FUTURE VISION BIE

One Stop for All Study Materials

& Lab Programs

Future Vision

By K B Hemanth Raj

Scan the QR Code to Visit the Web Page

Or

Visit : https://hemanthrajhemu.github.io

Gain Access to All Study Materials according to VTU,
CSE – Computer Science Engineering,
ISE – Information Science Engineering,
ECE - Electronics and Communication Engineering
& MORE...

Join Telegram to get Instant Updates: https://bit.ly/VTU_TELEGRAM

Contact: MAIL: futurevisionbie@gmail.com

INSTAGRAM: www.instagram.com/hemanthraj_hemu/

INSTAGRAM: www.instagram.com/futurevisionbie/

WHATSAPP SHARE: https://bit.ly/FVBIESHARE

INTRODUCTION TO EMBEDDED SYSTEMS

Shibu K V

Technical Architect
Mobility & Embedded Systems Practice
Infosys Technologies Ltd.,
Trivandrum Unit, Kerala

McGraw Hill Education (India) Private Limited
NEW DELHI

McGraw Hill Education Offices
New Delhi New York St Louis San Francisco Auckland Bogotá Caracas
Kuala Lumpur Lisbon London Madrid Mexico City Milan Montreal
San Juan Santiago Singapore Sydney Tokyo Toronto

https://hemanthrajhemu.github.io

Contents

Preface *xiii*
Acknowledgements *xvii*

Part 1
Embedded System: Understanding the Basic Concepts

1. Introduction to Embedded Systems **3**
1.1 What is an Embedded System? *4*
1.2 Embedded Systems vs. General Computing Systems *4*
1.3 History of Embedded Systems *5*
1.4 Classification of Embedded Systems *6*
1.5 Major Application Areas of Embedded Systems *7*
1.6 Purpose of Embedded Systems *8*
1.7 'Smart' Running Shoes from Adidas—The Innovative Bonding of Lifestyle with Embedded Technology *11*
Summary *13*
Keywords *13*
Objective Questions *14*
Review Questions *14*

2. The Typical Embedded System **15**
2.1 Core of the Embedded System *17*
2.2 Memory *28*
2.3 Sensors and Actuators *35*
2.4 Communication Interface *45*
2.5 Embedded Firmware *59*
2.6 Other System Components *60*
2.7 PCB and Passive Components *64*
Summary *64*
Keywords *65*

https://hemanthrajhemu.github.io

1 Introduction to Embedded Systems

LEARNING OBJECTIVES

- ✓ *Learn what an Embedded System is*
- ✓ *Learn the difference between Embedded Systems and General Computing Systems*
- ✓ *Know the history of Embedded Systems*
- ✓ *Learn the classification of Embedded Systems based on performance, complexity and the era in which they evolved*
- ✓ *Know the domains and areas of applications of Embedded Systems*
- ✓ *Understand the different purposes of Embedded Systems*
- ✓ *Analysis of a real life example on the bonding of embedded technology with human life*

Our day-to-day life is becoming more and more dependent on "embedded systems" and digital techniques. Embedded technologies are bonding into our daily activities even without our knowledge. Do you know the fact that the refrigerator, washing machine, microwave oven, air conditioner, television, DVD players, and music systems that we use in our home are built around an embedded system? You may be traveling by a 'Honda' or a 'Toyota' or a 'Ford' vehicle, but have you ever thought of the genius players working behind the special features and security systems offered by the vehicle to you? It is nothing but an intelligent embedded system. In your vehicle itself the presence of specialised embedded systems vary from intelligent head lamp controllers, engine controllers and ignition control systems to complex air bag control systems to protect you in case of a severe accident. People experience the power of embedded systems and enjoy the features and comfort provided by them. Most of us are totally unaware or ignorant of the intelligent embedded systems giving us so much comfort and security. Embedded systems are like reliable servants–they don't like to reveal their identity and neither they complain about their workloads to their owners or bosses. They are always sitting in a hidden place and are dedicated to their assigned task till their last breath. This book gives you an overview of embedded systems, the various steps involved in their design and development and the major domains where they are deployed.

https://hemanthrajhemu.github.io

1.1 WHAT IS AN EMBEDDED SYSTEM?

An embedded system is an electronic/electro-mechanical system designed to perform a specific function and is a combination of both hardware and firmware (software).

Every embedded system is unique, and the hardware as well as the firmware is highly specialised to the application domain. Embedded systems are becoming an inevitable part of any product or equipment in all fields including household appliances, telecommunications, medical equipment, industrial control, consumer products, etc.

1.2 EMBEDDED SYSTEMS vs. GENERAL COMPUTING SYSTEMS

The computing revolution began with the general purpose computing requirements. Later it was realised that the general computing requirements are not sufficient for the embedded computing requirements. The embedded computing requirements demand 'something special' in terms of response to stimuli, meeting the computational deadlines, power efficiency, limited memory availability, etc. Let's take the case of your personal computer, which may be either a desktop PC or a laptop PC or a palmtop PC. It is built around a general purpose processor like an Intel® Centrino or a Duo/Quad† core or an AMD Turion™ processor and is designed to support a set of multiple peripherals like multiple USB 2.0 ports, Wi-Fi, ethernet, video port, IEEE1394, SD/CF/MMC external interfaces, Bluetooth, etc and with additional interfaces like a CD read/writer, on-board Hard Disk Drive (HDD), gigabytes of RAM, etc. You can load any supported operating system (like Windows® XP/Vista/7, or Red Hat Linux/Ubuntu Linux, UNIX etc) into the hard disk of your PC. You can write or purchase a multitude of applications for your PC and can use your PC for running a large number of applications (like printing your dear's photo using a printer device connected to the PC and printer software, creating a document using Microsoft® Office Word tool, etc.) Now let us think about the DVD player you use for playing DVD movies. Is it possible for you to change the operating system of your DVD? Is it possible for you to write an application and download it to your DVD player for executing? Is it possible for you to add a printer software to your DVD player and connect a printer to your DVD player to take a printout? Is it possible for you to change the functioning of your DVD player to a television by changing the embedded software? The answers to all these questions are 'NO'. Can you see any general purpose interface like Bluetooth or Wi-Fi on your DVD player? Of course 'NO'. The only interface you can find out on the DVD player is the interface for connecting the DVD player with the display screen and one for controlling the DVD player through a remote (May be an IR or any other specific wireless interface). Indeed your DVD player is an embedded system designed specifically for decoding digital video and generating a video signal as output to your TV or any other display screen which supports the display interface supported by the DVD Player. Let us summarise our findings from the comparison of embedded system and general purpose computing system with the help of a table:

General Purpose Computing System	Embedded System
A system which is a combination of a generic hardware and a General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OS for executing a specific set of applications
Contains a General Purpose Operating System (GPOS)	May or may not contain an operating system for functioning

†The illustration given here is based on the processor details available till Dec 2008. Since processor technology is undergoing rapid changes, the processor names mentioned here may not be relevant in future.

https://hemanthrajhemu.github.io

Applications are alterable (programmable) by the user (It is possible for the end user to re-install the operating system, and also add or remove user applications)	The firmware of the embedded system is pre-programmed and it is non-alterable by the end-user (There may be exceptions for systems supporting OS kernel image flashing through special hardware settings)
Performance is the key deciding factor in the selection of the system. Always 'Faster is Better'	Application specific requirements (like performance, power requirements, memory usage etc.) are the key deciding factors
Less/not at all tailored towards reduced operating power requirements, options for different levels of power management	Highly tailored to take advantage of the power saving modes supported by the hardware and the operating system
Response requirements are not time critical	For certain category of embedded systems like mission critical systems, the response time requirement is highly critical
Need not be deterministic in execution behaviour	Execution behaviour is deterministic for certain types of embedded systems like 'Hard Real Time' systems

However, the demarcation between desktop systems and embedded systems in certain areas of embedded applications are shrinking in certain contexts. Smart phones are typical examples of this. Nowadays smart phones are available with RAM up to 256 MB and users can extend most of their desktop applications to the smart phones and it waives the clause "Embedded systems are designed for a specific application" from the characteristics of the embedded system for the mobile embedded device category. However, smart phones come with a built-in operating system and it is not modifiable by the end user. It makes the clause: "The firmware of the embedded system is unalterable by the end user", still a valid clause in the mobile embedded device category.

1.3 HISTORY OF EMBEDDED SYSTEMS

Embedded systems were in existence even before the IT revolution. In the olden days embedded systems were built around the old vacuum tube and transistor technologies and the embedded algorithm was developed in low level languages. Advances in semiconductor and nano-technology and IT revolution gave way to the development of miniature embedded systems. The first recognised modern embedded system is the Apollo Guidance Computer (AGC) developed by the MIT Instrumentation Laboratory for the lunar expedition. They ran the inertial guidance systems of both the Command Module (CM) and the Lunar Excursion Module (LEM). The Command Module was designed to encircle the moon while the Lunar Module and its crew were designed to go down to the moon surface and land there safely. The Lunar Module featured in total 18 engines. There were 16 reaction control thrusters, a descent engine and an ascent engine. The descent engine was 'designed to' provide thrust to the lunar module out of the lunar orbit and land it safely on the moon. MIT's original design was based on 4K words of fixed memory (Read Only Memory) and 256 words of erasable memory (Random Access Memory). By June 1963, the figures reached 10K of fixed and 1K of erasable memory. The final configuration was 36K words of fixed memory and 2K words of erasable memory. The clock frequency of the first microchip proto model used in AGC was 1.024 MHz and it was derived from a 2.048 MHz crystal clock. The computing unit of AGC consisted of approximately 11 instructions and 16 bit word logic. Around 5000 ICs (3-input NOR gates, RTL logic) supplied by Fairchild Semiconductor were used in this design. The user interface unit of AGC is known as DSKY (display/keyboard). DSKY looked like a calculator type keypad with an array of numerals. It was used for inputting the commands to the module numerically.

https://hemanthrajhemu.github.io

The first mass-produced embedded system was the guidance computer for the Minuteman-I missile in 1961. It was the '*Autonetics* ***D-17***' guidance computer, built using discrete transistor logic and a hard-disk for main memory. The first integrated circuit was produced in September 1958 but computers using them didn't begin to appear until 1963. Some of their early uses were in embedded systems, notably used by NASA for the Apollo Guidance Computer and by the US military in the Minuteman-II intercontinental ballistic missile.

1.4 CLASSIFICATION OF EMBEDDED SYSTEMS

It is possible to have a multitude of classifications for embedded systems, based on different criteria. Some of the criteria used in the classification of embedded systems are:

1. Based on generation
2. Complexity and performance requirements
3. Based on deterministic behaviour
4. Based on triggering.

The classification based on deterministic system behaviour is applicable for 'Real Time' systems. The application/task execution behaviour for an embedded system can be either deterministic or non-deterministic. Based on the execution behaviour, Real Time embedded systems are classified into *Hard* and *Soft*. We will discuss about hard and soft real time systems in a later chapter. Embedded Systems which are 'Reactive' in nature (Like process control systems in industrial control applications) can be classified based on the trigger. Reactive systems can be either *event triggered* or *time triggered.*

1.4.1 Classification Based on Generation

This classification is based on the order in which the embedded processing systems evolved from the first version to where they are today. As per this criterion, embedded systems can be classified into:

1.4.1.1 First Generation The early embedded systems were built around 8bit microprocessors like 8085 and Z80, and 4bit microcontrollers. Simple in hardware circuits with firmware developed in Assembly code. Digital telephone keypads, stepper motor control units etc. are examples of this.

1.4.1.2 Second Generation These are embedded systems built around 16bit microprocessors and 8 or 16 bit microcontrollers, following the first generation embedded systems. The instruction set for the second generation processors/controllers were much more complex and powerful than the first generation processors/controllers. Some of the second generation embedded systems contained embedded operating systems for their operation. Data Acquisition Systems, SCADA systems, etc. are examples of second generation embedded systems.

1.4.1.3 Third Generation With advances in processor technology, embedded system developers started making use of powerful 32bit processors and 16bit microcontrollers for their design. A new concept of application and domain specific processors/controllers like Digital Signal Processors (DSP) and Application Specific Integrated Circuits (ASICs) came into the picture. The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved. The processor market was flooded with different types of processors from different vendors. Processors like Intel Pentium, Motorola 68K, etc. gained attention in high performance embedded requirements. Dedicated embedded real time and general purpose operating systems entered into the embedded market. Embedded systems spread its ground to areas like robotics, media, industrial process control, networking, etc.

https://hemanthrajhemu.github.io

1.4.1.4 Fourth Generation The advent of System on Chips (SoC), reconfigurable processors and multicore processors are bringing high performance, tight integration and miniaturisation into the embedded device market. The SoC technique implements a total system on a chip by integrating different functionalities with a processor core on an integrated circuit. We will discuss about SoCs in a later chapter. The fourth generation embedded systems are making use of high performance real time embedded operating systems for their functioning. Smart phone devices, mobile internet devices (MIDs), etc. are examples of fourth generation embedded systems.

1.4.1.5 What Next? The processor and embedded market is highly dynamic and demanding. So 'what will be the next smart move in the next embedded generation?' Let's wait and see.

1.4.2 Classification Based on Complexity and Performance

This classification is based on the complexity and system performance requirements. According to this classification, embedded systems can be grouped into:

1.4.2.1 Small-Scale Embedded Systems Embedded systems which are simple in application needs and where the performance requirements are not time critical fall under this category. An electronic toy is a typical example of a small-scale embedded system. Small-scale embedded systems are usually built around low performance and low cost 8 or 16 bit microprocessors/microcontrollers. A small-scale embedded system may or may not contain an operating system for its functioning.

1.4.2.2 Medium-Scale Embedded Systems Embedded systems which are slightly complex in hardware and firmware (software) requirements fall under this category. Medium-scale embedded systems are usually built around medium performance, low cost 16 or 32 bit microprocessors/microcontrollers or digital signal processors. They usually contain an embedded operating system (either general purpose or real time operating system) for functioning.

1.4.2.3 Large-Scale Embedded Systems/Complex Systems Embedded systems which involve highly complex hardware and firmware requirements fall under this category. They are employed in mission critical applications demanding high performance. Such systems are commonly built around high performance 32 or 64 bit RISC processors/controllers or Reconfigurable System on Chip (RSoC) or multi-core processors and programmable logic devices. They may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor of the system. Decoding/encoding of media, cryptographic function implementation, etc. are examples for processing requirements which can be implemented using a co-processor/hardware accelerator. Complex embedded systems usually contain a high performance Real Time Operating System (RTOS) for task scheduling, prioritization and management.

1.5 MAJOR APPLICATION AREAS OF EMBEDDED SYSTEMS

We are living in a world where embedded systems play a vital role in our day-to-day life, starting from home to the computer industry, where most of the people find their job for a livelihood. Embedded technology has acquired a new dimension from its first generation model, the Apollo guidance computer, to the latest radio navigation system combined with in-car entertainment technology and the microprocessor based "Smart" running shoes launched by Adidas in April 2005. The application areas and the products in the embedded domain are countless. A few of the important domains and products are listed below:

https://hemanthrajhemu.github.io

1. *Consumer electronics:* Camcorders, cameras, etc.
2. *Household appliances:* Television, DVD players, washing machine, fridge, microwave oven, etc.
3. *Home automation and security systems:* Air conditioners, sprinklers, intruder detection alarms, closed circuit television cameras, fire alarms, etc.
4. *Automotive industry:* Anti-lock breaking systems (ABS), engine control, ignition systems, automatic navigation systems, etc.
5. *Telecom:* Cellular telephones, telephone switches, handset multimedia applications, etc.
6. *Computer peripherals:* Printers, scanners, fax machines, etc.
7. *Computer networking systems:* Network routers, switches, hubs, firewalls, etc.
8. *Healthcare:* Different kinds of scanners, EEG, ECG machines etc.
9. *Measurement & Instrumentation:* Digital multi meters, digital CROs, logic analyzers PLC systems, etc.
10. *Banking & Retail:* Automatic teller machines (ATM) and currency counters, point of sales (POS)
11. *Card Readers:* Barcode, smart card readers, hand held devices, etc.

1.6 PURPOSE OF EMBEDDED SYSTEMS

As mentioned in the previous section, embedded systems are used in various domains like consumer electronics, home automation, telecommunications, automotive industry, healthcare, control & instrumentation, retail and banking applications, etc. Within the domain itself, according to the application usage context, they may have different functionalities. Each embedded system is designed to serve the purpose of any one or a combination of the following tasks:

1. Data collection/Storage/Representation
2. Data communication
3. Data (signal) processing
4. Monitoring
5. Control
6. Application specific user interface

1.6.1 Data Collection/Storage/Representation

Embedded systems designed for the purpose of data collection performs acquisition of data from the external world. Data collection is usually done for storage, analysis, manipulation and transmission. The term "data" refers all kinds of information, viz. text, voice, image, video, electrical signals and any other measurable quantities. Data can be either analog (continuous) or digital (discrete). Embedded systems with analog data capturing techniques collect data directly in the form of analog signals whereas embedded systems with digital data collection mechanism converts the analog signal to corresponding digital signal using analog to digital (A/D) converters and then collects the binary equivalent of the analog data. If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

The collected data may be stored directly in the system or may be transmitted to some other systems or it may be processed by the system or it may be deleted instantly after giving a meaningful representation. These actions are purely dependent on the purpose for which the embedded system is designed. Embedded systems designed for pure measurement applications without storage, used in control and

https://hemanthrajhemu.github.io

instrumentation domain, collects data and gives a meaningful representation of the collected data by means of graphical representation or quantity value and deletes the collected data when new data arrives at the data collection terminal. Analog and digital CROs without storage memory are typical examples of this. Any measuring equipment used in the medical domain for monitoring without storage functionality also comes under this category.

Some embedded systems store the collected data for processing and analysis. Such systems incorporate a built-in/plug-in storage memory for storing the captured data. Some of them give the user a meaningful representation of the collected data by visual (graphical/quantitative) or audible means using display units [Liquid Crystal Display (LCD), Light Emitting Diode (LED), etc.] buzzers, alarms, etc. Examples are: measuring instruments with storage memory and monitoring instruments with storage memory used in medical applications. Certain embedded systems store the data and will not give a representation of the same to the user, whereas the data is used for internal processing.

A digital camera is a typical example of an embedded system with data collection/storage/representation of data. Images are captured and the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.

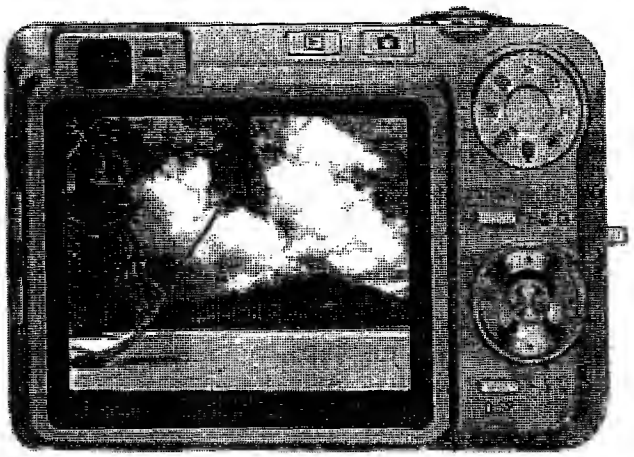


Fig. 1.1 **A digital camera for image capturing/storage/display**
(Photo courtesy of Casio-Model EXILIM ex-Z850 (www.casio.com))

1.6.2 Data Communication

Embedded data communication systems are deployed in applications ranging from complex satellite communication systems to simple home networking systems. As mentioned earlier in this chapter, the data collected by an embedded terminal may require transferring of the same to some other system located remotely. The transmission is achieved either by a wire-line medium or by a wireless medium. Wire-line medium was the most common choice in all olden days embedded systems. As technology is changing, wireless medium is becoming the de-facto standard for data communication in embedded systems. A wireless medium offers cheaper connectivity solutions and make the communication link free from the hassle of wire bundles. Data can either be transmitted by analog means or by digital means. Modern industry trends are settling towards digital communication.

The data collecting embedded terminal itself can incorporate data communication units like wireless

Fig. 1.2 **A wireless network router for data communication**
(Photo courtesy of Linksys (www.linksys.com). A division of CISCO system)

https://hemanthrajhemu.github.io

modules (Bluetooth, ZigBee, Wi-Fi, EDGE, GPRS, etc.) or wire-line modules (RS-232C, USB, TCP/IP, PS2, etc.). Certain embedded systems act as a dedicated transmission unit between the sending and receiving terminals, offering sophisticated functionalities like data packetizing, encrypting and decrypting. Network hubs, routers, switches, etc. are typical examples of dedicated data transmission embedded systems. They act as mediators in data communication and provide various features like data security, monitoring etc.

1.6.3 Data (Signal) Processing

As mentioned earlier, the data (voice, image, video, electrical signals and other measurable quantities) collected by embedded systems may be used for various kinds of data processing. Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, synthesis, audio video codec, transmission applications, etc.

A digital hearing aid is a typical example of an embedded system employing data processing. Digital hearing aid improves the hearing capacity of hearing impaired persons.

Fig. 1.3 **A digital hearing aid employing signal processing technique**
(Siemens TRIANO 3 Digital hearing aid; Siemens Audiology Copyright© 2005)

1.6.4 Monitoring

Embedded systems falling under this category are specifically designed for monitoring purpose. Almost all embedded products coming under the medical domain are with monitoring functions only. They are used for determining the state of some variables using input sensors. They cannot impose control over variables. A very good example is the electro cardiogram (ECG) machine for monitoring the heartbeat of a patient. The machine is intended to do the monitoring of the heartbeat. It cannot impose control over the heartbeat. The sensors used in ECG are the different electrodes connected to the patient's body.

Some other examples of embedded systems with monitoring function are measuring instruments like digital CRO, digital multimeters, logic analyzers, etc. used in Control & Instrumentation applications. They are used for knowing (monitoring) the status of some variables like current, voltage, etc. They cannot control the variables in turn.

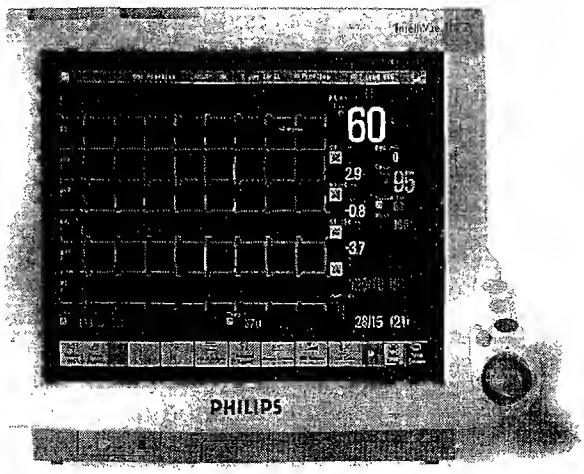


Fig. 1.4 **A patient monitoring system for monitoring heartbeat**
(Photo courtesy of Philips Medical Systems (www.medical.philips.com/))

1.6.5 Control

Embedded systems with control functionalities impose control over some variables according to the changes in input variables. A system with control functionality contains both sensors and actuators. Sensors are connected to the input port for capturing

https://hemanthrajhemu.github.io

the changes in environmental variable or measuring variable. The actuators connected to the output port are controlled according to the changes in input variable to put an impact on the controlling variable to bring the controlled variable to the specified range.

Air conditioner system used in our home to control the room temperature to a specified limit is a typical example for embedded system for control purpose. An airconditioner contains a room temperature-sensing element (sensor) which may be a thermistor and a handheld unit for setting up (feeding) the desired temperature. The handheld unit may be connected to the central embedded unit residing inside the airconditioner through a wireless link or through a wired link. The air compressor unit acts as the actuator. The compressor is controlled according to the current room temperature and the desired temperature set by the end user.

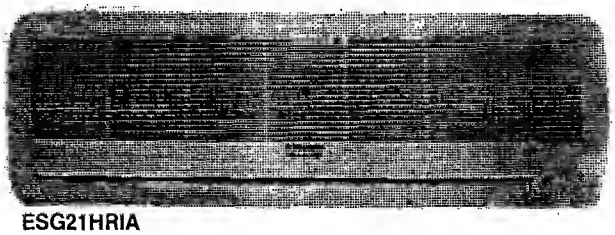


Fig. 1.5 **"An Airconditioner for controlling room temperature. Embedded System with Control functionality"**
(Photo courtesy of Electrolux Corporation (www.electrolux.com/au))

Here the input variable is the current room temperature and the controlled variable is also the room temperature. The controlling variable is cool air flow by the compressor unit. If the controlled variable and input variable are not at the same value, the controlling variable tries to equalise them through taking actions on the cool air flow.

1.6.6 Application Specific User Interface

These are embedded systems with application-specific user interfaces like buttons, switches, keypad, lights, bells, display units, etc. Mobile phone is an example for this. In mobile phone the user interface is provided through the keypad, graphic LCD module, system speaker, vibration alert, etc.

Fig. 1.6 **An embedded system with an application-specific user interface**
(Photo courtesy of Nokia Mobile Handsets (www.nokia.com))

1.7 'SMART' RUNNING SHOES FROM ADIDAS—THE INNOVATIVE BONDING OF LIFESTYLE WITH EMBEDDED TECHNOLOGY

After three years of extensive research work, Adidas launched the "Smart" running shoes in the market in April 2005. The term "Smart Shoe" may sound gimmicky. But adaptive cushioning provided by the shoe makes sense, and the design engineering behind the shoes is very impressive. The shoe constantly adapts its shock-absorbing characteristics to customize its value to the individual runner, depending on the running style, pace, body weight, and running surface. The shoe uses a magnetic sensing system to measure cushioning level, which is adjusted via a digital signal processing unit that controls a motor-driven cable system.

https://hemanthrajhemu.github.io

A hall effect sensor is positioned at the top of the "cushioning element", and the magnet is placed at the bottom of the element. As the cushioning compresses on each impact, the sensor measures the distance from top to bottom of mid-sole (accurate to 0.1 mm). About 1000 readings per second are taken and relayed to the shoe's microprocessor. The Microprocessor (MPU) is positioned under the arch of the shoe. It runs an algorithm that compares the compression messages received from the sensor to a preset range of proper cushioning levels, so it understands if the shoe is too soft or too firm. Then the MPU sends a command to a micro motor, housed in the mid-foot. The micro motor turns a lead screw to lengthen or shorten a cable secured to the walls of a plastic-cushioning element. When the cable is shortened, the cushioning element is pulled taut and compresses very little. A longer cable allows for a more cushioned feel. A replaceable 3-V battery powers the motor and lasts for about 100 hours of running.

The Portland, Ore-based Adidas Innovation Team that developed the shoe was led by Christian DiBenedetto. It also included electromechanical engineer Mark Oleson, as well as a footwear developer and two industrial designers. Oleson explains that the team chose a magnetic sensor because it could measure the amount of compression in addition to the time it took to reach full compression. Gathering sensor data, he says, meant little without building a comparative "running context". So one of the first steps in developing the MPU algorithms was building this database. Runners wore test shoes that gathered information about various compression levels during a run. Then the runners were interviewed to learn their thoughts about the different cushion levels. "When the two matched up, that helped validate our sensor," says Oleson.

Adaptations in the cushioning element account for the change of running surface and pace of the runner, and they're made gradually over an average of four running steps. The goal is for the runner not to feel any sudden changes. Adaptations are made during the "swing" phase rather than the "stance" phase of the stride (i.e. when the foot is off the ground). If the shoe's owner prefers a more cushioned or a firmer "ride," adjustments can be made via "+" and "–" buttons that also activate the intelligent functions of the shoe.

LED indicators confirm when the electronics are turned on (The lights do not remain on when the shoes are in use). If the shoes aren't turned on, they operate like old-fashioned "manual" running shoes. The shoes turn off if their owner is either inactive or at a walking pace for 10 minutes.

Source Electronic Design
www.electronicdesign.com/Articles/Index.cfm?AD=1&ArticleID=10113
Re-printed with permission

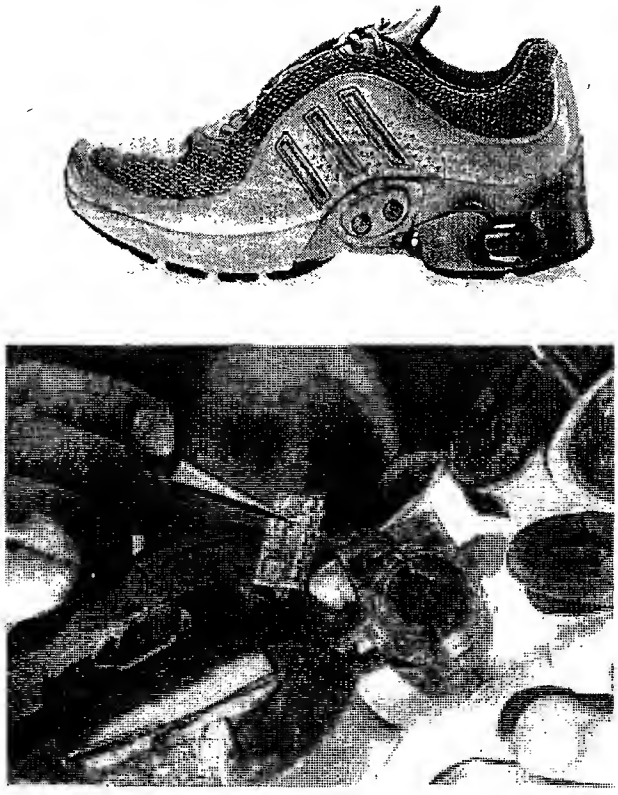


Fig. 1.7 **Electronics-enabled "Smart" running shoes from Adidas**
(Photo courtesy of Adidas – Salomon AG (www.adidas.com))

https://hemanthrajhemu.github.io

Summary

- ✓ An embedded system is an Electronic/Electro-mechanical system designed to perform a specific function and is a combination of both hardware and firmware (Software).
- ✓ A general purpose computing system is a combination of generic hardware and general purpose operating system for executing a variety of applications, whereas an embedded system is a combination of special purpose hardware and embedded OS/firmware for executing a specific set of applications.
- ✓ Apollo Guidance Computer (AGC) is the first recognised modern embedded system and 'Autonetics D-17', the guidance computer for the Minuteman-I missile, was the first mass-produced embedded system.
- ✓ Based on the complexity and performance requirements, embedded systems are classified into small-scale, medium-scale and large-scale/complex.
- ✓ The presence of embedded systems vary from simple electronic toys to complex flight and missile control systems.
- ✓ Embedded systems are designed to serve the purpose of any one or a combination of data collection/storage/representation, data communication, data (signal) processing, monitoring, control or application specific user interface.

Keywords

Embedded system : An electronic/electro-mechanical system which is designed to perform a specific function and is a combination of both hardware and firmware

Microprocessor : A silicon chip representing a Central Processing Unit (CPU)

Microcontroller : A highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays and integrated peripherals

DSP : Digital Signal Processor is a powerful special purpose 8/16/32 bit microprocessor designed specifically to meet the computational demands and power constraints

ASIC : Application Specific Integrated Circuit is a microchip designed to perform a specific or unique application

Sensor : A transducer device that converts energy from one form to another for any measurement or control purpose

Actuator : A form of transducer device (mechanical or electrical) which converts signals to corresponding physical action (motion)

LED : Light Emitting Diode. An output device producing visual indication in the form of light in accordance with current flow

Buzzer : A piezo-electric device for generating audio indication. It contains a piezo-electric diaphragm which produces audible sound in response to the voltage applied to it

Operating system : A piece of software designed to manage and allocate system resources and execute other pieces of software

Electro Cardiogram (ECG) : An embedded device for heartbeat monitoring

SCADA : Supervisory Control and Data Acquisition System. A data acquisition system used in industrial control applications

RAM : Random Access memory. Volatile memory

ADC : Analog to Digital Converter. An integrated circuit which converts analog signals to digital form

https://hemanthrajhemu.github.io

Bluetooth : A low cost, low power, short range wireless technology for data and voice communication
Wi-Fi : Wireless Fidelity is the popular wireless communication technique for networked communication of devices

Objective Questions

1. Embedded systems are
   (a) General purpose (b) Special purpose
2. Embedded system is
   (a) An electronic system (b) A pure mechanical system
   (c) An electro-mechanical system (d) (a) or (c)
3. Which of the following is not true about embedded systems?
   (a) Built around specialised hardware (b) Always contain an operating system
   (c) Execution behaviour may be deterministic (d) All of these
   (e) None of these
4. Which of the following is not an example of a 'Small-scale Embedded System'?
   (a) Electronic Barbie doll (b) Simple calculator
   (c) Cell phone (d) Electronic toy car
5. The first recognised modern embedded system is
   (a) Apple Computer (b) Apollo Guidance Computer (AGC)
   (c) Calculator (d) Radio Navigation System
6. The first mass produced embedded system is
   (a) Minuteman-I (b) Minuteman-II
   (c) Autonetics D-17 (d) Apollo Guidance Computer (AGC)
7. Which of the following is (are) an intended purpose(s) of embedded systems?
   (a) Data collection (b) Data processing (c) Data communication
   (d) All of these (e) None of these
8. Which of the following is an (are) example(s) of embedded system for data communication?
   (a) USB Mass storage device (b) Network router
   (c) Digital camera (d) Music player
   (e) All of these (f) None of these
9. A digital multi meter is an example of an embedded system for
   (a) Data communication (b) Monitoring (c) Control (d) All of these
   (e) None of these
10. Which of the following is an (are) example(s) of an embedded system for signal processing?
    (a) Apple iPOD (media player device) (b) SanDisk USB mass storage device
    (c) Both (a) and (b) (d) None of these

Review Questions

1. What is an embedded system? Explain the different applications of embedded systems.
2. Explain the various purposes of embedded systems in detail with illustrative examples.
3. Explain the different classifications of embedded systems. Give an example for each.

https://hemanthrajhemu.github.io

2 The Typical Embedded System

LEARNING OBJECTIVES

- ✓ *Learn the building blacks af a typical Embedded System*
- ✓ *Learn about General Purpase Processars (GPPs), Applicatian Specific Instruction Set Pracessors (ASIPs), Micropro-cessars, Microcontrollers, Digital Signal Processors, RISC & CISC processars, Harvard and Van-Neumann Processor Architecture, Big-endian v/s Little endian processars, Load Store operation and Instructian pipelining*
- ✓ *Learn about different PLDs like Complex Programmable Logic Devices (CPLDs), Field Programmable Gate Arrays (FPGAs), etc.*
- ✓ *Learn about the different memory technologies and memary types used in embedded system development*
- ✓ *Learn about Masked ROM (MROM), PROM, OTP, EPROM, EEPROM and FLASH memory for embedded firmware storage*
- ✓ *Learn about Serial Access Memory (SAM), Static Random Access Memary (SRAM), Dynamic Randam Access Memary (DRAM) and Nanvalatile SRAM (NVRAM)*
- ✓ *Understand the different factors to be considered in the selection af memory for embedded systems*
- ✓ *Understand the rale of sensars, actuatars and their interfacing with the I/O subsystems af an embedded system*
- ✓ *Learn about the interfacing of LEDs, 7-segment LED Displays, Piezo Buzzer, Stepper Motor, Relays, Optocouplers, Matrix keybaard, Push buttan switches, Pragrammable Peripheral Interface Device (e.g. 8255 PPI), etc. with the I/O subsystem of the embedded system*
- ✓ *Learn abaut the different communication interfaces af an embedded system*
- ✓ *Understand the various chip level communication interfaces like I2C, SPI, UART, 1-wire, parallel bus, etc.*
- ✓ *Understand the different wired and wireless external communicatian interfaces like RS-232C, RS-485, Parallel Part, USB, IEEE1394, Infrared (IrDA), Bluetooth, Wi-Fi, ZigBee, GPRS, etc.*
- ✓ *Knaw what embedded firmware is and its role in embedded systems*
- ✓ *Understand the different system components like Reset Circuit, Brawn-out protection circuit, Oscillator Unit, Real-Time Clock (RTC) and Watchdog Timer unit*
- ✓ *Understand the role of PCB in embedded systems*

A typical embedded system (Fig. 2.1) contains a single chip controller, which acts as the master brain of the system. The controller can be a Microprocessor (e.g. *Intel 8085*) or a microcontroller (e.g. Atmel AT89C51) or a Field Programmable Gate Array (FPGA) device (e.g. *Xilinx Spartan*) or a Digital Signal Processor (DSP) (e.g. *Blackfin® Processors* from Analog Devices) or an Application Specific Integrated

https://hemanthrajhemu.github.io

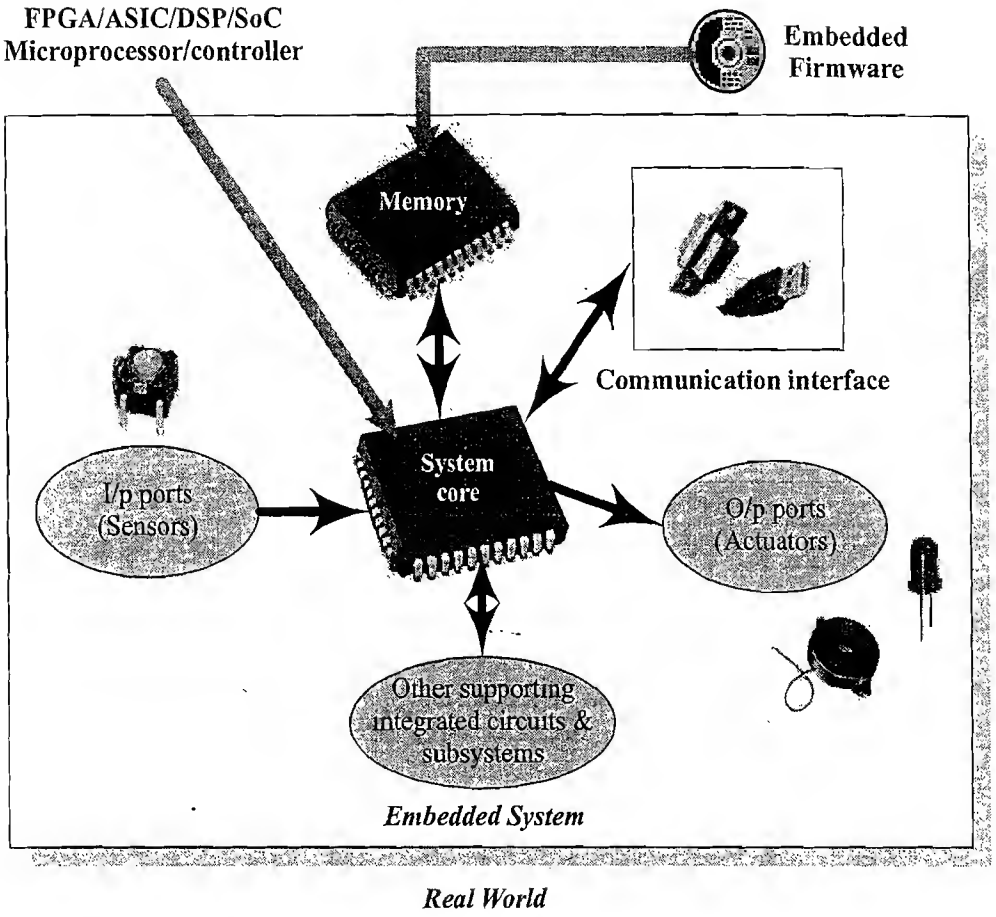


Fig. 2.1 **Elements of an embedded system**

Circuit (ASIC)/Application Specific Standard Product (ASSP) (e.g. ADE7760 Single Phase Energy Metreing IC from) Analog Devices for energy metering applications).

Embedded hardware/software systems are basically designed to regulate a physical variable or to manipulate the state of some devices by sending some control signals to the Actuators or devices connected to the O/p ports of the system, in response to the input signals provided by the end users or Sensors which are connected to the input ports. Hence an embedded system can be viewed as a reactive system. The control is achieved by processing the information coming from the sensors and user interfaces, and controlling some actuators that regulate the physical variable.

Key boards, push button switches, etc. are examples for common user interface input devices whereas LEDs, liquid crystal displays, piezoelectric buzzers, etc. are examples for common user interface output devices for a typical embedded system. It should be noted that it is not necessary that all embedded systems should incorporate these I/O user interfaces. It solely depends on the type of the application for which the embedded system is designed. For example, if the embedded system is designed for any handheld application, such as a mobile handset application, then the system should contain user interfaces like a keyboard for performing input operations and display unit for providing users the status of various activities in progress.

Some embedded systems do not require any manual intervention for their operation. They automatically sense the variations in the input parameters in accordance with the changes in the real world, to which they are interacting through the sensors which are connected to the input port of the system. The

https://hemanthrajhemu.github.io

sensor information is passed to the processor after signal conditioning and digitisation. Upon receiving the sensor data the processor or brain of the embedded system performs some pre-defined operations with the help of the firmware embedded in the system and sends some actuating signals to the actuator connected to the output port of the embedded system, which in turn acts on the controlling variable to bring the controlled variable to the desired level to make the embedded system work in the desired manner.

The Memory of the system is responsible for holding the control algorithm and other important configuration details. For most of embedded systems, the memory for storing the algorithm or configuration data is of fixed type, which is a kind of Read Only Memory (ROM) and it is not available for the end user for modifications, which means the memory is protected from unwanted user interaction by implementing some kind of memory protection mechanism. The most common types of memories used in embedded systems for control algorithm storage are OTP, PROM, UVEPROM, EEPROM and FLASH. Depending on the control application, the memory size may vary from a few bytes to megabytes. We will discuss them in detail in the coming sections. Sometimes the system requires temporary memory for performing arithmetic operations or control algorithm execution and this type of memory is known as "working memory". Random Access Memory (RAM) is used in most of the systems as the working memory. Various types of RAM like SRAM, DRAM and NVRAM are used for this purpose. The size of the RAM also varies from a few bytes to kilobytes or megabytes depending on the application. The details given under the section "Memory" will give you a more detailed description of the working memory.

An embedded system without a control algorithm implemented memory is just like a new born baby. It is having all the peripherals but is not capable of making any decision depending on the situational as well as real world changes. The only difference is that the memory of a new born baby is self-adaptive, meaning that the baby will try to learn from the surroundings and from the mistakes committed. For embedded systems it is the responsibility of the designer to impart intelligence to the system.

In a controller-based embedded system, the controller may contain internal memory for storing the control algorithm and it may be an EEPROM or FLASH memory varying from a few kilobytes to megabytes. Such controllers are called controllers with on-chip ROM, e.g. Atmel AT89C51. Some controllers may not contain on-chip memory and they require an external (off-chip) memory for holding the control algorithm, e.g. Intel 8031AH.

2.1 CORE OF THE EMBEDDED SYSTEM

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

1. General Purpose and Domain Specific Processors
   - 1.1 Microprocessors
   - 1.2 Microcontrollers
   - 1.3 Digital Signal Processors
2. Application Specific Integrated Circuits (ASICs)
3. Programmable Logic Devices (PLDs)
4. Commercial off-the-shelf Components (COTS)

If you examine any embedded system you will find that it is built around any of the core units mentioned above.

https://hemanthrajhemu.github.io

2.1.1 General Purpose and Domain Specific Processors

Almost 80% of the embedded systems are processor/controller based. The processor may be a microprocessor or a microcontroller or a digital signal processor, depending on the domain and application. Most of the embedded systems in the industrial control and monitoring applications make use of the commonly available microprocessors or microcontrollers whereas domains which require signal processing such as speech coding, speech recognition, etc. make use of special kind of digital signal processors supplied by manufacturers like, Analog Devices, Texas Instruments, etc.

2.1.1.1 Microprocessors A Microprocessor is a silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of instructions, which is specific to the manufacturer. In general the CPU contains the Arithmetic and Logic Unit (ALU), control unit and working registers. A microprocessor is a dependent unit and it requires the combination of other hardware like memory, timer unit, and interrupt controller, etc. for proper functioning. Intel claims the credit for developing the first microprocessor unit *Intel 4004*, a 4bit processor which was released in November 1971. It featured 1K data memory, a 12bit program counter and 4K program memory, sixteen 4bit general purpose registers and 46 instructions. It ran at a clock speed of 740 kHz. It was designed for olden day's calculators. In 1972, 14 more instructions were added to the 4004 instruction set and the program space is upgraded to 8K. Also interrupt capabilities were added to it and it is renamed as *Intel 4040*. It was quickly replaced in April 1972 by *Intel 8008* which was similar to *Intel 4040*, the only difference was that its program counter was 14 bits wide and the *8008* served as a terminal controller. In April 1974 Intel launched the first 8 bit processor, the *Intel 8080*, with 16bit address bus and program counter and seven 8bit registers (A-E,H,L: BC, DE, and HL pairs formed the 16bit register for this processor). *Intel 8080* was the most commonly used processors for industrial control and other embedded applications in the 1975s. Since the processor required other hardware components as mentioned earlier for its proper functioning, the systems made out of it were bulky and were lacking compactness.

Immediately after the release of *Intel 8080*, Motorola also entered the market with their processor, *Motorola 6800* with a different architecture and instruction set compared to *8080*.

In 1976 Intel came up with the upgraded version of *8080 – Intel 8085*, with two newly added instructions, three interrupt pins and serial I/O. Clock generator and bus controller circuits were built-in and the power supply part was modified to a single +5 V supply.

In July 1976 Zilog entered the microprocessor market with its *Z80 processor* as competitor to *Intel*. Actually it was designed by an ex-Intel designer, *Frederico Faggin* and it was an improved version of Intel's *8080* processor, maintaining the original *8080* architecture and instruction set with an 8bit data bus and a 16bit address bus and was capable of executing all instructions of *8080*. It included 80 more new instructions and it brought out the concept of register banking by doubling the register set. *Z80* also included two sets of index registers for flexible design.

Technical advances in the field of semiconductor industry brought a new dimension to the microprocessor market and twentieth century witnessed a fast growth in processor technology. 16, 32 and 64 bit processors came into the place of conventional 8bit processors. The initial 2 MHz clock is now an old story. Today processors with clock speeds up to 2.4 GHz are available in the market. More and more competitors entered into the processor market offering high speed, high performance and low cost processors for customer design needs.

Intel, AMD, Freescale, IBM, TI, Cyrix, Hitachi, NEC, LSI Logic, etc. are the key players in the processor market. Intel still leads the market with cutting edge technologies in the processor industry.

https://hemanthrajhemu.github.io

Different instruction set and system architecture are available for the design of a microprocessor. Harvard and Von-Neumann are the two common system architectures for processor design. Processors based on Harvard architecture contains separate buses for program memory and data memory, whereas processors based on Von-Neumann architecture shares a single system bus for program and data memory. We will discuss more about these architectures later, under a separate topic. Reduced Instruction Set Computing (RISC) and Complex Instruction Set Computing (CISC) are the two common Instruction Set Architectures (ISA) available for processor design. We will discuss the same under a separate topic in this section..

2.1.1.2 General Purpose Processor (GPP) vs. Application-Specific Instruction Set Processor (ASIP) A General Purpose Processor or GPP is a processor designed for general computational tasks. The processor running inside your laptop or desktop (Pentium 4/AMD Athlon, etc.) is a typical example for general purpose processor. They are produced in large volumes and targeting the general market. Due to the high volume production, the per unit cost for a chip is low compared to ASIC or other specific ICs. A typical general purpose processor contains an Arithmetic and Logic Unit (ALU) and Control Unit (CU). On the other hand, Application Specific Instruction Set Processors (ASIPs) are processors with architecture and instruction set optimised to specific-domain/application requirements like network processing, automotive, telecom, media applications, digital signal processing, control applications, etc. ASIPs fill the architectural spectrum between general purpose processors and Application Specific Integrated Circuits (ASICs). The need for an ASIP arises when the traditional general purpose processor are unable to meet the increasing application needs. Most of the embedded systems are built around application specific instruction set processors. Some microcontrollers (like automotive AVR, USB AVR from Atmel), system on chips, digital signal processors, etc. are examples for application specific instruction set processors (ASIPs). ASIPs incorporate a processor and on-chip peripherals, demanded by the application requirement, program and data memory.

2.1.1.3 Microcontrollers A Microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports. Microcontrollers can be considered as a super set of microprocessors. Since a microcontroller contains all the necessary functional blocks for independent working, they found greater place in the embedded domain in place of microprocessors. Apart from this, they are cheap, cost effective and are readily available in the market.

Texas Instrument's *TMS 1000* is considered as the world's first microcontroller. We cannot say it as a fully functional microcontroller when we compare it with modern microcontrollers. TI followed Intel's *4004/4040*, 4 bit processor design and added some amount of RAM, program storage memory (ROM) and I/O support on a single chip, there by eliminated the requirement of multiple hardware chips for self-functioning. Provision to add custom instructions to the CPU was another innovative feature of TMS 1000. TMS 1000 was released in 1974.

In 1977 Intel entered the microcontroller market with a family of controllers coming under one umbrella named *MCS-48*™ family. The processors came under this family were *8038HL*, *8039HL*, *8040AHL*, *8048H*, *8049H* and *8050AH*. *Intel 8048* is recognised as Intel's first microcontroller and it was the most prominent member in the MCS-48™† family. It was used in the original IBM PC keyboard. The inspiration behind *8048* was Fairchild's *F8* microprocessor and Intel's goal of developing a low cost and small size processor. The design of *8048* adopted a true Harvard architecture where program and data memory shared the same address bus and is differentiated by the related control signals.

†MCS-48™ is a trade mark owned by Intel

https://hemanthrajhemu.github.io

Eventually Intel came out with its most fruitful design in the 8bit microcontroller domain–the *8051 family* and its derivatives. It is the most popular and powerful 8bit microcontroller ever built. It was developed in the 1980s and was put under the family MCS-51. Almost 75% of the microcontrollers used in the embedded domain were *8051 family* based controllers during the 1980–90s. *8051* processor cores are used in more than 100 devices by more than 20 independent manufacturers like Maxim, Philips, Atmel, etc. under the license from Intel. Due to the low cost, wide availability, memory efficient instruction set, mature development tools and Boolean processing (bit manipulation operation) capability, *8051 family* derivative microcontrollers are much used in high-volume consumer electronic devices, entertainment industry and other gadgets where cost-cutting is essential.

Another important family of microcontrollers used in industrial control and embedded applications is the **PIC** family micro controllers from Microchip Technologies (It will be discussed in detail in a later section of this book). It is a high performance RISC microcontroller complementing the CISC (complex instruction set computing) features of *8051*. The terms RISC and CISC will be explained in detail in a separate heading.

Some embedded system applications require only 8bit controllers whereas some embedded applications requiring superior performance and computational needs demand 16/32bit microcontrollers. Infineon, Freescale, Philips, Atmel, Maxim, Microchip etc. are the key suppliers of 16bit microcontrollers. Philips tried to extend the *8051* family microcontrollers to use for 16bit applications by developing the Philips XA (eXtended Architecture) microcontroller series.

8bit microcontrollers are commonly used in embedded systems where the processing power is not a big constraint. As mentioned earlier, more than 20 companies are producing different flavours of the *8051* family microcontroller. They try to add more and more functionalities like built in SPI, I2C serial buses, USB controller, ADC, Networking capability, etc. So the competitive market is driving towards a one-stop solution chip in microcontroller domain. High processing speed microcontroller families like ARM11 series are also available in the market, which provides solution to applications requiring hardware acceleration and high processing capability.

Freescale, NEC, Zilog, Hitachi, Mitsubishi, Infineon, ST Micro Electronics, National, Texas Instruments, Toshiba, Philips, Microchip, Analog Devices, Daewoo, Intel, Maxim, Sharp, Silicon Laboratories, TDK, Triscend, Winbond, Atmel, etc. are the key players in the microcontroller market. Of these Atmel has got special significance. They are the manufacturers of a variety of Flash memory based microcontrollers. They also provide In-System Programmability (which will be discussed in detail in a later section of this book) for the controller. The Flash memory technique helps in fast reprogramming of the chip and thereby reduces the product development time. Atmel also provides another special family of microcontroller called AVR (it will be discussed in detail in a later chapter), an 8bit RISC Flash microcontroller, fast enough to execute powerful instructions in a single clock cycle and provide the latitude you need to optimise power consumption.

The instruction set architecture of a microcontroller can be either RISC or CISC. Microcontrollers are designed for either general purpose application requirement (general purpose controller) or domain-specific application requirement (application specific instruction set processor). The *Intel 8051* microcontroller is a typical example for a general purpose microcontroller, whereas the automotive AVR microcontroller family from Atmel Corporation is a typical example for ASIP specifically designed for the automotive domain.

2.1.1.4 Microprocessor vs Microcontroller The following table summarises the differences between a microcontroller and microprocessor.

https://hemanthrajhemu.github.io

Microprocessor	Microcontroller
A silicon chip representing a central processing unit (CPU), which is capable of performing arithmetic as well as logical operations according to a pre-defined set of instructions	A microcontroller is a highly integrated chip that contains a CPU, scratchpad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports
It is a dependent unit. It requires the combination of other chips like timers, program and data memory chips, interrupt controllers, etc. for functioning	It is a self-contained unit and it doesn't require external interrupt controller, timer, UART, etc. for its functioning
Most of the time general purpose in design and operation	Mostly application-oriented or domain-specific
Doesn't contain a built in I/O port. The I/O port functionality needs to be implemented with the help of external programmable peripheral interface chips like 8255	Most of the processors contain multiple built-in I/O ports which can be operated as a single 8 or 16 or 32 bit port or as individual port pins
Targeted for high end market where performance is important	Targeted for embedded market where performance is not so critical (At present this demarcation is invalid)
Limited power saving options compared to microcontrollers	Includes lot of power saving features

2.1.1.5 Digital Signal Processors Digital Signal Processors (DSPs) are powerful special purpose 8/16/32 bit microprocessors designed specifically to meet the computational demands and power constraints of today's embedded audio, video, and communications applications. Digital signal processors are 2 to 3 times faster than the general purpose microprocessors in signal processing applications. This is because of the architectural difference between the two. DSPs implement algorithms in hardware which speeds up the execution whereas general purpose processors implement the algorithm in firmware and the speed of execution depends primarily on the clock for the processors. In general, DSP can be viewed as a microchip designed for performing high speed computational operations for 'addition', 'subtraction', 'multiplication' and 'division'. A typical digital signal processor incorporates the following key units:

Program Memory Memory for storing the program required by DSP to process the data

Data Memory Working memory for storing temporary variables and data/signal to be processed.

Computational Engine Performs the signal processing in accordance with the stored program memory. Computational Engine incorporates many specialised arithmetic units and each of them operates simultaneously to increase the execution speed. It also incorporates multiple hardware shifters for shifting operands and thereby saves execution time.

I/O Unit Acts as an interface between the outside world and DSP. It is responsible for capturing signals to be processed and delivering the processed signals.

https://hemanthrajhemu.github.io

Audio video signal processing, telecommunication and multimedia applications are typical examples where DSP is employed. Digital signal processing employs a large amount of real-time calculations. Sum of products (SOP) calculation, convolution, fast fourier transform (FFT), discrete fourier transform (DFT), etc, are some of the operations performed by digital signal processors.

Blackfin®† processors from Analog Devices is an example of DSP which delivers breakthrough signal-processing performance and power efficiency while also offering a full 32-bit RISC MCU programming model. Blackfin processors present high-performance, homogeneous software targets, which allows flexible resource allocation between hard real-time signal processing tasks and non real-time control tasks. System control tasks can often run in the shadow of demanding signal processing and multimedia tasks.

2.1.1.6 RISC vs. CISC Processors/Controllers The term RISC stands for Reduced Instruction Set Computing. As the name implies, all RISC processors/controllers possess lesser number of instructions, typically in the range of 30 to 40. CISC stands for Complex Instruction Set Computing. From the definition itself it is clear that the instruction set is complex and instructions are high in number. From a programmers point of view RISC processors are comfortable since s/he needs to learn only a few instructions, whereas for a CISC processor s/he needs to learn more number of instructions and should understand the context of usage of each instruction (This scenario is explained on the basis of a programmer following Assembly Language coding. For a programmer following C coding it doesn't matter since the cross-compiler is responsible for the conversion of the high level language instructions to machine dependent code). Atmel AVR microcontroller is an example for a RISC processor and its instruction set contains only 32 instructions. The original version of *8051* microcontroller (e.g. *AT89C51*) is a CISC controller and its instruction set contains 255 instructions. Remember it is not the number of instructions that determines whether a processor/controller is CISC or RISC. There are some other factors like pipelining features, instruction set type, etc. for determining the RISC/CISC criteria. Some of the important criteria are listed below:

RISC	CISC
Lesser number of instructions	Greater number of Instructions
Instruction pipelining and increased execution speed	Generally no instruction pipelining feature
Orthogonal instruction set (Allows each instruction to operate on any register and use any addressing mode)	Non-orthogonal instruction set (All instructions are not allowed to operate on any register and use any addressing mode. It is instruction-specific)
Operations are performed on registers only, the only memory operations are load and store	Operations are performed on registers or memory depending on the instruction
A large number of registers are available	Limited number of general purpose registers
Programmer needs to write more code to execute a task since the instructions are simpler ones	Instructions are like macros in C language. A programmer can achieve the desired functionality with a single instruction which in turn provides the effect of using more simpler single instructions in RISC
Single, fixed length instructions	Variable length instructions
Less silicon usage and pin count	More silicon usage since more additional decoder logic is required to implement the complex instruction decoding.
With Harvard Architecture	Can be Harvard or Von-Neumann Architecture

I hope now you are clear about the terms RISC and CISC in the processor technology. Isn't it?

†Blackfin® is a Registered trademark of Analog Devices Inc.

https://hemanthrajhemu.github.io

2.1.1.7 Harvard vs. Von-Neumann Processor/Controller Architecture The terms Harvard and Von-Neumann refers to the processor architecture design.

Microprocessors/controllers based on the **Von-Neumann** architecture shares a single common bus for fetching both instructions and data. Program instructions and data are stored in a common main memory. Von-Neumann architecture based processors/controllers first fetch an instruction and then fetch the data to support the instruction from code memory. The two separate fetches slows down the controller's operation. Von-Neumann architecture is also referred as **Princeton** architecture, since it was developed by the Princeton University.

Microprocessors/controllers based on the **Harvard** architecture will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both buses. With Harvard architecture, the data memory can be read and written while the program memory is being accessed. These separated data memory and code memory buses allow one instruction to execute while the next instruction is fetched ("pre-fetching"). The pre-fetch theoretically allows much faster execution than Von-Neumann architecture. Since some additional hardware logic is required for the generation of control signals for this type of operation it adds silicon complexity to the system. Figure 2.2 explains the Harvard and Von-Neumann architecture concept.

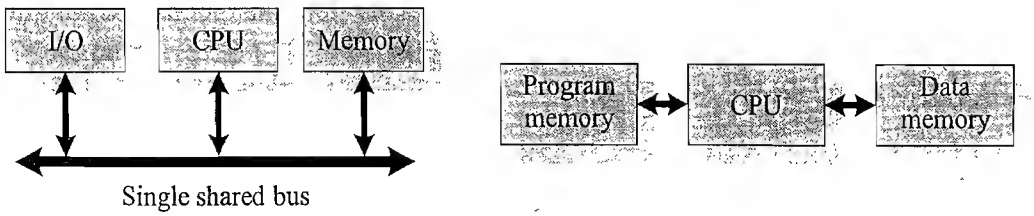


Fig. 2.2 **Harvard vs Von-Neumann architecture**

The following table highlights the differences between Harvard and Von-Neumann architecture.

Harvard Architecture	Von-Neumann Architecture
Separate buses for instruction and data fetching	Single shared bus for instruction and data fetching
Easier to pipeline, so high performance can be achieved	Low performance compared to Harvard architecture
Comparatively high cost	Cheaper
No memory alignment problems	Allows self modifying codes[†]
Since data memory and program memory are stored physically in different locations, no chances for accidental corruption of program memory	Since data memory and program memory are stored physically in the same chip, chances for accidental corruption of program memory

2.1.1.8 Big-Endian vs. Little-Endian Processors/Controllers Endianness specifies the order in which the data is stored in the memory by processor operations in a multi byte system (Processors whose word size is greater than one byte). Suppose the word length is two byte then data can be stored in memory in two different ways:

1. Higher order of data byte at the higher memory and lower order of data byte at location just below the higher memory.
2. Lower order of data byte at the higher memory and higher order of data byte at location just below the higher memory.

[†]Self-modifying code is a code/instruction which modifies itself while execution.

https://hemanthrajhemu.github.io

Little-endian (Fig. 2.3) means the lower-order byte of the data is stored in memory at the lowest address, and the higher-order byte at the highest address. (The little end comes first.) For example, a 4 byte long integer **Byte3 Byte2 Byte1 Byte0** will be stored in the memory as shown below:

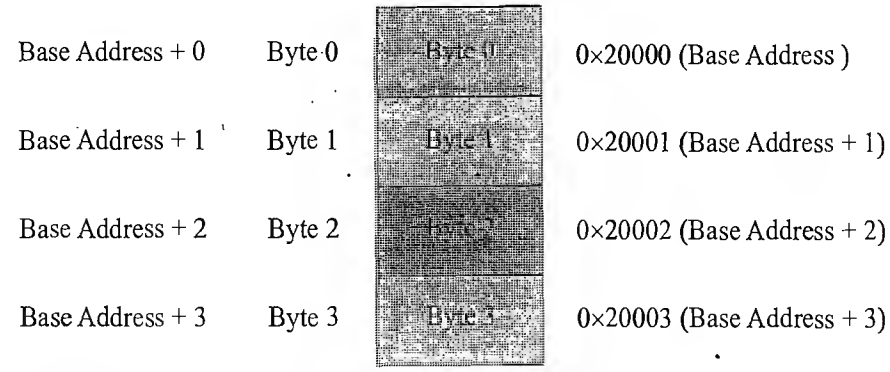


Fig. 2.3 **Little-Endian operation**

Big-endian (Fig. 2.4) means the higher-order byte of the data is stored in memory at the lowest address, and the lower-order byte at the highest address. (The big end comes first.) For example, a 4 byte long integer **Byte3 Byte2 Byte1 Byte0** will be stored in the memory as follows‡:

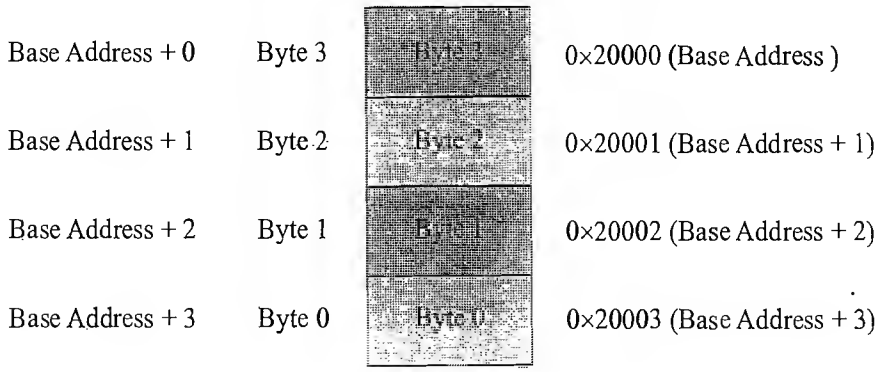


Fig. 2.4 **Big-Endian operation**

2.1.1.9 Load Store Operation and Instruction Pipelining As mentioned earlier, the RISC processor instruction set is orthogonal, meaning it operates on registers. The memory access related operations are performed by the special instructions *load* and *store*. If the operand is specified as memory location, the content of it is loaded to a register using the *load* instruction. The instruction *store* stores data from a specified register to a specified memory location. The concept of **Load Store Architecture** is illustrated with the following example:

Suppose *x*, *y* and *z* are memory locations and we want to add the contents of *x* and *y* and store the result in location *z*. Under the load store architecture the same is achieved with 4 instructions as shown in Fig. 2.5.

The first instruction *load R1, x* loads the register R1 with the content of memory location *x*, the second instruction *load R2,y* loads the register R2 with the content of memory location *y*. The instruction

‡ Note that the base address is chosen arbitrarily as 0×20000

https://hemanthrajhemu.github.io

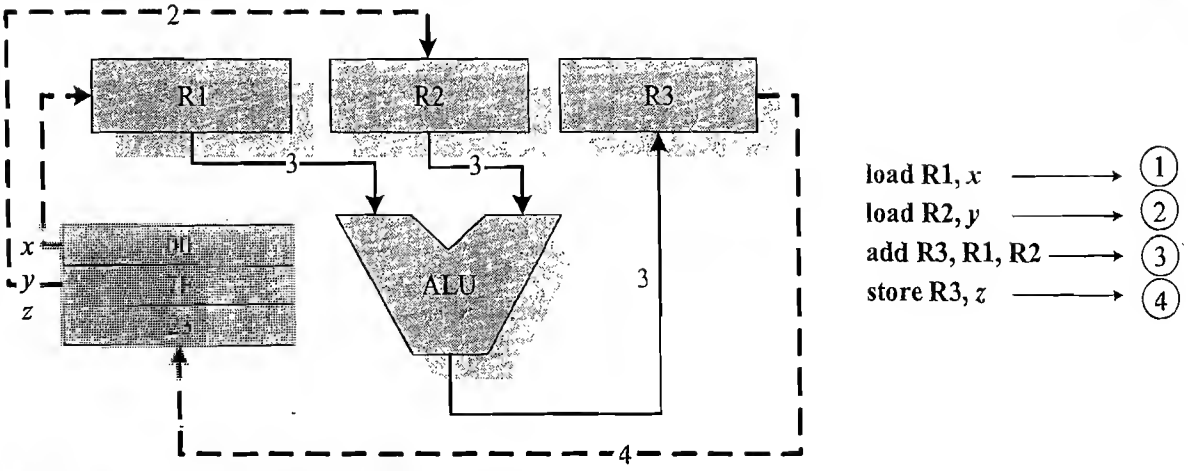


Fig. 2.5 **The concept of load store architecture**

add R3, R1, R2 adds the content of registers R1 and R2 and stores the result in register R3. The next instruction *store R3,z* stores the content of register R3 in memory location *z*.

The conventional instruction execution by the processor follows the fetch-decode-execute sequence. Where the 'fetch' part fetches the instruction from program memory or code memory and the decode part decodes the instruction to generate the necessary control signals. The execute stage reads the operands, perform ALU operations and stores the result. In conventional program execution, the fetch and decode operations are performed in sequence. For simplicity let's consider decode and execution together. During the decode operation the memory address bus is available and if it is possible to effectively utilise it for an instruction fetch, the processing speed can be increased. In its simplest form instruction pipelining refers to the overlapped execution of instructions. Under normal program execution flow it is meaningful to fetch the next instruction to execute, while the decoding and execution of the current instruction is in progress. If the current instruction in progress is a program control flow transfer instruction like jump or call instruction, there is no meaning in fetching the instruction following the current instruction. In such cases the instruction fetched is flushed and a new instruction fetch is performed to fetch the instruction. Whenever the current instruction is executing the program counter will be loaded with the address of the next instruction. In case of jump or branch instruction, the new location is known only after completion of the jump or branch instruction. Depending on the stages involved in an instruction (fetch, read register and decode, execute instruction, access an operand in data memory, write back the result to register, etc.), there can be multiple levels of instruction pipelining. Figure 2.6 illustrates the concept of Instruction pipelining for single stage pipelining.

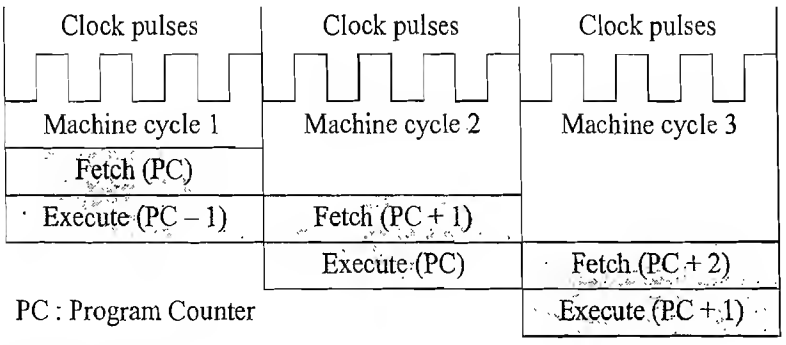


Fig. 2.6 **The single-stage pipelining concept**

https://hemanthrajhemu.github.io

2.1.2 Application Specific Integrated Circuits (ASICs)

Application Specific Integrated Circuit (ASIC) is a microchip designed to perform a specific or unique application. It is used as replacement to conventional general purpose logic chips. It integrates several functions into a single chip and there by reduces the system development cost. Most of the ASICs are proprietary products. As a single chip, ASIC consumes a very small area in the total system and thereby helps in the design of smaller systems with high capabilities/functionalities.

ASICs can be pre-fabricated for a special application or it can be custom fabricated by using the components from a re-usable '*building block*' library of components for a particular customer application. ASIC based systems are profitable only for large volume commercial productions. Fabrication of ASICs requires a non-refundable initial investment for the process technology and configuration expenses. This investment is known as Non-Recurring Engineering Charge (NRE) and it is a one time investment.

If the Non-Recurring Engineering Charges (NRE) is borne by a third party and the Application Specific Integrated Circuit (ASIC) is made openly available in the market, the ASIC is referred as Application Specific Standard Product (ASSP). The ASSP is marketed to multiple customers just as a general-purpose product is, but to a smaller number of customers since it is for a specific application. "The ADE7760 Energy Metre ASIC developed by Analog Devices for Energy metreing applications is a typical example for ASSP".

Since Application Specific Integrated Circuits (ASICs) are proprietary products, the developers of such chips may not be interested in revealing the internal details of it and hence it is very difficult to point out an example of it. Moreover it will create legal disputes if an illustration of such an ASIC product is given without getting prior permission from the manufacturer of the ASIC. For the time being, let us forget about it. We will come back to it in another part of this book series (Namely, Designing Advanced Embedded Systems).

2.1.3 Programmable Logic Devices

Logic devices provide specific functions, including device-to-device interfacing, data communication, signal processing, data display, timing and control operations, and almost every other function a system must perform. Logic devices can be classified into two broad categories–fixed and programmable. As the name indicates, the circuits in a fixed logic device are permanent, they perform one function or set of functions–once manufactured, they cannot be changed. On the other hand, Programmable Logic Devices (PLDs) offer customers a wide range of logic capacity, features, speed, and voltage characteristics–and these devices can be re-configured to perform any number of functions at any time.

With programmable logic devices, designers use inexpensive software tools to quickly develop, simulate, and test their designs. Then, a design can be quickly programmed into a device, and immediately tested in a live circuit. The PLD that is used for this prototyping is the exact same PLD that will be used in the final production of a piece of end equipment, such as a network router, a DSL modem, a DVD player, or an automotive navigation system. There are no NRE costs and the final design is completed much faster than that of a custom, fixed logic device. Another key benefit of using PLDs is that during the design phase customers can change the circuitry as often as they want until the design operates to their satisfaction. That's because PLDs are based on re-writable memory technology–to change the design, the device is simply reprogrammed. Once the design is final, customers can go into immediate production by simply programming as many PLDs as they need with the final software design file.

2.1.3.1 CPLDs and FPGAs The two major types of programmable logic devices are Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). Of the two, FPGAs

https://hemanthrajhemu.github.io

offer the highest amount of logic density, the most features, and the highest performance. The largest FPGA now shipping, part of the Xilinx **Virtex™†** line of devices, provides eight million "system gates" (the relative density of logic). These advanced devices also offer features such as built-in hardwired processors (such as the IBM power PC), substantial amounts of memory, clock management systems, and support for many of the latest, very fast device-to-device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, to instrumentation, telecommunications, and digital signal processing.

CPLDs, by contrast, offer much smaller amounts of logic–up to about 10,000 gates. But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications. CPLDs such as the Xilinx **CoolRunner™†** series also require extremely low amounts of power and are very inexpensive, making them ideal for cost-sensitive, battery-operated, portable applications such as mobile phones and digital handheld assistants.

Advantages of PLD Programmable logic devices offer a number of important advantages over fixed logic devices, including:

- PLDs offer customers much more flexibility during the design cycle because design iterations are simply a matter of changing the programming file, and the results of design changes can be seen immediately in working parts.
- PLDs do not require long lead times for prototypes or production parts–the PLDs are already on a distributor's shelf and ready for shipment.
- PLDs do not require customers to pay for large NRE costs and purchase expensive mask sets–PLD suppliers incur those costs when they design their programmable devices and are able to amortize those costs over the multi-year lifespan of a given line of PLDs.
- PLDs allow customers to order just the number of parts they need, when they need them, allowing them to control inventory. Customers who use fixed logic devices often end up with excess inventory which must be scrapped, or if demand for their product surges, they may be caught short of parts and face production delays.
- PLDs can be reprogrammed even after a piece of equipment is shipped to a customer. In fact, thanks to programmable logic devices, a number of equipment manufacturers now tout the ability to add new features or upgrade products that already are in the field. To do this, they simply upload a new programming file to the PLD, via the Internet, creating new hardware logic in the system.

Over the last few years programmable logic suppliers have made such phenomenal technical advances that PLDs are now seen as the logic solution of choice from many designers. One reason for this is that PLD suppliers such as Xilinx are "fabless" companies; instead of owning chip manufacturing foundries, Xilinx outsource that job to partners like Toshiba and UMC, whose chief occupation is making chips. This strategy allows Xilinx to focus on designing new product architectures, software tools, and intellectual property cores while having access to the most advanced semiconductor process technologies. Advanced process technologies help PLDs in a number of key areas: faster performance, integration of more features, reduced power consumption, and lower cost.

FPGAs are especially popular for prototyping ASIC designs where the designer can test his design by downloading the design file into an FPGA device. Once the design is set, hardwired chips are produced for faster performance.

Just a few years ago, for example, the largest FPGA was measured in tens of thousands of system gates and operated at 40 MHz. Older FPGAs also were relatively expensive, costing often more than $150 for the most advanced parts at the time. Today, however, FPGAs with advanced features offer

† **Virtex™** and **CoolRunner™** are the registered trademarks of Xilinx Inc.

https://hemanthrajhemu.github.io

millions of gates of logic capacity, operate at 300 MHz, can cost less than $10, and offer a new level of integrated functions such as processors and memory.

2.1.4 Commercial Off-the-Shelf Components (COTS)

A Commercial Off-the-Shelf (**COTS**) product is one which is used '*as-is*'. COTS products are designed in such a way to provide easy integration and interoperability with existing system components. The COTS component itself may be developed around a general purpose or domain specific processor or an Application Specific Integrated circuit or a programmable logic device. Typical examples of COTS hardware unit are remote controlled toy car control units including the RF circuitry part, high performance, high frequency microwave electronics (2–200 GHz), high bandwidth analog-to-digital converters, devices and components for operation at very high temperatures, electro-optic IR imaging arrays, UV/IR detectors, etc. The major advantage of using COTS is that they are readily available in the market, are cheap and a developer can cut down his/her development time to a great extent. This in turn reduces the time to market your embedded systems.

The TCP/IP plug-in module available from various manufactures like 'WIZnet', 'Freescale', 'Dynalog', etc. are very good examples of COTS product (Fig. 2.7). This network plug-in module gives the TCP/IP connectivity to the system you are developing. There is no need to design this module yourself and write the firmware for the TCP/IP protocol and data transfer. Everything will be readily supplied by the COTS manufacturer. What you need to do is identify the COTS for your system and give the plug-in option on your board according to the hardware plug-in connections given in the specifications of the COTS. Though multiple vendors supply COTS for the same application, the major problem faced by the end-user is that there are no operational and manufacturing standards. A Commercial off-the-shelf (COTS) component manufactured by a vendor need not have hardware plug-in and firmware interface compatibility with one manufactured by a second vendor for the same application. This restricts the end-user to stick to a particular vendor for a particular COTS. This greatly affects the product design.

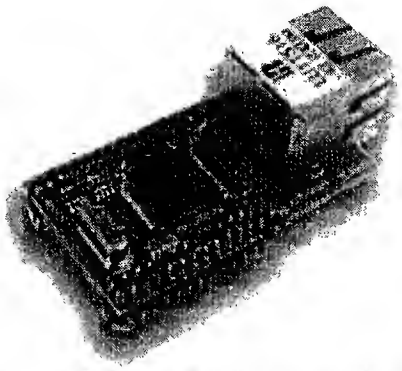


Fig. 2.7 **An example of a COTS product for TCP/IP plug-in from WIZnet**
(WIZnet NM7010A Plug in Module-Courtesy of WIZnet http://www.wiznet.co.kr/en/)

The major drawback of using COTS components in embedded design is that the manufacturer of the COTS component may withdraw the product or discontinue the production of the COTS at any time if a rapid change in technology occurs, and this will adversely affect a commercial manufacturer of the embedded system which makes use of the specific COTS product.

2.2 MEMORY

Memory is an important part of a processor/controller based embedded systems. Some of the processors/controllers contain built in memory and this memory is referred as **on-chip memory**. Others do not contain any memory inside the chip and requires external memory to be connected with the controller/processor to store the control algorithm. It is called **off-chip memory**. Also some working memory is required for holding data temporarily during certain operations. This section deals with the different types of memory used in embedded system applications.

https://hemanthrajhemu.github.io

2.2.1 Program Storage Memory (ROM)

The program memory or code storage memory of an embedded system stores the program instructions and it can be classified into different types as per the block diagram representation given in Fig. 2.8.

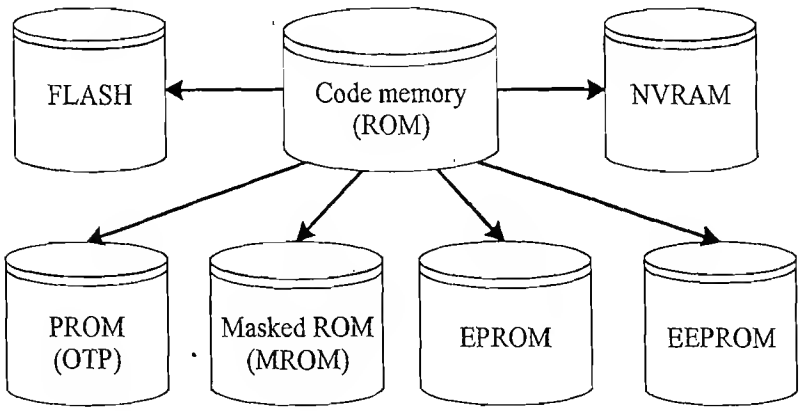


Fig. 2.8 **Classification of Program Memory (ROM)**

The code memory retains its contents even after the power to it is turned off. It is generally known as non-volatile storage memory. Depending on the fabrication, erasing and programming techniques they are classified into the following types.

2.2.1.1 Masked ROM (MROM) Masked ROM is a one-time programmable device. Masked ROM makes use of the hardwired technology for storing data. The device is factory programmed by masking and metallisation process at the time of production itself, according to the data provided by the end user. The primary advantage of this is low cost for high volume production. They are the least expensive type of solid state memory. Different mechanisms are used for the masking process of the ROM, like

1. Creation of an enhancement or depletion mode transistor through channel implant.
2. By creating the memory cell either using a standard transistor or a high threshold transistor. In the high threshold mode, the supply voltage required to turn ON the transistor is above the normal ROM IC operating voltage. This ensures that the transistor is always off and the memory cell stores always logic 0.

Masked ROM is a good candidate for storing the embedded firmware for low cost embedded devices. Once the design is proven and the firmware requirements are tested and frozen, the binary data (The firmware cross compiled/assembled to target processor specific machine code) corresponding to it can be given to the MROM fabricator. The limitation with MROM based firmware storage is the inability to modify the device firmware against firmware upgrades. Since the MROM is permanent in bit storage, it is not possible to alter the bit information.

2.2.1.2 Programmable Read Only Memory (PROM) / (OTP) Unlike Masked ROM Memory, One Time Programmable Memory (OTP) or PROM is not pre-programmed by the manufacturer. The end user is responsible for programming these devices. This memory has *nichrome* or *polysilicon* wires arranged in a matrix. These wires can be functionally viewed as fuses. It is programmed by a PROM programmer which selectively burns the fuses according to the bit pattern to be stored. Fuses which are not blown/burned represents a logic "1" whereas fuses which are blown/burned represents a logic "0". The default state is logic "1". OTP is widely used for commercial production of embedded systems whose proto-typed versions are proven and the code is finalised. It is a low cost solution for commercial production. OTPs cannot be reprogrammed.

https://hemanthrajhemu.github.io

2.2.1.3 Erasable Programmable Read Only Memory (EPROM) OTPs are not useful and worth for development purpose. During the development phase the code is subject to continuous changes and using an OTP each time to load the code is not economical. Erasable Programmable Read Only Memory (EPROM) gives the flexibility to re-program the same chip. EPROM stores the bit information by charging the floating gate of an FET. Bit information is stored by using an EPROM programmer, which applies high voltage to charge the floating gate. EPROM contains a quartz crystal window for erasing the stored information. If the window is exposed to ultraviolet rays for a fixed duration, the entire memory will be erased. Even though the EPROM chip is flexible in terms of re-programmability, it needs to be taken out of the circuit board and put in a UV eraser device for 20 to 30 minutes. So it is a tedious and time-consuming process.

2.2.1.4 Electrically Erasable Programmable Read Only Memory (EEPROM) As the name indicates, the information contained in the EEPROM memory can be altered by using electrical signals at the register/Byte level. They can be erased and reprogrammed in-circuit. These chips include a chip erase mode and in this mode they can be erased in a few milliseconds. It provides greater flexibility for system design. The only limitation is their capacity is limited when compared with the standard ROM (A few kilobytes).

2.2.1.5 FLASH FLASH is the latest ROM technology and is the most popular ROM technology used in today's embedded designs. FLASH memory is a variation of EEPROM technology. It combines the re-programmability of EEPROM and the high capacity of standard ROMs. FLASH memory is organised as sectors (blocks) or pages. FLASH memory stores information in an array of floating gate MOSFET transistors. The erasing of memory can be done at sector level or page level without affecting the other sectors or pages. Each sector/page should be erased before re-programming. The typical erasable capacity of FLASH is 1000 cycles. W27C512 from WINBOND (www.winbond.com) is an example of 64KB FLASH memory.

2.2.1.6 NVRAM Non-volatile RAM is a random access memory with battery backup. It contains static RAM based memory and a minute battery for providing supply to the memory in the absence of external power supply. The memory and battery are packed together in a single package. The life span of NVRAM is expected to be around 10 years. *DS1644* from Maxim/Dallas is an example of 32KB NVRAM.

2.2.2 Read-Write Memory/Random Access Memory (RAM)

RAM is the data memory or working memory of the controller/processor. Controller/processor can read from it and write to it. RAM is volatile, meaning when the power is turned off, all the contents are destroyed. RAM is a direct access memory, meaning we can access the desired memory location directly without the need for traversing through the entire memory locations to reach the desired memory position (i.e. random access of memory location). This is in contrast to the Sequential Access Memory (SAM), where the desired memory location is accessed by either traversing through the entire memory or through a 'seek' method. Magnetic tapes, CD ROMs, etc. are examples of sequential access memories. RAM generally falls into three categories: Static RAM (SRAM), dynamic RAM (DRAM) and non-volatile RAM (NVRAM) (Fig. 2.9).

2.2.2.1 Static RAM (SRAM) Static RAM stores data in the form of voltage. They are made up of flip-flops. Static RAM is the fastest form of RAM available. In typical implementation, an SRAM cell (bit) is realised using six transistors (or 6 MOSFETs). Four of the transistors are used for building the

https://hemanthrajhemu.github.io

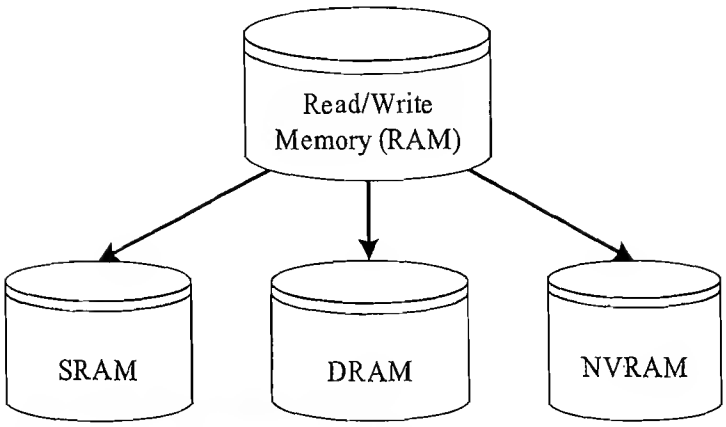


Fig. 2.9 **Classification of Working Memory (RAM)**

latch (flip-flop) part of the memory cell and two for controlling the access. SRAM is fast in operation due to its resistive networking and switching capabilities. In its simplest representation an SRAM cell can be visualised as shown in Fig. 2.10:

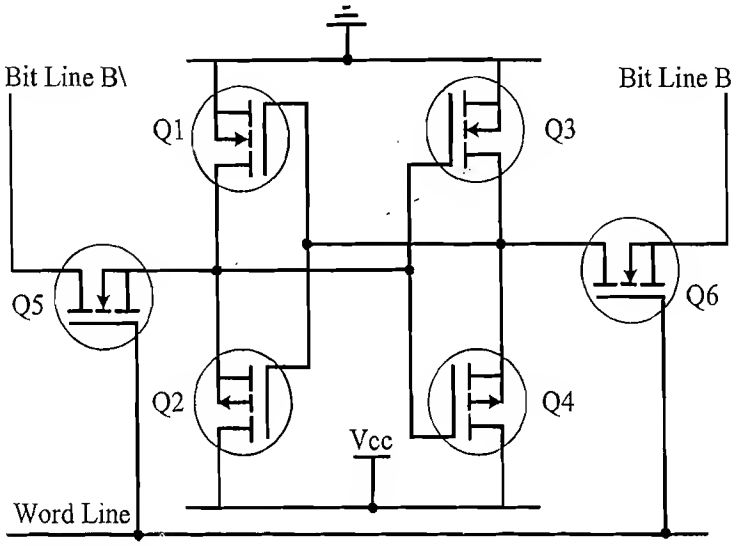


Fig. 2.10 **SRAM cell implementation**

This implementation in its simpler form can be visualised as two-cross coupled inverters with read/write control through transistors. The four transistors in the middle form the cross-coupled inverters. This can be visualised as shown in Fig. 2.11.

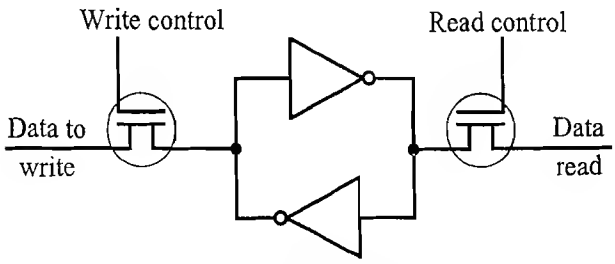


Fig. 2.11 **Visualisation of SRAM cell**

From the SRAM implementation diagram, it is clear that access to the memory cell is controlled by the line Word Line, which controls the access transistors (MOSFETs) Q5 and Q6. The access transistors control the connection to bit lines B & B\. In order to write a value to the memory cell, apply the desired value to the bit control lines (For writing 1, make B = 1 and B\ =0; For writing 0, make B = 0 and B\ =1) and assert the Word Line (Make Word line

https://hemanthrajhemu.github.io

high). This operation latches the bit written in the flip-flop. For reading the content of the memory cell, assert both B and B\ bit lines to 1 and set the Word line to 1.

The major limitations of SRAM are low capacity and high cost. Since a minimum of six transistors are required to build a single memory cell, imagine how many memory cells we can fabricate on a silicon wafer.

2.2.2.2 Dynamic RAM (DRAM) Dynamic RAM stores data in the form of charge. They are made up of MOS transistor gates. The advantages of DRAM are its high density and low cost compared to SRAM. The disadvantage is that since the information is stored as charge it gets leaked off with time and to prevent this they need to be refreshed periodically. Special circuits called DRAM controllers are used for the refreshing operation. The refresh operation is done periodically in milliseconds interval. Figure 2.12 illustrates the typical implementation of a DRAM cell.

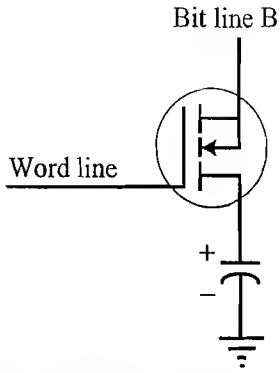


Fig. 2.12 **DRAM cell implementation**

The MOSFET acts as the gate for the incoming and outgoing data whereas the capacitor acts as the bit storage unit. Table given below summarises the relative merits and demerits of SRAM and DRAM technology.

SRAM cell	DRAM cell
Made up of 6 CMOS transistors (MOSFET)	Made up of a MOSFET and a capacitor
Doesn't require refreshing	Requires refreshing
Low capacity (Less dense)	High capacity (Highly dense)
More expensive	Less expensive
Fast in operation. Typical access time is 10ns	Slow in operation due to refresh requirements. Typical access time is 60ns. Write operation is faster than read operation.

2.2.2.3 NVRAM Non-volatile RAM is a random access memory with battery backup. It contains static RAM based memory and a minute battery for providing supply to the memory in the absence of external power supply. The memory and battery are packed together in a single package. NVRAM is used for the non-volatile storage of results of operations or for setting up of flags, etc. The life span of NVRAM is expected to be around 10 years. DS1744 from Maxim/Dallas is an example for 32KB NVRAM.

2.2.3 Memory According to the Type of Interface

The interface (connection) of memory with the processor/controller can be of various types. It may be a parallel interface [The parallel data lines (D0-D7) for an 8 bit processor/controller will be connected to D0-D7 of the memory] or the interface may be a serial interface like I2C (Pronounced as I Square C. It is a 2 line serial interface) or it may be an SPI (Serial peripheral interface, 2+*n* line interface where *n* stands for the total number of SPI bus devices in the system). It can also be of a single wire interconnection (like Dallas 1-Wire interface). Serial interface is commonly used for data storage memory like EEPROM. The memory density of a serial memory is usually expressed in terms of kilobits, whereas

https://hemanthrajhemu.github.io

that of a parallel interface memory is expressed in terms of kilobytes. Atmel Corporations AT24C512 is an example for serial memory with capacity 512 kilobits and 2-wire interface. Please refer to the section 'Communication Interface' for more details on I2C, SPI and 1-Wire Bus.

2.2.4 Memory Shadowing

Generally the execution of a program or a configuration from a Read Only Memory (ROM) is very slow (120 to 200 ns) compared to the execution from a random access memory (40 to 70 ns). From the timing parameters it is obvious that RAM access is about three times as fast as ROM access. Shadowing of memory is a technique adopted to solve the execution speed problem in processor-based systems. In computer systems and video systems there will be a configuration holding ROM called Basic Input Output Configuration ROM or simply BIOS. In personal computer systems BIOS stores the hardware configuration information like the address assigned for various serial ports and other non-plug '*n*' play devices, etc. Usually it is read and the system is configured according to it during system boot up and it is time consuming. Now the manufactures included a RAM behind the logical layer of BIOS at its same address as a shadow to the BIOS and the first step that happens during the boot up is copying the BIOS to the shadowed RAM and write protecting the RAM then disabling the BIOS reading. You may be thinking that what a stupid idea it is and why both RAM and ROM are needed for holding the same data. The answer is: RAM is volatile and it cannot hold the configuration data which is copied from the BIOS when the power supply is switched off. Only a ROM can hold it permanently. But for high system performance it should be accessed from a RAM instead of accessing from a ROM.

2.2.5 Memory Selection for Embedded Systems

Embedded systems require a program memory for holding the control algorithm (For a super-loop based design) or embedded OS and the applications designed to run on top of it (for OS based designs), data memory for holding variables and temporary data during task execution, and memory for holding non-volatile data (like configuration data, look up table etc) which are modifiable by the application (Unlike program memory, which is non-volatile as well unalterable by the end user). The memory requirement for an embedded system in terms of RAM and ROM (EEPROM/FLASH/NVRAM) is solely dependent on the type of the embedded system and the applications for which it is designed. There is no hard and fast rule for calculating the memory requirements. Lot of factors need to be considered when selecting the type and size of memory for embedded system. For example, if the embedded system is designed using SoC or a microcontroller with on-chip RAM and ROM (FLASH/EEPROM), depending on the application need the on-chip memory may be sufficient for designing the total system. As a rule of thumb, identify your system requirement and based on the type of processor (SoC or microcontroller with on-chip memory) used for the design, take a decision on whether the on-chip memory is sufficient or external memory is required. Let's consider a simple electronic toy design as an example. As the complexity of requirements are less and data memory requirement are minimal, we can think of a microcontroller with a few bytes of internal RAM, a few bytes or kilobytes (depending on the number of tasks and the complexity of tasks) of FLASH memory and a few bytes of EEPROM (if required) for designing the system. Hence there is no need for external memory at all. A PIC microcontroller device which satisfies the I/O and memory requirements can be used in this case. If the embedded design is based on an RTOS, the RTOS requires certain amount of RAM for its execution and ROM for storing the RTOS image (Image is the common name given for the binary data generated by the compilation of all RTOS source files). Normally the binary code for RTOS kernel containing all the services is stored in a non-volatile memory (Like FLASH) as either compressed or non-compressed data. During boot-up of the device,

https://hemanthrajhemu.github.io

the RTOS files are copied from the program storage memory, decompressed if required and then loaded to the RAM for execution. The supplier of the RTOS usually gives a rough estimate on the run time RAM requirements and program memory requirements for the RTOS. On top of this add the RAM requirements for executing user tasks and ROM for storing user applications. On a safer side, always add a buffer value to the total estimated RAM and ROM size requirements. A smart phone device with Windows mobile operating system is a typical example for embedded device with OS. Say 64MB RAM and 128MB ROM are the minimum requirements for running the Windows mobile device, indeed you need extra RAM and ROM for running user applications. So while building the system, count the memory for that also and arrive at a value which is always at the safer side, so that you won't end up in a situation where you don't have sufficient memory to install and run user applications. There are two parameters for representing a memory. The first one is the size of the memory chip (Memory density expressed in terms of number of memory bytes per chip). There is no option to get a memory chip with the exact required number of bytes. Memory chips come in standard sizes like 512bytes, 1024bytes (1 kilobyte), 2048bytes (2 kilobytes), 4Kb,† 8Kb, 16Kb, 32Kb, 64Kb, 128Kb, 256Kb, 512Kb, 1024Kb (1 megabytes), etc. Suppose your embedded application requires only 750 bytes of RAM, you don't have the option of getting a memory chip with size 750 bytes, the only option left with is to choose the memory chip with a size closer to the size needed. Here 1024 bytes is the least possible option. We cannot go for 512 bytes, because the minimum requirement is 750 bytes. While you select a memory size, always keep in mind the address range supported by your processor. For example, for a processor/controller with 16 bit address bus, the maximum number of memory locations that can be addressed is $2^{16} = 65536$ bytes = 64Kb. Hence it is meaningless to select a 128Kb memory chip for a processor with 16bit wide address bus. Also, the entire memory range supported by the processor/controller may not be available to the memory chip alone. It may be shared between I/O, other ICs and memory. Suppose the address bus is 16bit wide and only the lower 32Kb address range is assigned to the memory chip, the memory size maximum required is 32Kb only. It is not worth to use a memory chip with size 64Kb in such a situation. The second parameter that needs to be considered in selecting a memory is the word size of the memory. The word size refers to the number of memory bits that can be read/write together at a time. 4, 8, 12, 16, 24, 32, etc. are the word sizes supported by memory chips. Ensure that the word size supported by the memory chip matches with the data bus width of the processor/controller.

FLASH memory is the popular choice for ROM (program storage memory) in embedded applications. It is a powerful and cost-effective solid-state storage technology for mobile electronics devices and other consumer applications. FLASH memory comes in two major variants, namely, NAND and NOR FLASH. NAND FLASH is a high-density low cost non-volatile storage memory. On the other hand, NOR FLASH is less dense and slightly expensive. But it supports the Execute in Place (XIP) technique for program execution. The XIP technology allows the execution of code memory from ROM itself without the need for copying it to the RAM as in the case of conventional execution method. It is a good practice to use a combination of NOR and NAND memory for storage memory requirements, where NAND can be used for storing the program code and or data like the data captured in a camera device. NAND FLASH doesn't support XIP and if NAND FLASH is used for storing program code, a DRAM can be used for copying and executing the program code. NOR FLASH supports XIP and it can be used as the memory for bootloader or for even storing the complete program code.

The EEPROM data storage memory is available as either serial interface or parallel interface chip. If the processor/controller of the device supports serial interface and the amount of data to write and read to and from the device is less, it is better to have a serial EEPROM chip. The serial EEPROM saves the address space of the total system. The memory capacity of the serial EEPROM is usually expressed in

†Kb—Kilobytes

https://hemanthrajhemu.github.io

bits or kilobits. 512 bits, 1Kbits, 2Kbits, 4Kbits, etc. are examples for serial EEPROM memory representation. For embedded systems with low power requirements like portable devices, choose low power memory devices. Certain embedded devices may be targeted for operating at extreme environmental conditions like high temperature, high humid area, etc. Select an industrial grade memory chip in place of the commercial grade chip for such devices.

2.3 SENSORS AND ACTUATORS

At the very beginning of this chapter it is already mentioned that an embedded system is in constant interaction with the Real world and the controlling/monitoring functions executed by the embedded system is achieved in accordance with the changes happening to the Real world. The changes in system environment or variables are detected by the sensors connected to the input port of the embedded system. If the embedded system is designed for any controlling purpose, the system will produce some changes in the controlling variable to bring the controlled variable to the desired value. It is achieved through an actuator connected to the output port of the embedded system. If the embedded system is designed for monitoring purpose only, then there is no need for including an actuator in the system. For example, take the case of an ECG machine. It is designed to monitor the heart beat status of a patient and it cannot impose a control over the patient's heart beat and its order. The sensors used here are the different electrode sets connected to the body of the patient. The variations are captured and presented to the user (may be a doctor) through a visual display or some printed chart.

2.3.1 Sensors

A sensor is a transducer device that converts energy from one form to another for any measurement or control purpose. This is what I "by-hearted" during my engineering degree from the transducers paper.

If we look back to the "Smart" running shoe example given at the end of Chapter 1, we can identify that the sensor which measures the distance between the cushion and magnet in the smart running shoe is a magnetic hall effect sensor (Please refer back).

2.3.2 Actuators

Actuator is a form of transducer device (mechanical or electrical) which converts signals to corresponding physical action (motion). Actuator acts as an output device.

Looking back to the "Smart" running shoe example given at the end of Chapter 1, we can see that the actuator used for adjusting the position of the cushioning element is a micro stepper motor (Please refer back).

2.3.3 The I/O Subsystem

The I/O subsystem of the embedded system facilitates the interaction of the embedded system with the external world. As mentioned earlier the interaction happens through the sensors and actuators connected to the input and output ports respectively of the embedded system. The sensors may not be directly interfaced to the input ports, instead they may be interfaced through signal conditioning and translating systems like ADC, optocouplers, etc. This section illustrates some of the sensors and actuators used in embedded systems and the I/O systems to facilitate the interaction of embedded systems with external world.

https://hemanthrajhemu.github.io

2.3.3.1 Light Emitting Diode (LED) Light Emitting Diode (LED) is an important output device for visual indication in any embedded system. LED can be used as an indicator for the status of various signals or situations. Typical examples are indicating the presence of power conditions like 'Device ON', 'Battery low' or 'Charging of battery' for a battery operated handheld embedded devices.

Light Emitting Diode is a *p-n* junction diode (Refer Analog Electronics fundamentals to refresh your memory for *p-n* junction diode ☺) and it contains an anode and a cathode. For proper functioning of the LED, the anode of it should be connected to +ve terminal of the supply voltage and cathode to the –ve terminal of supply voltage. The current flowing through the LED must be limited to a value below the maximum current that it can conduct. A resister is used in series between the power supply and the LED to limit the current through the LED. The ideal LED interfacing circuit is shown in Fig. 2.13.

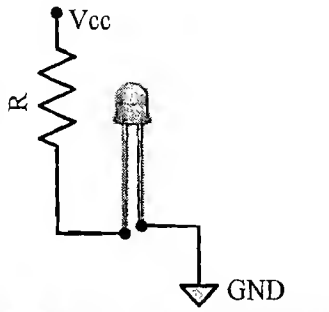


Fig. 2.13 **LED interfacing**

LEDs can be interfaced to the port pin of a processor/controller in two ways. In the first method, the anode is directly connected to the port pin and the port pin drives the LED. In this approach the port pin 'sources' current to the LED when the port pin is at logic High (Logic '1'). In the second method, the cathode of the LED is connected to the port pin of the processor/controller and the anode to the supply voltage through a current limiting resistor. The LED is turned on when the port pin is at logic Low (Logic '0'). Here the port pin 'sinks' current. If the LED is directly connected to the port pin, depending on the maximum current that a port pin can source, the brightness of LED may not be to the required level. In the second approach, the current is directly sourced by the power supply and the port pin acts as the sink for current. Here we will get the required brightness for the LED.

2.3.3.2 7-Segment LED Display The 7-segment LED display is an output device for displaying alpha numeric characters. It contains 8 light-emitting diode (LED) segments arranged in a special form. Out of the 8 LED segments, 7 are used for displaying alpha numeric characters and 1 is used for representing 'decimal point' in decimal number display. Figure 2.14 explains the arrangement of LED segments in a 7-segment LED display.

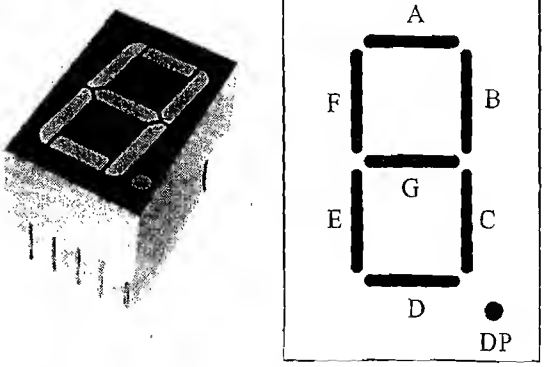


Fig. 2.14 **7-Segment LED Display**

The LED segments are named A to G and the decimal point LED segment is named as DP. The LED segments A to G and DP should be lit accordingly to display numbers and characters. For example, for displaying the number 4, the segments F, G, B and C are lit. For displaying 3, the segments A, B, C, D, G and DP are lit. For displaying the character 'd', the segments B, C, D, E and G are lit. All these 8 LED segments need to be connected to one port of the processor/controller for displaying alpha numeric digits. The 7-segment LED displays are available in two different configurations, namely; Common Anode and Common Cathode. In the common anode configuration, the anodes of the 8 segments are connected commonly whereas in the common cathode configuration, the 8 LED segments share a common cathode line. Figure 2.15 illustrates the Common Anode and Cathode configurations.

Based on the configuration of the 7-segment LED unit, the LED segment's anode or cathode is connected to the port of the processor/controller in the order 'A' segment to the least significant port pin and DP segment to the most significant port pin.

https://hemanthrajhemu.github.io

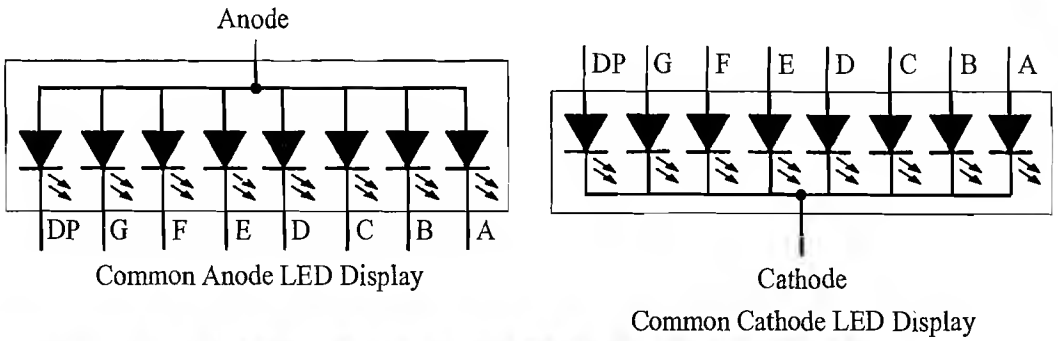


Fig. 2.15 **Common anode and cathode configurations of a 7-segment LED Display**

The current flow through each of the LED segments should be limited to the maximum value supported by the LED display unit. The typical value for the current falls within the range of 20mA. The current through each segment can be limited by connecting a current limiting resistor to the anode or cathode of each segment. The value for the current limiting resistors can be calculated using the current value from the electrical parameter listing of the LED display.

For common cathode configurations, the anode of each LED segment is connected to the port pins of the port to which the display is interfaced. The anode of the common anode LED display is connected to the 5V supply voltage through a current limiting resistor and the cathode of each LED segment is connected to the respective port pin lines. For an LED segment to lit in the Common anode LED configuration, the port pin to which the cathode of the LED segment is connected should be set at logic 0.

7-segment LED display is a popular choice for low cost embedded applications like, Public telephone call monitoring devices, point of sale terminals, etc.

2.3.3.3 Optocoupler Optocoupler is a solid state device to isolate two parts of a circuit. Optocoupler combines an LED and a photo-transistor in a single housing (package). Figure 2.16 illustrates the functioning of an optocoupler device.

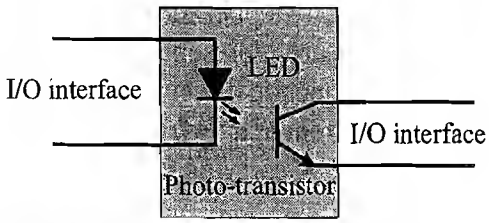


Fig. 2.16 **An optocoupler device**

In electronic circuits, an optocoupler is used for suppressing interference in data communication, circuit isolation, high voltage separation, simultaneous separation and signal intensification, etc. Optocouplers can be used in either input circuits or in output circuits. Figure 2.17 illustrates the usage

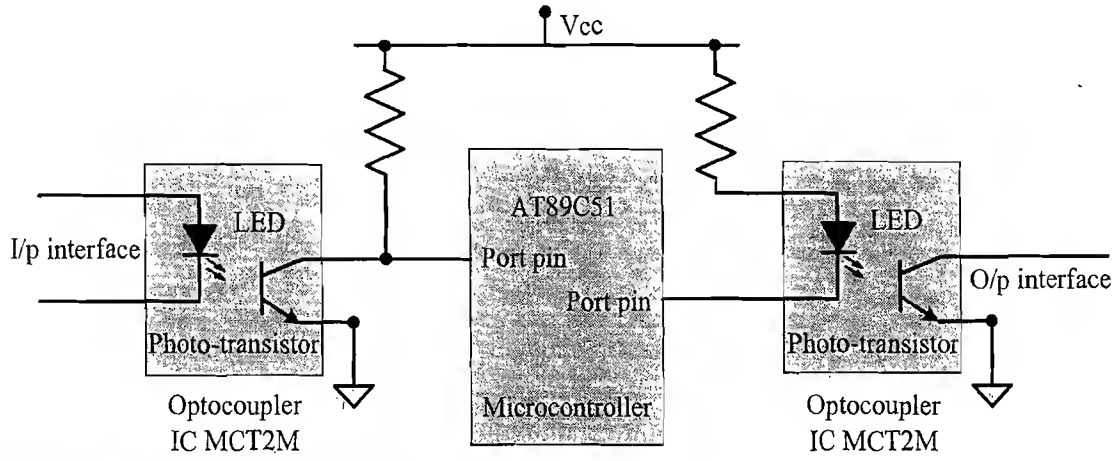


Fig. 2.17 **Optocoupler in Input and Output circuit**

https://hemanthrajhemu.github.io

of optocoupler in input circuit and output circuit of an embedded system with a microcontroller as the system core.

Optocoupler is available as ICs from different semiconductor manufacturers. The MCT2M IC from Fairchild semiconductor (http://www.fairchildsemi.com/) is an example for optocoupler IC.

2.3.3.4 Stepper Motor A stepper motor is an electro-mechanical device which generates discrete displacement (motion) in response to dc electrical signals. It differs from the normal dc motor in its operation. The dc motor produces continuous rotation on applying dc voltage whereas a stepper motor produces discrete rotation in response to the dc voltage applied to it. Stepper motors are widely used in industrial embedded applications, consumer electronic products and robotics control systems. The paper feed mechanism of a printer/fax makes use of stepper motors for its functioning.

Based on the coil winding arrangements, a two-phase stepper motor is classified into two. They are:

1. Unipolar
2. Bipolar

1. Unipolar A unipolar stepper motor contains two windings per phase. The direction of rotation (clockwise or anticlockwise) of a stepper motor is controlled by changing the direction of current flow. Current in one direction flows through one coil and in the opposite direction flows through the other coil. It is easy to shift the direction of rotation by just switching the terminals to which the coils are connected. Figure 2.18 illustrates the working of a two-phase unipolar stepper motor.

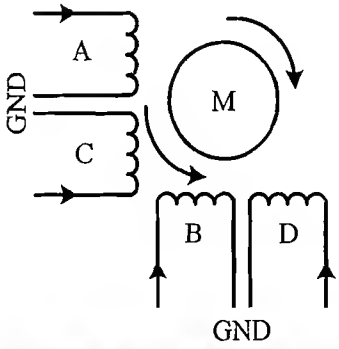


Fig. 2.18 2-Phase unipolar stepper motor

The coils are represented as A, B, C and D. Coils A and C carry current in opposite directions for phase 1 (only one of them will be carrying current at a time). Similarly, B and D carry current in opposite directions for phase 2 (only one of them will be carrying current at a time).

2. Bipolar A bipolar stepper motor contains single winding per phase. For reversing the motor rotation the current flow through the windings is reversed dynamically. It requires complex circuitry for current flow reversal. The stator winding details for a two phase unipolar stepper motor is shown in Fig. 2.19.

The stepping of stepper motor can be implemented in different ways by changing the sequence of activation of the stator windings. The different stepping modes supported by stepper motor are explained below.

Full Step In the full step mode both the phases are energised simultaneously. The coils A, B, C and D are energised in the following order:

Step	Coil A	Coil B	Coil C	Coil D
1	H	H	L	L
2	L	H	H	L
3	L	L	H	H
4	H	L	L	H

It should be noted that out of the two windings, only one winding of a phase is energised at a time.

https://hemanthrajhemu.github.io

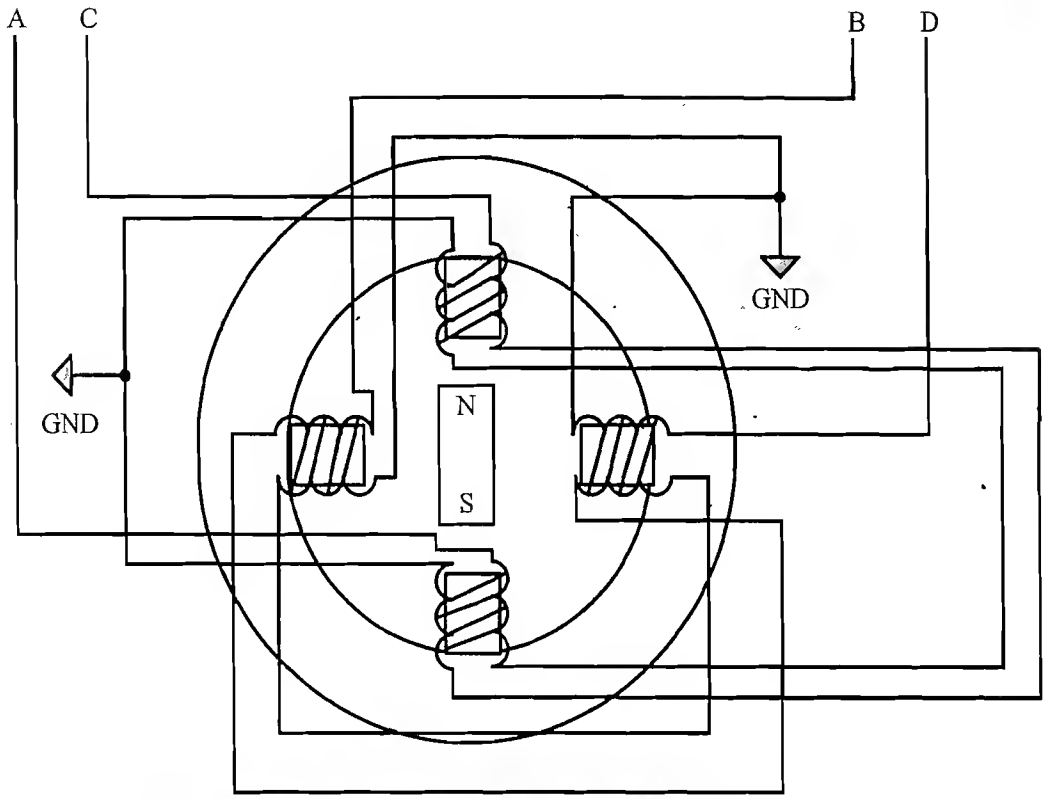


Fig. 2.19 **Stator Winding details for a 2 Phase unipolar stepper motor**

Wave Step In the wave step mode only one phase is energised at a time and each coils of the phase is energised alternatively. The coils A, B, C and D are energised in the following order:

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	L	H	L	L
3	L	L	H	L
4	L	L	L	H

Half Step It uses the combination of wave and full step. It has the highest torque and stability. The coil energising sequence for half step is given below.

Step	Coil A	Coil B	Coil C	Coil D
1	H	L	L	L
2	H	H	L	L
3	L	H	L	L
4	L	H	H	L
5	L	L	H	L
6	L	L	H	H
7	L	L	L	H
8	H	L	L	H

https://hemanthrajhemu.github.io

The rotation of the stepper motor can be reversed by reversing the order in which the coil is energised.

Two-phase unipolar stepper motors are the popular choice for embedded applications. The current requirement for stepper motor is little high and hence the port pins of a microcontroller/processor may not be able to drive them directly. Also the supply voltage required to operate stepper motor varies normally in the range 5V to 24 V. Depending on the current and voltage requirements, special driving circuits are required to interface the stepper motor with microcontroller/processors. Commercial off-the-shelf stepper motor driver ICs are available in the market and they can be directly interfaced to the microcontroller port. ULN2803 is an octal peripheral driver array available from ON semiconductors and ST microelectronics for driving a 5V stepper motor. Simple driving circuit can also be built using transistors.

The following circuit diagram (Fig. 2.20) illustrates the interfacing of a stepper motor through a driver circuit connected to the port pins of a microcontroller/processor.

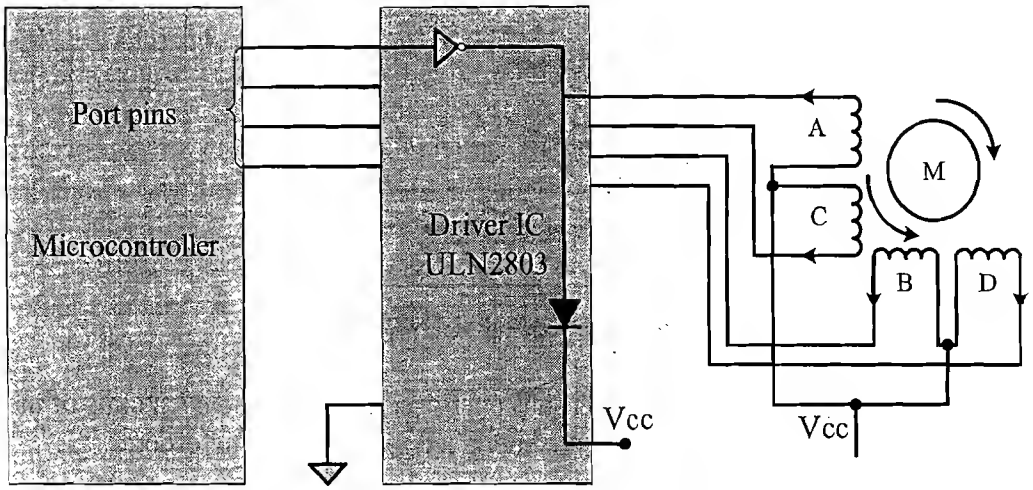
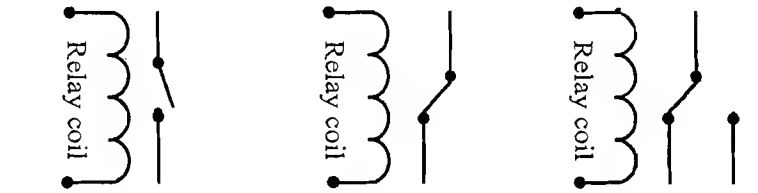


Fig. 2.20 **Interfacing of stepper motor through driver circuit**

2.3.3.5 Relay Relay is an electro-mechanical device. In embedded application, the 'Relay' unit acts as dynamic path selectors for signals and power. The 'Relay' unit contains a relay coil made up of insulated wire on a metal core and a metal armature with one or more contacts.

'Relay' works on electromagnetic principle. When a voltage is applied to the relay coil, current flows through the coil, which in turn generates a magnetic field. The magnetic field attracts the armature core and moves the contact point. The movement of the contact point changes the power/signal flow path. 'Relays' are available in different configurations. Figure 2.21 given below illustrates the widely used relay configurations for embedded applications.



Single pole single throw normally open

Single pole single throw normally closed

Single pole double throw

Fig. 2.21 **Relay configurations**

https://hemanthrajhemu.github.io

The **Single Pole Single Throw** configuration has only one path for information flow. The path is either open or closed in normal condition. For normally Open Single Pole Single Throw relay, the circuit is normally open and it becomes closed when the relay is energised. For normally closed Single Pole Single Throw configuration, the circuit is normally closed and it becomes open when the relay is energised. For Single Pole Double Throw Relay, there are two paths for information flow and they are selected by energising or de-energising the relay.

The Relay is normally controlled using a relay driver circuit connected to the port pin of the processor/controller. A transistor is used for building the relay driver circuit. Figure 2.22 illustrates the same.

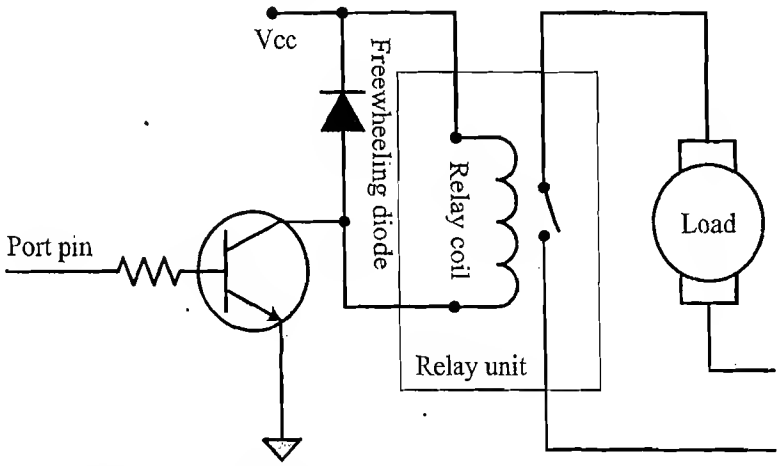


Fig. 2.22 **Transistor based Relay driving circuit**

A free-wheeling diode is used for free-wheeling the voltage produced in the opposite direction when the relay coil is de-energised. The freewheeling diode is essential for protecting the relay and the transistor.

Most of the industrial relays are bulky and requires high voltage to operate. Special relays called 'Reed' relays are available for embedded application requiring switching of low voltage DC signals.

2.3.3.6 Piezo Buzzer Piezo buzzer is a piezoelectric device for generating audio indications in embedded application. A piezoelectric buzzer contains a piezoelectric diaphragm which produces audible sound in response to the voltage applied to it. Piezoelectric buzzers are available in two types. 'Self-driving' and 'External driving'. The 'Self-driving' circuit contains all the necessary components to generate sound at a predefined tone. It will generate a tone on applying the voltage. External driving piezo buzzers supports the generation of different tones. The tone can be varied by applying a variable pulse train to the piezoelectric buzzer. A piezo buzzer can be directly interfaced to the port pin of the processor/control. Depending on the driving current requirements, the piezo buzzer can also be interfaced using a transistor based driver circuit as in the case of a 'Relay'.

2.3.3.7 Push Button Switch It is an input device. Push button switch comes in two configurations, namely 'Push to Make' and 'Push to Break'. In the 'Push to Make' configuration, the switch is normally in the open state and it makes a circuit contact when it is pushed or pressed. In the 'Push to Break' configuration, the switch is normally in the closed state and it breaks the circuit contact when it is pushed or pressed. The push button stays in the 'closed' (For Push to Make type) or 'open' (For Push to Break type) state as long as it is kept in the pushed state and it breaks/makes the circuit connection when it

https://hemanthrajhemu.github.io

is released. Push button is used for generating a momentary pulse. In embedded application push button is generally used as reset and start switch and pulse generator. The Push button is normally connected to the port pin of the host processor/controller. Depending on the way in which the push button interfaced to the controller, it can generate either a 'HIGH' pulse or a 'LOW' pulse. Figure 2.23 illustrates how the push button can be used for generating 'LOW' and 'HIGH' pulses.

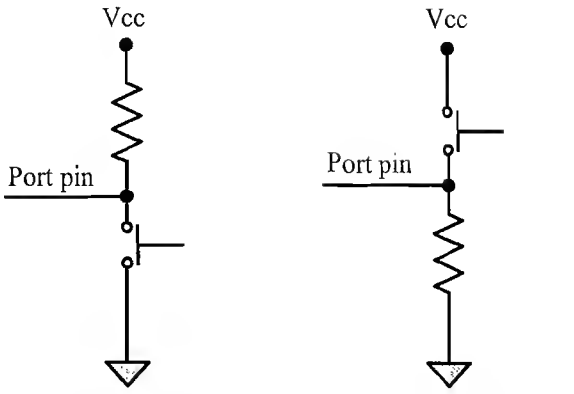


Fig. 2.23 **Push button switch configurations**

2.3.3.8 Keyboard Keyboard is an input device for user interfacing. If the number of keys required is very limited, push button switches can be used and they can be directly interfaced to the port pins for reading. However, there may be situations demanding a large number of keys for user input (e.g. PDA device with alpha-numeric keypad for user data entry). In such situations it may not be possible to interface each keys to a port pin due to the limitation in the number of general purpose port pins available for the processor/controller in use and moreover it is wastage of port pins. Matrix keyboard is an optimum solution for handling large key requirements. It greatly reduces the number of interface connections. For example, for interfacing 16 keys, in the direct interfacing technique 16 port pins are required, whereas in the matrix keyboard only 8 lines are required. The 16 keys are arranged in a 4 column × 4 Row matrix. Figure 2.24 illustrates the connection of keys in a matrix keyboard.

In a matrix keyboard, the keys are arranged in matrix fashion (i.e. they are connected in a row and column style). For detecting a key press, the keyboard uses the scanning technique, where each row of the matrix is pulled low and the columns are read. After reading the status of each columns corresponding to a row, the row is pulled high and the next row is pulled low and the status of the columns are read. This process is repeated until the scanning for all rows are completed. When a row is pulled low and if a key connected to the row is pressed, reading the column to which the key is connected will give logic 0. Since keys are mechanical devices, there is a possibility for de-bounce issues, which may give multiple key press effect for a single key press. To prevent this, a proper key de-bouncing technique should be applied. Hardware key de-bouncer circuits and software key de-bounce techniques are the key de-bouncing techniques available. The software key de-bouncing technique doesn't require any additional hardware and is easy to implement. In the software de-bouncing technique, on detecting a key-press, the key is read again after a de-bounce delay. If the key press is a genuine one, the state of the key will remain as 'pressed' on the second read also. Pull-up resistors are connected to the column lines to limit the current that flows to the Row line on a key press.

2.3.3.9 Programmable Peripheral Interface (PPI) Programmable Peripheral Interface (PPI) devices are used for extending the I/O capabilities of processors/controllers. Most of the processors/controllers provide very limited number of I/O and data ports and at times it may require more number of I/O ports than the one supported by the controller/processor. A programmable peripheral interface device expands the I/O capabilities of the processor/controller. *8255A* is a popular PPI device for 8bit processors/controllers. *8255A* supports 24 I/O pins and these I/O pins can be grouped as either three 8-bit parallel ports (Port A, Port B and Port C) or two 8bit parallel ports (Port A and Port B) with Port C in any one of the following configurations:

https://hemanthrajhemu.github.io

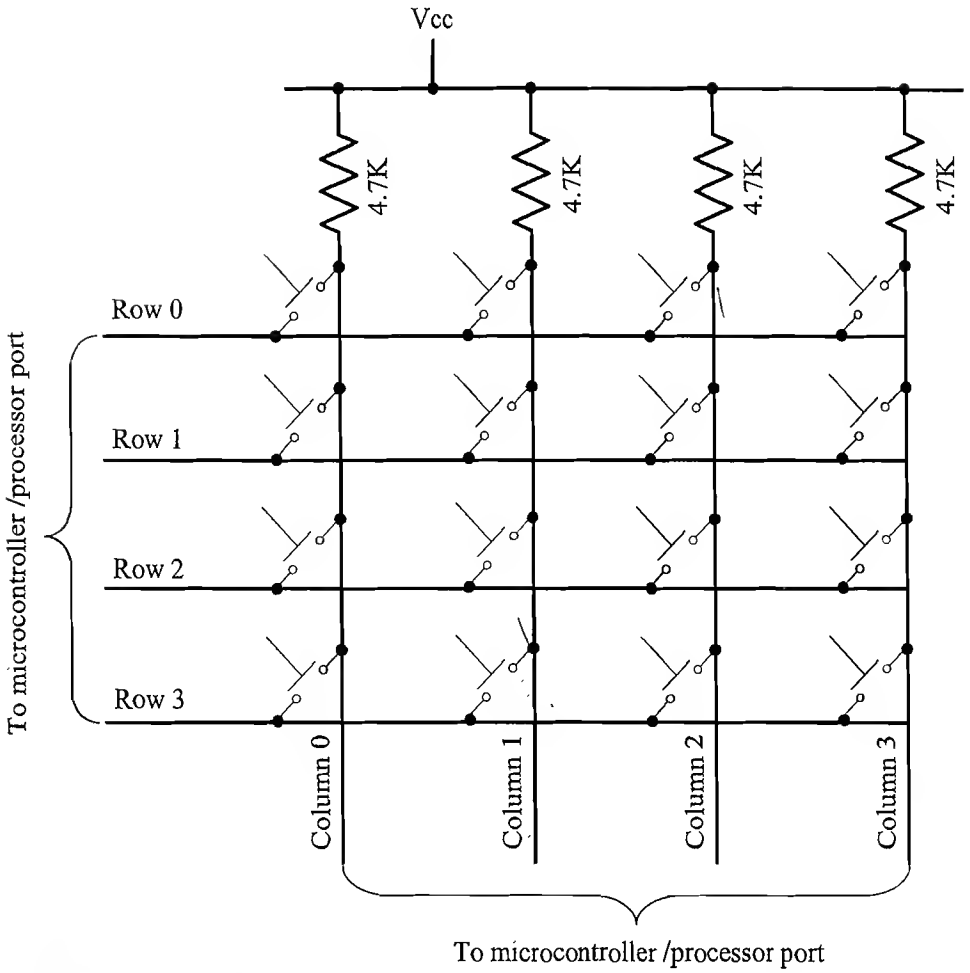


Fig. 2.24 Matrix keyboard Interfacing

1. As 8 individual I/O pins
2. Two 4bit ports namely Port C_{UPPER} (C_U) and Port C_{LOWER} (C_L)

This is configured by manipulating the control register of *8255A*. The control register holds the configuration for Port A, Port B and Port C. The bit details of control register is given below:

D7	D6	D5	D4	D3	D2	D1	D0

The table given below explains the meaning and use of each bit.

Bit	Description
D0	Port C Lower (C_L) I/O mode selector D0 = 1; Sets C_L as input port D0 = 0; Sets C_L as output port
D1	Port B I/O mode selector D1 = 1; Sets port B as input port D1 = 0; Sets port B as output port

https://hemanthrajhemu.github.io

D2	Mode selector for port C lower and port B D2 = 0; Mode 0 – Port B functions as 8bit I/O Port. Port C lower functions as 4bit port. D2 = 1; Mode 1 – Handshake mode. Port B uses 3 bits of Port C as handshake signals
D3	Port C Upper (C_U) I/O mode selector D3 = 1; Sets C_U as input port D3 = 0; Sets C_U as output port
D4	Port A I/O mode selector D4 = 1; Sets Port A as input port D4 = 0; Sets Port A as output port
D5, D6	Mode selector for port C upper and port A D6 D5 = 00; Mode 0 – Simple I/O mode D6 D5 = 01; Mode 1 – Handshake mode. Port A uses 3 bits of Port C as handshake signals D6 D5 = 1X; Mode 2. X can be 0 or 1 – Port A functions as bi-directional port
D7	Control/Data mode selector for port C D7 = 1; I/O mode. D7 = 0; Bit set/reset (BSR) mode. Functions as the control/status lines for ports A and B. The bits of port C can be set or reset just as if they were output ports.

Please refer to the *8255A* datasheet available at http://www.intersil.com/data/fn/fn2969.pdf for more details about the different operating modes of 8255.

Figure 2.25 illustrates the generic interfacing of a *8255A* device with an 8bit processor/controller with 16bit address bus (Lower order Address bus is multiplexed with data bus).

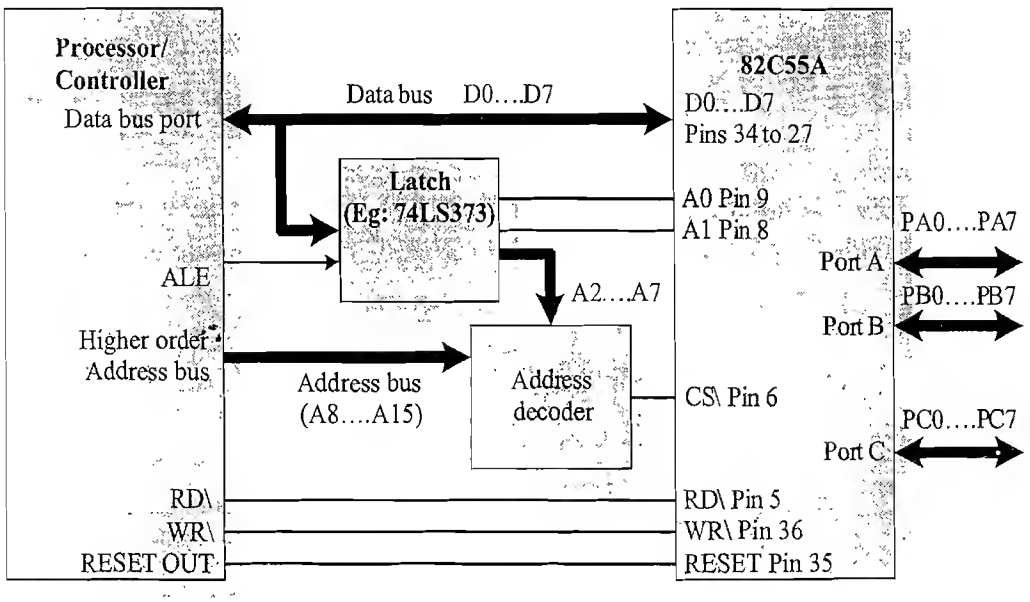


Fig. 2.25 **Interfacing of *8255* with an 8 bit microcontroller**

The ports of *8255* can be configured for different modes of operation by the processor/controller.

https://hemanthrajhemu.github.io

2.4 COMMUNICATION INTERFACE

Communication interface is essential for communicating with various subsystems of the embedded system and with the external world. For an embedded product, the communication interface can be viewed in two different perspectives; namely; Device/board level communication interface (Onboard Communication Interface) and Product level communication interface (External Communication Interface). Embedded product is a combination of different types of components (chips/devices) arranged on a printed circuit board (PCB). The communication channel which interconnects the various components within an embedded product is referred as device/board level communication interface (onboard communication interface). Serial interfaces like I2C, SPI, UART, 1-Wire, etc and parallel bus interface are examples of 'Onboard Communication Interface'.

Some embedded systems are self-contained units and they don't require any interaction and data transfer with other sub-systems or external world. On the other hand, certain embedded systems may be a part of a large distributed system and they require interaction and data transfer between various devices and sub-modules. The 'Product level communication interface' (External Communication Interface) is responsible for data transfer between the embedded system and other devices or modules. The external communication interface can be either a wired media or a wireless media and it can be a serial or a parallel interface. Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves (RF), GPRS, etc. are examples for wireless communication interface. RS-232C/RS-422/RS-485, USB, Ethernet IEEE 1394 port, Parallel port, CF-II interface, SDIO, PCMCIA, etc. are examples for wired interfaces. It is not mandatory that an embedded system should contain an external communication interface. Mobile communication equipment is an example for embedded system with external communication interface.

The following section gives you an overview of the various 'Onboard' and 'External' communication interfaces for an embedded product. We will discuss about the various physical interface, firmware requirements and initialisation and communication sequence for these interfaces in a dedicated book titled '*Device Interfacing*', which is planned under this series.

2.4.1 Onboard Communication Interfaces

Onboard Communication Interface refers to the different communication channels/buses for interconnecting the various integrated circuits and other peripherals within the embedded system. The following section gives an overview of the various interfaces for onboard communication.

2.4.1.1 Inter Integrated Circuit (I2C) Bus The Inter Integrated Circuit Bus (I2C–Pronounced 'I square C') is a synchronous bi-directional half duplex (one-directional communication at a given point of time) two wire serial interface bus. The concept of I2C bus was developed by 'Philips semiconductors' in the early 1980s. The original intention of I2C was to provide an easy way of connection between a microprocessor/microcontroller system and the peripheral chips in television sets. The I2C bus comprise of two bus lines, namely; Serial Clock–SCL and Serial Data–SDA. SCL line is responsible for generating synchronisation clock pulses and SDA is responsible for transmitting the serial data across devices. I2C bus is a shared bus system to which many number of I2C devices can be connected. Devices connected to the I2C bus can act as either 'Master' device or 'Slave' device. The 'Master' device is responsible for controlling the communication by initiating/terminating data transfer, sending data and generating necessary synchronisation clock pulses. 'Slave' devices wait for the commands

https://hemanthrajhemu.github.io

from the master and respond upon receiving the commands. 'Master' and 'Slave' devices can act as either transmitter or receiver. Regardless whether a master is acting as transmitter or receiver, the synchronisation clock signal is generated by the 'Master' device only. I2C supports multi masters on the same bus. The following bus interface diagram shown in Fig. 2.26 illustrates the connection of master and slave devices on the I2C bus.

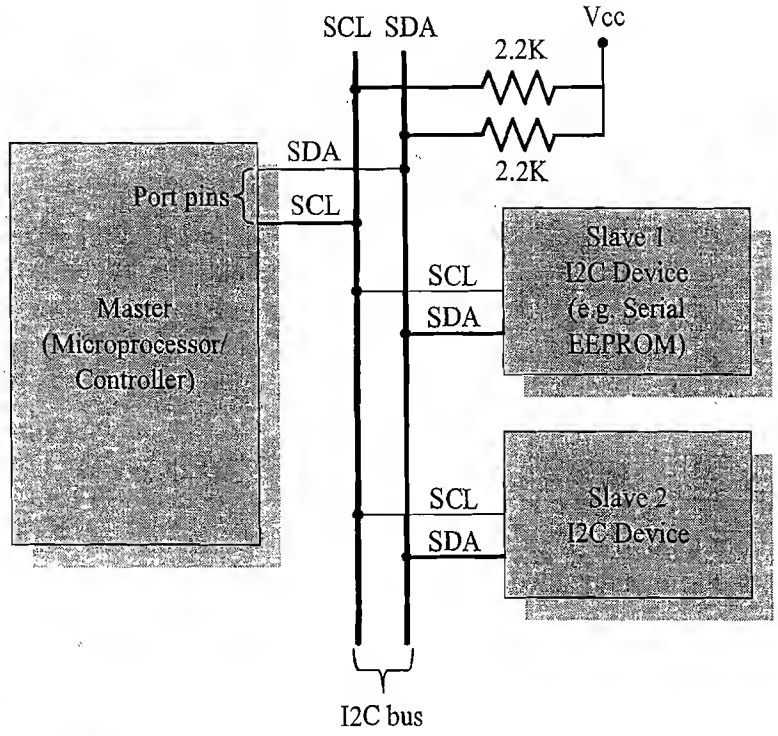


Fig. 2.26 **I2C Bus Interfacing**

The I2C bus interface is built around an input buffer and an open drain or collector transistor. When the bus is in the idle state, the open drain/collector transistor will be in the floating state and the output lines (SDA and SCL) switch to the 'High Impedance' state. For proper operation of the bus, both the bus lines should be pulled to the supply voltage (+5V for TTL family and +3.3V for CMOS family devices) using pull-up resistors. The typical value of resistors used in pull-up is 2.2K. With pull-up resistors, the output lines of the bus in the idle state will be 'HIGH'.

The address of a I2C device is assigned by hardwiring the address lines of the device to the desired logic level. The address to various I2C devices in an embedded device is assigned and hardwired at the time of designing the embedded hardware. The sequence of operations for communicating with an I2C slave device is listed below:

1. The master device pulls the clock line (SCL) of the bus to 'HIGH'
2. The master device pulls the data line (SDA) 'LOW', when the SCL line is at logic 'HIGH' (This is the 'Start' condition for data transfer)
3. The master device sends the address (7 bit or 10 bit wide) of the 'slave' device to which it wants to communicate, over the SDA line. Clock pulses are generated at the SCL line for synchronising the bit reception by the slave device. The MSB of the data is always transmitted first. The data in the bus is valid during the 'HIGH' period of the clock signal

https://hemanthrajhemu.github.io

4. The master device sends the Read or Write bit (Bit value = 1 Read operation; Bit value = 0 Write operation) according to the requirement
5. The master device waits for the acknowledgement bit from the slave device whose address is sent on the bus along with the Read/Write operation command. Slave devices connected to the bus compares the address received with the address assigned to them
6. The slave device with the address requested by the master device responds by sending an acknowledge bit (Bit value = 1) over the SDA line
7. Upon receiving the acknowledge bit, the Master device sends the 8bit data to the slave device over SDA line, if the requested operation is 'Write to device'. If the requested operation is 'Read from device', the slave device sends data to the master over the SDA line
8. The master device waits for the acknowledgement bit from the device upon byte transfer complete for a write operation and sends an acknowledge bit to the Slave device for a read operation
9. The master device terminates the transfer by pulling the SDA line 'HIGH' when the clock line SCL is at logic 'HIGH' (Indicating the 'STOP' condition)

I2C bus supports three different data rates. They are: Standard mode (Data rate up to 100kbits/sec (100 kbps)), Fast mode (Data rate up to 400kbits/sec (400 kbps)) and High speed mode (Data rate up to 3.4Mbits/sec (3.4 Mbps)). The first generation I2C devices were designed to support data rates only up to 100kbps. The new generation I2C devices are designed to operate at data rates up to 3.4Mbits/sec.

2.4.1.2 Serial Peripheral Interface (SPI) Bus The Serial Peripheral Interface Bus (SPI) is a synchronous bi-directional full duplex four-wire serial interface bus. The concept of SPI was introduced by Motorola. SPI is a single master multi-slave system. It is possible to have a system where more than one SPI device can be master, provided the condition only one master device is active at any given point of time, is satisfied. SPI requires four signal lines for communication. They are:

Master Out Slave In (MOSI):	Signal line carrying the data from master to slave device. It is also known as Slave Input/Slave Data In (SI/SDI)
Master In Slave Out (MISO):	Signal line carrying the data from slave to master device. It is also known as Slave Output (SO/SDO)
Serial Clock (SCLK):	Signal line carrying the clock signals
Slave Select (SS):	Signal line for slave device select. It is an active low signal

The bus interface diagram shown in Fig. 2.27 illustrates the connection of master and slave devices on the SPI bus.

The master device is responsible for generating the clock signal. It selects the required slave device by asserting the corresponding slave device's slave select signal 'LOW'. The data out line (MISO) of all the slave devices when not selected floats at high impedance state.

The serial data transmission through SPI bus is fully configurable. SPI devices contain a certain set of registers for holding these configurations. The serial peripheral control register holds the various configuration parameters like master/slave selection for the device, baudrate selection for communication, clock signal control, etc. The status register holds the status of various conditions for transmission and reception.

SPI works on the principle of 'Shift Register'. The master and slave devices contain a special shift register for the data to transmit or receive. The size of the shift register is device dependent. Normally it is a multiple of 8. During transmission from the master to slave, the data in the master's shift register is shifted out to the MOSI pin and it enters the shift register of the slave device through the MOSI pin of the slave device. At the same time the shifted out data bit from the slave device's shift register enters

https://hemanthrajhemu.github.io

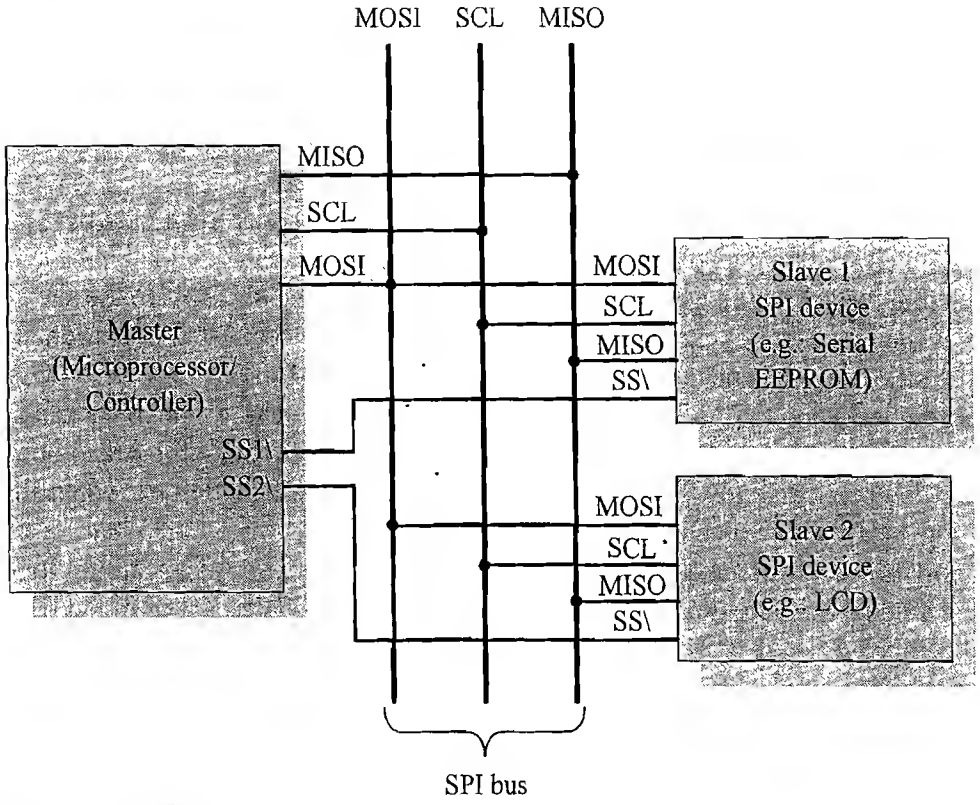


Fig. 2.27 **SPI bus interfacing**

the shift register of the master device through MISO pin. In summary, the shift registers of 'master' and 'slave' devices form a circular buffer. For some devices, the decision on whether the LS/MS bit of data needs to be sent out first is configurable through configuration register (e.g. LSBF bit of the SPI control register for Motorola's 68HC12 controller).

When compared to I2C, SPI bus is most suitable for applications requiring transfer of data in 'streams'. The only limitation is SPI doesn't support an acknowledgement mechanism.

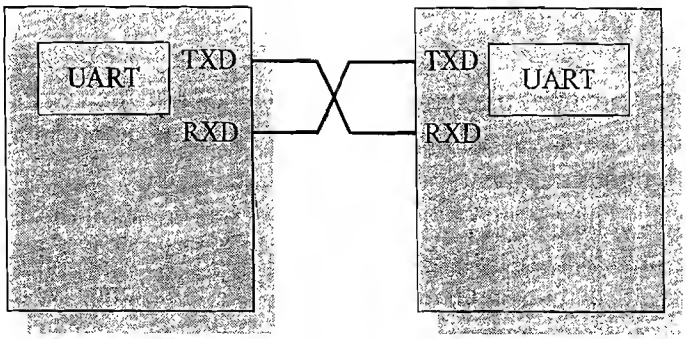
2.4.1.3 Universal Asynchronous Receiver Transmitter (UART) Universal Asynchronous Receiver Transmitter (UART) based data transmission is an asynchronous form of serial data transmission. UART based serial data transmission doesn't require a clock signal to synchronise the transmitting end and receiving end for transmission. Instead it relies upon the pre-defined agreement between the transmitting device and receiving device. The serial communication settings (Baudrate, number of bits per byte, parity, number of start bits and stop bit and flow control) for both transmitter and receiver should be set as identical. The start and stop of communication is indicated through inserting special bits in the data stream. While sending a byte of data, a start bit is added first and a stop bit is added at the end of the bit stream. The least significant bit of the data byte follows the 'start' bit.

The 'start' bit informs the receiver that a data byte is about to arrive. The receiver device starts polling its 'receive line' as per the baudrate settings. If the baudrate is 'x' bits per second, the time slot available for one bit is $1/x$ seconds. The receiver unit polls the receiver line at exactly half of the time slot available for the bit. If parity is enabled for communication, the UART of the transmitting device adds a parity bit (bit value is 1 for odd number of 1s in the transmitted bit stream and 0 for even number of 1s). The UART of the receiving device calculates the parity of the bits received and compares it with the received parity bit for error checking. The UART of the receiving device discards the 'Start', 'Stop' and 'Parity'

https://hemanthrajhemu.github.io

bit from the received bit stream and converts the received serial bit data to a word (In the case of 8 bits/byte, the byte is formed with the received 8 bits with the first received bit as the LSB and last received data bit as MSB).

For proper communication, the 'Transmit line' of the sending device should be connected to the 'Receive line' of the receiving device. Figure 2.28 illustrates the same.



TXD: Transmitter line
RXD: Receiver line

Fig. 2.28 **UART Interfacing**

In addition to the serial data transmission function, UART provides hardware handshaking signal support for controlling the serial data flow. UART chips are available from different semiconductor manufacturers. National Semiconductor's 8250 UART chip is considered as the standard setting UART. It was used in the original IBM PC.

Nowadays most of the microprocessors/controllers are available with integrated UART functionality and they provide built-in instruction support for serial data transmission and reception.

2.4.1.4 1-Wire Interface 1-wire interface is an asynchronous half-duplex communication protocol developed by Maxim Dallas Semiconductor (http://www.maxim-ic.com). It is also known as **Dallas 1-Wire® protocol**. It makes use of only a single signal line (wire) called DQ for communication and follows the master-slave communication model. One of the key feature of 1-wire bus is that it allows power to be sent along the signal wire as well. The 12C slave devices incorporate internal capacitor (typically of the order of 800 pF) to power the device from the signal line. The 1-wire interface supports a single master and one or more slave devices on the bus. The bus interface diagram shown in Fig. 2.29 illustrates the connection of master and slave devices on the 1-wire bus.

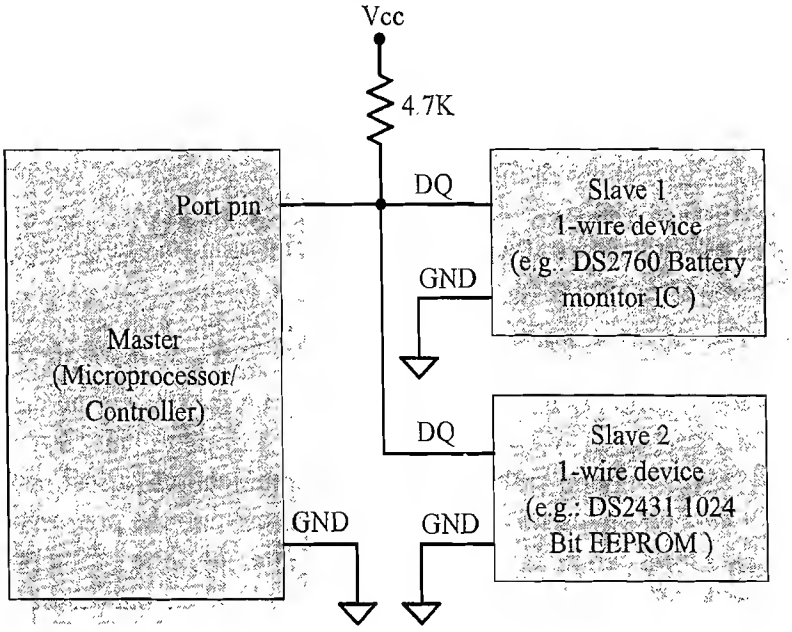


Fig. 2.29 **1-Wire Interface bus**

https://hemanthrajhemu.github.io

Every 1-wire device contains a globally unique 64bit identification number stored within it. This unique identification number can be used for addressing individual devices present on the bus in case there are multiple slave devices connected to the 1-wire bus. The identifier has three parts: an 8bit family code, a 48bit serial number and an 8bit CRC computed from the first 56 bits. The sequence of operation for communicating with a 1-wire slave device is listed below.

1. The master device sends a 'Reset' pulse on the 1-wire bus.
2. The slave device(s) present on the bus respond with a 'Presence' pulse.
3. The master device sends a ROM command (Net Address Command followed by the 64bit address of the device). This addresses the slave device(s) to which it wants to initiate a communication.
4. The master device sends a read/write function command to read/write the internal memory or register of the slave device.
5. The master initiates a Read data/Write data from the device or to the device

All communication over the 1-wire bus is master initiated. The communication over the 1-wire bus is divided into timeslots of 60 microseconds. The 'Reset' pulse occupies 8 time slots. For starting a communication, the master asserts the reset pulse by pulling the 1-wire bus 'LOW' for at least 8 time slots (480µs). If a 'slave' device is present on the bus and is ready for communication it should respond to the master with a 'Presence' pulse, within 60µs of the release of the 'Reset' pulse by the master. The slave device(s) responds with a 'Presence' pulse by pulling the 1-wire bus 'LOW' for a minimum of 1 time slot (60µs). For writing a bit value of 1 on the 1-wire bus, the bus master pulls the bus for 1 to 15µs and then releases the bus for the rest of the time slot. A bit value of '0' is written on the bus by master pulling the bus for a minimum of 1 time slot (60µs) and a maximum of 2 time slots (120µs). To Read a bit from the slave device, the master pulls the bus 'LOW' for 1 to 15µs. If the slave wants to send a bit value '1' in response to the read request from the master, it simply releases the bus for the rest of the time slot. If the slave wants to send a bit value '0', it pulls the bus 'LOW' for the rest of the time slot.

2.4.1.5 Parallel Interface The on-board parallel interface is normally used for communicating with peripheral devices which are memory mapped to the host of the system. The host processor/controller of the embedded system contains a parallel bus and the device which supports parallel bus can directly connect to this bus system. The communication through the parallel bus is controlled by the control signal interface between the device and the host. The 'Control Signals' for communication includes 'Read/Write' signal and device select signal. The device normally contains a device select line and the device becomes active only when this line is asserted by the host processor. The direction of data transfer (Host to Device or Device to Host) can be controlled through the control signal lines for 'Read' and 'Write'. Only the host processor has control over the 'Read' and 'Write' control signals. The device is normally memory mapped to the host processor and a range of address is assigned to it. An address decoder circuit is used for generating the chip select signal for the device. When the address selected by the processor is within the range assigned for the device, the decoder circuit activates the chip select line and thereby the device becomes active. The processor then can read or write from or to the device by asserting the corresponding control line (RD\ and WR\ respectively). Strict timing characteristics are followed for parallel communication. As mentioned earlier, parallel communication is host processor initiated. If a device wants to initiate the communication, it can inform the same to the processor through interrupts. For this, the interrupt line of the device is connected to the interrupt line of the processor and the corresponding interrupt is enabled in the host processor. The width of the parallel interface is determined by the data bus width of the host processor. It can be 4bit, 8bit, 16bit, 32bit or 64bit etc. The bus width supported by the device should be same as that of the host processor. The bus interface diagram shown in Fig. 2.30 illustrates the interfacing of devices through parallel interface.

https://hemanthrajhemu.github.io

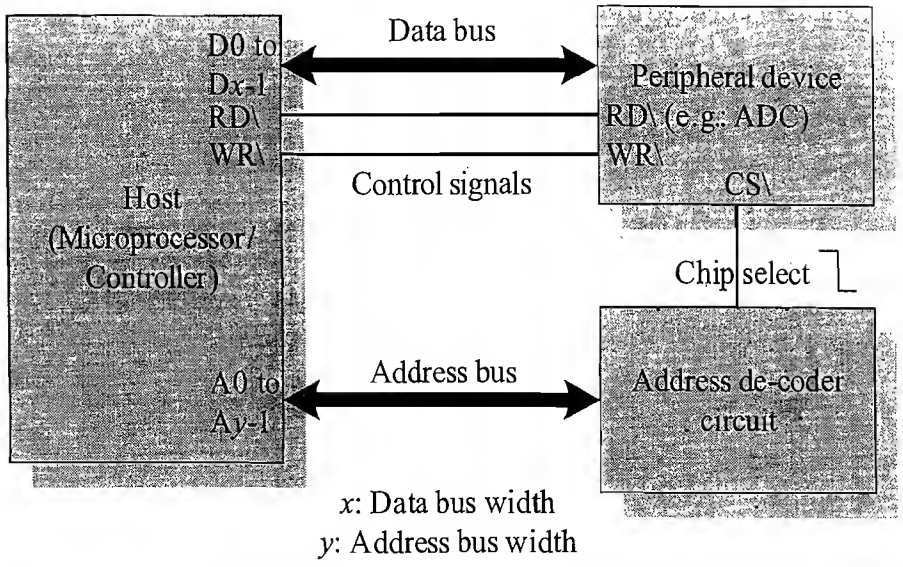


Fig. 2.30 **Parallel Interface Bus**

Parallel data communication offers the highest speed for data transfer.

2.4.2 External Communication Interfaces

The External Communication Interface refers to the different communication channels/buses used by the embedded system to communicate with the external world. The following section gives an overview of the various interfaces for external communication.

2.4.2.1 RS-232 C & RS-485 RS-232 C (Recommended Standard number 232, revision C from the Electronic Industry Association) is a legacy, full duplex, wired, asynchronous serial communication interface. The RS-232 interface is developed by the Electronics Industries Association (EIA) during the early 1960s. RS-232 extends the UART communication signals for external data communication.

UART uses the standard TTL/CMOS logic (Logic 'High' corresponds to bit value 1 and Logic 'Low' corresponds to bit value 0) for bit transmission whereas RS-232 follows the EIA standard for bit transmission. As per the EIA standard, a logic '0' is represented with voltage between +3 and +25V and a logic '1' is represented with voltage between -3 and -25V. In EIA standard, logic '0' is known as 'Space' and logic '1' as 'Mark'. The RS-232 interface defines various handshaking and control signals for communication apart from the 'Transmit' and 'Receive' signal lines for data communication. RS-232 supports two different types of connectors, namely; DB-9: 9-Pin connector and DB-25: 25-Pin connector. Figure 2.31 illustrates the connector details for DB-9 and DB-25.

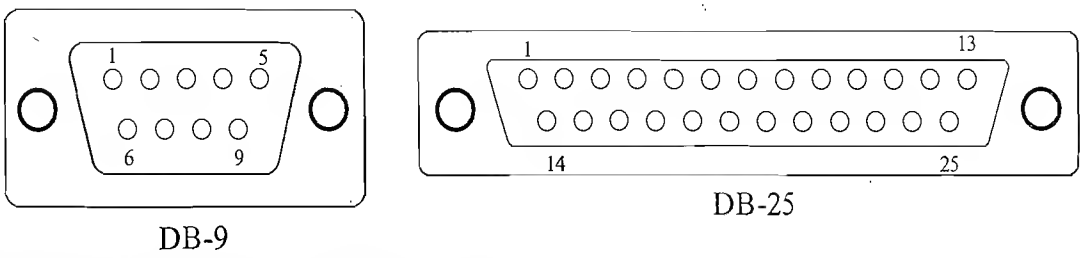


Fig. 2.31 **DB-9 and DB-25 RS-232 Connector Interface**

https://hemanthrajhemu.github.io

The pin details for the two connectors are explained in the following table:

Pin Name	Pin no: (For DB-9 Connector)	Pin no: (For DB-25 Connector)	Description
TXD	3	2	Transmit Pin for Transmitting Serial Data
RXD	2	3	Receive Pin for Receiving Serial Data
RTS	7	4	Request to send.
CTS	8	5	Clear To Send
DSR	6	6	Data Set Ready
GND	5	7	Signal Ground
DCD	1	8	Data Carrier Detect
DTR	4	20	Data Terminal Ready
RI	9	22	Ring Indicator
FG		1	Frame Ground
SDCD		12	Secondary DCD
SCTS		13	Secondary CTS
STXD		14	Secondary TXD
TC		15	Transmission Signal Element Timing
SRXD		16	Secondary RXD
RC		17	Receiver Signal Element Timing
SRTS		19	Secondary RTS
SQ		21	Signal Quality detector
NC		9	No Connection
NC		10	No Connection
NC		11	No Connection
NC		18	No Connection
NC		23	No Connection
NC		24	No Connection
NC		25	No Connection

RS-232 is a point-to-point communication interface and the devices involved in RS-232 communication are called 'Data Terminal Equipment (DTE)' and 'Data Communication Equipment (DCE)'. If no data flow control is required, only TXD and RXD signal lines and ground line (GND) are required for data transmission and reception. The RXD pin of DCE should be connected to the TXD pin of DTE and vice versa for proper data transmission.

If hardware data flow control is required for serial transmission, various control signal lines of the RS-232 connection are used appropriately. The control signals are implemented mainly for modem communication and some of them may not be relevant for other type of devices. The Request To Send (RTS) and Clear To Send (CTS) signals co-ordinate the communication between DTE and DCE. Whenever the DTE has a data to send, it activates the RTS line and if the DCE is ready to accept the data, it activates the CTS line.

https://hemanthrajhemu.github.io

The Data Terminal Ready (DTR) signal is activated by DTE when it is ready to accept data. The Data Set Ready (DSR) is activated by DCE when it is ready for establishing a communication link. DTR should be in the activated state before the activation of DSR.

The Data Carrier Detect (DCD) control signal is used by the DCE to indicate the DTE that a good signal is being received.

Ring Indicator (RI) is a modem specific signal line for indicating an incoming call on the telephone line.

The 25 pin DB connector contains two sets of signal lines for transmit, receive and control lines. Nowadays DB-25 connector is obsolete and most of the desktop systems are available with DB-9 connectors only.

As per the EIA standard RS-232 C supports baudrates up to 20Kbps (Upper limit 19.2 Kbps) The commonly used baudrates by devices are 300bps, 1200bps, 2400bps, 9600bps, 11.52Kbps and 19.2Kbps. 9600 is the popular baudrate setting used for PC communication. The maximum operating distance supported by RS-232 is 50 feet at the highest supported baudrate.

Embedded devices contain a UART for serial communication and they generate signal levels conforming to TTL/CMOS logic. A level translator IC like MAX 232 from Maxim Dallas semiconductor is used for converting the signal lines from the UART to RS-232 signal lines for communication. On the receiving side the received data is converted back to digital logic level by a converter IC. Converter chips contain converters for both transmitter and receiver.

Though RS-232 was the most popular communication interface during the olden days, the advent of other communication techniques like Bluetooth, USB, Firewire, etc are pushing down RS-232 from the scenes. Still RS-232 is popular in certain legacy industrial applications.

RS-232 supports only point-to-point communication and not suitable for multi-drop communication. It uses single ended data transfer technique for signal transmission and thereby more susceptible to noise and it greatly reduces the operating distance.

RS-422 is another serial interface standard from EIA for differential data communication. It supports data rates up to 100Kbps and distance up to 400 ft. The same RS-232 connector is used at the device end and an RS-232 to RS-422 converter is plugged in the transmission line. At the receiver end the conversion from RS-422 to RS-232 is performed. RS-422 supports multi-drop communication with one transmitter device and receiver devices up to 10.

RS-485 is the enhanced version of RS-422 and it supports multi-drop communication with up to 32 transmitting devices (drivers) and 32 receiving devices on the bus. The communication between devices in the bus uses the 'addressing' mechanism to identify slave devices.

2.4.2.2 Universal Serial Bus (USB) Universal Serial Bus (USB) is a wired high speed serial bus for data communication. The first version of USB (USB1.0) was released in 1995 and was created by the USB core group members consisting of Intel, Microsoft, IBM, Compaq, Digital and Northern Telecom. The USB communication system follows a star topology with a USB host at the centre and one or more USB peripheral devices/USB hosts connected to it. A USB host can support connections up to 127, including slave peripheral devices and other USB hosts. Figure 2.32 illustrates the star topology for USB device connection.

USB transmits data in packet format. Each data packet has a standard format. The USB communication is a host initiated one. The USB host contains a host controller which is responsible for controlling the data communication, including establishing connectivity with USB slave devices, packetizing and formatting the data. There are different standards for implementing the USB Host Control interface; namely Open Host Control Interface (OHCI) and Universal Host Control Interface (UHCI).

https://hemanthrajhemu.github.io

The physical connection between a USB peripheral device and master device is established with a USB cable. The USB cable supports communication distance of up to 5 metres. The USB standard uses two different types of connector at the ends of the USB cable for connecting the USB peripheral device and host device. 'Type A' connector is used for upstream connection (connection with host) and Type B connector is used for downstream connection (connection with slave device). The USB connector present in desktop PCs or laptops are examples for 'Type A' USB connector. Both Type A and Type B connectors contain 4 pins for communication. The Pin details for the connectors are listed in the table given below.

Pin no:	Pin name	Description
1	V_{BUS}	Carries power (5V)
2	D–	Differential data carrier line
3	D+	Differential data carrier line
4	GND	Ground signal line

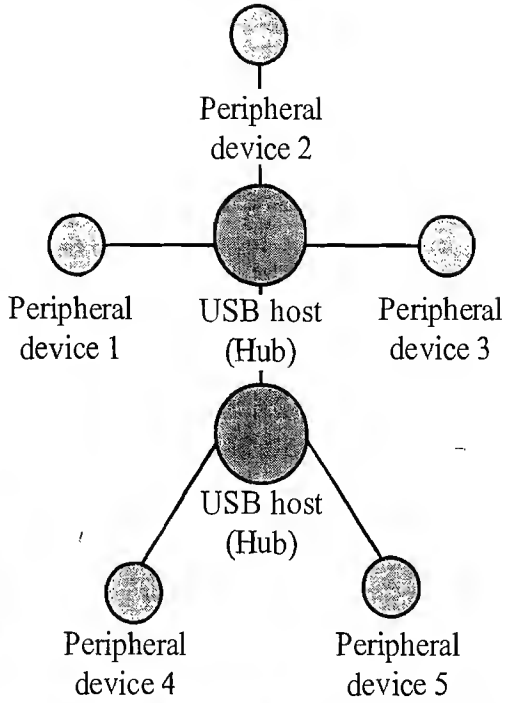


Fig. 2.32 **USB Device Connection topology**

USB uses differential signals for data transmission. It improves the noise immunity. USB interface has the ability to supply power to the connecting devices. Two connection lines (Ground and Power) of the USB interface are dedicated for carrying power. It can supply power up to 500 mA at 5 V. It is sufficient to operate low power devices. Mini and Micro USB connectors are available for small form factor devices like portable media players.

Each USB device contains a Product ID (PID) and a Vendor ID (VID). The PID and VID are embedded into the USB chip by the USB device manufacturer. The VID for a device is supplied by the USB standards forum. PID and VID are essential for loading the drivers corresponding to a USB device for communication.

USB supports four different types of data transfers, namely; Control, Bulk, Isochronous and Interrupt. **Control transfer** is used by USB system software to query, configure and issue commands to the USB device. **Bulk transfer** is used for sending a block of data to a device. Bulk transfer supports error checking and correction. Transferring data to a printer is an example for bulk transfer. **Isochronous data** transfer is used for real-time data communication. In Isochronous transfer, data is transmitted as streams in real-time. Isochronous transfer doesn't support error checking and re-transmission of data in case of any transmission loss. All streaming devices like audio devices and medical equipment for data collection make use of the isochronous transfer. **Interrupt transfer** is used for transferring small amount of data. Interrupt transfer mechanism makes use of polling technique to see whether the USB device has any data to send. The frequency of polling is determined by the USB device and it varies from 1 to 255 milliseconds. Devices like Mouse and Keyboard, which transmits fewer amounts of data, uses Interrupt transfer.

USB.ORG (www.usb.org) is the standards body for defining and controlling the standards for USB communication. Presently USB supports four different data rates namely; Low Speed (1.5Mbps), Full Speed (12Mbps), High Speed (480Mbps) and Super Speed (4.8Gbps). The Low Speed and Full Speed specifications are defined by USB 1.0 and the High Speed specification is defined by USB 2.0. USB 3.0

https://hemanthrajhemu.github.io

defines the specifications for Super Speed. USB 3.0 is expected to be in action by year 2009. There is a move happening towards wireless USB for data transmission using Ultra Wide Band (UWB) technology. Some laptops are already available in the market with wireless USB support.

2.4.2.3 IEEE 1394 (Firewire) *IEEE 1394* is a wired, isochronous high speed serial communication bus. It is also known as High Performance Serial Bus (HPSB). The research on 1394 was started by Apple Inc. in 1985 and the standard for this was coined by IEEE. The implementation of it is available from various players with different names. Apple Inc's (www.apple.com) implementation of 1394 protocol is popularly known as ***Firewire***. ***i.LINK*** is the *1394* implementation from Sony Corporation (www.sony.net) and ***Lynx*** is the implementation from Texas Instruments (www.ti.com). *1394* supports peer-to-peer connection and point-to-multipoint communication allowing 63 devices to be connected on the bus in a tree topology. *1394* is a wired serial interface and it can support a cable length of up to 15 feet for interconnection.

The *1394* standard has evolved a lot from the first version *IEEE 1394–1995* released in 1995 to the recent version *IEEE 1394–2008* released in June 2008. The *1394* standard supports a data rate of 400 to 3200Mbits/second. The *IEEE 1394* uses differential data transfer (The information is sent using differential signals through a pair of twisted cables. It increases the noise immunity) and the interface cable supports 3 types of connectors, namely; 4-pin connector, 6-pin connector (alpha connector) and 9 pin connector (beta connector). The 6 and 9 pin connectors carry power also to support external devices (In case an embedded device is connected to a PC through an *IEEE 1394* cable with 6 or 9 pin connector interface, it can operate from the power available through the connector.) It can supply unregulated power in the range of 24 to 30V. (The Apple implementation is for battery operated devices and it can supply a voltage in the range 9 to 12V.) The table given below illustrates the pin details for 4, 6 and 9 pin connectors.

Pin name	Pin no: (4 Pin Connector)	Pin no: (6 Pin Connector)	Pin no: (9 Pin Connector)	Description
Power		1	8	Unregulated DC supply, 24 to 30V
Signal Ground		2	6	Ground connection
TPB–	1	3	1	Differential Signal line for Signal line B
TPB+	2	4	2	Differential Signal line for Signal line B
TPA–	3	5	3	Differential Signal line for Signal line A
TPA+	4	6	4	Differential Signal line for Signal line A
TPA(S)			5	Shield for the differential signal line A. Normally grounded
TPB(S)			9	Shield for the differential signal line B. Normally grounded
NC			7	No connection

There are two differential data transfer lines A and B per connector. In a *1394* cable, normally the differential lines of A are connected to B (TPA+ to TPB+ and TPA–to TPB–) and vice versa.

1394 is a popular communication interface for connecting embedded devices like Digital Camera, Camcorder, Scanners to desktop computers for data transfer and storage.

Unlike USB interface (Except USB OTG), *IEEE 1394* doesn't require a host for communicating between devices. For example, you can directly connect a scanner with a printer for printing. The data-

https://hemanthrajhemu.github.io

rate supported by *1394* is far higher than the one supported by *USB2.0 interface*. The *1394* hardware implementation is much costlier than USB implementation.

2.4.2.4 Infrared (IrDA) Infrared (IrDA) is a serial, half duplex, line of sight based wireless technology for data communication between devices. It is in use from the olden days of communication and you may be very familiar with it. The remote control of your TV, VCD player, etc. works on Infrared data communication principle. Infrared communication technique uses infrared waves of the electromagnetic spectrum for transmitting the data. IrDA supports point-point and point-to-multipoint communication, provided all devices involved in the communication are within the line of sight. The typical communication range for IrDA lies in the range 10 cm to 1 m. The range can be increased by increasing the transmitting power of the IR device. IR supports data rates ranging from 9600bits/second to 16Mbps. Depending on the speed of data transmission IR is classified into Serial IR (SIR), Medium IR (MIR), Fast IR (FIR), Very Fast IR (VFIR) and Ultra Fast IR (UFIR). SIR supports transmission rates ranging from 9600bps to 115.2kbps. MIR supports data rates of 0.576Mbps and 1.152Mbps. FIR supports data rates up to 4Mbps. VFIR is designed to support high data rates up to 16Mbps. The UFIR specs are under development and it is targeting a data rate up to 100Mbps.

IrDA communication involves a transmitter unit for transmitting the data over IR and a receiver for receiving the data. Infrared Light Emitting Diode (LED) is the IR source for transmitter and at the receiving end a photodiode acts as the receiver. Both transmitter and receiver unit will be present in each device supporting IrDA communication for bidirectional data transfer. Such IR units are known as 'Transceiver'. Certain devices like a TV remote control always require unidirectional communication and so they contain either the transmitter or receiver unit (The remote control unit contains the transmitter unit and TV contains the receiver unit).

'Infra-red Data Association' (IrDA - http://www.irda.org/) is the regulatory body responsible for defining and licensing the specifications for IR data communication. IrDA communication has two essential parts; a physical link part and a protocol part. The physical link is responsible for the physical transmission of data between devices supporting IR communication and protocol part is responsible for defining the rules of communication. The physical link works on the wireless principle making use of Infrared for communication. The IrDA specifications include the standard for both physical link and protocol layer.

The IrDA control protocol contains implementations for Physical Layer (PHY), Media Access Control (MAC) and Logical Link Control (LLC). The Physical Layer defines the physical characteristics of communication like range, data rates, power, etc.

IrDA is a popular interface for file exchange and data transfer in low cost devices. IrDA was the prominent communication channel in mobile phones before Bluetooth's existence. Even now most of the mobile phone devices support IrDA.

2.4.2.5 Bluetooth (BT) Bluetooth is a low cost, low power, short range wireless technology for data and voice communication. Bluetooth was first proposed by 'Ericsson' in 1994. Bluetooth operates at 2.4GHz of the Radio Frequency spectrum and uses the Frequency Hopping Spread Spectrum (FHSS) technique for communication. Literally it supports a data rate of up to 1Mbps and a range of approximately 30 feet for data communication. Like IrDA, Bluetooth communication also has two essential parts; a physical link part and a protocol part. The physical link is responsible for the physical transmission of data between devices supporting Bluetooth communication and protocol part is responsible

https://hemanthrajhemu.github.io

for defining the rules of communication. The physical link works on the wireless principle making use of RF waves for communication. Bluetooth enabled devices essentially contain a Bluetooth wireless radio for the transmission and reception of data. The rules governing the Bluetooth communication is implemented in the 'Bluetooth protocol stack'. The Bluetooth communication IC holds the stack. Each Bluetooth device will have a 48 bit unique identification number. Bluetooth communication follows packet based data transfer.

Bluetooth supports point-to-point (device to device) and point-to-multipoint (device to multiple device broadcasting) wireless communication. The point-to-point communication follows the master-slave relationship. A Bluetooth device can function as either master or slave. When a network is formed with one Bluetooth device as master and more than one device as slaves, it is called a Piconet. A Piconet supports a maximum of seven slave devices.

Bluetooth is the favourite choice for short range data communication in handheld embedded devices. Bluetooth technology is very popular among cell phone users as they are the easiest communication channel for transferring ringtones, music files, pictures, media files, etc. between neighbouring Bluetooth enabled phones.

The Bluetooth standard specifies the minimum requirements that a Bluetooth device must support for a specific usage scenario. The Generic Access Profile (GAP) defines the requirements for detecting a Bluetooth device and establishing a connection with it. All other specific usage profiles are based on GAP. Serial Port Profile (SPP) for serial data communication, File Transfer Profile (FTP) for file transfer between devices, Human Interface Device (HID) for supporting human interface devices like keyboard and mouse are examples for Bluetooth profiles.

The specifications for Bluetooth communication is defined and licensed by the standards body 'Bluetooth Special Interest Group (SIG)'. For more information, please visit the website www.bluetooth.org.

2.4.2.6 Wi-Fi Wi-Fi or Wireless Fidelity is the popular wireless communication technique for networked communication of devices. Wi-Fi follows the IEEE 802.11 standard. Wi-Fi is intended for network communication and it supports Internet Protocol (IP) based communication. It is essential to have device identities in a multipoint communication to address specific devices for data communication. In an IP based communication each device is identified by an IP address, which is unique to each device on the network. Wi-Fi based communications require an intermediate agent called Wi-Fi router/Wireless Access point to manage the communications. The Wi-Fi router is responsible for restricting the access to a network, assigning IP address to devices on the network, routing data packets to the intended devices on the network. Wi-Fi enabled devices contain a wireless adaptor for transmitting and receiving data in the form of radio signals through an antenna. The hardware part of it is known as Wi-Fi Radio.

Wi-Fi operates at 2.4GHz or 5GHz of radio spectrum and they co-exist with other ISM band devices like Bluetooth. Figure 2.33 illustrates the typical interfacing of devices in a Wi-Fi network.

For communicating with devices over a Wi-Fi network, the device when its Wi-Fi radio is turned ON, searches the available Wi-Fi network in its vicinity and lists out the Service Set Identifier (SSID) of the available networks. If the network is security enabled, a password may be required to connect to a particular SSID. Wi-Fi employs different security mechanisms like Wired Equivalency Privacy (WEP) Wireless Protected Access (WPA), etc. for securing the data communication.

Wi-Fi supports data rates ranging from 1Mbps to 150Mbps (Growing towards higher rates as technology progresses) depending on the standards (802.11a/b/g/n) and access/modulation method. Depending on the type of antenna and usage location (indoor/outdoor), Wi-Fi offers a range of 100 to 300 feet.

https://hemanthrajhemu.github.io

2.4.2.7 ZigBee ZigBee is a low power, low cost, wireless network communication protocol based on the IEEE 802.15.4-2006 standard. ZigBee is targeted for low power, low data rate and secure applications for Wireless Personal Area Networking (WPAN). The ZigBee specifications support a robust mesh network containing multiple nodes. This networking strategy makes the network reliable by permitting messages to travel through a number of different paths to get from one node to another.

ZigBee operates worldwide at the unlicensed bands of Radio spectrum, mainly at 2.400 to 2.484 GHz, 902 to 928 MHz and 868.0 to 868.6 MHz. ZigBee Supports an operating distance of up to 100 metres and a data rate of 20 to 250Kbps.

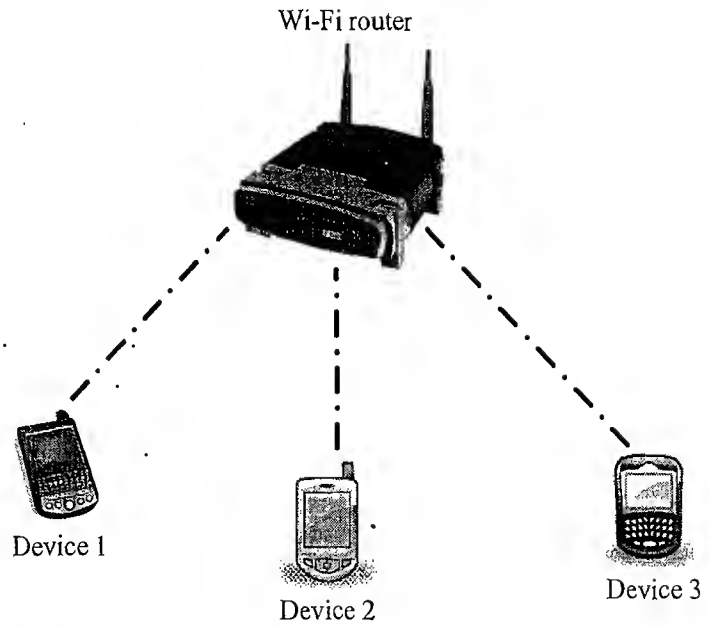


Fig. 2.33 **Wi-Fi network**

In the ZigBee terminology, each ZigBee device falls under any one of the following ZigBee device category.

ZigBee Coordinator (ZC)/Network Coordinator: The ZigBee coordinator acts as the root of the ZigBee network. The ZC is responsible for initiating the ZigBee network and it has the capability to store information about the network.

ZigBee Router (ZR)/Full function Device (FFD): Responsible for passing information from device to another device or to another ZR.

ZigBee End Device (ZED)/Reduced Function Device (RFD): End device containing ZigBee functionality for data communication. It can talk only with a ZR or ZC and doesn't have the capability to act as a mediator for transferring data from one device to another.

The diagram shown in Fig. 2.34 gives an overview of ZC, ZED and ZR in a ZigBee network.

ZigBee is primarily targeting application areas like home & industrial automation, energy management, home control/ security, medical/patient tracking, logistics & asset tracking and sensor networks & active RFID. Automatic Meter Reading (AMR), smoke detectors, wireless telemetry, HVAC control, heating control, lighting controls, environmental controls, etc. are examples for applications which can make use of the ZigBee technology.

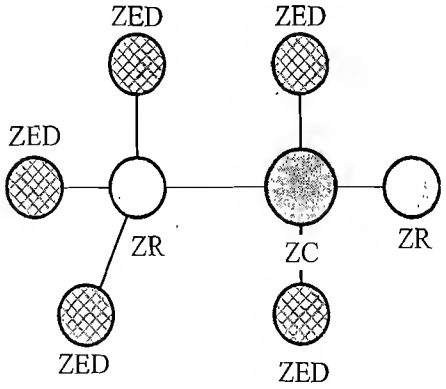


Fig. 2.34 **A ZigBee network model**

The specifications for ZigBee is developed and managed by the ZigBee alliance (www.zigbee.org), a non-profit consortium of leading semiconductor manufacturers, technology providers, OEMs and end-users worldwide.

2.4.2.8 General Packet Radio Service (GPRS) General Packet Radio Service (GPRS) is a communication technique for transferring data over a mobile communication network like GSM. Data is sent as packets in GPRS communication. The transmitting device splits the data into several related packets. At the receiving end the data is re-constructed by combining the received data packets. GPRS

https://hemanthrajhemu.github.io

supports a theoretical maximum transfer rate of 171.2kbps. In GPRS communication, the radio channel is concurrently shared between several users instead of dedicating a radio channel to a cell phone user. The GPRS communication divides the channel into 8 timeslots and transmits data over the available channel. GPRS supports Internet Protocol (IP), Point to Point Protocol (PPP) and X.25 protocols for communication.

GPRS is mainly used by mobile enabled embedded devices for data communication. The device should support the necessary GPRS hardware like GPRS modem and GPRS radio. To accomplish GPRS based communication, the carrier network also should have support for GPRS communication. GPRS is an old technology and it is being replaced by new generation data communication techniques like EDGE, High Speed Downlink Packet Access (HSDPA), etc. which offers higher bandwidths for communication.

2.5 EMBEDDED FIRMWARE

Embedded firmware refers to the control algorithm (Program instructions) and or the configuration settings that an embedded system developer dumps into the code (Program) memory of the embedded system. It is an un-avoidable part of an embedded system. There are various methods available for developing the embedded firmware. They are listed below.

1. Write the program in high level languages like Embedded C/C++ using an Integrated Development Environment (The IDE will contain an editor, compiler, linker, debugger, simulator, etc. IDEs are different for different family of processors/controllers. For example, Keil micro vision3 IDE is used for all family members of *8051* microcontroller, since it contains the generic *8051* compiler C51).
2. Write the program in Assembly language using the instructions supported by your application's target processor/controller.

The instruction set for each family of processor/controller is different and the program written in either of the methods given above should be converted into a processor understandable machine code before loading it into the program memory.

The process of converting the program written in either a high level language or processor/controller specific Assembly code to machine readable binary code is called '*HEX File Creation*'. The methods used for '*HEX File Creation*' is different depending on the programming techniques used. If the program is written in Embedded C/C++ using an IDE, the cross compiler included in the IDE converts it into corresponding processor/controller understandable '*HEX File*'. If you are following the Assembly language based programming technique (method 2), you can use the utilities supplied by the processor/controller vendors to convert the source code into '*HEX File*'. Also third party tools are available, which may be of free of cost, for this conversion.

For a beginner in the embedded software field, it is strongly recommended to use the high level language based development technique. The reasons for this being: writing codes in a high level language is easy, the code written in high level language is highly portable which means you can use the same code to run on different processor/controller with little or less modification. The only thing you need to do is re-compile the program with the required processor's IDE, after replacing the include files for that particular processor. Also the programs written in high level languages are not developer dependent. Any skilled programmer can trace out the functionalities of the program by just having a look at the program. It will be much easier if the source code contains necessary comments and documentation lines. It is very easy to debug and the overall system development time will be reduced to a greater extent.

https://hemanthrajhemu.github.io

The embedded software development process in assembly language is tedious and time consuming. The developer needs to know about all the instruction sets of the processor/controller or at least s/he should carry an instruction set reference manual with her/him. A programmer using assembly language technique writes the program according to his/her view and taste. Often he/she may be writing a method or functionality which can be achieved through a single instruction as an experienced person's point of view, by two or three instructions in his/her own style. So the program will be highly dependent on the developer. It is very difficult for a second person to understand the code written in Assembly even if it is well documented.

We will discuss both approaches of embedded software development in a later chapter dealing with design of embedded firmware, in detail. Two types of control algorithm design exist in embedded firmware development. The first type of control algorithm development is known as the infinite loop or '*super loop*' based approach, where the control flow runs from top to bottom and then jumps back to the top of the program in a conventional procedure. It is similar to the *while (1) { };* based technique in C. The second method deals with splitting the functions to be executed into tasks and running these tasks using a scheduler which is part of a General Purpose or Real Time Embedded Operating System (GPOS/RTOS). We will discuss both of these approaches in separate chapters of this book.

2.6 OTHER SYSTEM COMPONENTS

The other system components refer to the components/circuits/ICs which are necessary for the proper functioning of the embedded system. Some of these circuits may be essential for the proper functioning of the processor/controller and firmware execution. Watchdog timer, Reset IC (or passive circuit), brown-out protection IC (or passive circuit), etc. are examples of circuits/ICs which are essential for the proper functioning of the processor/controllers. Some of the controllers or SoCs integrate these components within a single IC and doesn't require such components externally connected to the chip for proper functioning. Depending on the system requirement, the embedded system may include other integrated circuits for performing specific functions, level translator ICs for interfacing circuits with different logic levels, etc. The following section explains the essential circuits for the proper functioning of the processor/controller of the embedded system.

2.6.1 Reset Circuit

The reset circuit is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON. The reset signal brings the internal registers and the different hardware systems of the processor/controller to a known state and starts the firmware execution from the reset vector (Normally from vector address 0x0000 for conventional processors/controllers. The reset vector can be relocated to an address for processors/controllers supporting bootloader). The reset signal can be either active high (The processor undergoes reset when the reset pin of the processor is at logic high) or active low (The processor undergoes reset when the reset pin of the processor is at logic low). Since the processor operation is synchronised to a clock signal, the reset pulse should be wide enough to give time for the clock oscillator to stabilise before the internal reset state starts. The reset signal to the processor can be applied at power ON through an external passive reset circuit comprising a Capacitor and Resistor or through a standard Reset IC like MAX810 from Maxim Dallas (www.maxim-ic.com). Select the reset IC based on the type of reset signal and logic level (CMOS/TTL) supported by the processor/controller in use. Some microprocessors/controllers contain built-in internal

https://hemanthrajhemu.github.io

reset circuitry and they don't require external reset circuitry. Figure 2.35 illustrates a resistor capacitor based passive reset circuit for active high and low configurations. The reset pulse width can be adjusted by changing the resistance value *R* and capacitance value *C*.

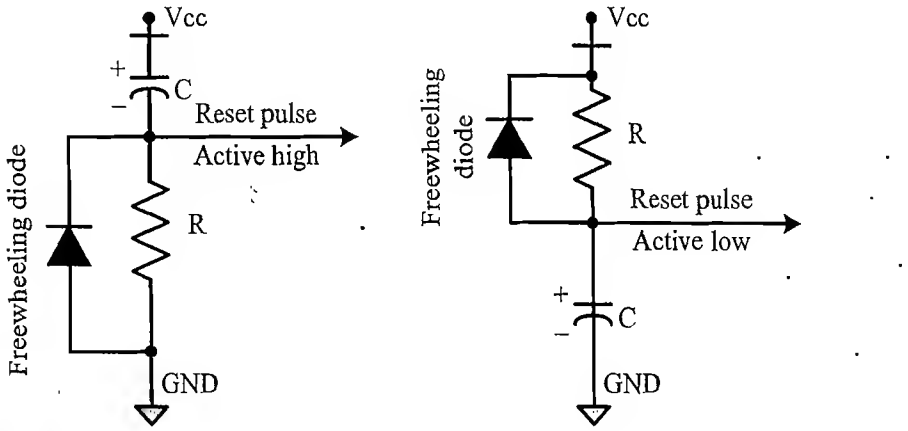


Fig. 2.35 **RC based reset circuit**

2.6.2 Brown-out Protection Circuit

Brown-out protection circuit prevents the processor/controller from unexpected program execution behaviour when the supply voltage to the processor/controller falls below a specified voltage. It is essential for battery powered devices since there are greater chances for the battery voltage to drop below the required threshold. The processor behaviour may not be predictable if the supply voltage falls below the recommended operating voltage. It may lead to situations like data corruption. A brown-out protection circuit holds the processor/controller in reset state, when the operating voltage falls below the threshold, until it rises above the threshold voltage. Certain processors/controllers support built in brown-out protection circuit which monitors the supply voltage internally. If the processor/controller doesn't integrate a built-in brown-out protection circuit, the same can be implemented using external passive circuits or supervisor ICs. Figure 2.36 illustrates a brown-out circuit implementation using Zener diode and transistor for processor/controller with active low Reset logic.

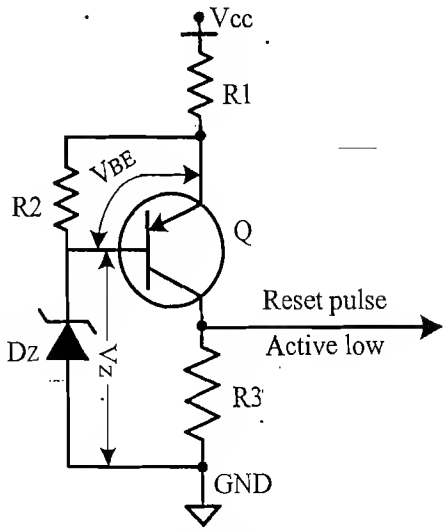


Fig. 2.36 **Brown-out protection circuit with Active low output**

The Zener diode Dz and transistor Q forms the heart of this circuit. The transistor conducts always when the supply voltage V_{cc} is greater than that of the sum of V_{BE} and V_z (Zener voltage). The transistor stops conducting when the supply voltage falls below the sum of V_{BE} and V_Z. Select the Zener diode with required voltage for setting the low threshold value for *V*cc. The values of R1, R2, and R3 can be selected based on the electrical characteristics (Absolute maximum current and voltage ratings) of the transistor in use. Microprocessor Supervisor ICs like DS1232 from Maxim Dallas (www.maxim-ic.com) also provides Brown-out protection.

https://hemanthrajhemu.github.io

2.6.3 Oscillator Unit

A microprocessor/microcontroller is a digital device made up of digital combinational and sequential circuits. The instruction execution of a microprocessor/controller occurs in sync with a clock signal. It is analogous to the heartbeat of a living being which synchronises the execution of life. For a living being, the heart is responsible for the generation of the beat whereas the oscillator unit of the embedded system is responsible for generating the precise clock for the processor. Certain processors/controllers integrate a built-in oscillator unit and simply require an external ceramic resonator/quartz crystal for producing the necessary clock signals. Quartz crystals and ceramic resonators are equivalent in operation, however they possess physical difference. A quartz crystal is normally mounted in a hermetically sealed metal case with two leads protruding out of the case. Certain devices may not contain a built-in oscillator unit and require the clock pulses to be generated and supplied externally. Quartz crystal Oscillators are available in the form chips and they can be used for generating the clock pulses in such a cases. The speed of operation of a processor is primarily dependent on the clock frequency. However we cannot increase the clock frequency blindly for increasing the speed of execution. The logical circuits lying inside the processor always have an upper threshold value for the maximum clock at which the system can run, beyond which the system becomes unstable and non functional. The total system power consumption is directly proportional to the clock frequency. The power consumption increases with increase in clock frequency. The accuracy of program execution depends on the accuracy of the clock signal. The accuracy of the crystal oscillator or ceramic resonator is normally expressed in terms of +/-ppm (Parts per million). Figure 2.37 illustrates the usage of quartz crystal/ceramic resonator and external oscillator chip for clock generation.

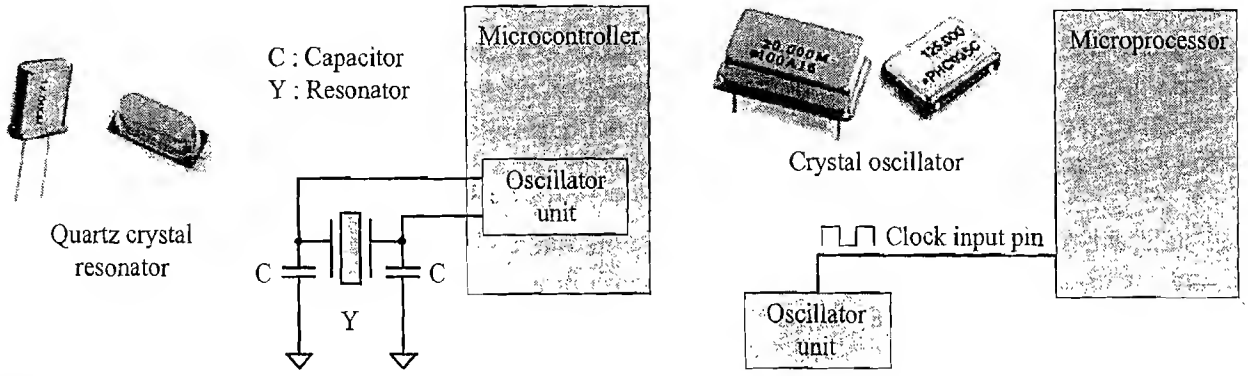


Fig. 2.37 **Oscillator circuitry using quartz crystal and quartz crystal oscillator**

2.6.4 Real-Time Clock (RTC)

Real-Time Clock (RTC) is a system component responsible for keeping track of time. RTC holds information like current time (In hours, minutes and seconds) in 12 hour/24 hour format, date, month, year, day of the week, etc. and supplies timing reference to the system. RTC is intended to function even in the absence of power. RTCs are available in the form of Integrated Circuits from different semiconductor manufacturers like Maxim/Dallas, ST Microelectronics etc. The RTC chip contains a microchip for holding the time and date related information and backup battery cell for functioning in the absence of power, in a single IC package. The RTC chip is interfaced to the processor or controller of the embedded system. For Operating System based embedded devices, a timing reference is essential for synchronising

https://hemanthrajhemu.github.io

the operations of the OS kernel. The RTC can interrupt the OS kernel by asserting the interrupt line of the processor/controller to which the RTC interrupt line is connected. The OS kernel identifies the interrupt in terms of the Interrupt Request (IRQ) number generated by an interrupt controller. One IRQ can be assigned to the RTC interrupt and the kernel can perform necessary operations like system date time updation, managing software timers etc when an RTC timer tick interrupt occurs. The RTC can be configured to interrupt the processor at predefined intervals or to interrupt the processor when the RTC register reaches a specified value (used as alarm interrupt).

2.6.5 Watchdog Timer

In desktop Windows systems, if we feel our application is behaving in an abnormal way or if the system hangs up, we have the 'Ctrl + Alt + Del' to come out of the situation. What if it happens to our embedded system? Do we really have a 'Ctrl + Alt + Del' to take control of the situation? Of course not ☹, but we have a watchdog to monitor the firmware execution and reset the system processor/microcontroller when the program execution hangs up. A watchdog timer, or simply a watchdog, is a hardware timer for monitoring the firmware execution. Depending on the internal implementation, the watchdog timer increments or decrements a free running counter with each clock pulse and generates a reset signal to reset the processor if the count reaches zero for a down counting watchdog, or the highest count value for an upcounting watchdog. If the watchdog counter is in the enabled state, the firmware can write a zero (for upcounting watchdog implementation) to it before starting the execution of a piece of code (subroutine or portion of code which is susceptible to execution hang up) and the watchdog will start counting. If the firmware execution doesn't complete due to malfunctioning, within the time required by the watchdog to reach the maximum count, the counter will generate a reset pulse and this will reset the processor (if it is connected to the reset line of the processor). If the firmware execution completes before the expiration of the watchdog timer you can reset the count by writing a 0 (for an upcounting watchdog timer) to the watchdog timer register. Most of the processors implement watchdog as a built-in component and provides status register to control the watchdog timer (like enabling and disabling watchdog functioning) and watchdog timer register for writing the count value. If the processor/controller doesn't contain a built in watchdog timer, the same can be implemented using an external watchdog timer IC circuit. The external watchdog timer uses hardware logic for enabling/disabling, resetting the watchdog count, etc instead of the firmware based 'writing' to the status and watchdog timer register. The Microprocessor supervisor IC DS1232 integrates a hardware watchdog timer in it. In modern systems running on embedded operating systems, the watchdog can be implemented in such a way that when a watchdog timeout occurs, an interrupt is generated instead of resetting the processor. The interrupt handler for this handles the situation in an appropriate fashion. Figure 2.38 illustrates the implementa-

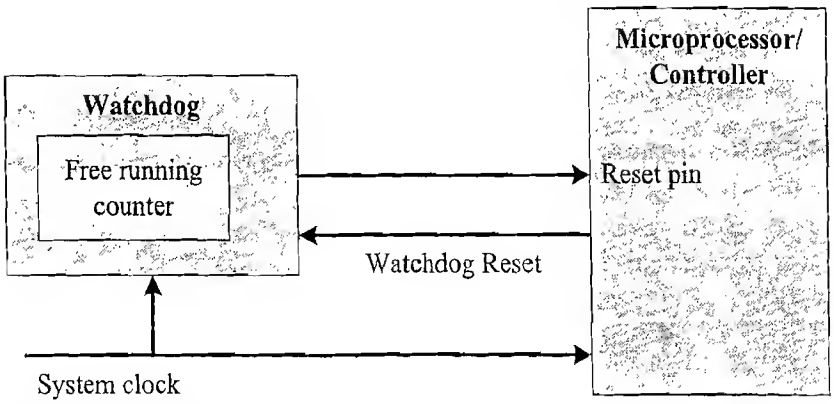


Fig. 2.38 Watchdog timer for firmware execution supervision

https://hemanthrajhemu.github.io

tion of an external watchdog timer based microprocessor supervisor circuit for a small scale embedded system.

2.7 PCB AND PASSIVE COMPONENTS

Printed Circuit Board (PCB) is the backbone of every embedded system. After finalising the components and the inter-connection among them, a schematic design is created and according to the schematic the PCB is fabricated. This will be described in detail in a chapter dedicated for "Embedded Hardware Design and Development". PCB acts as a platform for mounting all the necessary components as per the design requirement. Also it acts as a platform for testing your embedded firmware. Apart from the above-mentioned important subsystems of an embedded system, you can find some passive electronic components like resistor, capacitor, diodes, etc. on your board. They are the co-workers of various chips contained in your embedded hardware. They are very essential for the proper functioning of your embedded system. For example for providing a regulated ripple-free supply voltage to the system, a regulator IC and spike suppressor filter capacitors are very essential.

Summary

- ✓ The core of an embedded system is usually built around a commercial off-the-shelf component or an application specific integrated circuit (ASIC) or a general purpose processor like a microprocessor or microcontroller or application specific instruction set processors (ASIP like DSP, Microcontroller, etc.) or a Programmable Logic Device (PLD) or a System on Chip (SoC)
- ✓ Processors/controllers support either Reduced Instruction Set Computing (RISC) or Complex Instruction Set Computing (CISC)
- ✓ Microprocessors/controllers based on the Harvard architecture will have separate data bus and instruction bus, whereas Microprocessors/controllers based on the Von-Neumann architecture shares a single common bus for fetching both instructions and data
- ✓ The Big-endian processors store the higher-order byte of the data in memory at the lowest address, whereas Little-endian processors store the lower-order byte of data in memory at the lowest address
- ✓ Field Programmable Gate Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs) are the two major types of programmable logic devices
- ✓ The Read Only Memory (ROM) is a non-volatile memory for storing the firmware and embedded OS files. MROM, PROM (OTP), EPROM, EEPROM and FLASH are the commonly used firmware storage memory
- ✓ Random Access Memory (RAM) is a volatile memory for temporary data storage. RAM can be either Static RAM (SRAM) or Dynamic RAM (DRAM). SRAM is made up of flip-flops, whereas DRAM is made up of MOS Transistor and Capacitor
- ✓ The sensors connected to the input port of an embedded system senses the changes in input variables and the actuators connected at the output port of an embedded system controls some variables in accordance with changes in input
- ✓ Light Emitting Diode (LED), 7-Segment LED displays, Liquid Crystal Display (LCD), Piezo Buzzer, Speaker, Optocoupler, Stepper Motor, Digital to Analog Converters (DAC), Relays etc are examples for output devices of an embedded system
- ✓ Keyboard, Touch screen, Push Button switches, Optocoupler, Analog to Digital Converter (ADC) etc are examples for Input devices in an embedded system

https://hemanthrajhemu.github.io

- ✓ The Programmable Peripheral Interface (PPI) device extends the I/O capabilities of the processor used in embedded system
- ✓ I2C, SPI, UART, 1-Wire, Parallel bus etc are examples for onboard communication interface and RS-232C, RS-485, USB, IEEE1394 (FireWire), Wi-Fi, ZigBee, Infrared (IrDA), Bluetooth, GPRS, etc. are examples for external communication interface
- ✓ The control algorithm for the embedded device is known as Embedded Firmware. Embedded firmware can be developed on top of an embedded operating system or without an operating system
- ✓ The reset circuit ensures that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON. The reset signal brings the internal registers and the different hardware systems of the processor/controller to a known state and starts the firmware execution from the reset vector
- ✓ The brown-out protection circuit prevents the processor/controller from unexpected program execution behaviour when the supply voltage to the processor/controller falls below a specified voltage
- ✓ The oscillator unit generates clock signals for synchronising the operations of the processor
- ✓ The time keeping activity for the embedded system is performed by the Real Time Clock (RTC) of the system. RTC holds current time, date, month, year, day of the week, etc.
- ✓ The watchdog timer monitors the firmware execution and resets the processor or generates an Interrupt in case the execution time for a task is exceeding the maximum allowed limit
- ✓ Printed circuit board or PCB acts as a platform for mounting all the necessary hardware components as per the design requirement

Keywords

COTS	: Commercial-off-the-Shelf. Commercially available ready to use component
ASIC	: Application Specific Integrated Circuit is a microchip designed to perform a specific or unique application
ASSP	: Application Specific Standard Product—An ASIC marketed to multiple customers just as a general-purpose product is, but to a smaller number of customers
Microprocessor	: A silicon chip representing a Central Processing Unit (CPU)
GPP	: General Purpose Processor or GPP is a processor designed for general computational tasks
ASIP	: Application Specific Instruction Set processors are processors with architecture and instruction set optimized to specific domain/application requirements
Microcontroller	: A highly integrated chip that contains a CPU, scratchpad RAM, Special and General purpose Register Arrays and Integrated peripherals
DSP	: Digital Signal Processor is a powerful special purpose 8/16/32 bit microprocessors designed specifically to meet the computational demands and power constraints
RISC	: Reduced Instruction Set Computing. Refers to processors with Reduced and Orthogonal Instruction Set
CISC	: Refers to processors with Complex Instruction Set Computing
Harvard Architecture	: A type of processor architecture with separate buses for program instruction and data fetch
Von-Neumann Architecture	: A type of processor architecture with a shared single common bus for fetching both instructions and data
Big-Endian	: Refers to processors which store the higher-order byte of the data in memory at the lowest address
Little-Endian	: Refers to processors which store the lower-order byte of the data in memory at the lowest address

https://hemanthrajhemu.github.io

FPGA	: Field Programmable Gate Array Device. A programmable logic device with reconfigurable function. Popular for prototyping ASIC designs
MROM	: Masked ROM is a one-time programmable memory, which uses the hardwired technology for storing data
OTP	: One Time Programmable Read Only Memory made up of nichrome or polysilicon wires arranged in a matrix
EPROM	: Erasable Programmable Read Only Memory. Reprogrammable ROM. Erased by exposing to UV light
EEPROM	: Electrically Erasable Programmable Read Only Memory. Reprogrammable ROM. Erased by applying electrical signals
FLASH	: Electrically Erasable Programmable Read Only Memory. Same as EEPROM but with high capacity and support for block level memory erasing
RAM	: Random Access memory. Volatile memory
SRAM	: Static RAM. A type of RAM, made up of flip-flops
DRAM	: Dynamic RAM. A type of RAM, made up of MOS Transistor and Capacitor
NVRAM	: Non-volatile SRAM. Battery-backed SRAM
ADC	: Analog to Digital Converter. An integrated circuit which converts analog signals to digital form
LED	: Light Emitting Diode. An output device which produces visual indication in the form of light in accordance with current flow
7-Segment LED Display	: The 7-segment LED display is an output device for displaying alpha numeric characters. It contains 8 light-emitting diode (LED) segments arranged in a special form
Optocoupler	: A solid state device to isolate two parts of a circuit. Optocoupler combines an LED and a photo-transistor in a single housing (package)
Stepper motor	: An electro-mechanical device which generates discrete displacement (motion) in response to dc electrical signals
Relay	: An electro-mechanical device which acts as dynamic path selector for signals and power
Piezo Buzzer	: A piezo-electric device for generating audio indication. It contains a piezo-electric diaphragm which produces audible sound in response to the voltage applied to it
Push Button switch	: A mechanical device for electric circuit 'make' and 'break' operation
PPI	: Programmable Peripheral Interface is a device for extending the I/O capabilities of processors/ controllers
I2C	: The Inter Integrated Circuit Bus (I2C–Pronounced 'I square C') is a synchronous bi-directional half duplex two wire serial interface bus.
SPI	: The Serial Peripheral Interface Bus (SPI) is a synchronous bi-directional full duplex four wire serial interface bus
UART	: The Universal Asynchronous Receiver Transmitter is an asynchronous communication implementation standard
1-Wire interface	: An asynchronous half-duplex communication protocol developed by Maxim Dallas Semiconductor. It is also known as Dallas 1-Wire® protocol.
RS-232 C	: Recommended Standard number 232, revision C from the Electronic Industry Association, is a legacy, full duplex, wired, asynchronous serial communication interface
RS-485	: The enhanced version of RS-232, which supports multi-drop communication with up to 32 transmitting devices (drivers) and 32 receiving devices on the bus
USB	: Universal Serial Bus is a wired high speed serial bus for data communication
IEEE 1394	: A wired, isochronous high speed serial communication bus
Firewire	: The Apple Inc.'s implementation of the 1394 protocol

https://hemanthrajhemu.github.io

Infrared (IrDA)	:	A serial, half duplex, line of sight based wireless technology for data communication between devices
Bluetooth	:	A low cost, low power, short range wireless technology for data and voice communication
Wi-Fi	:	Wireless Fidelity is the popular wireless communication technique for networked communication of devices
ZigBee	:	A low power, low cost, wireless network communication protocol based on the IEEE 802.15.4-2006 standard. ZigBee is targeted for low power, low data-rate and secure applications for Wireless Personal Area Networking (WPAN)
GPRS	:	General Packet Radio Service is a communication technique for transferring data over a mobile communication network like GSM
Reset Circuit	:	A passive circuit or IC device to supply a reset signal to the processor/controller of the embedded system
Brown-out Protection Circuit	:	A passive circuit or IC device to protect the processor from unexpected program execution flow due to the drop in power supply voltage
RTC	:	Real Time Clock is a system component keeping track of time
Watchdog Timer (WDT)	:	Timer for monitoring the firmware execution
PCB	:	Printed Circuit Board is the place holder for arranging the different hardware components required to build the embedded product

Objective Questions

1. Embedded hardware/software systems are basically designed to
   (a) Regulate a physical variable
   (b) Change the state of some devices
   (c) Measure/Read the state of a variable/device
   (d) Any/All of these
2. Little Endian processors
   (a) Store the lower-order byte of the data at the lowest address and the higher-order byte of the data at the highest address of memory
   (b) Store the higher-order byte of the data at the lowest address and the lower-order byte of the data at the highest address of memory
   (c) Store both higher order and lower order byte of the data at the same address of memory
   (d) None of these
3. An integer variable with value 255 is stored in memory location at 0x8000. The processor word length is 8 bits and the processor is a big endian processor. The size of integer is considered as 4 bytes in the system. What is the value held by the memory location 0x8000?
   (a) 0xFF (b) 0x00 (c) 0x01 (d) None of these
4. The instruction set of RISC processor is
   (a) Simple and lesser in number
   (b) Complex and lesser in number
   (c) Simple and larger in number
   (d) Complex and larger in number
5. Which of the following is true about CISC processors?
   (a) The instruction set is non-orthogonal
   (b) The number of general purpose registers is limited
   (c) Instructions are like macros in C language
   (d) Variable length Instructions
   (e) All of these
   (f) None of these

https://hemanthrajhemu.github.io

6. Which of the following processor architecture supports easier instruction pipelining?
   (a) Harvard (b) Von Neumann (c) Both of them (d) None of these
7. Microprocessors/controllers based on the Harvard architecture will have separate data bus and instruction bus. This allows the data transfer and program fetching to occur simultaneously on both buses. State True or False
   (a) True (b) False
8. Which of the following is one-time programmable memory?
   (a) SRAM (b) PROM (c) FLASH (d) NVRAM
9. Which of the following memory type is best suited for development purpose?
   (a) EEPROM (b) FLASH (c) UVEPROM
   (d) OTP (e) (a) or (b)
10. EEPROM memory is alterable at byte level. State True or False
    (a) True (b) False
11. Non-volatile RAM is a Random Access Memory with battery backup. State True or False
    (a) True (b) False
12. Execution of program from ROM is faster than the execution from RAM. State True or False
    (a) True (b) False
13. Dynamic RAM stores data in the form of voltage. State True or False
    (a) True (b) False
14. The control algorithm (Program instructions) and or the configuration settings that are kept in the code (Program) memory of the embedded system are known as Embedded Software. State True or False
    (a) True (b) False
15. Which of the following is an example for Wireless Communication interface?
    (a) RS-232C (b) Wi-Fi (c) Bluetooth
    (d) EEE1394 (e) both (b) and (c)
16. Which of the following is (are) examples for Application Specific Instruction set Processor(s)
    (a) Intel Centrino (b) Atmel Automotive AVR
    (c) AMD Turion (d) Microchip CAN PIC (e) All of these (f) both (b) and (d)
17. How many memory cells are present in 1Kb RAM
    (a) 1024 (b) 8192 (c) 512 (d) 4096
    (e) None of these
18. Which of the following memory supports Execute in Place (XIP)?
    (a) EEPROM (b) NOR FLASH (c) NAND FLASH
    (d) both (b) and (c) (e) None of these
19. How many memory cells are present in 1Kb Serial EEPROM
    (a) 1024 (b) 8192 (c) 512 (d) 4096
    (e) None of these
20. Which of the following is (are) example(s) for the input subsystem of an embedded system dealing with digital data?
    (a) ADC (b) Optocoupler (c) DAC (d) All of them
    (e) only (a) and (b)
21. Which of the following is (are) example(s) for the output subsystem of an embedded system dealing with digital data?
    (a) LED (b) Optocoupler (c) Stepper Motor (d) All of these
    (e) only (a) and (c)
22. Which of the following is true about optocouplers
    (a) Optocoupler acts as an input device only
    (b) Optocoupler acts as an output device only
    (c) Optocoupler can be used in both input and output circuitry
    (d) None of these

https://hemanthrajhemu.github.io

23. Which of the following is true about a unipolar stepper motor
    (a) Contains only a single winding per stator phase (b) Contains two windings per stator phase
    (c) Contains four windings per stator phase (d) None of these
24. Which of the following is (are) true about normally open single pole relays?
    (a) The circuit remains open when the relay is not energised
    (b) The circuit remains closed when the relay is energised
    (c) There are two output paths
    (d) Both (a) and (b) (e) None of these
25. What is the minimum number I/O line required to interface a 16-Key matrix keyboard?
    (a) 16 (b) 8 (c) 4 (d) 9
26. Which is the optimal row-column configuration for a 24 key matrix keyboard?
    (a) 6×4 (b) 8×3 (c) 12×2 (d) 5×5
27. Which of the following is an example for on-board interface in the embedded system context?
    (a) I2C (b) Bluetooth (c) SPI (d) All of them
    (e) Only (a) and (c)
28. What is the minimum number of interface lines required for implementing I2C interface?
    (a) 1 (b) 2 (c) 3 (d) 4
29. What is the minimum number of interface lines required for implementing SPI interface?
    (a) 2 (b) 3 (c) 4 (d) 5
30. Which of the following are synchronous serial interface?
    (a) I2C (b) SPI (c) UART (d) All of these
    (e) Only (a) and (b)
31. RS-232 is a synchronous serial interface. State True or False
    (a) True (b) False
32. What is the maximum number of USB devices that can be connected to a USB host?
    (a) Unlimited (b) 128 (c) 127 (d) None of these
33. In the ZigBee network, which of the following ZigBee entity stores the information about the network?
    (a) ZigBee Coordinator (b) ZigBee Router
    (c) ZigBee Reduced Function Device (d) All of them
34. What is the theoretical maximum data rate supported by GPRS
    (a) 8Mbps (b) 12Mbps (c) 100Kbps (d) 171.2Kbps
35. GPRS communication divides the radio channel into _____ timeslots
    (a) 2 (b) 3 (c) 5 (d) 8

Review Questions

1. Explain the components of a typical embedded system in detail
2. Which are the components used as the core of an embedded system? Explain the merits, drawbacks, if any, and the applications/domains where they are commonly used
3. What is Application Specific Integrated Circuit (ASIC)? Explain the role of ASIC in Embedded System design?
4. What is the difference between Application Specific Integrated Circuit (ASIC) Application Specific Standard Product (ASSP)?
5. What is the difference between microprocessor and microcontroller? Explain the role of microprocessors and controllers in embedded system design?
6. What is Digital Signal Processor (DSP)? Explain the role of DSP in embedded system design?
7. What is the difference between RISC and CISC processors? Give an example for each.

https://hemanthrajhemu.github.io

8. What is processor architecture? What are the different processor architectures available for processor/controller design? Give an example for each.
9. What is the difference between big-endian and little-endian processors? Give an example of each.
10. What is Programmable Logic Device (PLD)? What are the different types of PLDs? Explain the role of PLDs in Embedded System design.
11. What is the difference between PLD and ASIC?
12. What are the advantages of PLD over fixed logic device?
13. What are the different types of memories used in Embedded System design? Explain the role of each.
14. What are the different types of memories used for Program storage in Embedded System Design?
15. What is the difference between Masked ROM and OTP?
16. What is the difference between PROM and EPROM?
17. What are the advantages of FLASH over other program storage memory in Embedded System design?
18. What is the difference between RAM and ROM?
19. What are the different types of RAM used for Embedded System design?
20. What is memory shadowing? What is its advantage?
21. What is Sensor? Explain its role in Embedded System Design? Illustrate with an example.
22. What is Actuator? Explain its role in Embedded System Design? Illustrate with an example.
23. What is Embedded Firmware? What are the different approaches available for Embedded Firmware development?
24. What is the difference between General Purpose Processor (GPP) and Application Specific Instruction Set Processor (ASIP). Give an example for both.
25. Explain the concept of Load Store architecture and instruction pipelining.
26. Explain the operation of Static RAM (SRAM) cell.
27. Explain the merits and limitations of SRAM and DRAM as Random Access Memory.
28. Explain the difference between Serial Access Memory (SAM) and Random Access Memory (RAM). Give an example for both.
29. Explain the different factors that needs to be considered in the selection of memory for Embedded Systems.
30. Explain the different types of FLASH memory and their relative merits and de-merits.
31. Explain the different Input and output subsystems of Embedded Systems.
32. What is stepper motor? How is it different from ordinary dc motor?
33. Explain the role of Stepper motor in embedded applications with examples.
34. Explain the different step modes for stepper motor.
35. What is Relay? What are the different types of relays available? Explain the role of relay in embedded applications.
36. Explain the operation of the transistor based Relay driver circuit.
37. Explain the operation of a Matrix Keyboard.
38. What is Programmable Peripheral Interface (PPI) Device? Explain the interfacing of 8255 PPI with an 8bit microprocessor/controller.
39. Explain the different on-board communication interfaces in brief.
40. Explain the different external communication interfaces in brief.
41. Explain the sequence of operation for communicating with an I2C slave device.
42. Explain the difference between I2C and SPI communication interface.
43. Explain the sequence of operation for communicating with a 1-Wire slave device.
44. Explain the RS-232 C serial interface in detail.
45. Explain the merits and limitations of Parallel port over Serial RS-232 interface.
46. Explain the merits and limitations of IEEE1394 interface over USB.
47. Compare the operation of ZigBee and Wi-Fi network.
48. Explain the role of Reset circuit in Embedded System.
49. Explain the role of Real Time Clock (RTC) in Embedded System.
50. Explain the role of Watchdog Timer in Embedded System.

https://hemanthrajhemu.github.io

Lab Assignments

1. Write a 'C' program to find the *endianness* of the processor in which the program is running. If the processor is big endian, print "The processor architecture is Big endian", else print "The processor architecture is Little endian" on the console window. Execute the program separately on a PC with Windows, Linux and MAC operating systems.
2. Draw the interfacing diagram for connecting an LED to the port pin of a microcontroller. The LED is turned ON when the micro controller port pin is at Logic '0'. Calculate the resistance required to connect in series with the LED for the following design parameters.
   (a) LED voltage drop on conducting = 1.7V
   (b) LED current rating = 20 mA
   (c) Power Supply Voltage = 5V
3. Design an RC (Resistor Capacitor) based reset circuit for producing an active low Power-On reset pulse of width 0.1 milli seconds.
4. Design a zener diode and transistor based brown-out protection circuit with active low reset pulse for the following design parameters.
   (a) Use BC327 PNP transistor for the design.
   (b) The supply voltage to the system is 5V.
   (c) The reset pulse is asserted when the supply voltage falls below 4.7V.

https://hemanthrajhemu.github.io