Lecture Notes

Embedded Systems

G Padmanabha Sivakumar

Assistant Professor, EIE Department

EMBEDDED SYSTEMS
CHAPTER - 1
EMBEDDED COMPUTING

Pre-requisites:

Microprocessor and Microcontroller, Basics of C programming.

Aim:

To give an insight of Embedded Systems computing

Objectives:

The course will enable the students to:

1. Get introduced to features that build an embedded system.

2. Learn about the various components within an embedded system.

3. Learn the techniques of interfacing between processors & peripheral device related to embedded processing .

4. Do the efficient programs on any dedicated processor.

Outcome:

The students should be able to:

1. Understand Basic building blocks of embedded systems.

2. Interface various peripherals to processors.

3. Program embedded systems.

4. Use the basic concepts of systems programming like operating system, assembler compilers etc., and to understand the management task needed for developing embedded systems.

Pre-Test - MCQ Type

1. An embedded system must have

   (a) hard disk
   (b) processor and memory
   (c) operating system
   (d) processor and input-output unit(s).

2. An embedded system hardware can

   (a) have microprocessor or microcontroller or single purpose processor
   (b) have digital signal processor
   (c) one or several microprocessor or microcontroller or digital signal processor or single purpose processors
   (d) not have a single purpose processor (s).

3. An embedded system has RAM memory

   (a) for storing the variables during program run, stack and input or output buffers, for example, for speech or image
   (b) for storing all the instructions and data
   (c) for storing the programs from external secondary memory
   (d) for fetching instructions and data into cache(s).

4. 8051 microcontrollers are manufactured by which of the following companies?

   a) Atmel
   b) Philips
   c) Intel

d) All of the mentioned

5. If we push data onto the stack then the stack pointer

a) increases with every push
b) decreases with every push
c) increases & decreases with every push
d) none of the mentioned

6. Embedded Systems communicate with the outside world through

a) Peripherals
b) Processors
c) OS
d) Microcontrollers

7. Which is the most commonly used language(s) used in embedded system?

a) C
b) java
c) COBOL
d) Both a and c

8. ________________ provides a simulation of all aspects of the hardware

a) Emulator
b) ICD
c) ICE
d) HLL

9. What is an In-circuit emulator?

a) It replaces the microprocessor with a simulated equivalent
b) It is a hardware device that connects to the microprocessor
c) It is an embedded application with real time operating system
d) A tool that allows external debugging

10. Embedded middleware sits between

a) embedded application and operating system
b) kernel and the real time operating system
c) embedded application and the real time operating system
d) kernel and the real time operating system

11. 8051 series has how many 16 bit registers?

a) 2
b) 3
c) 1
d) 0

12. Which microcontrollers are adopted for designing medium scale embedded systems?

a. 8-bit
b. 16-bit to 32-bit
c. 64-bit
d. All of the above

13. Which types of an embedded systems involve the coding at a simple level in an embedded 'C', without any necessity of RTOS?

a. Small Scale Embedded Systems
b. Medium Scale Embedded Systems
c. Sophisticated Embedded Systems
d. All of the above

14. Which stage associated with pipelining mechanism recognizes the instruction that is to be executed?

a. Fetch
b. Decode
c. Execute
d. None of the above

15. In CPU structure, where is one of the operand provided by an accumulator in order to store the result?

a. Control Unit
b. Arithmetic Logic Unit
c. Memory Unit
d. Output Unit

1.1 INTRODUCTION TO EMBEDDED SYSTEMS

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.

1.1.1 DEFINITION OF AN EMBEDDED SYSTEM

- An embedded system is a combination of 3things:

    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.

1.2 HISTORY OF EMBEDDED SYSTEM

- The first recognised embedded system is the Apollo Guidance Computer(AGC) developed by MIT lab.
- AGC was designed on 4K words of ROM & 256 words ofRAM. 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 as DSKY(display/keyboard) which resembles a calculator type keypad with an 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.

1.3 CLASSIFICATION OF EMBEDDED SYSTEM

- The classification of embedded system is based on following criteria's:
    1. On generation
    2. On complexity & performance
    3. On deterministic behaviour
    4. On triggering

1.3.1 On generation

1.3.1.1. First generation(1G):

- ❖ Built around an 8bit microprocessor & microcontroller.
- ❖ Simple in hardware circuit & firmware developed.
- ❖ Examples: Digital telephone keypads.

1.3.1.2. Second generation(2G):

- ❖ Built around 16-bit μp & 8-bit μc.
- ❖ They are more complex & powerful than 1G μp &μc.
- ❖ Examples: SCADA systems

1.3.1.3. 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.

1.3.1.4. 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.

1.3.2 On complexity & performance

1.3.2.1. Small-scale:

- ➔ Simple applications need Performance not time-critical.
- ➔ Built around low performance & low cost 8 or 16 bit µp/µc.
- ➔ Example: an electronic toy

1.3.2.2. Medium-scale:

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

1.3.2.3. 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.

1.3.3 On deterministic behavior

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

1.3.4 On triggering

- ❖ Embedded systems which are ―Reactive‖ in nature can be based on triggering.
- ❖ Reactive systems can be:
  1. Event triggered
  2. Time triggered

1.4 TYPES OF EMBEDDED SYSTEMS

- Depending on the classification embedded systems are of these types;
  1. Stand alone embedded systems
  2. Real time embedded system
  3. Networked embedded system
  4. Mobile embedded system

1.4.1 Stand alone Embedded systems:

- A stand-alone embedded system works by itself. It is a self-contained device which does not require any host system like a computer. It takes either digital or analog inputs from its input ports, calibrates, converts, and processes the data, and outputs the resulting data to its attached output device, which either displays data, or controls and drives the attached devices.
- **EX:** Temperature measurement systems, Video game consoles, MP3 players, digital cameras, and microwave ovens are the examples for this category.

1.4.2Real-time embedded systems:

- An embedded system which gives the required output in a specified time or which strictly follows the time deadlines for completion of a task is known as a Real time system. i.e. a Real Time system , in addition to functional correctness, also satisfies the time constraints .
- There are two types of Real time systems.
  (i) Soft real time system and
  (ii) Hard real time system.

1.4.2.1 Soft Real-Time system:

- A Real time system in which the violation of time constraints will cause only degraded quality, but the system can continue to operate is known as a Soft real time system. In soft real-time systems, the design focus is to offer a guaranteed bandwidth to each real time task and to distribute the resources to the tasks.
- **Ex:** A Microwave Oven, washing machine, TV remote etc.

1.4.2.2 Hard Real-Time system:

- A Real time system in which the violation of time constraints will cause critical failure and loss of life or property damage or catastrophe is known as a Hard Real time system.
- These systems usually interact directly with physical hardware instead of through a human being .The hardware and software of hard real-time systems must allow a worst case execution (WCET) analysis that guarantees the execution will be completed within a strict deadline. The chip selection and RTOS selection become important factors for hard real-time system design.
- Ex: Deadline in a missile control embedded system , Delayed alarm during a Gas leakage , car airbag control system , A delayed response in pacemakers ,Failure in RADAR functioning etc.

1.4.3 Networked embedded systems:

- The networked embedded systems are related to a network with network interfaces to access the resources. The connected network can be a Local Area Network (LAN) or a Wide Area Network (WAN), or the Internet. The connection can be either wired or wireless.

- The networked embedded system is the fastest growing area in embedded systems applications. The embedded web server is such a system where all embedded devices are connected to a web server and can be accessed and controlled by any web browser.
- **Ex:** A home security system is an example of a LAN networked embedded system where all sensors (e.g. motion detectors, light sensors, or smoke sensors) are wired and running on the TCP/IP protocol.

1.4.4.Mobile Embedded systems:

The portable embedded devices like mobile and cellular phones, digital cameras, MP3 players, PDA (Personal Digital Assistants) are the example for mobile embedded systems. The basic limitation of these devices is the limitation of memory and other resources.

1.5 APPLICATION OF EMBEDDED SYSTEM

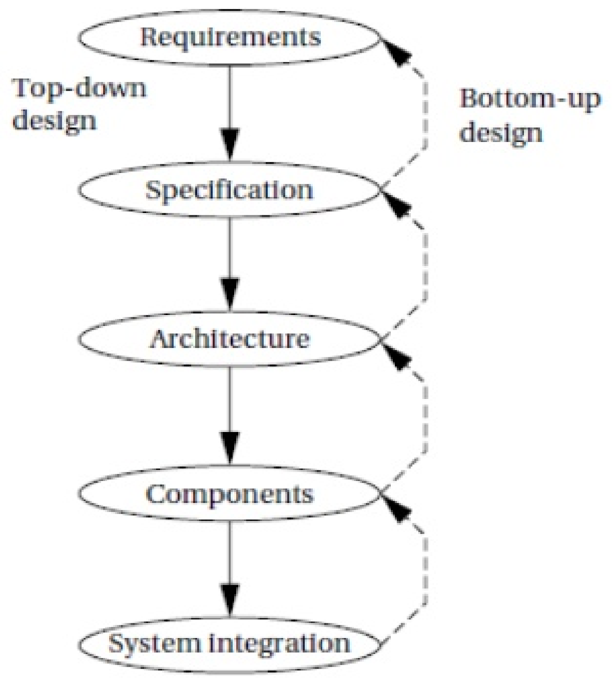
The application areas and the products in the embedded domain are countless.

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.

1.6 THE EMBEDDED SYSTEM DESIGN PROCESS

- Overview of the embedded system design process aimed at two objectives.
- First,it will give us an introduction to the various steps in embedded system design before we delve into them in more detail.
- Second, it will allow us to consider the design ***methodology*** itself. This figure summarizes the major steps in the embedded system design process.



- In this top–down view,we start with the system ***requirements*** in the next step comes ***Specification***.

1.6.1 SPECIFICATIONS

- we create a more detailed description of what we want. But the specification states only how the system behaves, not how it is built.
- The details of the system‘s internals begin to take shape when we develop the architecture, which gives the system structure in terms of large components.
- Once we know the components we need, we can design those components, including both software modules and any specialized hardware we need.
- Based on those components, we can finally build a complete system.
- ***Top–down Design***—we will begin with the most abstract description of the system and conclude with concrete details.
- The alternative is a **bottom–up** view in which we start with components to build a system. Bottom–up design steps are shown in the figure as dashed-line arrows.
- We need bottom–up design because we do not have perfect insight into how later stages of the design process will turn out.
- During the design process we have to consider the major goals of the design such as

01.manufacturing cost;

02. performance (both overall speed and deadlines); and

03. power consumption.

1.6.2 STEPS INVOLVED IN DESIGN ARE AS FOLLOWS:-

1.6.2.1 REQUIREMENTS

Requirements may be ***functional*** or ***nonfunctional***. We must capture the basic functions of the embedded system, but functional description is often not sufficient. Typical nonfunctional requirements include:

1. ***Performance**:*

- The speed of the system is often a major consideration both for the usability of the system and for its ultimate cost. Performance may be a combination of soft performance metrics such as approximate time to perform a user-level function and hard deadlines by which a particular operation must be completed.

2. ***Cost***:

The target cost or purchase price for the system is almost always a consideration. Cost typically has two major components:

Manufacturing cost includes the cost of components and assembly;

NonRecurring engineering (NRE) costs include the personnel and other costs of designing the system.

3. Physical size and weight:

- The physical aspects of the final system can vary greatly depending upon the application. e.g) An industrial control system for an assembly line may be designed to fit into a standard-size rack with no strict limitations on weight. But a handheld device typically has tight requirements on both size and weight that can ripple through the entire system design.

4. Power consumption:

- Power, of course, is important in battery-powered systems and is often important in other applications as well. Power can be specified in the requirements stage in terms of battery life.

5. Mock-up.

- The mock-up may use canned data to simulate functionality in a restricted demonstration, and it may be executed on a PC or a workstation. But it should give the customer a good idea of how the system will be used and how the user can react to it. Physical,nonfunctional models of devices can also give customers a better idea of characteristics such as size and weight.

Example:

Requirements analysis of a GPS moving map

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

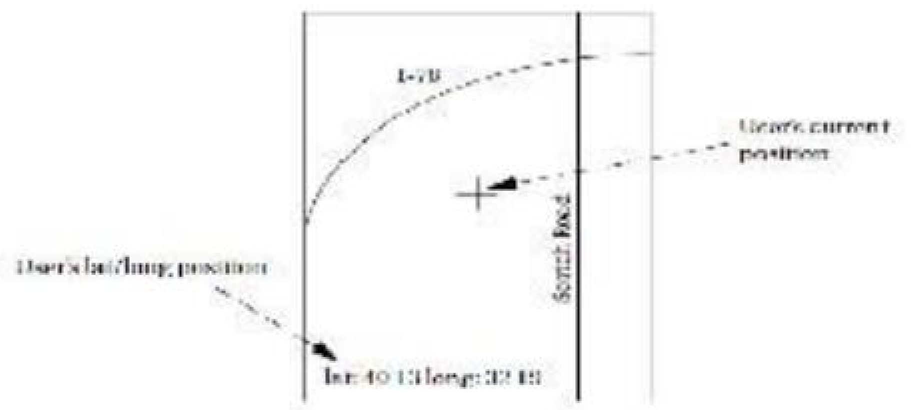


Fig.1.3.1.1 G.P.S Moving map

REQUIREMENTS

1. *User interface:*

The screen should have at least 400_600 pixel resolution. The device should be controlled by no more than three buttons.

2. *Performance:*

The map should scroll smoothly. Upon power-up, a display should take no more than one second to appear, and the system should be able to verify its position and display the current map within 15 s.

3. *Cost:*

The selling cost (street price) of the unit should be no more than $100.

4. *Physical size and weight:*

The device should fit comfortably in the palm of the hand.

5. *Power consumption:*

The device should run for at least eight hours on four AA batteries.

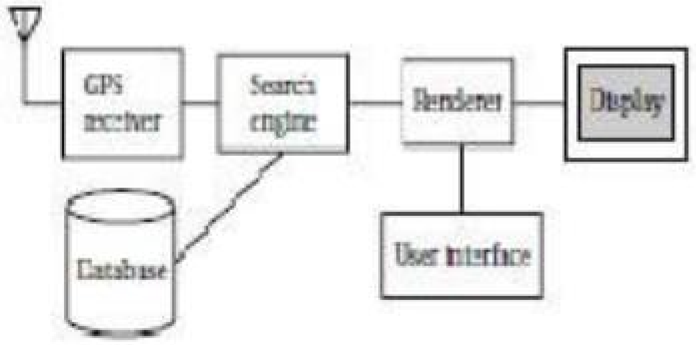
Specification

- The specification is more precise—it serves as the contract between the customer and the architects.
- The specification should be understandable enough so that someone can verify that it meets system requirements and overall expectations of the customer. It should also be unambiguous.
- The specification must be carefully written so that it accurately reflects the customer's requirements and does so in a way that can be clearly followed during design.
- A specification of the GPS system would include several components:
- Data received from the GPS satellite constellation.

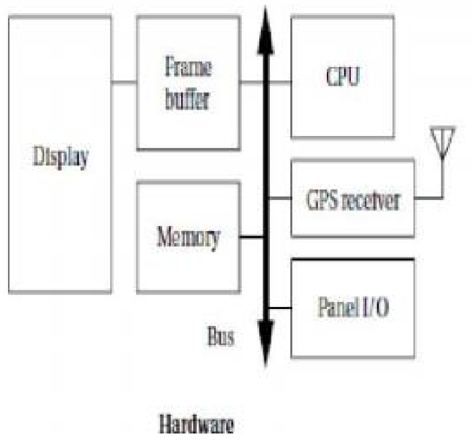
- Map Data
- User Interface
- Operations that must be performed to satisfy customer requests. Background actions required to keep the system running, such as operating the GPS receiver. UML, a language for describing specifications

Architecture Design

1. The specification does not say how the system does things, only what the system does. Describing how the system implements those functions is the purpose of the architecture. Figure 1.3 shows a sample system architecture in the form of a block diagram that shows major operations and data flows among them.



2. Many implementation details should we refine that system block diagram into two block diagrams: one for hardware and another for software. These two more refined block diagrams are shown in Figure 1.4.The hardware block diagram clearly shows that we have one central CPU surrounded by memory and I/O devices. In particular, we have chosen to use two memories: a frame buffer for the pixels to be displayed and a separate program/data memory for general use by the CPU .



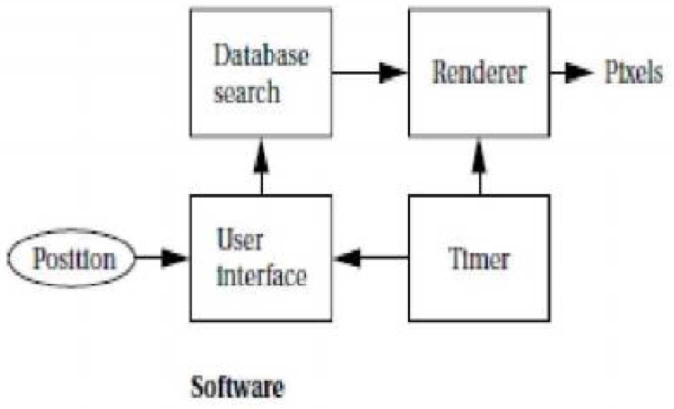


Fig 1.3.3.2 Hardware design Fig 1.3.3.3 Software design

3.

Designing Hardware and Software Components

- The component design effort builds those components in conformance to the architecture and specification. The components will in general include both hardware—FPGAs, boards, and so on—and software modules. Some of the components will be ready-made. In the moving map, the GPS receiver is a good example of a specialized component that will nonetheless be a predesigned, standard component. We can also make use of standard software modules. One good example is the topographic database.

System Integration

The components built are put together and see how the system works. If we debug only a few modules at a time, we are more likely to uncover the simple bugs and able to easily recognize them. Only by fixing the simple bugs early will we be able to uncover the more complex or obscure bugs.

1.7 Embedded processors – 8051 Microcontroller

- The Intel 8051 microcontroller is one of the most popular general purpose microcontrollers in use today. The success of the Intel 8051 spawned a number of clones, which are collectively referred to as the MCS-51 family of microcontrollers, which includes chips from vendors such as Atmel, Philips, Infineon, and Texas Instruments.
- The Intel 8051 is an 8-bit microcontroller which means that most available operations are limited to 8 bits. There are 3 basic "sizes" of the 8051: Short, Standard, and Extended. The Short and Standard chips are often available in DIP (dual in-line package) form, but the Extended 8051 models often have a different form factor, and are not "drop-in compatible".
- All these things are called 8051 because they can all be programmed using 8051 assembly language, and they all share certain features (although the different models all have their own special features).

1.7.1 FEATURES OF 8051 MICRO CONTROLLER

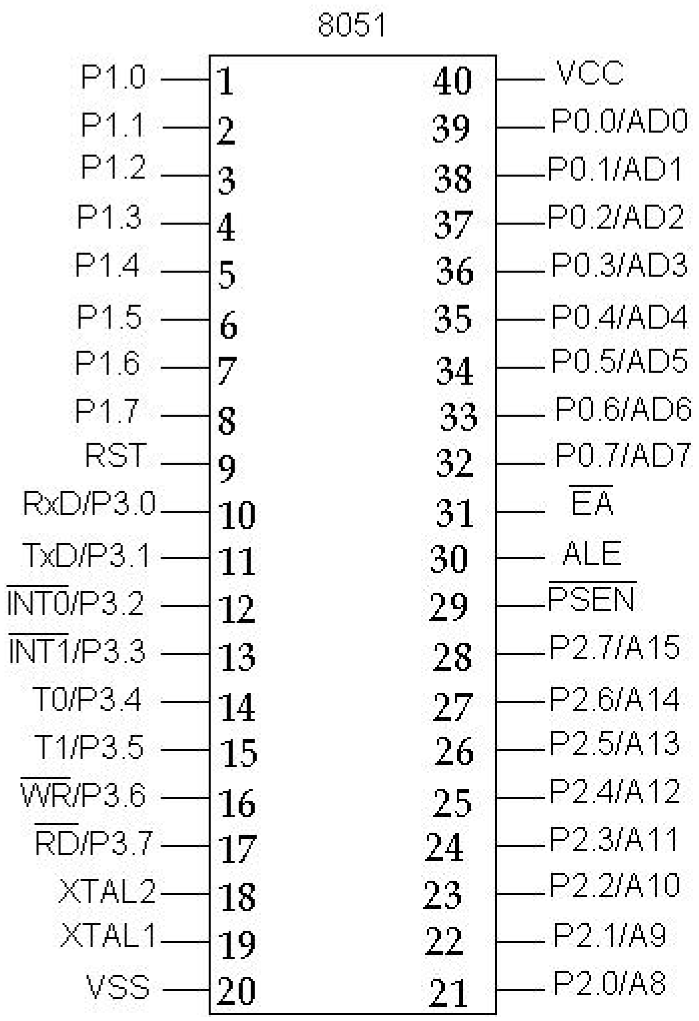
- 4 KB on chip program memory.
- 128 bytes on chip data memory(RAM)
- 32 bytes devoted to register banks
- 16 bytes of bit-addressable memory
- 80 bytes of general-purpose memory
- 4 reg banks.
- 128 user defined software flags.
- 8-bit data bus
- 16-bit address bus
- 16 bit timers (usually 2, but may have more, or less).
- 3 internal and 2 external interrupts.
- Bit as well as byte addressable RAM area of 16 bytes.
- Four 8-bit ports, (short models have two 8-bit ports).
- 16-bit program counter and data pointer.
- 1 Microsecond instruction cycle with 12 MHz Crystal.

1.7.2 BASIC PIN SETUP

PIN 9: PIN 9 is the reset pin which is used to reset the microcontroller's internal registers and ports upon starting up. (Pin should be held high for 2 machine cycles.)

PINS 18 & 19: The 8051 has a built-in oscillator amplifier hence we need to only connect a crystal at these pins to provide clock pulses to the circuit.

PIN 40 and 20: Pins 40 and 20 are VCC and ground respectively. The 8051 chip needs +5V 500mA to function properly, although there are lower powered versions like the Atmel 2051 which is a scaled down version of the 8051 which runs on +3V.



PINS 29, 30 & 31: As described in the features of the 8051, this chip contains a built-in flash memory. In order to program this we need to supply a voltage of +12V at pin 31. If external memory is connected then PIN 31, also called EA/VPP, should be connected to ground to indicate the presence of external memory. PIN 30 is called ALE (address latch enable), which is used when multiple memory chips are connected to the controller and only one of them needs to be selected.We will deal with this in depth in the later chapters. PIN 29 is called PSEN. This is "program store enable". In order to use the external memory it is required to provide the low voltage (0) on both PSEN and EA pins.

Pin 29: If we use an external ROM then it should have a logic 0 which indicates Microcontroller to read data from memory.

Pin 30: This Pin is used for ALE that is Address Latch Enable. If we use multiple memory chips then this pin is used to distinguish between them.It is activated periodically with a constant rate of 1/6th of oscillator frequency. This Pin also gives program pulse input during programming of EPROM.

Pin 31: If we have to use multiple memories then by applying logic 1 to this pin instructs Microcontroller to read data from both memories first internal and afterwards external.

1.7.3 PORTS

There are 4 8-bit ports: P0, P1, P2 and P3.

PORT P1 (Pins 1 to 8): The port P1 is a general purpose input/output port which can be used for a variety of interfacing tasks. The other ports P0, P2 and P3 have dual roles or additional functions associated with them based upon the context of their usage.The port 1 output buffers can sink/source four TTL inputs. When 1s are written to port 1 pins are pulled high by the internal pull-ups and can be used as inputs.

PORT P3 (Pins 10 to 17): PORT P3 acts as a normal IO port, but Port P3 has additional functions such as, serial transmit and receive pins, 2 external interrupt pins, 2 external counter inputs, read and write pins for memory access.

PORT P2 (pins 21 to 28): PORT P2 can also be used as a general purpose 8 bit port when no external memory is present, but if external memory access is required then PORT P2 will act as an address bus in conjunction with PORT P0 to access external memory. PORT P2 acts as A8-A15, as can be seen from fig 1.1

PORT P0 (pins 32 to 39) PORT P0 can be used as a general purpose 8 bit port when no external memory is present, but if external memory access is required then PORT P0 acts as a multiplexed address and data bus that can be used to access external memory in conjunction with PORT P2. P0 acts as AD0-AD7, as can be seen from fig 1.1

PORT P10: asynchronous communication input or Serial synchronous communication output.

PIN 11: Serial Asynchronous Communication Output or Serial Synchronous Communication clock Output.

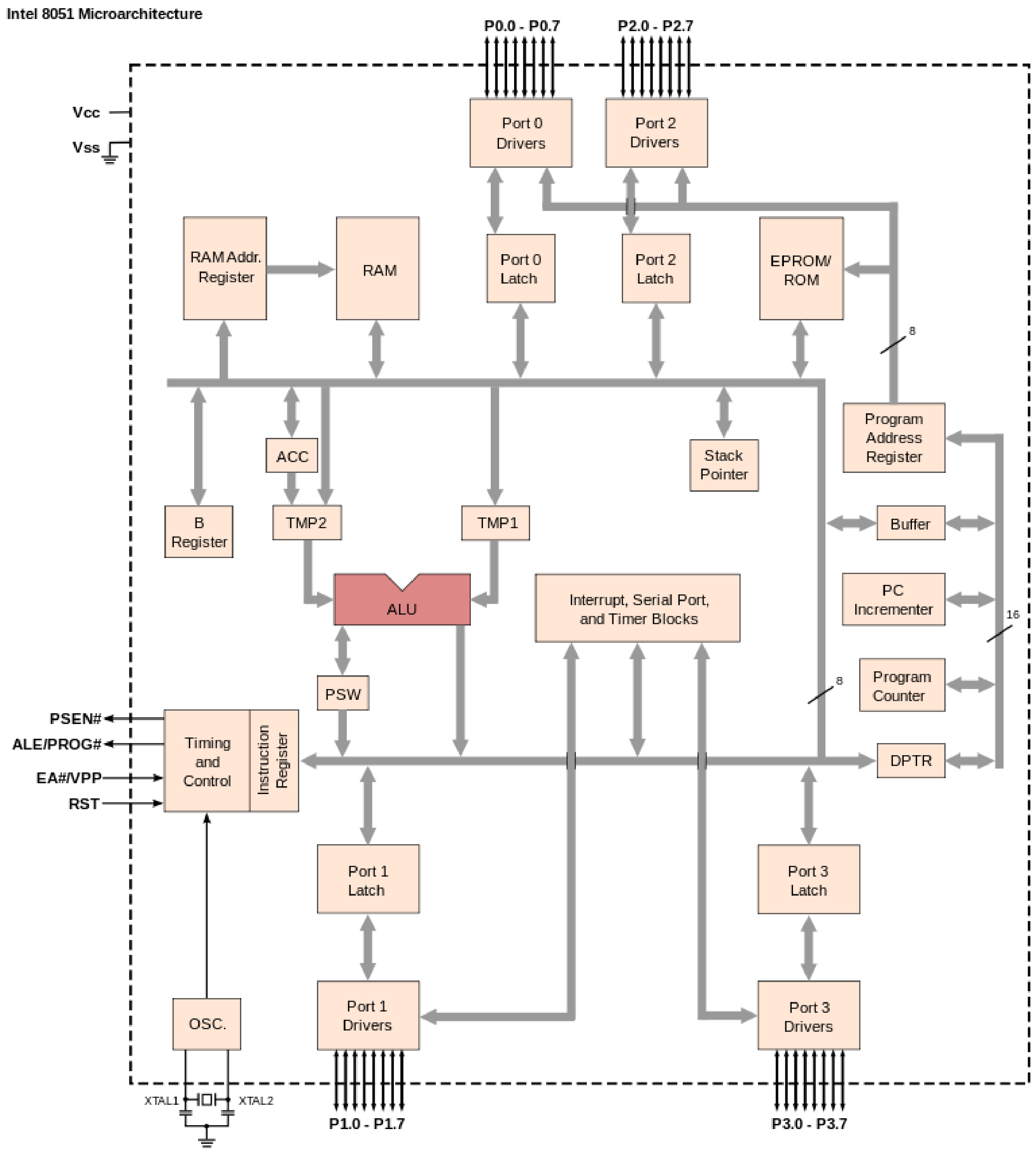
1.7.3 OSCILLATOR CIRCUIT

- The 8051 requires an external oscillator circuit. The oscillator circuit usually runs around 12 MHz, although the 8051 (depending on which specific model) is capable of running at a maximum of 40 MHz. Each machine cycle in the 8051 is 12 clock cycles, giving an effective cycle rate at 1 MHz (for a 12 MHz clock) to 3.33 MHz (for the maximum 40 MHz clock). The oscillator circuit generates the clock pulses so that all internal operations are synchronized.

1.7.4 DATA AND PROGRAM MEMORY

- The 8051 Microcontroller can be programmed in PL/M, 8051 Assembly, C and a number of other high-level languages. Some compilers even have support for compiling C++ for an 8051.
- Program memory in the 8051 is read-only, while the data memory is considered to be read/write accessible. When stored on EEPROM or Flash, the program memory can be rewritten when the microcontroller is in the special programmer circuit or, if not using a 8031, through a preinstalled bootloader.
- The 8051 starts executing program instructions from address 0000 in the program memory.

1.7.5 ARCHITECTURE

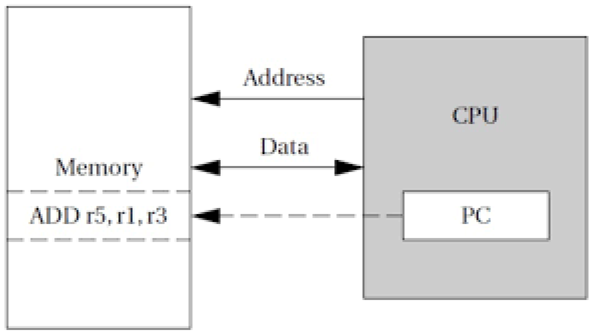


1.8 INTRODUCTION TO ARCHITECTURES:

- Two Computing architectures are available:
- ❏ 1. von Neumann architecture computer
- ❏ 2. Harvard architecture

1.8.1 von Neumann architecture computer:

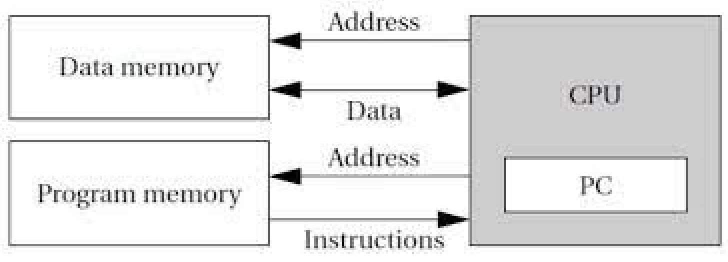
- The memory holds both data and instructions, and can be read or written when given an address.
- A computer whose memory holds both data and instructions is known as a von Neumann machine.
- The CPU has several internal registers that store values used internally. One of those registers is the program counter (PC) ,which holds the address in memory of an instruction.
- The CPU fetches the instruction from memory,decodes the instruction, and executes it. The program counter does not directly determine what the machine does next,but only indirectly by pointing to an instruction in memory.



A von Neumann architecture computer.

1.8.2 Harvard architecture:

- Harvard machine has separate memories for data and program.
- The program counter points to program memory, not data memory.
- As a result, it is harder to write self-modifying programs (programs that write data values, then use those values as instructions) on Harvard machines.



A Harvard architecture.

- Advantage: The separation of program and data memories provides higher performance for digital signal processing.

1.8.3Differences between Von neumann and harvard architecture:

Von-Neumann Architecture	**Harvard Architecture**
1. Under pure von Neumann architecture the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same bus system.	1. In a computer using the Harvard architecture, the CPU can both read an instruction and perform a data memory access at the same time, even without a cache.
2. Von-Neumann architecture is much slower as it has a single communication pathway.	2. A Harvard architecture computer is more faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.
3. A Von-Neumann architecture machine does not have distinct code and data address spaces.	3. A Harvard architecture machine has distinct code and data address spaces.
4. It is not possible to have two separate memory systems for a Von-Neumann architecture.	4. It is possible to have two separate memory systems for a Harvard architecture.
5. Von Neumann architectures usually have a single unified cache, which stores both instructions and data.	5. Harvard architecture usually have a multiple cache, which stores both instructions and data separately.
6. It needs external memory all the time.	6. It may not need external memory at all.

CISC	RISC
The original microprocessor ISA	Redesigned ISA that emerged in the early 1980s
Instructions can take several clock cycles	Single-cycle instructions
Hardware-centric design – the ISA does as much as possible using hardware circuitry	Software-centric design – High-level compilers take on most of the burden of coding many software steps from the programmer
More efficient use of RAM than RISC	Heavy use of RAM (can cause bottlenecks if RAM is limited)
Complex and variable length instructions	Simple, standardized instructions
May support microcode (micro-programming where instructions are treated like small programs)	Only one layer of instructions
Large number of instructions	Small number of fixed-length instructions
Compound addressing modes	Limited addressing modes

1.9 ARM PROCESSOR

- ARM(Advanced RISC Machine) Processor: ARM uses RISC architecture
- ARM uses assembly language for writing programs
- ARM instructions are written one per line, starting after the first column. Comments begin with a semicolon and continue to the end of the line.

- A label, which gives a name to a memory location, comes at the beginning of the line, starting in the first column.
- Here is an example:
- ➢ LDR r0,[r8];
- ➢ a comment label ADD r4,r0,r1

1.9.1 Memory Organization in ARM Processor:

- ➔ The ARM architecture supports two basic types of data:
- ➔ The standard ARM word is 32 bits long.
- ➔ The word may be divided into four 8-bit byte ARM allows addresses up to 32 bits long .
- ➔ The ARM processor can be configured at power-up to address the bytes in a word in either little-endian mode (with the lowest-order byte residing in the low-order bits of the word) or big-endian mode,

1.9.2 Data Operations in ARM:

- In the ARM processor, arithmetic and logical operations cannot be performed directlyon memory locations.
- ARM is a load-store architecture—data operands must first be loaded into the CPU and then stored back to main memory to save the results.

1.9.3 ARM Programming Model:

1. Programming model gives information about various registers supported by ARM
2. ARM has 16 general-purpose registers, r0 to r15
3. . Except for r15, they are identical—any operation that can be done on one of them can be done on the other one also
4. r15 register is also used as program counter(PC)
5. current program status register (CPSR):

- ➢ This register is set automatically during every arithmetic, logical, or shifting operation.
- ➢ The top four bits of the CPSR hold the following useful information about the results of that arithmetic/logical operation:
  - ❏ The negative (N) bit is set when the result is negative in two_scomplement arithmetic.
  - ❏ The zero (Z) bit is set when every bit of the result is zero.
  - ❏ The carry (C) bit is set when there is a carry out of the operation.
  - ❏ The overflow (V) bit is set when an arithmetic operation results in an overflow.

1.9.4 Types of Instructions supported by ARM Processor:

1. Arithmetic Instructions
2. Logical Instructions
3. shift / rotate Instructions
4. Comparison Instructions
5. move instructions
6. Load store instructions

INSTRUCTIONS EXAMPLE:-

ADD r0,r1,r2

This instruction sets register r0 to the sum of the values stored in rl and r2. ADD r0,r1,#2 (immediate operand are allowed during addition)

RSB r0, rl, r2 sets r0 to be r2-rl.

bit clear: BIC r0, r1, r2 sets r0 to rl and not r2.

Multiplication:

- no immediate operand is allowed in multiplication
- two source operands must be different registers

MLA:

- The MLA instruction performs a multiply-accumulate operation, particularly useful in matrix operations and signal processing

MLA r0,rl,r2,r3 sets ro to the value rlx r2+r3.

Shift operations:

1. Logical shift(LSL, LSR)
2. Arithmetic shifts (ASL, ASR)

➢ A left shift moves bits up toward the most-significant bits,

➢ the right shift moves bits down to the least-significant bit in the word.

➢ The LSL and LSR modifiers perform left and right logical shifts, filling the least significant bits of the operand with zeroes.

➢ The arithmetic shift left is equivalent to an LSL, but the ASR copies the sign bit-if the sign is 0, a 0 is copied, while if the sign is 1, a 1 is copied.

Rotate operations: (ROR, RRX)

❏ The rotate modifiers always rotate right, moving the bits that fall off the least-significant bit up to the most-significant bit in the word.

❏ The RRX modifier performs a 33-bit rotate, with the CPSR's C bit being inserted above the sign bit of the word; this allows the carry bit to be included in the rotation

Compare instructions: (CMP, CMN)

❏ compare instruction modifies flags values (Negative flag, zero flag, carry flag, Overflow flag)

❏ CMP r0, rl computes r0 - r1, sets the status bits, and throws away the result of the subtraction.

❏ CMN uses an addition to set the status bits.

❏ TST performs a bit-wise AND on the operands,

❏ while TEQ performs an exclusive-or

Load store instructions:

- ❏ ARM uses register-indirect addressing
- ❏ The value stored in the register is used as the address to be fetched from memory; the result of that fetch is the desired operand value.
- ❏ LDR r0.[rl] sets r0 to the value of memory location Ox100.
- ❏ Similarly, STR 10,[rl] would store the contents of ro in the memory location whose address is given in rl

LDR r0,[rl. -r2]

ARM Register indirect addressing:

LDR r0,[r1, #4] loads r0 from the address 1+4.

ARM Base plus offset addressing mode:

- ➢ The register value is added to another value to form the address.
- ➢ For instance, LDR r0,[r1,#16] loads r0 with the value stored at location rl+16.(ri-base address, 16 is offset)
- ➢ Auto-indexing updates the base register, such that LDR r0,[rl,#16]!--first adds 16 to the value of rl, and then uses that new value as the address. The ! operator causes the base register to be updated with the computed address so that it can be used again later.
- ➢ Post-indexing does not perform the offset calculation until after the fetch has been performed Consequently
- ➢ LDR r0,fr1],#
- ➢ 16 will load ro with the value stored at the memory location whose address is given by rl, and then add 16 to rl and set it to the new value.

POST-TEST- MCQ TYPE

1. (i) A compiler generates an object file.
   (ii) The object file is linked with library functions using linker.
   (iii) After re-allocation of addresses a locator sends the codes to device programmer for burning as ROM image in the embedded system ROM.
   (iv) After re-allocation of addresses a loader loads the codes to device programmer for burning as ROM image in embedded system ROM.
   (v) After re-allocation of addresses a loader loads the codes in RAM.
   Steps for embedded system development are steps

   (a) i, ii and iv
   (b) i, iii, iv and vi
   (c) Steps i, ii and iii
   (d) i , ii and vi.

2. In a multitasking OS,

   (i) each process (task) has a distinct process control block
   (ii) each process (task) has memory allocation of its own
   (iii) a task has one or more functions or procedures for a specific job.
   (iv) a task may share the memory (data) with other tasks. processor may process multiple tasks separately or concurrently
   (v) each process (task) has a separate stack in memory
   (vi) a process calls another process, which can call another process, similar to nested call of the functions.

   (a) i, ii, iv and vi correct
   (b) all are correct except vi
   (c) iii, iv and v correct
   (d) ii, iii and vi correct.

3. RTOS is used in most embedded systems when the system does ________
   (a) concurrent processing of multiple real time processes
   (b) sequential processing of multiple processes when the tasks have real time constraints
   (c) real time processing of multiple processes
   (d) the concurrent processing of multiple processes, tasks have real time constraints and deadlines, and high priority task preempts low priority task as per the real time constraints.

4. A device driver is software for ________

(a) opening or connecting or binding or reading or writing or closing or other actions of the device
(b receiving input or sending outputs from device
(c) access to parallel or serial port by the device
(d) controlling and configuring the device for read and write functions.

5. Device Manager is software________
(i) for allocating and registering port (in fact, it may be a register or memory) addresses for the various devices at distinctly different addresses
(ii) codes for detecting the presence of devices
(iii) for initializing these and for testing the devices that are present
(iv) codes for detecting any collision between the device accesses in the system
(v) managing driver functions of all physical and virtual devices.

(a) v correct
(b) i, iv and v correct
(c) all correct
(d) all correct except iv.

6. (i) Model/analyze requirements and specifications of system
(ii) Design data structure, software architecture, interfaces and algorithms
(iii) Test the design
(iv) Implementation of design
(v) component level design
(vi) Test all internal logic and external functions of the system and
(vii) system integration

Activities during software design   cycle are in sequence of
(a) i to vii
(b) i, ii, v, iv, vii and vi
(c) i, ii, v, iv, vi and vii
(d) i, ii, v, iii, iv, vi and vii.

7. Design metrics are
(i) engineering cost
(ii) time to market
(iii) power dissipation
(iv) flexibility
(v) system and user safety
(vi) performance
(vii) prototype development time
(viii) maintenance of the system

(a) all
(b) all except v and viii
(c) iii and vi
(d) all except i and ii

8. Total power dissipation reduced by
   (i) reducing operating voltages,
   (ii) operating at lower clock frequency if processes meet the deadlines
   (iii) use of wait and stop instructions when system is inactive or idle
   (iv) use of cache disabling instructions
   (v) optimizing the amount and type of hardware required for the system

   (a) all except v (b) i, ii and iii (c) all except iv **(d) all**

9. Cache
   (i) is a fast read and write on-chip memory for the processor execution unit
   (ii) stores instructions and data fetched in advance from ROM or RAM for use of execution unit and for data write back for RAM
   (iii) has advantage that processor execution unit does not have to wait for instruction and data from external buses and also does faster write back of data meant for RAM
   (iv) use is must in embedded system with large memory requirements
   (v) has lower power dissipation compared to RAM

   (a) i, ii, iii and iv
   (b) i, iii, iv and v
   (c) i, ii, iii and v
   (d) i, ii and iii

10. A communication protocol specifies ___________
    (i) the ways of communication of signals on the bus
    (ii) ways of arbitration when several devices need to communicate through the bus or the ways of polling from the devices need of the bus at an instance
    (iii) memory requirement during communication
    (iv) minimum rate of data transfer during communication
    (v) interrupt service mechanism

    (a) i, ii, iii and iv
    (b) i, iii, iv and v
    (c) i and ii
    (d) i, ii and iii

EMBEDDED SYSTEMS

UNIT - 2

MEMORY AND INPUT / OUTPUT MANAGEMENT

Aim :

To Give an insight of Embedded systems

Objectives :

The Course will enable the students to:
1.Get introduced to features that build an embedded systems
2.Learn about the various components within an embedded systems
3.Learn the techniques of interfacing between processors & peripheral devices related to embedded processing
4.Do the efficient programs on any dedicated processor

Outcomes :

The students should be able to :

1. Understand basic building blocks of embedded systems
2. Interface various peripherals to processors
3. Program embedded systems
4. Use the basic concepts of system programming like operating systems,assembler compilers etc.., and to understand the management task needed for developing embedded systems.

Pre-Test - MCQ Type

1. Which is the most basic non-volatile memory?

   a) Flash memory

   b) PROM

   c) EPROM

   d) ROM

Answer: d

2. Which of the following is serial access memory?

   a) RAM

   b) Flash memory

   c) Shifters

   d) ROM

   Answer: c

3. Which is the early form of non-volatile memory?

   a) magnetic core memory

   b) ferrimagnetic memory

   c) anti-magnetic memory

   d) anti-ferromagnetic

   Answer: a

4. Which of the following memories has more speed in accessing data?

   a) SRAM

   b) DRAM

   c) EPROM

   d) EEPROM

   Answer: a

5. In which memory, the signals are multiplexed?

   a) DRAM

   b) SRAM

   c) EPROM

   d) EEPROM

   Answer: a

6. How many main signals are used with memory chips?

   a) 2
   b) 4
   c) 6
   d) 8

   Answer: b

7. What is the purpose of the address bus?

   a) to provide data to and from the chip

   b) to select a specified chip

   c) to select a location within the memory chip

   d) to select a read/write cycle

   Answer: c

8. Which are the two main types of processor connection to the motherboard?

   a) sockets and slots

   b) sockets and pins

   c) slots and pins

   d) pins and ports

   Answer: a

9. Which of the following has programmable hardware?

   a) microcontroller
   b) microprocessor
   c) coprocessor
   d) FPGA

   Answer:D

10. How many numbers of ways are possible for allocating the memory to the modular blocks?

a) 1

b) 2

c) 3

d) 4

Answer:C

11. Which of the following can periodically trigger the context switch?

a) software interrupt

b) hardware interrupt

c) peripheral

d) memory

Answer:B

12. Which of the following are interfaced as the outputs to the parallel ports?

a) keyboards

b) switches

c) LEDs

d) knobs

Answer:C

13. The RS232 is also known as

a) UART

b) SPI

c) Physical interface

d) Electrical interface

Answer:D

14. Which of the following is not a serial protocol?

a) SPI

b) I2C

c) Serial port

d) RS232

Answer: d

15. Which of the following can transfer multiple bits of data simultaneously?

a) serial port

b) sequential port

c) concurrent unit

d) parallel port

Answer: d

2.1 Programming Input and Output

- ❏ An embedded system is useless if it cannot communicate with the outside world. embedded systems need to employ I/O mechanisms to both receive outside data, and transmit commands back to the outside world.

2.1.1 Programming the IO Bus

- ❏ When programming IO bus controls, there are 5 major variations on how to handle it- the main thread poll, the multithread poll, the interrupt method, the interrupt+thread method, and using a DMA controller.

2.1.2 Main thread poll

- ❏ whenever you have output ready to be sent, you check if the bus is free and send it. Depending on how the bus works, sending it can take a large amount of time, during which you may not be able to do anything else.
- ❏ Input works similarly- every so often you check the bus to see if input exists.

2.1.3 Multithread polling

- ❏ In this method, we spawn off a special thread to poll. If there is no IO when it polls, it puts itself back to sleep for a predefined amount of time.
- ❏ If there is IO, it deals with it on the IO thread, allowing the main thread to do whatever is needed.
- ❏ This technique is good if your system supports threading, but does not support interrupts or has run out of interrupts.
- ❏ It does not work well when frequent IO is expected.

2.1.4 Interrupt architecture

- ❏ In this method, the bus fires off an interrupt to the processor whenever IO is ready.
- ❏ The processor then jumps to a special function, dropping whatever else it was doing.
- ❏ The special function (called an interrupt handler, or interrupt service routine) takes care of all IO, then goes back to whatever it was doing.
- ❏ This technique is great as long as dealing with the IO is a short process, such as when you just need to set up DMA.
- ❏ If its a long process, use multithreaded polling or interrupts with threads.

2.1.5 Interrupts and threads

- ❏ In this technique, you use an interrupt to detect when IO is ready. Instead of dealing with the IO directly, the interrupt signals a thread that IO is ready and let's that thread deal with it.
- ❏ Signalling the thread is usually done via semaphore- the semaphore is initialized to the taken state.
- ❏ The IO thread tries to take the semaphore, which fails and the OS puts it to sleep.
- ❏ When IO is ready, the interrupt is fired and releases the semaphore.
- ❏ The thread then wakes up, and handles the IO before trying to take the semaphore and being put back to sleep.

2.1.6 DMA (Direct Memory Access) Controller

- ➢ In some specialised situations, such as where a set of data must be transferred to a communications IO device, a DMA controller may be present that can automatically detect when the IO device is ready for more data, and transfer that data.

➢ This technique may be used in conjunction with many of the other techniques, for instance an interrupt may be used when the data transfer is complete.

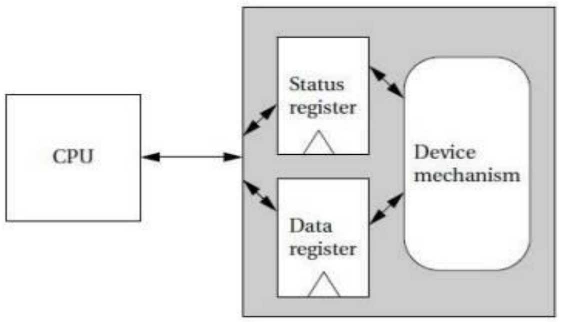


FIGURE 2.1 Structure of a typical I/O device

➢ The basic techniques for I/O programming can be understood relatively independent of the instruction set.

2.1.7 INPUT AND OUTPUT DEVICES

❏ Input and output devices usually have some analog or non-electronic component— for instance, a disk drive has a rotating disk and analog read/write electronics.

❏ But the digital logic in the device that is most closely connected to the CPU very strongly resembles the logic you would expect in any computer system.

❏ The interface between the CPU and the device's internals is a set of registers. The CPU talks to the device by reading and writing the registers.

❏ Devices typically have several registers,

❖ *Data registers* hold values that are treated as data by the device, such as the data read or written by a disk.

❖ *Status registers* provide information about the device's operation, such as whether the current transaction has completed.

❏ Some registers may be read-only, such as a status register that indicates when the device is done, while others may be readable or writable.

Example : The 8251 UART (Universal Asynchronous Receiver/Transmitter) is the original device used for serial communications, such as the serial port connections on PCs.

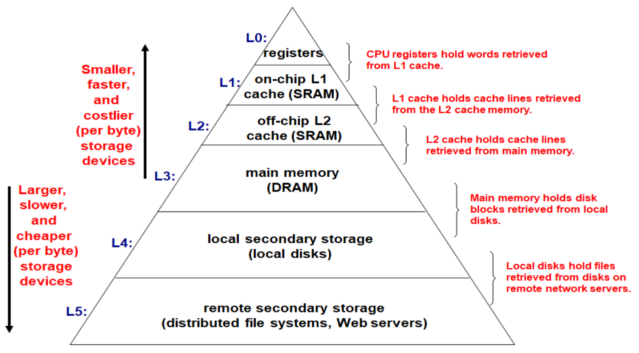
2.1.8 INPUT AND OUTPUT PRIMITIVES

- Microprocessors can provide programming support for input and output in two ways: I/O instructions and memory-mapped I/O.
- Some architectures, such as the Intel x86, provide special instructions (in and out in the case of the Intel x86) for input and output.
- These instructions provide a separate address space for I/O devices.
- The most common way to implement I/O is by memory mapping; even CPUs that provide I/O instructions can also implement memory-mapped I/O.
- As the name implies, memory-mapped I/O provides addresses for the registers in each I/O device.
- Programs use the CPU's normal read and write instructions to communicate with the devices.

2.2 Memory System Mechanism

1. It Is an approach for organizing memory and storage systems.
2. A memory hierarchy is organized into several levels – each smaller, faster, & more expensive / byte than the next lower level.

An Example Memory Hierarchy



L0:
registers
CPU registers hold words retrieved from L1 cache.
L1:
on-chip L1 cache (SRAM)
L1 cache holds cache lines retrieved from the L2 cache memory.
L2:
off-chip L2 cache (SRAM)
L2 cache holds cache lines retrieved from main memory.
L3:
main memory (DRAM)
Main memory holds disk blocks retrieved from local disks.
L4:
local secondary storage (local disks)
Local disks hold files retrieved from disks on remote network servers.
L5:
remote secondary storage (distributed file systems, Web servers)
Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices

3. Modern microprocessors do more than just read and write a monolithic memory. Architectural features improve both the speed and capacity of memory systems. Microprocessor clock rates are increasing at a faster rate than memory speeds
4. As a result, computer architects resort to caches to increase the average performance of the memory system. Although memory capacity is increasing steadily, program sizes are increasing as well, and designers may not be willing to pay for all the memory demanded by an application
5. Modern microprocessor units (MMUs) perform address translations that provide a larger virtual memory space in a small physical memory. In this section, we review both caches and MMUs.

2.2.1 Caches

- ➢ Caches are widely used to speed up memory system performance. The cache speeds up average memory access time when properly used.
- ➢ It increases the variability of memory access times—accesses in the cache will be fast, while access to locations not cached will be slow.
- ➢ A cache is a small, fast memory that holds copies of some of the contents of main memory. Because the cache is fast, it provides higher-speed access for the CPU; but since it is small, not all requests can be satisfied by the cache, forcing the system to wait for the slower main memory.
- ➢ Caching makes sense when the CPU is using only a relatively small set of memory locations at any one time; the set of active locations is often called the working set.
- ➢ A cache controller mediates between the CPU and the memory system comprising the main memory.
- ➢ The cache controller sends a memory request to the cache and main memory.
- ➢ If the requested location is in the cache, the cache controller forwards the location's contents to the CPU and aborts the main memory request; this condition is known as a cache hit.
- ➢ If the location is not in the cache, the controller waits for the value from main memory and forwards it to the CPU; this situation is known as a cache miss.

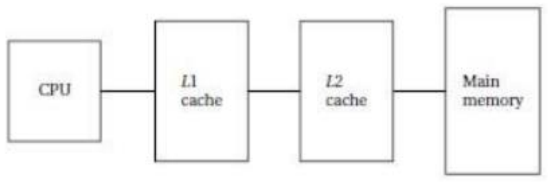


FIGURE 3.7
A two-level cache system.

- ➢ As the program's working set changes, we expect locations to be removed from the cache to make way for new locations.
- ➢ The simplest way to implement a cache is a *direct-mapped cache*, as shown in Figure.
- ➢ The cache consists of cache *blocks*, each of which includes a tag to show which memory location is represented by this block, a data field holding the contents of that memory, and a valid tag to show whether the contents of this cache block are valid. An address is divided into three sections.
- ➢ The index is used to select which cache block to check. The tag is compared against the tag value in the block selected by the index.
- ➢ If the address tag matches the tag value in the block, that block includes the desired memory location. If the length of the data field is longer than the

minimum addressable unit, then the lowest bits of the address are used as an offset to select the required value from the data field.

- ➢ Given the structure of the cache, there is only one block that must be checked to see whether a location is in the cache—the index uniquely determines that block.
- ➢ If the access is a hit, the data value is read from the cache.
- ➢ The direct-mapped cache is both fast and relatively low cost, but it does have limits in its caching power due to its simple scheme for mapping the cache onto main memory.
- ➢ The limitations of the direct-mapped cache can be reduced by going to the *set-associative* cache structure .

2.2.2 Memory Management Units and Address Translation

- ❖ AMMU translates addresses between the CPU and physical memory. This translation process is often known as memory mapping since addresses are mapped from a logical space into a physical space. MMUs in embedded systems appear primarily in the host processor. It is helpful to understand the basics of MMUs for embedded systems complex enough to require them.
- ❖ MMUs are used to provide virtual addressing.The MMU accepts logical addresses from the CPU. Logical addresses refer to the program's abstract address space but do not correspond to actual RAM locations. The MMU translates them from tables to physical addresses that do correspond to RAM. By changing the MMU's tables, you can change the physical location at which the program resides without modifying the program's code or data.
- ❖ If we add a secondary storage unit such as flash or a disk, we can eliminate parts of the program from main memory. In a virtual memory system, the MMU keeps track of which logical addresses are actually resident in main memory; those that do not reside in main memory are kept on the secondary storage device.

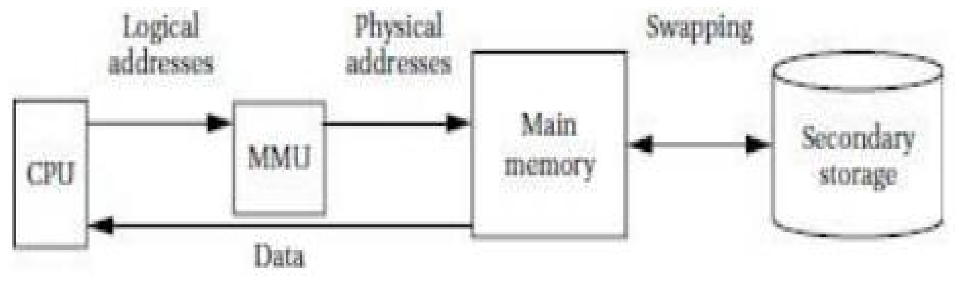


FIGURE 3.10

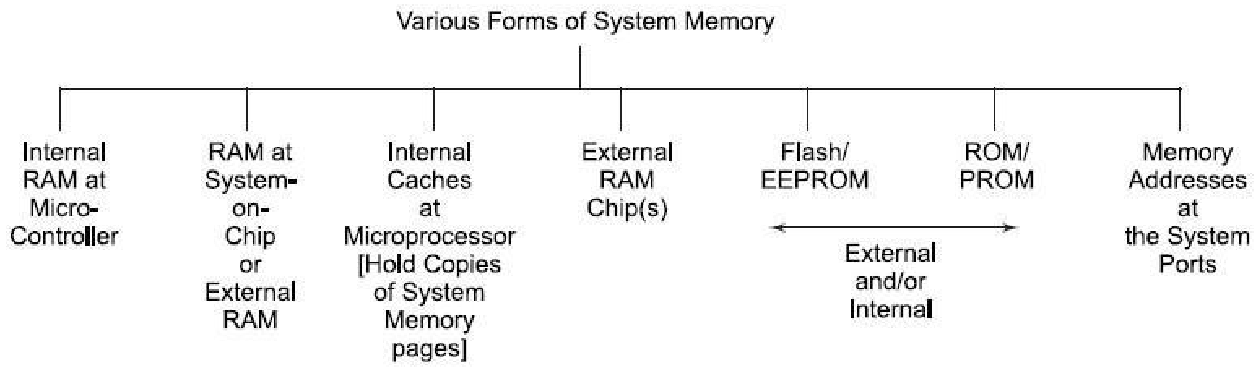
A virtually addressed memory system.

❑

2.3 Memory,Input output devices and interfacing

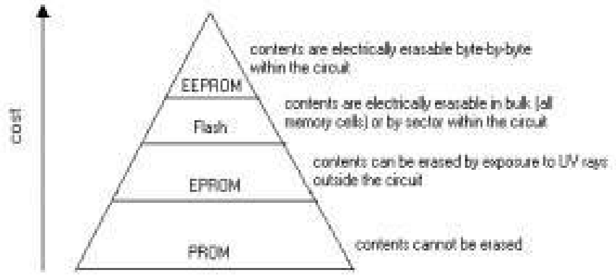
In a system, there are various types of memories.

1. Internal RAM
2. Internal ROM/PROM/EPROM
3. External RAM for the temporary data and stack
4. Internal caches
5. EEPROM or flash
6. External ROM or PROM for embedding software
7. RAM Memory buffers at the ports


Various Forms of System Memory
Internal RAM at Micro-Controller
RAM at System-on-Chip or External RAM
Internal Caches at Microprocessor [Hold Copies of System Memory pages]
External RAM Chip(s)
Flash/ EEPROM
ROM/ PROM
External and/or Internal
Memory Addresses at the System Ports

- ❏ The memory unit in an embedded system should have low access time and high density (a memory chip has greater density if it can store more bits in the same amount of space).
- ❏ Memory in an embedded system consists of ROM (only read operations permitted) and RAM (read and write operations are permitted).
- ❏ The contents of ROM are non-volatile (power failure does not erase the contents) while RAM is volatile.

- ❏ ROM stores the program code while RAM is used to store transient input or output data. Embedded systems generally do not possess secondary storage devices such as magnetic disks.
- ❏ As programs of embedded systems are small there is no need for virtual storage.


cost
EEPROM
contents are electically erasable byte-by-byte within the circuit
Flash
contents are electrically erasable in bulk (all memory cells) or by sector within the circuit
EPROM
contents can be erased by exposure to UV rays outside the circuit
PROM
contents cannot be erased

2.3.1 Volatile memory

- ❑ A primary distinction in memory types is volatility. Volatile memories only hold their contents while power is applied to the memory device.
- ❑ As soon as power is removed, the memories lose their contents; consequently, volatile memories are unacceptable if data must be retained when the memory is switched off.
- ❑ Examples of volatile memories include static RAM (SRAM),synchronous static RAM (SSRAM), synchronous dynamic RAM (SDRAM), and FPGA on-chip memory.

2.3.2 Nonvolatile memory

- ❖ Non-volatile memories retain their contents when power is switched off, making them good choices for storing information that must be retrieved after a system power-cycle. Processor boot-code, persistent application settings, and FPGA configuration data are typically stored in non- volatile memory.
- ❖ Although non-volatile memory has the advantage of retaining its data when power is removed, it is typically much slower to write to than volatile memory, and often has more complex writing and erasing procedures.
- ❖ Non-volatile memory is also usually only guaranteed to be erasable a given number of times, after which it may fail. Examples of non- volatile memories include all types of flash, EPROM, and EEPROM.

2.3.3 ROM Overview

- ❑ Although there are exceptions, the ROM is generally viewed as read only device.when the ROM is implemented,positions in the array that are to store a logical 0 have a transistor connected as shown in figure. Those positions intended to store a logical 1 have none.

2.3.4 Static RAM overview

- ❑ A high level interface to the SRAM is very similar to that for the ROM.The major differences arise from support for write capability. Figure 3.4 represents the major I/O signals and a typical cell in an SRAM array.

2.3.5 SDRAM

- ➢ SDRAM is another type of volatile memory. It is similar to SRAM, except that it is dynamic and must be refreshed periodically to maintain its content.
- ➢ The dynamic memory cells in SDRAM are much smaller than the static memory cells used in SRAM.
- ➢ This difference in size translates into very high-capacity and low-cost memory devices.
- ➢ In addition to the refresh requirement, SDRAM has other very specific interface requirements which typically necessitate the use of special controller hardware.
- ➢ Unlike SRAM, which has a static set of address lines, SDRAM divides up its memory space into banks, rows, and columns.
- ➢ Switching between banks and rows incurs some overhead, so that efficient use of SDRAM involves the careful ordering of accesses.
- ➢ SDRAM also multiplexes the row and column addresses over the same address lines, which reduces the pin count necessary to implement a given size of SDRAM.
- ➢ Higher speed varieties of SDRAM such as DDR, DDR2, and DDR3 also have strict signal integrity requirements which need to be carefully considered during the design of the PCB.
- ➢ SDRAM devices are among the least expensive and largest-capacity types of RAM devices available, making them one of the most popular.

2.3.6 Dynamic RAM Overview

- ❑ Larger microcomputer systems use Dynamic RAM (DRAM) rather than Static RAM (SRAM) because of its lower cost per bit.
- ❑ DRAMs require more complex interface circuitry because of their multiplexed address bus and because of the need to refresh each memory cell periodically.
- ❑ A typical DRAM memory is laid out as a square array of memory cells with an equal number of rows and columns.
- ❑ Each memory cell stores one bit. The bits are addressed by using half of the bits (the most significant half) to select a row and the other half to select a column.
- ❑ Each DRAM memory cell is very simple – it consists of a capacitor and a MOSFET switch. A DRAM memory cell is therefore much smaller than an SRAM cell which needs at least two gates to implement a flip-flop.
- ❑ The Input / output organization of a computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the

computer, provides an efficient mode of communication between the central system and the outside environment.

- ❖ The most common input output devices are:
  - i) Monitor
  - ii) Keyboard
  - iii) Mouse
  - iv) Printer
  - v) Magnetic tapes
- ❑ The devices that are under the direct control of the computer are said to be connected online.

➔ Input Output Interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.

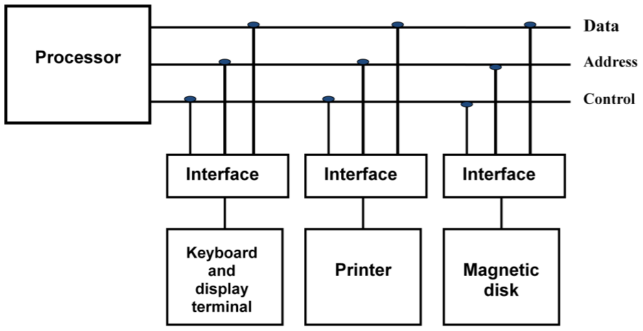
The Major Differences are:-

1. Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in the peripherals differ from the word format in the CPU and memory.

1) The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.
2) To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervise and synchronize all input and out transfers.
3) These components are called Interface Units because they interface between the processor bus and the peripheral devices.
4) I/O BUS and Interface Module It defines the typical link between the processor and several peripherals.
5) The I/O Bus consists of data lines, address lines and control lines. The I/O bus from the processor is attached to all peripherals interface.
6) To communicate with a particular device, the processor places a device address on address lines.
7) Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller.
8) It is also synchronizes the data flow and supervises the transfer between peripheral and processor.
9) Each peripheral has its own controller. For example, the printer controller controls the paper motion, the print timing The control lines are referred as I/O command.

The commands are as following:

- ➢ Control command- A control command is issued to activate the peripheral and to inform it what to do.
- ➢ Status command- A status command is used to test various status conditions in the interface and the peripheral.
- ➢ Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers. Data Input command- The data input command is the opposite of the data output.



Connection of I/O bus to input-output devices

- ➢ To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines.
- ➢ There are 3 ways that computer buses can be used to communicate with memory and I/O:
  i.Use two Separate buses , one for memory and other for I/O.
  ii. Use one common bus for both memory and I/O but separate control lines for each.
  iii. Use one common bus for memory and I/O with common control lines.
- ➢ I/O Processor In the first method, the computer has independent sets of data, address and control buses one for accessing memory and other for I/O. This is done in computers that provide a separate I/O processor (IOP).
- ➢ The purpose of IOP is to provide an independent pathway for the transfer of information between external device and internal memory,

2.4 Interrupt Handling

- ❖ An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
- ❖ Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an Interrupt Service Routine (ISR) or Interrupt Handler.
- ❖ ISR tells the processor or controller what to do when the interrupt occurs. The interrupts can be either hardware interrupts or software interrupts.

2.4.1 Hardware Interrupt

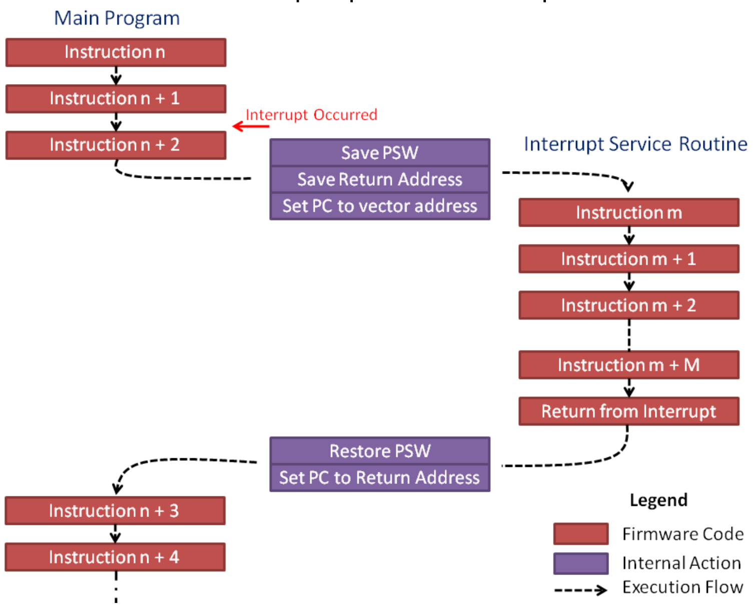
1. A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like a disk controller or an external peripheral.
2. For example, when we press a key on the keyboard or move the mouse, they trigger hardware interrupts which cause the processor to read the keystroke or mouse position.

2.4.2 Software Interrupt

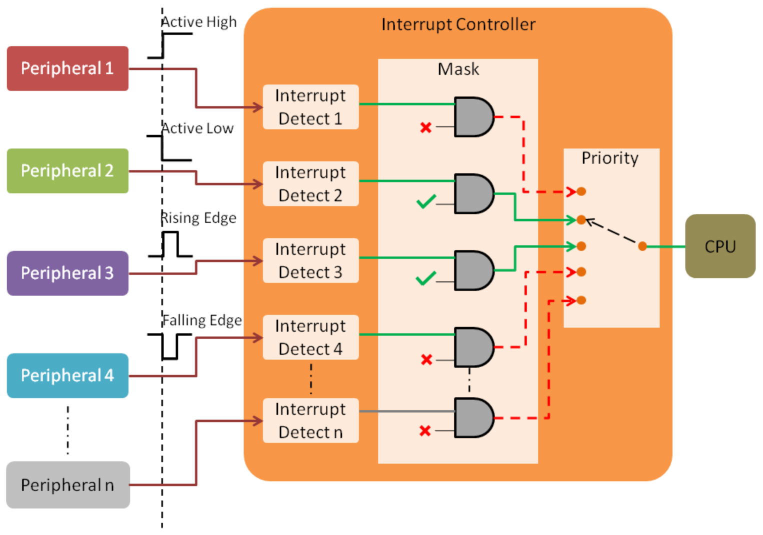
1. A software interrupt is caused either by an exceptional condition or a special instruction in the instruction set which causes an interrupt when it is executed by the processor.
2. For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero exception, thus causing the computer to abandon the calculation or display an error message. Software interrupt instructions work similar to subroutine calls.

2.4.3 Interrupt Mechanism and ISR

1. Assume the CPU is performing some activity by running a sequence of instructions. Now when a peripheral controller wants to break the sequence, it generates the Interrupt signal, which is typically a single line of connection from the peripheral/interrupt controller.



2. Upon reception of the same, the general sequence of handling is as follows
   - Current instruction being executed is completed.
   - CPU copies the internal register called PSW – Program Status Word (the register that has the flags like Carry, Zero etc) on to stack or an internal backup register
   - Address of the next instruction to be executed is saved on the stack or another internal backup register so that it can be resumed after interrupt processing.
   - The Program counter is set to the vector address specific to that interrupt.
3. A piece of code called the Interrupt Service Routine (ISR) is placed in the vector location of an interrupt, to handle it. Typical flow of operation includes
   - If needed, store the previous PSW and return address in memory
   - Store the context – any more internal registers, status registers etc that might be modified during the course of execution of this ISR
   - Perform necessary operations to process that interrupt like copying data to/from the peripheral, prepare for next cycle etc.
   - Up on completion, restore the context back to the original values from the stored values
   - Restore the PSW and jump to the address of the instruction next to the interrupted one.
4. While the overall interrupt mechanism remains same, the exact handling, ways to return from interrupt etc are different for CPU architectures like 8051, ARM, AVR etc.



5. Interrupt Sources, Level, Masking and Priority Interrupt Controller
6. In a typical SoC, there could be hundreds of peripherals – interrupt sources that may vie for processor attention. It is impossible to route all these signals to the CPU.
7. So there is another special purpose peripheral called Interrupt Controller to which all the peripheral interrupts signals are connected.
8. The controller is in turn connected to the CPU with one or few lines of signals, with which it can interrupt the CPU.
9. Controllers are provided with various registers to mask/unmask the interrupts, read pending/raw status, set priority etc.
10. The controller monitors the signals from peripherals and if there are any active interrupts, based on the preconfigured priority decides the source to be processed first. Then it signals the CPU with an interrupt.
11. The CPU, in the ISR, can then read the pending interrupt register and process the same.

2.4.4 Interrupt Sources

➢ Interrupts could be from hardware, software or other sources.
➢ Hardware interrupts are the most discussed types originating from on-chip/on-board peripherals. Apart of them, most architectures provides instructions that can be used to generate an interrupt – typically to enter kernel mode. Also exceptions like page fault, instruction fetch errors etc are handled as interrupts.

2.4.5 Interrupt Types and Levels

➢ The interrupt signal could either level triggered or edge triggered. Level triggered interrupts remains active till the condition of interrupt remains, where as with edge triggered interrupts there is a change in signal level and then it again changes back to original state.
➢ The type of interrupt can be decided based on various conditions like transition in state (level triggered) or occurrence of events (edge) and is mostly specific to the hardware implementation.
➢ The level triggered interrupt could be
   1. Low level triggered
   2. High level triggered
➢ And the edge triggered interrupt could be
   1. Rising edge triggered
   2. Falling edge triggered
   3. Both edge triggered

2.4.6 Interrupt Maskability

- ➢ While most of the interrupts could be enabled or disabled by the will of the user, in some implementations, there might be some interrupts that cannot be disabled or masked.
- ➢ These Non-Maskable Interrupts or NMI, as they are called, are used to inform CPU of critical conditions like power going down, occurrence of watchdog interrupt etc. Most likely, there is no need for special handling of the same in software perspective.

2.4.7 Interrupt Priority

- ➢ As we have seen the Interrupt controller can accept hundreds of interrupt sources, there is a need for a mechanism to determine the priority when two or more interrupts are active simultaneously.
- ➢ So the controller provides configurations to set the priority. Also the CPU itself may accept multiple lines of interrupts each with different priorities and handles them suitably.

2.4.8 Interrupt Nesting and Preemption

- ➢ The concept of priority automatically leads us to the concept of nesting and preemption.
- ➢ It might be required to process a higher priority interrupt even when the CPU is servicing a lower priority interrupts.
- ➢ So we need to support nesting, whereby interrupts are enabled as soon as possible after entering the ISR and current priority level of interrupt processing is set.
- ➢ When a higher priority interrupt occurs, it enters the ISR of that interrupt, interrupts enabled soon and the cycle goes on.

2.4.9 Interrupt Latency

- ❏ There is always a delay from the time of assertion of interrupt and the action handling of that interrupt. This might be due to various reasons like the CPU executing a critical section with interrupts disabled, or executing a higher priority interrupt etc.
- ❏ Handling of this delay or latency determines the difference between a General Purpose (no guaranteed latency) and a Real Time Operating System (guaranteed latency).

POST TEST MCQ TEST

1.What happens when 8 bits are transferred in the SPI?

a) wait statement

b) ready statement

c) interrupt

d) remains unchanged

ANSWER: C

2. Which signal is used to select the slave in the serial peripheral interfacing?

a) slave select

b) master select

c) interrupt

d) clock signal

ANSWER: A

3. How much time period is necessary for the slave to receive the interrupt and transfer the data?

a) 4 clock time period

b) 8 clock time period

c) 16 clock time period

d) 24 clock time period

ANSWER: B

4. What happens to the interrupts in an interrupt service routine?

a) disable interrupt

b) enable interrupts

c) remains unchanged

d) ready state

ANSWER: A

5. What is CAM stands for?

   a) content-addressable memory

   b) complex addressable memory

   c) computing addressable memory

   d) concurrently addressable memory

   ANSWER: A

6. Which of the following is replaced with the absolute addressing mode?

   a) relative addressing mode

   b) protective addressing mode

   c) virtual addressing mode

   d) temporary addressing mode

   ANSWER: A

7. What is the main purpose of the memory management unit?

   a) address translation

   b) large storage

   c) reduce the size

   d) provides address space

8. Which of the following are interfaced as inputs to the parallel ports?

   a) LEDs

   b) switch

   c) alphanumeric display

   d) seven segmented display

   ANSWER: B

9.How many registers are there to control the parallel port in the basic form?

a) 1

b) 3

c) 2

d) 5

ANSWER: C

10. Which of the following is also known as tri-state?

a) output port

b) input port

c) parallel port

d) output-input port

ANSWER: A

EMBEDDED SYSTEMS

CHAPTER 3

PROCESSES AND OPERATING SYSTEMS

Pre-requisites : Basics of Embedded systems

Aim :

The students are expected to learn the various aspects in Embedded systems and its different types of systems with their respective developments. The inputs of the subject is to make students familiarize with the developmental types and functions of embedded systems.

Objectives :

The course should enable the students to

1. To gain the knowledge on the embedded systems development
2. To have a clear understanding of the different types of developments in embedded systems and their various types.

Outcomes :

At the end of the course student should be able to

1. Help to examine the definitions and internalize the need for understanding the various types of embedded systems.
2. Develop the responsible attitude towards the use of embedded systems as well as the technology
3. Able to envision the social impact on the products/projects they develop in their career.
4. Analyse the professional responsibility and empowering access to information in the workplace.

Pre-test MCQ type

1. Which of the following works by dividing the processor's time?

a) single task operating system

b) multitask operating system

c) kernel

d) applications

2.Which of the following decides which task can have the next time slot?

a) single task operating system

b) applications

c) kernel

d) software

3.Which of the following controls the time slicing mechanism in a multitasking operating system?

a) kernel

b) single tasking kernel

c) multitasking kernel

d) application manager

4. Which of the following provides a time period for the context switch?

a) timer

b) counter

c) time slice

d) time machine

5. Which of the following can periodically trigger the context switch?

a) software interrupt

b) hardware interrupt

c) peripheral

d) memory

6. Which interrupt provides system clock in the context switching?

a) software interrupt

b) hardware interrupt

c) peripheral

d) memory

7. The special tale in the multitasking operating system is also known as

a) task control block

b) task access block

c) task address block

d) task allocating block

8. Which of the following stores all the task information that the system requires?

a) task access block

b) register

c) accumulator

d) task control block

9. Which of the following can implement the message passing and control?

a) application software

b) operating system

c) software

d) kernel

10. What is the Real-time systems?

a) Used for monitoring events as they occur

b) Primarily used on mainframe computers

c) Used for real-time interactive users

d) Used for program development

11. In real time operating system is__________

A. kernel is not required

B. process scheduling can be done only once task

C. must be serviced by its deadline period

D. all processes have the same priority

12. When the System processes data instructions without any delay is called as

A. online system

B. real-time system

C. instruction system

D. offline system

13.In Which of the following algorithm the process that requests the CPU first is allocated the CPU first

a)Shortest-Job-First Scheduling / EDF

b)First-Come, First-Served Scheduling

c)Priority Scheduling

d)Round-Robin Scheduling

14. Which scheduling policy is most suitable for time shared operating system ?

a.Shortest job first / EDF

b.FCFS

c.LCFS

d.Round robin

15. Which of the following scheduling algorithm associates with each process the length of the process's next CPU burst

a)Shortest-Job-First Scheduling / EDF

b)First-Come, First-Served Scheduling

c)Priority Scheduling

d)Round-Robin Scheduling

3.1 MULTIPLE TASKS AND MULTIPLE PROCESSES

3.1.1Tasks and Processes

Many (if not most) embedded computing systems do more than one thing that is, the environment can cause mode changes that in turn cause the embedded system to behave quite differently. For example, when designing a telephone answering machine,

We can define recording a phone call and operating the user's control panel as distinct tasks, because they perform logically distinct operations and they must be performed at very different rates. These different **tasks** are part of the system's functionality, but that application-level organization of functionality is often reflected in the structure of the program as well.

A ***process*** is a single execution of a program. If we run the same program two different times, we have created two different processes. Each process has its own state that includes not only its registers but all of its memory. In some OSs, the memory management unit is used to keep each process in a separate address space. In others, particularly lightweight RTOSs, the processes run in the same address space. Processes that share the same address space are often called ***threads***.

As shown in Figure 3.1, this device is connected to serial ports on both ends. The input to the box is an uncompressed stream of bytes. The box emits a compressed string of bits on the output serial line, based on a predefined compression table. Such a box may be used, for example, to compress data being sent to a modem.

The program's need to receive and send data at different rates for example, the program may emit 2 bits for the first byte and then 7 bits for the second byte will obviously find itself reflected in the structure of the code. It is easy to create irregular, ungainly code to solve this problem; a more elegant solution is to create a queue of output bits, with those bits being removed from the queue and sent to the serial port in 8-bit sets.

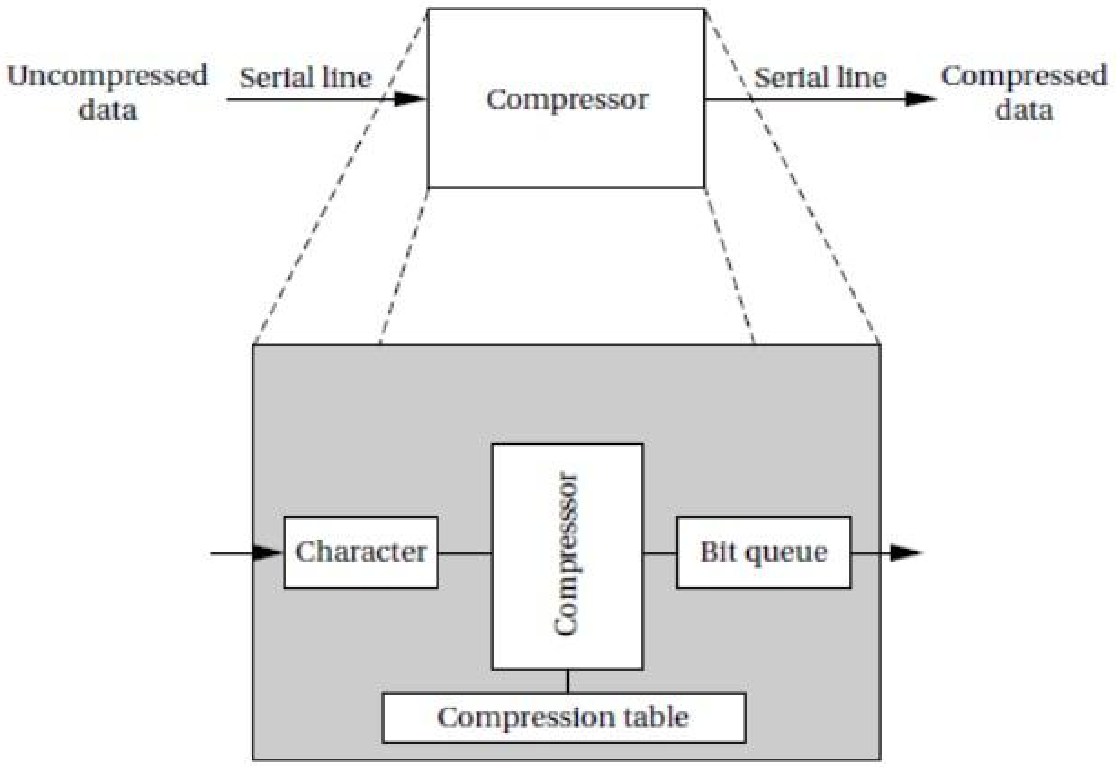


Fig 3.1 An on-the-fly compression box.

But beyond the need to create a clean data structure that simplifies the control structure of the code, we must also ensure that we process the inputs and outputs at the proper rates. For example, if we spend too much time in packaging and emitting output characters, we may drop an input character. Solving timing problems is a more challenging problem.

The text compression box provides a simple example of rate control problems. A control panel on a machine provides an example of a different type of rate control problem, the ***asynchronous input***.

The control panel of the compression box may, for example, include a compression mode button that disables or enables compression, so that the input text is passed through unchanged when compression is disabled. We certainly do not know when the user will push the compression mode button the button may be depressed asynchronously relative to the arrival of characters for compression.

3.1.2Multirate Systems

Implementing code that satisfies timing requirements is even more complex when multiple rates of computation must be handled. ***Multirate*** embedded computing systems are very common, including automobile engines, printers, and cell phones. In all these systems, certain operations must be executed periodically, and each operation is executed at its own rate.

3.1.3 Timing Requirements on Processes

Processes can have several different types of timing requirements imposed on them by the application. The timing requirements on a set of processes strongly influence the type of scheduling that is appropriate. A scheduling policy must define the timing requirements that it uses to determine whether a schedule is valid. Before studying scheduling proper, we outline the types of process timing requirements that are useful in embedded system design.

Figure 3.2 illustrates different ways in which we can define two important requirements on processes: ***release time*** and ***deadline***.

The release time is the time at which the process becomes ready to execute; this is not necessarily the time at which it actually takes control of the CPU and starts to run. An aperiodic process is by definition initiated by an event, such as external data arriving or data computed by another process.

The release time is generally measured from that event, although the system may want to make the process ready at some interval after the event itself. For a periodically executed process, there are two common possibilities.

In simpler systems, the process may become ready at the beginning of the period. More sophisticated systems, such as those with data dependencies between processes, may set the release time at the arrival time of certain data, at a time after the start of the period.

A deadline specifies when a computation must be finished. The deadline for an aperiodic process is generally measured from the release time, since that is the only reasonable time reference. The deadline for a periodic process may in general occur at some time other than the end of the period.

Rate requirements are also fairly common. A rate requirement specifies how quickly processes must be initiated.

The ***period*** of a process is the time between successive executions. For example, the period of a digital filter is defined by the time interval between successive input samples.

The process's ***rate*** is the inverse of its period. In a multirate system, each process executes at its own distinct rate.

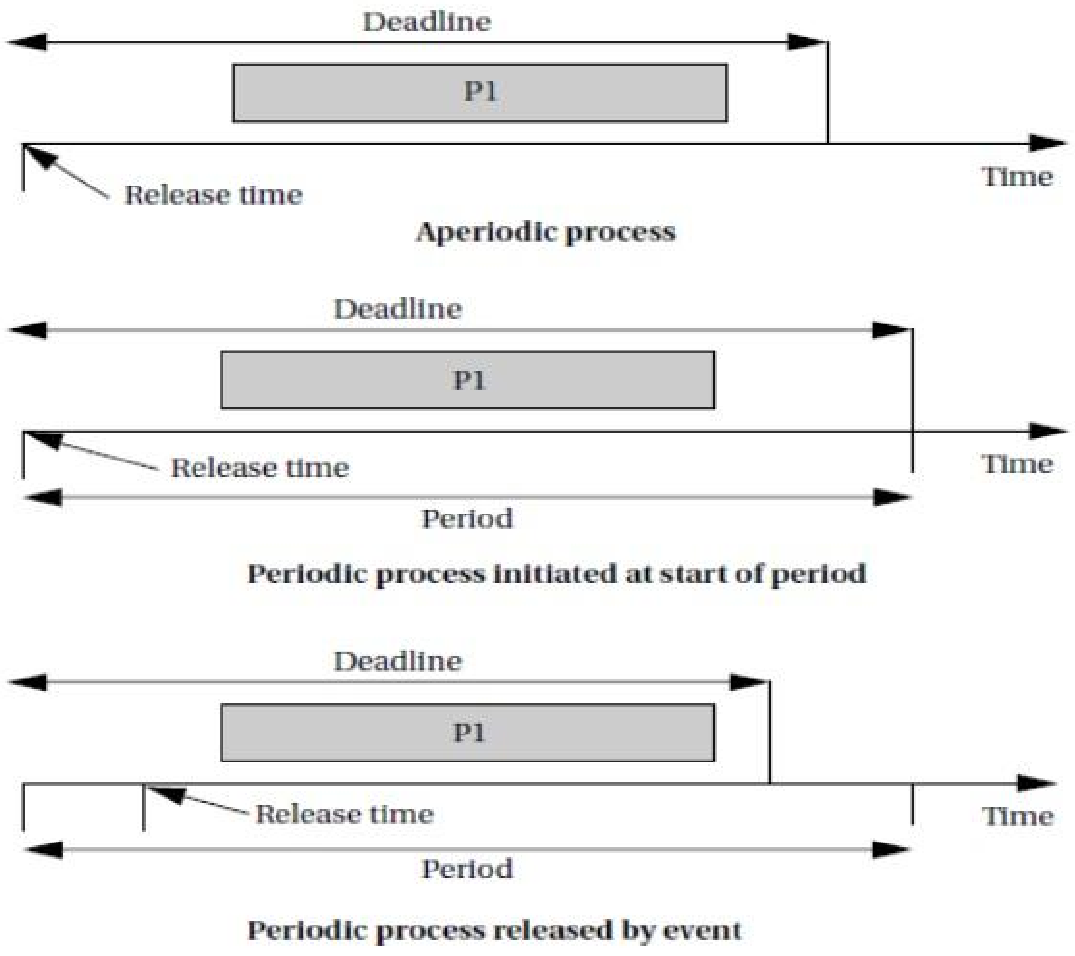


Fig 3.2 Example definitions of release times and deadlines.

The most common case for periodic processes is for the initiation interval to be equal to the period. However, pipelined execution of processes allows the initiation interval to be less than the period. Figure 3.3 illustrates process execution in a system with four CPUs.

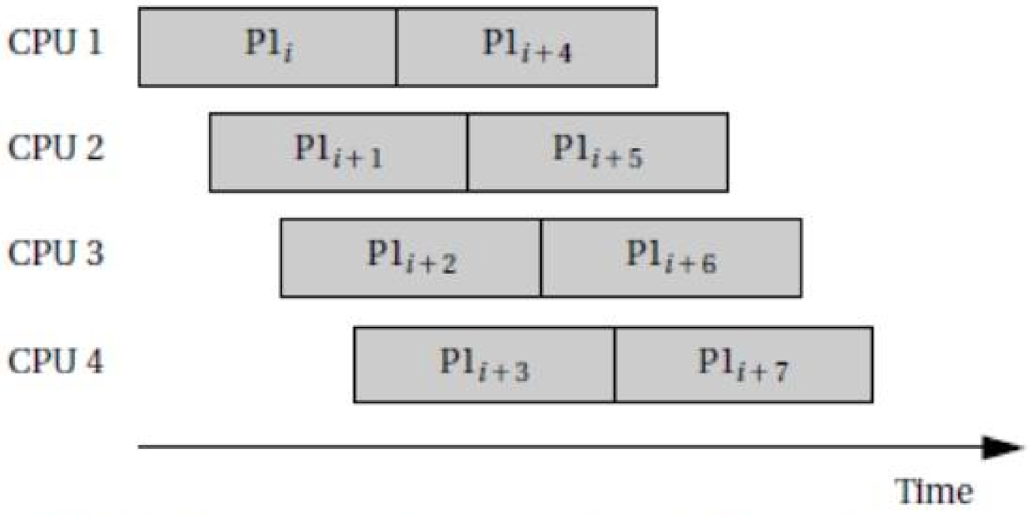


Fig 3.3 A sequence of processes with a high initiation rate.

3.1.4 CPU Metrics

We also need some terminology to describe how the process actually executes. The ***initiation time*** is the time at which a process actually starts executing on the CPU. The ***completion time*** is the time at which the process finishes its work.

The most basic measure of work is the amount of ***CPU time*** expended by a process. The CPU time of process *i* is called *Ci* . Note that the CPU time is not equal to the completion time minus initiation time; several other processes may interrupt execution. The total CPU time consumed by a set of processes is

$$T= \sum Ti$$

We need a basic measure of the efficiency with which we use the CPU. The simplest and most direct measure is ***utilization***:

U=CPU time for useful work/total available CPU time

Utilization is the ratio of the CPU time that is being used for useful computations to the total available CPU time. This ratio ranges between 0 and 1, with 1 meaning that all of the available CPU time is being used for system purposes. The utilization is often expressed as a percentage. If we measure the total execution time of all processes over an interval of time *t*, then the CPU utilization is

U=T/t

3.2 Processes and Context Switching

The best way to understand processes and context is to dive into an RTOS implementation. We will use the FreeRTOS.org kernel as an example; in particular, we will use version 4.7.0 for the ARM7 AT91 platform. A process is known in FreeRTOS.org as a task. Task priorities in FreeRTOS.org are ranked opposite to the convention we use in the rest of the book: higher numbers denote higher priorities and the priority 0 task is the idle task.

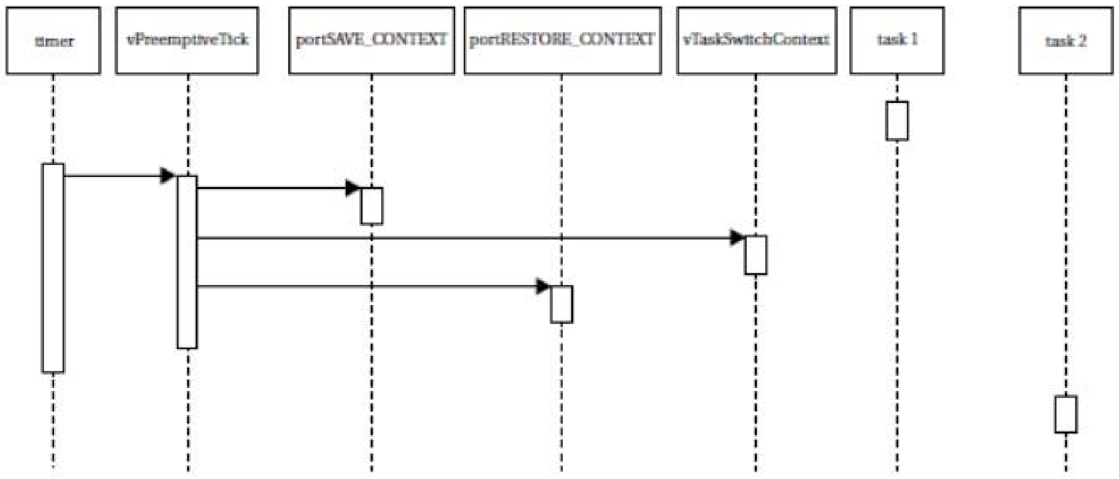


Fig 3.4 Sequence diagram for freeRTOS.org context switch.

To understand the basics of a context switch, let's assume that the set of tasks is in steady state:

Everything has been initialized, the OS is running, and we are ready for a timer interrupt. Figure 3.4 shows a sequence diagram for a context switch in freeRTOS.org. This diagram shows the application tasks, the hardware timer, and all the functions in the kernel that are involved in the context switch:

vPreemptiveTick () is called when the timer ticks.

portSAVE_CONTEXT() swaps out the current task context..

vTaskSwitchContext ( ) chooses a new task.

portRESTORE_CONTEXT() swaps in the new context.

3.2.1Operating Systems

An Operating system is a program that controls the execution of application programs and acts as an interface between the user of a computer and the computer hardware.

A more common definition is that the operating system is the one program running at all times on the computer (usually called the kernel), with all else being applications programs.

An Operating system is concerned with the allocation of resources and services, such as memory, processors, devices and information. The Operating System correspondingly includes programs to manage these resources, such as a traffic controller, a scheduler, memory management module, I/O programs, and a file system.

3.3 Scheduling Policies

A ***scheduling policy*** defines how processes are selected for promotion from the ready state to the running state. Every multitasking OS implements some type of scheduling policy. Choosing the right scheduling policy not only ensures that the system will meet all its timing requirements, but it also has a profound influence on the CPU horsepower required to implement the system's functionality.

Schedulability means whether there exists a schedule of execution for the processes in a system that satisfies all their timing requirements. In general, we must construct a schedule to show schedulability, but in some cases we can eliminate some sets of processes as unschedulable using some very simple tests.

Utilization is one of the key metrics in evaluating a scheduling policy. Our most basic requirement is that CPU utilization be no more than 100% since we can't use the CPU more than 100% of the time.

When we evaluate the utilization of the CPU, we generally do so over a finite period that covers all possible combinations of process executions. For periodic processes, the length of time that must be considered is the ***hyperperiod***, which is the least-common multiple of the periods of all the processes. (The complete schedule for the least-common multiple of the periods is sometimes called the ***unrolled schedule***.) If we evaluate the hyperperiod, we are sure to have considered all possible combinations of the periodic processes.

We will see that some types of timing requirements for a set of processes imply that we cannot utilize 100% of the CPU's execution time on useful work, even ignoring context switching overhead.

However, some scheduling policies can deliver higher CPU utilizations than others, even for the same timing requirements.

One very simple scheduling policy is known as ***cyclostatic*** scheduling or sometimes as ***Time Division Multiple Access*** scheduling. As illustrated in Figure 3.5, a cyclostatic schedule is divided into equal-sized time slots over an interval equal to the length of the hyperperiod *H*. Processes always run in the same time slot.

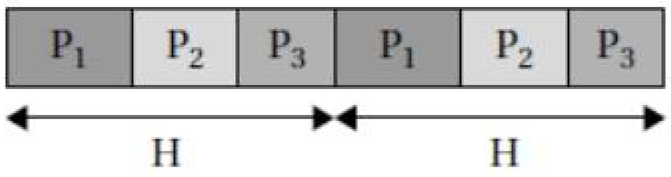


Fig 3.5 Cyclostatic scheduling.

Two factors affect utilization: the number of time slots used and the fraction of each time slot that is used for useful work. Depending on the deadlines for some of the processes, we may need to leave some time slots empty. And since the time slots are of equal size, some short processes may have time left over in their time slot

Another scheduling policy that is slightly more sophisticated is ***round robin***. As illustrated in Figure 3.6, round robin uses the same hyperperiod as does cyclostatic. It also evaluates the processes in order.

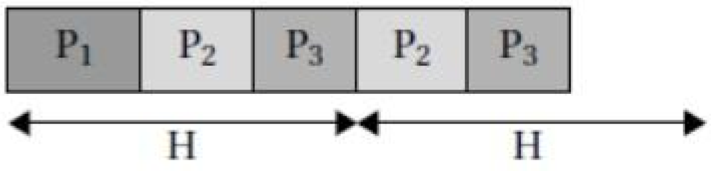


Fig 3.6 Round-robin scheduling.

But unlike cyclostatic scheduling, if a process does not have any useful work to do, the round-robin scheduler moves on to the next process in order to fill the time slot with useful work. In this example, all three processes execute during the first hyperperiod, but during the second one, *P*1 has no useful work and is skipped.

The processes are always evaluated in the same order. The last time slot in the hyperperiod is left empty; if we have occasional, non-periodic tasks without deadlines, we can execute them in these empty time slots. Round-robin scheduling is often used in hardware such as buses because it is very simple to implement but it provides some amount of flexibility.

In addition to utilization, we must also consider **scheduling overhead**—the execution time required to choose the next execution process, which is incurred in addition to any context switching overhead.

In general, the more sophisticated the scheduling policy, the more CPU time it takes during system operation to implement it. Moreover, we generally achieve higher theoretical CPU utilization by applying more complex scheduling policies with higher overheads.

The final decision on a scheduling policy must take into account both theoretical utilization and practical scheduling overhead.

3.3.1 Multiprocessor

A ***multiprocessor*** is, in general, any computer system with two or more processors coupled together. Multiprocessors used for scientific or business applications tend to have regular architectures: several identical processors that can access a uniform memory space. We use the term ***processing element (PE)*** to mean any unit responsible for computation, whether it is programmable or not.

Embedded system designers must take a more general view of the nature of multiprocessors. As we will see, embedded computing systems are built on top of an astonishing array of different multiprocessor architectures.

The first reason for using an embedded multiprocessor is that they offer significantly better cost/performance—that is, performance and functionality per dollar spent on the system—than would be had by spending the same amount of money on a uniprocessor system. The basic reason for this is that processing element purchase price is a *nonlinear* function of performance [Wol08].

The cost of a microprocessor increases greatly as the clock speed increases. We would expect this trend as a normal consequence of VLSI fabrication and market economics. Clock speeds are normally distributed by normal variations in VLSI processes; because the fastest chips are rare, they naturally command a high price in the marketplace.

Because the fastest processors are very costly, splitting the application so that it can be performed on several smaller processors is usually much cheaper.

Even with the added costs of assembling those components, the total system comes out to be less expensive. Of course, splitting the application across multiple processors does entail higher engineering costs and lead times, which must be factored into the project.

In addition to reducing costs, using multiple processors can also help with real time performance. We can often meet deadlines and be responsive to interaction much more easily when we put those time-critical processes on separate processors. Given that scheduling multiple processes on a single

Because we pay for that overhead at the nonlinear rate for the processor, as illustrated in Figure 3.7, the savings by segregating time-critical processes can be large—it may take an extremely large and powerful CPU to provide the same responsiveness that can be had from a distributed system.

Many of the technology trends that encourage us to use multiprocessors for performance also lead us to multiprocessing for low power embedded computing.

Several processors running at slower clock rates consume less power than a single large processor: performance scales linearly with power supply voltage but power scales with V2.

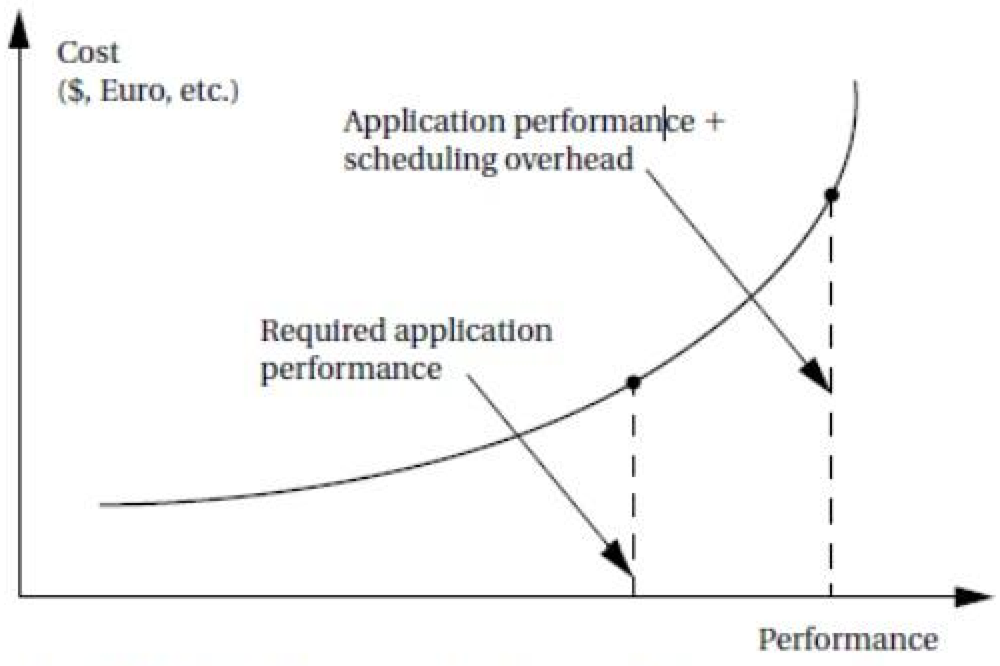


Fig 3.7 Scheduling overhead is paid for at a nonlinear rate.

Austin *et al.* [Aus04] showed that general-purpose computing platforms are not keeping up with the strict energy budgets of battery-powered embedded computing. Figure 3.8 compares the performance of power requirements of desktop processors with available battery power. Batteries can provide only about 75 Mw of power.

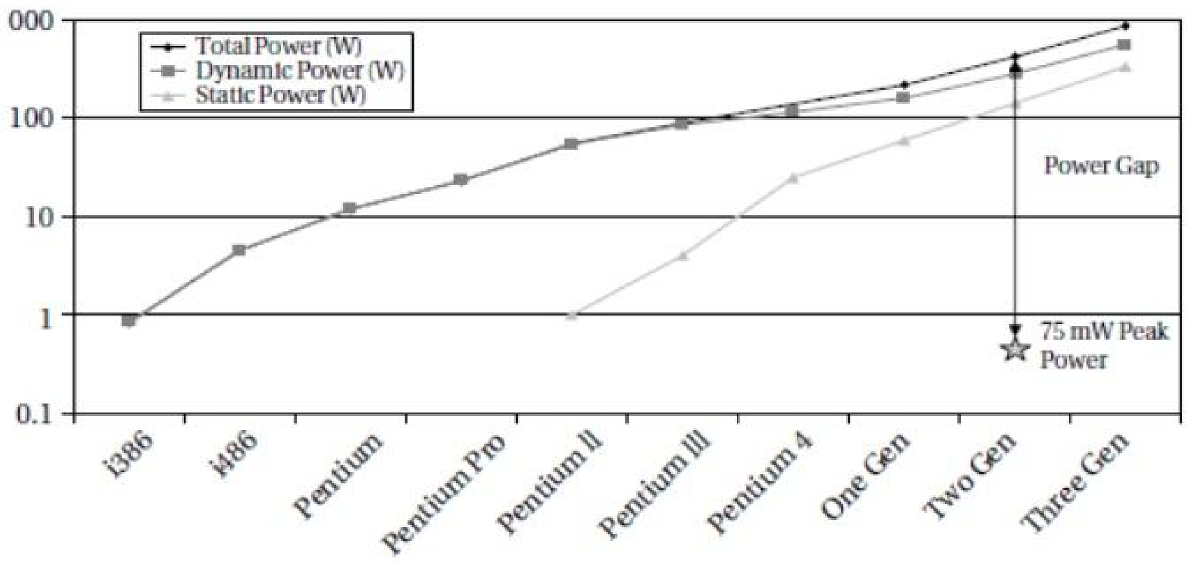


Fig 3.8 Power consumption trends for desktop processors [Aus04].

Desktop processors require close to 1000 times that amount of power to run. That huge gap cannot be solved by tweaking processor architectures or software. Multiprocessors provide a way to break through this power barrier and build substantially more efficient embedded computing platforms.

3.4 Interprocess Communication Mechanisms

Processes often need to communicate with each other. ***Interprocess communication mechanisms*** are provided by the operating system as part of the process abstraction.

In general, a process can send a communication in one of two ways: ***blocking*** or ***nonblocking***. After sending a blocking communication, the process goes into the waiting state until it receives a response.

Nonblocking communication allows the process to continue execution after sending the communication. Both types of communication are useful. There are two major styles of interprocess communication: ***shared memory*** and ***message passing***.

3.4.1 Shared Memory Communication:

Figure 3.9 illustrates how shared memory communication works in a bus-based system. Two components, such as a CPU and an I/O device, communicate through a shared memory location. The software on the CPU has been designed to know the address of the shared location.

The shared location has also been loaded into the proper register of the I/O device. If, as in the figure, the CPU wants to send data to the device, it writes to the shared location. The I/O device then reads the data from that location. The read and write operations are standard and can be encapsulated in a procedural interface.

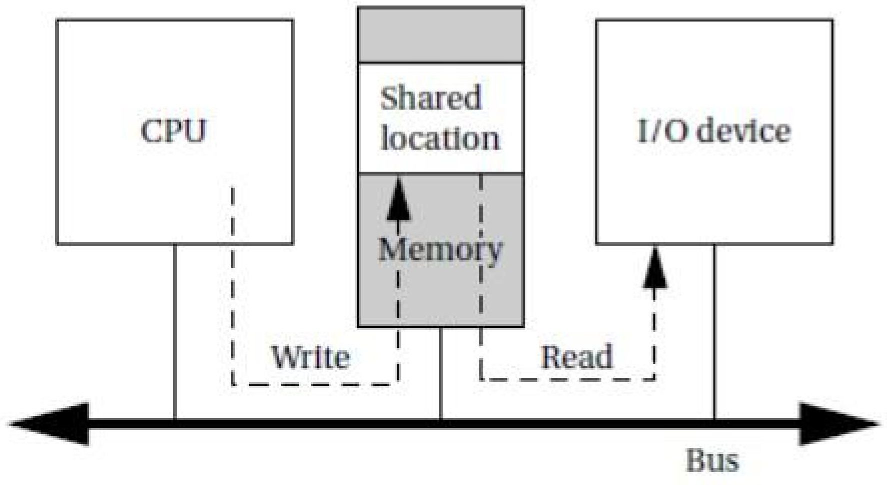


Fig 3.9 Shared memory communication implemented on a bus.

As an application of shared memory, let us consider the situation of Figure 6.14 in which the CPU and the I/O device want to communicate through a shared memory block. There must be a flag that tells the CPU when the data from the I/O device is ready.

The flag, an additional shared data location, has a value of 0 when the data are not ready and 1 when the data are ready. If the flag is used only by the CPU, then the flag can be implemented using a standard memory write operation. If the same flag is used for bidirectional signaling between the CPU and the I/O device, care must be taken. Consider the following scenario:

· CPU reads the flag location and sees that it is 0.

· I/O device reads the flag location and sees that it is 0.

· CPU sets the flag location to 1 and writes data to the shared location.

· I/O device erroneously sets the flag to 1 and overwrites the data left by the CPU.

3.4.2 Message Passing:

Message passing communication complements the shared memory model. As shown in Figure 3.10, each communicating entity has its own message send/receive unit. The message is not stored on the communications link, but rather at the senders/ receivers at the end points.

In contrast, shared memory communication can be seen as a memory block used as a communication device, in which all the data are stored in the communication link/memory.

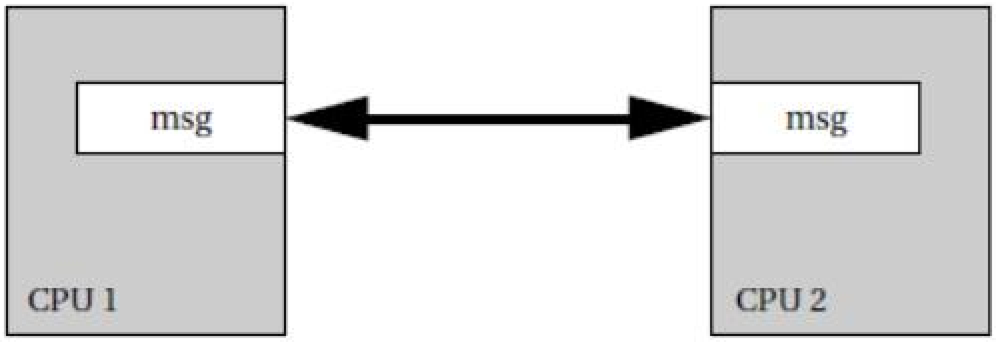


Fig 3.10 Message passing communication.

Applications in which units operate relatively autonomously are natural candidates for message passing communication. For example, a home control system has one microcontroller per household device—lamp, thermostat, faucet, appliance, and so on.

The devices must communicate relatively infrequently; furthermore, their physical separation is large enough that we would not naturally think of them as sharing a central pool of memory.

Passing communication packets among the devices is a natural way to describe coordination between these devices. Message passing is the natural implementation of communication in many 8-bit microcontrollers that do not normally operate with external memory.

3.4.3 Signals

Another form of interprocess communication commonly used in Unix is the ***signal***. A signal is simple because it does not pass data beyond the existence of the signal itself. A signal is analogous to an interrupt, but it is entirely a software creation. A signal is generated by a process and transmitted to another process by the operating system.

A UML signal is actually a generalization of the Unix signal. While a Unix signal carries no parameters other than a condition code, a UML signal is an object. As such, it can carry parameters as object attributes. Figure 3.11 shows the use of a signal in UML. The *sigbehavior* ( ) behavior of the class is responsible for throwing the signal, as indicated by <<*send*>>.The signal object is indicated by the <<*signal*>> stereotype.

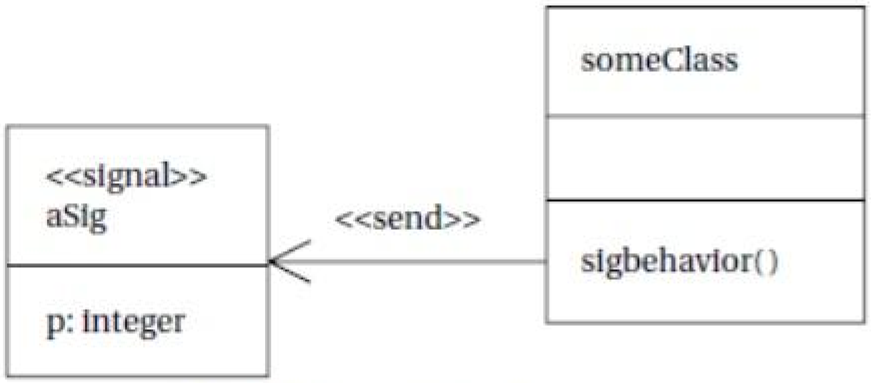


Fig 3.11 Use of a UML signal.

3.5 Evaluating Operating System Performance

The scheduling policy does not tell us all that we would like to know about the performance of a real system running processes. Our analysis of scheduling policies makes some simplifying assumptions:

1. We have assumed that context switches require zero time. Although it is often reasonable to neglect context switch time when it is much smaller than the process execution time, context switching can add significant delay in some cases.

2. We have assumed that we know the execution time of the processes. In fact, we learned in Section 5.6 that program time is not a single number, but can be bounded by worst-case and best-case execution times.

3. We probably determined worst-case or best-case times for the processes in isolation. But, in fact, they interact with each other in the cache. Cache conflicts among processes can drastically degrade process execution time.

The zero-time context switch assumption used in the analysis of RMS is not correct—we must execute instructions to save and restore context, and we must execute additional instructions to implement the scheduling policy. On the other hand, context switching can be implemented efficiently—context switching need not kill performance.

The effects of nonzero context switching time must be carefully analyzed in the context of a particular implementation to be sure that the predictions of an ideal scheduling policy are sufficiently accurate.

In most real-time operating systems, a context switch requires only a few hundred instructions, with only slightly more overhead for a simple real-time scheduler like RMS. When the overhead time is very small relative to the task periods, then the zero-time context switch assumption is often a reasonable approximation.

Problems are most likely to manifest themselves in the highest-rate processes, which are often the most critical in any case.

Completely checking that all deadlines will be met with nonzero context switching time requires checking all possible schedules for processes and including the context switch time at each preemption or process initiation. However, assuming an average number of context switches per process and computing CPU utilization can provide at least an estimate of how close the system is to CPU capacity.

POST TEST MCQs

1.What is Inter process communication?

a) allows processes to communicate and synchronize their actions when using the same address space

b) allows processes to communicate and synchronize their actions without using the same address space

c) allows the processes to only synchronize their actions without communication

d) none of the mentioned

2. Message passing system allows processes to __________

a) communicate with one another without resorting to shared data

b) communicate with one another by resorting to shared data

c) share data

d) name the recipient or sender of the message

3. Messages sent by a process __________

a) have to be of a fixed size

b) have to be a variable size

c) can be fixed or variable sized

d) None of the mentioned

4.Which of the following two operations are provided by the IPC facility?

a) write & delete message

b) delete & receive message

c) send & delete message

d) receive & send message

5. In a multi threaded environment _______.

a.Each thread is allocated with new memory from main memory

b.Main thread terminates after the termination of child threads

c.Every process can have only one thread

d.None of the above

6. A thread___________

a.is a lightweight process where the context swithching is high

b.is used to speed up paging

c.is a lightweight process where the context switching is low

d.none of the above

7.State whether true or false.

i) Multithreading is useful for application that perform a number of essentially independent tasks that do not be serialized.

ii) An example of multithreading is a database server that listens for and process numerous client request.

a.i-True, ii-False

b.i-True, ii-True

c.i-False, ii-True

d.i-False, ii-False

8. Which of the following Statement is TRUE for Shortest-Job-First Scheduling

a)1,2,4

b)1,2,3

c)1,3,4

d)1,2,3,4

9. The Priority Scheduling algorithm can be

a) preemptive Only

b) nonpreemptive Only

c) either preemptive or nonpreemptive

d) None of the above

10. Operating system's fundamental responsibility is to control the

a.Control of Processes

b.Access of Processes

c.Execution of Processes

d.Termination of Processes

Embedded Systems
Chapter - 4
EMBEDDED SOFTWARE

Pre-requisites : Basics of Embedded systems

Aim :

The students are expected to learn the various aspects in Embedded systems and its different types of systems with their respective developments. The inputs of the subject is to make students familiarize with the developmental types and functions of embedded systems.

Objectives :

The course should enable the students to

1. To gain the knowledge on the embedded systems development
2. To have a clear understanding of the different types of developments in embedded systems and their various types.

Outcomes :

At the end of the course student should be able to

1. Help to examine the definitions and internalize the need for understanding the various types of embedded systems.
2. Develop the responsible attitude towards the use of embedded systems as well as the technology
3. Able to envision the social impact on the products/projects they develop in their career.
4. Analyse the professional responsibility and empowering access to information in the workplace.

Pre-test MCQ type

1) Which system software is used to convert a "C" language program in to language of another processor?
   a)Compiler
   b)Linker
   c)Cross compiler
   d)Cross Linker

2) Embedded C programming language support ________ instructions of normal "C"language
   a)All
   b)Some
   c)Specific
   d)None

3) In CCS, The C is_______
   a)Compiler
   b)Linker
   c)Cross compiler
   d)Cross Linker

4) Cross Compiler converts
   a)Program into C language into binary language
   b)Programming C language into another language.
   c)Program in C language into program of another processors language
   d)Both A & B

5) Embedded system is designed to
   a) execute single program repeatedly
   b) execute many programs
   c) both
   d) none

6) Embedded system is
   a) Reactive
   b) Real time
   c) Proactive
   d) Reactive & Real time

7) Software written for embedded system is called
   a) Embedded Software
   b) system program
   c) operating system

8) Software for embedded system is written in

a) Flash ROM
b) RAM
c) EEPROM

9) Embedded system has

a) response time constraints
b) strict deadlines
c) turn around time
d) response time constraints & strict deadlines

10) Assembly code embedded within C programs is called

a) inline assembly code
b) External assembly code
c) Embedded Assembly code
d) Standard Assembly Code

11) Embedded C requires compilers to create files to be downloaded to the

a) microcontrollers
b) microprocessors
c) operating system
d) microcontrollers & microprocessors

12) Embedded C is used for

a) microcontrollers
b) desktop computers
c) laptops

13) Embedded systems are programmed using

a) Machine Code
b) Low level
c) High level
d) Machine Code, Low level, High level

14) Which software resides only in read only memory and is used to control products and systems for the consumer and industrial markets.

a) Business
b) Embedded
c) System
d) Personal

15) It is a characteristic provision of some debuggers to stop the execution after each
instruction because

a) it facilitates to analyze or vary the contents of memory and register
b) it facilitates to move the break point to a later point
c) it facilitates to rerun the program
d) it facilitates to load the object code program to system memory

4.1Programming embedded systems in assembly and C

- Assembly languages were developed to provide mnemonics or symbols for the machine level code instructions. Assembly language programs consist of mnemonics, thus they should be translated into machine code.
- A program that is responsible for this conversion is known as assembler. Assembly language is often termed as a low-level language because it directly works with the internal structure of the CPU.
- To program in assembly language, a programmer must know all the registers of the CPU.
- Different programming languages such as C, C++, Java and various other languages are called high-level languages because they do not deal with the internal details of a CPU.
- In contrast, an assembler is used to translate an assembly language program into machine code (sometimes also called object code or opcode).
- Similarly, a compiler translates a high-level language into machine code. For example, to write a program in C language, one must use a C compiler to translate the program into machine language.

4.1.1 Structure of Assembly Language

- An assembly language program is a series of statements, which are either assembly language instructions such as ADD and MOV, or statements called directives.
- An instruction tells the CPU what to do, while a directive (also called pseudo-instructions) gives instruction to the assembler. For example, ADD and

MOV instructions are commands which the CPU runs, while ORG and END are assembler directives.

- The assembler places the opcode to the memory location 0 when the ORG directive is used, while END indicates to the end of the source code. A program language instruction consists of the following four fields −

```
[ label: ]   mnemonics  [ operands ]   [;comment ]
```

A square bracket ([ ]) indicates that the field is optional.

The label field allows the program to refer to a line of code by name. The label fields cannot exceed a certain number of characters.

The mnemonics and operands fields together perform the real work of the program and accomplish the tasks. Statements like ADD A , C & MOV C, #68 where ADD and MOV are the mnemonics, which produce opcodes ; "A, C" and "C, #68" are operands.

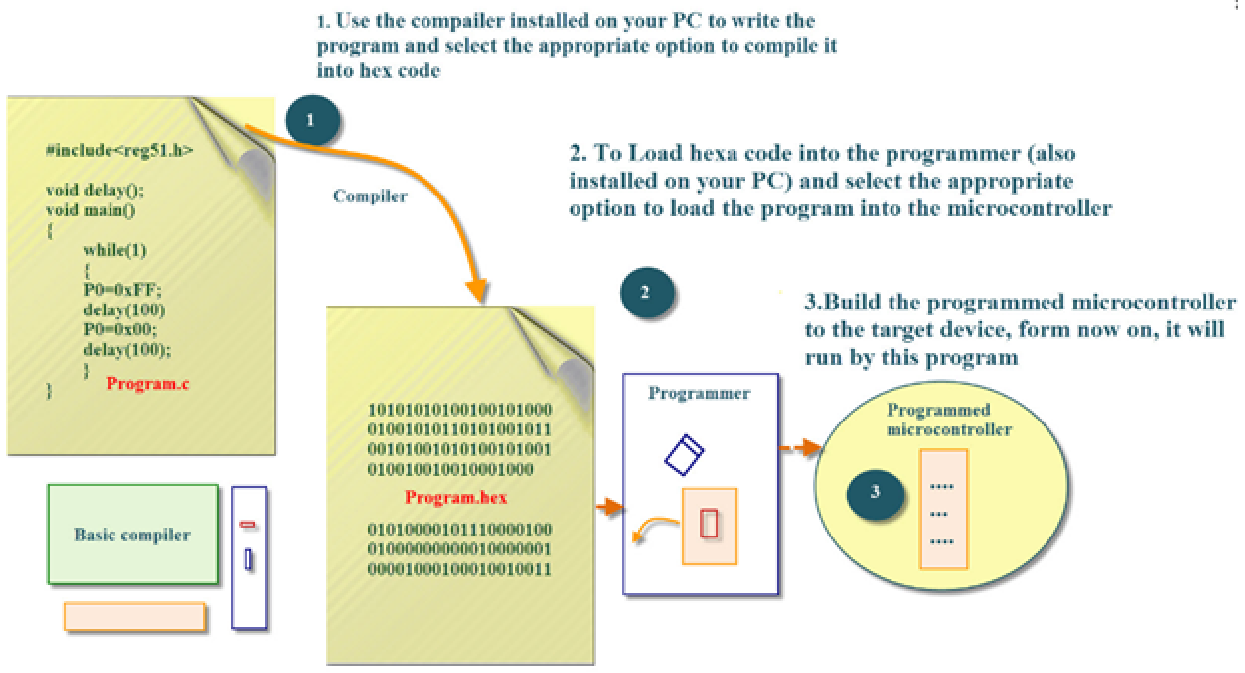
These two fields could contain directives. Directives do not generate machine code and are used only by the assembler, whereas instructions are translated into machine code for the CPU to execute.

```
1.0000       ORG  0H            ;start (origin) at location 0
2 0000 7D25    MOV  R5,#25H      ;load 25H into R5
3.0002 7F34    MOV  R7,#34H      ;load 34H into  R7
4.0004 7400    MOV  A,#0         ;load 0 into A
5.0006 2D      ADD  A,R5         ;add contents of R5 to A
6.0007 2F      ADD  A,R7         ;add contents of R7 to A
7.0008 2412    ADD  A,#12H       ;add to A value 12 H
8.000A 80FE    HERE: SJMP HERE    ;stay in this loop
9.000C END                      ;end of asm source file
```

- The comment field begins with a semicolon which is a comment indicator.
- Notice the Label "HERE" in the program. Any label which refers to an instruction should be followed by a colon.

4.1.2 Embedded C Programming with Keil Language

- ❏ Embedded C is the most popular programming language in the software field for developing electronic gadgets.
- ❏ Each processor used in an electronic system is associated with embedded software.
- ❏ Embedded C programming plays a key role in performing specific functions by the processor.
- ❏ In day-to-day life we used many electronic devices such as mobile phones, washing machines, digital cameras, etc. These all devices working are based on microcontrollers that are programmed by embedded C.
- ❏ Let's see the block diagram representation of embedded system programming:


1. Use the compailer installed on your PC to write the program and select the appropriate option to compile it into hex code
1
Compiler
2. To Load hexa code into the programmer (also installed on your PC) and select the appropriate option to load the program into the microcontroller
2
3.Build the programmed microcontroller to the target device, form now on, it will run by this program
Programmer
Programmed microcontroller
3
Basic compiler
Program.c
Program.hex

```
#include<reg51.h>

void delay();
void main()
{
    while(1)
    {
    P0=0xFF;
    delay(100)
    P0=0x00;
    delay(100);
    }
}
```

```
10101010100100101000
01001010110101001011
00101001010100101001
0100100100010000

01010000101110000100
01000000000010000001
00001000100010010011
```

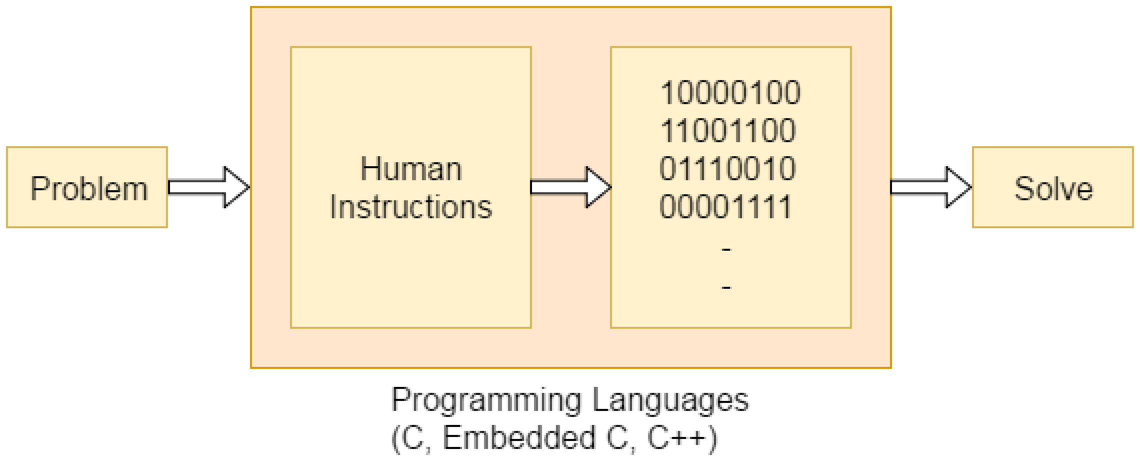
- ❏ The Embedded C code written in the above block diagram is used for blinking the LED connected with Port0 of the microcontroller.

- ➢ In embedded system programming C code is preferred over other language. Due to the following reasons:
- Easy to understand
- High Reliability
- Portability
- Scalability

4.1.3 Embedded System Programming:

Basic Declaration

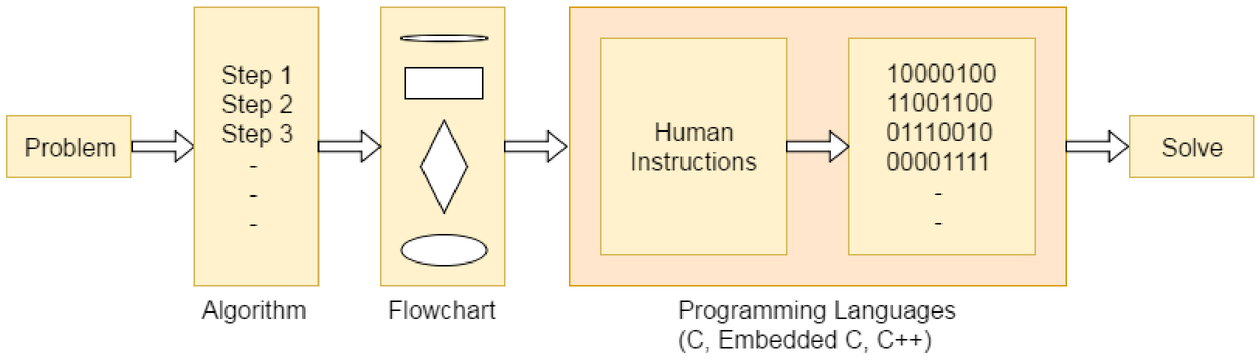
★ Let's see the block diagram of Embedded C Programming development:



- ➢ Function is a collection of statements that is used for performing a specific task and a collection of one or more functions is called a programming language. Every language is consisting of basic elements and grammatical rules.
- ➢ The C language programming is designed for function with variables, character set, data types, keywords, expression and so on are used for writing a C program.
- ➢ The extension in C language is known as embedded C programming language. As compared to above the embedded programming in C is also have some additional features like data types, keywords and header file etc is represented by

1. #include<microcontroller name.h>

Let's see the block diagram representation of Embedded C Programming Steps:



- The microcontroller programming is different for each type of operating system. Even though many operating systems exist such as Windows, Linux, RTOS, etc but RTOS has several advantages for embedded system development.

4.2 Meeting real time constraints

4.2.1 Timing constraints

- It is a vital attribute in real-time systems. Timing constraints decides the total correctness of the result sin real-time systems. The correctness of results in real-time system does not depends only on logical correctness but also the result should be obtained within the time constraint. There might be several events happening in real time system and these events are scheduled by schedulers using timing constraints.

4.2.1.1 Classification of Timing Constraints :

➢ Timing constraints associated with the real-time system is classified to identify the different types of timing constraints in a real-time system. Timing constraints are broadly classified into two categories:

1. Performance Constraints :

- The constraints enforced on the response of the system is known as Performance Constraints. This basically describes the overall performance of the system. This shows how quickly and accurately the system is responding. It ensures that the real-time system performs satisfactorily.

2. Behavioral Constraint :

- The constraints enforced on the stimuli generated by the environment is known as Behavioral Constraints.
- This basically describes the behavior of the environment. It ensures that the environment of a system is well behaved.

★ Further, the both performance and behavioral constraints are classified into three categories: **Delay Constraint, Deadline Constraint, and Duration Constraint**. These are explained as follows below.

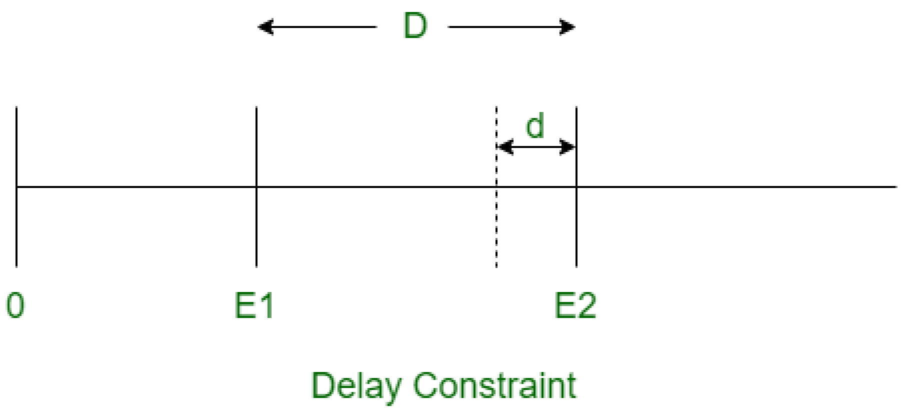
Delay Constraint –

❏ A delay constraint describes the minimum time interval between occurrence of two consecutive events in the real-time system.

❏ If an event occurs before the delay constraint, then it is called a delay violation. The time interval between occurrence of two events should be greater than or equal to delay constraint.

❏ If D is the actual time interval between occurrence of two events and d is the delay constraint, then

```
D >= d
```

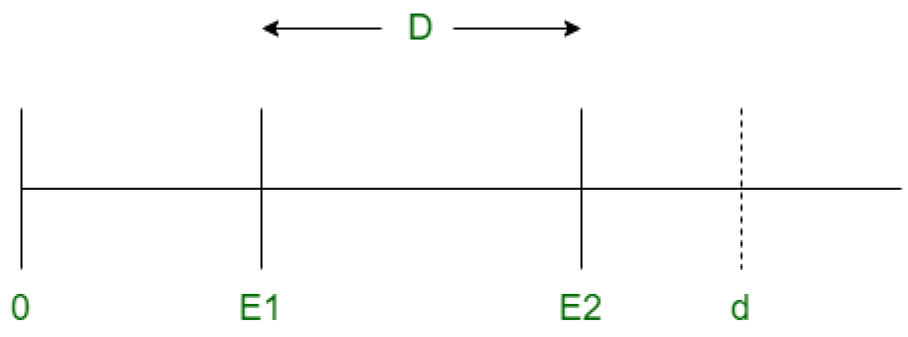


Deadline Constraint –

- A deadline constraint describes the maximum time interval between occurrence of two consecutive events in the real-time system.
- If an event occurs after the deadline constraint, then the result of event is considered incorrect. The time interval between occurrence of two events should be less than or equal to deadline constraint.

If D is the actual time interval between occurrence of two events and d is the deadline constraint, then

D <= d



Deadline Constraint

Duration Constraint –

- Duration constraint describes the duration of an event in real-time system. It describes the minimum and maximum time period of an event. On this basis it is further classified into two types:

  Minimum Duration Constraint: It describes that after the initiation of an event, it can not stop before a certain time.
  Maximum Duration Constraint: It describes that after the starting of an event, it must end before a certain time.

4.3 Embedded Operating System

- ➢ As the name suggests Embedded Operating System is an Embedded System's Operating System. It has limited features.
- ➢ It is usually designed for some particular operations to control an electronic device. For instance, all mobile phones essentially consist of an operating system that always boots up
- ➢ when the mobile phone is in running condition. It controls all the features and basic interface of the mobile phone.
- ➢ There are some other programs that can be loaded onto the mobile phones. Mostly, JAVA Apps run on the top. Embedded operating systems runs on **embedded processors**.

4.3.1 characteristics of Embedded Operating Systems

- ❑ The main characteristics of Embedded Operating Systems are as follows
    - Direct use of interrupts
    - Reactive operation
    - Real-time operation
    - Streamlined protection mechanisms
    - I/O device flexibility
    - Configurability
- ❖ There are two different kinds of operating systems, either a general purpose operating system that is modified in such a way that it runs on top of a device or the operating system can be custom written.
- ❖ The approaches for the design of operating systems include that either we take an embedded Operating System that is existing and adapt it to our embedded application or we can design and use a new operating system that is particularly for our Embedded System.
- ❖ We can adapt the existing Operating System to our embedded application by streamline operation, real-time capability and be adding other necessary functions. The advantage of this approach is that it has a familiar interface and its disadvantage is that it is not optimized for real-time.

4.3.2 Types of Embedded Operating Systems

4.3.2.1 Single System Control Loop

- ❑ Single system control loop is the simplest type of embedded operating system. It is so like operating system but it is designed to run the only single task.
- ❑ It still under debate that this system should be classified as a type of operating system or not.

4.3.2.2 Multi-Tasking Operating System

- ❏ As the name suggests that this operating system can perform multiple tasks. In multi-tasking operating system there are several tasks and processes that execute simultaneously.
- ❏ More than one function can be performed if the system has more than one core or processor.
- ❏ The operating system is switched between tasks. Some tasks wait for events while other receive events and become ready to run.
- ❏ If one is using a multitasking operating system, then software development is simplified because different components of software can be made independent to each other

4.4 Multi-state Systems and Function Sequences

4.4.1Multi-State System Definition:

- ➢ A system that can exist in multiple states (one state at a time) and transition from one state to another.

4.4.2 Characteristics

- ➢ A series of system states
- ➢ Each state requires one or more functions
- ➢ Rules exist to determine when to transition from one state to another

4.4.3 Typical Solution

- ➢ Every time tick, the system should check if it is time to transition to the next state
- ➢ When it is time to transition, appropriate control variables are updated to reflect the new state

4.4.4 Categories

- ➢ Timed Multi-state systems: Transitions depend only on time
- ➢ Input-based multi-state system: Transitions depend only on external input
- ➢ Not commonly used due to danger of indefinite wait

- ➢ Input-based/Timed Multi-state systems: Transitions depend both on external input and time

Example

Timed System: Traffic Light

- ➢ States: 1. Red 2. Red 3. Amber 4. Green 5. Ambe

Time Constants

```
#define RED_DURATION 20

#define RED_AND_AMBER_DURATION 5

#define GREEN_DURATION 30

#define AMBER_DURATION 5
```

State Update Code

```
 switch (Light_state_G)

{

case RED:

{

Red_light = ON;

 Amber_light = OFF;

Green_light = OFF;

if (++Time_in_state == RED_DURATION)

 {

Light_state_G = RED_AND_AMBER;
Time_in_state = 0;

 }

Break;

 }
```

POST-TEST- MCQ TYPE

1. Which component is replaced by an in-circuit emulator on the development board for testing purposes?
   a) RAM
   b) I/O Ports
   c) Micro-controller IC
   d) ROM

2. It is feasible for an in-circuit emulator to terminate at the middle of the program execution so as to examine the contents of _________
   a) memory
   b) registers
   c) memory & registers
   d) cache

3. Which operations are not feasible to perform by simulator programs in accordance to real time programming?
   a) Memory Operations
   b) I/ O Operations
   c) Register Operations
   d) Debugging Operations

4. Which software is used to control products and systems for the consumer and industrial
markets?
   a) System software
   b) Artificial intelligence software
   c) Embedded software
   d) Engineering and scientific software

5. The loops are interchangeable, in which design activity ?

   A. hardware/software partitioning
   B. **high-level transformation**
   C. scheduling
   D. compilation

6. Multiprogramming systems

   (A) Are used only on large mainframe computers
   (B) Are easier to develop than single programming systems
   (C) Execute more jobs at the same time
   (D) Execute each faster

7. The operating system determines the manner in which all of the following occurs except

   (A) Data displayed on the monitor
   (B) Printer output
   (C) User interaction with the processor
   (D) User creation of a document

8. Which of the following allows the programmer to define constants?
   a) pre-processor
   b) compiler
   c) emulator
   d) debugger

9. Which is the standard C compiler used for the UNIX systems?
   a) simulator
   b) compiler
   c) cc
   d) sc

10. Which of the following is also known as loader?
   a) locater
   b) linker
   c) assembler
   d) compiler

Embedded Systems
Chapter - 5
EMBEDDED SYSTEMS DEVELOPMENT

Pre-requisites : Basics of Embedded systems

Aim :

The students are expected to learn the various aspects in Embedded systems and its different types of systems with their respective developments. The inputs of the subject is to make students familiarize with the developmental types and functions of embedded systems.

Objectives :

The course should enable the students to

1. To gain the knowledge on the embedded systems development
2. To have a clear understanding of the different types of developments in embedded systems and their various types.

Outcomes :

At the end of the course student should be able to

1. Help to examine the definitions and internalize the need for understanding the various types of embedded systems.
2. Develop the responsible attitude towards the use of embedded systems as well as the technology
3. Able to envision the social impact on the products/projects they develop in their career.
4. Analyse the professional responsibility and empowering access to information in the workplace.

Pre-test MCQ type

1. Which system software is used to convert a "C" language program in to language of another processor?
   a)Compiler
   b)Linker
   c)Cross compiler
   d)Cross Linker

2)Embedded C programming language support ________ instructions of normal "C"language
   a)All
   b)Some
   c)Specific
   d)None

3)In CCS, The C is_______
   a)Compiler
   b)Linker
   c)Cross compiler
   d)Cross Linker

4) Cross Compiler converts
   a)Program into C language into binary language
   b)Programming C language into another language.
   c)Program in C language into program of another processors language
   d)Both A & B

5)Embedded system is designed to
   a) execute single program repeatedly
   b) execute many programs
   c) both
   d) none

6) Embedded system is
   a. Reactive
   b. Real time
   c. Proactive
   d. Reactive & Real time

7) Software written for embedded system is called
   a. Embedded Software
   b. system program
   c. operating system

8) Software for embedded system is written in

- **a. Flash ROM**
- b. RAM
- c. EEPROM
- d. ROM

9) Embedded system has

- a. response time constraints
- b. strict deadlines
- c. turn around time
- **d. response time constraints & strict deadlines**

10) Assembly code embedded within C programs is called

- **a. inline assembly code**
- b. External assembly code
- c. Embedded Assembly code
- d. Standard Assembly Code

11)Embedded C requires compilers to create files to be downloaded to the

- a. microcontrollers
- b. microprocessors
- c. operating system
- **d. microcontrollers & microprocessors**

12) Embedded C is used for

- **a. microcontrollers**
- b. desktop computers
- c. Laptops
- d. computers

13) Embedded systems are programmed using

- a. Machine Code
- b. Low level
- c. High level
- **d. Machine Code, Low level, High level**

14) Which software resides only in read only memory and is used to control products and systems for the consumer and industrial markets.

- a. Business
- **b. Embedded**

c. System
d. Personal

15) It is a characteristic provision of some debuggers to stop the execution after each instruction because

a. it facilitates to analyze or vary the contents of memory and register
b. it facilitates to move the break point to a later point
c. it facilitates to rerun the program
d. it facilitates to load the object code program to system memory

5.1 Embedded Software Development

- Embedded software is always a constituent of a larger system, for instance, a digital watch, a smartphone, a vehicle or automated industrial equipment.
- Such embedded systems must have real-time response under all circumstances within the time specified by design and operate under the condition of limited memory, processing power and energy supply.
- Moreover, embedded software must be immune to changes in its operating environment – processors, sensors, and hardware components may change over time.
- Other challenging requirements to embedded software are portability and autonomy.

Stability

- Stability is of paramount importance. Unexpected behavior from an embedded system is inadmissible and poses serious risks.
- End users demand that embedded systems must have uniform behavior under all circumstances and be able to operate durably without service.

Safety

- Safety is a special feature of embedded systems due to their primary application associated with lifesaving functionality in critical environments.

- Software Development Life Cycle (SDLC) for embedded software is characterized by more strict requirements and limitations in terms of quality, testing, and engineering expertise.

Security

- Security became a burning issue in the digital world.
- The related risks grow exponentially, especially so for IoT devices gaining popularity worldwide and becoming more interconnected to each other.
- Because modern home appliances like electric cookers, refrigerators and washing machines have connectivity features integrated by default, the Internet of Things now is exposed to a serious risk of hacking attacks.

Launch Phase

- Time-to-market and time-to-revenue have always been tough indicators in embedded system development, especially in the IoT segment.
- That is why the apps and platforms supposed to support zillions of IoT devices expected to appear by 2020 still are in their concept stage.
- Fabrication of hardware components housing embedded systems require extreme integration and flexibility due to very fast development of IoT industry.
- In addition, taking into account longer IoT device lifespan, future updates and releases become an issue for component designers.

Design Limitations

➢ The challenges in design of embedded systems have always been in the same limiting requirements for decades:
  - Small form factor;
  - Low energy;
  - Long-term stable performance without maintenance.

- ➢ The market demands from designers to pack more processing power and longer battery life into smaller spaces, which is often a tradeoff.
- ➢ Finally, depending on applications in IoT, there is a growing demand for manufacture of very scalable processor families ranging from cheap and ultra-low-power to maximum performance and highly configurable processors with forward-compatible instruction sets.
- ➢ There is similar demand for increased performance of system buses and internal/external memory caches.

Compatibility and Integrity

- Gartner Group estimation shows that, presently, most of the apps in the market are launched by businesses younger than 3 years old.
- With all their probable expertise in software development, many of them lack hands-on experience in implementing and updating their applications in IoT environment, especially with regard to security implications.
- Further expansion of IoT devices on the background of their connectivity puts more pressure on their adaptability.
- Users must be capable of administering the app through a simple user interface via all available channels including over-the-air firmware updates, which needs extreme compatibility across the entire ecosystem.
- Integrity becomes a function of security.
- To protect the IoT from malicious attacks or compromising, security must be implemented within each device at every level: the end node, gateway, cloud, etc.

5.1.1 Key Issues of Embedded Software Development

- Embedded solution developers are facing many specific issues along this way.
- We do not intend covering all of them in detail; let us have a look at a few of them

Connectivity

- There are so many different ways to connect device to the internet. Wireless connection can be established through Wi-Fi, Ethernet, Edge, LoRa, a Bluetooth bridge, and other channels.
- Leaving apart their pros and cons, the fact is each of them is created with a different technology stack, which means that developers have to have expertise in all of them.
- In addition, it is complicated with an issue, which protocol to use: UDP, COAP, TCP/IP, etc. or a few protocols at the same time plus TLS and/or MQTT on top of them.
- Pre-engineered software stacks partially resolve this complexity, but the issue remains for developers to have enough knowledge to understand a problem if something breaks or requires modification.

Over-the-air Updates

- The issue standing next to connection to the internet is remote updates of the firmware. In the case of standalone devices, it is enough to send updates to a secure site and notify users to download and install it.
- The situation is different with the IoT devices; the updates must be delivered and executed on their own without user's intervention. Now imagine that even a small IoT deployment involves a few thousand devices.
- Then, developers have to fulfill the following tasks: generate a firmware update, save it to the devices, validate that they are delivered from a trusted source, run the update on the devices at appropriate time, and be ready to roll back the update if there is an issue.
- This job is fairly tough and time-consuming, requires a lot of skill from developers, who have to be experienced in deploying the updates in the IoT environment.

Debugging

- Debugging is a general issue growing together with the number of connected devices – time and effort for debugging grows in parallel.
- Along with the process of open source software integration, there occurs more unexpected behaviors in a system adopting innumerous free flow devices than in the one, which was specifically designed to interact with them from the start.

- Many embedded software developers voice that every embedded project incurs extra cost for debugging consuming up to 40% of developers' time.

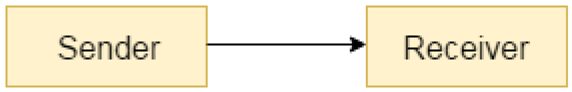
Pace of Change

- For a few decades, technologies used for embedded systems were almost the same with some new higher capacity processor to come out once a year.
- Then things eventually began speeding up. In the last 5 years, we witnessed fast development of emerging technologies including artificial intelligence.
- It creates a problem for developers, because available technologies are changing faster than they can get hold of them.

Embedded system development using serial interface

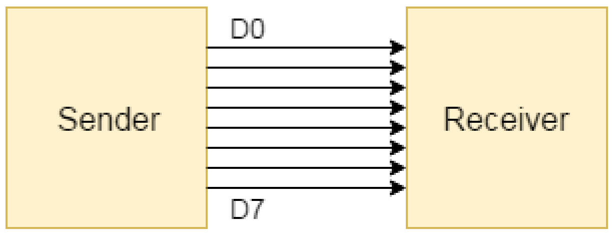
computer transfer data in two different ways:-

- **Serial transfer**: In serial transfer, data is transfer to device located many meters away this method is used for long distance data transfer.
- Let's see the block diagram of serial data transfer:

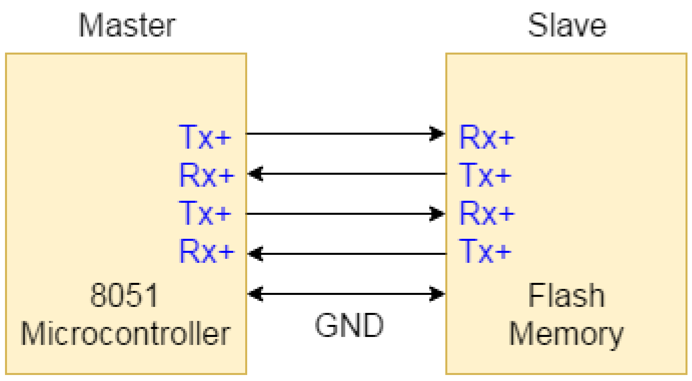


- Parallel transfer: In parallel transfer, data is transferred in 8 or more lines. In this wire conductor is used for transferring data to a device that is only a few feet away.

  Let's see the block diagram of parallel data transfer:



- Serial communication is mostly used for transmitting and receiving the signal. The 8051 microcontroller is consisting of Universal Asynchronous Receiver Transmitter (UART) used for serial communication. The signals are transmitted and received by the Rx and Tx pins of microcontroller.

- The UART take individual bytes of data and sends the individual bits in a sequential manner. The registers are used for collecting and storing the data inside a memory. UART is based on half-duplex protocol.
- Half-duplex means transferring and receiving the data, but not at the same time.
    - Let's see the block diagram representation of showing serial communication between flash memory and 8051 microcontroller:
    - 

    - Let's see the program for transmitting character 'S' using the serial window at baud rate of 9600:
    - Consider the 28800 is the maximum baud rate of the 8051 microcontroller For obtaining the 9600 as the baud rate, the timer value is,
    - $$\frac{28800}{9600} = 3$$

    This baud rate '3' is stored inside a timer.

    - #include<reg51.h>
    - void main()
    - {
    - SCON=0x50; //starting of a serial communication//
    - TMOD=0x20; //selected the timer mode//
    - TH1=3; // load the baud rate//
    - TR1=1; //Timer is ON//

```
SBUF='S';  //store the character inside a register//
while(TI==0);   //check the interrupt register//
TI=0;
TR1=0;      //OFF the timer//
while(1);  //continuous loop//
}
```

- ❏ Let's see the program for receiving the data from the HyperTerminal and sending of that data to PORT 0 of the microcontroller at 9600 baud rate:
- ❏ Consider the 28800 is the maximum baud rate of the 8051 microcontroller For obtaining the 9600 as the baud rate, the timer value is,

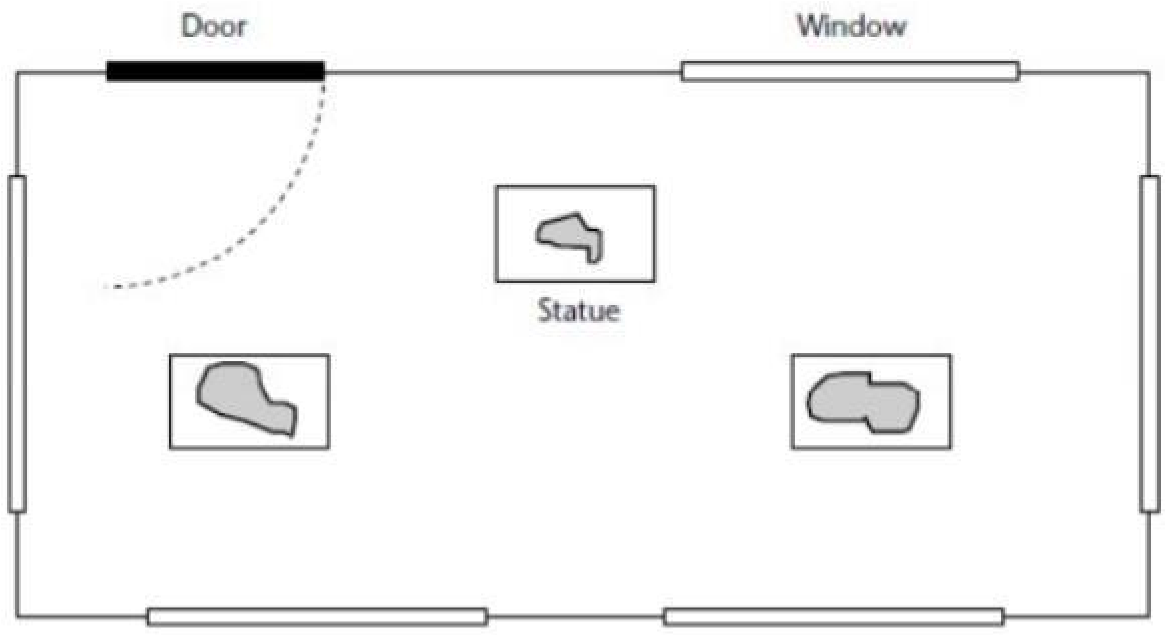
$$\frac{28800}{9600}=3$$

This baud rate '3' is stored inside a timer.

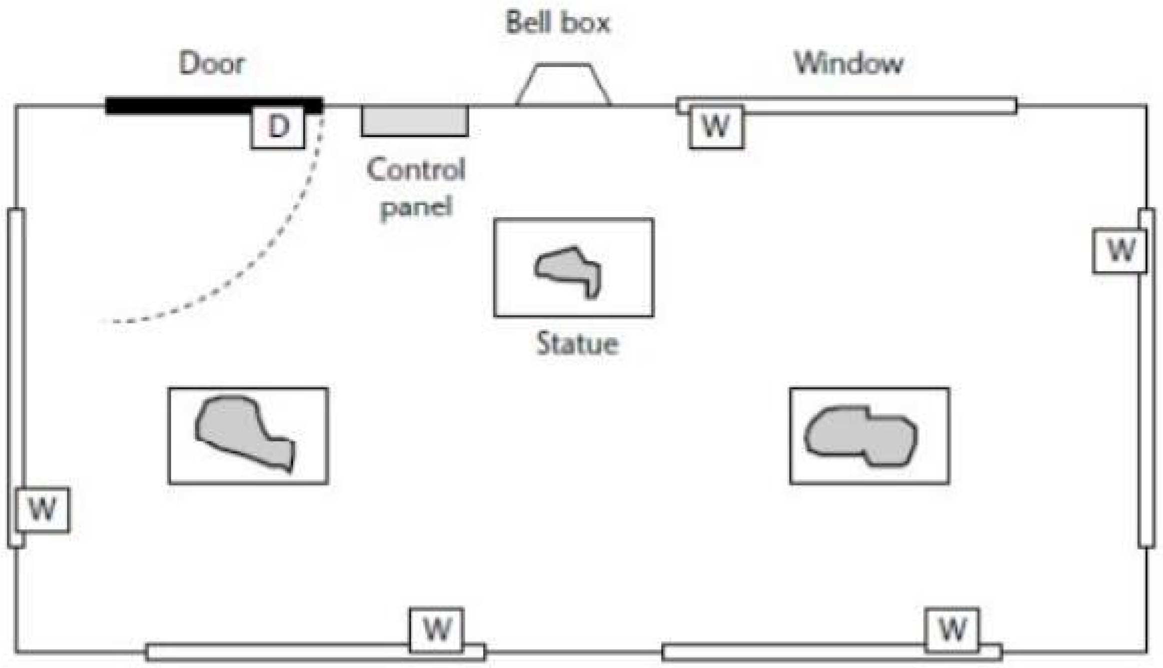
```
#include<reg51.h>
void main()
{
SCON=0x50;      //starting of a serial communication//
TMOD=0x20;  //selection of a timer mode//
TH1=3;       // load the baud rate//
TR1=1;      //Timer is ON//
PORT0=SBUF;  //send the data from SBUF to port0//
while(RI==0);       //checking of an interrupt register//
RI=0;
TR1=0;       //OFF the timer//
while(1); //stop the program when character is received//
}
```

5.2 Intruder Alarm System

- In this case study, we will consider the design and implementation of a small intruder alarm system suitable for detecting attempted thefts in a home or business environment Figure



- Figure shows the same gallery with the alarm system installed. In this figure, each of the windows has a sensor to detect class breakage. A magnetic sensor is also attached to the door. In each case, the sensors appear to be simple switches as far as the alarm system is concerned. Foll fig also shows a 'bell box' outside the property: this will sound if an intruder is detected



- Inside the door (in Figure 10.2), we have the alarm control panel: this consists mainly of a small keypad, plus an additional 'buzzer' to indicate that the alarm has sounded.
- The alarm system is designed in such a way that the user – having set the alarm by entering a four-digit password – has time to open the door and leave the room before the monitoring process starts. Similarly, if the user opens the door when the system is armed, he or she will have time to enter the password before the alarm begins to sound. When initially activated, the system is in 'Disarmed' state.
- In Disarmed state, the sensors are ignored. The alarm does not sound. The system remains in this state until the user enters a valid password via the keypad (in our demonstration system, the password is '1234'). When a valid password is entered, the systems enters 'Arming' state.
- In Arming state, the system waits for 60 seconds, to allow the user to leave the area before the monitoring process begins. After 60 seconds, the system enters 'Armed' state.
- In Armed state, the status of the various system sensors is monitored. If a window sensor is tripped,31 the system enters 'Intruder' state. If the door sensor is tripped, the system enters 'Disarming' state.

The keypad activity is also monitored: if a correct password is typed in, the system enters ‘Disarmed’ state.

In Disarming state, we assume that the door has been opened by someone who may be an authorized system user.

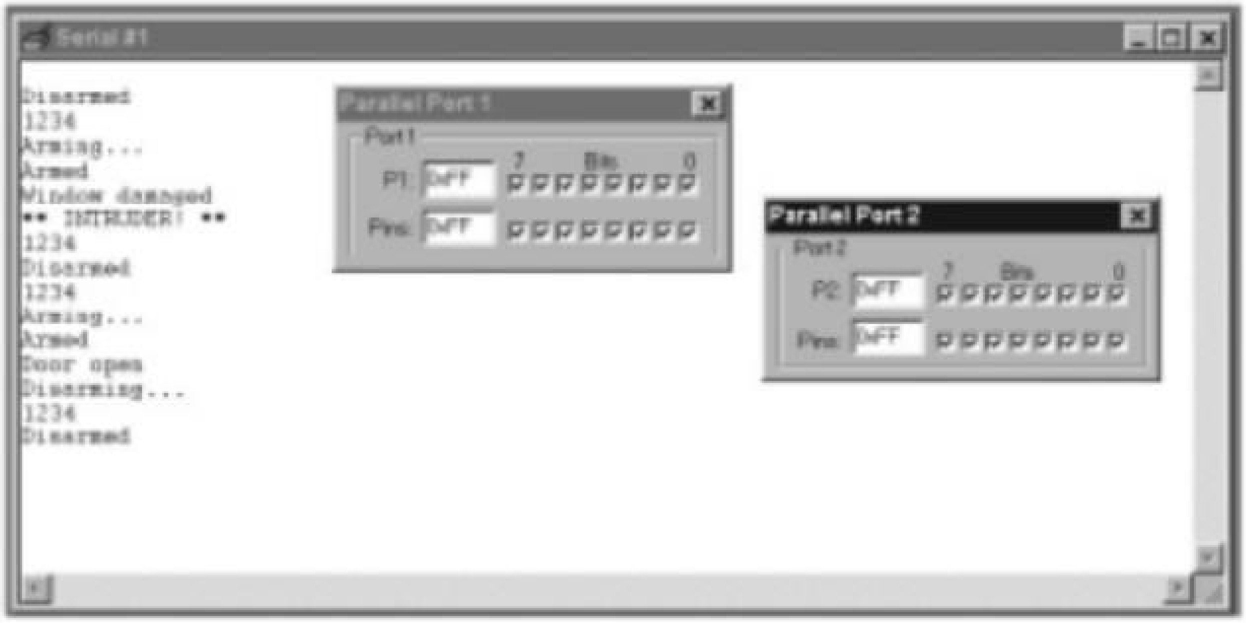
The system remains in this state for up to 60 seconds, after which – by default – it enters Intruder state. If, during the 60- second period, the user enters the correct password, the system enters ‘Disarmed’ state.

In Intruder state, an alarm will sound. The alarm will keep sounding (indefinitely), until the correct password is entered

The embedded operating system, sEOS,.

A simple ‘keypad’ library, based on a bank of switches.This final system would probably use at least 10 keys (see Figure 10.3): support for additional keys can be easily added if required.

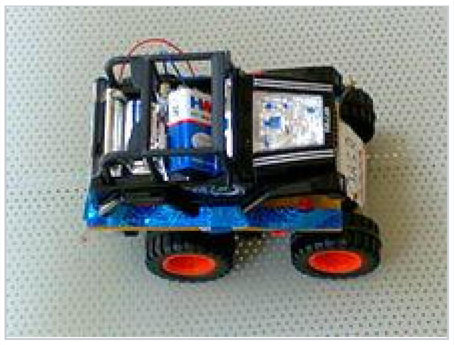
The RS-232 library Running the program



```
Disarmed
1234
Arming...
Armed
Window damaged
** INTRUDER! **
1234
Disarmed
1234
Arming...
Armed
Door open
Disarming...
1234
Disarmed
```

5.3 Controlling a Mobile Robot

- A mobile robot, is a robot that is capable of moving in the surrounding (locomotion). Mobile robotics is usually considered to be a subfield of robotics and information engineering.



A spying robot is an example of a mobile robot capable of movement in a given environment.

- Mobile robots have the capability to move around in their environment and are not fixed to one physical location. Mobile robots can be "autonomous" (AMR - autonomous mobile robot) which means they are capable of navigating an uncontrolled environment without the need for physical or electro-mechanical guidance devices.

- Alternatively, mobile robots can rely on guidance devices that allow them to travel a predefined navigation route in relatively controlled space (AGV - autonomous guided vehicle).
- By contrast, industrial robots are usually more-or-less stationary, consisting of a jointed arm (multi-linked manipulator) and gripper assembly (or end effector), attached to a fixed surface.
- Mobile robots have become more commonplace in commercial and industrial settings. Hospitals have been using autonomous mobile robots to move materials for many years. Warehouses have installed mobile robotic systems to efficiently move materials from stocking shelves to order fulfillment zones.
- Mobile robots are also a major focus of current research and almost every major university has one or more labs that focus on mobile robot research.[3] Mobile robots are also found in industrial, military and security settings.
- The components of a mobile robot are a controller, sensors, actuators and power system. The controller is generally a microprocessor, embedded microcontroller or a personal computer (PC).
- The sensors used are dependent upon the requirements of the robot. The requirements could be dead reckoning, tactile and proximity sensing, triangulation ranging, collision avoidance, position location and other specific applications.
- Actuators usually refer to the motors that move the robot can be wheeled or legged. To power a mobile robot usually we use DC power supply (which is battery) instead of AC.

POST-TEST- MCQ TYPE

1. Which of the following is approximated during hardware/software partitioning, during task level concurrency management?
   a) scheduling
   b) compilation
   c) task-level concurrency management
   d) high-level transformation

2. What does ICE stand for?
   a) in-circuit emulation
   b) in-code EPROM

c) in-circuit EPOM
d) in-code emulation

3. What are the major components of the intrusion detection system?
a) Analysis Engine
b) Event provider
c) Alert Database
d) All of the mentioned

4. Which are the serial ports of the IBM PC?
a) COM1
b) COM4 and COM1
c) COM1 and COM2
d) COM3

5. Which of the following can provide hardware handshaking?
a) RS232
b) Parallel port
c) Counter
d) Timer

6. Which is applied to a manufactured system?
a) bit pattern
b) parity pattern
c) test pattern
d) byte pattern

7. How is the quality of the test pattern evaluated?
a) fault coverage
b) test pattern
c) size of the test pattern
d) number of errors
Answer: a

8. What is DfT?
a) discrete Fourier transform
b) discrete for transaction
c) design for testability
d) design Fourier transform
Answer: c

9. What is CRC?
   a) code reducing check
   b) counter reducing check
   c) counting redundancy check
   d) cyclic redundancy check

   Answer: d

10. Which of the following is a meet-in-the-middle approach?
   a)peripheral based design
   b) platform based design
   c) memory based design
   d) processor design