# Module-5: Embedded System Components and Embedded Environment

**Embedded System Components:** Embedded Vs General computing system, History of embedded systems, Classification of Embedded systems, Major applications areas of embedded systems, purpose of embedded systems. Core of an Embedded System including all types of processor/controller, Memory.

**Chapter 01 & 02 from Text book 02**

**Embedded system Development Environment** – Block diagram (excluding Keil), Disassembler/decompiler, simulator, emulator and debugging techniques, target hardware debugging, boundary scan

**Chapter 13 from Text book 02(Shibu K V, "Introduction to Embedded Systems")**

# Embedded System Components

# Introduction

- Our day-to-day life is becoming-more and more **dependent** on "**embedded systems**" and digital techniques.
  - Home appliances, Vehicles, Industry control systems, etc.,
- Embedded technologies are bonding into our daily activities even **without our knowledge**.
- People experience the **power of** embedded systems and enjoy the features and comfort provided by them.
- Embedded systems are like **reliable servants-they** don't like to reveal their identity and neither they complain about their workloads.

# What is an Embedded System?

# Embedded system

- An embedded system is a **small computer** that's part of a larger mechanical or electrical system, performing a "**specific task**."
- Embedded systems can be **independent** or part of a larger system, and can range in complexity from a single microcontroller to a network of processors.

# What is an Embedded System?

- An embedded system is an electronic/electro-mechanical system designed to perform a **specific function**
- It 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 **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/provocations, meeting the computational deadlines, power efficiency, limited memory capability,* etc.

| 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 |

| General Purpose Computing System | Embedded System |
| --- | --- |
| • Applications are **alterable** (programmable) by the user<br>• i.e., the end user can 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 system 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** |

| General Purpose Computing System | Embedded System |
| --- | --- |
| • **Less tailored or not 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** |

# History of Embedded Systems

- Embedded systems **were in existence** even **before** the IT revolution.
  - Built around the old **vacuum tube and transistor** technologies.
- **Advances** in **semiconductor and nanotechnology** and IT revolution gave way to the development of miniature/tiny embedded systems.
- The first recognised modern embedded system is the **Apollo Guidance Computer (AGC)** developed by the MIT Instrumentation Laboratory for the lunar project/expedition carried out by a team..

# History of Embedded Systems

- AGC Specification:
  - It had 36K words of **fixed memory** and 2K words of **erasable memory.**
  - The **clock frequency** of was 1.024 MHz and it was derived from a 2.048 MHz crystal clock.
- The first mass-produced embedded system was the **Autonetics D-17** guidance computer for the **Minuteman-I missile** in **1961**.
  - It was built using discrete transistor logic and a hard-disk for main memory.
- The first **integrated circuit** was produced in September **1958** and computers using them began to appear in **1963**.

# Classification of Embedded Systems

- There are 4 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**

# 1. Classification Based on Generation

- First Generation - 1960
- Second Generation - 1970
- Third Generation - 1980
- Fourth Generation - 1990
- Next Generation

# Classification Based on Generation (continued)

- First Generation (1970-1990)
  - Early embedded systems were built around **8-bit microprocessors** like **8085 and Z80 and 4-bit microcontrollers**.
  - **Simple** in hardware circuits with firmware developed in assembly code.
  - These systems used individual components like **transistors and resistors.**

# Classification Based on Generation (continued)

- **First Generation examples:**
  - Digital telephone keypads, stepper motor control units, etc.
  - D-17B computer, developed by Autonetics in 1965 for the Minuteman I missile guidance system.
  - The first modern, real-time embedded computing system 1960:
    - The Apollo Guidance Computer (AGC), developed in the 1960s by Dr. Charles Stark Draper at MIT for the Apollo Program.

# Classification Based on Generation (continued)

- Second Generation (1970s)
  - Embedded systems built around 16-bit microprocessors and 8-bit or 16-bit microcontrollers.
  - Instruction set were much more complex and powerful than the first generation.
  - **It was powerful and intricate systems than the first generation.**
  - Some of the second generation embedded systems contained embedded operating systems for their operation.
  - E.g.: Data acquisition systems, SCADA systems(Supervisory Control And Data Acquisition, etc.

# Classification Based on Generation (continued)

- **Third Generation 1980s**
  - Embedded systems built around **32-bit microprocessors** and **16-bit microcontrollers.**
  - Application and domain specific processors/controllers
    - Such as Digital Signal Processors (DSP) and Application Specific Integrated Circuits (ASICs) came into picture.
  - The instruction set of processors became more complex(CISC) and powerful and the concept of instruction pipelining also evolved.

# Classification Based on Generation (continued)

- Third Generation 1980s
  - **Dedicated embedded real time** and general purpose operating systems entered into the embedded market.
  - Embedded systems spread its ground to areas
  - Such as robotics, media, industrial process control, networking, etc.

# Classification Based on Generation (continued)

- **Fourth Generation 1990s**
  - The advent of System on Chips (SoC), reconfigurable processors and multicore processors are bringing these features
    - **High performance,**
    - **Tight integration and**
    - **Miniaturization into the embedded device market.**
  - The SoC technique implements a total **system on a chip** by implementing different functionalities with a processor core on an integrated circuit.

# Classification Based on Generation (continued)

- **Fourth Generation 1990s**
  - Modern advancements in microprocessors and microcontrollers, along with new concepts.
  - Such as **System-on-Chip (SoC), reconfigurable**, **multicore** processors, and **coprocessors**, have significantly **improved the performance of embedded systems.**
  - These systems often **use high-performance real-time operating systems** for their operation.
  - E.g.: Smart phone devices, Mobile Internet Devices (MIDs), etc.

# Classification Based on Generation (continued)

- **Next Generation**
  - The processor and embedded market is **highly dynamic** and **demanding**.
  - The next generation embedded systems are **expected to meet growing demands in the market.**
  - Communication with server on the cloud computing

## Demands in the market

- **Security:**
  - Hardware-level encryption and multi-factor authentication to protect data and systems from breaches
- **AI and Machine Learning:**
  - Integration of AI and Machine Learning (ML) for predictive maintenance, data analysis, and real-time decision-making
- **Edge computing:**
  - Network connectivity for new functions related to control, device status, and data
- **Sustainability:**
  - Energy-efficient designs and power optimization to align with the global emphasis on environmentally conscious technology solutions (green computing)

# 2. Classification Based on Complexity and Performance

- These systems can range from **simple to highly sophisticated designs**, depending on their memory, processing power, and applications.
- The three main categories of embedded systems based on complexity include:
  - **Small-Scale Embedded Systems**
  - **Medium-Scale Embedded Systems**
  - **Large-Scale Embedded Systems**

# Classification Based on Complexity and Performance (continued)

- **Small-Scale Embedded Systems**
  - These systems use either a single 8 or 16-bit microprocessor or controller.
  - They have limited memory and processing power.
  - They also have relatively lower hardware and software complexities.
  - They may or may not contain an operating system for their functioning.
  - Usually built around low performance and low cost 8-bit or 16-bit microprocessors/microcontrollers.

# Classification Based on Complexity and Performance (continued)

- **Small-Scale Embedded Systems**
  - They are majorly found in gadgets like electronic toys, smartcards, etc.
  - The main programming tools used for these systems include an editor, cross assembler, and integrated development environment (IDE).
  - Simple in application needs and the performance requirements are **not time critical.**
  - E.g.: An electronic toy, smart cards, simple home appliances, etc.,

# Classification Based on Complexity and Performance

- **Medium-Scale Embedded Systems**
  - These come with 16-bit or 32-bit microprocessors or controllers, ASICs(Application Specific Integrated Circuit), or DSPs.
  - Also, they have a fair amount of memory and processing power.
  - These systems have **complexities** in both hardware and software and run on a real-time operating system **(RTOS).**
  - They are often employed in home appliances, medical devices, and automotive systems.
  - The main programming tools used for these systems include C, C++, JAVA, Visual C++, RTOS, etc.

# Classification Based on Complexity and Performance

- **Medium-Scale Embedded Systems**
  - Slightly complex in hardware and firmware (software) requirements.
  - Usually built around **medium performance**, low cost 16-bit or 32- bit microprocessors/microcontrollers or digital signal processors.
  - Usually contain an **embedded operating system** (either general purpose or real time operating system) for functioning.

# Based on Complexity and Performance (continued)

- **Large-Scale Embedded Systems**
  - These systems have **highly complex hardware and software**, like ones built on 32-bit or 64-bit **RISC processors**,
  - System-on-Chip (SoC), processors/controllers, and scalable and configurable processors.
  - A **high-performance real-time OS** is usually required for task scheduling, prioritization, and management.
  - They are used **in innovative applications** that demand hardware and software design.
  - Such as **aerospace** technologies, **industrial** automation, and **wireless communication** systems.

# Classification Based on Complexity and Performance (continued)

- Highly complex in hardware and firmware (software) requirements.
- They are employed in **mission critical applications** demanding **high performance**.
- Decoding/encoding of media, cryptographic function implementation, etc. are examples of processing requirements which can be implemented using a co-processor/hardware accelerator.

# 3. Classification Based on Deterministic Behaviour

- Applicable for **'Real Time'** systems. The application/task execution behaviour can be either **deterministic** or **non-deterministic.**
- Based on the execution behaviour, real time embedded systems are classified into
    - **Hard Real Time systems**
    - **Soft Real Time systems**.

# Classification Based on Deterministic Behaviour

- **Soft Real-Time Systems**
- These systems **do not enforce strict timing constraints** for task deadlines.
- While they still require a response time to specific events, missing a deadline is acceptable.
- Examples include **ATMs** and **multimedia systems**.
- In these cases, a late answer is still an acceptable answer.

# Classification Based on Deterministic Behaviour

- **Hard Real-Time Systems**
- These systems **demand strict adherence to their timing constraints**, as failing to meet the required response time can result in severe consequences or system failure.
- So, for these systems, a late answer is always considered a wrong answer.
- Examples are **airbag control systems** and **antilock braking systems** in vehicles, Military systems, etc.,

# 4. Classification Based on Triggering

- Embedded systems which are '**Reactive**' in nature (like process control systems in industrial control applications) can be classified based on the **trigger**.
- These systems are of the following two types:
    - **Event-triggered systems**
    - **Time-triggered systems**

# Classification Based on Triggering

- **Event-triggered systems**
- These systems **rely on specific external events** or **activities** to **initiate tasks.**
- They respond to changes in variables
- Such as temperature, pressure, or user inputs, making them more adaptive to their environment.
- They are particularly suited for applications requiring immediate responses or real-time reactions, such as intrusion detection systems, medical device monitoring, etc.

# Classification Based on Triggering

- **Time-triggered systems**
- These systems are **activated or initiated at pre-determined intervals or at a specific point in time**.
- In these systems, the tasks are scheduled to be executed **based on a predefined time or periodic timer**, which ensures the timely and consistent execution of processes.
- They are used for data collection, industrial systems with routine control loops, and scheduled car maintenance systems.

# Major Application Areas of Embedded Systems

1. Consumer electronics: Camcorders, cameras, etc.
2. Household appliances: Television, DVD players, washing machine, refrigerators, microwave oven, etc.
3. Home automation and security systems: Air conditioners, sprinklers, intruder detection alarms, closed circuit television (CCTV) cameras, fire alarms, etc.
4. Automotive industry: Anti-lock braking systems (ABS), engine control, ignition systems, automatic navigation systems, etc.
5. Telecom: Cellular telephones, telephone switches, handset multimedia applications, etc.

# Major Application Areas of Embedded Systems (continued)

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. Measurements & Instrumentation: Digital multimeters, digital CROs, logic analyzers, PLC systems, etc.
10. Banking & Retail: Automated teller machines (ATM) and currency counters, point of sales (POS), etc.
11. Card readers: Barcode, smart card readers, hand held devices, etc.

# Embedded  Systems

- What is an Embedded systems?
- Embedded Vs General computing system,
- History of embedded systems,
- Classification of Embedded systems.
    1. Based on generation
    2. Complexity and performance requirements
    3. Based on deterministic behaviour
    4. Based on triggering
- Major applications areas of embedded systems

# Purpose of Embedded Systems

- 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

# Purpose of Embedded Systems (continued)

- **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).
  - The collected data may be **stored or transmitted** or it may be **processed** or it may be **deleted** instantly after giving a meaningful representation.

# Purpose of Embedded Systems (continued)

- 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.

# Purpose of Embedded Systems (continued)

- **Data Communication**
  - Embedded data communication systems are **deployed** in applications ranging from complex satellite communication systems to simple home networking systems.
  - The **transmission** is achieved either by a wire- line medium or by a wireless medium.
  - The data collecting embedded terminal itself can **incorporate** data communication units like
    - **wireless modules** (Bluetooth, ZigBee, Wi-Fi, EDGE, GPRS, etc.) or
    - **wire-line modules** (RS- 232C, USB, TCP/IP, PS2, etc.).

# Purpose of Embedded Systems (continued)

- 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.

Fig: A wireless network router for data communication

# Purpose of Embedded Systems (continued)

- **Data (Signal) Processing**
  - 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.

# Purpose of Embedded Systems (continued)

- 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.

# Purpose of Embedded Systems (continued)

- **Monitoring**
  - Almost embedded products coming under the **medical domain** are used for monitoring.
  - 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.

# Purpose of Embedded Systems (continued)

- 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.

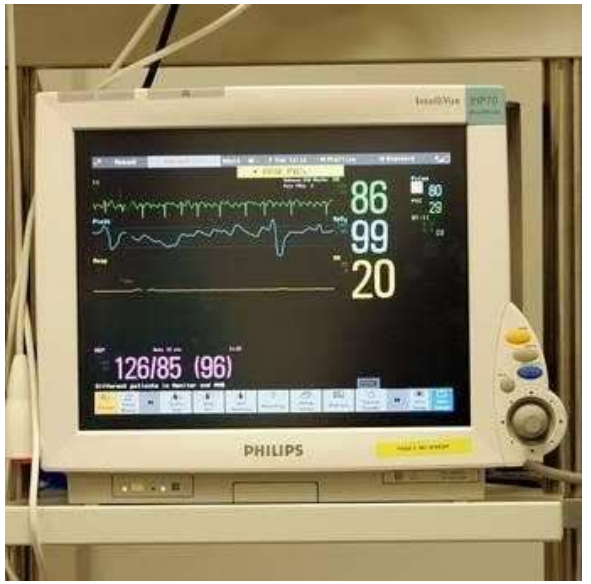



Fig: A patient monitoring system for monitoring heartbeat

# Purpose of Embedded Systems (continued)

- **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 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.

# Purpose of Embedded Systems (continued)

- An Air Conditioner System used to control the room temperature to a **specified limit** is a typical example for embedded system for **control purpose**.
- An air conditioner contains a room **temperature-sensing element (sensor)** which may be a **thermistor** and a handheld unit for setting up (feeding) the **desired temperature.**

# Purpose of Embedded Systems (continued)

- 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.


- A mobile phone is an example for embedded system with an application-specific user interfaces.

# A Typical Embedded System

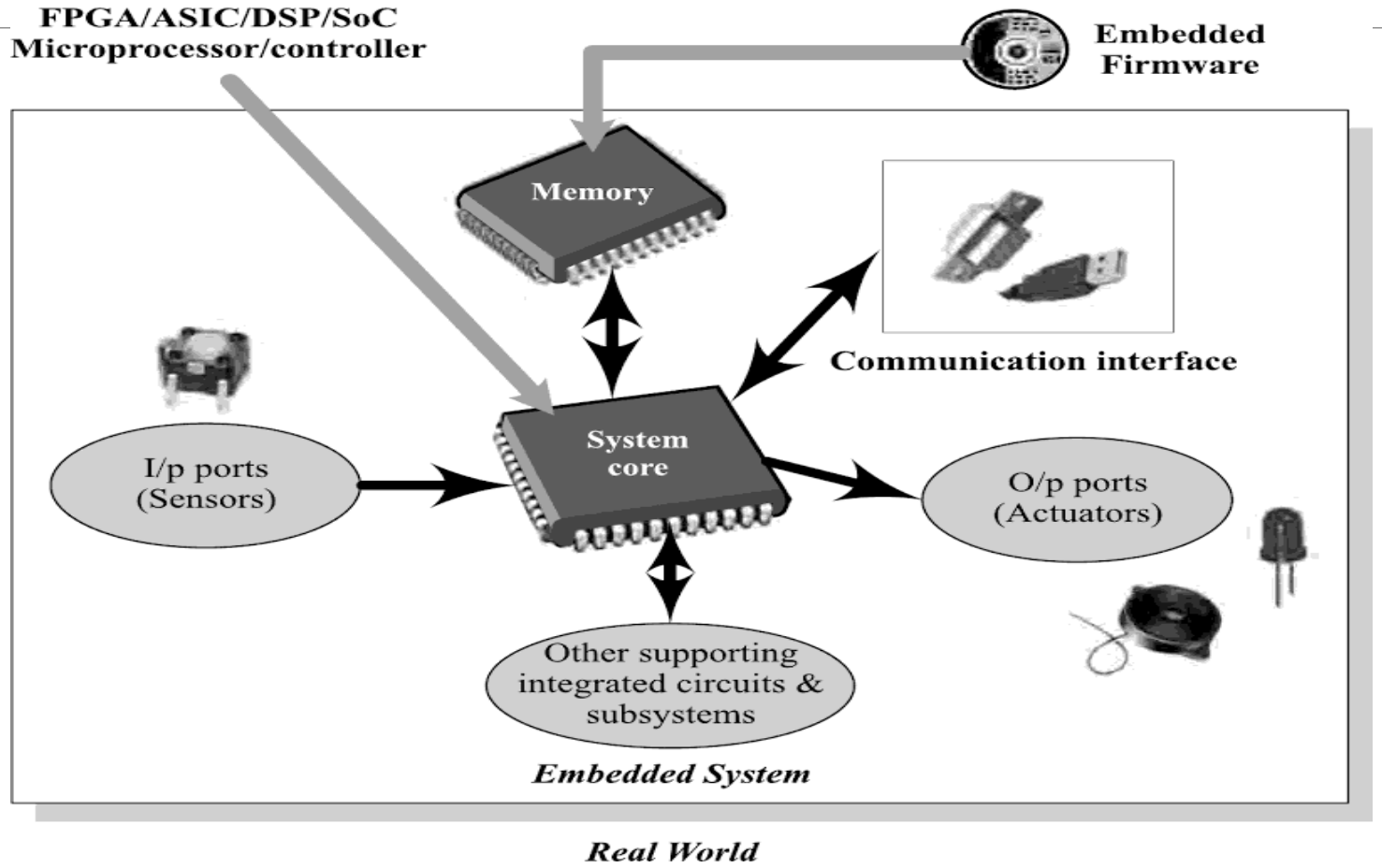



**Fig: Elements of an Embedded System**

➢A typical embedded system (Fig. 2.1) contains a single chip controller , it can be

Microprocessor' (e.g. Intel 8085)

microcontroller (e.g. Atmel AT89C51)

Field Programmable Gate Array (FPGA) device

(e.g. Xilinx Spartan)

Digital Signal Processor (DSP)

(e.g. Blackfin® Processors from Analog Devices)

Application Specific Integrated Circuits

(e.g. ADE7760 Single Phase Energy Metreing IC)

- 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

- Where as LEDs, liquid crystal displays, electric buzzers, etc. are examples for common user interface output devices for a typical embedded system.

- The Memory of the system is responsible for holding the control algorithm and other important configuration details.
- The most **common types of memories** used in embedded systems for control algorithm storage are **ROM, PROM, UVEPROM, EEPROM and FLASH.**
- Depending on the control application, the memory size may **vary** from a few bytes to megabytes.
- 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.

# 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
     a) Microprocessors
     b) Microcontrollers
     c) Digital Signal Processors
  2. Application Specific Integrated Circuits (ASICs)
  3. Programmable Logic Devices (PLDs)
  4. Commercial off-the-shelf Components (COTS)

# 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.
- **Domains** which require signal processing such as speech coding, speech recognition, etc. make use of special kind of digital signal processors.

# a) 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.
- 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, AMD, Freescale, IBM, TI, Cyrix, Hitachi, NEC, LSI Logic, etc. are the key players in the processor market.

# General Purpose Processor (GPP) vs. Application-Specific Instruction Set Processor (ASIP)

## General Purpose Processor (GPP)

- A General Purpose Processor or GPP is a processor designed for general computational tasks.
  - The processor running inside laptop or desktop is a typical example for general purpose processor.
- Due to the high volume production, the per unit cost for a chip is low.
- A typical general purpose processor contains an Arithmetic and Logic Unit (ALU) and Control Unit (CU).

# General Purpose Processor (GPP) vs. Application-Specific Instruction Set Processor (ASIP)

## Application-Specific Instruction Set Processor (ASIP)

- ASIPs are processors with **architecture** and **instruction set** **optimised** to specific-domain/application requirements (network processing, automotive, telecom, media applications, digital signal processing, control applications, etc.
- 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.

# b). Microcontrollers

- A Microcontroller is a **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.
- A microcontroller contains all the necessary functional blocks for independent working.
  - Have greater place in embedded domain **in place of microprocessors**.
  - They are **cheap, cost effective** and are readily available in the market.
  - Atmel, Texas Instruments, Toshiba, Philips, Freescale, NEC, Zilog, Hitachi, Mitsubishi, Infineon, ST Micro Electronics, National, Microchip, Analog Devices, Daewoo, Intel, Maxim, Sharp, Silicon Laboratories, TDK, Triscend, Winbond, etc. are the key players in the microcontroller market.

# c). Digital Signal Processors – domain specific

- **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

# c). Digital Signal Processors

- 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.

# DSP

- 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.

# DSP

- Computational Engine incorporates many **specialized 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

# 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 we need to learn only a few instructions,
- Whereas for a CISC processor we need to learn more number of instructions.

# RISC vs. CISC Processors/Controllers

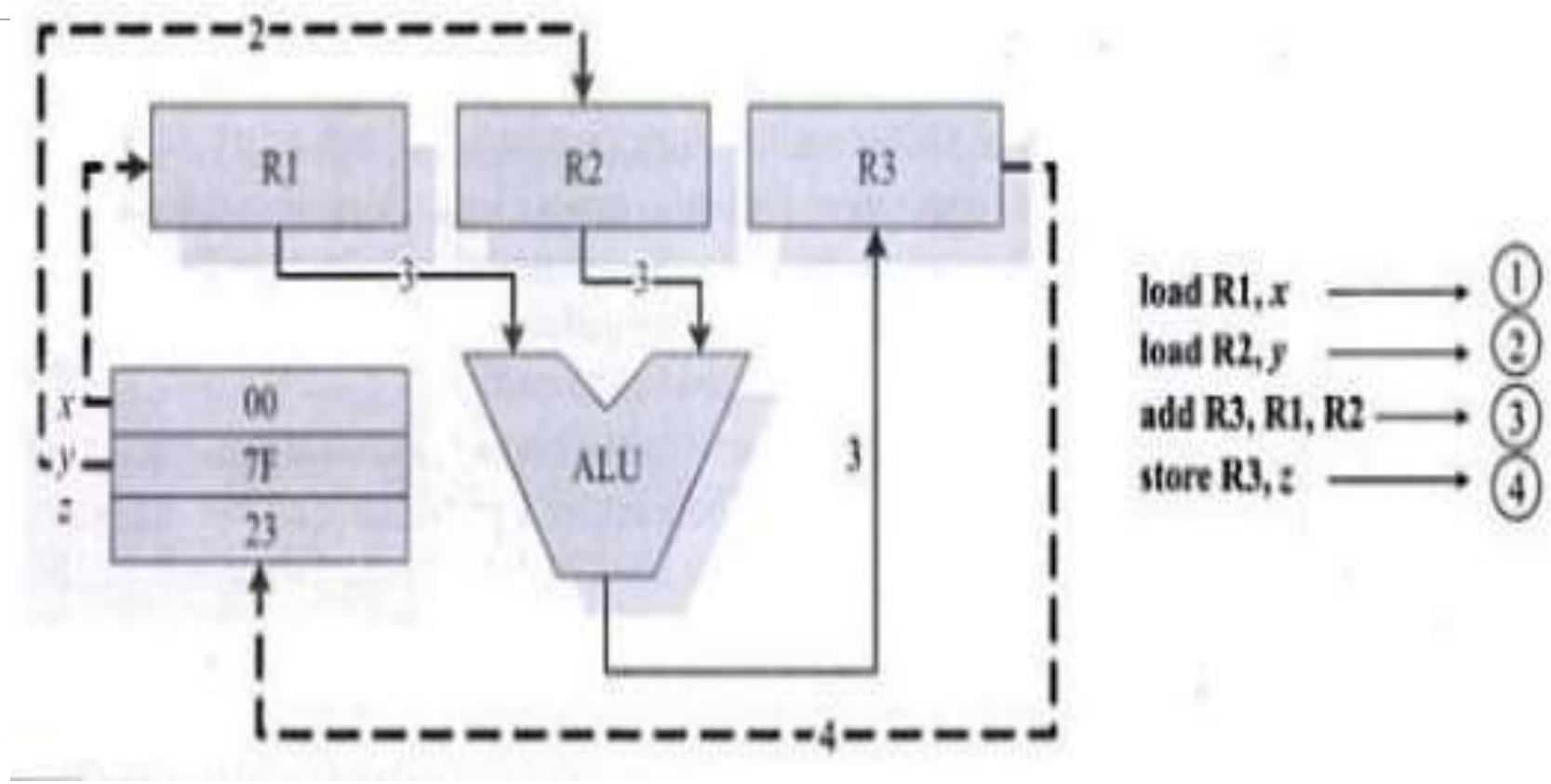
- Atmel AVR microcontroller is an example for a RISC processor and its instruction set contains only 32 instructions.
- The original version of 8051 microcontroller (e.g. AT89C51) is a CISC controller and its instruction set contains 255 instructions.
- Remember it is not the **number of instructions** 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

| 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 |

# Load Store Operation and Instruction Pipelining

- We know that, the RISC processor instruction set is **orthogonal**, meaning it **operates on registers.**
- The memory access related operations are perforated 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 **store instruction** stores data from a specified register to a specified memory location.
- The concept of Load Store Architecture illustrated with the following example

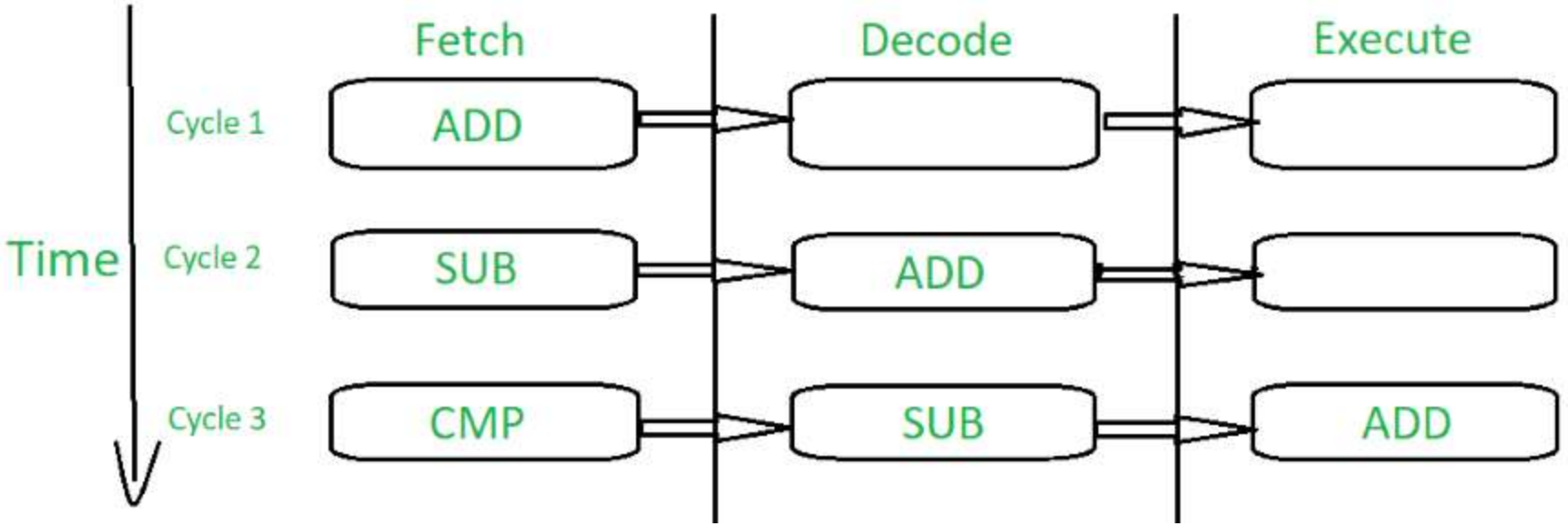
# The concept of Load Store Architecture



- 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**
- The first instruction **load Rl, 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 **add R3, Rl, R2 adds the content of registers Rl 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.
- 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 utilize it for an instruction fetch, the processing speed can be increased.
- In its simplest form instruction **pipelining refers to the overlapped execution of instructions.**
- Depending on the stages involved in an instuction (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.




The three-stage pipelining concept

# 4. 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.
- The circuits in a **fixed logic device are permanent**, they perform one function or set of functions—once manufactured, **they cannot be changed.**

# 4. Programmable Logic Devices

- 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.
  - Designers use 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 with suitable tools.

# Programmable Logic Devices (continued)

- There are no NRE costs (Non Recurring Engineering costs are one-time costs that are incurred during the pre-production phase of a new product or product enhancement.
- They include the costs of research, design, development, testing, and other functions required to create the product.
- The final design is completed much faster than that of a custom, fixed logic device.

# Programmable Logic Devices (continued)

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

# 5.Commercial Off-the-Shelf Components (COTS)

- A Commercial Off-the-Shelf (COTS) product is one which is used 'as-is'. (reusability)
- 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.

# 5.Commercial Off-the-Shelf Components (COTS)

- Advantages:
  - They are readily available in the market
  - Cheap means less expensive
  - Developer can cut down his development time to a great extent
  - Reduces the time to market

# Commercial Off-the-Shelf Components (COTS) (continued)

- 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.

# Commercial Off-the-Shelf Components (COTS) (continued)

- E.g.: The TCP/IP plug-in module available from various manufactures like 'WIZnet', 'Freescale', 'Dynalog', etc. are very good examples of COTS product.

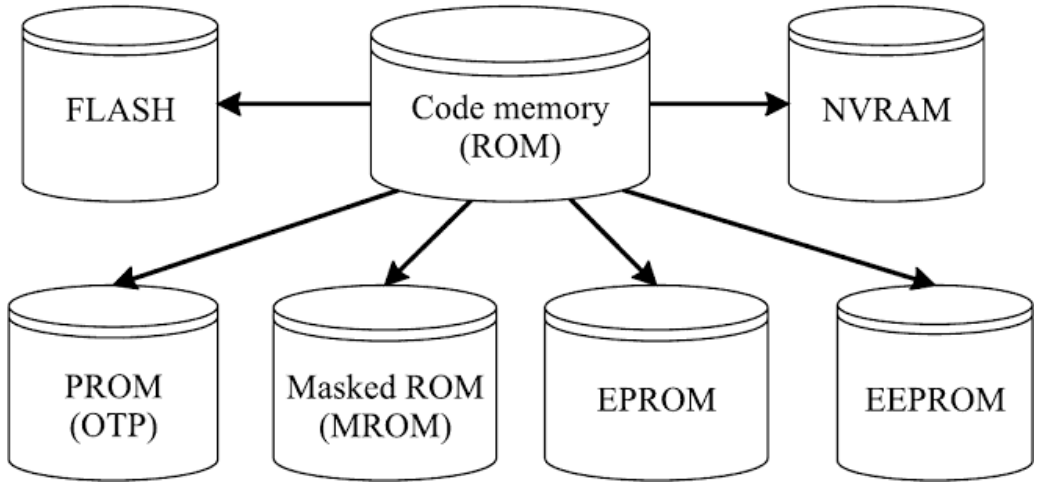
Fig: An example of a COTS product for TCP/IP plug-in from WIZnet (WIZnet NM7010A Plug in Module)

# 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.

# Program Storage Memory (ROM)

- The program memory or code storage memory of an embedded system stores the program instructions.
- The code memory retains its contents even after the power is turned off. It is generally known as non-volatile storage memory.
- It can be classified into different types as shown:



# 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.
  - Fast due to its resistive networking and switching capabilities.
- In typical implementation, an SRAM cell (bit) is realised using six transistors (or 6 MOSFETs).
  - Four of the transistors are used for building the latch (flip-flop) part of the memory cell and two for controlling the access.

# Dynamic RAM (DRAM)

- Dynamic RAM stores data in the form of charge.
- They are made up of MOS transistor gates.
- Advantages – high density and low cost compared to SRAM.
- Disadvantage – 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.

# Dynamic RAM (DRAM)

- 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. |

# Memory According to the Type of Interface

- The interface (connection) of memory with the processor/controller can be of various types.
  - Parallel interface
  - Serial interface like I2C
  - SPI (Serial peripheral interface)
  - 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 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.

# Memory Selection for Embedded Systems (continued)

- There are two parameters for representing a memory:
  1. Size of the memory chip – Memory density expressed in terms of number of memory bytes per chip.
     - Memory chips come in standard sizes like 512 bytes, 1024 bytes (1 kilobyte), 2048 bytes (2 kilobytes), 4KB, 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, 512KB, 1024KB(1 megabytes), etc.
  2. Word size of the memory – The number of memory bits that can be read/written together at a time.
     - Word size can be 4, 8, 12, 16, 24, 32, etc.
     - The word size supported by the memory chip must match with the data bus width of the processor/controller.

# Sensors and Actuators

- 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.

# Sensors and Actuators (continued)

- A sensor is a transducer device that converts energy from one form to another for any measurement or control purpose.
  - E.g.: Temperature sensor, magnetic hall effect sensor, humidity sensor, etc.
- An actuator is a form of transducer device (mechanical or electrical) which converts signals to corresponding physical action (motion).
  - Actuator acts as an output device.
  - E.g.: Stepper motor

# The I/O Subsystem

- The I/O subsystem of the embedded system facilitates the interaction of the embedded system with the external world.
- 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.

# 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.
  - E.g.: 'Device ON', 'Battery low' or 'Charging of battery' conditions
- Light Emitting Diode is a p-n junction diode and it contains an anode and a cathode.
- For proper functioning of the LED, the anode is 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 to limit the current through the LED.
- The ideal LED interfacing circuit is shown in the figure.

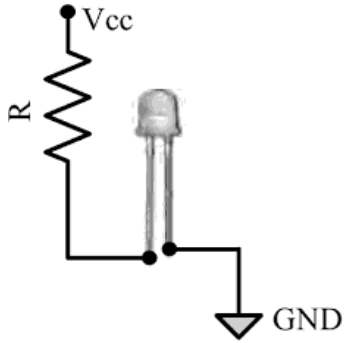


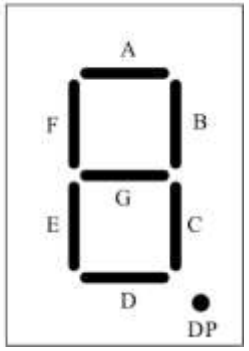

Fig: LED interfacing

# Light Emitting Diode (LED) (continued)

- 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.
    - 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.

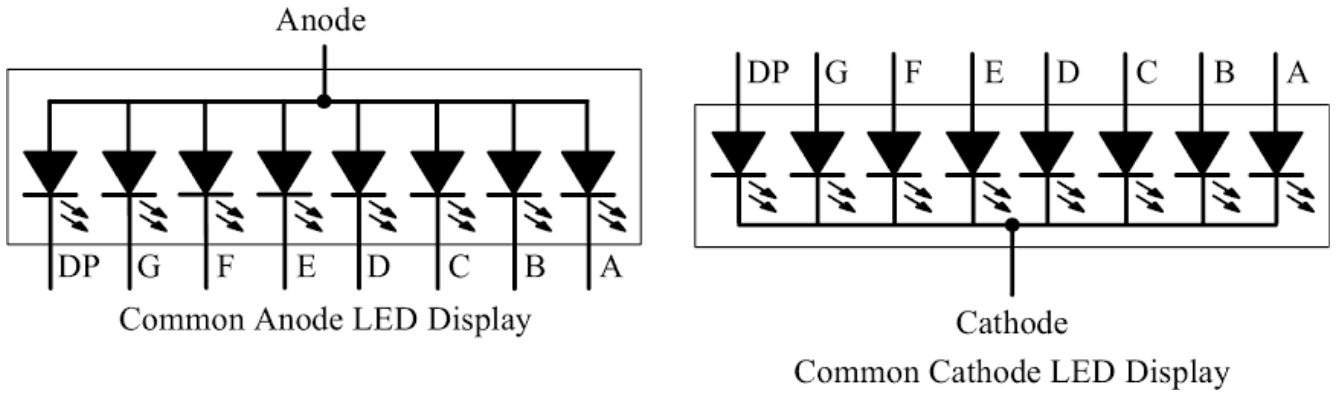
# 7-Segment LED Display

- The 7-segment LED display is an output device for displaying alpha numeric characters.
- It contains 7 LED segments arranged in a special form used for displaying alpha numeric characters and 1 LED used for representing 'decimal point' in decimal number 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.



# 7-Segment LED Display (continued)

- 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 cathodes of 8 LED segments are connected commonly.
- Figure illustrates the Common Anode and Cathode configurations.



# 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.
- Figure illustrates the functioning of an optocoupler device.

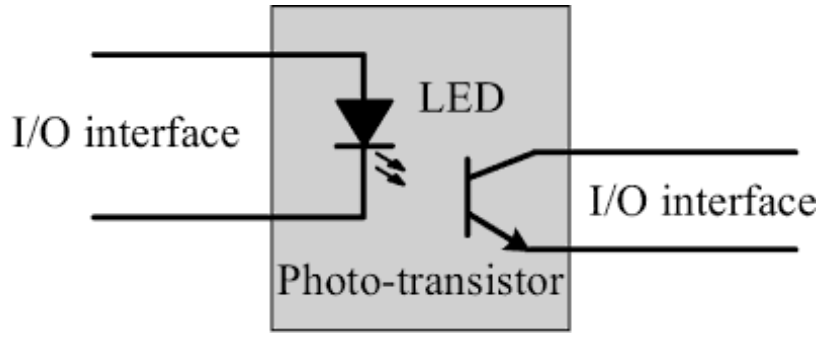



Fig: An optocoupler device

# Optocoupler (continued)

- 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.
- Optocoupler is available as ICs from different semiconductor manufacturers.
  - The MCT2M IC from Fairchild semiconductor is an example for optocoupler IC.

# Relay

- Relay is an electro-mechanical device.
- In embedded application, the Relay unit acts as dynamic path selector 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.

# Relay (continued)

- Relays are available in different configurations.
- Figure given below illustrates the widely used relay configurations for embedded applications.

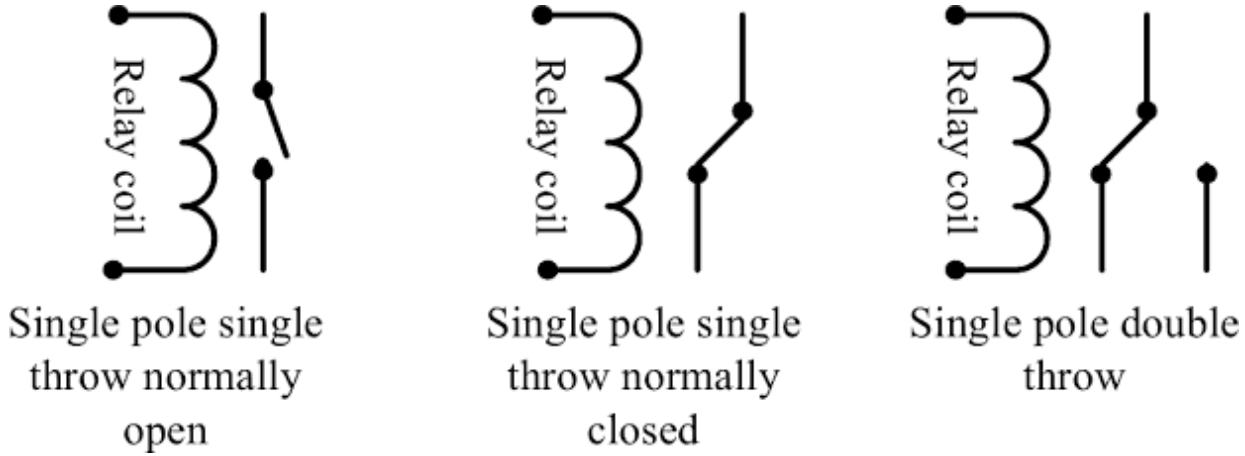



Fig: Relay configurations

# Relay (continued)

- 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 relay, the circuit is normally closed and it becomes open when the relay is energised.
- For Single Pole Double Throw configuration, there are two paths for information flow and they are selected by energising or de- energising the relay.

# 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 support 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'.

# 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 is released.

# 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:
  - Onboard Communication Interface (Device/board level communication interface)
    - E.g.: Serial interfaces like I2C, SPI, UART, 1-Wire, etc and parallel bus interface.
  - External Communication Interface (Product level communication interface)
    - E.g.: Wireless interfaces like Infrared (IR), Bluetooth (BT), Wireless LAN (Wi-Fi), Radio Frequency waves (RF), GPRS, etc. and wired interfaces like RS-232C/RS-422/RS-485, USB, Ethernet IEEE 1394 port, Parallel port, CF-II interface, SDIO, PCMCIA, etc.

# Onboard Communication Interfaces

- An embedded system is a combination of different types of components (chips/devices) arranged on a printed circuit board (PCB).
- Onboard Communication Interface refers to the different communication channels/buses for interconnecting the various integrated circuits and other peripherals within the embedded system.
- E.g.: Serial interfaces like I2C, SPI, UART, 1-Wire, etc and parallel bus interface

# Inter Integrated Circuit (I2C) Bus

- The Inter Integrated Circuit Bus (I2C Pronounced 'I square C') is a synchronous bi-directional half duplex two wire serial interface bus.
  - (Half duplex - one-directional communication at a given point of time)
- 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:
  - Serial Clock (SCL line) – responsible for generating synchronisation clock pulses
  - Serial Data (SDA line) – responsible for transmitting the serial data across devices

# Inter Integrated Circuit (I2C) Bus (continued)

- 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' or 'Slave'.
  - 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 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.

# Inter Integrated Circuit (I2C) Bus (continued)

- The following bus interface diagram illustrates the connection of master and slave devices on the I2C bus.

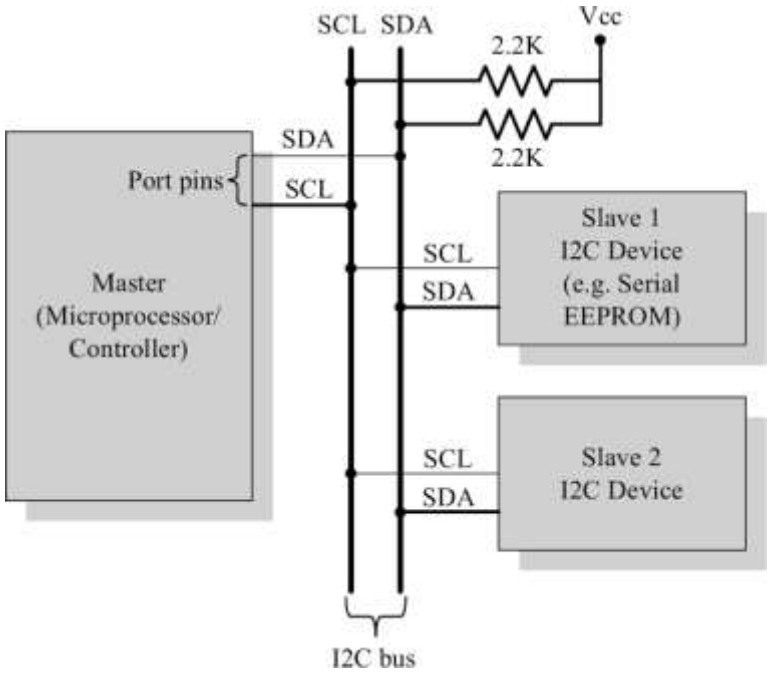



Fig: I2C Bus Interfacing

# Inter Integrated Circuit (I2C) Bus (continued)

- 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

# Inter Integrated Circuit (I2C) Bus (continued)

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 8 bit 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)

# Inter Integrated Circuit (I2C) Bus (continued)

- I2C bus supports three different data rates:
  - Standard mode (Data rate up to 100kbits/sec (100 kbps))
  - Fast mode (Data rate up to 400kbits/sec (400 kbps))
  - High speed mode (Data rate up to 3.4Mbits/sec (3.4 Mbps))

# 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.
  - There can be more than one masters, but only one master device can be active at any given point of time.
- 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

# Serial Peripheral Interface (SPI) Bus (continued)

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

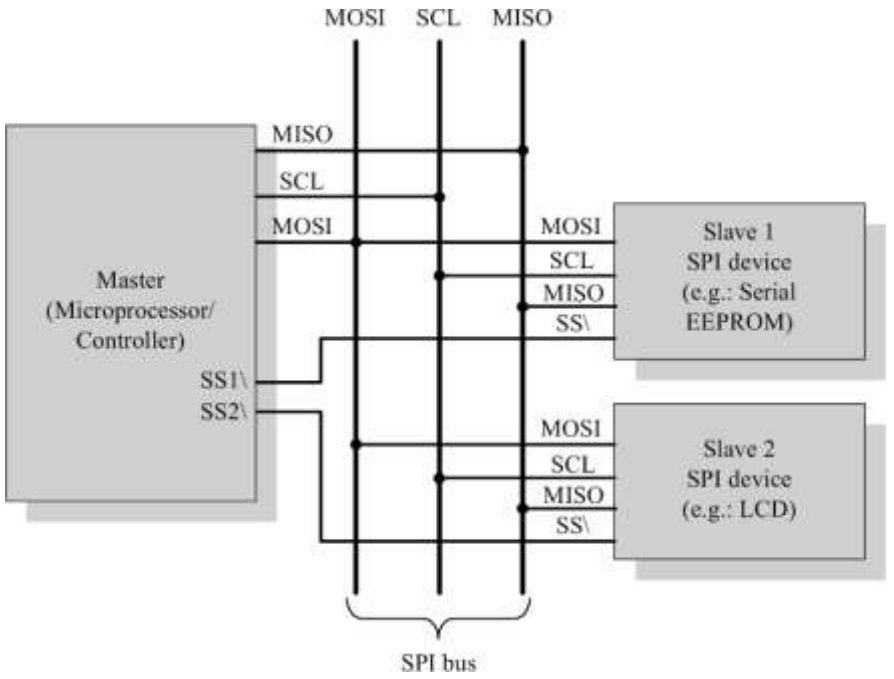



Fig: SPI Bus Interfacing

# Serial Peripheral Interface (SPI) Bus (continued)

- 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 control register holds the various configuration parameters like master/slave selection for the device, baud rate selection for communication, clock signal control, etc.
  - The status register holds the status of various conditions for transmission and reception.

# Universal Asynchronous Receiver Transmitter (UART)

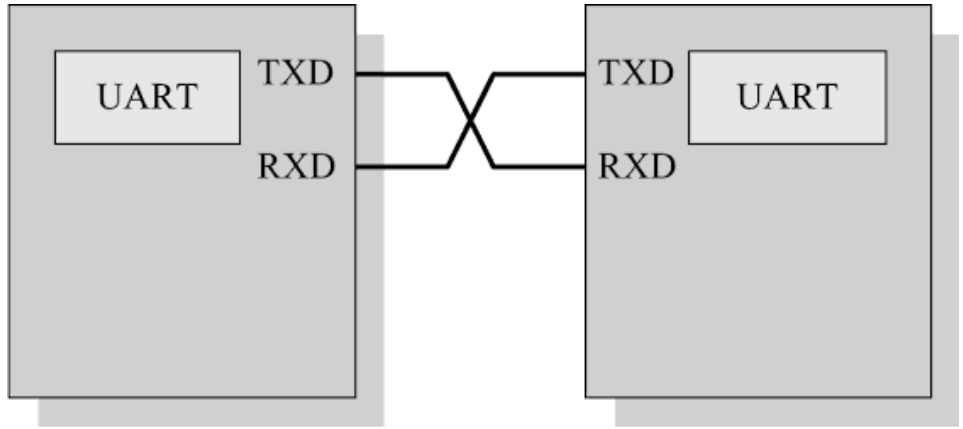
- Universal Asynchronous Receiver Transmitter (UART) based data transmission is an asynchronous form of serial data transmission.
- It doesn't require a clock signal to synchronise the transmitting end and receiving end for transmission.
- Instead it relies upon the pre-defined agreement between the transmitting device and receiving device.

# Universal Asynchronous Receiver Transmitter (UART) (continued)

- 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.

# Universal Asynchronous Receiver Transmitter (UART) (continued)

- Figure illustrates the UART interfacing.




TXD: Transmitter line
RXD: Receiver line

Fig: UART Interfacing

# 1-Wire Interface

- 1-wire interface is an asynchronous half-duplex communication protocol developed by Maxim Dallas Semiconductor.
- 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.

# 1-Wire Interface (continued)

- The bus interface diagram shown in the figure illustrates the connection of master and slave devices on the 1-wire bus.

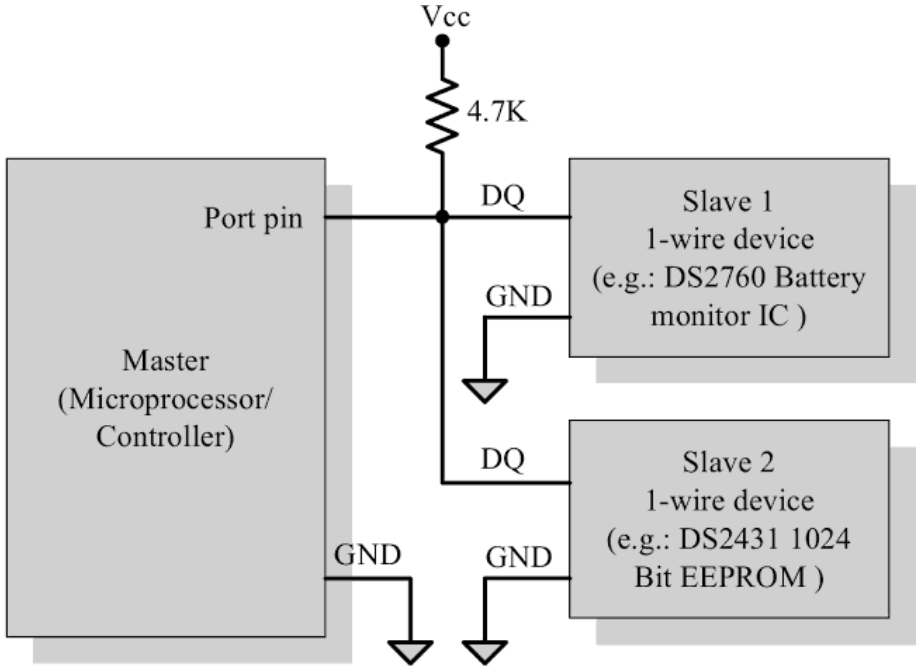



Fig: 1-Wire Interface Bus

# 1-Wire Interface (continued)

- 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 64 bit 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.

# 1-Wire Interface (continued)

- 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 (480 µs).
- 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 60 µs 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 (60 µs).

# Parallel Interface (continued)

- The bus interface diagram shown in the figure illustrates the interfacing of devices through parallel interface.

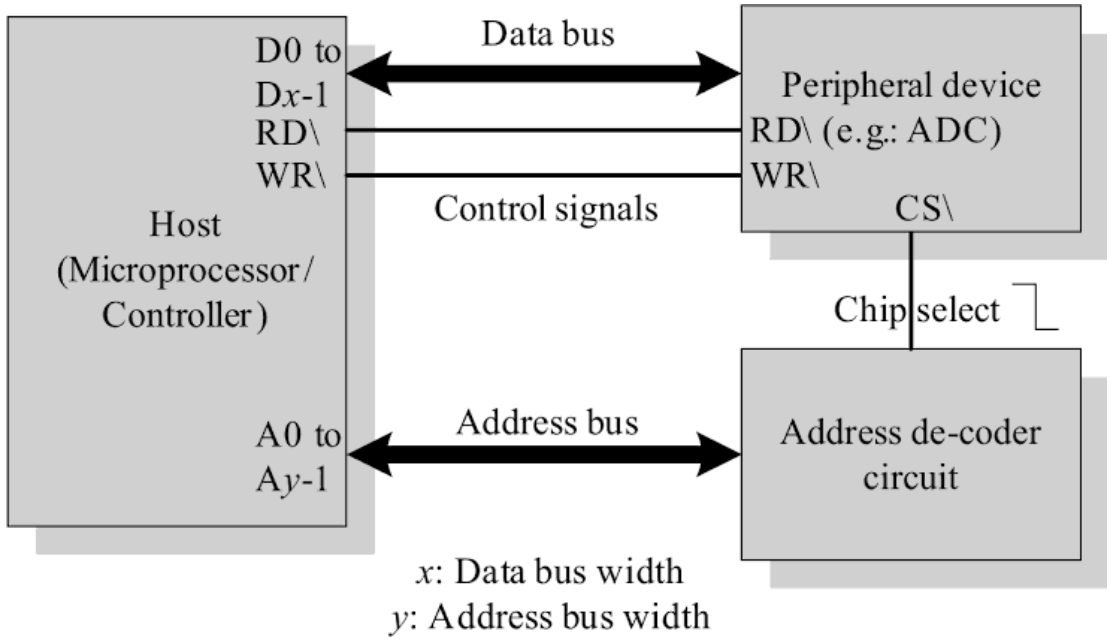



Fig: Parallel Interface Bus

# Parallel Interface (continued)

- 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 4 bit, 8 bit, 16 bit, 32 bit or 64 bit etc.
  - The bus width supported by the device should be same as that of the host processor.
- Parallel data communication offers the highest speed for data transfer.

# External Communication Interfaces

- External Communication Interface refers to the different communication channels/buses used by the embedded system to communicate with the external world.
- E.g.: RS-232 C & RS-485, Universal Serial Bus (USB), IEEE 1394 (Firewire), Infrared (IR), Bluetooth (BT), Wi-Fi, ZigBee, GPRS, etc.

# RS-232 C & RS-485 (continued)

- 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:
  - DB-9: 9-Pin connector
  - DB-25: 25-Pin connector.

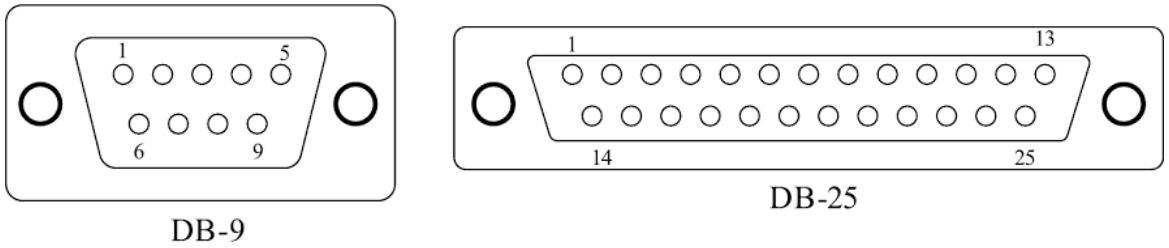



Fig: DB-9 and DB-25 RS-232 Connector Interface

# Universal Serial Bus (USB)

- Universal Serial Bus (USB) is a wired high speed serial bus for data communication.
- The first version of USB (USB 1.0) was released in 1995.
- 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.

# Universal Serial Bus (USB) (continued)

- Figure illustrates the star topology for USB device connection.

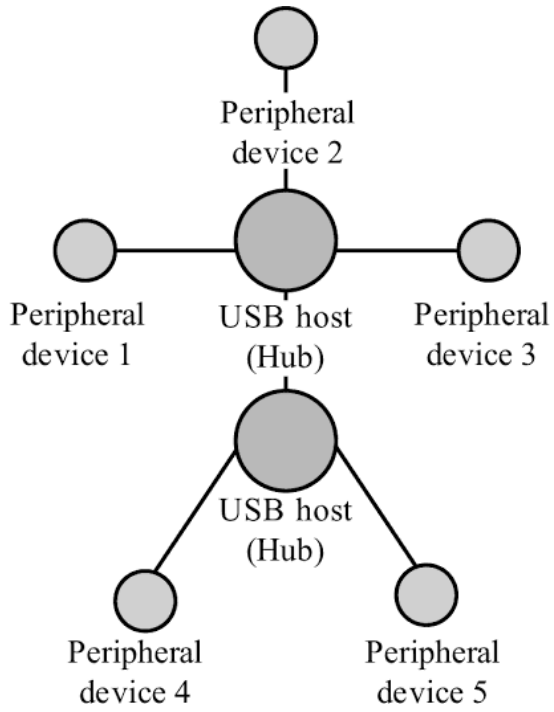



Fig: USB Device Connection topology

# Universal Serial Bus (USB) (continued)

- 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:
  - Open Host Control Interface (OHCI)
  - Universal Host Control Interface (UHCI)

# Universal Serial Bus (USB) (continued)

- 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.

# Universal Serial Bus (USB) (continued)

Type A Connector

Type B Connector

Type C Connector

# Universal Serial Bus (USB) (continued)

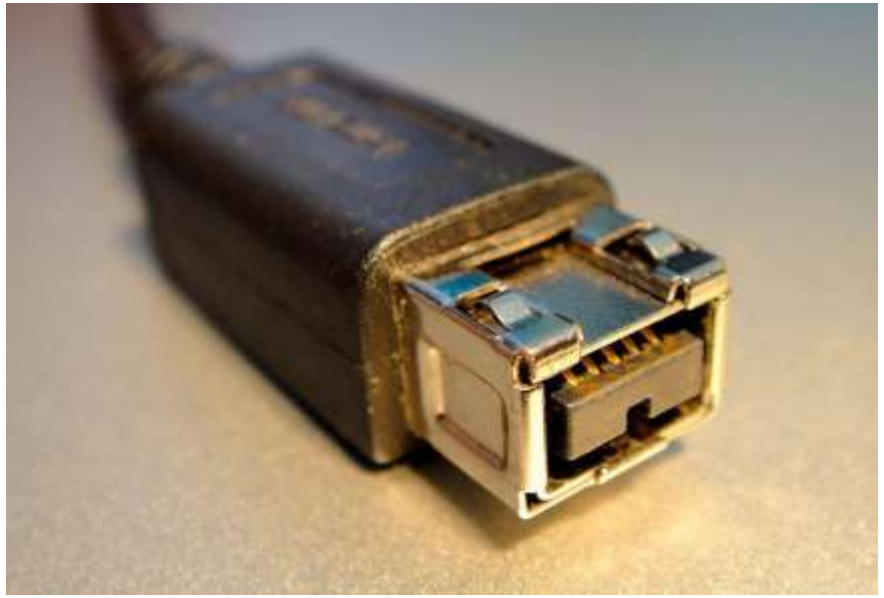
- USB.ORG is the standards body for defining and controlling the standards for USB communication.
- Presently USB supports four different data rates:
  - Low Speed (1.5Mbps) – USB 1.0
  - Full Speed (12Mbps) – USB 1.0
  - High Speed (480Mbps) – USB 2.0
  - Super Speed (4.8Gbps) – USB 3.0

# 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 1394 is available from various players with different names:
  - Firewire is the implementation from Apple Inc
  - i.LINK is the implementation from Sony Corporation
  - Lynx is the implementation from Texas Instruments

# IEEE 1394 (Firewire) (continued)

4-pin and 6-pin Connectors



9-pin Connector

# 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.
  - E.g.: The remote control of TV, VCD player, etc. works on Infrared.
- Infrared communication technique uses infrared waves of the electromagnetic spectrum for transmitting the data.
- It 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.

# Infrared (IrDA) (continued)

- 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.

# 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.
- It supports a data rate of up to 1Mbps and a range of approximately 30 feet for data communication.

# Bluetooth (BT) (continued)

- Bluetooth 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 Bluetooth communication
  - The protocol part is responsible 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.

# 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 (continued)

- Figure illustrates the typical interfacing of devices in a Wi-Fi network.

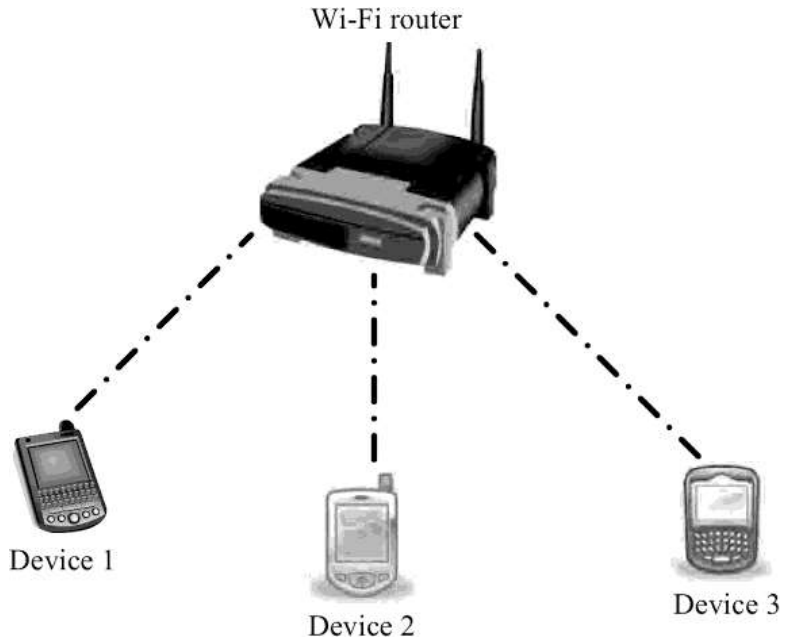



Fig: Wi-Fi Network

# 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 250 Kbps.

# ZigBee (continued)

- 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.

# ZigBee (continued)

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

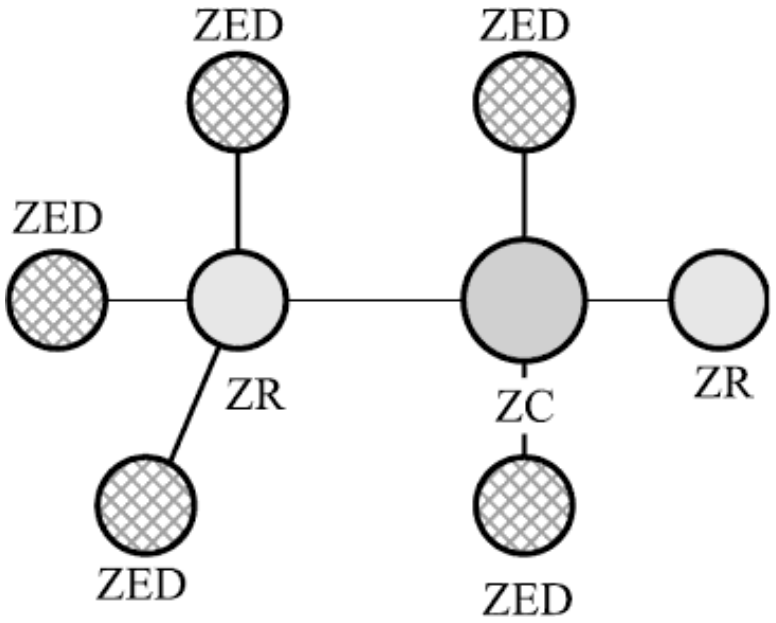



Fig: A ZigBee network model

# 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 supports a theoretical maximum transfer rate of 171.2 kbps.
- In GPRS communication, the radio channel is concurrently shared between several users instead of dedicating a radio channel to a cell phone user.

# References


1. Shibu K V, ***"Introduction to Embedded Systems"***, Tata McGraw Hill, 2009.
2. Raj Kamal, ***"Embedded Systems: Architecture and Programming",*** Tata McGraw Hill, 2008.