SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT -I

EMBEDDED SYSTEM – SEC1320

I INTRODUCTION TO EMBEDDED SYSTEM

Embedded system- characteristics of embedded system- categories of embedded system- requirements of embedded systems- challenges and design issues of embedded system- trends in embedded system- system integration- hardware and software partition- applications of embedded system- control system and industrial automation-biomedical- data communication system-network information appliances- IVR systems- GPS systems

Introduction to Embedded system

An embedded system is one kind of a computer system mainly designed to perform several tasks like to access, process, store and also control the data in various electronics-based systems. Embedded systems are a combination of hardware and software where software is usually known as firmware that is embedded into the hardware. One of its most important characteristics of these systems is, it gives the o/p within the time limits. Embedded systems support to make the work more perfect and convenient. So, we frequently use embedded systems in simple and complex devices too. The applications of embedded systems mainly involve in our real life for several devices like microwave, calculators, TV remote control, home security and neighborhood traffic control systems

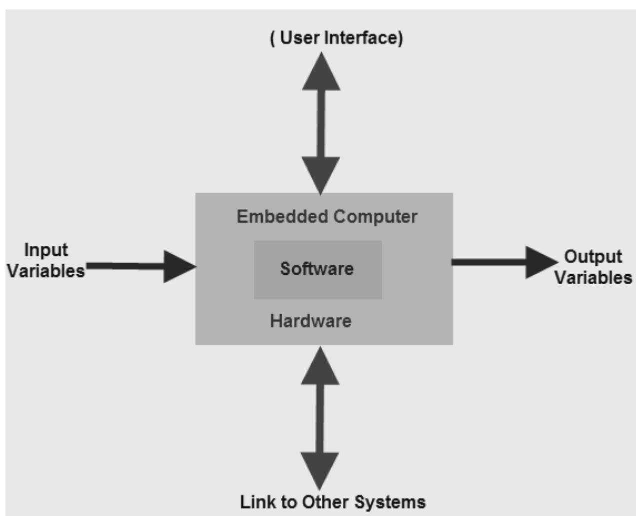


Fig:1.1:Block Diagram of Embedded system

The embedded system basics are the combination of embedded system hardware and embedded system software.

Embedded System Hardware

An embedded system uses a hardware platform to perform the operation. Hardware of the embedded system is assembled with a microprocessor/microcontroller. It has the elements

such as input/output interfaces, memory, user interface and the display unit. Generally, an embedded system comprises of the following

- **Power Supply**
- **Memory**
- **Processor**
- **Timers**
- **Output/Output circuits**
- **Serial communication ports**
- **SASC (System application specific circuits)**

Embedded System Software

The software of an embedded system is written to execute a particular function. It is normally written in a high-level setup and then compiled down to offer code that can be stuck within a non-volatile memory in the hardware. An embedded system software is intended to keep in view of the following three limits

- **Convenience of system memory**
- **Convenience of processor's speed**
- **When the embedded system runs constantly, there is a necessity to limit power dissipation for actions like run, stop and wake up.**

Embedded System Characteristics

- **Generally, an embedded system executes a particular operation and does the similar continually. For instance: A pager is constantly functioning as a pager.**
- **All the computing systems have limitations on design metrics, but those can be especially tight. Design metric is a measure of an execution features like size, power, cost and also performance.**
- **It must perform fast enough and consume less power to increase battery life.**
- **Several embedded systems should constantly react to changes in the system and also calculate particular results in real time without any delay. For instance, a car cruise controller; it continuously displays and responds to speed & brake sensors. It must calculate acceleration/de-accelerations frequently in a limited time; a delayed computation can consequence in letdown to control the car.**
- **It must be based on a microcontroller or microprocessor based.**
- **It must require a memory, as its software generally inserts in ROM. It does not require any secondary memories in the PC.**
- **It must need connected peripherals to attach input & output devices.**
- **An Embedded system is inbuilt with hardware and software where the hardware is used for security and performance and Software is used for more flexibility and features.**

Embedded System Applications

The applications of an embedded system basics include smart cards, computer networking, satellites, telecommunications, digital consumer electronics, missiles, etc.

Categories of Embedded System

Embedded systems are classified into four categories based on their performance and functional requirements:

- **Stand alone embedded systems**
- **Real time embedded systems**
- **Networked embedded systems**
- **Mobile embedded systems**

Embedded Systems are classified into three types based on the performance of the microcontroller such as

- **Small scale embedded systems**
- **Medium scale embedded systems**
- **Sophisticated embedded systems**

Stand Alone Embedded Systems

Stand alone embedded systems do not require a host system like a computer, it works by itself. It takes the input from the input ports either analog or digital and processes, calculates and converts the data and gives the resulting data through the connected device-Which either controls, drives and displays the connected devices. Examples for the stand alone embedded systems are mp3 players, digital cameras, video game consoles, microwave ovens and temperature measurement systems.

Real Time Embedded Systems

A real time embedded system is defined as, a system which gives a required o/p in a particular time. These types of embedded systems follow the time deadlines for completion of a task. Real time embedded systems are classified into two types such as soft and hard real time systems. Further this Real-Time Embedded System is divided into two types i.e.

Soft Real Time Embedded Systems –

In these types of embedded systems time/deadline is not so strictly followed. If deadline of the task is passed (means the system didn't give result in the defined time) still result or output is accepted.

Hard Real-Time Embedded Systems –

In these types of embedded systems time/deadline of task is strictly followed. Task must be completed in between time frame (defined time interval) otherwise result/output may not be accepted.

Examples

- **Traffic control system**
- **Military usage in defense sector**
- **Medical usage in health sector**

Networked Embedded Systems

These types of embedded systems are related to a network to access the resources. The connected network can be LAN, WAN or the internet. The connection can be any wired or wireless. This type of embedded system is the fastest growing area in embedded system applications. The embedded web server is a type of system wherein all embedded devices are connected to a web server and accessed and controlled by a web browser.Example for the LAN networked embedded system is a home security system wherein all sensors are connected and run on the protocol TCP/IP

Mobile Embedded Systems

Mobile embedded systems are used in portable embedded devices like cell phones, mobiles, digital cameras, mp3 players and personal digital assistants, etc.The basic limitation of these devices is the other resources and limitation of memory.

Small Scale Embedded Systems

These types of embedded systems are designed with a single 8 or 16-bit microcontroller, that may even be activated by a battery. For developing embedded software for small scale embedded systems, the main programming tools are an editor, assembler, cross assembler and integrated development environment (IDE).

Medium Scale Embedded Systems

These types of embedded systems design with a single or 16 or 32 bit microcontroller, RISCs or DSPs. These types of embedded systems have both hardware and software complexities. For developing embedded software for medium scale embedded systems, the main programming tools are C, C++, JAVA, Visual C++, RTOS, debugger, source code engineering tool, simulator and IDE.

Sophisticated Embedded Systems

These types of embedded systems have enormous hardware and software complexities, that may need ASIPs, IPs, PLAs, scalable or configurable processors. They are used for cutting-edge applications that need hardware and software Co-design and components which have to assemble in the final system.

Requirements of Embedded system

Reliability
Cost-effectiveness
Low power consumption
Efficient use of processing power
Efficient use of memory

Appropriate execution time

Reliability

Embedded system have to work without the need for resetting or rebooting. This call for a very reliable hardware and software. For example : if an embedded system comes to a halt because of hardware error, the system must reset itself without the need for human intervention. However the embedded software developers must make the reliability of the hardware as well as that of the software

Cost-effectiveness

If an embedded system is designed for a very special purpose such as for deep space or for nuclear power plant station cost may not be an issue. However if the embedded system is designed for a mass market purpose like CD players, toys and mobile devices cost is a major concern. Application Specific Integrated Circuit (ASIC) is used by the designers to reduce the hardware components and hence the cost

Low power consumption

Most of the embedded systems are powered by battery, rather than a main supply. In such case the power consumption should be minimized to avoid draining the Batteries. For example : by reducing the number of hardware component the power consumption can be reduced. As well as by designing the processor to revert to low power or sleep mode when there is no operation to perform

Efficient use of processing power

A wide variety of processors with varying processing powers are available to embedded systems. Developers must keep processing power, memory and cost in mind while choosing the right processor. The processing power requirement is specified in ,Million Instruction Per Second (MIPS). With the availability of so many processor, choosing a processor has become a tough task nowadays

Efficient use of memory

Most of the embedded systems do not have secondary storage such as hard disk. The memory chip available on the embedded systems are only Read Only memory and Random Access memory. As most of the embedded systems do not have secondary storage, "flash memory" is used to store the program. Nowadays micro-controller and Digital signal processors also comes with onboard memory. Such processors are used for small embedded system as the cost generally is low and the execution generally is fast

Appropriate execution time

In real time embedded systems, certain task must be performed within a specified period of time. Normally desktop pc cannot achieve real time performance. Therefore,

special operating system known as real time operating systems run on these embedded systems. In hard real time embedded system deadlines has to be strictly met but whereas in soft real time embedded system the task may not be performed in a timely manner. The software developer needs to ascertain whether the embedded system is a hard real time or soft one and has to perform the performance analysis accordingly

Challenges and issues in embedded system

Co-design
Embedding an operating system
Code optimization
Efficient input or output
Testing and debugging

Co-design

An embedded system consists of hardware and software, deciding which function of the system should be implemented in hardware and software is of a major consideration. For example in hardware implementation the task execution is faster compared with the other one. On the downside a chip cost money, consumes valuable power and occupies space. A software implementation is better if these are the major concern. This issue of choosing between hardware and software implementation is known as a co-design issue

Embedding an operating system

It is possible to write embedded software without any operating system embedded into the system. Developers can implement services such as memory management, input/output management and so on. Writing your own routines necessary for a particular application results in compact and efficient coding. Embedded operating system provide the necessary Application Programming Interfaces(API)

Code optimization

Developers need not worry much about the code optimization, because the processor is highly powerful, plenty of memory is generally available. Memory and Execution time are the important constraints in embedded system. Sometimes to achieve the required response time the programmer has to write certain portion of coding in assembly language. Of course, with the availability of sophisticated development tools, this is less of an issue in recent years

Efficient input or output

In most of the embedded system, the input interfaces have limited functionality. Writing embedded software is a different ball game compared with writing a user interface with a full-fledged keyboard, a mouse and a large display. Many systems available in process control take electrical signal as input and produce electrical signal as output, since they don't use I/O devices. Developing, testing and debugging such systems is much more challenging than doing the same with the desktop systems.

Testing and debugging

Software for an embedded system cannot be tested on the target hardware during the development phase because debugging will be extremely difficult. Testing and debugging the software on the host system by actual simulation of field conditions is very challenging. Nowadays, the job is made a bits simpler with the availability of "profilers" that tell you clearly which line of code are executed and which lines are not executed. Using the output of such profilers we can locate the untested lines of code and ensure that they are also executed by providing the necessary test input data. It is these challenges that made embedded software development a "black art" in earlier days. This is no longer the case, however the developments in embedded software are changing the scenario completely.

Recent trends in embedded system

Processors
Memory
Operating Systems
Programming Languages
Development Tools

Processors

In an effort to cater to different applications, several semiconductor electronics vendors have released many processors. We can find 8-bit, 16-bit, and 32 bit processors with different processing powers and memory addressing capabilities. Many sophisticated DSP are available to cater to numerous application needs including audio and video coding and image processing. The processor boards around which the embedded systems can be built come with the necessary RAM and ROM as well as peripherals such as a serial port, USB port and Ethernet connectivity

Memory

Both RAM and ROM memory devices are becoming increasingly cheaper paving the way for devices that can store large numbers of programs and their data. Secondary devices such as Hard disk are also being integrated into embedded systems such as mobile communication and computing devices . Devices that do not have secondary storage use flash memory and the capacity of flash memory chips is also rising very rapidly making it possible to incorporate heavy OS

Operating Systems

As most everyone knows Microsoft currently holds the lion share of the market in operating systems that run desktop computers. Many operating systems which are available nower days are categorized as embedded operating systems, real time operating systems and mobile operating system. These operating system occupies much less memory. This reduces the development time and the effort considerably

Programming Languages

The era of writing the embedded software in assembly languages is now almost history. High level languages are extensively used for embedded software development. Object oriented programming languages are also extensively used. Another important development is the use of JAVA. Because of JAVA platform independence it has become very popular for embedded software development

Development Tools

Many advances in development tools are accelerating embedded software development. These development tools include Cross compiler, Debbugers and Emulators. Using these tools developers can write programs on host machines, test the software thoroughly and port to the target hardware. The cycle time for the development has been reduced considerably in recent years because of these development tools. Many of them are available free of cost from major software vendors

Control System and Industrial Automation

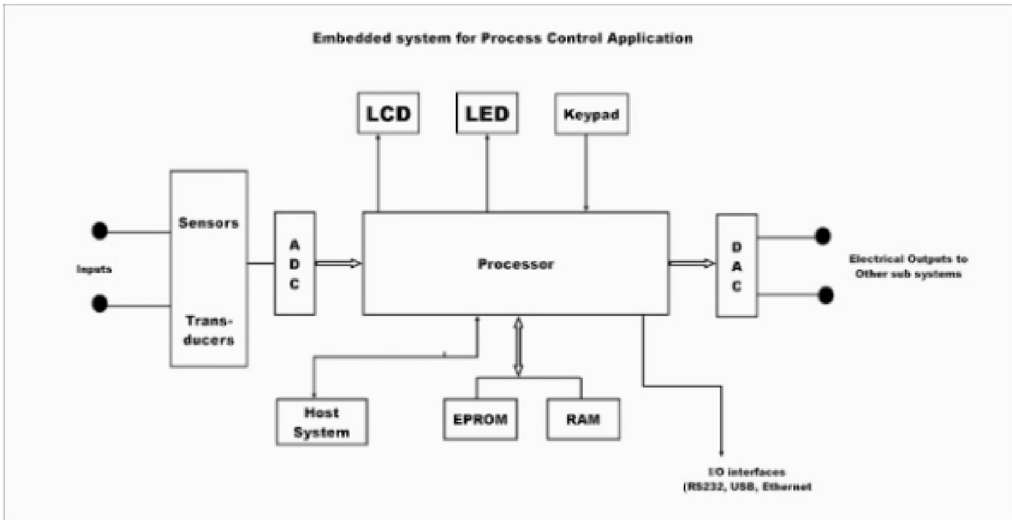


Fig:1.2:Block Diagram of control system and industrial automation

The embedded system takes electrical signals as input. Generally sensors or transducers are used to convert the physical entity into an electrical signal. The processor can process only digital signals, the ADC(Analog to Digital Convertor) converts the analog signals to its equivalent digital signals, which is an electrical representation of a bit stream of 0s and 1s. RAM is used to store the volatile data. A DAC(Digital to Analog Convertor) is used to convert the output digital signal to analog format. The processor board also includes input/output interface, such as serial interface , USB port and an Ethernet port for connectivity to the external systems. For the user interaction LCD and LED and a keypad are provided. These modules may or may not be required depending on the application. Depending on the application the designer chooses the necessary modules and carries out the design. While designing the reliability, performance and the cost need to be kept in the mind. Some of the typical process control applications in nuclear plants and telemetry and tele command units in satellite communication systems. Some of the embedded systems have to operate in very hostile environments

Biomedical systems

Much of the progress made in the health care industry is due to the development in the electronic industry Hospitals are full of embedded systems, including X-ray control units, EEG and ECG units and equipments used for diagnostic testing such as endoscopy and so on. These systems use PC add on cards which take the ECG signals and process them and the PC monitor is used for the display. Even the PC secondary storage is used to store the ECG records. Biometric systems for finger print and face recognition are gaining wide use in the agencies concerned with the securities

The input fingerprint must be processed and compared with the available database using pattern recognition algorithm, which requires intensive processing. The biometric systems use a Digital Signal Processor(DSP) for signal processing such as filtering and edge enhancement of the image. And a general purpose processor for implementing the pattern matching algorithms

Data communication system

Internet has acted as the catalyst for the embedded system. Modem that connects two computers is an embedded system. Dialup modems normally used to access the internet are embedded systems with a DSP inside. Using the DSP and the associated software the modem establishes the connection using the standard protocols. As the digital signal is modulated a lot of signal processing is involved therefore, DSP is used.

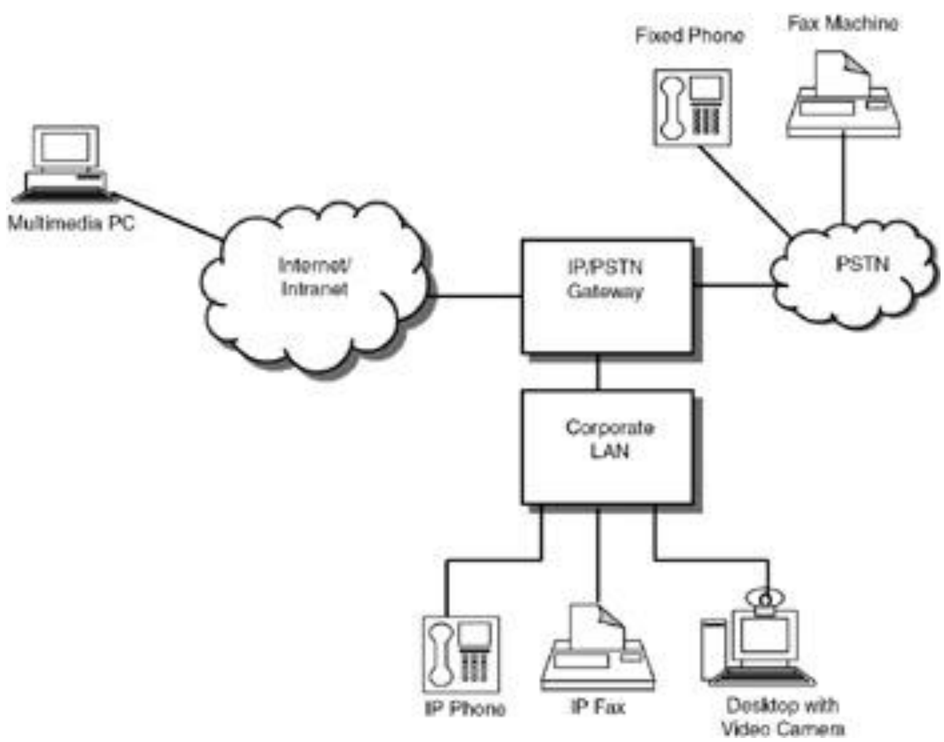


Fig:1.3:Multimedia Communication over IP Network

"Convergence" is the mantra nowadays. For years we used different networks for different services. Telephone network(PSTN) for making voice call and sending fax messages. Internet for data rate services such as email, file transfer and web services. WAN act as the backbone network supporting the data, voice and video communication services

Telecommunication

Telecommunication infrastructure element includes networking components such as Telephone switches, Loop carriers, terminal adapters, ATM switches, frame relays and so on. Mobile communication components includes base station, Mobile switching centers and so on. Satellite communication equipment includes earth station controller, onboard processing elements, telemetry and so on

Audio codec

Normally when voice is transmitted over the telephone network the voice is coded at the rate of 64kbps using a technique known as PCM. In radio system speech is compressed to save the bandwidth. At the transmitting side the audio signal are compressed to achieve data rates and at the receiving end the audio signal is expanded to retrieve the original Signal. These codecs use DSP extensively and gets embedded into cell phones and equipment of mobile and fixed communication systems. MP3 player is a good example, where are the signals are encoded and transmitted from a music kiosk to be played on the MP3 device

Automatic speech generation system

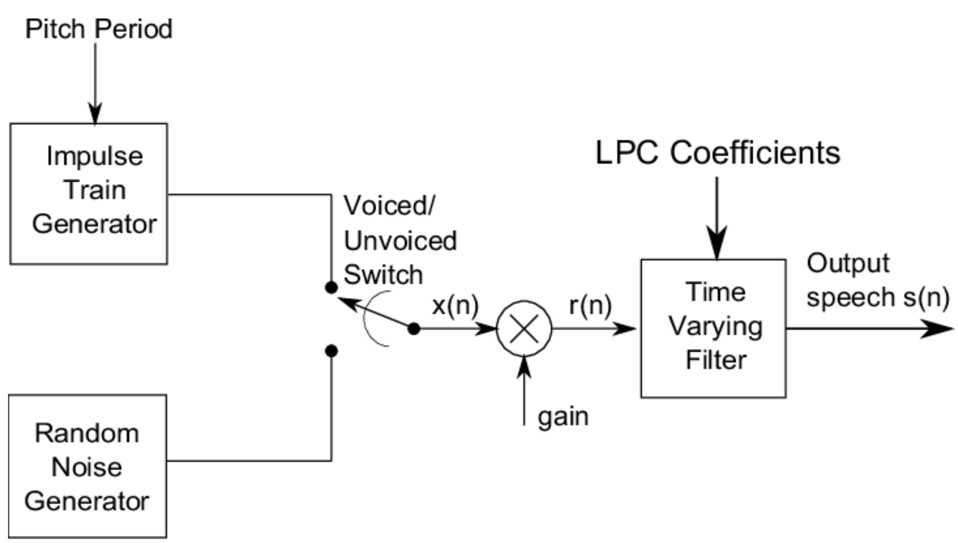


Fig:1.4: Human speech model

Video codec

Video conferencing has become very popular in recent years. Video occupies very large bandwidth however and to transmit video over the internet, video signals must be compressed to reduce the data Standards such as MPEG and JPEG are used to achieve

video compression. To compress the video signal a video coder is used and to bring to the original signal a decoder is used. These embedded systems use DSP to implement video compression Algorithm

IVR System

It is a stand alone embedded system connected to the computer through a parallel port or USB port or it can be implemented on a PC with an add on card. IVR system is an embedded system connected to the computer holding the bank database. IVR system also has a telephone interface and it is connected in parallel to a telephone line. Once the bank assigns a specific number to the IVR system any subscriber can call this number to get the information about his/her bank account details. IVR system comprises of PSTN interface, ADC and DAC, S to P and P to S Convertors and an interface circuitry with microphone and speaker PSTN interface receives the telephone calls and answers them. Filters limit the audio signal to the desired frequency band up to 4khz. ADC converts input to digital format and digitized voice data is converted to parallel format using S to P convertor and vice versa. FIFO are buffers that temporarily holds the speech data. An IC MT8880 is used. Using this technology coupled with speech recognition and speech synthesis we can develop applications to browse the web through voice commands

GPS System

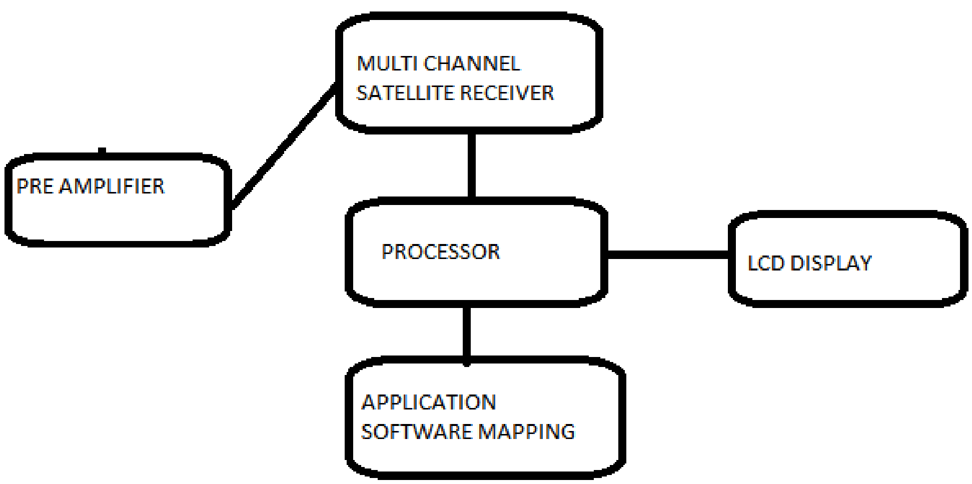


Fig:1.5:Block Diagram of GPS Receiver

It is a gift from the U.S from DOD to the humankind. Using a set of 24 NAVSTAR satellites, the DOD provides the GPS service for any moving or fixed object. A GPS receiver receives the satellite signals and process them to find the position parameters of the GPS receiver location. GPS receiver is a powerful embedded system that uses a DSP to process the satellite signals. GPS receiver computes its latitude, longitude, altitude, velocity and so on. It has an RS 232 serial communication interface or a USB interface from which the position data is available

TEXT / REFERENCE BOOKS

1. KVKK Prasad, "Embedded / Real Time Systems", Dreamtech Press, 2005.

2. David Simon, "An Embedded Software Primer", Pearson Education Asia, First Indian Reprint 2000.

3. Raj Kamal, 'Embedded system-Architecture, Programming, Design', Tata McGraw Hill, 2011.

4. Arnold Berger, "Embedded system design", CMP books, 1st Edition, 2005

5. Wayne Wolf, "Computers as components", Morgan Kaufmann publishers, 2nd Edition, 2008.

6. Tammy Noergaard, "Embedded Systems Architecture", Elsevier, 2006

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT -II

EMBEDDED SYSTEM – SEC1320

II EMBEDDED SOFTWARE DEVELOPMENT AND TOOLS

Software architectures, Round - Robin, Round-Robin with Interrupts, Function Queue Scheduling architecture, Introduction to assembler - Compiler -Cross compilers and Integrated Development Environment IDE, Linker/ Locators, Simulators, Getting Embedded software into target System- Debugging Strategies

A system that must respond rapid1y to many different events and that has various processing requirements, all with different deadlines and different priorities, will require a more complex architecture. We will discuss four architectures, starting with the simplest one, which others you practically no control of your response and priorities, and moving on to others that give you greater control but at the cost of increased complexity. The four are round-robin, round-robin with interrupts, function-queue-scheduling, and real-time operating system

Round-Robin Architecture

The simplest possible software architecture is called "round robin." Round robin architecture has no interrupts; the software organization consists of one main loop wherein the processor simply polls each attached device in turn, and provides service if any is required. After all devices have been serviced, start over from the top. One can think of many examples where round robin is a perfectly capable architecture: A vending machine, ATM, or household appliance such as a microwave oven (check for a button push, decrement timer, update display and start over). Basically, anything where the processor has plenty of time to get around the loop, and the user won't notice the delay. The main advantage to round robin is that it's very simple, and often it's good enough. On the other hand, If a device has to be serviced in less time than it takes the processor to get around the loop, then it won't work. The worst case response time for round robin is the sum of the execution times for all of the task code

- **Very simple**
- **No interrupts**
- **No shared data „**
- **No latency concerns**

Main loop: „ checks each I/O device in turn „ services any device requests „
E.g.: Digital Multimeter

Round-Robin Architecture

```
Void main( void)
{
  while(TRUE)
        {
          if (!! I/O Device A needs service)
```

```
                    {
                     !!Take care of I/O Device A
                     !! Handle data to or from I/O Device A
                    }
                    if (!! I/O Device B needs service)
                    {
                     !!Take care of I/O Device B
                     !! Handle data to or from I/O Device B
                    }
etc.
etc.
           if (!! I/O Device Z needs service)
                    {
                     !!Take care of I/O Device Z
                     !! Handle data to or from I/O Device Z
                    }
          }
}
```

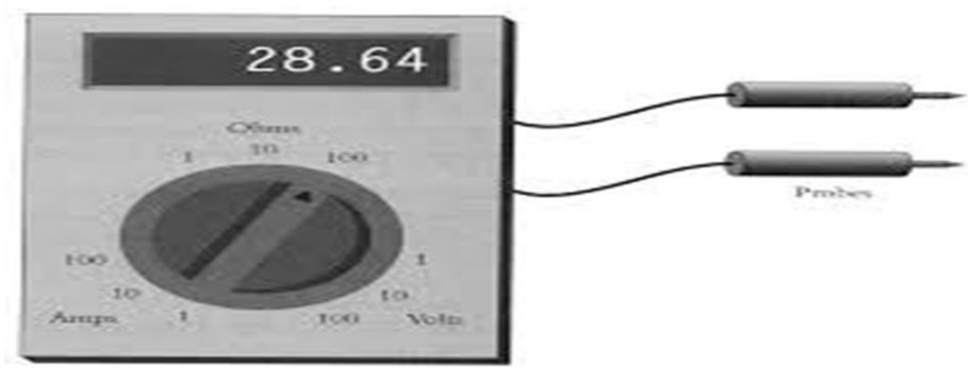


Fig:2.1: Digital Multimeter

This is a marvelously simple architecture-no interrupts, no shared data, no latency concerns—and therefore always an attractive potential architecture, as long as you can get away with it. Simple as it is, the round-robin architecture is adequate fot some jobs. Consider, for example, a digital multimeter such as the one shown in Figure. A digital multimeter measures electrical resistance, current, and potential in units of ohms, amps, and volts, each in several different ranges. A typical multimeter has two probes that the user touches to two points on the circuit to be measured, a digital display, and a big rotary switch that selects which measurement to make and in what range. The system makes continuous measurements and changes the display to reflect the most recent measurement. Each time around its loop, it checks the position of the rotary switch and then branches to code to make the appropriate measurement, to format its results, and to write the results to the display. Even a very modest microprocessor can go around this loop many times each second.

Code for Digital Multimeter

```
Void v Digital Multimeter Main( void)
        {
          enum{OHMS_1, OHMS_10,…..VOLTS_100} eswitchPosition;
          while(TRUE)
          {
           eswitchPosition = !! Read the position of the switch;
          switch(eswitch position)
                  {
                   case OHMS_1;
                  !!Read hardware to measure ohms
                  !! Format result
                  break;
                   }
        case OHMS_10;
                  !!Read hardware to measure ohms
                  !! Format result
                  break;
                   .
                   .
                   .
        case VOLTS_100;
                  !!Read hardware to measure volts
                  !! Format result
                  break;
          }
          !!write result to display
              }
         }
```

- **This architecture is fragile.**
- **Even if you manage tune it up so that the microprocessor gets around the loop quickly enough to satisfy all the requirements**
- **A single additional device or requirement may break everything**
- **Because of these shortcomings, a round robin architecture is probably suitable only for very simple devices such as digital watches and microwave ovens**

Round-Robin with Interrupt Architecture

This is somewhat more sophisticated architecture, which we will call as round robin with interrupts. In this architecture interrupt routines deal with very urgent needs of the hardware and then set flags. The main loop polls the flags and does not follow up processing required by the interrupts. This interrupt gives a little more control over priorities. The interrupt routines can get good response because the hardware interrupt signal causes the microprocessor to stop whatever it is doing in the main function and execute the interrupt routine instead.

Effectively, all of the processing that you put into the interrupt routines has a higher priority than the task code in the main routine. Further, since you can usually assign priorities to the various interrupts in your system.

In this, urgent tasks get handled in an interrupt service routine, possibly with a flag set for follow-up processing in the main loop. If nothing urgent happens (emergency stop button pushed, or intruder detected), then the processor continues to operate round robin, managing more mundane tasks in order around the loop. The obvious advantage to round robin with interrupts is that the response time to high-priority tasks is improved, since the ISR always has priority over the main loop (the main loop will always stop whatever it's doing to service the interrupt), and yet it remains fairly simple. The worst case response time for a low priority task is the sum of the execution times for all of the code in the main loop plus all of the interrupt service routines.

Fig:2.2: Priority levels for Round robin Architecture

It offers more control over priorities via hardware interrupts . Interrupt handlers implement higher priority functions (allowing the assignment of levels of priority among devices/handlers). The handlers set flags, which are polled by the task code to continue when the handlers complete their job

- Advantage: Setting and controlling using priorities
- Disadvantage: Danger of having shared data Priorities set in hardware
- Round robin with interrupt –A simple communication bridge

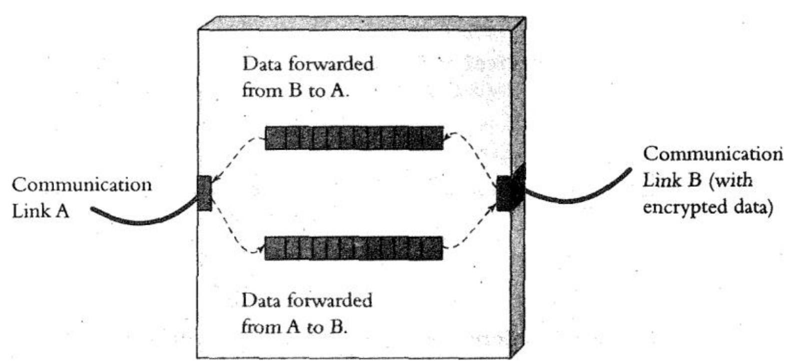


Fig:2.3:Communications Bridge

Whenever a character is received on one of the communication link, it causes an interrupt and that interrupt must be serviced quickly. Because the microprocessor must read character out of the I/O hardware before the next character arrives. Microprocessor must write characters to the I/O hardware one at a time. There is no hardware deadline by which the microprocessor must write the next character to the hardware unit. The encryption and decryption routine can encrypt and decrypt characters one at a time

Round-Robin-with-Interrupts Example: The Cordless Bar-Code Scanner

Similarly, the round robin-with-interrupts architecture would work well for the cordless bar-code scanner. Although more complicated than the simple bridge the bar-code scanner is essentially a device that gets the data from the laser that reads the bar codes and sends that data out on the radio. In this system, as in the bridge, the only real response requirements are to service the hardware quickly enough. The task code processing will get done quickly enough in a round-robin loop.

Characteristics of the Round-Robin-with-Interrupts Architecture

The primary shortcoming of the round-robin-with-interrupts architecture (other than that it is not as simple as the plain round-robin architecture) is that all of the task code executes at the same priority. Suppose that the parts of the that deal with devices A, B, and C take 200 milliseconds each. If devices A, B, and C all interrupt when the microprocessor is executing the statements at the top of the loop, then the task code for device C may have to wait for 400 milliseconds before it starts to execute.

If this is not acceptable, one solution is to move the task code for device C into the interrupt routine for device C. Putting code into interrupt routines is the only way to get it to execute at a higher priority in this architecture. This, however, will make the interrupt routine for device C take 200 milliseconds more than before, which increases the response times for the interrupt routines for lower-priority devices D, E, and F by 200 milliseconds, which may also be unacceptable.

Alternatively, you could have your main loop test the flags for the devices in a sequent e something like this: A, C, B, C, D, C, E, C, ... , testing the flag more frequently. This will improve the response for the task code for device C ... at the expense the task code for every other device

Frequency Scheduling architecture

Function queue scheduling provides a method of assigning priorities to interrupts. In this architecture, interrupt service routines accomplish urgent

processing from interrupting devices, but then put a pointer to a handler function on a queue for follow-up processing. The main loop simply checks the function queue, and if it's not empty, calls the first function on the queue. Priorities are assigned by the order of the function in the queue – there's no reason that functions have to be placed in the queue in the order in which the interrupt occurred. They may just as easily be placed in the queue in priority order: high priority functions at the top of the queue, and low priority functions at the bottom. The worst case timing for the highest priority function is the execution time of the longest function in the queue (think of the case of the processor just starting to execute the longest function right before an interrupt places a high priority task at the front of the queue). The worst case timing for the lowest priority task is infinite: it may never get executed if higher priority code is always being inserted at the front of the queue. The advantage to function queue scheduling is that priorities can be assigned to tasks; the disadvantages are that it's more complicated than the other architectures discussed previously, and it may be subject to shared data problems

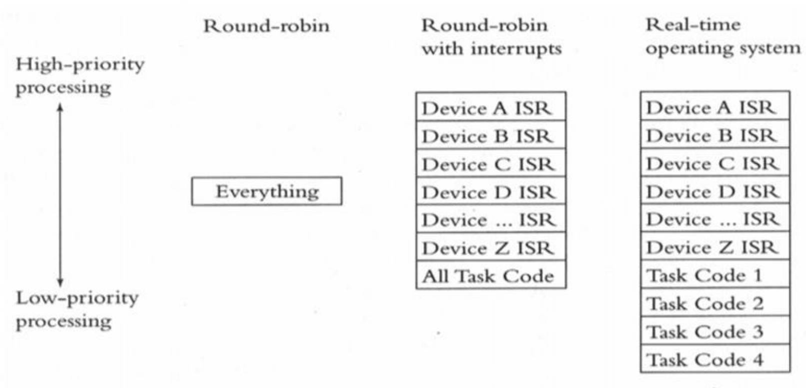


Fig:2.4:Comparison of priority level

Embedded software Development Tools

- **Compiler**
- **Cross-compiler**
- **Linker**
- **Loader**
- **Locators**
- **Simulator and**
- **IDE**

Compiler

A compiler is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses. The name "compiler" is primarily used for programs that translate source code from a high level programming language to a lower level

language (e.g., assembly language, object code, or machine code) to create an executable program

Cross-compiler

It is a type of compiler that can create executable code for different machines other than the machine it runs on . It includes a cross-compiler, one which runs on the host but produces code for the target processor. Cross-compiling doesn't guarantee correct target code due to (e.g., differences in word sizes, instruction sizes, variable declarations, library functions)

Cross-Assemblers: Host uses cross-assembler to assemble code in target's instruction syntax for the target. A cross assembler which can convert instructions into machine code for a computer other than that on which it is run.

Linker

A linker or link editor is a computer system program that takes one or more object files (generated by a compiler or an assembler) and combines them into a single executable file, library file, or another 'object' file. Linking is process of collecting and maintaining piece of code and data into a single file. Linker also link a particular module into system library. It takes object modules from assembler as input and forms an executable file as output for loader

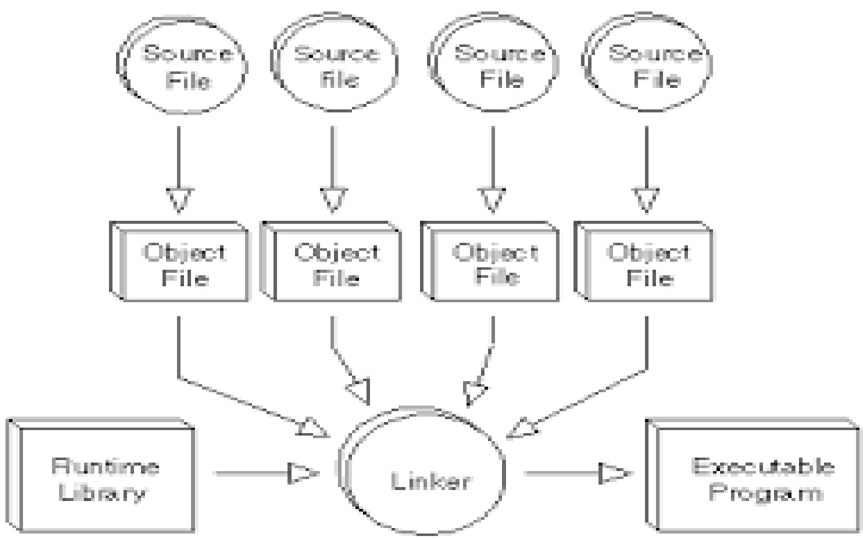


Fig:2.5:Linker Development tool

Loader and Locators:

A loader is the part of an operating system that is responsible for loading programs and libraries. Once loading is complete, the operating system starts the program by passing control to the loaded program code. The locator will use this information to assign physical memory addresses to each of the code and

data sections within the re locatable program. It will then produce an output file containing a binary memory image that can be loaded into the target ROM. In many cases, the locator is a separate development tool.

Simulator

It is, essentially, a program that allows the user to observe an operation through simulation without actually performing that operation. Simulation software is used widely to design equipment so that the final product will be as close to design specs as possible without expensive in process modification.

Applications of embedded system and development tools

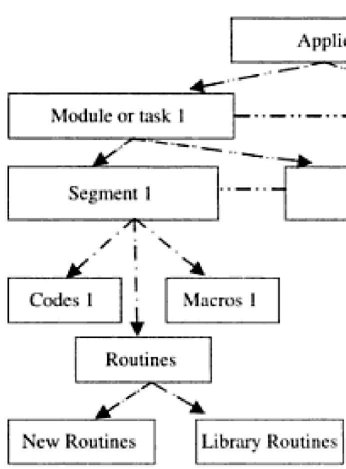
Applications of Embedded system

Integrated Development Environment (IDE) consists of software development tools for application-development. An IDE includes the project manager, editor, code-optimizing compiler, RTOS, macro assembler, library manager, linker/locator, object to hex converter, hex-file generator, simulator and debugger. The chapter describes development tools through the example of 8051 IDE from Keil Software (an ARM company).

We will also learn the hardware development tools— emulators, in-circuit emulators (ICE), target monitor-based target debugger and device-programmer.

Development phases of a microcontroller based system

Figure shows the contents of an application program. A module or task of an application is assumed to consist of several program segments. Each *segment* consists of the codes, macros and routines.



e Application Contents—Modules or Tasks, Segments, Codes, Macros, Routines (Functions) and Library Routines

A *macro* is a named entity. The entity corresponds to a set of codes within in a C function. Macro permits common sequence of codes to be developed only once and permits use of it as a software building block. A macro can be repeatedly used in different functions. The sequences of codes in the macros are given in the pre-processor statements. Within the functions, macro is used and the macros names are later replaced by the corresponding statements in preprocessor statements at the time of compilation.

Integrated Development Environment

An integrated development environment (IDE) is software for building applications that combines common developer tools into a single graphical user interface (GUI). An IDE, or Integrated Development Environment, enables programmers to consolidate the different aspects of writing a computer program. IDEs increase programmer productivity by combining common activities of writing software into a single application: editing source code, building executables, and debugging.

Getting Embedded Software into Target System

The locator will build a file as an image for the target software. There are few ways to getting the embedded software file into target system.

- **PROM programmers**
- **ROM emulators**
- **In circuit emulators**
- **Flash**
- **Monitors**

PROM Programmers:

The classic way to get the software from the locator output file into target system by creating file in ROM or PROM. Creating ROM is appropriate when software development has been completed, since cost to build ROMs is quite high. Putting the program into PROM requires a device called PROM programmer device. PROM is appropriate if software is small enough, if you plan to make changes to the software and debug. To do this, place PROM in socket on the Target than being soldered directly in the circuit (the following figure shows). When we find bug, you can remove the PROM containing the software with the bug from target and put it into the eraser (if it is an erasable PROM) or into the waste basket. Otherwise program a new PROM with software which is bug fixed and free, and put that PROM in the socket. We need small tool called chip puller (inexpensive) to remove PROM from the socket. We can insert the PROM into socket without any tool than thumb (see figure8). If PROM programmer and the locator are from different vendors, its upto us to make them compatible.

Moving maps into ROM or PROM, is to create a ROM using hardware tools or a PROM programmer (for small and changeable software, during debugging). If PROM programmer is used (for changing or debugging software), place PROM in a socket (which makes it erasable – for EPROM, or removable/replaceable) rather than 'burnt' into circuitry. PROM's can be pushed into sockets by hand, and pulled using a chip puller. The PROM programmer must be compatible with the format (syntax/semantics) of the Map

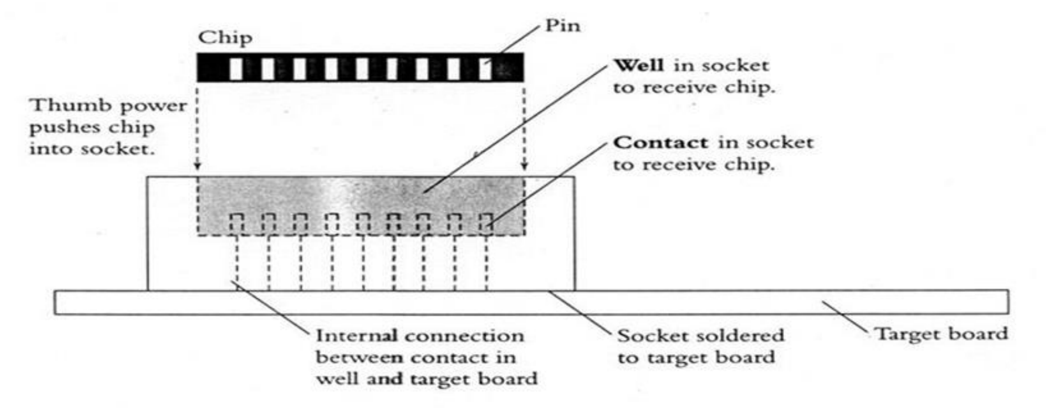


Fig: 2.5 : Schematic edge view of a socket

Getting Embedded Software into Target System – 1

ROM Emulators – Another approach is using a ROM emulator (hardware) which emulates the target system, has all the ROM circuitry, and a serial or network interface to the host system. It is used to get software into target. ROM emulator is a device that replaces the ROM into target system. It just looks like ROM, as shown figure9; ROM emulator consists of large box of electronics and a serial port or a network connection through which it can be connected to your host. Software running on your host can send files created by the locator to the ROM emulator. Ensure the ROM emulator understands the file format which the locator creates. The locator loads the Map into the emulator, especially, for debugging purposes. Software on the host that loads the Map file into the emulator must understand (be compatible with) the Map's syntax/semantics

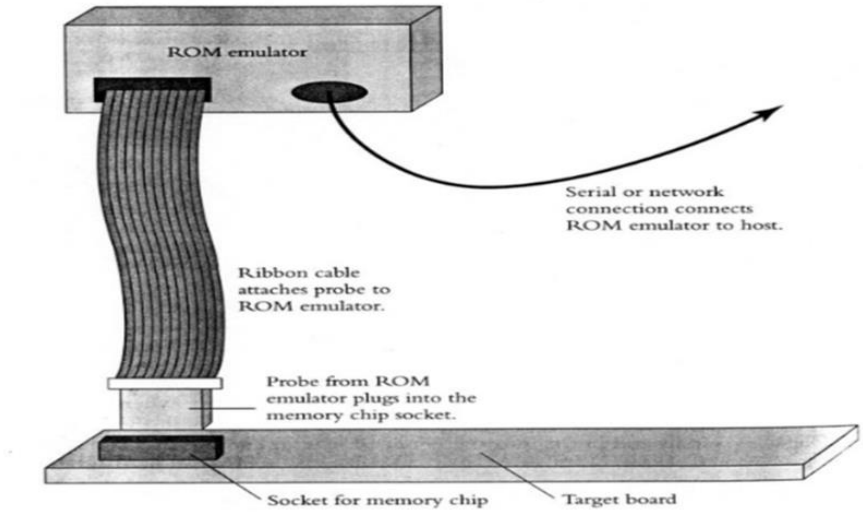


Fig: 2.6: ROM Emulator

In circuit emulators: If we want to debug the software, then we can use overlay memory which is a common feature of in-circuit emulators. In-circuit emulator is a mechanism to get software into target for debugging purposes. Flash: If your target stores its program in flash memory, then one option you always have is to place flash memory in socket and treat it like an EPROM .However, If target has a serial port, a network connection, or some other mechanism for communicating with the outside world, link then target can communicate with outside world, flash memories open up another possibility: you can write a piece of software to receive new programs from your host across the communication link and write them into the flash memory. Although this may seem like difficult

The reasons for new programs from host:

- **You can load new software into your system for debugging, without pulling chip out of socket and replacing.**
- **Downloading new software is fast process than taking out of socket, programming and returning into the socket.**
- **If customers want to load new versions of the software onto your product. The following are some issues with this approach:**
- **Here microprocessor cannot fetch the instructions from flash.**
- **The flash programming software must copy itself into the RAM, locator has to take care all these activities how those flash memory instructions are executing.**
- **We must arrange a foolproof way for the system to get flash programming software into the target i.e target system must be able to download properly even if earlier download crashes in the middle.**
- **To modify the flash programming software, we need to do this in RAM and then copy to flash.**

Monitors: It is a program that resides in target ROM and knows how to load new programs onto the system. A typical monitor allows you to send the data across a serial port, stores the software in the target RAM, and then runs it. Sometimes monitors will act as locator also, offers few debugging services like setting break points, display memory

and register values. You can write your own monitor program
Getting Embedded Software into Target System – 2

Using Flash Memory

For debugging, a flash memory can be loaded with target Map code using a software on the host over a serial port or network connection (just like using an EPROM)

Advantages:

- **No need to pull the flash (unlike PROM) for debugging different embedded code**
- **Transferring code into flash (over a network) is faster and hassle-free**

DEBUGGING TECHNIQUES

I. Testing on host machine
II. using laboratory tools
III. an example system

Introduction: While developing the embedded system software, the developer will develop the code with the lots of bugs in it. The testing and quality assurance process may reduce the number of bugs by some factor. But only the way to ship the product with fewer bugs is to write software with few fewer bugs. The world extremely intolerant of buggy embedded systems. The testing and debugging will play a very important role in embedded system software development process.

Testing on host machine :

Goals of Testing process are
– Find bugs early in the development process
– Exercise all of the code
– Develop repeatable , reusable tests
– Leave an audit trail of test results

Find the bugs early in the development process: This saves time and money. Early testing gives an idea of how many bugs you have and then how much trouble you are in. But the target system is available early in the process, or the hardware may be buggy and unstable, because hardware engineers are still working on it.

Exercise all of the code: Exercise all exceptional cases, even though, we hope that they will never happen, exercise them and get experience how it works. It is impossible to exercise all the code in the target. For example, a laser printer may have code to deal with the situation that arise when the user presses the one of the buttons just as a paper jams, but in the real time to test this case. We have to make paper to jam and then press the button within a millisecond, this is not very easy to do. Develop reusable, repeatable tests: It is frustrating to see the bug once but not able to find it. To make refuse to happen again, we need to repeatable tests. It is difficult to create repeatable tests at target environment. Example: In bar code scanner, while scanning it will show the pervious scan results every time, the bug will be difficult to find and fix.

TEXT / REFERENCE BOOKS

1. KVKK Prasad, “Embedded / Real Time Systems”, Dreamtech Press, 2005.
2. David Simon, “An Embedded Software Primer”, Pearson Education Asia, First Indian Reprint 2000.
3. Raj Kamal, ‘Embedded system-Architecture, Programming, Design’, Tata McGraw Hill, 2011.
4. Arnold Berger, “Embedded system design”, CMP books, 1st Edition, 2005
5. Wayne Wolf, “Computers as components”, Morgan Kaufmann publishers, 2nd Edition, 2008.
6. Tammy Noergaard, “Embedded Systems Architecture”, Elsevier, 2006

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT -III

EMBEDDED SYSTEM – SEC1320

III EMBEDDED NETWORKING

Embedded Networking: Introduction, I/O Device Ports - Serial Bus communication protocols- RS232 standard- RS485 - CAN Bus - RS485 - Serial Peripheral Interface (SPI) - Inter-Integrated Circuits (I2C) - PC Parallel port communication Protocols - Bluetooth- network using ISA, PCI- Wireless and Mobile System Protocols.

IO port types- Serial and parallel IO ports

A port is a device to receive the bytes from external peripheral(s) [or device(s) or processor(s) or controllers] for reading them later using instructions executed on the processor to send the bytes to external peripheral or device or processor using instructions executed on processor. A Port connects to the processor using address decoder and system buses. The processor uses the addresses of the port-registers for programming the port functions or modes, reading port status and for writing or reading bytes.

Example

- **SI serial interface in 8051**
- **SPI serial peripheral interface in 68HC11**
- **PPI parallel peripheral interface 8255**
- **Ports P0, P1, P2 and P3 in 8051 or PA, PB,PC and PD in 68HC11**
- **COM1 and COM2 ports in an IBM**

PC IO Port Types

Types of Serial ports

- **Synchronous Serial Input**
- **Synchronous Serial Output**
- **Asynchronous Serial UART input**
- **Asynchronous Serial UART output (both as input and as output, for example,modem.)**
- **Types of parallel ports**
- **Parallel port one bit Input**
- **Parallel one bit output**
- **Parallel Port multi-bit Input**
- **Parallel Port multi-bit**

Output Synchronous Serial Input

Example

Inter-processor data transfer, reading from CD or hard disk, audio input, video input, dial tone, network input, transceiver input, scanner input, remote controller input, serial I/O bus input, writing to flash memory using SDIO (Secure Data Association IO based card).

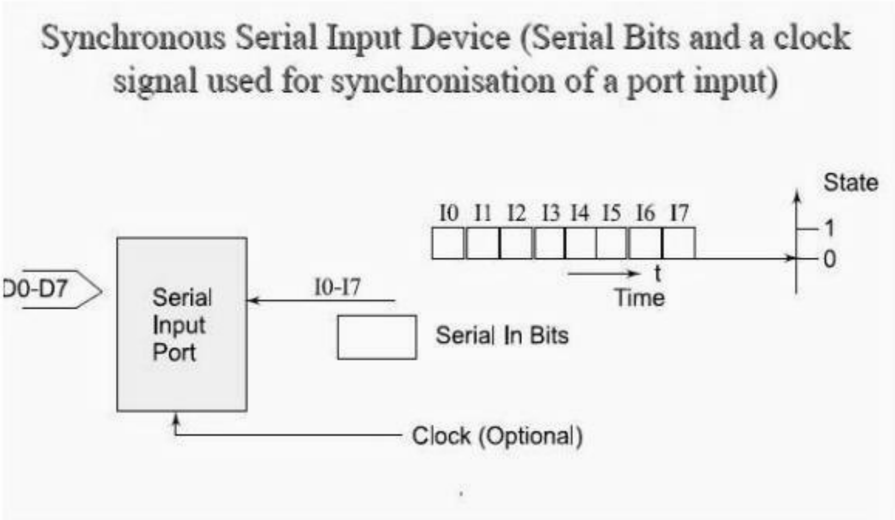


Fig:3.1:Synchronous Serial Input

The sender along with the serial bits also sends the clock pulses SCLK (serial clock) to the receiver port pin. The port synchronizes the serial data input bits with clock bits. Each bit in each byte as well as each byte in synchronization Synchronization means separation by a constant interval or phase difference.

If clock period = T, then each byte at the port is received at input in period = 8T. The bytes are received at constant rates. Each byte at input port separates by 8T and data transfer rate or the serial line bits is (1/T) bps. [1bps = 1 bit per s] Serial data and clock pulse-inputs On same input line − when clock pulses either encode or modulate serial data input bits suitably. Receiver detects the clock pulses and receives data bits after decoding or demodulating. On separate input line − When a separate SCLK input is sent, the receiver detects at the middle or+ ve edge or –ve edge of the clock pulses that whether the data-input is 1 or 0 and saves the bits in an 8-bit shift register. The processing element at the port (peripheral) saves the byte at a port register from where the microprocessor reads the byte.

Master output slave input (MOSI) and Master input slave output (MISO)

MOSI when the SCLK is sent from the sender to the receiver and slave is forced to synchronize sent inputs from the master as per the inputs from master clock.

MISO when the SCLK is sent to the sender (slave)from the receiver (master) and slave is forced to synchronize for sending the inputs to master as per the master clock

outputs.

Synchronous serial input is used for inter processor transfers, audio inputs and streaming data inputs.

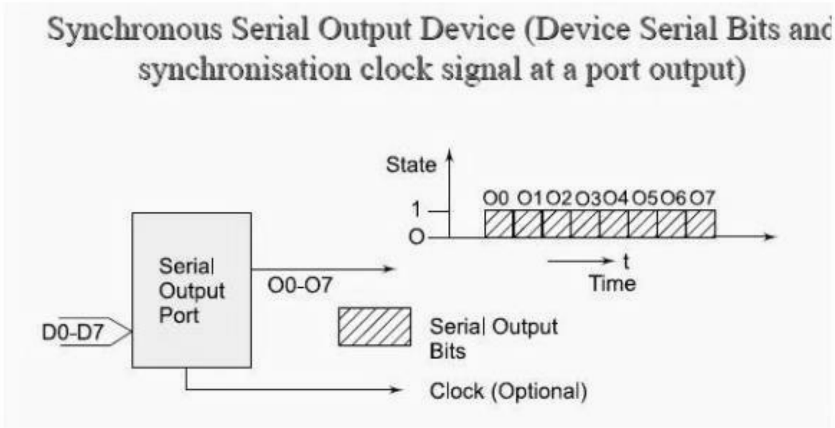


Fig:3.2: Synchronous Serial Output

Inter-processor data transfer, multiprocessor communication, writing to CD or hard disk, audio Input/output, video Input/output,dialer output, network device output, remote TV Control, transceiver output, and serial I/O bus output or writing to flash memory using SDIO

Synchronous Serial Output

Each bit in each byte sent in synchronization with a clock. Bytes sent at constant rates. If clock period= T, then data transfer rate is (1/T) bps.

- **Sender either sends the clock pulses at SCLK pin or sends the serial data output and clock**
- **pulse-input through same output line with clock pulses either suitably modulate or encode the serial output bits.**

Synchronous serial output using shift register

The processing element at the port (peripheral) sends the byte through a shift register at the port to where the microprocessor writes the byte. Synchronous serial output is used for inter processor transfers, audio outputs and streaming data outputs.

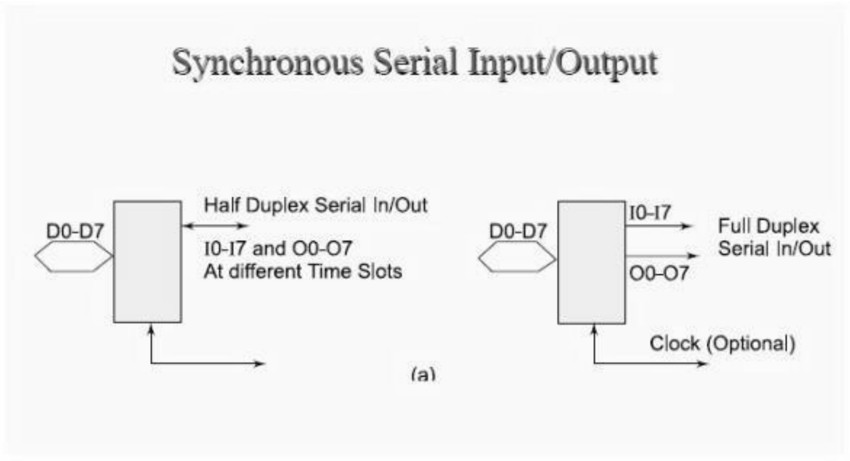


Fig:3.3: Synchronous Serial Input/output

Each bit in each byte is in synchronization at input and each bit in each byte is in synchronization at output with the master clock output. The bytes are sent or received at constant rates. The I/Os can also be on same I/O line when input/output clock pulses either suitably modulate or encode the serial input/output, respectively. If clock period = T, then data transfer rate is (1/T)bps. The processing element at the port (peripheral)sends and receives the byte at a port register to or from where the microprocessor writes or reads the byte

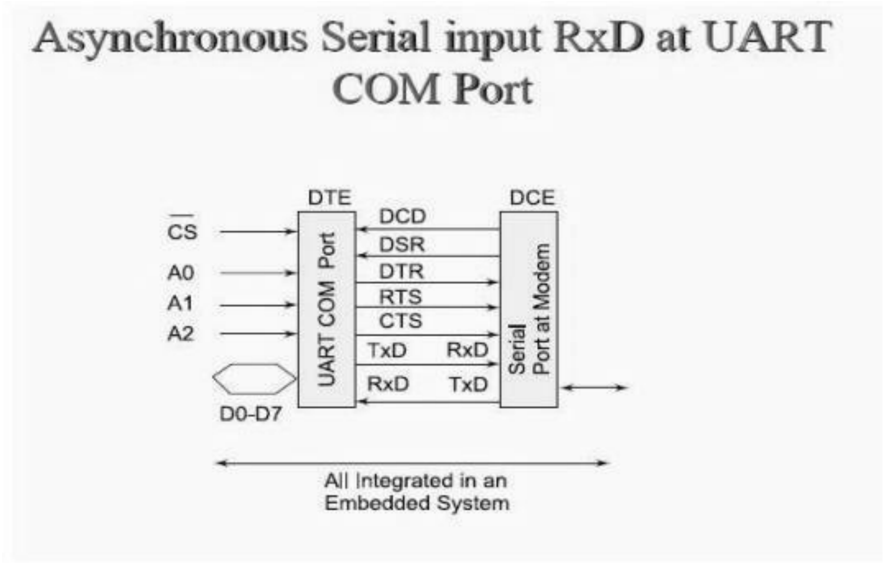


Fig:3.4: Asynchronous Serial port line

Asynchronous Serial port line RxD (receive data). Does not receive the clock pulses or clock information along with the bits.

Each bit is received in each byte at fixed intervals but each received byte is not in synchronization. Bytes separate by the variable intervals or phase differences. Asynchronous serial input also called UART input if serial input is according to UART protocol

Example Serial Asynchronous Input

Asynchronous serial input is used for keypad inputs and modem inputs in computers Keypad controller serial data-in, mice, keyboard controller, modem input, character send inputs on serial line [also called UART (universal receiver and transmitter) input when according to UART mode
Starting point of receiving the bits for each byte is indicated by a line transition from 1to 0 for a period = T. [T−1 called baud rate.

If senders shift-clock period = T, then a byte at the port is received at input in period= 10.T or 11.T due to use of additional bits at start and end of each byte. Receiver detects n bits at the intervals of T from the middle of the start indicating bit. The n = 0, 1, …, 10 or 11 and finds whether the data-input is 1 or 0 and saves the bits in an 8-bit shift register.

Processing element at the port (peripheral)saves the byte at a port register from where the microprocessor reads the byte.

Asynchronous Serial Output

Asynchronous output serial port line TxD(transmit data).

Each bit in each byte transmit at fixed intervals but each output byte is not in synchronization (separates by a variable interval or phase difference). Minimum separation is 1 stop bit interval TxD. Does not send the clock pulses along with the bits. Sender transmits the bytes at the minimum intervals of n.T. Bits receiving starts from the middle of the start indicating bit, n = 0, 1, …, 10 or 11 and sender sends the bits through a 10 or 11 -bit shift register. The processing element at the port(peripheral) sends the byte at a port register to where the microprocessor is to write the byte.

Synchronous serial output is also called UART output if serial output is according to UART protocol

Example Serial Asynchronous Output. Output from modem, output for printer, the output on a serial line [also called UART output when according to UART

Parallel Data Communication

Half Duplex

- Half duplex means as follows: at an instant communication can only be one way (input or output) on a bi-directional line.
- An example of half-duplex mode─ telephone communication. On one telephone line, the talk can only in the half duplex way mode.

Full Duplex

- Full duplex means that at an instant,the communication can be both ways.

An example of the full duplexasynchronous mode of communicationis the communication between themodem and the computer though TxDand RxD lines or communication using
SI in modes 1, 2 and 3 in 8051

Parallel Port single bit input

- Completion of a revolution of a wheel,
- Achievingpreset pressure in a boiler,
- Exceeding the upper limit of permittedweight over the pan of an electronicbalance,
- Presence of a magnetic piece in the vicinityof or within reach of a robot arm to its endpoint and Filling of a liquid up to a fixed level.

Parallel Port Output- single bit

- PWM output for a DAC, which controlsliquid level, or temperature, or pressure, orspeed or angular position of a rotating shaftor a linear displacement of an object or ad.c. motor control
- Pulses to an external circuit
- Control signal to an external circuit

Parallel Port Input- multi-bit

- ADC input from liquid level measuringsensor or temperature sensor or pressuresensor or speed sensor or d.c. motor rpmsensor

Parallel Port Output- multi-bit

- LCD controller for Multilane LCD displaymatrix unit in a cellular phone to display onthe screen the phone number, time,messages, character outputs or pictogrambit-images for display screen or e-mail orweb page
- Print controller output
- Stepper-motor coil driving bits

Parallel Port Input-Output

- PPI 8255
- Touch screen in mobile phone

Ports or DevicesCommunication and communicationprotocols

Two Modes of communication between the devices and computer system

Full Duplex – Both devices or device and computer system simultaneously communicate each other.

Half Duplex – Only one device can communicate with another at an instance

Three ways of communication betweenthe ports or devices

1. Synchronous
2. Iso-synchronous
3. Asynchronous

1. Synchronous and Iso-synchronous Communication in Serial Ports or Devices **Synchronous Communication.**

When a byte (character) or a frame (acollection of bytes) in of the data isreceived or transmitted at the constanttime intervals with uniform phasedifferences, the communication iscalled as ***synchronous***. Bits of a fullframe are sent in a prefixed maximumtime interval.

Iso-synchronous

Synchronous communication special case—when bits of a full frame are sent in themaximum time interval, which can bevariable.

Synchronous Communication

Clock information is transmittedexplicitly or implicitly insynchronous communication. Thereceiver clock continuously maintainsconstant phase difference with thetransmitter clock. Bits of a data framemaintain uniform phase differenceand are sent within a fixed maximumtime interval.

Example of synchronous serial communication

- Frames sent over a LAN. Frames of data communicate with the constant time intervals between each frame remaining constant.
- Another example is the inter-processor communication in a multiprocessor system Optional Synchronous Code bits
- Optional Sync Code bits or bi-sync code bits orframe start and end signaling bits— Duringcommunication few bits (each separated byinterval ΔT) sent as Sync code to enable the framesynchronization or frame start signaling.
- Code bits precede the data bits.
- May be inversion of code bits after each frame incertain protocols.
- Flag bits at start and end are also used in certainprotocols. Always present Synchronous device portdata bits

- Reciprocal of T is the bit per second(bps).
- Data bits— *m* frame bits or 8 bitstransmit such that each bit is at the linefor time ΔT or, each frame is at the linefor time (*m*. T)*m* may be 8 or a large number. Itdepends on the protocolSynchronous device clock bits
- Clock bits ─ Either on a separate clockline or on data line such that the clockinformation is also embedded with thedata bits by an appropriate encoding ormodulation
- Generally not optional

Two characteristics of asynchronouscommunication

1. Bytes (or frames) need not maintain a constantphase difference and are asynchronous, i.e., notin synchronization. There is permission to sendeither bytes or frames at variable timeintervals— This*facilitates in-betweenhandshaking* between the serial transmitter portand serial receiver port

2. Though the *clock* must ticking at a certain ratealways has to be there to transmit the bits of asingle byte (or frame) serially, it is *alwaysimplicit* to the asynchronous data receiver and isindependent of the transmitter

Clock Features

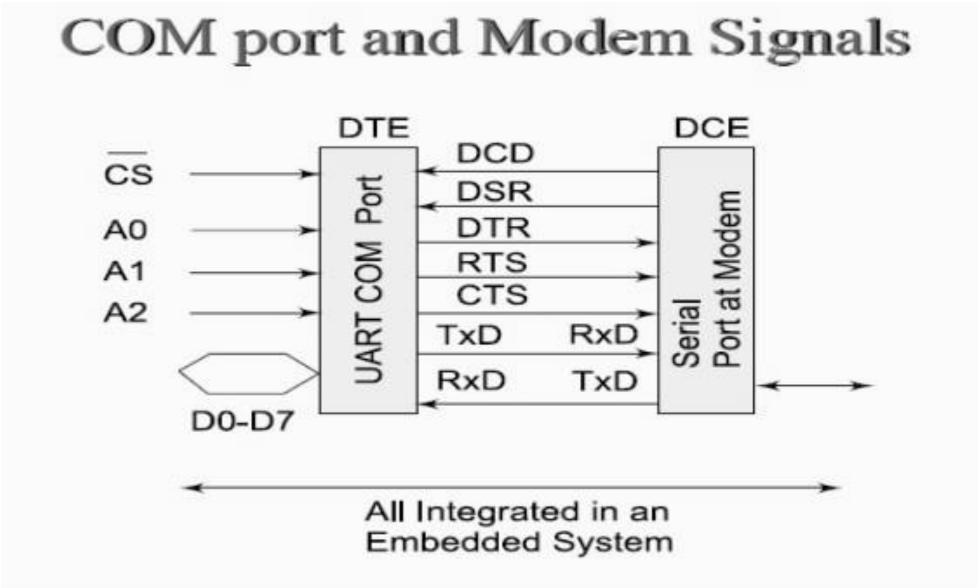
_ The transmitter *does not transmit* (neitherseparately nor by encoding using modulation)along with the serial stream of bits any *clockrate information* in the asynchronouscommunication and *receiver clock thus is notable to maintain identical frequency andconstant phase difference* with transmitter clock

Example: IBM personal computer has two COMports (communication ports)

- _ COM1 and COM2 at IO addresses 0x2F8-0xFFand 0xx38-0x3FF
- _ Handshaking signals— RI, DCD, DSR, DTR,RTS, CTS, DTR
- _ Data Bits— RxD and TxDExample: COM port and Modem Handshakingsignals
- _ When a modem connects, modem sends*data carrier detect* DCD signal at aninstance t0.
- _ Communicates *data set ready* (DSR)signal at an instance t1 when it receives thebytes on the line.
- _ Receiving computer (terminal) responds atan instance t2 by data terminal ready(DTR) signal.

After DTR, *request to send* (RTS) signal is sent at aninstance t3

- _ Receiving end responds by *clear to send* (CTS) signalat an instance t4. After the response CTS, the data bitsare transmitted by modem from an instance t5 to thereceiver terminal.
- _ Between two sets of bytes sent in asynchronous mode,the handshaking signals RTS and CTS can again beexchanged. This explains why the bytes do not remainsynchronized during asynchronous transmission.



3. Communication Protocols

1. Protocol

A protocol is a standard adopted,which tells the way in which the bits ofa frame must be sent from a device (orcontroller or port or processor) toanother device or system

[Even in personal communication wefollow a protocol – we say Hello! Thentalk and then say good bye!]

A protocol defines how are the framebits:

1) sent− synchronously or Isosynchronouslyor asynchronously and at what rate(s)?

2) preceded by the header bits?How the receiving device addresscommunicated so that only destineddevice activates and receives the bits?

[Needed when several devicesaddressed though a common line(bus)]

3) How can the transmitting deviceaddress defined so that receivingdevice comes to know the sourcewhen receiving data from severalsources?

4) How the frame-length defined so thatreceiving device know the frame-sizein advance?

5) Frame-content specifications –Arethe sent frame bits specify the controlor device configuring or commend ordata?

6) Are there succeeding to frame thetrailing bits so that receiving devicecan check the errors, if any inreception before it detects end of theframe ?

A protocol may also define:

7) Frame bits minimum and maximumlength permitted per frame

8) Line supply and impedances andline-Connectors specifications

Specified protocol at an embedded systemport or communication deviceIO port bits sent after first formattedaccording to a specified protocol, whichis to be followed when communicatingwith another device through an IO portor channel

Protocols

- _ HDLC, Frame Relay, for synchronouscommunication
- _ For asynchronous transmission from a deviceport– RS232C, UART, X.25, ATM, DSL and

ADSL

- _ For networking the physical devices intelecommunication and computer networks –

Ethernet and token ring protocols used in LANNetworks

Protocols in embedded network devices

- _ For Bridges and routers
- _ Internet appliances application protocolsand Web protocols —HTTP (hyper texttransfer protocol), HTTPS (hyper texttransfer protocol Secure Socket Layer),SMTP (Simple Mail Transfer Protocol),POP3 (Post office Protocol version 3),ESMTP (Extended SMTP),

File transfer, Boot Protocols in embedded devicesnetwork

- _ TELNET (Tele network),
- _ FTP (file transfer protocol),
- _ DNS (domain network server),
- _ IMAP 4 (Internet Message ExchangeApplication Protocol) and
- _ Bootp (Bootstrap protocol).Wireless Protocols in embedded devices network
- _ Embedded wireless appliances useswireless protocols— WLAN 802.11,802.16, Bluetooth, ZigBee, WiFi, WiMax,

Serial Data Communication

Data Communication is one of the most challenging fields today as far as technology development is concerned. Data, essentially meaning information coded in digital form, that is, 0s and 1s, is needed to be sent from one point to the other either directly or through a network. And when many such systems need to share the same information or different information through the same medium, there arises a need for proper organization (rather, "socialization") of the whole network of the systems, so that the whole system works in a cohesive fashion. Therefore, in order for a proper interaction between the data transmitter (the device needing to commence data communication) and the data receiver (the system which has to receive the data sent by a transmitter) there has to be some set of rules or ("protocols") which all the interested parties must obey. The requirement above finally paves the way for some data communication standards.

Depending on the requirement of applications, one has to choose the type of communication strategy. There are basically two major classifications, namely SERIAL and PARALLEL, each with its variants.

Serial data communication strategies and, standards are used in situations having a limitation of the number of lines that can be spared for communication. This is the primary mode of transfer in long-distance communication. But it is also the situation in embedded systems where various subsystems share the communication channel and the speed is not a very critical issue. Standards incorporate both the software and hardware aspects of the system while buses mainly define the cable characteristics for the same communication type. Serial data communication is the most common low-level protocol for communicating between two or more devices. Normally, one device is a computer, while the other device can be a modem, a printer, another computer, or a scientific instrument such as an oscilloscope or a function generator. As the name suggests, the serial port sends and receives bytes of information, rather characters (used in the other modes of communication), in a serial fashion - one bit at a time. These bytes are transmitted using either a binary (numerical) format or a text format.

The most common serial communication system protocols can be studied under the following categories: Asynchronous, Synchronous and Bit-Synchronous communication standards

RS-232

This is the original serial port interface "standard" and it stands for "Recommended Standard Number 232" or more appropriately EIA Recommended Standard 232 is the oldest and the most popular serial communication standard. It was first introduced in 1962 to help ensure connectivity and compatibility across manufacturers for simple serial data communications.

Applications

Peripheral connectivity for PCs (the PC COM port hardware), which can range beyond modems and printers to many different handheld devices and modern scientific instruments.

All the various characteristics and definitions pertaining to this standard can be summarized according to the following

The maximum bit transfer rate capability and cable length. • Communication Technique: names, electrical characteristics and functions of signals. • The mechanical connections and pin assignments.

The Standard

Maximum Bit Transfer Rate, Signal Voltages and Cable Length

• RS-232"s capabilities range from the original slow data rate of up to 20 kbps to over 1 Mbps for some of the modern applications.

• RS-232 is mainly intended for short cable runs, or local data transfers in a range up to 50 feet maximum, but it must be mentioned here that it also depends on the Baud Rate

It is a robust interface with speeds to 115,200 baud, and

• It can withstand a short circuit between any 2 pins. • It can handle signal voltages as high / low as ±15 volts.

Signal States and the Communication Technique

Signals can be in either an active state or an inactive

state. RS232 is an Active LOW voltage driven

interface where:

ACTIVE STATE: An active state corresponds to the binary value 1. An active signal state can also be indicated as logic "1", "on", "true", or a "mark".

INACTIVE STATE: An inactive signal state is stated as logic "0", "off", "false", or a "space". • For data signals, the "true" state occurs when the received signal voltage is more negative than -3 volts, while the "false" state occurs for voltages more positive than 3 volts. • For control signals, the "true" state occurs when the received signal voltage is more positive than 3 volts, while the "false" state occurs for voltages more negative than -3 volts.

Transition or “Dead Area”

Signal voltage region in the range >-3.0V and < +3.0V is regarded as the 'dead area' and allows for absorption of noise. This same region is considered a transition region, and the signal state is undefined.

To bring the signal to the "true" state, the controlling device un asserts (or lowers) the value for data pins and asserts (or raises) the value for control pins. Conversely, to bring the signal to the "false" state, the controlling device asserts the value for data pins and unasserts the value for control pins. The "true" and "false" states for a data signal and for a control signal are as shown below.

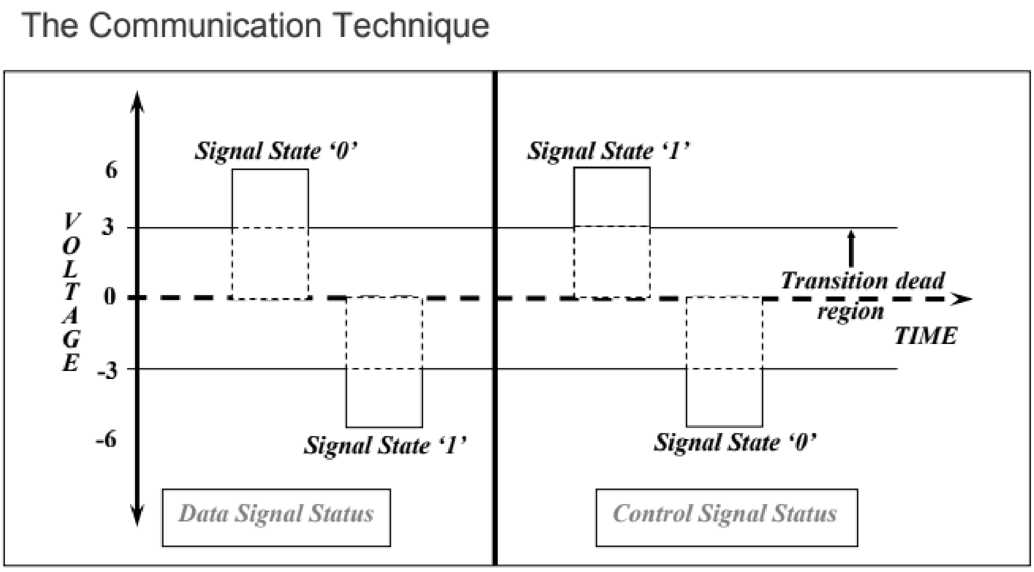


Fig:3.5: Graphical representation of communication technique

A factor that limits the distance of reliable data transfer using RS-232 is the signaling technique that it uses.

This interface is “single-ended” meaning that communication occurs over a SINGLE WIRE referenced to GROUND, the ground wire serving as a second wire. Over that single wire, marks and spaces are created. • While this is very adequate for slower applications, it is not suitable for faster and longer applications.

The communication technique

RS-232 is designed for a unidirectional half-duplex communications mode. That simply means that a transmitter (driver) is feeding the data to a receiver over a copper line. The data always follows the direction from driver to receiver over that line. If return transmission is desired, another set of driver- receiver pair and separate wires are needed. In other words, if bi-directional or full-duplex capabilities are needed, two separate communications paths are required.

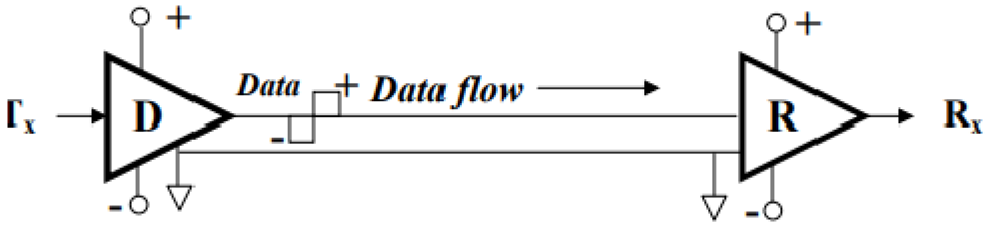


Fig:3.7: Single ended, Unidirectional, Half Duplex

Disadvantages

Being a single-ended system it is more susceptible to induced noise, ground loops and ground shifts, a ground at one end not the same potential as at the other end of the cable e.g. in applications under the proximity of heavy electrical installations and machineries

Some Modern Perspectives/Advantages

Most applications for RS-232 today are for data connectivity between portable handheld devices and a PC. Some of the differences between the modern RS-232 integrals from the older versions are:

• Such devices require that the RS-232 IC to be very small, have low current drain, operate from a +3 to +5-V supply.

• They provide ESD protection on all transmit and receive pins. For example, some RS232 interfaces have specifically been designed for handheld devices and support data rates greater than 250 kbps, can operate down to +2.7 V.

• They can automatically go into a standby mode drawing very small currents of the order of only 150 nA when not in use, provide 15 kV ESD protection on data pins and are in the near- chip-scale 5 X 5 mm quad flat no-lead package. Nevertheless, for portable and handheld applications the older RS-232 is still the most popular one.

RS-485

This is an improved RS-422 with the capability of connecting a number of devices (transceivers) on one serial bus to form a network.

The Standard Maximum Bit Transfer Rate, Signal Voltages and Cable Length

Such a network can have a "daisy chain" topology where each device is connected to two other devices except for the devices on the ends.

• Only one device may drive data onto the bus at a time. The standard does not specify the rules for deciding who transmits and when on such a network. That solely depends upon the system designer to define.

• **Variable data rates are available for this standards but the standard max. data rate is 10 Mbps, however ,some manufacturers do offer up to double the standard range i.e. around 20 Mbps,but of course, it is at the expense of cable width.**

• **It can connect upto 32 drivers and receivers in fully differential mode similar to the RS – 422.**

Communication Technique

EIA Recommended Standard 485 is designed to provide bi-directional half-duplex multi- point data communications over a single two-wire bus.

• **Like RS-232 and RS-422, full-duplex operation is possible using a four-wire, two-bus network but the RS-485 transceiver ICs must have separate transmit and receive pins to accomplish this.**

• **RS-485 has the same distance and data rate specifications as RS-422 and uses differential signaling but, unlike RS-422, allows multiple drivers on the same bus. As depicted in the Figure below, each node on the bus can include both a driver and receiver forming a multi- point star network. Each driver at each node remains in a disabled high impedance state until called upon to transmit. This is different than drivers made for RS422 where there is only one driver and it is always enabled and cannot be disabled.**

• **With automatic repeaters and tri-state drivers the 32-node limit can be greatly exceeded. In fact, the ANSI-based SCSI-2 and SCSI-3 bus specifications use RS-485 for the physical (hardware) layer.**

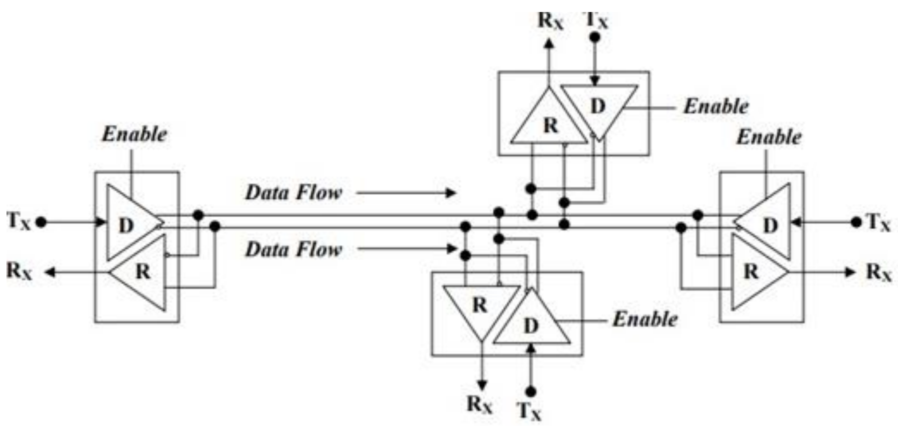


Fig:3.8:RS 485-Differential signaling, Bidirectional, Half duplex, Multipoint

Advantages

Among all of the asynchronous standards mentioned above this standard offers the maximum data rate. • Apart from that special hardware for avoiding bus contention and ,

• **A higher receiver input impedance with lower Driver load impedances are its other**

assets. Differences between the various standards at a glance

All together the important electrical and mechanical characteristics for application purposes may be classified and summarized according to the table below.

	RS-232	RS-422/423	RS-485
Signaling Technique	Single-Ended (Unbalanced)	Differential (Balanced)	Differential (Balanced)
Drivers and Receivers on Bus	1 Driver 1 Receiver	1 Driver 10 Receivers	32 Drivers 32 Receivers
Maximum Cable Length	50 feet	4000 feet	4000 feet
Original Standard Maximum Data Rate	20 kbps	10 Mbps down to 100 kbps	10 Mbps down to 100 kbps
Minimum Loaded Driver Output Voltage Levels	+/-5.0 V	+/-2.0 V	+/-1.5 V
Driver Load Impedance	3 to 7 k	100	54
Receiver Input Impedance	3 to 7 k	4 k or greater	12 k or greater

Data Communication buses

The role of networking in present-day data communication hardly needs any elaboration. The situation is also similar in the case of embedded systems, particularly those which are distributed over a larger geographical region – the so-called distributed embedded systems. Unfortunately, the most common network standard, namely the Ethernet, is not suitable for such distributed systems, especially when there are real-time constraints to be satisfied. This is due to the lack of any service time guarantee in the Ethernet standard. On the other hand, alternatives like Token Ring, which do provide a service-time guarantee, are not very suitable because of the requirement of a ring-type topology not very convenient to implement in the industrial environment. The industry therefore proposed a standard called „Token-bus" (and got it approved as the IEEE 802.5 specification) to cater to such requirements. However, the standard became too complex and inefficient as a result. Subsequently different manufacturers have come up with their own standards, which are being implemented in specific applications. In this lesson we learn about three such standards, namely

- **I 2 C Bus**
- **Field Bus**
- **CAN Bus**

CAN BUS

CAN was the solution developed by Robert Bosch GmbH, Germany in 1986 for the development of a communication system between three ECUs (electronic control units) in vehicles being designed by Mercedes. The UART, which had been in use for

long, had been rendered unsuitable in their situation because of its point-to-point communication methodology. The need for a multi-master communication system became a stringent requirement. Intel then fabricated the first CAN in 1987. Controller Area Network (CAN) is a very reliable and message-oriented serial network that was originally designed for the automotive industry, but has become a sought after bus in industrial automation as well as other applications. The CAN bus is primarily used in embedded systems, and is actually a network established among micro controllers. The main features are a two-wire, half duplex, high-speed network system mainly suited for high-speed applications using short messages. Its robustness, reliability and compatibility to the design issues in the semiconductor industry are some of the remarkable aspects of the CAN technology.

Main Features

1. **CAN can link up to 2032 devices (assuming one node with one identifier) on a single network. But accounting to the practical limitations of the hardware (transceivers), it may only link up to110 nodes (with 82C250, Philips) on a single network. ‰**
2. **It offers high-speed communication rate up to 1 Mbits/sec thus facilitating real-time control. ‰**
3. **It embodies unique error confinement and the error detection features making it more trustworthy and adaptable to a noise critical environment.**

CAN Versions

Originally, Bosch provided the specifications. However the modern counterpart is designated as Version 2.0 of this specification, which is divided into two parts:

• Version 2.0A or Standard CAN; Using 11 bit identifiers.

• Version 2.0B or Extended CAN; Using 29 bit identifiers.

The main aspect of these Versions is the formats of the MESSAGE FRAME; the main difference being the IDENTIFIER LENGTH.

CAN Standards

There are two ISO standards for CAN. The two differ in their physical layer descriptions. ‰

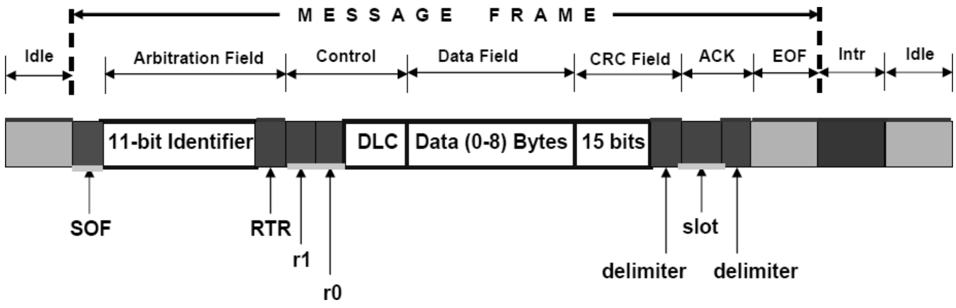
1. **ISO 11898 handles high-speed applications up to 1Mbit/second. ‰**
2. **ISO 11519 can go upto an upper limit of 125kbit/second.**

The Can Protocol/Message Formats

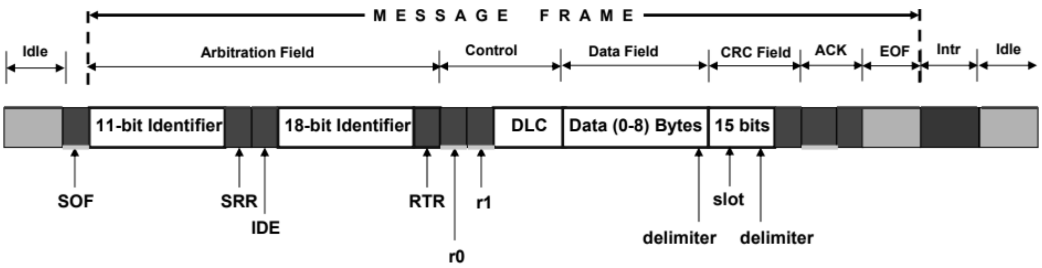
In a CAN system, data is transmitted and received using Message Frames. Message Frames carry data from any transmitting node to single or multiple receiving nodes.

CAN protocol can support two Message Frame formats:

Version 2.0A - Standard CAN



Version 2.0B - Extended CAN



BASIC CAN Controller

The basic topology for the CAN Controller has been shown in figure 2 below. The basic controller involves FIFOs for message transfers and it has an enhanced counterpart in Full-CAN controller, which uses message BUFFERS instead.

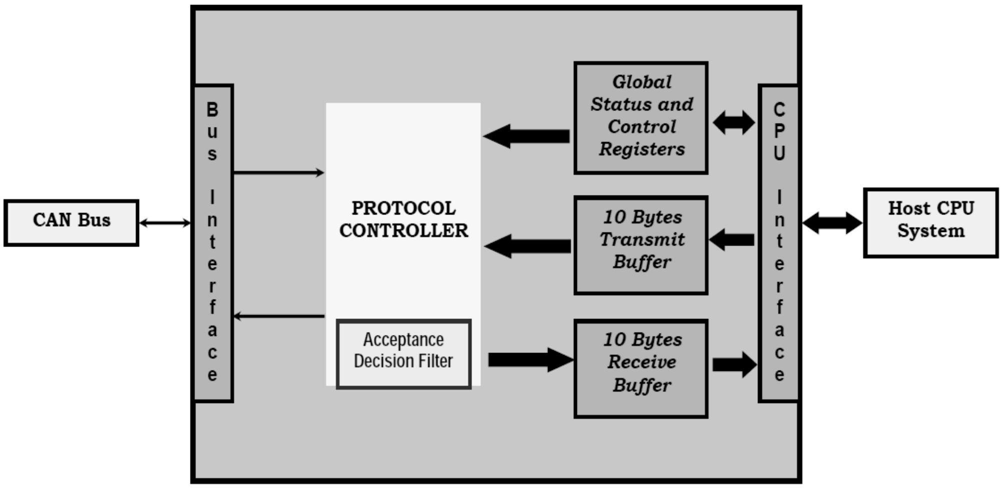


Fig:3.9: Basic CAN Controller

Distributed Control Area Network example - a network of embedded systems in automobile

CAN-bus line usually interconnects to a CAN controller between line and host at the node. It gives the input and gets output between the physical and data link layers at the host node.

The CAN controller has a BIU (bus interface unit consisting of buffer and driver), protocol controller, status-cum control registers, receiver-buffer and message objects. These units connect the host node through the host interface circuit

Wireless and mobile system protocols
Wireless Personal Area Network (WPAN)

IrDA (Infrared Data Association)
Bluetooth 2.4 GHz
802.11 WLAN and 802.11b WiFi
ZigBee 900 MHz

IrDA (Infrared Data Association)

- **Used in mobile phones, digital cameras, keyboard, mouse, printers to communicate to laptop computer and for data and pictures download and synchronization.**
- **Used for control TV, air-conditioning, LCD projector, VCD devices from a distance**
- **Use infrared (IR) after suitable modulation of the data bits.**
- **Communicates over a line of sight**

- **Phototransistor receiver for infrared rays**

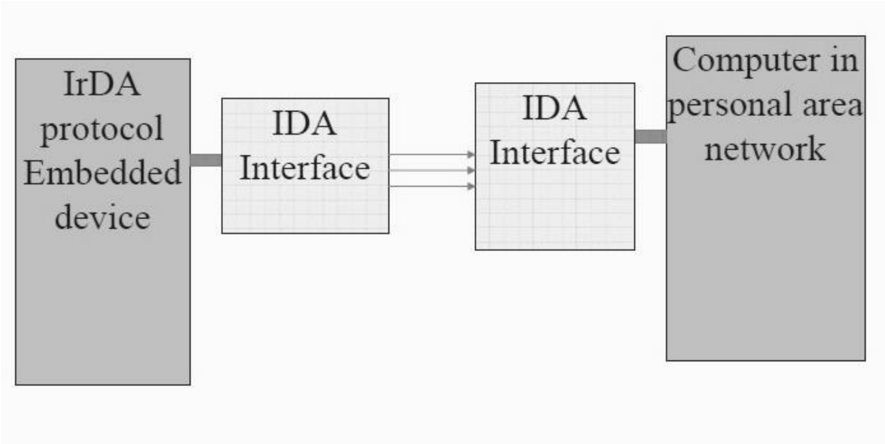


Fig:3.10: Block diagram of IrDA

IrDA protocol suite

- **Supports data transfer rates of up to 4 Mbps**
- **Supports bi-directional serial communication over viewing angle between ± 15° and distance of nearly 1 m**
- **At 5 m, the IR transfer data can be up to data transfer rates of 75 kbps**
- **Should be no obstructions or wall in between the source and receiver**

Five levels of communication

Level 1─ minimum required communication.
Level 2 ─ access-based communication.
Level 3 ─index-based communication.
Level 4 ─ sync communication. Synchronization software, for example, ActiveSync or HotSync is used.
Level 5 ─ SyncML (synchronization markup language) based communication. A SyncML protocol is used for device management and synchronization with server and client devices connected by IrDA.

IrDA Physical Layer

Lower layer─ physical layer 1.0 or 1.1.
1.0─ supports data transfer rates of 9.6 kbps to 115.2 kbps
1.1─ 115.2 kbps to 4 Mbps

5 layers of IrDA

Application for example, IrDA Alliance Sync protocol
Session Layer IrLAN, IrBus, IrMC, IrTran, IrOBEX (Object Exchange) and standard serial port emulator protocol IrCOMM (IR communication). IrBus
Transport Layer Tiny TP or IrLMIAS
Data-link IrLMP and IrLAP Sublayers
Physical 1.0 (9.6 kbps to 115.2 kbps) or 1.1 (115.2 kbps to 4 Mbps)

Two sub-layers at IrDA data-link layer

- **IrLMP (IR link management protocol) upper sub-layer**
- **IrLAP (IR link access protocol) lower sub-layer.**
- **IrLAP─ HDLC synchronous communication**

IrDA upper layer protocols

- **for Transport**
- **for Session**
- **for Application**

IrDA Transport layer protocol

- **During transmission specifies ways of flow control, segmentation of data and packetization.**
- **During reception, specifies assembling of the segments and packets.**
- **Tiny TP (transport protocol).**
- **IrLMIAS (IR Link Management Information Access Service Protocol).**

IrDA Session Layer

- **IrLAN**
- **IrBus**
- **IrMC**
- **IrTran**
- **IrOBEX (Object Exchange) and**
- **IrCOMM (IR communication) standard serial port emulator protocol**
- **IrBus to provide serial bus access to game ports, joysticks, mice and keyboard**.

IrDA Application layer protocol

- **Specifies security and application**
- **For example, IrDA Alliance Sync protocol used to synchronize mobile devices personal information manager (PIM) data─supports Object Push (PIM) or Binary File Transfer.**

Windows and the several operating systems support

- **Infrared Monitor in Windows monitors the IR port of the IR device.**
- **Detects a nearby IR source.**
- **Controls, detects and selects the IR communication activity.**
- **On command, the device sets up connection using IrDA.**
- **On command starts the IR communication.**
- **When IR communication is inactive, the Monitor enables plug and play (unless disabled).**

Advantages:

IrDA protocol overhead between 2% to 50% of Bluetooth device overhead. Communication setup latency is just few milliseconds.

Disadvantages:

Line of sight and unobstructed communication

ZigBee Wireless Personal Area connected devices

- **IEEE standard 802.15.4 protocol.**
- **Physical layer radio operates 2.4 GHz band carrier frequencies with DSSS**
- **(direct sequence spread spectrum).**
- **Supports range up to 70 m.**
- **Data transfer rate supported 250 kbps.**
- **Supports sixteen channels.**

ZigBee network feature

Self-organising and supports peer-to-peer and mesh networks. Self-organising means detects nearby Zigbee device and establishes communication and network.

Peer-to-peer and mesh network

- **Each node at network function as requesting device as well as responding device.**
- **Mesh network means the each nodes network function as a mesh.**
- **A node can connect to another directly or through mutually interconnected intermediate nodes. Data transfer is between two devices in Peer-to-Peer or between a device**

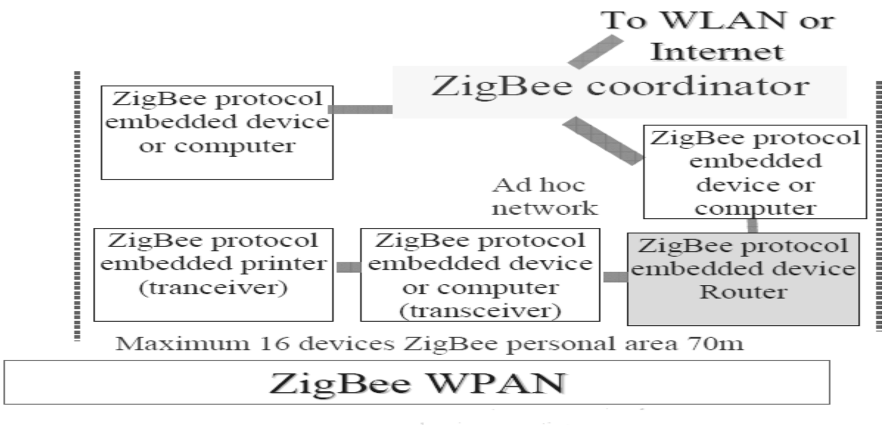


Fig:3.11: ZigBee supporting devices

ZigBee protocol supports large number of sensors, lighting devices, air conditioning, industrial controller and other devices for home and office automation and their remote control and formation of WPAN (wireless personal area network).

ZigBee network

- **Zigbee router - Transfers packets received from a neighboring source to nearby node in the path to destination.**
- **Zigbee coordinator - Connects one Zigbee network with another, or connects to WLAN or cellular network.**
- **Zigbee end devices – Transceiver of data**

ZigBee features

- **Communication latency 30 ms**
- **Protocol stack overhead 28 kB**

802.11 Wireless LAN connected devices

- **IEEE standards 802.11a to 802.11g**
- **802.11a data transfer rates─ 1 Mbps and 2 Mbps**
- **802.11b data transfer rates─ 5.5 Mbps and 2 Mbps**
- **FHSS or DSSS or Infrared 250 ns**

802.11b

- **Called wireless fidelity (WiFi)**
- **802.11b support data rates of 5.5 Mbps by mapping 4 bits**
- **11 Mbps mapping 8 bits simultaneously during modulation.**

Basic service set (BSS)

- **Has one wireless station, which communicates to an access point, also called hotspot.**
- **BSS support ad-hoc network, which as and when node come nearby in range of access point it forms the network through extended service set (ESS).**
- **A node free to move from one BSS to another.**

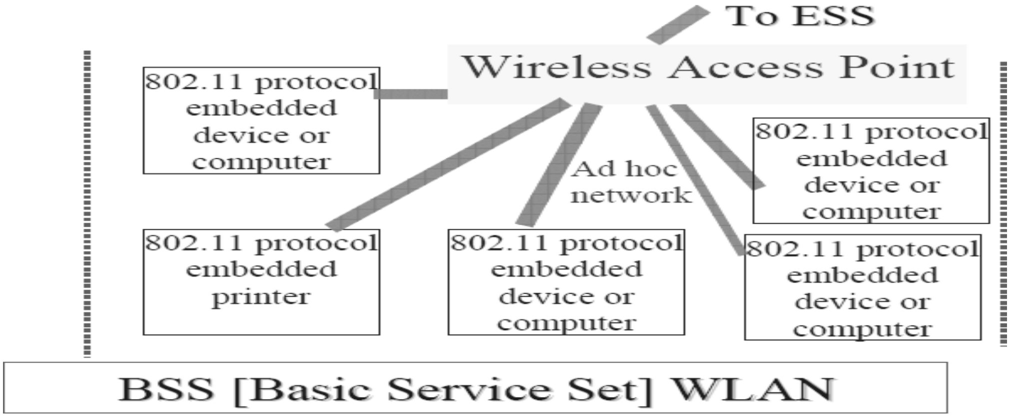


Fig:3.12: Basic service set

Independent basic service set (IBSS)

- No access point.
- Does not connect to the distribution system.
- May have multiple stations, which also cannot communicate among themselves.
- IBSS support ad-hoc network, which as and when nodes come nearby in range they form the network

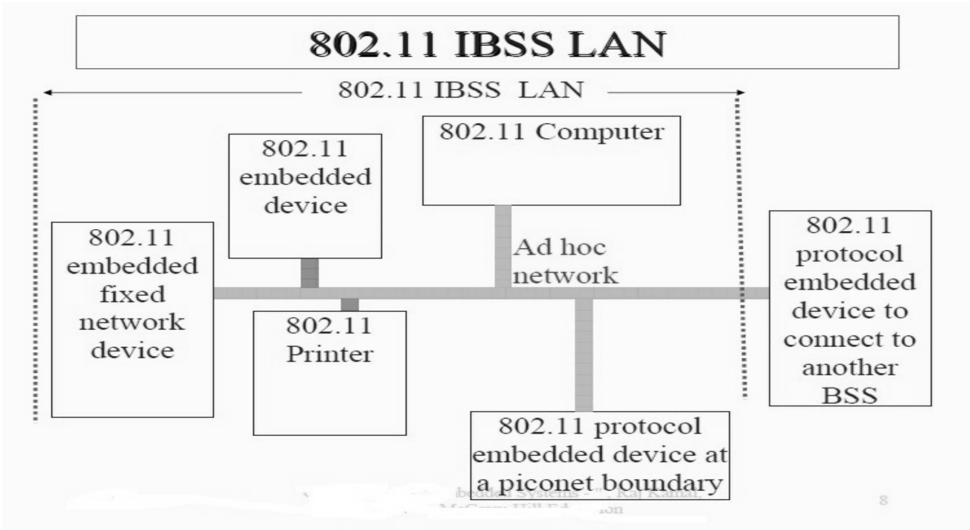


Fig:3.13: Independent basic service set

LAN-station access-points networked together

- **Called extended service set (ESS)**
- **Backbone distribution system.**
- **A backbone set may network through Internet**
- **ESS support fixed infrastructure network**

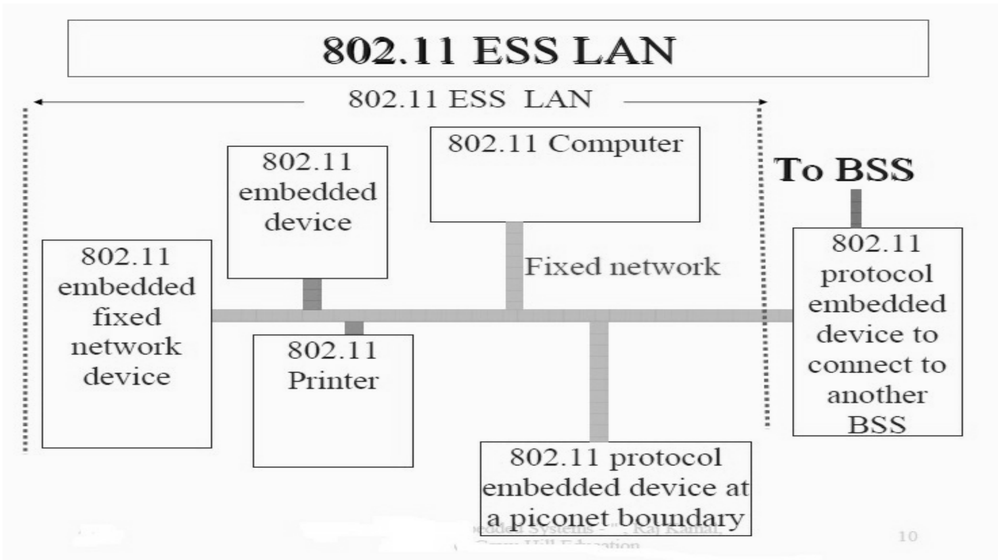


Fig:3.14: Called extended service set

802.11 protocol data link layer

- **Specifies a MAC layer**
- **MAC layer specifies power management, handover and registration of roaming mobile node within the backbone network at a new BSS within the ESS**

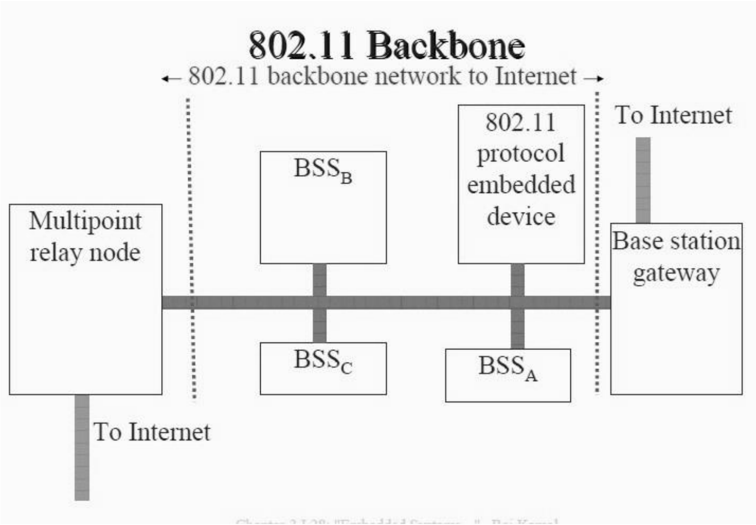


Fig:3.15: 802.11 physical and data link layers

802.11 packet for MAC

- **Packet called request to send (RTS), which is first sent**
- **If other end responses by the packet called clear to send (CTS), then the layer data is transmitted.**

MAC layer

- **Uses carrier sense multiple access and collision avoidance (CSMA/CA) protocol.**
- **A station listens to the presence of carrier during a time interval is called distributed**
- **inter-frame spacing (DIFS) interval.**
- **If carrier is not sensed (detected) during DIFS then the station backs off for a random time interval to avoid collision and retries after that interval.**

802.11 protocol MAC Acknowledgment

- **A receiver always acknowledges within a short inter-frame spacing (SIFS)**
- **Acknowledgment after successful CRC (cyclic redundancy check)**
- **If there is no acknowledgement within SIFS, then transmitter retransmits and upto 7 retransmission attempts are made**

802.11 Physical Layer communication methods

- **Three─ FHSS or DSSS or Infrared 250 ns pulses.**
- **802.11a Physical layer has two sublayers**
- **One is Physical Medium Dependent (PMD) protocol, Physical Layer Convergence Protocol (PLCP)**
- **802.11b additional sub-layer for specifying Complementary Code Keying (CCK)**

Physical Medium Dependent (PMD) protocol 802.11 sublayer - Specifies the modulation and coding methods.

Physical Layer Convergence Protocol (PLCP) 802.11 sub-layer - Specifies the header and payload for transmission. It specifies the sensing of the carrier at receiver. It specifies how packet formation takes place at the transmitter and packets assemble at the receiver. It specifies ways to converge MAC (Medium Access Control) to PMD at transmitter and separate MAC (Medium Access Control) from PMD at the receiver.

An additional sub-layer in 802.11b- Specifying Complementary Code Keying (CCK).

BLUETOOTH

Bluetooth enabled devices:

- **Synchronizing music, image, PIM (personal information manager) files with Computer using Serial emulator at Bluetooth device**
- **Large number of CD players mobile devices are Bluetooth Digital camera**
- **Bluetooth enabled ear buds─ Hands free listening of Bluetooth enabled iPod or CD music player or mobile phone**.

Bluetooth - serial COM port interface

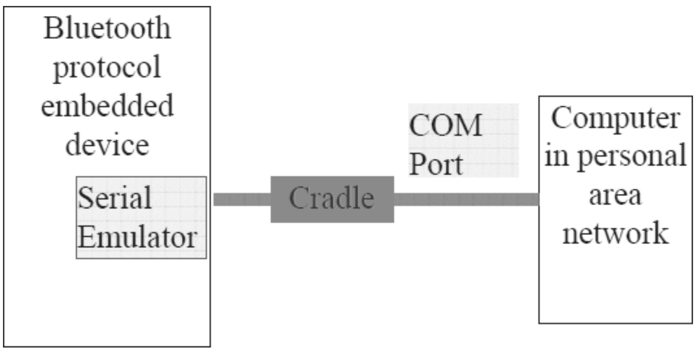


Fig:3.16: Bluetooth wireless protocol

WPAN using Bluetooth wireless protocol

- **Software embeds in the system to support WPAN using Bluetooth wireless protocol**
- **Bluetooth devices─ piconet within 10m**
- **Bluetooth devices─ scatternet within 100m**

- **Data transfer between two devices or between a device and multiple devices**

Bluetooth PICONET

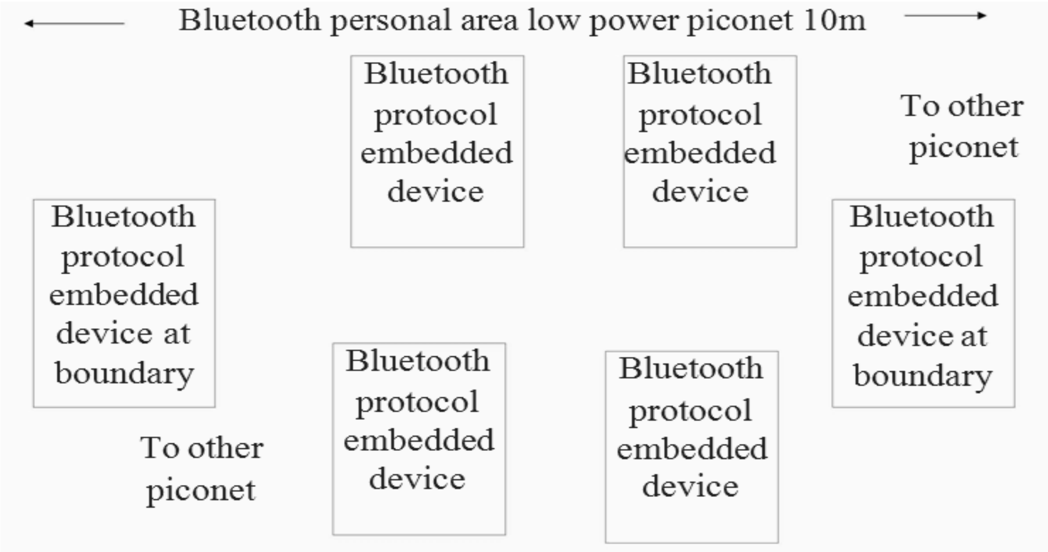


Fig:3.17: Bluetooth PICONET

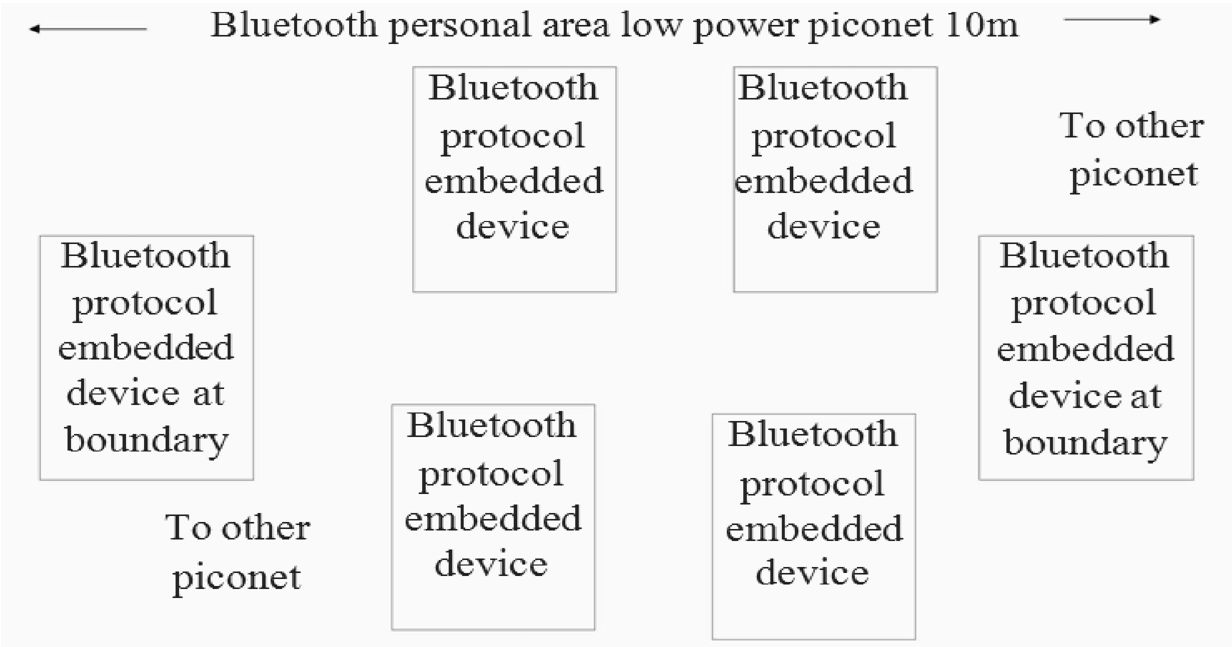


Fig:3.18:Bluetooth scatternet

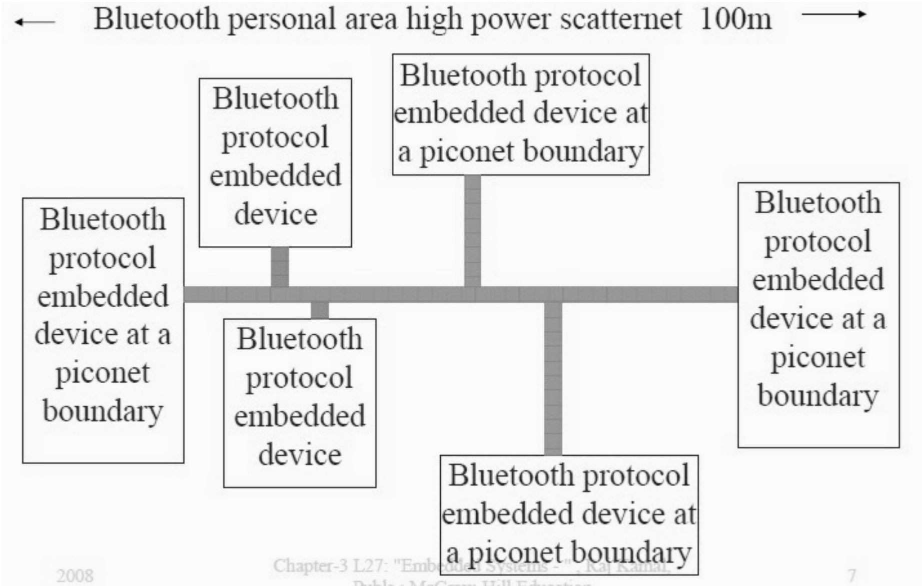


Fig:3.19:Bluetooth Protocol

- **Hopping interval is 625 μs and number of hopped frequencies are 79**
- **Bluetooth 1.x data transfer rate supported = 1 Mbps**
- **Bluetooth 2.0 enhanced maximum data rate of 3.0 Mbps over 100 m**
- **IEEE standard 802.15.1 protocol**
- **Physical layer radio communicates at carrier frequencies in 2.4 GHz band with FHSS (frequency hopping spread spectrum)**

Bluetooth Protocol Features

•Supports automatic self-discovery
•Supports self-organization of network in number of devices.
•Bluetooth device self discovers nearby devices (< 10m) and they synchronize and form WPAN (wireless personal area network).
•Bluetooth protocol supports power control so that the devices communicate at minimum required power level
•This prevents drowning of signals by superimpositions of high power signals with lower level signals

Bluetooth protocol Power control features

- **Bluetooth protocol supports power control so that the devices communicate at minimum required power level**
- **This prevents drowning of signals by superimpositions of high power signals with lower level signals**

Bluetooth Physical Layer

- **Three sub-layers- radio, baseband and link manager or host controller interface**
- **There are two types of links, best effort traffic links and real-time voice traffic**

links

- **The real-time traffic uses reserved bandwidth. Packet is of about 350 bytes**
- **Physical layer─ radio, baseband and link manager or host controller interface**

Link manager sub-layer

.

- **Specifies formation of device pairs for Bluetooth communication.**
- **Gives specifications for state transmission mode, supervision, power level monitoring, synchronization, and exchange of capability, packet flow latency, peak data rate, average data rate, maximum burst size parameters from lower and higher layers.**
- **Manages the master and slave link**
- **Specifies data encryption and device authentication handling.**

Host Controller Interface (HCI) interface

- **Provides for emulation of serial port, for example, 3-wire UART emulation.**
- **Hardware abstraction sub-layer**
- **Used in place of link manager sub-layer**
- **Bluetooth device can thus interface to COM port of computer**

Bluetooth protocol features

- **Communication latency is 3 s.**
- **Large protocol stack overhead of 250kB.**
- **Provision of encrypted secure communication, self-discovery and self-organization and radio based communication between tiny antennae are three main features of Bluetooth**

TEXT / REFERENCE BOOKS

1. KVKK Prasad, "Embedded / Real Time Systems", Dreamtech Press, 2005.
2. David Simon, "An Embedded Software Primer", Pearson Education Asia, First Indian Reprint 2000.
3. Raj Kamal, 'Embedded system-Architecture, Programming, Design', Tata McGraw Hill, 2011.
4. Arnold Berger, "Embedded system design", CMP books, 1st Edition, 2005
5. Wayne Wolf, "Computers as components", Morgan Kaufmann publishers, 2nd Edition, 2008.
6. Tammy Noergaard, "Embedded Systems Architecture", Elsevier, 2006

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT -IV

EMBEDDED SYSTEM – SEC1320

IV EMBEDDED PROGRAMMING

Programming in assembly language (ALP) vs High Level Language - C Program Elements:- Macros and functions, Use of Date Types, Structure, Pointers, Function Calls - Concepts of Embedded Programming in C++- Objected Oriented Programming, Embedded Programming in C++,„C" Program compilers - Cross compiler - Optimization of memory needs-Java programming advantages, disadvantages and J2ME concept.

SOFTWARE PROGRAMMING IN ASSEMBLY LANGUAGE (ALP) AND IN HIGH LEVEL LANGUAGE „C"

***Assembly language coding* of an application has the following advantages:**

1. ***It gives a precise control* of the processor internal devices and full use of processor specific features in its instruction set and its addressing modes.**
2. **The machine codes are compact. This is because the codes for declaring the conditions, rules, and data type do not exist. The system thus needs a smaller memory. Excess memory needed does not depend on the programmer data type selection and rule-declarations. It is also not the compiler specific and library functions specific.**
3. **Device driver codes may need only a few assembly instructions. For example, consider a small-embedded system, a timer device in a microwave oven or an automatic washing machine or an automatic chocolate vending machine. Assembly codes for these can be compact and precise,
and are conveniently written.**

It becomes convenient to develop the *source files* in C or C++ or Java for complex systems because of the following advantages of high-level languages for such systems.

-

1. ***The development cycle is short for complex systems* due to the use of functions (procedures),standard library functions, modular programming approach and top down design. Application programs are structured to ensure that the software is based on sound software engineering principles.**
   (a) **Let us recall Example 4.8 of a UART serial line device driver. Direct use of this function makes the repetitive coding redundant as this device is used in many systems. We simply change some of the arguments (for the variables) passed when needed and use it at another instance of the device use.**
   (b) **Should the square root codes be written again whenever the square root of another value (argument) is to be taken? The use of the standard library *function*, square root ( ), saves the programmer time for coding. New sets of library functions exist in an embedded system specific C or C++ compiler. Exemplary functions are the delay ( ), wait ( ) and sleep ( ).**
   (c) **Modular programming approach is an approach in which the building blocks are reusable software components. Consider an analogy to an IC (Integrated Circuit). Just as an IC has several circuits integrated into one, similarly a building block may call several functions and library functions. A module should however, be well tested. It must have a well-defined goal and the well- defined data inputs and outputs. It should have only one calling procedure. There should be one return point from it. It should not affect any data other than that which is targeted. [Data Encapsulation.] It**

must return (report) error conditions encountered during its execution.

(d) Bottom up design is a design approach in which programming is first done for the sub-modules of the specific and distinct sets of actions. An example of the modules for specific sets of actions is a program for a software timer, RTCSWT:: run. Programs for delay, counting, finding time intervals and many applications can be written. Then the final program is designed. The approach to this way of designing a program is to first code the basic functional modules and then use these to build a bigger module.

(a) Top-Down design is another programming approach in which the *main* program is first designed, then its modules, sub-modules, and finally, the functions.

2. **Data type declarations provide programming ease. For example, there are four types ofintegers, *int*, *unsigned int*, *short* and *long*. When dealing with positive only values, we declare a variable as *unsigned int*. For example, numTicks (Number of Ticks of a clock before the timeout) has to be unsigned. We need a signed integer, *int* (32 bit) in arithmetical calculations. An integer can also be declared as data type, *short* (16 bit) or *long* (64 bit).To manipulate the text and strings for a character, another data type is *char*. *Each data type is an abstraction forthe methods to use, to manipulate, to represent, and for a set of permissible operations*.**
3. ***Type checking* makes the program less prone to error. For example, type checking does notpermit subtraction, multiplication and division on the *char* data types. Further, it lets + be used for concatenation. [For example, micro + controller concatenates into microcontroller, where micro is an array of *char* values and controller is another array of *char* values.]**
4. **Control Structures (for examples, *while*, *do - while*, *break* and *for*) and Conditional Statements(for examples, *if*, *if- else*, *else - if* and *switch - case*) make the program- flow path design tasks simple.**
5. **Portability of non-processor specific codes exists. Therefore, when the hardware changes, only the modules for the device drivers and device management, initialization and locator modules [Section 2.5.2] and initial boot up record data need modifications.**

Additional advantages of C as a high level languages are as follows:

1. **It is a language between low (assembly) and high level language. Inserting the assembly language codes in between is called in-line assembly. A direct hardware**
2. **Control is thus also feasible by inline assembly and the complex part of the program can be in high-level language. Example 4.5 showed the use of in-line assembly codes in C for a Port A Driver Program.**

High level language programming makes the program development cycle short, enables use of the modular programming approach and lets us follow sound software engineering principles. It facilitates the program development with Bottom up design" and „top down design" approaches. Embedded system programmers have long preferred C for the following reasons: (i) The feature of embedding assembly codes using in-line assembly. (ii) Readily available modules in C compilers for the embedded system and library codes that can directly port into the system-programmer codes.

PROGRAM ELEMENTS: MACROS ANDFUNCTIONS

Table 4.1 lists these elements and gives their uses.

***Preprocessor Macros*: A macro is a collection of codes that is defined in a program by a name. It differs from a function in the sense that once a macro is defined by a name, the compiler puts the corresponding codes for it at every place where that macro name appears. The „enable_Maskable_Intr ( )" and „disable_Maskable_Intr ( )" are the macros [The pair of brackets is optional. If it is present, it improves readability as it distinguishes a macro from a constant]. Whenever the name enable_Maskable_Intr appears, the compiler places the codes designed for it. Macros, called test macros or test vectors are also designed and used for debugging a system.**

How does a macro differ from a function? The codes for a function are compiled once only. On calling that function, the processor has to save the context, and on return restore the context. Further, a function may return nothing (*void* declaration case) or return a Boolean value, or an integer or any primitive or reference type of data. [Primitive means similar to an integer or character. Reference type means similar to an array or structure.] The enable_PortA_Intr ( ) and disable_PortA_Intr ( ) are the function calls . [The brackets are now not optional]. Macros are used for short codes only. This is because, if a function call is used instead of macro, the overheads (context saving and other actions on function call and return) will take a time, $T_{overheads}$ that is the same order of magnitude as the time, T_{exec} for execution of short codes within a function. We use a function when the $T_{overheads} << T_{exec}$, and a macro when $T_{overheads}$ ~= or > T_{exec}.

Macros and functions are used in C programs. Functions are used when the requirement is that the codes should be compiled once only. However, on calling a function, the processor has to save the context, and on return, restore the context. Further, a function may return nothing (void declaration case) or return a Boolean value, or an integer or any primitive or reference type of data. Macros are used when short functional codes are to be inserted in a number of places or functions.

Use of Data Types

Whenever a data is named, it will have the address(es) allocated at the memory. The number of addresses allocated depends upon the data type. „C" allows the following primitive data types. The *char* (8 bit) for characters, *byte* (8 bit), *unsigned short* (16 bit), *short* (16 bit), *unsigned int* (32 bit), *int* (32 bit),*long double* (64 bit), *float* (32 bit) and *double* (64 bit). [Certain compilers do not take the „byte" as a data type definition. The char" is then used instead of „byte". Most C compilers do not take a Boolean variable as data type. As in second line of Example 4.6, *typedef*is used to create a Boolean type variable in the C program.]

A data type appropriate for the hardware is used. For example, a 16-bit timer can have only the unsigned short data type, and its range can be from 0 to 65535 only. The typedef is also used. It is made clear by the following example. A compiler version may not process the declaration as an unsigned byte. The „unsigned character" can then be used as a data type. It can then be declared as follows:

```
typedef unsigned character
portAdata #define Pbyte portAdata
Pbyte = 0xF1
```

Use of Data Structures: Queues, Stacks, Lists and Trees

Marks (or grades) of a student in the different subjects studied in a semester are put in a proper table. The table in the mark-sheet shows them in an organised way. When there is a large amount of data, it must be organised properly. A data structure is a way of organising large amounts of data. A data element can then be identified and accessed with the help of a few pointers and/or indices and/or functions. [The reader may refer to a standard textbook for the data structure algorithms in C and C++. For example, "Data Structures and Algorithms in C++" by Adam Drozdek from Brooks/Cole Thomson Learning (2001).]

The queues, stacks and lists, respectively. Table 5.2 gives the uses and show exemplary uses of queues, stacks, arrays, lists and trees.

Table:5.1:Use of the various data structures in a program element

Table 5.1

Uses of the Various Data Structures in a Program Element

Data Structure	Definition and when used	Example (s) of its use
Queue	It is a structure with a series of elements with the first element waiting for an operation. An operation can be done only in the first in first out (FIFO) mode. It is used when an element is not to be accessible by any index and pointer directly, but only through the FIFO. An element can be inserted only at the end in the series of elements waiting for an operation. There are two pointers, one for deleting after the operation and other for inserting. Both increment after an operation.	(1) Print buffer. Each character is to be printed in FIFO mode. (2) Frames on a network [Each frame also has a queue of a stream of bytes.] Each byte has to be sent for receiving as a FIFO. (3) Image frames in a sequence. [These have to be processed as a FIFO.]
Stack	It is a structure with a series of elements with its last element waiting for an operation. An operation can be done only in the last in first out (LIFO) mode. It is used when an element is not to be accessible by any index or pointer directly, but only through the LIFO. An element can be pushed (inserted) only at the top in the series of elements still waiting for an operation. There is only one pointer used for pop (deleting) after the operation as well as for push	(1) Pushing of variables on interrupt or call to another function. (2) Retrieving the pushed data onto a stack.

	(inserting). Pointers increment or decrement after an operation. It depends on insertion or deletion.	
Array (one dimensional vector)	It is a structure with a series of elements with each element accessible by an identifier name and an index. Its element can be used and operated easily. It is used when each element of the structure is to be given a distinct identity by an index for easy operation. Index stars from 0 and is +ve integers.	*ts* = 12 * s(1); Total salary, *ts* is 12 times the first month salary. marks_weight [4] = marks_weight [0]; Weight of marks in the subject with index 4 is assigned the same as in the subject with index 0.
Multi-dimensional array	It is a structure with a series of elements each having another sub-series of elements. Each element is accessible by identifier name and two or more indices. It is used when every element of the structure is to be given a distinct identity by two or more indices for easy operation. The dimension of an array equals the number of indices that are needed to distinctly identify an array-element. Indices start from 0 and are +ve integers.	Handling a matrix or tensor. Consider a pixel in an image frame. Consider Quarter-CIF image pixel in 144 x 176 size image frame. [Recall Section 1.2.7.] *pixel* [108, 88] will represent a pixel at 108-th horizontal row and 88-th vertical column. #See following note also.
List	Each element has a pointer to its next element. Only the first element is identifiable and it is done by list-top pointer (Header). No other element is identifiable and hence is not accessible directly. By going through the first element, and then consecutively through all the succeeding elements, an element can be read, or read and deleted, or can be added to a neighbouring element or replaced by another element.	A series of tasks which are active Each task has pointer for the next task. Another example is a menu that point to a submenu.

Data Structure	Definition and when used	Example (s) of its use
Tree	There is a root element. It has two or more branches each having a daughter element. Each daughter element has two or more daughter elements. The last one does not have daughters. Only the root element is identifiable and it is done by the treetop pointer (Header). No other element is identifiable and hence is not accessible directly. By traversing the root element, then proceeding continuously through all the succeeding daughters, a tree element can be read or read and deleted, or can be added to another daughter or replaced by another element. A tree has data elements arranged as branches. The last daughter, called node has no further daughters. A binary tree is a tree with a maximum of two daughters (branches) in each element.	An example is a directory. It has number of file-folders. Each file-folder has a number of other file folders and so on In the end is a file.

Use of Pointers, NULL Pointers

Pointers are powerful tools when used correctly and according to certain basic principles. Exemplary uses are as follows. Let a byte each be stored at a memory address.

1 **Let a port *A* in system have a buffer register that stores a byte. Now a program using a pointer declares the byte at port A as follows: „unsigned byte *portA‟. [or Pbyte *portA.] The * means „the contents at‟. This declaration means that there is apointer and an unsigned byte for portA, The compiler will reserve one memory address for that byte. Consider „unsigned short **timer1*‟. A pointer *timer1* will point to two bytes, and the compiler will reserve two memory addresses for contents of *timer1*.**

2 **Consider declarations as follows. void *portAdata; The void means the undefined data type for portAdata. The compiler will allocate for the *portAdata without any type check.**

3 **A pointer can be assigned a constant fixed address as in Example 4.5. Recall two preprocessor directives: „# define portA (volatile unsigned byte *) 0x1000‟ and „# define PIOC (volatile unsigned byte *) 0x1001‟. Alternatively, the addresses in afunction can be assigned as follows. „volatile unsigned byte * portA = (unsigned byte *) 0x1000‟ and „volatile unsigned byte *PIOC = (unsigned byte *) 0x1001‟. An instruction, „portA ++;‟ will make the portA pointer point to the next address and to which is the PIOC.**

4 **Consider, unsigned byte portAdata; unsigned byte *portA = &portAdata. The first statement directs the compiler to allocate one memory address for portAdata because there is a byte each at an address. The & (ampersand sign) means „at the address of‟. This declaration means the positive number of 8 bits (byte) pointed by portA is replaced by the byte at the address of portAdata. The right side of the expression evaluates the contained byte from the address, and the left side puts that byte at the pointed address. Since the right side variable portAdata is not a declared pointer, the ampersand sign is kept to point to its address so that the right side pointer gets the contents (bits) from that address. [Note: The equality sign in a program statement means „is replaced by‟].**

5 **Consider two statements, „*unsigned short *timer1;*" and „*timer1*++;". The second statement adds 0x0002 in the address of timer1. Why? timer1 ++ means point to next address, and unsigned short declaration allocated two addresses for timer1. [timer1 ++; or timer1 +=1 or timer = timer +1; will have identical actions.] Therefore, the next address is 0x0002 more than the address of timer1 that was originally defined. Had the declaration been „unsigned int" (in case of 32 bit timer), the second statement would have incremented the address by 0x0004.**

When the index increments by 1 in case of an array of characters, the pointer to the previous element actually increments by 1, and thus the address will increment by 0x0004 in case of an array of integers. For array data type, * is never put before the identifier name, but an index is put within a pair of square brackets after the identifier. Consider a declaration, „*unsigned charportAMessageString [80];*‟. The port A message is a string, which is an array of 80 characters.Now, portAMessageString is itself a pointer to an address without the star sign before it. [Note: Array is therefore known as a reference data type.] However,
***portAMessageString will now**

refer to all the 80 characters in the string. portAMessageString [20] will refer to the twentieth element (character) in the string. Assume that there is a list of RTCSWT (Real Time Clock interrupts triggered Software Timers) timers that are active at an instant. The top of the list can be pointed as „*RTCSWT_List.top‟ using the pointer. RTCSWT_List.top is now the pointer to the top of the contents in a memory for a list of the active RTCSWTs. Consider the statement
„*RTCSWT_List.top ++;*" It increments this pointer in a loop. It *will not point* to the next top of another object in the list (another RTCSWT) but to some address that depends on the memory addresses allocated to an item in the RTCSWT_List. Let *ListNow* be a pointer within the memory block of the list top element. A statement
„RTCSWT_List. ListNow =*RTCSWT_List.top;*" will do the following. RTCSWT_List pointer is now replaced byRTCSWT list-top pointer and now points to the next list element (object). [Note: RTCSWT_List.top ++ for pointer to the next list-object can only be used when RTCSWT_List elements are placed in an array. This is because an array is analogous to consecutively located elements of the list at the memory. Recall Table 5.2.]**

6 **A NULL pointer declares as following: „*#define NULL (void*) 0x0000*‟. [We can assign any address instead of 0x0000 that is not in use in a given hardware.] NULL pointer is very useful. Consider a statement: „*while (* RTCSWT_List. ListNow -*
>state != NULL) { numRunning++;‟. When a pointer to ListNow in a list of software timers that are running at present is notNULL, then only execute the set of statements in the given pair of opening and closing curly braces. One of the important uses of the NULL pointer is in a list. The last element to point to the end of a list, or to no more contents in a queue or empty stack, queue or list.**

Use of Function Calls

There are functions and a special function for starting the program execution, *void main (void)*". Given below are the steps to be followed when using a function in the program.

7. ***Declaring a function:* Just as each variable has to have a declaration, each function must be declared. Consider an example. Declare a function as follows: „*int*run (int *indexRTCSWT,* unsigned int *maxLength,* unsigned int *numTicks,* SWT_Type *swtType,* SWT_Action *swtAction,* boolean *loadEnable*);". Here *int* specifies the returned data type. The run is the function name. There are arguments inside the brackets. Data type of each argument is also declared. A modifier is needed to specify the data type of the returned element (variable or object) from any function. Here, the data type is specified as an integer. [A modifier for specifying the returned element may be also be *static, volatile, interrupt* and *extern*.]**
8. ***Defining the statements in the function:* Just as each variable has to be given the contents orvalue, each function must have its statements. Consider the statements of the function „run". These are *within a pair of curly braces* as follows: „*int RTCSWT:: run (int indexRTCSWT,unsigned int maxLength, unsigned int numTicks, SWT_Type swtType, SWT_Action swtAction, boolean loadEnable) {...};*". The last statement in a function is for the *return* and may also befor returning an element.**
9. ***Call to a function:* Consider an example:„if (delay_F = = true && SWTDelayIEnable = = true)ISR_Delay ( );". There is a call on fulfilling a condition. The call can occur several times and can be repeatedly made. On each *call,* the values of the arguments given within the pair of bracket pass for use in the function statements.**

(i) Passing the Values (elements)

The values are copied into the arguments of the functions. When the function is executed in this way, it does not change a variable"s value at the*called* program. A function can only use the copied values in its own variables through the arguments. Consider a statement, „run (int *indexRTCSWT,* unsigned int *maxLength,* unsigned int *numTicks,* SWT_Type *swtType,* SWT_Action *swtAction,* boolean*loadEnable*) {...}". Function „run" arguments *indexRTCSWT, maxLength, numTick, swtType,* and *loadEnable* original values in the calling program during execution of the codes will remain unchanged.The advantage is that the same values are present on return from the function. The arguments that are *passed by the values* are saved temporarily on a stack and retrieved on return from the function.

(ii) Reentrant Function

Reentrant function is usable by the several tasks and routines synchronously (at the same time). This is because all its argument values are retrievable from the stack. A function is called*reentrant func-tion* when the following three conditions are satisfied.

1. **All the arguments pass the values and none of the argument is a pointer (address) whenever a calling function calls that function. There is no pointer as an argument in the above example offunction „run".**
2. **When an operation is not atomic, that function should not operate on any variable, which is declared outside the function or which an interrupt service routine uses or which is a global variable but passed by reference and not passed by value as an argument into the function. [The value of such a variable or variables, which is not local, does not save on the stack when there is call to another program.]**

The following is an example that clarifies it further. Assume that at a server (software), there is a 32 bit variable *count* to count the number of clients (software) needing service. There is no option except to declare the *count* as a global variable that shares with all clients. Each client on a connection to a server sends a call to increment the *count*. The implementation by the assembly code for increment at that memory location is non-atomic when (*i*) the processor is of eight bits, and (ii) the server-compiler design is such that it does not account for the possibility of interrupt in-between the four instructions that implement the increment of 32-bit count on 8-bit processor. There will be a wrong value with the server after an instance when interrupt occurs midway during implementing an increment of *count*.

3. ***That function does not call any other function that is not itself Reentrant.* Let RTI_Count be aglobal declaration. Consider an ISR, *ISR_RTI*. Let an „*RTI_Count ++;*" instruction be where the RTI_Count is variable for counts on a real-time clock interrupt. Here *ISR_RTI* is a not a Reentrant routine because the second condition may not be fulfilled in the given processor hardware. There is no precaution that may be taken here by the programmer against shared data problems at the address of the *RTI_Count* because there may be no operation that modifies RTI_Counts in any other routine or function than the *IST_RTI*. But if there is another operation that modifies the RTI_Count the shared-data problem will arise.**

(iii) Passing the References

When an argument value to a function passes through a pointer, the function can change this value. On returning from this function, the new value will be available in the calling program or another function called by this function. [There is no saving on stack of a value that either (*a*) *passes through a pointerin the function-arguments* or (*b*) operates in the function as a global variable or (c) operates through a variable declared outside the function block.4.7 Multiple Function Calls in Cyclic Order in the Main.

One of the most common methods is for the multiple function-calls to be made in a cyclic order in an infinite loop of the *main*. Recall the 64 kbps network problem. Let us design the C codes for an infinite loop for this problem. Example 5.4 shows how the multiple function calls are defined in the main for execution in the cyclic orders. Figure 5.1 shows the model adopted here.

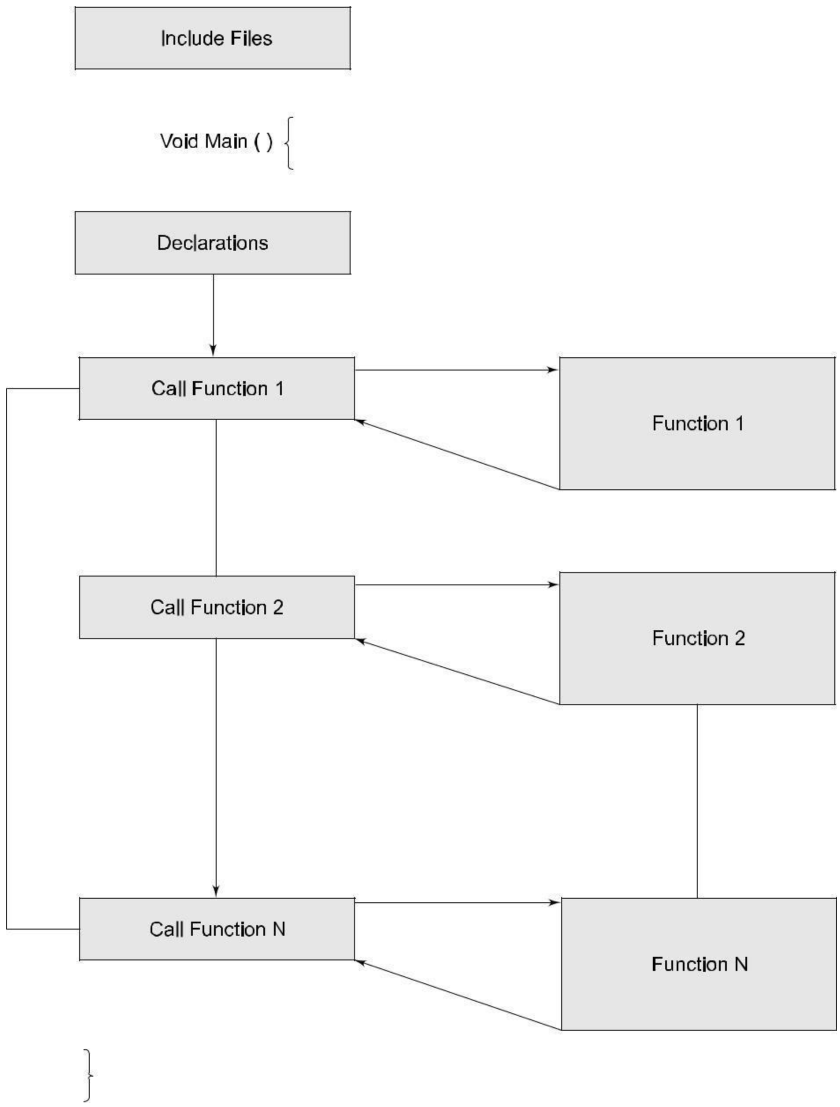


Fig:5.1: Programming model for multiple function calls

.

Example 5.4

```
typedef unsigned char
int8bit; # define int8bit
boolean
# define false 0
# define true 1
void main (void) {
/* The Declarations of all variables, pointers, functions here and also
initializations here */
unsigned char *portAdata;
boolean        charAFlag;
boolean checkPortAChar ( );
void inPortA (unsigned char
*);
void decipherPortAData (unsigned char
*); void encryptPortAData (unsigned
char *); void outPortB (unsigned char
*);
.
while (true) {
/* Codes that repeatedly execute */
```

```
/* Function for availability check of a
 character at port A*/ while (charAFlag !=
 true) checkPortAChar ( );
/* Function for reading PortA
 character*/ inPortA (unsigned char
 *portAdata);
/* Function for deciphering */
 decipherPortAData (unsigned char  *portAdata);
/* Function for encoding */
 encryptPortAData       (unsigned  char   *portAdata);
/* Function for retransmit output to
 PortB*/ outPort B (unsigned char
 *portAdata);
 };
}
```

Function Pointers, Function Queues and Interrupt Service Routines Queues

Let the * sign not be put before a function name, but there are arguments within the pair of brackets, and the statements for those executed on a call for the function. The statements are inside a pair of the curly braces. Consider a declaration in the Example 5.4, „*boolean checkPortAChar* ( );".„*checkPortAChar*" is a function, which returns a *Boolean* value. Now, *checkPortAChar* is itself a pointer to the starting address of the statements of the function inside the curly braces without the star sign before it. The program counter will fetch the address of *checkPortAChar*, and CPU sequentially executes the function-statements from here.

Now, let the * sign be put before the function. „* *checkPortAChar*" will now refer to all the compiled form statements in the memory that are specified within the curly braces. Consider a declaration in the example, „void *inPortA* (unsigned char *);".

1. *inPortA*means a pointer to the statements of the function. Inside the bracket, there is anunsigned character pointed by some pointer.
2. **inPortA* will refer to all the compiled form statements of *inPortA*.
3. (* *inPortA*) will refer to calls to the statements of *inPortA*.
4. What will a statement, „void *create* (void (**inPortA*) (unsigned char *), void *portAStack, unsigned char port Apriority);" mean?
   (a) First modifier „void" means *create* function does not return any thing.
   (b) „create" is another function.
   (c) Consider the argument of this function „void (**inPortA*) (*unsigned char *portAdata*)". (**inPortA*) means call the statements of *inportA* the argument of which is „unsigned char *portAdata".
   (d) The second argument of *create* function is a pointer for the portA stack at the memory.
   (e) The third argument of *create* function is a byte that defines the portA priority.

An important lesson to be remembered from above discussion is that a returning data type specifi-cation (for example, void) followed by „*(*functionName) (functionArguments)*" calls the statements of the *functionName* using the *functionArguments*, and on a return it returns the specified data object. We can thus use the function pointer for invoking a call to the function.

When there are multiple ISRs, a high priority interrupt service routine is executed first and the lowest priority, last. [Refer Section 4.6.4.] It is possible that function calls and statements in any of the higher priority interrupts may block the execution of low priority ISR within the deadline. How is the deadline problem for low priority routines to be solved? One solution is by using the function pointers in the routines, and forming a queue for them. The functions are then executed at a later stage. Pointers are needed in number of situations, for example, port bit manipulation and read or write. Software designers must learn the uses of pointers in depth. An innovative concept is use of function queues and the queues of the function pointers built by the ISRs. It reduces significantly the ISR latency periods. Each device ISR is therefore able to execute within its stipulated deadline.

Objected Oriented Programming

An objected oriented language is used when there is a need for re-usability of the defined object or set of objects that are common within a program or between the many *applications*. When a large pro-gram is to be made, an object-oriented language offers many advantages. Data encapsulation, design of reusable software components and inheritance are the advantages derived from the OOPs.

An object-oriented language provides for defining the objects and methods that manipulate the objects without modifying their definitions. It provides for the data and methods for encapsulation. An object can be characterized by the following:

An i*dentity* (a reference to a memory block that holds its state and behavior).
A *state* (its data, property, fields and attributes).
A *behavior* (method or methods that can manipulate the *state* of the object).

In a procedure-based language, like FORTRAN, COBOL, Pascal and C, large programs are split into simpler functional blocks and statements. In an object-oriented language like Smalltalk, C++ or Java, logical groups (also known as *classes*) are first made. Each group defines the data and the methods of using the data. A set of these groups then gives an application program. Each group has internal user-level fields for the data and the methods of processing that data at these fields. Each group can then create many objects by copying the group and making it functional. Each object is functional. Each object can interact with other objects to process the user‟s data. The language pro-vides for formation of classes by the definition of a group of objects having similar attributes and common behavior. A class *creates the objects. An object is an instance of a class.*

5.8.2 Embedded Programming in C++

1. What are programming advantages of C++?

C++ is an *object oriented Program (OOP) language, which in addition, supports the procedure ori-ented codes of C.* Program coding in C++ codes provides the advantage of objected oriented program-ming as well as the advantage of C and in-line assembly. Programming concepts for embedded pro-gramming in C++ are as follows:

(i) A class binds all the member functions together for creating objects. The objects will have memory allocation as well as default assignments to its variables that are not declared*static*. Let us assume that each software timer that gets the count input from a real time clock is an object. Now consider the codes for a C++ *class RTCSWT*. A number of software timer objects can be created as the instances of *RTCSWT*.

(ii) A class can derive (inherit) from another class also. Creating a *child* class from RTCSWT as a *parent* class creates a new application of the RTCSWT.

(iii) Methods (C functions) can have same name in the inherited class. This is called method over loading. Methods can have the same name as well as the same number and type of arguments in the inherited class. This is called *method overriding*. These are the two significant features that are extremely useful in a large program.

(*iv*) Operators in C++ can be overloaded like in method overloading. Recall the following statements and expressions in Example 5.8. The operators ++ and ! are overloaded to perform a set of operations. [Usually the++ operator is used for post-increment and pre-increment and the ! operator is used for a *not* operation.]

```
const OrderedList & operator ++ ( ) {if (ListNow != NULL) ListNow = ListNow -
> pNext;
return *this;}
boolean int OrderedList & operator ! ( ) const {return (ListNow != NULL) ;};
```

[Java does not support operator overloading, except for the + operator. It is used for summation as well string-concatenation.]

There is *struct* that binds all the member functions together in C. But a C++ *class* has object features. It can be extended and child classes can be derived from it. A number of child classes can be derived from a common class. This feature is called polymorphism. A class can be declared as public or private. The data and methods access is restricted when a class is declared private. *Struct* does not have these features.

2. What are then the disadvantages of C++ ?

Program codes become lengthy, particularly when certain features of the standard C++ are used.

Examples of these features are as follows:

(a) Template.
(b) Multiple Inheritance (Deriving a class from many parents).
(c) Exceptional handling.
(d) Virtual base classes.
(e) Classes for IO Streams. [Two library functions are *cin* (for character (s) in) and *cout* (for character (s) out). The I/O stream class library provides for the input and output streams of characters (bytes). It supports *pipes*, *sockets* and *file management features*. Refer to Section 8.3 for the use of these in inter task communications.]

3. Can optimization codes be used in Embedded C++ programs to eliminate the disadvan-tages?

Embedded system codes can be optimised when using an OOP language by the following

(a) Declare private as many classes as possible. It helps in optimising the generated codes.
(b) Use *char*, *int* and *boolean* (scalar data types) in place of the objects (reference data types) as arguments and use local variables as much as feasible.
(c) Recover memory already used once by changing the reference to an object to NULL.

A *special compiler for an embedded system* can facilitate the disabling of specific features provided in C++. Embedded C++ is a version of C++ that provides for a selective disabling of the above features so that there is a less runtime overhead and less runtime library. The solutions for the library functions are available and ported in C directly. The IO stream library functions in an embedded C++ compiler are also reentrant. So using embedded C++ compilers or the special compilers make the

C++ a significantly more powerful coding language than C for embedded systems.

GNU C/C++ compilers (called *gcc*) find extensive use in the C++ environment in embedded soft-ware development. Embedded C++ is a new programming tool with a compiler that provides a small runtime library. It satisfies small runtime RAM needs by selectively de-configuring features like, tem-plate, multiple inheritance, virtual base class, etc. when there is a less runtime overhead and when the less runtime library using solutions are available. Selectively removed (de-configured) features could be template, run time type identification, multiple Inheritance, exceptional handling, virtual base classes, IO streams and foundation classes. [Examples of foundation classes are GUIs (graphic user interfaces). Exemplary GUIs are the buttons, checkboxes or radios.]

An embedded system C++ compiler (other than *gcc*) is Diab compiler from Diab Data. It also provides the target (embedded system processor) specific optimisation of the codes. [Section 5.12] The run-time analysis tools check the expected run time error and give a profile that is visually interactive.

Embedded C++ is a C++ version, which makes large program development simpler by providing object-oriented programming (OOP) features of using an object, which binds state and behavior and which is defined by an instance of a class. We use objects in a way that minimizes memory needs and run-time overheads in the system. Embedded system programmers use C ++ due to the OOP features of software re-usability, extendibility, polymorphism, function overriding and overloading along portability of C codes and in-line assembly codes. C++ also provides for overloading of operators. A compiler, gcc, is popularly used for embedded C++ codes compilation. Diab compiler has two special features: (i) processor specific code optimization and (ii) Run time analysis tools for finding expected run-time errors.

Object Oriented Programming

An object oriented language is used when there is a need for re-usability of the defined object or set of object that are common within a program or between many applications. When a large program is to be made an object oriented programming language offers many advantages. Data encapsulations, design of reusable software components and inheritance are the advantages derived from the OOPs.

An object oriented language provides for defining the objects and methods that manipulate the objects without modifying their definitions. It provides for the data and the methods for encapsulation. An object can be characterized by the following;

1. **An identity**
2. **A state**
3. **A behavior**

In a procedure Based Language like FORTRAN, COBOL, Pascal and C, large programs are split into simpler functional blocks and statements. In an object oriented language like small talk, C++ or java logical groups are first made. Each group defines the data and the methods of using the data. Each group has internal user level fields for the data and the methods of processing that data at these fields. Each group can then create many objects to process the user data. The language provides the formation of classes by the definition of a group of objects having similar attributes and common behavior. A class creates the objects. An object is an instance of a class

Embedded Programming in C++

C++ is an object oriented language which in addition supports the procedure oriented codes of C. program coding in C++ codes provides the advantage of object oriented programming as well as the advantage of C and in-line assembly.

Programming concepts for embedded programming in C++ are as follows;

i. **A class binds all the member function together for creating objects. The objects will have memory allocation as well as default assignments to its variables that are not declared static. Let us assume that each software timer that gets the count input from a real time clock is an object. A number of software timer objects can be created as the instances of RTCSWT**
ii. **A class can derive from another class also. Creating a child class from RTCSWT as a parent class creates a new application of the RTCSWT**
iii. **Methods can have same name in the inherited class. This is called method overloading. Methods can have the same name aswell as the same number and type of arguments in the inherited class. This is called method overriding. These are the two significant features that are extremely useful in a large program**
iv. **Operators in C++ can be overloaded like in method overloading. There is „struct" that binds all the member function together in C. but a C++ class has object features. It can be extended and child classes can be derived from it. A number of child classes can be derived from a common class. This feature is called polymorphism. A class can be declared as private or public. The data and methods access is restricted when a class is declared as private. „struct" does not have thesefeatures**

Disadvantages of C++

Program codes become lengthy particularly when certain features of the standard used. Example of these features are as follows;

a. **Template**
b. **Multiple inheritance**
c. **Exceptional handling**
d. **Virtual base classes**
e. **Classes for IO streams**

Embedded system codes can be optimized when using the OOP language

i. **Declare private as many classes as possible. It helps in optimizing the generated codes**
ii. **Use „char", int and Boolean in place of the objects as arguments and use local variables as much as feasible**
iii. **Recover memory already used once by changing the reference to an object to NULL**

A Special compiler for an embedded system can facilitate the disabling of specific features provided in C++. The solution for the library functions are available and ported in C directly. The IO stream library function is an embedded C++ compiler are also reentrant. So using embedded C++ compilers or the special compilers make the C++ significantly more powerful coding language than C for embedded systems

4.2 C Program compilers and cross compilers

Two compilers are needed. One is c program compiler and the other one is cross compiler. Compiler is for the host computer which does the development and design and also the testing and debugging.

The second compiler is the cross compiler. The cross compiler runs on the host but develops the machine codes for a targeted system. There is a popular free ware called GNU. A GNU compiler is configurable both as host compiler as well as cross compiler. A compiler generates an object file. For compilation of the host alone the compiler can be turbo C, turbo C++ and Borland C++. The target system specific or multi choice cross compilers that are available commercially may be used. These are available for most of the embedded systems, microprocessor and microcontrollers. The host runs the cross compilers that offer an integrated development environment. It means that a target system can emulate and simulate the application

system on the host.

Use of appropriate compilers and cross compilers is essential in any embedded software development

Optimization of memory needs

When codes are made compact and fitted in small memory areas without affecting the code performance, it is called memory optimization. It also reduces the total number of CPU cycles and thus the total energy requirements. The following are used to optimize the use of memory in a system;

i. **Use declaration as unsigned byte if there is a variable, which always has a value between 0 and 255. When using the data structures, limit the maximum size of queues, list and stacks size to 256. Byte arithmetic takes less time than integer arithmetic**
ii. **Avoid use of library functions if a simpler coding is possible. Library functions are the general functions. Use of general function needs more memory in several cases**
iii. **When the software designer knows fully the instruction set of the target processor, assembly codes must be used. This also allows the efficient use of memory. The device driver programs in assembly especially provides efficiency due to the need to use the bit set-reset instruction for the control and status registers. Only the few assembly codes for using the device IO port, addresses and control and status registers are needed. Assembly coding also helps in coding for atomic operations. A modifier register can be used in the C program for a fast access to a frequently used variable**
iv. **Calling a function causes context saving on a memory stack and on return the context is retrieved. This involves time and can increase the worst case interrupt- latency. There is a modifier in-line. When the in-line modifier is used the compiler inserts the actual code at all the places where these operations are used. This reduces the time and stack overheads in the function call and return. Using modifier directs the compiler to put the codes for the function instead of calling that function**
v. **As long as shared data problem doesnot arise, the use of global variables can be optimized. These are not used as the arguments for passing the values. A good function is one that has no arguments to be passed. The passed values are saved on the stacks in case of interrupt service calls and other function calls. Besides obviating the need for repeated declaration the use of global variables will thus reduce the worst case interrupt latency and the time and the stack overheads in the function call and return. But this is at the cost of the codes for eliminating shared data problem. When a variable is declared static the processor access with less instruction than from the stack.**
vi. **Combine two function if possible. The search function for finding pointers to a list item and pointers of previous list items combine into one. It present is false the pointer of the previous list item retrieves the one that has the item**
vii. **All the timers and a conditional statement that changes the count input in case of a running count and does not change it in case of ideal state timers could have also been used. More number of calls will however be needed and not once but repeatedly on each real time clock interrupt tick. The RAM memory needed will be more. Their fore creating a list of running counts is a more efficient way, similarly bringing the task first into an initiated task list will reduce the frequent**

interaction with the OS and context savings and retrieval stack and time overheads.

viii. Use of feasible alternatives to the switch statements with a table of pointers to the functions. This saves processor time in deciding which set of statements to execute while performing the conditional tests all down a chain.

ix. Use the delete function when there is no longer a need for a set of statements after that execute. As a rule to free the RAM used by a set of statements use the delete function and destructor functions.

x. When using C++, configure the compiler for not permitting the multi-inheritance, templates,exceptional handling,new style casts, virtual base classes and name spaces.

Embedded programming in Java

Java programming advantages

Java has advantages for embedded programming as

i. Java is completely an OOP language
ii. Java has an in-built support for creating multiple threads. It obviates the need for an operating system based scheduler for handling the tasks.
iii. Java is the language for the most web applications and allows machines of different types to communicate on the web.
iv. There is a huge class library on the network that makes program development quick.
v. Platform independence in hosting the compiled codes on the network is because java generates the byte codes. These are executed on an installed JVM (Java Virtual Machine) on a machine. Platform independence gives portability with respect to the processor used.
vi. Java doesnot permit pointer manipulation instructions. So it is robust in the sense that memory leaks and memory related errors do not occur. A memory leak occurs, for example when attempting to write to the end of a boundary array
vii. Java byte codes that are generated need a large memory when a method has more than three or four local variables
viii. Java being platform independent is expected to run on a machine with an RISC like instruction execution with few addressing modes only.

Java programming disadvantages

Use of J2ME (Java 2 Micro Edition) or java card or embedded java helps in reducing the code size to 8KB for the usual applications like smart card.

i. Use core classes only. Classes for basic run time environment form the VM internal format and only the programmers new java classes are not in internal format.
ii. Provide for configuring the run time environment.
iii. Create one object at a time when running the multiple threads.
iv. Reuse the objects instead of using a larger number of objects
v. Use scalar type only as long as feasible

A smart card is an electronic circuit with a memory and CPU or a synthesized VLSI circuit. It is packed like an ATM card. For smart cards, there is Java card technology. Internal formats for the run time environment are available mainly for the few classes in Java card technologies. Java classes used are the connections, data grams, input, output and streams, security and cryptography. Hence these are the advantages and disadvantages of Java applications in the embedded system. Java card, embedded Java and J2ME are the three versions of Java that generate a reduced code size. Cons.

Consider an embedded system such as a smart card, it is a simple application that uses a running Java card. The Java advantage of platform independency in byte codes is an asset. The smart card connects to a remote server. The card stores the user account past balance and user details for the remote server information in an encrypted format. It deciphers and communicates to the server the user needs after identifying and certifying the user. The intensive codes for the complex applications run at the server. A restricted run time environment exist in java classes for connections, datagrams, character input, output and streams, security and cryptography only

J2ME (Java 2 Micro Edition) concept

J2ME provides the optimized run time environment. Instead of the use of packages, J2ME provides for the codes for the core classes only. These codes are stored at the ROM of the embedded system. It provides for two alternative configurations.

- Connected Device Configuration(CDC) and
- Connected Limited Device Configuration (CLDC).

CDC inherits a few classes from packages for net, security, input output, reflect, until, jar and zip. CLDC doesnot provide for the applets, beans, math, security and text packages in java. Lang. A Personal Digital Assistant uses CDC or CLDC.

There is a scalable OS feature in J2ME. There is a new virtual machine, KVM as an alternative to JVM. When using the KVM, the system needs a 64KB instead of 512KB run time environment.

J2ME need not be restricted to configure the SVM to limit the classes. The configuration can be augmented by profiler classes. For example Mobile Information Device Profiler (MIDP). A profile defines the support of Java to a device family. The profiler is a layer between the application and the configuration. For example MIDP is between CLDC and application. Between the devices and configuration there is an OS, which is specific to the device needs

TEXT / REFERENCE BOOKS

1. KVKK Prasad, “Embedded / Real Time Systems”, Dreamtech Press, 2005.
2. David Simon, “An Embedded Software Primer”, Pearson Education Asia, First Indian Reprint 2000.
3. Raj Kamal, ‘Embedded system-Architecture, Programming, Design’, Tata McGraw Hill, 2011.
4. Arnold Berger, “Embedded system design”, CMP books, 1st Edition, 2005
5. Wayne Wolf, “Computers as components”, Morgan Kaufmann publishers, 2nd Edition, 2008.
6. Tammy Noergaard, “Embedded Systems Architecture”, Elsevier, 2006

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

UNIT -V

EMBEDDED SYSTEM – SEC1320

UNIT V - EMBEDDED SYSTEM TESTING AND APPLICATION

Introduction to embedded system testing - Types of testing: Unit testing, Regression testing, Functional testing, Coverage tests, Gray box test and performance testing - Embedded applications: Case study of Smart card, Interfacing stepper motor, RFID-system, Application, Tag Reader - Handheld Device - Washing Machine.

INTRODUCTION TO EMBEDDED SYSTEM TESTING

TESTING EMBEDDED SOFTWARE:

Analogous to most software systems, testing embedded software is an integral part of the software development life cycle. To ensure the robustness of embedded software, both its functional and non-functional properties need to be examined. In the following discussion, we outline some salient features that make the testing of embedded systems unique and challenging, compared to traditional software systems.

Generally the traits that separate embedded software from applications software are:

- **Embedded software must run reliably without crashing for long periods of time.**
- **Embedded software is often used in applications in which human lives are at stake.**
- **Embedded systems are often so cost-sensitive that the software has little or no margin for inefficiencies of any kind.**
- **Embedded software must often compensate for problems with the embedded hardware.**
- **Real-world events are usually asynchronous and nondeterministic, making simulation tests difficult and unreliable.**

Reasons for testing:

To find bugs in software (testing is the only wayto do this)
To reduce risk to both users and the company
To reduce development and maintenance
costs To improve performance

To Find the Bugs

It‘s important to remember that testing isn‘t about proving the –correctness‖ of a program but about finding bugs. Experienced programmers understand that every program has bugs. The only way to know how many bugs are left in a program is to test it with a carefully designed and measured test plan.

To Reduce Risk

The objectives in testing are to demonstrate to user (and regulatory agencies, if appropriate) that the system and software works correctly and as designed. It assured that the product is as safe as it can be. In short, user wants to discover every conceivable fault or weakness in the system and software before it‘s deployed in the field.

To Reduce Costs

The earlier a bug is found, the less expensive it is to fix. The cost of finding errors and bugs in a released product is significantly higher than during unit testing, for example (Figure 5.1).

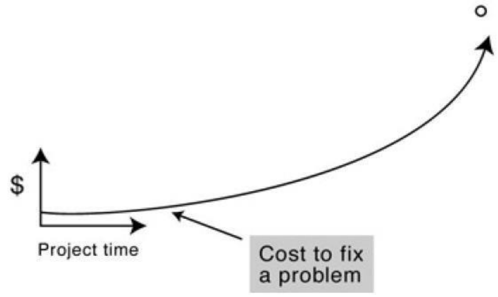


Fig: 5.1: The cost to fix a problem.

Simplified graph showing the cost to fix a problem as a function of the time in the product life cycle when the defect is found. The costs associated with finding and fixing the Y2K problem in embedded systems is a close approximation to an infinite cost model.

To Improve Performance

Testing maximizes the performance of the system. Finding and eliminating dead code and inefficient code can help ensure that the software uses the full potential of the hardware and thus avoids the dreaded –hardware re-spin.

Best Time to Test:

It should be clear from Figure 5.1 that testing should begin as soon as feasible. Usually, the earliest tests are module or unit tests conducted by the original developer. Unfortunately, few developers know enough about testing to build a thorough set of test cases. Because carefully developed test cases are usually not employed until integration testing, many bugs that could be found during unit testing are not discovered until integration testing. For example, a major network equipment manufacturer in Silicon Valley did a study to figure out the key sources of its software integration problems. The manufacturer discovered that 70 percent of the bugs found during the integration phase of the project were generated by code that had never been exercised before that phase of the project.

2

Testing Functional Properties

The functionality of software systems, captures the way such systems should behave. Therefore, testing functional properties is a critical phase for all applications. Typically, the functionality testing of software aims to discover –buggy‖ scenarios. For instance, such buggy scenarios may capture the violation of software behavior with respect to the specification or an implementation bug (eg, null pointer dereference and assertion failure). To discover and investigate a buggy scenario, the designer must be provided with appropriate test inputs that trigger the respective bug. Therefore, software testing tools should have a clear domain knowledge of the relevant inputs to the system.

For embedded software, the functionality is often (partially) controlled by the physical environment. Such physical environment might include air pressure, temperature, physical movement, among others. Unfortunately, the physical environment, where an embedded software is eventually deployed, is often not present during the testing time. For instance, consider the fall-detection application, it is crucial that the designed software invokes appropriate actions according to the movement of the patient. In the actual working environment, such movements are sampled from sensor inputs. for embedded software, whose functionality might depend on air pressure or temperature, the testing process should ensure that the respective software acts appropriately in different environmental conditions. In general, to simulate the physical environment, the designer may potentially take the following approaches:

The physical environment (eg, inputs read from sensors) might be made completely unconstrained during the time of testing. This enables the testing of software under all operating conditions of the physical environment. However, such an approach might turn infeasible for complex embedded software. Besides, unconstraining the physical environment might lead to unnecessary testing for irrelevant inputs. Such inputs may include sensor readings (such as 300 K for air temperature readings) that may never appear in the environment where the software is deployed.

The physical environment might be simulated by randomly generating synthetic inputs (eg, generating random temperatures readings). However, such an approach may fail to generate relevant inputs. However, like traditional software testing, search-based techniques might improve the simulation of physical environment via evolutionary methods and metaheuristics.

With a clear knowledge of the embedded software, the testing process can be improved. For instance, in the fall-detection system, it is probably not crucial to simulate the movement for all possible movement angles. It is, however, important to test the application for some inputs that indicate a fall of the patient (hence, indicating safety) and also for some inputs that does not capture a fall (hence, indicating the absence of false positives). In general, building such abstractions on the input space is challenging and it also requires a substantial domain knowledge of the input space.

We shall now discuss some non-functional properties that most embedded software is required to satisfy.

Timing Constraints:

Timing constraints capture the criteria to complete tasks within some time budgets. The violation of such constraints may lead to a complete failure of the respective software. This, in turn, may have serious consequences. For instance, consider the fall-detection application. The computation of a potential fall should have real-time constraints. More precisely, the time-frame between the sampling of sensor inputs and triggering an alarming situation should have strict timing constraints. Violation of such constraints may lead to the possibility of detecting a fall too late, hence making the respective software impractical.

Therefore, it is crucial that the validation process explicitly targets to discover the violation of timing-related constraints. It is, however, challenging to determine the timing behavior of an application, as the timing critically depends on the execution platform.

The execution platform, in turn, may not be available during the testing phase. As a result, the validation of timing-related constraints, may often involve building a timing model of the underlying execution platform. Such a timing model should be able to estimate the time taken by each executed instruction. In general, building such timing models is challenging. This is because, the time taken by each instruction depends on the specific instruction set architecture (ISA) of the processor, as well as the state of different hardware components (eg, cache, pipeline, and interconnect).

Energy Constraints:

Like timing, energy consumption of embedded software may also need careful consideration. In particular, if the respective software is targeted for a battery-operated device, the energy consumption of the software may pose a serious bottleneck. For instance, if a fall-detection software is battery-operated, the power drained from the battery should be acceptable in a way to trigger the alarming situation. Like timing, the energy consumption of software is also highly sensitive to the underlying execution platform. Therefore, in the absence of the execution platform, an appropriate energy-model needs to be developed. Such an energy model can be used during the test time to estimate the energy consumption of software and to check whether the software satisfies certain energy constraints. Similar to timing constraints, energy constraints can be captured systematically via assertions or via computing the worst case energy consumption of the respective software.

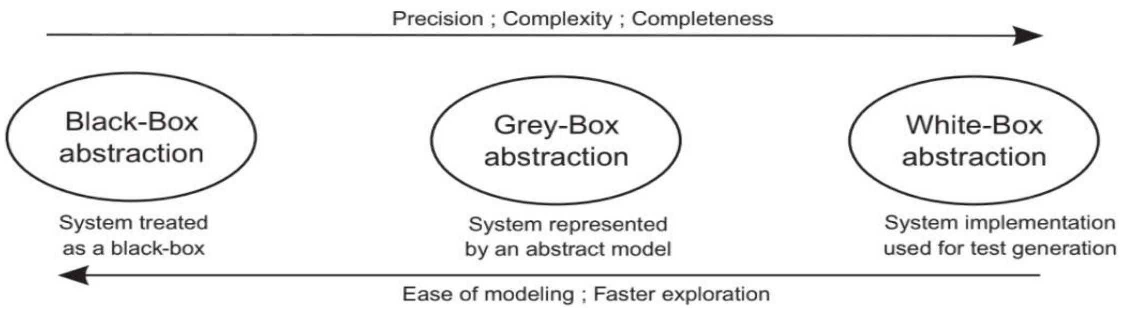
Reliability Constraints:

As embedded software often interacts with the physical environment, it needs to reliably capture the data acquired from the physical world. Usually, this is accomplished via sensors (eg, gyroscope and accelerometers), which interacts with the software via communicating the data from the physical world. For instance, in the fall detection application, the data read via the sensors are sent via wireless sensor network. In general, it is potentially infeasible to get the sensor data accurately. This might be due to the inaccuracy of sensor chips or due to potential packet drops in the network. Therefore, the reliability of different software components may pose a concern for a critical embedded software, such as a fall detector. Besides, the reliability of a component and its cost has nontrivial trade-offs. For instance, a more accurate sensor (or a reliable network) might incur higher cost. Overall, the designer must ensure that the respective software operates with an acceptable level of reliability.

TYPES OF TESTING:

Real-time and embedded systems are used extensively in a wide variety of applications, ranging from automotive and avionics to entertainment and consumer electronics. Depending on the application, the constraints applicable on such systems may range from mission-critical to soft-real time in nature. Additionally, embedded systems often have to interact with the physical environment that may be deterministic or non-deterministic. Such factors imply that embedded systems have to be designed and developed with varying operational requirements and no single testing technique is well suited to all systems. In some scenarios, the system under test (SUT) may be too complex to model and hence, approximate, yet fast sampling-based techniques are suitable. In other scenarios, where the SUT has mission- critical constraints and requires thorough testing, a fine-grained modeling of the system is crucial. In the following paragraphs, we shall categorize and discuss some of the existing works on

testing embedded systems, with a specific focus on literature embedded software testing can be categorized into following three divisions,



UNIT TESTING

Individual developers test at the module level by writing stub code to substitute for the rest of the system hardware and software. At this point in the development cycle, the tests focus on the logical performance of the code. Typically, developers test with some average values, some high or low values, and some out-of-range values (to exercise the code‘s exception processing functionality). Unfortunately, these –black-box‖ derived test cases are seldom adequate to exercise more than a fraction of the total code in the module.

REGRESSION TESTING

It isn‘t enough to pass a test once. Every time the program is modified, it should be retested to assure that the changes didn‘t unintentionally –break‖ some unrelated behavior called regression testing, these tests are usually automated through a test script. For example, if you design a set of 100 input/output (I/O) tests, the regression test script would automatically execute the 100 tests and compare the output against a –gold standard‖ output suite. Every time a change is made to any part of the code, the full regression suite runs on the modified code base to insure that something else wasn‘t broken in the process.

Selecting the Test:

Because no practical set of tests can prove a program correct, the key issue becomes what subset of tests has the highest probability of detecting the most errors, as noted in The Art of Software Testing by Glen Ford Myers. The problem of selecting appropriate test cases is known as *test case design*. Although dozens of strategies exist for generating test cases, they tend to fall into two fundamentally different approaches: functional testing and coverage testing. Functional testing (also known as black-box testing) selects tests that assess how well the implementation meets the requirements specification. Coverage testing (also known as *white-box testing*) selects cases that cause certain portions of the code to be executed. Both kinds of testing are necessary to test rigorously your embedded design. Of the two, coverage testing implies that the code is stable, so it is reserved for testing a completed or nearly completed product. Functional tests, on the other hand, can be written in parallel with the requirements documents. In fact, by starting with the functional tests, one can minimize any duplication of efforts and rewriting of tests. Thus, functional tests can do first. Everyone agrees that functional tests can be written first, but few authors, clearly believe they are most useful during

system integration not unit testing.
The following is a simple process algorithm for integrating functional and coverage testing strategies:
1. Identify which of the functions have NOT been fully covered by the functional tests.
2. Identify which sections of each function have not been executed.
3. Identify which additional coverage tests are required.
4. Run new additional tests.
5. Repeat.

FUNCTIONAL TESTS

Functional testing is often called black-box testing because the test cases for functional tests are devised without reference to the actual code — that is, without looking ―inside the box.‖ An embedded system has inputs and outputs and implements some algorithm between them. Black-box tests are based on what is known about which inputs should be acceptable and how they should relate to the outputs. Black-box tests know nothing about how the algorithm in between is implemented.
Example black-box tests include:
Stress tests: Tests that intentionally overload input channels, memory buffers, disk controllers, memory management systems, and so on.
Boundary value tests: Inputs that represent ―boundaries‖ within a particular range (for example, largest and smallest integers together with –1, 0, +1, for an integer input) and input values that should cause the output to transition across a similar boundary in the output range. Exception tests: Tests that should trigger a failure mode or exception mode.
Error guessing: Tests based on prior experience with testing software or from testing similar programs.
Random tests: Generally, the least productive form of testing but still widely used to evaluate the robustness of user-interface code.
Performance tests: Because performance expectations are part of the product requirement, performance analysis falls within the sphere of functional testing. Because black-box tests depend only on the program requirements and its I/O behavior, they can be developed as soon as the requirements are complete. This allows black-box test cases to be developed in parallel with the rest of the system design. Like all testing, functional tests should be designed to be *destructive*, that is, to prove the program doesn‘t work. this is one of my primary test methodologies. If 40 hours of abuse testing could be logged with no serious or critical defects logged against the product, the product could be released. If a significant defect was found, the clock started over again after the defect was fixed.

COVERAGE TESTS

The weakness of functional testing is that it rarely exercises all the code. Coverage tests attempt to avoid this weakness by (ideally) ensuring that each code statement, decision point, or decision path is exercised at least once. (Coverage testing also can show how much of your data space has been accessed.) Also known as white-box tests or glass-box tests, coverage tests are devised with full knowledge of how the software is implemented, that is, with permission to ―look inside the box.‖ White-box tests are designed with the source code handy. They exploit the programmer‘s knowledge of the program‘s APIs, internal control structures, and exception handling capabilities. Because white-box tests depend on specific implementation decisions, they can‘t be

designed until after the code is written. From an embedded systems point of view, coverage testing is the most important type of testing because the degree to which you can show how much of your code has been exercised is an excellent predictor of the risk of undetected bugs that user will be facing later. Example white-box tests include:

Statement coverage: Test cases selected because they execute every statement in the program at least once.

Decision or branch coverage: Test cases chosen because they cause every branch (both the true and false path) to be executed at least once.

Condition coverage: Test cases chosen to force each condition (term) in a decision to take on all possible logic values. Theoretically, a white-box test can exploit or manipulate whatever it needs to conduct its test. Thus, a white-box test might use the JTAG interface to force a particular memory value as part of a test. More practically, white-box testing might analyze the execution path reported by a logic analyzer.

GRAY-BOX TESTING

Because white-box tests can be intimately connected to the internals of the code, they can be more expensive to maintain than black-box tests. Whereas black-box tests remain valid as long as the requirements and the I/O relationships remain stable, white-box tests might need to be re-engineered every time the code is changed. Thus, the most cost-effective white-box tests generally are those that exploit knowledge of the implementation without being intimately tied to the coding details. Tests that only know a little about the internals are sometimes called gray-box tests. Gray-box tests can be very effective when coupled with error guessing.‖ If programmer know, or at least suspect, where the weak points are in the code, you can design tests that stress those weak points. These tests are gray box because they cover specific portions of the code; they are error guessing because they are chosen based on a guess about what errors are likely. This testing strategy is useful when you‘re integrating new functionality with a stable base of legacy code. Because the code base is already well tested, it makes sense to focus your test efforts in the area where the new code and the old code come together.

PERFORMANCE TESTING

Performance testing and, consequently, performance tuning, are not only important as part of your functional testing but also as important tools for the maintenance and upgrade phase of the embedded life cycle. Performance testing is crucial for embedded system design and, unfortunately, is usually the one type of software characterization test that is most often ignored. Dave Stewart, in ―The Twenty-Five Most Common Mistakes with Real-Time Software Development,‖ considers the failure to measure the execution time of code modules the number one mistake made by embedded system designers. Measuring performance is one of the most crucial tests you need to make on your embedded system. The typical response is that the code is ―good enough‖ because the product works to specification. For products that are incredibly cost sensitive, however, this is an example of engineering at its worst. Why overdesign a system with a faster processor and more and faster RAM and ROM, which adds to the manufacturing costs, lowers the profit margins, and makes the product less competitive, when the solution is as simple as finding and eliminating the hot spots in the code? On any cost-sensitive embedded system design, one of the most dramatic events is the decision to redesign the hardware because you believe you are at the limit of performance gains from software redesign.

Mostly, this is a gut decision rather than a decision made on hard data. On many occasions, intuition fails. Modern software, especially in the presence of an RTOS, is extremely difficult to fully unravel and understand. Just because you can‘t see an obvious way to improve the system throughput by software-tuning does not imply that the next step is a hardware redesign. Performance measurements made with real tools and with sufficient resources can have tremendous payback and prevent large R&D outlays for needless redesigns.

To do Performance test

In performance testing, you are interested in the amount of time that a function takes to execute. Many factors come into play here. In general, it‘s a nondeterministic process, so it must be measured from a statistical perspective. Some factors that can change the execution time each time the function is executed are:

Contents of the instruction and data caches at the time the function is entered
RTOS task loading
Interrupts and other exceptions
Data-processing requirements in the function

Thus, the best hope one can expect is some statistical measure of the minimum, maximum, average, and cumulative execution times for each function that is of interest. For instance, CodeTEST performance analysis test tool, which uses software instrumentation to provide the stimulus for the entry-point and exit point measurements. These tags can be collected via hardware tools or RTOS services.

EMBEDDED APPLICATIONS:

Case study of Smart card

Smart card is one of the most used embedded system today. It is used for credit, debit bank card, e-wallet card, identification card, medical card (for history and diagnosis details) and card for a number of new innovative applications.

Smart Card System Requirements:
Purpose: Enabling authentication and verification of card and card holder by a host. Enabling GUI at host machine to interact with the card holder/user for the required transactions, for example, financial transactions with a bank or credit card transactions.
Input: Received header and messages at IO port Port_IO from host through the antenna.
Internal Signals, Events and Internal Signals, Events and Notifications:
On power up, radiation-powered chargepump supply of the card activated and a signal to start the system boot program at resetTask
Card start requestHeader message to task_ReadPort from resetTask
Host authentication request requestStart message to task_ReadPort from resetTask to enable requests for Port_IO
UserPW verification message (notification) through Port_IO from
host Card application close request requestApplClose message to
Port_IO Outputs: Transmitted headers and messages at Port_IO
through antenna.
Control panel: No control panel is at the card. The control panel and GUIs activate at the host machine (for example, at ATM or credit card reader)

Functions of the system:

- **The card inserts at a host machine.**
- **The radiations from the host activate a charge pump at the card.**
- **The charge pump powers the SoC circuit consisting of card processor, memory, timer, interrupt handler and IO port, Port_IO.**
- **On power up, system reset signals resetTask to start.**
- **The resetTask sends the messages requestHeader and requestStart for waiting task task_ReadPort**
- **task_ReadPort sends requests for host identification and reads through the Port_IO the host-identification message and request for card identification**
- **task_PW sends through Port_IO the requested card identification after system receives the host identity through Port_IO.**
- **task_Appl then runs required API. The requestApplClose message closes the application.**
- **The card can now be withdrawn**
- **All transactions between cardholder/user now takes place through GUIs using at the host control panel (screen or touch screen or LCD display panel).**

Design metrics:

Power Source and Dissipation: Radiation powered contact less.

Code size: optimum. card system memory needs should not exceed 64 kB memory.

Limited use of data types; multidimensional arrays, long 64-bit integer and floating points and very limited use of the error handlers, exceptions, signals, serialization, debugging and profiling.

File system(s): Three-layered file system for data.

File management: There is either a fixed length file management or a variable file length management with each file with a predefined offset.

Microcontroller hardware: Generates distinct coded physical addresses for the program and data logical addresses. Protected once writable memory space.

Validity: System is embedded with expiry date, after which the card authorization through the hosts disables.

Extendibility: The system expiry date is extendable by transactions and authorization of master control unit (for example, bank servee).

Performance: Less than 1s for transferring control from the card to host machine.

Process Deadlines: None.

User Interfaces: At host machine, graphic at LCD or touch screen display on LCD and commands for card holder (card user) transactions.

Test and validation conditions: Tested on different host machine versions for fail proof card-host communication

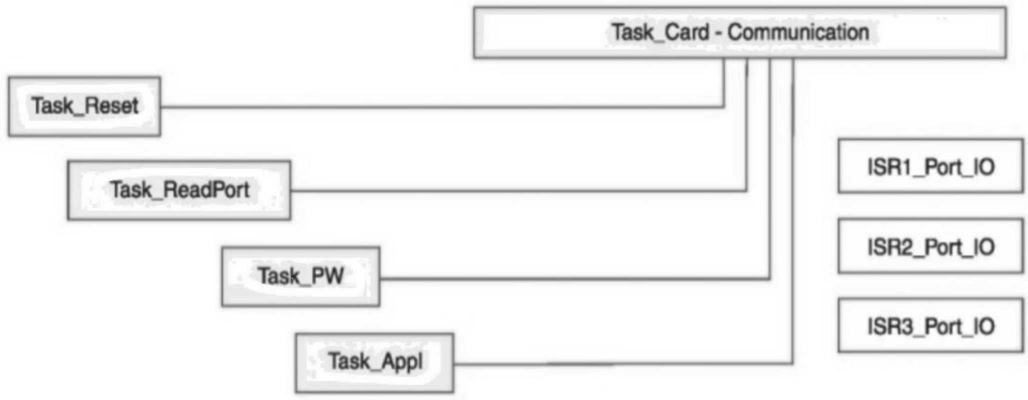


Fig:5.2 :Task_CardCommunication is an abstract class from which extended to class (es) derive to read port and authenticate.

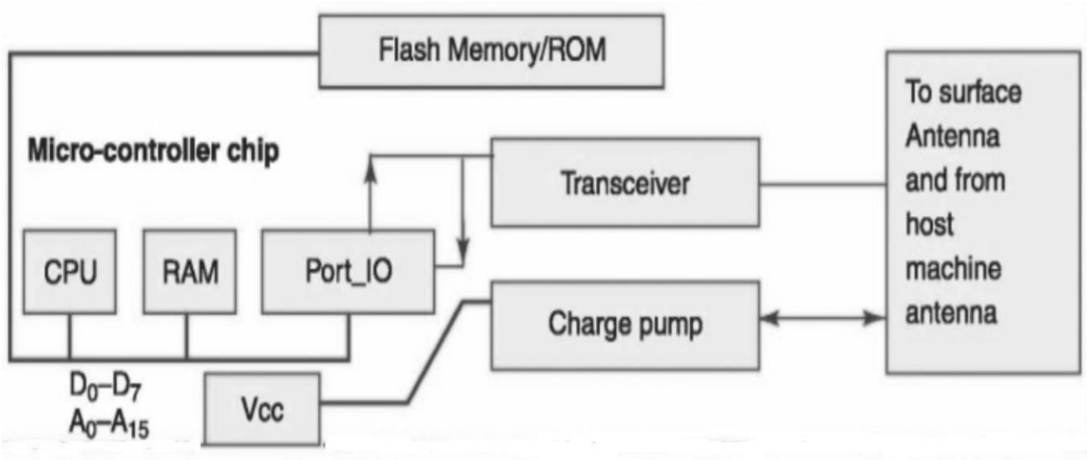


Fig: 5.3: Hardware Architecture.

Hardware Architecture: A plastic card in ISO standard dimensions, 85.60 mm x 53.98 0.80 mm. It is an embedded SoC (System-OnChip). [ISO standards - ISO7816 (1 to 4) for host-machine contact based card and ISO14443 (Part A or B) for the contactless cards.]. Microcontroller MC68HC11D0 or PIC16C84 or a smart card processor Philips Smart XA or an ASIP Processor. Needs 8 kB+ internal RAM and 32 kB EPROM and 2/3 wire protected memory. CPU special features, for example, a security lock. CPU locks certain section of memory - protect 1 kB or more data from modification and access by any external source or instruction outside that memory. Other way of protecting - CPU access through the physical addresses, which are different from logical address used in the program.

Standard ROM 8 kB for usual or 64 kB when using advanced cryptographic features. Full or part of ROM bus activates take place after a security check only.

ROM Contains:

i. **Fabrication key and Personalization key (after insertion of this key, RTOS and application use only the logical addresses)**
ii. **RTOS codes**
iii. **Application codes**
iv. **Utilization lock**

EEPROM or Flash scalable – only needed part unlocks when storing P.I.N., unlocking P.I.N., access condition, card-user data, post activation application run generated non- volatile data, invalidation lock to invalidate card after the expiry date or server instruction.
RAM – run time temporary
variables Chip-supply system using
charge pump I/O system

Software Architecture:
Needs cryptographic software, needs special features in its operating system over and above the MS DOS or UNIX system features.

Table:5.1 Tasks and their priority, action and 6. Tasks and their priority, action and IPCs

Task function	priority	Action	IPC 10 pending	IPC posted	Host input	String Output
Reset Task	1	Initiates system timer ticks, creates tasks, sends initial messages and suspends itself	none	SigReset, MsgQStart		request-Header; requestStart
task_ ReadPort	2	Wait for resetTask suspension, sends the queue messages and receives the messages. Starts the application and seeks closure permission for closing the application	SigReset, MsgQStart, MsgQPW MsgQAppl, MsgQAppl- Close	SemPW		request-password, request-Appl, requestApplClose
task_PW	3	Sends request for password on verification of host when SemPW = 1	SemPW	MsgQPW	request-Password	
task_Appl	8	when SemPW = 1, runs the application program	SemAppl	MsgQAppl		

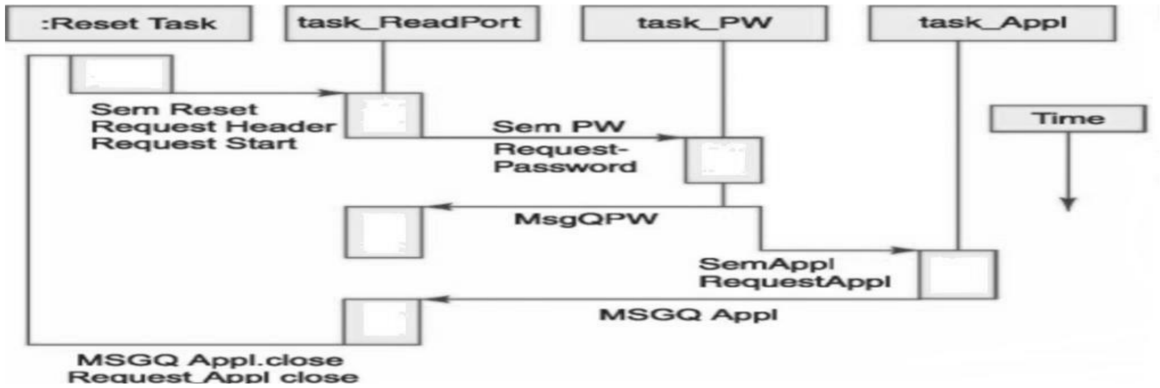


Fig:5.4: Tasks and their synchronization model.

INTERFACING STEPPER MOTOR.

Parallel Port Interfacing Parallel Port Interfacing with Stepper Motor:

Four outputs may be used as driver to four coils of a stepper motor. One step angle when its four coils are given the currents in a specific sequence and that sequence is altered. Assume that currents at an instance equal + i, 0, 0, 0 in four coils X, X‘, Y, Y‘. The motor rotates by one step, when the currents change to 0, + i, 0, 0.

Sequences at the intervals of T are changed as follows: 1000, 0100, 0010, 0001, 1000, 0100, …. [The bits in the nibble (set of 4 bits) rotate by right shift.] Here 1 corresponds to + i. The motor rotates n step angles in interval = (n.T).

Reverse motion Sequences at the intervals of T

Sequences are changed to rotate the motor in reverse direction 0001, 00010, 0100, 1000, 0001, 0010, ….[The bits in the nibble (set of 4 bits) rotate by left shift.]

Port Interfacing Port Interfacing

4 coils connect to parallel-port 4 outputs.

Stepper motor driver : It‘s a processing element

Driver is given two outputs from the port─ clock pulses and rotating direction bit r. For example, if r = 1, the motor rotates clockwise and if 0 then anti-clockwise. The motor rotates as long as clock pulses are given at the output pin.

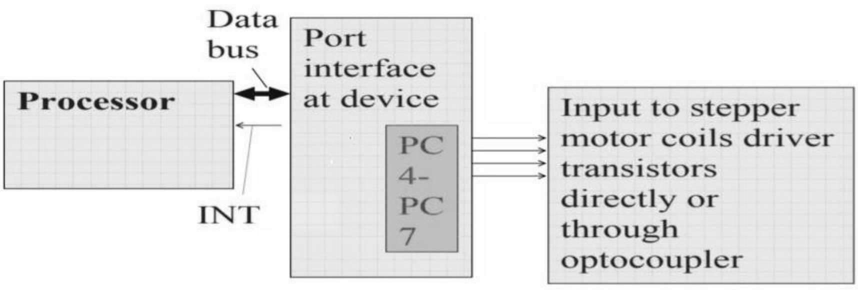


Fig:5.5: Four -bit parallel output port C connected to a stepper motor.

RFID

RFID is not a new technology and has passed through many decades of use in military, airline, library, security, healthcare, sports, animal farms and other areas. Industries use RFID for various applications such as personal/vehicle access control, departmental store security, equipment tracking, baggage, fast food establishments,

logistics, etc. The enhancement in RFID technology has brought advantages that are related to resource optimization, increased efficiency within business processes, and enhanced customer care, overall improvements in business operations and healthcare. Our research is part of a big project; its aim is to produce a model for mobile technology implementation of hospital patients' movement process. RFID stands for Radio Frequency Identification and is a term that describes a system of identification . RFID is based on storing and remotely retrieving information or data as it consists of RFID tag, RFID reader and backend Database . RFID tags store unique identification information of objects and communicate the tags so as to allow remote retrieval of their ID. RFID technology depends on the communication between the RFID tags and RFID readers. The range of the reader is dependent upon its operational frequency. Usually the readers have their own software running on their ROM and also, communicate with other software to manipulate these unique identified tags . Basically, the application which manipulates tag deduction information for the end user, communicates with the RFID reader to get the tag information through antennas. Many researchers have addressed issues that are related to RFID reliability and capability . RFID is continuing to become popular because it increases efficiency and provides better service to stakeholders . RFID technology has been realized as a performance differentiator for a variety of commercial applications, but its capability is yet to be fully utilized

RFID Evalution

RFID technology has passed through many phases over the last few decades (see figure 1). The technology has been used in tracking delivery of goods, in courier services and in baggage handling. Other applications includes automatic toll payments, departmental access control in large buildings, personal and vehicle control in a particular area, security of items which shouldn't leave the area, equipment tracking in engineering firms, hospital filing systems, etc.

Working

Most RFID systems consist of tags that are attached to the objects to be identified. Each tag has its own –read-only‖ or –rewrite‖ internal memory depending on the type and application. Typical configuration of this memory is to store product information, such as an object's unique ID manufactured date, etc. The RFID reader generates magnetic fields that enable the RFID system to locate objects (via the tags) that are within its range. The high-frequency electromagnetic energy and query signal generated by the reader triggers the tags to reply to the query; the query frequency could be up to 50 times per second. As a result communication between the main components of the system i.e. tags and reader is established. As a result large quantities of data are generated. Supply chain industries control this problem by using filters that are routed to the backend information systems. In other words, in order to control this problem, software such as Savant is used. This software acts as a buffer between the Information Technology and RFID reader.

Several protocols manage the communication process between the reader and tag. These protocols (ISO 15693 and ISO 18000-3 for HF or the ISO 18000-6, and EPC for UHF) begin the identification process when the reader is switched on. These protocol works on selected frequency bands (e.g. 860 – 915 MHz for UHF or 13.56 MHz for HF). If the reader is on and the tag arrives in the reader fields, then it automatically wakes-up and decodes the signal and replies to the reader by modulating the reader's field. All the tags in the reader range may reply at the same time, in this case the reader must

detect signal collision (indication of multiple tags). Signal collision is resolved by applying anti-collision algorithm which enables the reader to sort tags and select/handle each tag based on the frequency range (between 50 tags to 200 tags) and the protocol used. In this connection the reader can perform certain operations on the tags such as reading the tag's identifier number and writing data into a tag.

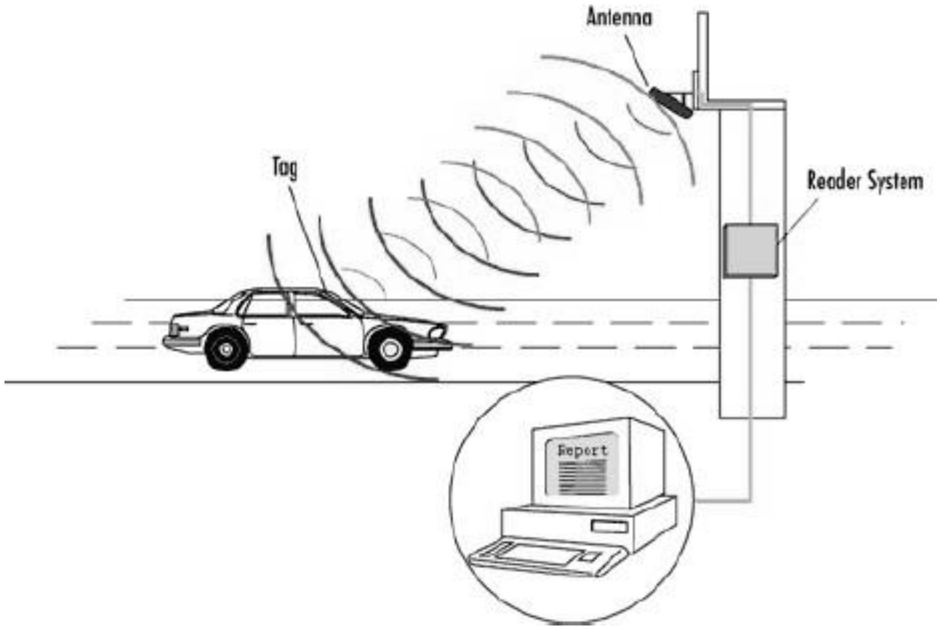


Fig:5.6: A typical RFID System

The reader performs these operations one by one on each tag. A typical RFID system work cycle can be seen in figure 2.

Components of an RFID System

The RFID system consists of various components which are intergrated in a manner defined in the above section. This allows the RFID system to deduct the objects (tag) and perform vaious operations on it. The intergration of RFID components enables the implementation of an RFID solution. The RFID system consists of following five components (as shown in Figure 3):

- **Tag (attached with an object, unique identification).**
- **Antenna (tag detector, creates magnetic field).**
- **Reader (receiver of tag information, manipulator).**
- **Communication infrastructure (enable reader/RFID to work through IT infrastructure).**
- **Application software (user database/application/ interface).**

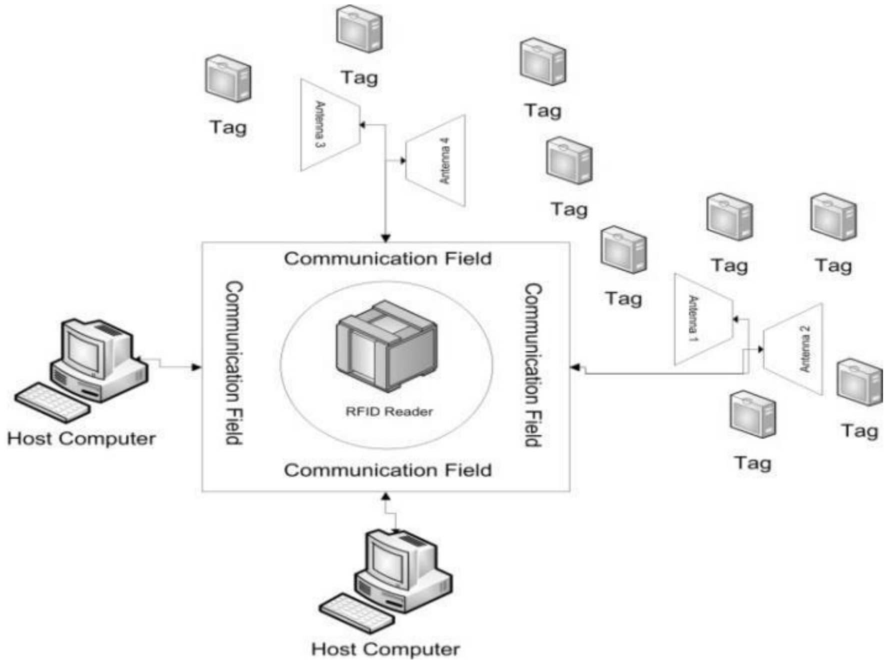


Fig:5.7: Components of an RFID System

Antennas

RFID antennas collect data and are used as a medium for tag reading. It consists of the following:

Fig:5.8: RFID antennas types

(1) Patch antennas, (2) Gate antennas, (3) Linear polarized, (4) Circular polarized (5) Di-pole or multi-pole antennas, (6) Stick antennas, (7) Beam-forming or phased- array element antennas, (8) Adaptive antennas, and (9) Omni directional antennas.

Applications:

General RFID applications according to its capabilities

Object identification can be given through various ways such as barcode, biometric and RFID. RFID has two basic categories (short & long range). The short range applications need tags to be near reader, it is useful in various condition such as when a patient is required to come near the door/reader and only one person can get access (access control). The long range applications may not need tags that are closer to reader. Similar scenarios are successfully been used on various items in the warehouse (logistics). The most common applications are:

- **Healthcare Applications**
- **Security & Control Applications**
- **Patrolling Log Applications**
- **Baggage Applications**
- **Toll Road Applications**

TAG READER:

Tags contain microchips that store the unique identification (ID) of each object. The ID is a serial number stored in the RFID memory. The chip is made up of intergrated circuit and embedded in a silicon chip. RFID memory chip can be permanent or changeable depending on the read/write characteristics. Read-only and rewrite circuits are different as read-only tag contain fixed data and can not be changed without re-program electonically . On the other hand, re-write tags can be programmed through the reader at any time without any limit. RFID tags can be different sizes and shapes depending on the application and the environment at which it will be used. A variety of materials are intergrated on these tags. For example, in the case of the credit cards, small plastic peaces are stuck on various objects, and the labels. Labels are also emmbeded in a variety of objects such as documents, cloths, maufacturing mateirals etc . Figure 4 demonstrates the different sizes and shapes of the RFID tags.

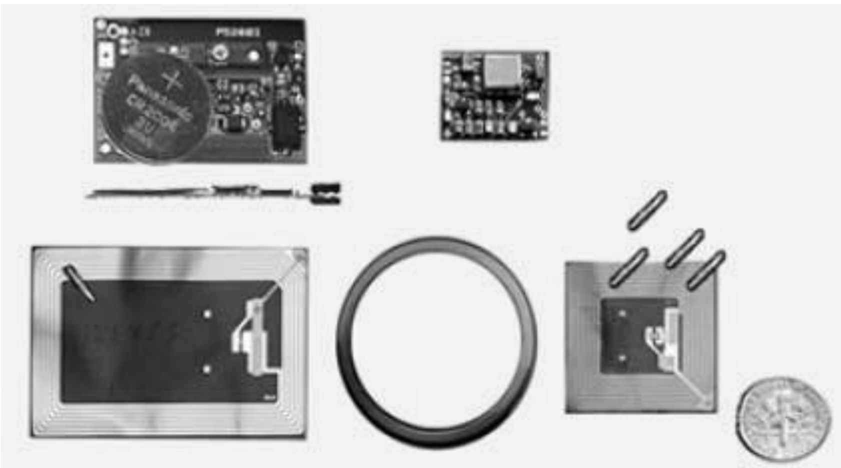


Fig:5.9: Varity of RFID tags (various shape & sizes)

RFID tags can also be classified by their capabilities such as read and write data. The above figure shows the five classifications of the RFID tags. There are three types of tags: the passive, semi-active and active. Semi-active tags have a combination of active and passive tags characteristics. So, mainly two types of tags (active and passive) are being used by industry and most of the RFID system. The essential characteristics of RFID tags are their function to the RFID system. This is based on their range, frequency, memory, security, type of data and other characteristics. These characteristics are core for RFID performance and differ in usefulness/support to the RFID system operations. While considering these characteristics, figure 6 compares the active and passive tags.

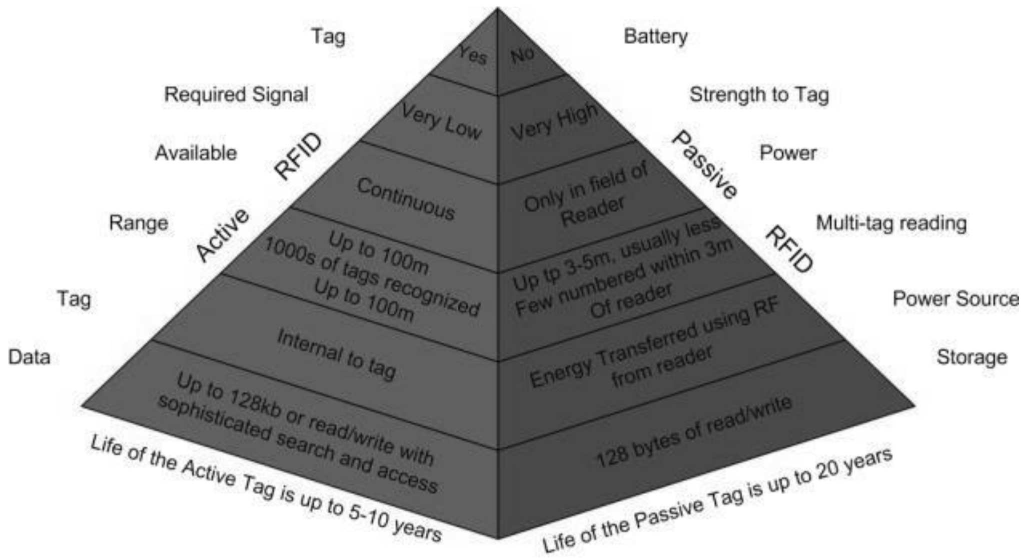


Fig:5.10: RFID active and passive tags comparison

Tag Frequencies

The range of the RFID tags depends on their frequency. This frequency determines the resistance to interference and other performance attributes [12]. The use/selection of RFID tag depends on the application; different frequencies are used on different RFID tags [10]. EPCglobal and International Standards Organization (ISO) are the major organizations working to develop international standards for RFID technologies in the UHF band. These two organizations are still evolving and are not fully compatible with each other [14]. In order to avoid the use of different radio frequencies standards, most of the international communities are obligated to comply with the International Telecommunication Union (ITU) standards. The following are the commonly used frequencies:

- **Microwave works on 2.45 GHz, it has good reader rate even faster than UHF tags. Although at this frequency the reading rate results are not the same on wet surfaces and near metals, the frequency produce better results in applications such as vehicle tracking (in and out with barriers), with approximately 1 meter of tags read range.**

- **Ultra High Frequency works within a range of 860-930 MHz, it can identify large numbers of tags at one time with quick multiple read rate at a given time. So, it has a considerable good reading speed. It has the same limitation as Microwave when is applied on wet surface and near metal. However, it is faster than high frequency data transfer with a reading range of 3 meters.**

- **High Frequency works on 13.56MHz and has less than one meter reading range but is inexpensive and useful for access control, items identifications on sales points etc as it can**
implanted inside thin things such as paper.

19

- **Low Frequency works on 125 kHz, it has approximately half a meter reading range and mostly used for short reading range applications such as shops, manufacturing factories, inventory control through in and out counts, access control through showing a card to the reader. These low frequency tags are mostly not affected when applied on wet and near metal surfaces.**

RFID Reader

RFID reader works as a central place for the RFID system. It reads tags data through the RFID antennas at a certain frequency. Basically, the reader is an electronic apparatus which produce and accept a radio signals. The antennas contains an attached reader, the reader translates the tags radio signals through antenna, depending on the tags capacity. The readers consist of a build-in anti-collision schemes and a single reader can operate on multiple frequencies. As a result, these readers are expected to collect or write data onto tag (in case) and pass to computer systems. For this purpose readers can be connected using RS-232, RS- 485, USB cable as a wired options (called serial readers) and connect to the computer system. Also can use WiFi as wireless options which also known as network readers. Readers are electronic devices which can be used as standalone or be integrated with other devices and the following components/hardware into it.

Power for running reader, (2) Communication interface, (3) Microprocessor, (4) Channels, (5) Controller, (6) Receiver, (7) Transmitter, (8) Memory.

Tag Standards

Readers use near and far fields of methodology to communicate to the tag through its antennas. If a tag wants to respond to the reader then the tag will need to receive energy and communicate with a reader. For example, passive tags use either one of the two following methods

Near Fields: Near field uses method similar to transformer, and employs inductive coupling of the tag to the magnetic field circulating around the reader antenna

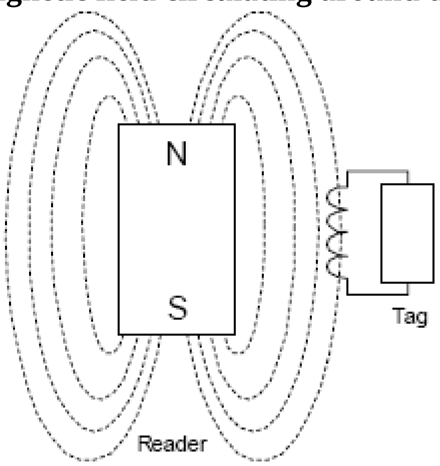


Fig:5.11: Magnetic field around the reader antenna

Far Field: Far field uses method similar to radar, backscatter reflection by coupling with the electric field. The distinction between the RFID systems with far fields to the near fields is that the near fields use LF (lower frequency) and HF (higher frequency) bands. While RFID systems with far fields usually use longer read range UHF and microwave.

HAND HELD DEVICE

Despite of rapid growth and development in every aspect the companies or government sector"s in 1980"s, delivery-confirmation and billing systems have historically relied upon manual systems for collecting and managing data. Sorting, loading and issuing bills were systematic, but required manual data entry at several stages. The personnel at the site carry large volumes of books, files and receipts that contain all the details. The

whole process was very much time consuming and confusing also for the new workers. They have to work hard to understand all those details. The cumbersome process was eliminated with the advent of hand held machine. Thus in order to decrease man power and eliminate manual blunders and save time, hand held computer with processing power, keypad, display and a mini built-in printer was excellent solution. Hawkins and his team developed the GriDPad in 1998, the first successful hand held. It was large and awkward, but it worked The churning ideas for employing this technology in very small, portable, general purpose computers led to development of latest palmtops.

Hand held computer (HHC) is an electro-mechanical device with limited memory required for the specific function therefore it is quite cheaper than the PDA, Palmtop and laptop. Its affordable price makes it easy to users to have it for its versatile usage. HHC are more rugged and are designed to present the user with a simple one-question one-answer environment rather than the normal windows office automation screens. The application software is so simple that any non-technical common person can use it.

General Advantages Of Hand held computers:

Hand held computer (HHC) is a portable compact-computing device. It has user interface to interact with user, it has 16-bit processor and a printer built-in. This can be carried to any field application site. This device works by downloading the application from a desktop computer and carried to the fields to use it. It loads master data from desktop, collects the data, prints the bills at customer site and uploads the data collected from customer onto the desktop computer from hand held computer. As shown in figure1. It is integrated with 36 keys keypad for input, 4X20 LCD matrix screen for display and 24-column dot matrix printer for printing. With all these features and flexible down loadable applications the embedded device HHC can be used in many applications like, on spot bill issue for electricity billings at the consumer site, stock verification in grocery, ticket issue for buses or railways etc.

Few general advantages are listed down.

- **It is portable.**
- **It is scalable.**
- **It provides mobile data collection.**
- **It saves time.**
- **It provides spot billings.**
- **Elimination of manual blunders.**
- **Decreases man power.**
- **Easy and simple to operate.**
- **Secure.**
- **Customizable.**
- **Maintainable.**

DEVELOPMENT OF HHC

The primary task is to program the micro controller XAG3 to work as a hand held computer. The detailed study of internal architecture of XAG3 and hardware interfaces is essential to write application programs .The micro controller XAG3, member of Philips 80C51 family is used having special features like high speed, low power consumption and high memory capacity and support for multitasking serial communications. The code for the application program is written in Embedded C language. But we cannot get the required output with simple application code, for this the drivers of HHC have to be developed. Thus drivers for LCD, printer, RTC, ADC and serial communication have to be developed. These drivers are like subroutines or libraries, which are called by the application programs . These device driver programs have to be executed in tasking compiler. Tasking EDE compiler generates a hex code, which has to be is burnt into XAG3. In this project we have developed device drivers for HHC and application down loadable program. HHC can be used in many applications like, on spot bill issuing, maintaining records and mobile data collections. Normally software products developed are compiled and executed on the same system but in embedded system we have cross-compilers that are, the program is developed and compiled in one system and is executed in other system. Tasking EDE compiler is used as a development tool for writing device drivers as well as application programs. The code is written in embedded C .We are making Hex file of the program which has to be downloaded into target device.

Table : 5.2 XA-G3 Microprocessor

Hardware requirements

Processor (Operating speed 30 MHz)	16bit XA Processor (XA-G3)
Flash Memory:	56K x 8 Bit
Random Access Memory	1024Kx8 Bit.
Keypad	6x6alpha numeric keypad with 36 keys
Real Time Clock (RTC)	PCF8563 (32.768 kHz)
Battery	NiMH battery with 6V and 1.6 AH.
Interface	RJ-16 (2Nos).
Printer	24 column DOT matrix printer
Liquid Crystal Display (LCD)	4/20 Characters with backlight

Software requirements

Operating system	Win9x, 2000, Me, XP.
Compiler	Tasking EDE Compiler.
Language	Assembly, C, Embedded C.

The XA-G39 is a member of Philips 80C51 XA (eXtended Architecture) family of high performance 16-bit single-chip micro controllers. The XA-G39 contains 32 Kbytes of flash program memory, and provides three general purpose timers/counters, a watchdog timer, dual UARTs, and four general purposes I/O ports with programmable output configurations. A default serial loader program in the Boot ROM allows In-System Programming (ISP) of the flash memory without the need for a loader in the flash code. User programs may erase and reprogram the flash memory at will through the use of standard routines contained in the Boot ROM (In-application programming).

Keypad

The predominant interface between human and computers is the keypad or keyboard. The keypad application program must guard against the following possibilities like more than one key pressed, key pressed and held or rapid key pressed and released. The universal key characteristic is the ability to bounce. The key contacts vibrate open and closed for a number of milliseconds when the key is hit and often when is released. The key may be debounced by using proper time delays in software.

RTC (Real Time Clock)

The PCF8563 is a CMOS real-time clock/calendar optimized for low power consumption. A programmable clock output, interrupt output and voltage-low detectors are also provided. All address and data are transferred serially via a two-line bi-directional I2C-bus. Maximum bus speed is 400 Kbits/s. The built-in word address register is incremented automatically after each written or read data byte.

Flash memory

The flash is 5-volt-only in the system flash programmable and erasable read only memory (PEROM). Its 2Mb of memory is organized as 262,144 bytes. Manufactured with Atmel‟s advanced non volatile CMOS Technology. Reprogramming the flash is performed on a sector basis; 256 Bytes of data are loaded into the device and then simultaneously programmed. In circuit programming and erasing allows its use when loading an updated version of your program‟s object code into memory through a serial interface.

EPROM

The EPROM is a high speed, low power consumption electrically erasable and programmable read only memory organized as 131,072 S8 bits. It requires only one supply in the range of 5V+-5% in normal read mode. This provides an electrical chip erase function.

RAM

1MB of external RAM (data) is used. RD (read) and WR (write) signals are needed during external RAM accesses. The low mode enables write option whereas high mode enables read option.

Printer

The printer used here is 24-line dot matrix printer. It consists of a motor, main solenoid (home position) and seven printer solenoids. The motor is used for the movement of the printer head, which must be enabled initially. Main solenoid must also be enabled whenever the printing starts. Initially the printer head must be brought to home position .A printer head has 7 print wires (solenoids) arranged in a vertical column and electromagnetic mechanism able to shoot the wires.

Power supply

NiMH battery is used as the power supply for the HHC. When the battery is fully charged it contains 6v. We will get the low battery indication at 5.3 Volts. When it reaches 4.8 V it will stop working and we have to charge it completely.

Software platform to program micro controller

When designing software for a smaller embedded system with the 8051, it is very common to develop the entire product using assembly code. With many projects, this is a feasible approach since the amount of code that must be generated is typically less than 8 kilobytes and is relatively simple in nature. The trouble with projects done with assembly code can is that they can be difficult to read and maintain, especially if they are not well commented. Additionally, the amount of code reusable from a typical assembly language project is usually very low. Use of a higher-level language like C can directly address these issues. A program written in C is easier to read than an assembly program. Since a C program possesses greater structure, it is easier to understand and maintain.

Because of its modularity, a C program can better lend itself to reuse of code from project to project. The division of code into functions will force better structure of the software and lead to functions that can be taken from one project and used in another, thus reducing overall development time. A high order language such as C allows a developer to write code, which resembles a human's thought, process more closely than does the equivalent assembly code. The developer can focus more time on designing the algorithms of the system rather than having to concentrate on their individual implementation. This will greatly reduce development time and lower debugging time since the code is more understandable. By using a language like C, the programmer does not have to be intimately familiar with the architecture of the processor. This means that someone new to a given processor can also be able to develop a project and make it run, since the internals and organization of the target processor do not have to be learned. Additionally, code developed in C will be more portable to other systems than code developed in assembly. Many target processors have C compilers available, which support ANSI C.

Embedded C

The C programming language was designed for computers, though, and not embedded systems. It does not support direct access to registers, nor does it allow for the reading and setting of single bits, two very important requirements for 8051 software. In addition, most software developers are accustomed to writing programs that will be executed by an operating system, which provides system calls, the program may use to access the hardware. However, much code for the 8051 is written for direct use on the processor, without an operating system. To support this, the Keil/Tasking Compiler has

added several extensions to the C language to replace what might have normally been implemented in a system call, such as the connecting of interrupt handlers.
Associated Compiler Experts (ACE) announced that C language extensions have been officially adopted and approved as part of the industry specification by the ISO technical committee. The resulting efforts can be found in technical Report 18037[16], extensions for programming language C to support embedded processors. The Embedded C technical report specifies a range of extensions to the ISO/IEC 9899:1999 C language specification, also known as ISO C99 usually known as Embedded C.

Tasking EDE compiler

The TASKING EDE (cross compiler) differs from a native program development; a native program development is often used to develop applications for systems where the target and the host system are one. Therefore, it is possible to run a compiled application directly from the interactive development environment. In an embedded environment this is no longer true. Of course you can still compile a module and make it compile error free. However, to run an application, a simulator or target hardware is required. TASKING offers a number of simulators and target hardware debuggers. The generic name of the debugger product is Cross View Pro.
EDE is an integrated software development platform that compiles a powerful editor, project manager with a make facility. EDE supports all TASKING tools for all targets and is at the same time designed to be open and extensible (i.e. integrate with third party tools). EDE integration helps to develop your embedded application by providing the user friendly features.

Downloading code to target processor

The final step in development environment is to down load the Hex code to the target device, which comprises of development processor and target processor. The development processor is nothing but the processor on which we write and debug our programs. This can be an ordinary PC loaded with a tasking compiler (TC) on which we can develop a C-program and convert it into an output format, which the micro controller can understand. As shown in the figure 2 the various C files are compiled/assembled and linked to generate an Obj/Hex file, which is downloaded to the target processor using a device programmer. The Cross-compiler is used here assembles or compiles code on development processor (PC) for use on target processor (here on HHC). The code runs in a real time environment.

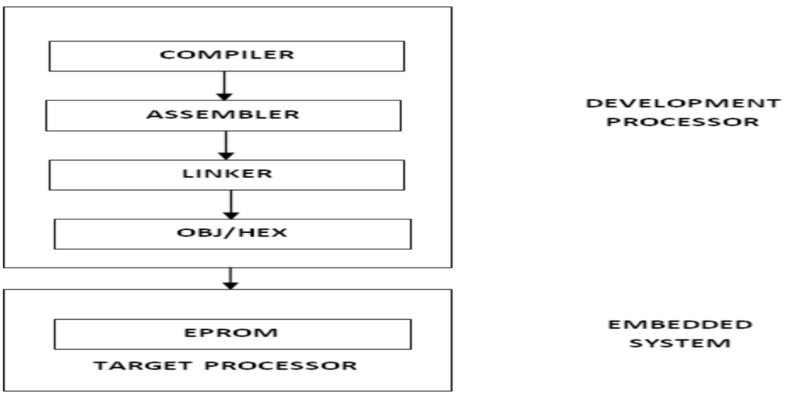


Fig:5.12: Downloading code to target processor

HHC used for water billing

The real time application developed in our project is spot water billing at the doorstep using HHC. A simple receipt at customer door step is shown in the figure

```
      XXX OF CEDEC LIMITED,
            WATER BILL
            MALAYSIA

 ERO: 1     SEC: 1     GRP: A1
 DT:  13/ 05/ 2010
SCNO:  V 0032     A.C: 1095
NAME: ABUBAKAR BIN ABULHASAN
ADDR:  H.No: 46/06
          Taman Pekaka
            Malaysia
CAT: 1         PH: 1        MF: 1
LOAD: 1000

       READING    MONTH   STS
PRES:   360        APR, 10      01
PREV:   180        FEB, 10      01
UNITS:  180           AVG: 11
ENERGY CHARGES :   10.00
CUST CHARGES    :   10.00
ADDL. CHARGES   :   0.00
LOSS/GAIN       :  +0.00
TOTAL AMOUNT    :   380.00
ARREARS         :   0.00
NET AMOUNT      :  380.00
DUE DATE:  25 / 05 / 2010
------------------------------------
TOTAL AMT        :  380.RM
------------------------------------
```

WASHING MACHINE

The design uses the PIC18F series microcontroller. All the control functionalities of the system are built around this. Upgradeability is the unique feature of this system. Control card hardware and software allows the manufacturer to add or remove the features as per customer requirement and model. Thus once 27 the whole system is designed it is very economic in large quantity production. Single-phase motor is considered for the design. Front
panel consists of a keypad and LCD display. Keypad provides automatic and manual wash options to the user. LCD display is convenient to convey machine information to user. One more design possibility is to use brushless DC motors or three phase induction motor. These types of motors are very efficient but requires complex control algorithm. To implement such a complex and real time algorithm dedicated controller and software is required which a master controller controls. Even though cost is important criteria modern washing machines are designed with BLDC motors owing to efficiency and energy conservation

Design Specifications
This include both hardware and software specifications. 1. The system should provide fully automatic mode, semi-automatic mode and manual mode. Modes should be selectable by a keypad.
Under fully automatic mode user intervention requirement should be zero. Once the system is started in this mode it should perform its work independently and after the completion of work it should notify the user about the completion of work. This mode instantaneously should sense cloth quality and requirement of water, water temperature, detergent, load, wash cycle time and perform operation accordingly.

In semi-automatic mode also user requirement should be nil. But user has to choose any one of the semi-automatic mode in which washing conditions are predefined. Once the predefined mode is started the system should perform its job and after completion it should inform the user.

In manual mode continuous intervention of user is required. User has to specify which operation he wants to do and has to provide related information to the control system. For example, if user wants to wash only, he has to choose ‗wash' option in manual mode. Then the system should ask the user to enter the wash time, amount of water and the load. After these data are entered, the user should start the machine. When the specified operation is completed system should inform the user.

When the lid is open system should not work. If door is accidentally opened in between wash operation, then the system should stop working in minimum possible time (<10s)>

The system should provide all basic features of a washing machine like washing, rinsing, spinning, drying, cold wash, hot wash etc.

The system should provide easy options for upgradeability of new features. The hardware and the software should be compatible to both machines, which have fewer features, or more features. Removal of any feature should not affect the working of any other features or overall working of the system.

The system should work on single phase AC from 190VAC to 250VAC. The system should protect itself from power supply voltage variations.

In the event of power failure, the washing machine should automatically start its cycle from the point of interruption when power is resumed.

Hardware Design

Heart of this system is PIC18F452. Most of the peripheral features have been utilized to implement the design. Controlling the motor is very crucial part of the design. The PWM feature of the microcontroller controls motor speed. PWM output is fed to driver circuit and then to motor. To rotate the motor in two different directions ‗forward' and ‗reverse' direction control blocks are used. Motor speed sensor is interfaced to microcontroller. Microcontroller reads the speed of the motor and appropriately controls the speed of the motor in different phases of washing using PWM output. Door sensor, pressure sensor, keypad are also interfaced to microcontroller. Serial port in connected to GSM module.

EEPROM and RTC are interfaced to MSSP module of controller. In circuit serial programming facility is provided for quick and easy programming and debugging.

Schematic Design

A detailed schematic with pin connection of PIC microcontroller is provided in the below figure

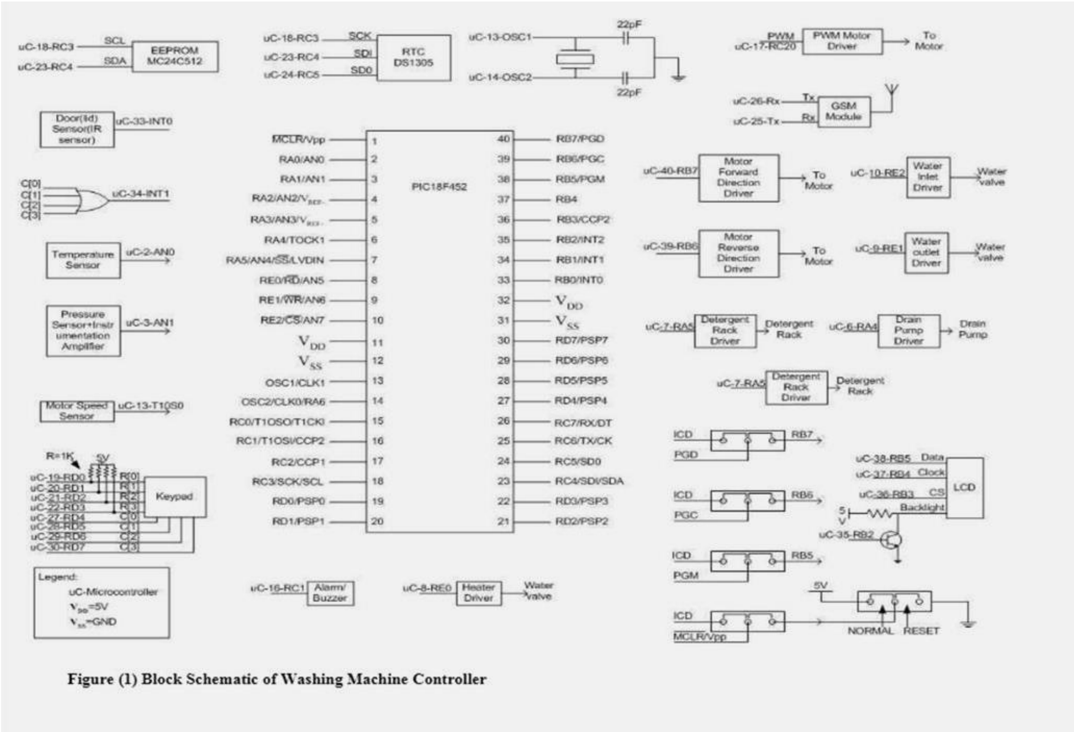


Fig:5.13: Block diagram of washing machine controller

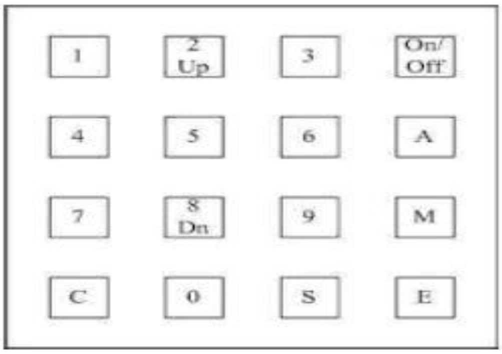
Washing machine default parameters and user settable parameters are stored in external EEPROM. Internal EEPROM of the PIC is used to store status of the washing machine. The status is regularly logged to the internal EEPROM. In the event of power failure or whenever program resets, status flags are read from the internal EEPROM and thus status of the machine is determined and operation is continued from the point of interruption. Accessing of internal EEPROM is faster compared to external EEPROM. PIC18F452 provide 256 bytes of internal EEPROM that is not sufficient for storing parameters of machine. For this purpose external EEPROM is used. Depending on the mode flag and status flag conditions corresponding machine parameters are read from the external EEPROM and temporarily stored in RAM and operations are performed.

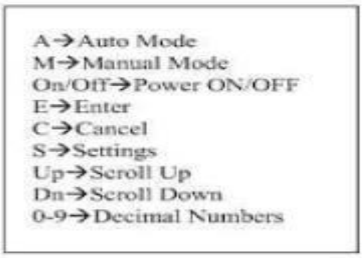
RTC DS1305 is interfaced to SPI port of the microcontroller. This RTC is used as timing reference for all timing calculation of machine. Whenever a particular mode starts RTC is initialized to zero and there onwards RTC is read and compared with the set timings; with the battery backup provision actual RTC can also be implemented. Since PIC allows either I2C or SPI mode at a time, whenever we need to access EEPROM or RTC, MSSP port of the PIC has to be configured to respective protocol.

In Circuit Serial Programming (ICSP) is accomplished using pins PGM, PGC, PGC and active low MCLR. These pins are also used to RB6 and RB7. To satisfy both conditions jumpers are provided. When programming of IC is required jumper settings ‗1‘ has to be used. After programming, jumpers have to be replaced to setting ‗2‘. This allows the use of RB5 to RB7 and brings the controller from programming mode to normal working mode. Thus ICSP helps speedy programming and debugging of software.

Some of the sensor outputs are fed to instrumentation amplifier to bring the output level to 0V to 5V range. Door (lid) sensor is connected to external interrupt 0. High priority is assigned to this interrupt. Thus opening of the door causes triggering of INT0 and INT0 ISR immediately stops the machine and informs the user. Analog input channels AN2, AN3 and AN4 can be used to upgrade the system with additional sensors.

Keypad is connected to PORTD. Pull-up resistors are connected to RD0 to RD3 to enable keypad press detection. ORing RD4 to RD7 achieves this and output is given to external interrupt 1(INT1). When any of the keys is pressed ORed output becomes high and INT1 triggers. INT1 ISR does a keypad scan and appropriately performs the





operation.

Motor direction driver circuit determines the direction of the motor. Motor forward direction drives the current in forward direction and motor rotates forward. Motor reverse direction driver does the opposite of it. Software has to take care that both are not activated at a time. Speed of the motor is controlled using PWM driver. Speed of the motor and hence the drum varies in different phases of washing cycles. Speed profile of the washing machine drum is illustrated in Figure. Before changing the direction of the motor, PWM output must be made zero. PWM output is varied as per the calculated speed to maintain constant speed during wash cycles. in different phases of washing cycles

Port pins RE1 and RE2 are fed to water valve drivers, which control the flow of water, while RE0 is connected to heater relay. Dedicated LCD with 3-wire interface is used. 3 wires consist of data line, clock and Chip Select (CS). Backlight control is also provided. GSM module is interfaced using serial port. Standard AT command set is used to communicate with GSM module. It is assumed that when sensor outputs reach microcontroller inputs it has maximum swing of 0V to 5V. It is also assumed that PWM driver circuit controls gearbox and clutch mechanisms. Power supply circuits and regulators. in different phases of washing cycles Suitably designed SMPS and AC regulators take care of power supply requirement of each block.

TEXT / REFERENCE BOOKS

1. KVKK Prasad, “Embedded / Real Time Systems”, Dreamtech Press, 2005.
2. David Simon, “An Embedded Software Primer”, Pearson Education Asia, First Indian Reprint 2000.
3. Raj Kamal, ‘Embedded system-Architecture, Programming, Design’, Tata McGraw Hill, 2011.
4. Arnold Berger, “Embedded system design”, CMP books, 1st Edition, 2005
5. Wayne Wolf, “Computers as components”, Morgan Kaufmann publishers, 2nd Edition, 2008.
6. Tammy Noergaard, “Embedded Systems Architecture”, Elsevier, 2006