memory.
- When control is '1', the port is used for address/data interfacing.
- When the control is '0', the port can be used as a normal bidirectional I/O port.
- Port 1(P1):



Input Output Port P1

	(MSB) P1.7	P1.6	P1.5	P1.4	P1.3	P1.2	P1.1	(LSB) P1.0
Direct Address 90H								
Bit Address	97	96	95	94	93	92	91	90

Figure 10.12: I/O Port P1 Structure

- Port-1 does not have any alternate function i.e. it is dedicated solely for I/O interfacing. When used as output port, the pin is pulled up or down through internal pull-up.
- To use port-1 as input port, '1' has to be written to the latch. In this input mode when '1' is written to the pin by the external device then it read fine. But when '0' is written to the pin by the external device then the external source must sink current due to internal pull-up.
- If the external device is not able to sink the current the pin voltage may rise, leading to a possible wrong reading.
- Port 2(P2):

Input Output Port P2

	(MSB) P2.7	P2.6	P2.5	P2.4	P2.3	P2.2	P2.1	(LSB) P2.0
Direct Address A0H								
Bit Address	A7	A6	A5	A4	A3	A2	A1	A0

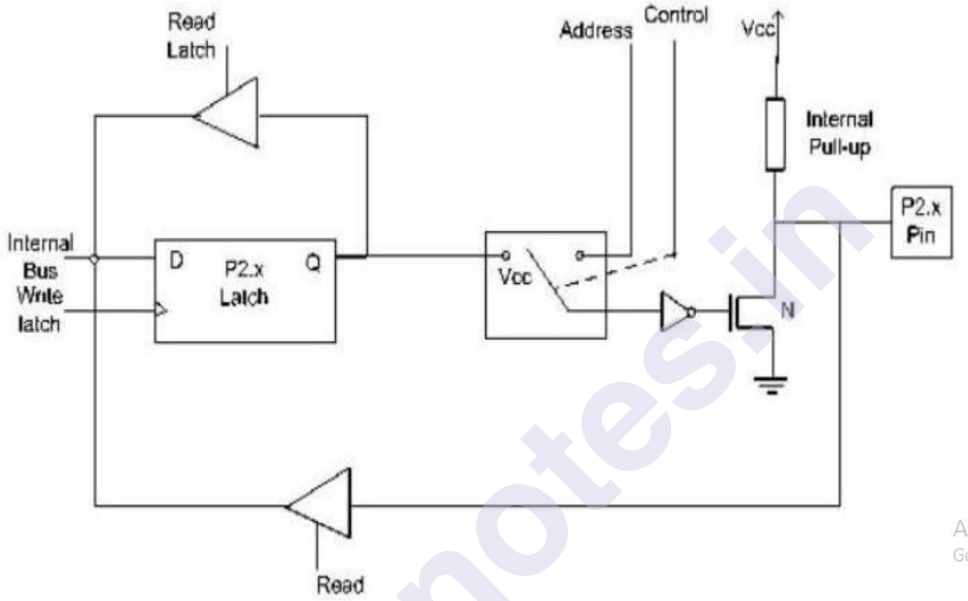


Figure 10.13: I/O Port P2 Structure

- Port-2 is used for higher external address byte or a normal input/output port. The I/O operation is similar to Port-1.
- Port-2 latch remains stable when Port-2 pin are used for external memory access.
- Here again due to internal pull-up there is limited current driving capability.
- Port 3(P3) :

	(MSB) P3.7	P3.6	P3.5	P3.4	P3.3	P3.2	P3.1	(LSB) P3.0
Direct Address B0H								
Bit Address	B7	B6	B5	B4	B3	B2	B1	B0

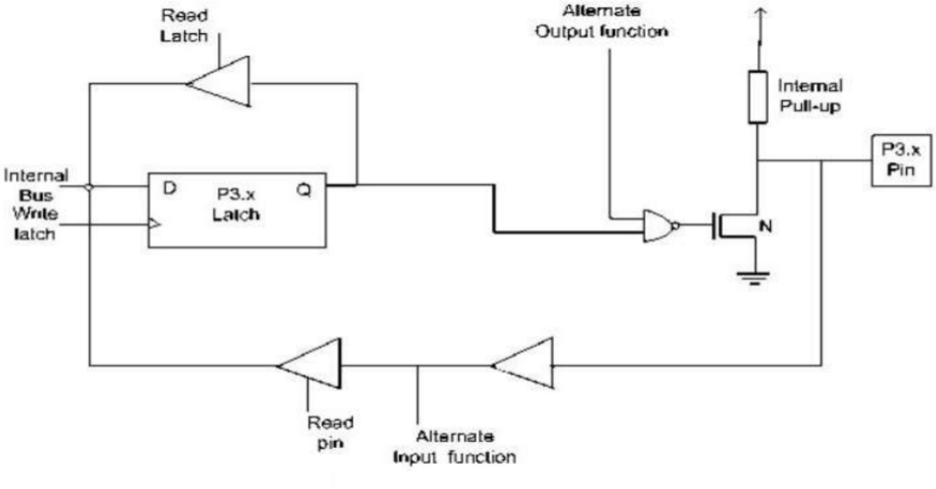


Figure 10.14: I/O Port P3 Structure

- Each pin of Port-3 can be individually programmed for I/O operation or for alternate function.
- The alternate function can be activated only if the corresponding latch has been written to '1'.
- To use the port as input port, '1' should be written to the latch.
- This port also has internal pull-up and limited current driving capability.

10.9 Chapter End Questions

Answer the following questions

- Explain the function of PORT 0 and PORT 3
- Explain TMOD and TCON
- Explain data memory interfacing.
- List and explain factors that need to be consider while selecting microcontroller.
- Explain memory organization in details.
- Explain interfacing external data memory access

10.10 Summary:

- All the factors that needs to be considered while selecting micro-controller is discussed.
- Various memory interfacing technique that is used to connect with micro-controller.
- Importance and functionality of 8051 ports.

10.11 Reference for further reading

- Embedded Systems by Rajkamal
- Introduction to embedded systems by Shibu K V
- https://www.tutorialspoint.com/microprocessor/microcontrollers_8051_architecture.htm
- https://www.watelectronics.com/8051-microcontroller-architecture/

❄❄❄❄❄❄❄

Unit 4

11

PROGRAMMING EMBEDDED SYSTEM

Unit Structure

11.0 Objectives

From this chapter student will be able to explain the structure of embedded program

- Students will gain the knowledge about Infinite loop usage in embedded system
- Student will be able to gain knowledge about various debugging technique.

11.1 Structure of Embedded Program:

1. **Comments:**

   - Comments are readable text are written to help us (the reader) understand the code easily. They are ignored by the compiler and do not take up any memory in the final code (after compilation).
   - There are two ways you can write comments: one is the single line comments denoted by // and the other is multiline comments denoted by /*….*/.

2. **Preprocessor Directive:**

- A Preprocessor Directive in Embedded C is an indication to the compiler that it must look in to this file for symbols that are not defined in the program.
- In C Programming Language (also in Embedded C), Preprocessor Directives are usually represented using #include… or #define….
- In Embedded C Programming, we usually use the preprocessor directive to indicate a header file specific to the microcontroller, which contains all the SFRs and the bits in those SFRs.
- In case of 8051, Keil Compiler has the file "reg51.h", which must be written at the beginning of every Embedded C Program.

3. Global Variables:

- Global Variables, as the name suggests, are Global to the program i.e. they can be accessed anywhere in the program.

4. Local Variables:

- Local Variables, in contrast to Global Variables, are confined to their respective function.

5. Main Function:

- Every C or Embedded C Program has one main function, from where the execution of the program begins.

Example:

```
#include<reg51.h> // Preprocessor Directive
void delay (int); // Delay Function Declaration
void main(void) // Main Function
{
P1 = 0x00;  /* Making PORT1 pins LOW. All the LEDs are OFF.
(P1 is PORT1, as defined in reg51.h) */
while(1) // infinite loop
{
P1 = 0xFF; // Making PORT1 Pins HIGH i.e. LEDs are ON.
delay(1000);
/* Calling Delay function with Function parameter as 1000.
This will cause a delay of 1000mS i.e. 1 second */
P1 = 0x00; // Making PORT1 Pins LOW i.e. LEDs are OFF.
delay(1000);
}
}
void delay (int d) // Delay Function Definition
{
unsigned int i=0; // Local Variable. Accessible only in this function.

/* This following step is responsible for causing delay of 1000mS (or as per the
value entered while calling the delay function) */
for(;d>0;d–)
{
for(i=250;i>0;i – -);
for(i=248;i>0;i – -);
}
}
```

11.2 Infinite Loop

- Almost every embedded ap-plication involves delay programming.
  - Embedded applications employ delay programming for waiting for a fixed time interval till a device is ready, for inserting delay between displays updating to give the user sufficient time to view the contents displayed, delays involved in bit transmission and reception in asynchronous serial transmissions like I2C, I-Wire data transfer, delay for key de-bouncing etc.
  - One of the most fundamental differences between programs developed for embedded systems and those written for other computer platforms is that the embedded programs almost always end with an infinite loop.
  - Some delay requirements in .embedded application may be critical, meaning delay accuracy should be within a very narrow tolerance band.
  - Typical example is delay used in bit data transmission. If the delay employed is not accurate, the bits may lost while transmission or reception. Certain delay requirements in embedded application may not be much critical, e.g. display updating delay.
  - It is easy to code delays in desktop applications under DOS or Windows operating systems. The library function delay0 in DOS and Sleep O in Windows provides delays in milliseconds with reasonable accuracy. Coding delay routines in embedded applications is bit difficult.

□ Delay routine requires a complete re-work if the target clock frequency is changed. Normally 'for loops' are used for coding delays. Infinite loops are created using various loop control instructions like while (), do while (), for and goto labels.

□ The super loop created by while (1) instruction in a traditional super loop based embedded firmware design is a typical example for infinite loop in embedded application development.

- The infinite loop is necessary because the embedded software's job is never done. It is intended to be run until either the world comes to an end or the board is reset, whichever happens first.
- In addition, most embedded systems have only one piece of software running on them. And although the hardware is important, it is not a digital watch or a cellular phone or a microwave oven without that embedded software. If the software stops running, the hardware is rendered useless.
- So the functional parts of an embedded program are almost always surrounded by an infinite loop that ensures that they will run forever.

□ Infinite loop using while:

```
While(1)
{
              }
```

□ Infinite loop using do while:

```
do
{
              } while(1);
```

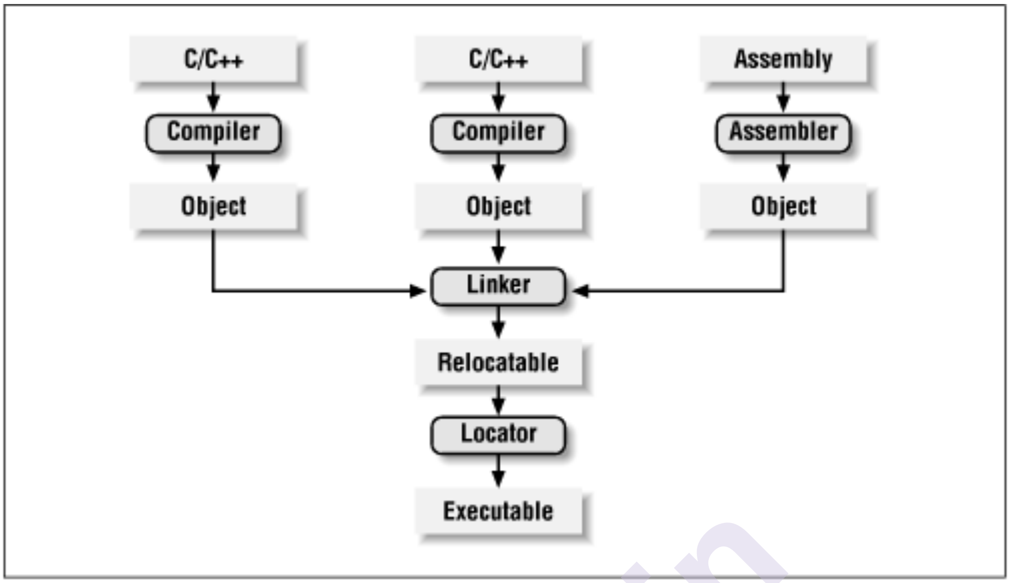
□ Infinite loop using for :

```
for ( ; ; ; )
{
       }
```

□ Infinite loop using goto (‘goto' when combined with a 'label' can create infinite loops)

```
label : // Task to be repeated
    //………..
  //………...
Goto label ;
```

11.3 Embedded Software Development Process



- The process of converting the source code representation of your embedded software into an executable binary image involves three distinct steps.
- First, each of the source files must be compiled or assembled into an object file.
- Second, all of the object files that result from the first step must be linked together to produce a single object file, called the relocatable program.
- Finally, physical memory addresses must be assigned to the relative offsets within the relocatable program in a process called relocation.
- The result of this third step is a file that contains an executable binary image that is ready to be run on the embedded system.
- The diagram above shows the embedded development process.

11.4 Compiling

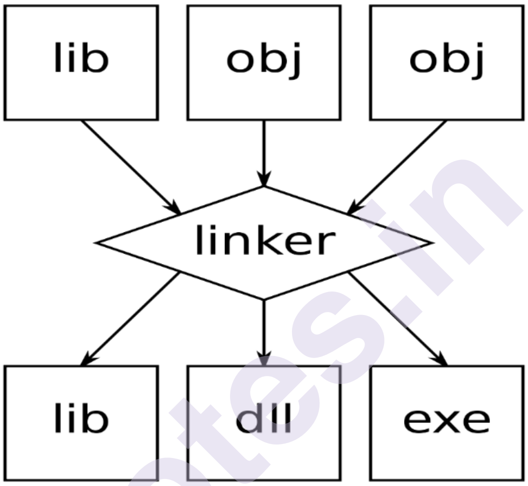
- The job of a compiler is mainly to translate programs written in some human-readable language into an equivalent set of opcodes for a particular processor.
- In that sense, an assembler is also a compiler (you might call it an "assembly language compiler") but one that performs a much simpler one-to-one translation from one line of human-readable mnemonics to the equivalent opcode.

- Together these tools make up the first step of the embedded software build process.
- Each processor has its own unique machine language, so you need to choose a compiler that is capable of producing programs for your specific target processor.
- In the embedded systems case, this compiler almost always runs on the host computer. It simply doesn't make sense to execute the compiler on the embedded system itself.
- A compiler such as this-that runs on one computer platform and produces code for another-is called a cross-compiler.
- The use of a cross-compiler is one of the defining features of embedded software development.
- The GNU C/C++ compiler (gcc) and assembler (as) can be configured as either native compilers or cross-compilers. As cross-compilers these tools support an impressive set of host-target combinations.
- Regardless of the input language (C/C++, assembly, or any other), the output of the cross-compiler will be an object file. This is a specially formatted binary file that contains the set of instructions and data resulting from the language translation process.
    - The contents of an object file can be thought of as a very large, flexible data structure. The structure of the file is often defined by a standard format such as the Common Object File Format (COFF) or Executable and Linkable Format (ELF).
    - If you'll be using more than one compiler (i.e., you'll be writing parts of your program in different source languages), you need to make sure that each compiler is capable of producing object files in the same format; gcc supports both of the file formats previously mentioned.
    - Although many compilers (particularly those that run on Unix platforms) support standard object file formats such as COFF and ELF, some others produce object files only in proprietary formats. If you're using one of the compilers in the latter group, you might find that you need to get all of your other development tools from the same vendor.
    - Most object files begin with a header that describes the sections that follow. Each of these sections contains one or more blocks of code or data that originated within the source file you created.

  - □ However, the compiler has regrouped these blocks into related sections. For example, in gcc all of the code blocks are collected into a section called text, initialized global variables (and their initial values) into a section called data, and uninitialized global variables into a section called bss.

11.5 Linking

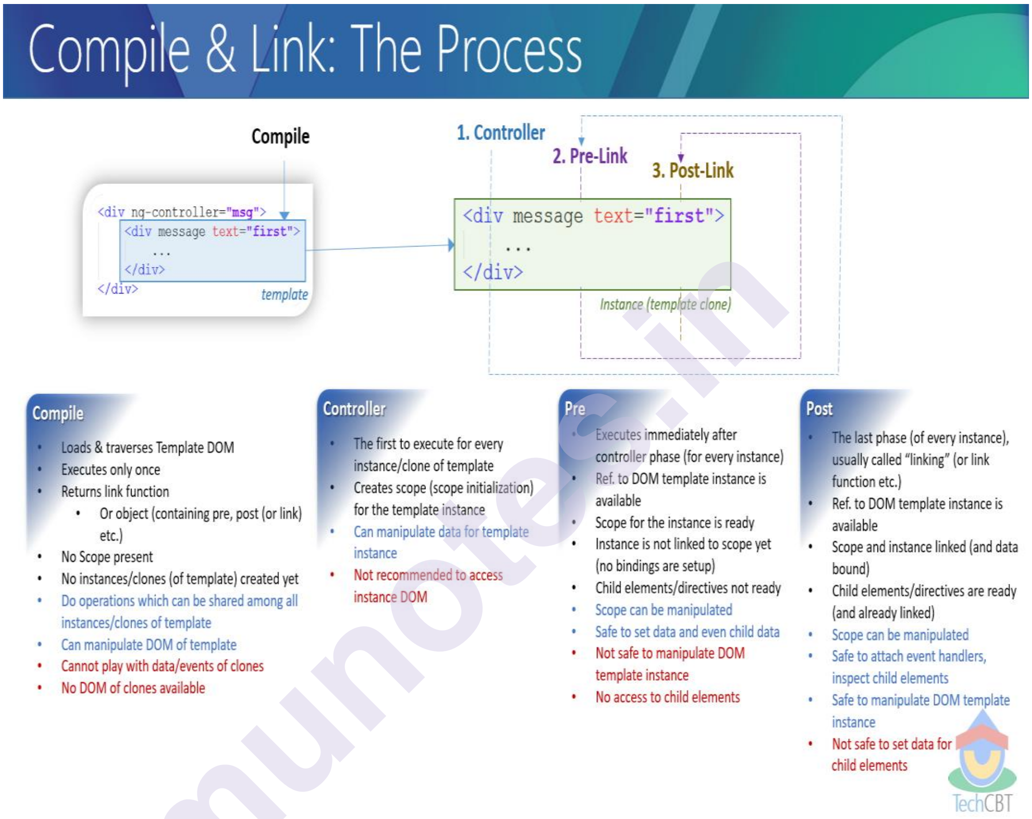
- All of the object files resulting from step one must be combined in a special way before the program can be executed.



- The object files themselves are individually incomplete, most notably in that some of the internal variable and function references have not yet been resolved.
- The job of the linker is to combine these object files and, in the process, to resolve all of the   unresolved symbols.
- The output of the linker is a new object file that contains all of the code and data from the input object files and is in the same object file format. It does this by merging the text, data, and bss sections of the input files.
- So, when the linker is finished executing, all of the machine language code from all of the input object files will be in the text section of the new file, and all of the initialized and   uninitialized variables will reside in the new data and bss sections, respectively.
- While the linker is in the process of merging the section contents, it is also on the lookout for unresolved symbols.
  - □ For example, if one object file contains an unresolved reference to a variable named foo and a variable with that same name is declared in one of the other object files, the linker will match them up.

- The unresolved reference will be replaced with a reference to the actual variable. section, its entry in the symbol table will now contain that address.
- The GNU linker (ld) runs on all of the same host platforms as the GNU compiler. It is a command-line tool that takes the names of all the object files, and possibly libraries, to be linked as arguments.
- With embedded software, a special object file that contains the compiled startup code, which is covered later in this section, must also be included within this list.
- The GNU linker also has a scripting language that can be used to exercise tighter control over the object file that is output.
- If the same symbol is declared in more than one object file, the linker is unable to proceed. It will likely complain to the programmer (by displaying an error message) and exit.
- On the other hand, if a symbol reference remains unresolved after all of the object files have been merged, the linker will try to resolve the reference on its own.
- The reference might be to a function, such as memcpy, strlen, or malloc, that is part of the standard C library, so the linker will open each of the libraries described to it on the command line (in the order provided) and examine their symbol tables.
- If the linker thus discovers a function or variable with that name, the reference will be resolved by including the associated code and data sections within the output object file. Note that the GNU linker uses selective linking, which keeps other unreferenced functions out of the linker's output image.
- After merging all of the code and data sections and resolving all of the symbol references, the linker produces an object file that is a special "relocatable" copy of the program.
- In other words, the program is complete except for one thing: no memory addresses have yet been assigned to the code and data sections within. If you weren't working on an embedded system, you'd be finished building your software now.
- But embedded programmers aren't always finished with the build process at this point. The addresses of the symbols in the linking process are relative.

- Even if your embedded system includes an operating system, you'll probably still need an absolutely located binary image.
- In fact, if there is an operating system, the code and data of which it consists are most likely within the relocatable program too. The entire embedded application—including the operating system—is frequently statically linked together and executed as a single binary image.
- Compile and Linking Process can be well explained by below flowchart



Compile & Link: The Process

```
<div ng-controller="msg">
  <div message text="first">
    ...
  </div>
</div>
```

```
<div message text="first">
  ...
</div>
```

Compile

- Loads & traverses Template DOM
- Executes only once
- Returns link function
  - Or object (containing pre, post (or link) etc.)
- No Scope present
- No instances/clones (of template) created yet
- Do operations which can be shared among all instances/clones of template
- Can manipulate DOM of template
- Cannot play with data/events of clones
- No DOM of clones available

Controller

- The first to execute for every instance/clone of template
- Creates scope (scope initialization) for the template instance
- Can manipulate data for template instance
- Not recommended to access instance DOM

Pre

- Executes immediately after controller phase (for every instance)
- Ref. to DOM template instance is available
- Scope for the instance is ready
- Instance is not linked to scope yet (no bindings are setup)
- Child elements/directives not ready
- Scope can be manipulated
- Safe to set data and even child data
- Not safe to manipulate DOM template instance
- No access to child elements

Post

- The last phase (of every instance), usually called "linking" (or link function etc.)
- Ref. to DOM template instance is available
- Scope and instance linked (and data bound)
- Child elements/directives are ready (and already linked)
- Scope can be manipulated
- Safe to attach event handlers, inspect child elements
- Safe to manipulate DOM template instance
- Not safe to set data for child elements

11.6 Locating:

- The tool that performs the conversion from relocatable program to executable binary image is called a locator.
- It takes responsibility for the easiest step of the three. In fact, you will have to do most of the work in this step yourself, by providing information about the memory on the target board as input to the locator.
- The locator will use this information to assign physical memory addresses to each of the code and data sections within the relocatable program.
- It will then produce an output file that contains a binary memory image that can be loaded into the target ROM.

- In many cases, the locator is a separate development tool. However, in the case of the GNU tools, this functionality is built right into the linker.

11.7 Debugging:

- Debugging in embedded applications is the process of checking the firmware execution, monitoring the target processor, register and memory while the execution process is going ON.
- Debugging in embedded application is broadly classified into two types:
  i. Hardware debugging
  ii. Software debugging

11.7.1 Hardware debugging

- Even though the firmware is bug free and everything is intact in the board, your embedded product need not function as per the expected behaviour in the first attempt for various hardware related reasons like dry soldering of components, missing connections in the due to any un-noticed errors in the PCB layout design, misplaced components, signal corruption due to noise, etc.

  □ The only way to sbrt out these issues and figure out the real problem creator is debugging the target board. Hardware debugging is not similar to firmware debugging.

  □ Hardware debugging involves the monitoring of various signals

  □ The various hardware debugging tools are as follows:

i. Magnifying glass or lens:

□ You might have noticed watch repairer wearing a small magnifying glass while engaged·in repairing a watch. They use the magnifying glass to view the minute components inside the watch in an enlarged manner so that they can easily work with them.

- Similar to a watch repairer, magnifying glass is the pri-mary hardware debugging tool for an embedded hardware debugging professional.

- A magnifying glass is a powerful visual inspection tool. With a magnifying glass (lens), the surface of the target board can be examined thoroughly for dry soldering of components, missing components, improper placement of components, improper.soldering, track (PCB connection) damage, short of tracks, etc.

- Nowadays high quality magnifying stations are available for visual inspection. The magnifying station incorporates magnifying glasses attached to a stand with CFL tubes for providing proper illumination for inspection. The station usually incorporates multiple magnifying lenses. The main lens acts as a visual inspection tool for the entire hardware board whereas the other small lens within the station is used for magnifying a relatively small area of the board which requires thorough inspection.

ii. Multimeter:

- I believe the name of the instrument itself is sufficient to give· an outline of its usage. A multimeter is used for measuring various electrical quantities like voltage (Both AC and DC), current (DC as well as AC), resistance, capacitance, continuity checking, transistor checking, cathode and anode identification -of diode, etc.

- Any multimeter will work over a specific range for each measurement. A multimeter is the most valuable tool in the toolkit of an-embedded hardware developer.

- It is the primary debugging tool for physical contact based hardware debugging and almost all developers start debugging the hardware with it.

- In embedded hardware debugging it is mainly used for checking the circuit continuity between different points on the board, measuring the supply voltage, checking the signal value, polarity, etc.

- Both analog and digital versions of a multimeter are available. The digital version is preferred over analog the one for various reasons like readability, accuracy, etc. Fluke, Rishab, Philips,. etc. are the manufacturers of commonly available high quality digital multimeters

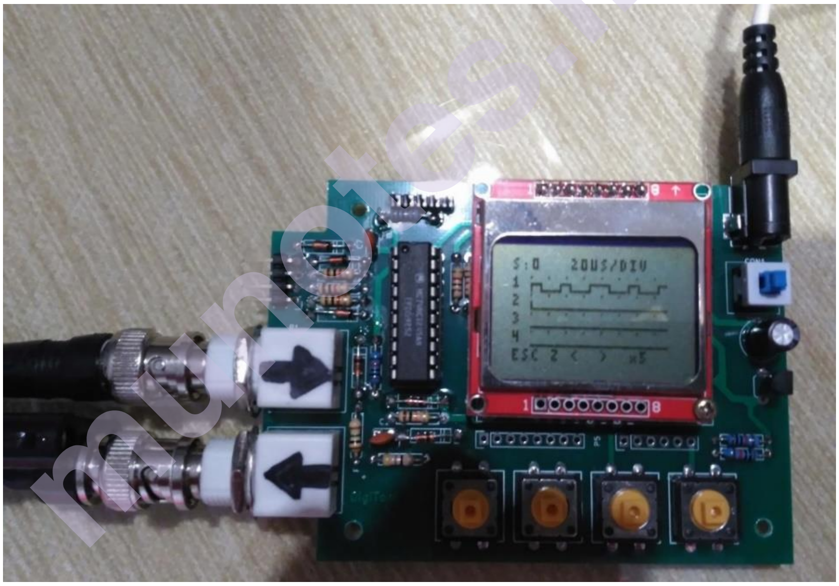
iii. CRO:

- Cathode Ray Oscilloscope (CRO) is a little more sophisticated tool compared to a multimeter. You might have studied the operation and use of a CRO in your basic electronic lab.
- Just to refresh your brain, CRO is used for waveform capturing and analysis, measurement of signal strength, etc.
- By connecting the point under observation on the target board ts>, the Channels of the Oscilloscope, the waveforms can be captured and analysed for expected behaviour.
- CRO is a very good tool in analysing interference noise in the power supply line and other signal lines.
- Monitoring the crystal oscillator signal from the target board is a typical example of the usage of CRO for waveform capturing and analysis in target board debugging. CROs are available in both analog and digital

versions. Though Digital CROs are costly, feature wise they are best suited for target board debugging applications.

- Digital CROs are avail-able for high frequency support and they also incorporate modern techniques for recording waveform over a period of time, capturing waves on the basis of a configurable event (trigger) from the target board (e.g. High to low transition of a port pin of the target processor).
- Most of the modern digital CROs contain more than one channel and it is easy to capture and analyse various signals from the target board using multiple channels simultaneously.
- Various measurements like phase, amplitude, etc. is also possible with CROs. Tektronix, Agilent, Philips, etc. are the manufacturers of high precision good quality digital CROs.

iv. Logic analyzer:



- A logic analyser is the big brother of digital CRO. Logic analyser is used for capturing digital data (logic 1 and 0) from a digital circuitry whereas CRO is employed in capturing all kinds of waves including logic signals.
- Another major limitation of CRO is that the total number of logic signals/waveforms that can be ·captured with a CRO is limited to the number of channels.
- A logic analyser contains special con-nectors and clips which can be attached to the target board for capturing digital data.

□ In target board de-bugging applications, a logic analyser captures the states of various port pins, address bus and data bus I of the target processor/controller, etc.

□ Logic analysers give an exact reflection of what happens when a particular line of firmware is running.

□ This is achieved by capturing the address line logic and data j line logic of target hardware. Most modern logic analysers contain provisions for storing captured selecting a desired region of the captured waveform, zooming selected region of the captured waveform, etc. Tektronix, Agilent, etc. are the giants in the logic analyser market.

v. Function generator:

□ Function generator is not a debugging tool. It is an input signal simulator tool.

□ A function generator is capable of producing various periodic waveforms like sine wave, square wave, saw-tooth wave, etc. with different frequencies and amplitude.

□ Sometimes the target board may require some kind of periodic waveform with a particular frequency as input to some part of the board.

□ Thus, in a debugging environment, the function generator serves the purpose of generating and supplying required signals.

11.7.2 Software Debugging:

- Software debugging is concerned with the monitoring declaration of processor directives, variables associated with the program and functions.
- The various method of Software debugging are as follows

  i. Incremental Debugging Techniques.

  ii. Inline Break Point Based Software Debugging

- Incremental Debugging Techniques.
  - □ In this technique the entire code is not burned into the EEPROM chip.
  - □ The entire code is divided into many modules and each module is burned one after the other.
  - □ After burning the first module it is check for errors by running it, if no error is detected then other module is burned.
  - □ This technique is time consuming but it is more reliable debugging technique.
  - □ This debugging technique is one time process once the software is tested it can go for mass production.
  - □ This technique is widely used in small, simple system developments and in those products where time is not a big constraint.
- Inline Break Point Based Software Debugging
  - □ It is a primitive method of software debugging.
  - □ In this technique we insert an inline debugging code immediately after reaching at a specified point of the code.
  - □ The debug code is a “printf” command which prints a string as per the software depending on the execution of the software at the inline debug code.

11.8 Chapter End Questions

Answer the following questions.

1. Explain structure of embedded program with suitable example.
2. Why infinite loop is preferred in embedded system? Give example.
3. Explain Software Debugging
4. What is hardware debugging? Explain any 2 in detail.
5. What is Inline break point?
6. Explain Linker.
7. Explain locator.
8. What is compiler?

11.9 Summary

- In depth explanation of various terms associated with embedded program was discussed and understood.
- The program flow concept was understood through embedded structure program.
- In depth hardware and software debugging technique using various electronic tools such as CRO, Multimeter, Logic Analyser was discussed.

11.10 Reference for further reading

- Introduction to embedded systems by Shibu K V
- The 8051 Microcontroller and Embedded Systems by Muhammad Ali Mazidi
- DyVuw.png (1920×1080) (imgur.com)
- Compiling, Linking, and Locating | Barr Group
- Basics of Embedded C Program : Introduction, Structure and Example (electronicshub.org)

❄❄❄❄❄❄❄

Unit 5

12

REAL TIME OPERATING SYSTEM

Unit Structure

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

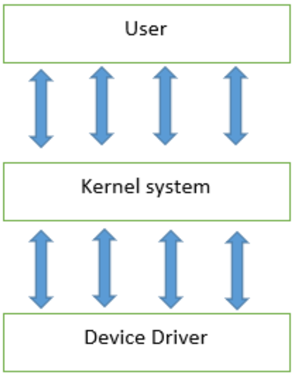
A real-time operating system (RTOS) is an operating system (OS) intended to serve real-time applications that process data as it comes in, typically without buffer delays. Processing time requirements (including any OS delay) are measured in tenths of seconds or shorter increments of time. In a RTOS, 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.



OS supports the all extended devices. Operating system is a combination of resources manager and peripheral device. The kernel is a core system in operating system. It is responsible for managing the interaction in between user application and device driver like CPU, Memory and Device etc. Kernel handle the peripherals devices all input/output requested. Kernel code is loaded into a separated memory space, which is protected from user space. In a kernel space, kernel handles all hardware devices such as hard disk running process task. In control user space, handle application program like browser, audio/video player.

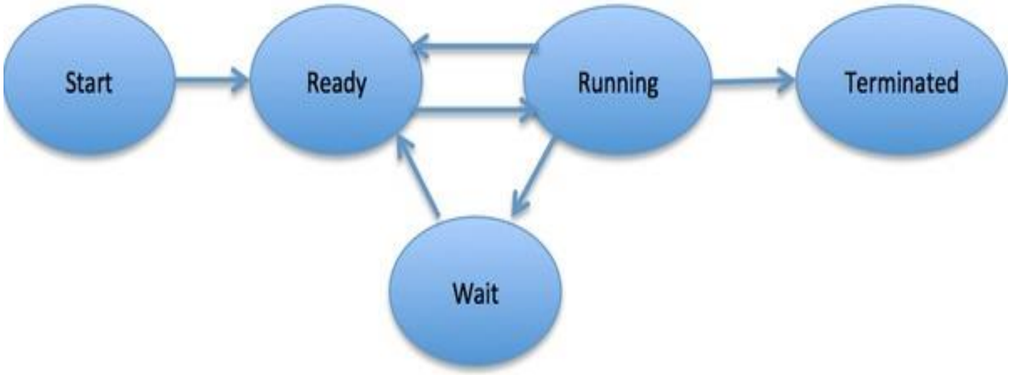
12.4.1 Tasks & Task states

Task is a basic piece of software that can be executed of others. In multitasking multiple task is executed at a time, it executed parallel. Operating system is responsible for deciding which task perform the task first. Task context record the all information about the task states, before task taking control of it. The utility programs may also be considered tasks or subtask as all program may make request of other program. It's easy to confuse the terms Multithreading and multitasking. Multithreading is the management of multiple uses of the same program.

The terms *task* and *process* are often used interchangeably, although some operating systems make a distinction between the two. The terms *task* and *process* are often used interchangeably, although some operating systems make a distinction between the two.

Task is that executing unit of computation, which is controlled by OS process scheduling mechanism, which lets it execute on the CPU and OS process resource-management mechanism that lets it use the system memory and other system-resources such as network, file, display or printer.

Task states



- **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.

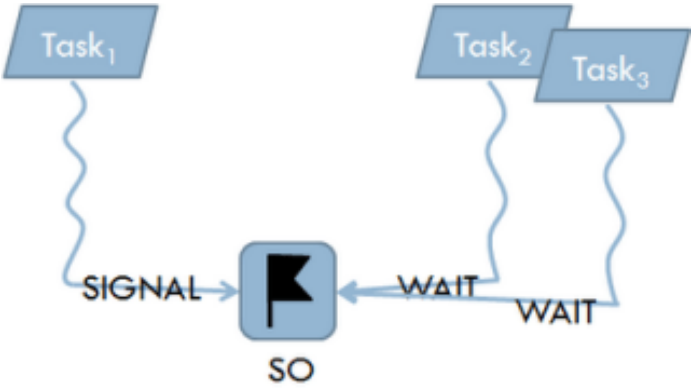
12.4.2 Task Synchronization

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:



We now have two tasks, Task$_2$ and Task$_3$ blocked waiting on the SO. Task$_1$ now reaches the synchronisation point and signals the event. Again we could have the case of bilateral and unilateral sync, though typically Task$_1$ doesn't block if no other task is waiting (unilateral model).

12.4.3 Schedulers

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 a put to a waiting state if its execution time expires.

   Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

   - Long-Term Scheduler
   - Short-Term Scheduler
   - Medium-Term Scheduler

- **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.

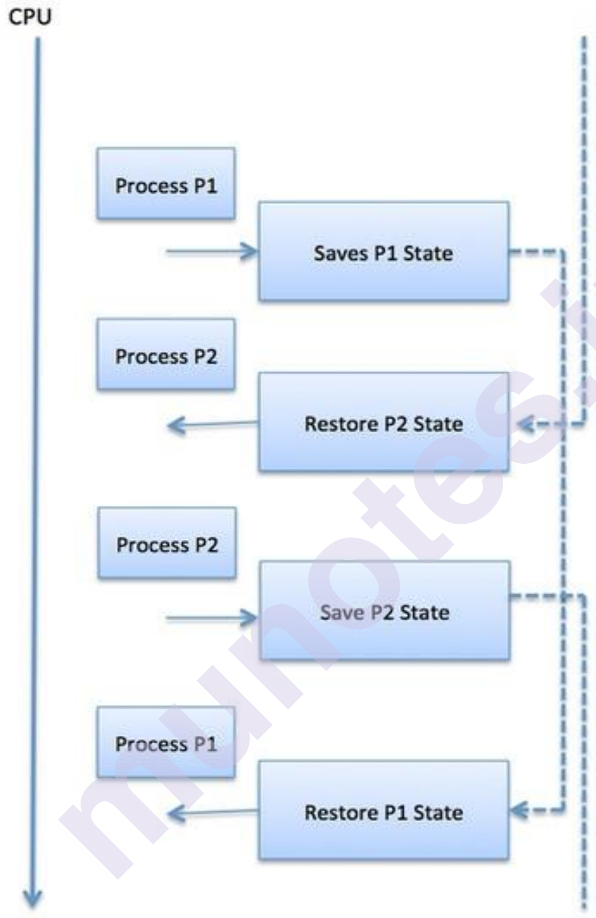
- **Medium Term Scheduler**

  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 processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

When the scheduler switches the CPU from executing one process to execute another, the state from the current running process is stored into the process control block. After this, the state for the process to run next is loaded from its own PCB and used to set the PC, registers, etc. At that point, the second process can start executing.



12.5 Types of Operating System

12.5.1 General Operating System

- Distributed OS
- Batch OS
- Multitasking OS
- Network OS
- Mobile OS

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.

Application
RTOS (Deadline)
Hardware

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.

Examples of Real-Time OS: Medical imaging systems, robots, Coffee wending machine, Flight control system, Anti Lock breaking system for vehicles etc.

Types of the real-time operating system are:

- **Hard real-time OS**
  The hard real-time OS is the operating system for mainly the applications in which the slightest delay is also unacceptable. The ***time constraints*** of such applications are very strict. Such systems are built for life-saving equipment like parachutes, airbags, Nuclear power plant control etc. which immediately need to be in action if an accident happens. Flexibility not allowed & All deadlines must be followed.

- **Soft real-time OS**
  The soft real-time OS is the operating system for applications where time constraint is not very strict. In a soft real-time system, an important task is prioritized over less important tasks, and this priority remains active until the completion of the task. Here Tasks are performed as fast as possible (Best Affordable system). Furthermore, a time limit is always set for a specific job, enabling short time delays for future tasks, which is acceptable. For Example- Multimedia network, Online Database & Games, virtual reality, reservation system, etc.

12.6 Real Time Characteristics:

Real-time System is a system that is put through real time which means response is obtained within a specified timing constraint or system meets the specified deadline. Real time system is of two types – Hard and Soft. Both are used in different cases. Hard real time systems are used where even the delay of some nano or micro seconds are not allowed. Soft real time systems provide some relaxation in time expression.

Following are the some of the characteristics of Real-time System:

Time Constraint
Time constraints related with real-time systems simply means that time interval allotted for the response of the ongoing program. This deadline means that the task should be completed within this time interval. Real-time system is responsible for the completion of all tasks within their time intervals.

Correctness:-
Correctness is one of the prominent part of real-time systems. Real-time systems

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.

The process of selecting an operating system for an embedded system is outlined and reviewed. On Desktop computer the selection of an operating system is largely a matter of taste. Having conclude that an OS is required for a project, there is the question of making a selection. There are four option: select high end operating system, such as Linux or an embedded variant of windows; select a real time operating system from among many choices; deploy one of the free operating system that are widely available.

12.8 DESIGN AND DEVELOPMENT

Design and develop tiny RTOS for low cost, vehicle monitoring system we plot sensors on one board for monitoring vehicle so this board provide security to the vehicle's owner. The proposed system would use new technology based on embedded board and its advanced database at a real-time for storing all updates of sensors. The proposed system works on atmega328 microcontroller of avr family which is used for handling vehicle tracking sensors on this board. If the speed of vehicle crosses the limit then alert message will display on your smartphone these all sensors worked in real time means all are work simultaneously. Each sensor provides a specified time delay. This operating system has to respond quickly. The microcontroller is connected to owner's mobile via Bluetooth module. These all sensors provide all its sensor status to the vehicle owners on smartphone.so all the status can be viewed by the owner to check the vehicle condition. All the previous history of sensors stored in the database.

12.9 Embedded System Development Environment – IDE

12.9.1 Introduction of IDE

An Integrated Development Environment (IDE) is a software application that provides a programming environment to streamline developing and debugging software. Rather than performing all the steps required to make an executable program as unrelated individual tasks, it brings all the tools needed into one application and workspace. Each of the tools has an awareness of the environment, and they work together to present a seamless development set for the developer.

Even a simple search for IDEs will turn up quite a few choices. IDEs are available from Open Source communities, vendors, and software companies. They range from free to pricing dependent upon the number of licenses required. There isn't a standard for IDEs and each has its own capabilities, along with strengths and weaknesses. Generally, an IDE provides an easy-to-use interface, automates development steps, and allows developers to run and debug programs all from one screen. It can also provide the link from a development operating system to an application target platform, like a desktop environment, smartphone or microprocessor.

12.9.2 Best Embedded Software Development Tools

Here's the list of the best embedded software development tools

12.9.2.1 NetBeans

NetBeans is a free embedded software development tool that allows you to create apps using the most popular programming languages. This C ++ IDE has a distribution for all popular OS, and for all other platforms, you can build NetBeans on your own from the source.

Advantages:

- Free;
- Has cross-platform support;
- Large selection of plugins;
- Code completion, refactoring tools;
- A welcoming community of developers.

Disadvantages:

- Slow start-up;
- Problems with your cache when building ready-made programs;
- Needs JDK for installation.

12.9.2.2 PyCharm

Based on our team's experience, PyCharm is the best Python embedded software development tool with both free and paid options. PyCharm is available on all popular OS.

This program supports out-of-the-box Python development with the ability to directly run and debug any code. In addition, the IDE has project support and source control.

Advantages

- Support for everything and everyone
- A good community
- Init feature "out of the box".

Disadvantages:

PyCharm may load slowly, and the default settings may need to be tweaked for existing projects.

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

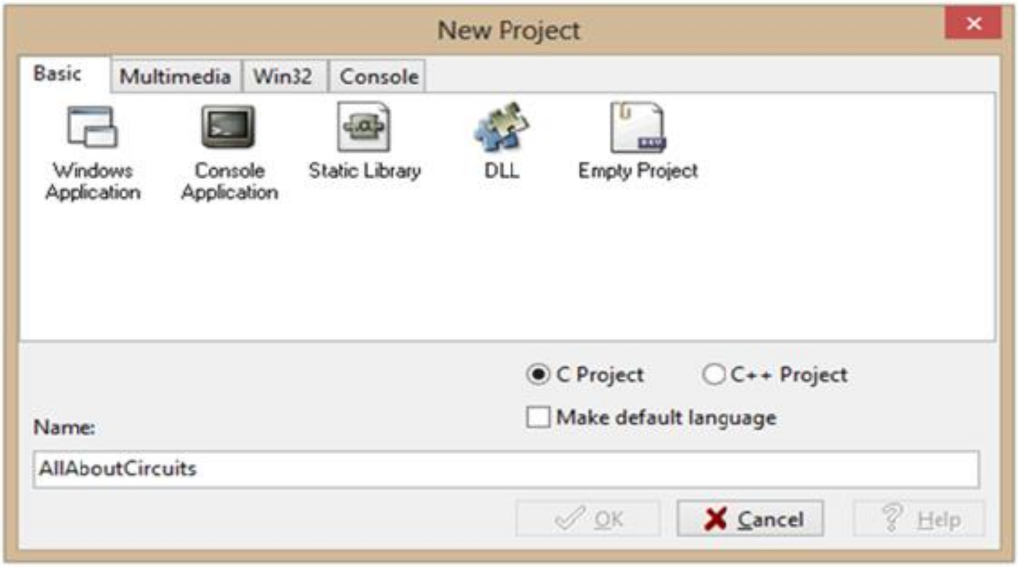
When not using an IDE, developers use an editor, compiler, and linker installed on their development machine to create code files, compile, and link them. Using the editor to create a source file, the code blocks, comments, and program statements are entered and the file saved. There are no "corrective 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.

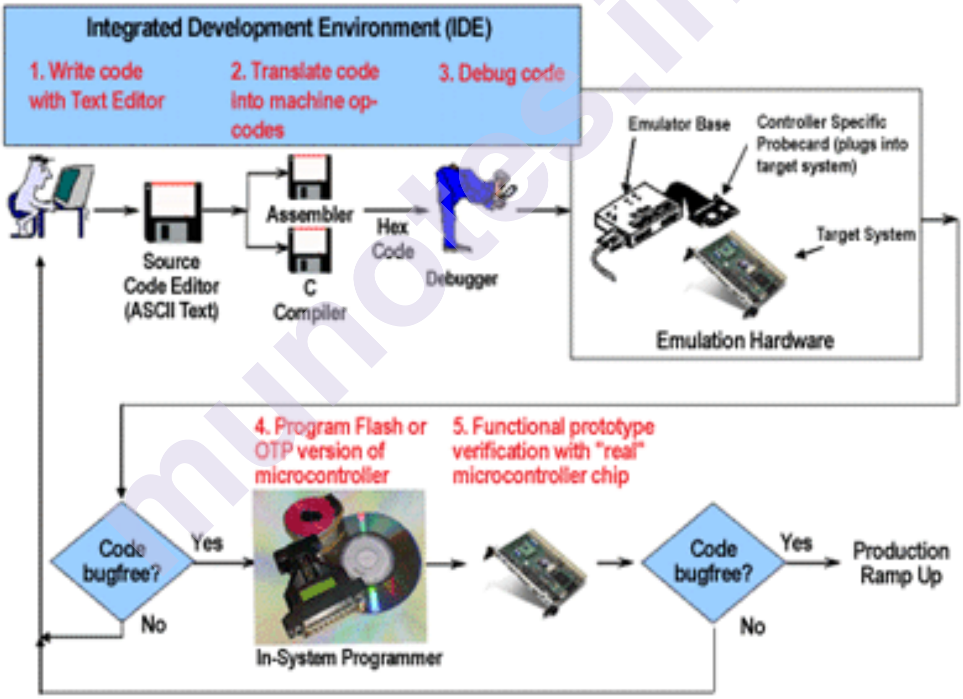
12.9.6 Choosing AN IDE

When selecting an IDE you'll find there are a lot to choose from and the price can vary from no cost to pricing options dependent on the environment and number of users. The license type is important as well if you intend to produce commercial code. Some things to check:

Does it provide support for you are?

- Development platform? (Some only run on specific operating systems.)
- Programming language(s)? (Some only support a specific language.)
- Target environment(s)? (IDEs targeting desktops may not support android environments.)

12.10 The Microcontroller Development Cycle


Integrated Development Environment (IDE)
1. Write code with Text Editor
2. Translate code into machine op-codes
3. Debug code
Source Code Editor (ASCII Text)
Assembler
C Compiler
Hex Code
Debugger
Emulator Base
Controller Specific Probecard (plugs into target system)
Target System
Emulation Hardware
4. Program Flash or OTP version of microcontroller
5. Functional prototype verification with "real" microcontroller chip
Code bugfree?
Yes
No
In-System Programmer
Code bugfree?
Yes
No
Production Ramp Up

- An Integrated Development Environment (IDE) is software that assists/supports programmers in developing software
- IDEs normally consist of a source code editor, a compiler, a linker/locater and usually a debugger.
- Sometimes, an IDE is devoted to one specific programming language or one (family of) specific processor or hardware, But more often the IDEs support multiple languages, processors, etc. Some commonly used IDEs for embedded systems are the GNU compiler collection (gcc), Eclipse, Delphi.

- **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:

An emulator is a hardware device or software program that enables one computer system (also known as a host) to imitate the functions of another computer system (known as the guest). It enables the host system to run software, tools, peripheral devices and other components which are designed for the guest system. Emulators can be of different types, replicating things such as hardware, software, OS or CPU.

Base Unit and Probe card: Many emulators consist of a base unit and a "probe card". The base unit is connected to a PC via the serial, parallel or USB port. It contains the majority of the emulator electronics, with the exception of the emulation chip itself. The emulation chip is a special bond-out version of the actual microcontroller and is mounted on a separate small PCB, called a probe card. This probe card connects via a ribbon cable to the base unit and has a pin adapter at the bottom, which allows the probe card to be plugged into a socket on the actual target application board in place of the actual microcontroller.

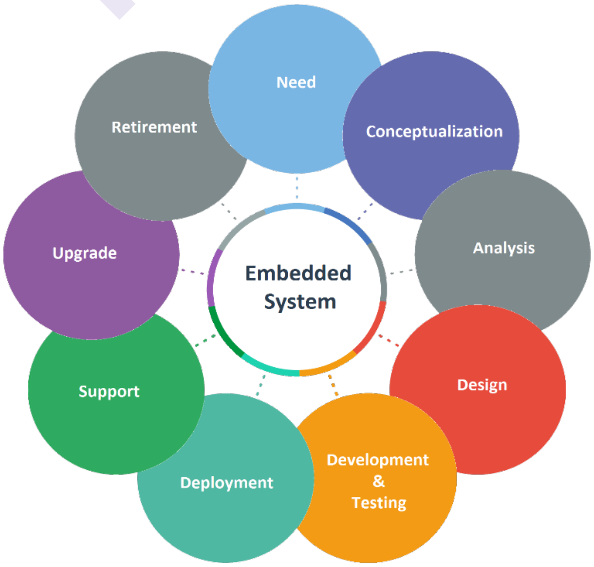
12.14 Debugger:

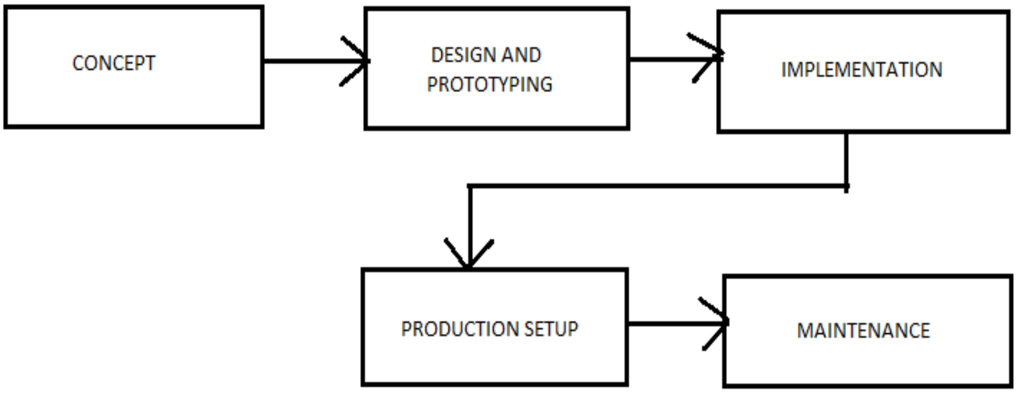
A debugger is a computer program that is used to test and debug other programs. A debugger is a piece of software running on the PC, which has to be tightly integrated with the emulator that you use to validate your code. A Debugger allows you to download your code to the emulator's memory and then control all of the functions of the emulator from a PC.

Debugging Tools:

When it comes to debugging your code and testing your application there are several different tools you can utilize that differ greatly in terms of development time spend and debugging features available. In this section we take a look at simulators and emulators.

12.15 Embedded Development Life-Cycle (EDLC)





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 machine control applications to various other new verticals with the convergence of communications. Various classes of embedded systems such as home media systems, portable players, smart phones, embedded medical devices and sensors, automotive embedded systems have surrounded us and with continued convergence of communications and computing functions within these devices, embedded systems are transforming themselves into really complex systems, thus creating newer opportunities and challenges to develop and market more powerful, energy efficient processors, peripherals and other accessories

In embedded system is more than the electronics as most people perceive it. It has electronics both digital and analog, special purpose sensors and actuators, software, mechanical items etc., and with design challenges of space, weight, cost and power consumption. In order to achieve key requirements, generally embedded systems are restricted to limited resources in terms of computing, memory, display size etc. With continued convergence of other technologies a lot more functionalities are being pushed into embedded devices which were once part of traditional computing platforms.

12.17 Summary

OS acts as Bridge between Application / Task & underlying system resources. Primary function of OS is to make system convenient to use, Organise & manage the system resources efficiently. Also other functions of OS are **Management** of Process, Memory, I/O, File system etc. So OS is called as Manager.

RTOS is an operating system intended to serve real time application that process data as it comes in, mostly without buffer delay. It offers priority-based scheduling, which allows you to separate analytical processing from non-critical processing. Important components of RTOS system are: 1) The Scheduler, 2) Symmetric Multiprocessing, 3) Function Library, 4) Memory Management, 5) Fast dispatch latency, and 6) User-defined data objects and classes.

Most of the commercial Embedded Operating systems available today are designed for possible inclusion in Real time system. Process of selecting Operating system for embedded system is outlined & reviewed. RTOS system occupy very less memory and consume fewer resources Performance is the most important factor required to be considered while selecting for a RTOS. General-Purpose Operating System (GPOS) is used for desktop PC and laptop while Real-Time Operating System (RTOS) only applied to the embedded application. Real-time systems are used in Airlines reservation system, Air traffic control system etc. The biggest drawback of RTOS is that the system concentrates on a few tasks.

12.18 Questions

1. Explain the embedded Operating system with respect to
   i) Tasks
   ii) Task States
   iii) Idle task
2. State & explain The Kernal in OS.
3. Explain the RTOS
4. Explain Scheduler useful in OS with respect to:
   i) Scheduling Points
   ii) Ready List
   iii) Context Switch
5. Write a short note on Trends in Embedded industry.
6. Give the Real Time Characteristic of embedded operating system.
7. What is the process behind selection of RTOS.
8. Write note on Embedded system Development Environment – IDE.
9. Write note on Simulator & Emulator
10. Explain the phases of EDLC.

12.19 References & Future Reading

1) Book Programming in ANSI C by E Balgurusamy.

2) Book *Embedded Systems Architecture, Programming & Design* (Second Edition / The McGraw Hill publications) by Raj kamal.

3) Introduction to *Embedded system* By Shibu K V (Tata McGraw Hill Publications)

4) Book *Embedded System* by Ashwini Somnathe & Abhijit Somnathe for SYBSc IT (Sheth Publications)

5) www.google.com

6) www.youtube.com

❄❄❄❄❄❄❄

Unit 5

13

EMBEDDED SYSTEMS: INTEGRATED DEVELOPMENT ENVIRONMENT

Unit Structure

13.0 OBJECTIVES

After reading this chapter you will understand:

- Embedded IDE
- Types of files involved
- Disassembler/ Decompiler
- Simulator
- Firmware Debugging and Emulator

13.1 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/ Decompiler, Simulator and then Firmware Debugging.

13.2 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.

13.3 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:

I. 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

II. 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

III. 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
```

IV. Assembly listing

- It contains the assembly 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
```

V. Symbol listing

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

VI. 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

VII. 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).

3. 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.

5. 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

2. **HEX FILE (.hex file)**

i. It is a binary executable file created from the source code.

ii. The file created by linker/locater is converted into processor understandable binary code.

iii. The tool used for converting and object file into a hex file is known as object to Hex converter.

iv. 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.

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

13.4 DISASSEMBLER/ DECOMPILER

- A **Disassembler/ Decompiler** 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.

13.5 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**

     I **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.

     II **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.

13.6 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'*

13.7 SUMMARRY

- Integrated Development Environment with respect to embedded system IDE stands for an Integrated Environment for developing and debugging the target processor specific embedded software.
- A **DISASSEMBLER** is a utility program which converts machine codes into target processor specific assembly code/instruction.
- Simulator simulates the target hardware, while the code execution can be inspected.

13.8 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

13.8 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

❄❄❄❄❄❄❄

Unit 5

14

EMBEDDED DEVELOPMENT LIFE CYCLE

Unit Structure

14.0 OBJECTIVES

After Reading this chapter you will understand

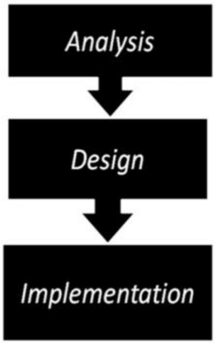
- The Embedded Development Life Cycle
- Phases Involved in the EDLC

14.1 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.

14.2 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.

14.2.1 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

14.2.2 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

14.3 DIFFERENT PHASES OF EDLC

The following figure depicts the different phases in EDLC:

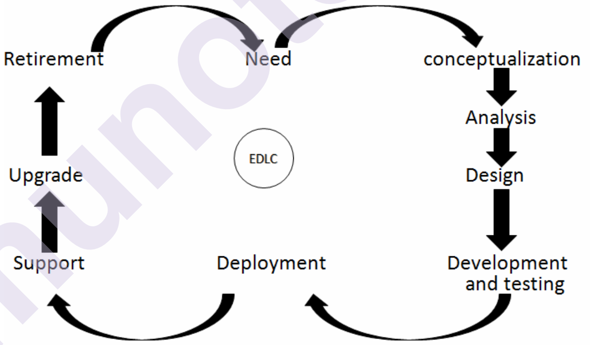


Figure : Phases of EDLC

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.

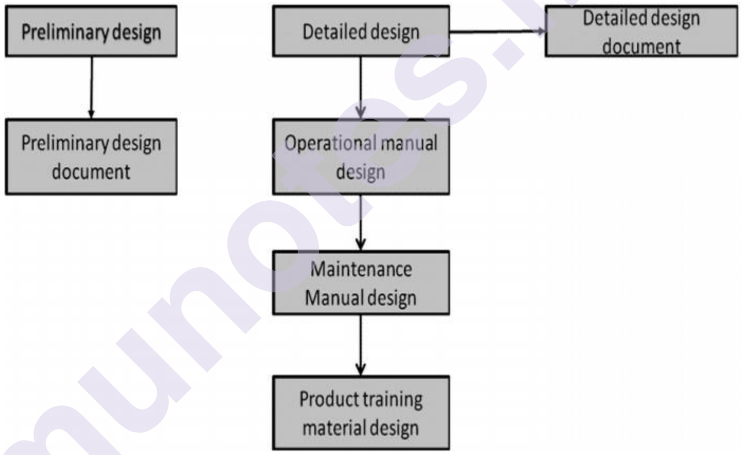
3. Analysis

- 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'.


Preliminary design
Preliminary design document
Detailed design
Detailed design document
Operational manual design
Maintenance Manual design
Product training material design

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

14.4 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

14.6 SUMMARRY

- Objectives of EDLC: i. Ensure that high quality products are delivered to user ii. Risk minimization defect prevention in product development through project management iii. Maximize the productivity
- Different phases of EDLC

14.5 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.

14.6 REFERENCES & FURTHER READING

Introduction to Embedded systems – Shibu K. V

❄❄❄❄❄❄❄

Unit 5

15

TRENDS IN EMBEDDED SYSTEMS

Unit 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.
    - o **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.

- o **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.
- o **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.
- o **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.
- o **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:
  - o Dual Core – 2 cores
  - o Tri Core – 3 cores
  - o 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.

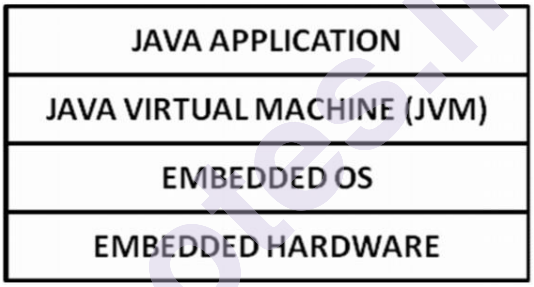


Figure: Java based Embedded Application Development

- 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:
  - 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.

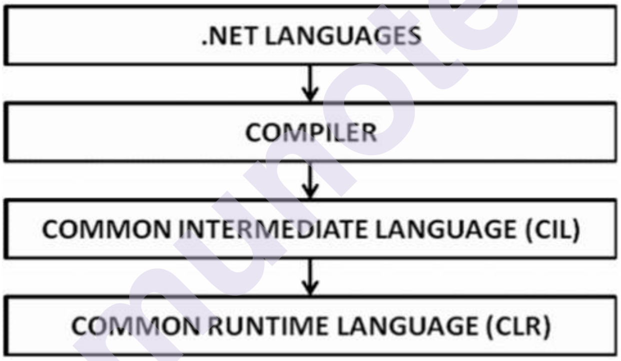

.NET LANGUAGES
COMPILER
COMMON INTERMEDIATE LANGUAGE (CIL)
COMMON RUNTIME LANGUAGE (CLR)

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

❄❄❄❄❄❄❄