# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedded Systems

- **Suggested Textbooks:**
  - Raj Kamal, “Embedded systems Architecture, Programming and Design”, Third Edition, Tata McGraw Hill, 2017.
  - Rob Toulson and Tim Wilmshurst, “Fast and Effective Embedded Systems Design – Applying the ARM mbed”, Elsevier, 2017.
  - Arnold S. Berger, “Embedded Systems Design: An Introduction to Processes, Tools, and Techniques”, CRC Press, 2002.
  - K. Shibu, “Introduction to Embedded Systems”, Tata McGraw Hill, 2009.

- **Other sources**
  - Lecture notes
  - MOOC courses

# Embedded Systems - Modules

- Module 1 : Embedded Systems
- Module 2 : Building Embedded Hardware & Software
- Module 3 : Embedded Networking Technologies
- Module 4 : Embedded Real –Time Operating Systems
- Module 5 : Embedded C programming using 8051 & ARM

# Module 1

# Introduction to Embedded System

# Module 1 : Embedded Systems

- General Purpose vs Embedded Systems
- Embedded System - Classifications
- Embedded System - Characteristics
- Embedded System - Components
- Embedded System - Examples

# Consider the Evolution of Watches

# How about Refrigerators?

# Picture Frames

# Cameras

# Glasses

# Cars

# What is the trend?

# Physical Things Augmented with Computing/Communication

**OR**

# Computing/Communication "Embedded" into Physical Things

# So, What Is Embedded System?

**A computer, pretending not to be a computer**

(Stephen A. Edwards)

# What Is Embedded System?

- "An embedded system is an application that contains at least one programmable computer … and which is used by individuals who are unaware that the system is computer-based."
  -- Michael J. Pont, Embedded C

- **Programmable computers require programs**
  → embedded software

# What Is an Embedded System?

- Information processing systems <u>embedded</u> into a larger product [Peter Marwedel]
  - Main reason for buying is **not** information processing
- Any device that includes a programmable processor but is not itself a *general-purpose computer*

➔*Application-specific*: take advantage of application characteristics to optimize the design:

- Do not need all general-purpose bells and whistles

# Same Basics Inside, However

# Why Embedded Systems?

- After all, we can still make phone calls with

or

- Why embed a computer into a phone?

# Embedded vs Pure Hardware

- Many electronic products are implemented in pure hardware (ASICs, boards)
  - Lack of flexibility (changing standards, system revisions, bug fixes, extra functionalities)
  - Costly for specialized application-specific integrated circuits (ASICs) (M$ range, technology-dependent)

- Trend towards implementation in software running on embedded processors (or possibly FPGAs)
  - Commodity microprocessors
  - **Software-defined X**

(Peter Marwedel)

# Some Concepts to Clarify

- Embedded systems refer to not only small devices or gadgets

- But also large, complex systems requiring strict reliability and real-time responses

# Some Concepts to Clarify

- A product, e.g., video decoder, may be implemented using pure hardware or microprocessor + embedded software
  - A chip may be implemented using pure logic gates or a microprocessor + peripheral logic + software (SoC)

- An embedded system may or may not have OS
  - Simple systems may be implemented by a single program that runs continuously
  - Systems that need to control and respond to many activities may require an OS for management

# Embedded = Smart

- Computers embedded into objects
  - Augment objects with programmatic control, communication, sensing, and actuation
- Let the world know you:
  - Make physical objects/phenomena accessible to digital world
- Let you know the world:
  - Give intelligence/life to physical objects so that they can sense/react
  - Put a “robot” inside everything!

# A New Paradigm of Computing

# Future Embedded Systems

Smart Grid

Smart City

(Source: oncor.com, Prof. L.G. Chen)

# Future Embedded Systems

Smart Glasses

Self-driving Cars

Retinal Implant

# Implications

- Infinite opportunities, innovations, execution
- Integration: must know application domain, packaging, A/D, sensor/actuator, power, …
- System view

# About This Course

- Principles behind the design of the course:
  - Build the course around labs: Use labs to carry out the course contents. Labs are to develop a simple embedded system, from I/O device to system
  - Cover basic concepts in embedded system development: interrupt, clocking, I/O, real-time system, OS service, development tools
  - Term project development: innovation, development process, learning-by-doing

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Module 1

# Introduction to Embedded System

# Module 1 : Embedded Systems

- General Purpose vs Embedded Systems
- Embedded System - Classifications
- Embedded System - Characteristics
- Embedded System - Components
- Embedded System - Examples

# What is a System?

- A black box that takes the input, processes it and provides the required output
- A way of working, organizing or performing one or many tasks according to a fixed set of rules, program or plan
- Example
  - Time display system – A watch
  - Automatic cloth washing system – A washing machine
  - Photograph capturing system – A digital camera

# Embedded Systems

**Definition :**

- 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 specialized to the application domain.

# Embedded Systems

- Every embedded system consists of custom-built hardware built around a Central Processing Unit (CPU)
- Memory hardware also contains memory chips onto which the software is loaded.

# General Purpose Systems

## General Purpose System : Computer

- Components:
  - Microprocessor
  - Large Memory – 2 types
    - Primary Memory – RAM, ROM
    - Secondary Memory – Hard disks, SSDs, CD-ROMs
  - I/O units
    - Input units – Keyboard, mouse, digitizer,
    - Output units – LCD screen, printer, Speakers
  - Networking Units – Ethernet card, wireless card
  - Operating System – for general purpose use + application softwares

# Embedded Systems

- Components:
  - **Hardware**
    - Processor, Timers, Interrupt Controller,
    - I/O devices, Memories, Ports, etc.
  - **Application Software**
    - Which may perform series of tasks in concurrent manner
  - **Real Time Operating System**
    - Defines the way system works
    - Supervises application software
    - A small-scale embedded system may not need an RTOS

# Embedded Systems - Classification

# Embedded Systems - Classification

- Small scale embedded systems: Use 8-bit or 16-bit microcontrollers and have minimum hardware and software. They are so small and require little power they may be powered by a battery.
- Medium scale embedded systems: Use either one or a few 16 bit or 32-bit microcontrollers, RISCs or DSPs. The hardware and software is more complex as compared to small scale embedded systems.
- Sophisticated embedded systems: The most complex of the three classifications mentioned. Used for cutting-edge applications that need hardware and software co-design.

# Embedded Systems - Classification

- Small scale embedded systems:
  - 8051 Family
  - PIC 16F8X
  - Hitachi H8

- Mostly the C programing language is used in such embedded systems
- Examples
  - Bluetooth headphones
  - Digital pedometer

# Embedded Systems - Classification

- Medium scale embedded systems:
- Programming languages like Java, C, C++ are used to develop software for medium-scale embedded systems.
- Examples : ATM Machines

# Embedded Systems - Classification

- Sophisticated embedded systems:
- Sophisticated embedded systems, as the name suggests are highly advanced and developed in terms of hardware and software.
- Examples : Medical Systems, Vehicles, etc

# Embedded Systems - Classification

## Stand alone embedded systems

- Do not require a host system like a computer, it works by itself.
- It takes the input from the input ports either analog or digital and processes, calculates and converts the data and gives the resulting data through the connected device

# Embedded Systems - Classification

## Real time embedded systems

- Give the required o/p in a particular time.
- These types of embedded systems follow the time deadlines for completion of a task.
- Classified into two types such as soft and hard real time systems.

# Embedded Systems - Classification

## Networked Embedded Systems

- Related to a network to access the resources.
- The connected network can be LAN, WAN or the internet.
- This type of embedded system is the fastest growing area in embedded system applications.

# Embedded Systems - Classification

## Mobile embedded systems

- Used in portable embedded devices like cell phones, mobiles, digital cameras, mp3 players and personal digital assistants, etc.
- The basic limitation of these devices is having limited resources.

# Characteristics of Embedded Systems

- Perform a specific task
- Memory constrained
- Time constrained
- Low power consumption
- Highly reliable
- Fault tolerant architecture
- Robust
- Simplified user interface (generally no GUI)
- Less human interaction

# Characteristics of Embedded Systems

# Embedded Systems - Constraints

- Top three constraints
  - Available **system memory**
  - Available **processor speeds**
  - The need to limit **power consumptions** during system execution

- Other constraints
  - Performance
  - Size
  - Design and manufacturing costs

# Components of Embedded Systems

- Analog Components
  - Sensors, Actuators, Controllers, …
- Digital Components
  - Processor, Coprocessors
  - Memories
  - Controllers, Buses
  - Application Specific Integrated Circuits (ASIC)
- Converters – A2D, D2A, …
- Software
  - Application Programs
  - Exception Handlers

**Hardware**

**Software**

# Embedded Systems: Simplified Block Diagram

# Microprocessor vs Microcontroller

| | Microprocessor | Microcontroller |
|---|---|---|
| 1 | Microprocessor acts as a heart of computer system. | Microcontroller acts as a heart of embedded system. |
| 2 | It is a processor in which memory and I/O output component is connected externally. | It is a controlling device in which memory and I/O output component is present internally. |
| 3 | Since memory and I/O output is to be connected externally. Therefore, the circuit is more complex. | Since on chip memory and I/O output component is available. Therefore, the circuit is less complex. |
| 4 | It cannot be used in compact system. Therefore, microprocessor is inefficient. | It can be used in compact system. Therefore, microcontroller is more efficient. |
| 5 | Microprocessor has a smaller number of registers. Therefore, most of the operations are memory based. | Microcontroller has a greater number of registers. Therefore, a program is easier to write. |
| 6 | It is mainly used in General Purpose Systems | It is mainly used in Embedded Systems |

# Microprocessor vs Microcontroller

# Processors

- What is a processor?
  - Artifact that computes (runs algorithms)
  - Controller and data-path
- General-purpose (GP) processors:
  - Variety of computation tasks
  - Functional flexibility and low cost at high volumes (maybe)
  - Slow and power hungry
- Single-purpose (SP) processors (or ASIC, ASIP)
  - One particular computation task
  - Fast and power efficient
  - Functional inflexibility and high cost at low volumes (maybe)

# General-purpose processors

- Programmable device used in a variety of applications
- Features
  - Program memory
  - General datapath with large register file and general ALU
- User benefits
  - Low time-to-market and NRE costs
  - High flexibility
- Examples
  - Pentium, Athlon, PowerPC

# Application-specific processors (ASICs)

- Programmable processor optimized for a particular class of applications
- Features
- Program memory
- Optimized data path
- Special functional units
- Benefits
  - Some flexibility, good performance, size and power
- Examples
  - DSPs, Video Signal Processors, Network Processors,..

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Module 1

# Introduction to Embedded System

# Embedded Systems: Simplified Block Diagram

# Components of Embedded Systems

Hardware

# Components of Embedded Systems - Hardware

**Power Supply**

- Key component to provide the power to the circuit
- Usually, the embedded system requires 5 V supply or can be range from 1.8 to 3.3. V
- The power supply source can be battery or can be provided by a wall adaptor
- The power supply is selected as per user requirements and application requirements

# Components of Embedded Systems - Hardware

**Microcontroller**

- An embedded system is either a microcontroller-based or microprocessor-based system
- The embedded hardware performance is mainly dependent on the processor which is normally called the brain of the embedded system
- Pick from a range of processors including 8-bit, 16-bit, and 32-bit processors

# Components of Embedded Systems - Hardware

**Microcontroller**

- They are different in terms of processing speed
- For example, a 32-bit processor comes with more processing speed and can manipulate 32-bits at a time while an 8-bit processor comes with less processing speed and can manipulate 8-bits at a time
- 8-bit processor would suffice while for basic computations. For complex and advanced applications, processors with more bits are used
- 8-bit processor is normally clocked to 8MHz while the 32-bit processor can run up to hundreds of MHz

# Components of Embedded Systems - Hardware

**Memory**

- Memory is essential to store important information in the embedded computer system
- Memory is integrated into a microcontroller
- Two types of memories including ROM (read-only-memory) and RAM (random access memory)
- ROM – Code Memory, RAM – Data Memory

# Components of Embedded Systems - Hardware

**Timers/Counters**

- Some of the applications there is always a requirement of delay that needed to provide in the application
- The programming can be done in such a way so that delay can be generating the embedded system
- The delay time span can be decided by using the crystal oscillator and system frequency so that delay can be generated as per user requirement

# Components of Embedded Systems - Hardware

## Communication Ports

- The communication port is the type of interface that is used to communicate with other types of embedded systems
- In the embedded system there is multiple types of communication ports like UART, USB, Ethernet, RS-485, and many more
- For simple applications, communications ports are utilized from the microcontroller, and for complex and advanced applications these ports are externally installed inside the embedded systems

# Components of Embedded Systems - Hardware

**Input/Output**

- Input is required to interact with the embedded system
- A sensor or user input can be used to provide input to the system
- The microcontroller used in the system can be configured as an input or output port
- The proper configuration needs to be done for using the input and output port
- In the embedded system there are fixed input and output ports so that devices can be connected to that specified ports only

# Components of Embedded Systems

Software

# Components of Embedded Systems - Software

**Editor**

- The editor is the first tool you required for embedded system software
- The code you write in C and C++ or even assembly programming languages will be saved in a text file in the editor

# Components of Embedded Systems - Software

## Compiler

- The code is written in a text editor. But how does a machine understand this code?
- A **compiler** is used to turn this written code into low-level machine language that the machine can comprehend
- The main purpose of this tool is to develop an executable program
- The name '**compiler**' is mainly used for the written programs that convert high-level programming language source code into a low-level programming language

# Components of Embedded Systems - Software

**Assembler**

- The assembler tool converts the written code into a machine language
- It is slightly different than a compiler
- The compiler directly converts the written code into machine language while the assembler
- The assembler, on the other hand, first converts source code to object code and then to the language that the machine can understand

# Components of Embedded Systems - Software

**Emulator**

- An emulator is a software tool that is used to execute the functions of the host system
- All the components can be controlled by the emulator tool
- The emulator is also used for finding the bugs and for debugging code
- The emulator also used to transfer the code from the host system to the target system

# Components of Embedded Systems - Software

## Linker

- Typically, software code is written in small modules and pieces
- A linker, also called a link editor, is a tool that takes one or more object files and combines them to develop a single executable code

# Components of Embedded Systems - Software

**Debugger**

- A debugger is a tool used for testing and debugging purposes
- It scans the code thoroughly and removes the errors and bugs, and identifies the places where they occur
- Programmers can quickly address the errors and fix them

# Choosing microcontroller for an Embedded System

- Selecting the right microcontroller for a product can be a daunting task
- Not only are there several technical features to consider, but there are also business case issues such as cost and lead-times that can cripple a project
- At the start of a project there is a great temptation to jump in and start selecting a microcontroller before the details of the system has been hashed out
- This is of course a bad idea

# Choosing microcontroller for an Embedded System

- Before any thought is given to the microcontroller, the hardware and software engineers should work out the high levels of the system
- Block diagram and flowchart them and only then is there enough information to start making a rational decision on microcontroller selection
- When that point is reached, there are few steps that can be followed to ensure that the right choice is made

# Choosing microcontroller for an Embedded System - Steps

**Step 1 - Make a list of required hardware interfaces**

- Using the general hardware block diagram, make a list of all the external interfaces that the microcontroller will need to support
- There are two general types of interfaces that need to be listed
  - The first are communication interfaces - USB, I2C, SPI, UART, and so on
  - The second type of interface is digital inputs and outputs, analog to digital inputs, PWM's, etc.
- These two interface types will dictate the number of pins that will be required by the microcontroller

# Choosing microcontroller for an Embedded System - Steps

**Step 1 - Make a list of required hardware interfaces**

Generic example of a block diagram with the i/o requirements listed

# Choosing microcontroller for an Embedded System - Steps

**Step 2 - Examine the software architecture**

- The software architecture and requirements can greatly affect the selection of a microcontroller
- How heavy or how light the processing requirements will determine whether you go with an 80 MHz DSP or an 8 MHz 8051
- Just like with the hardware, make notes of any requirements that will be important
- For example, do any of the algorithms require floating point mathematics? Are there any high frequency control loops or sensors?

# Choosing microcontroller for an Embedded System - Steps

**Step 2 - Examine the software architecture**

- Estimate how long and how often each task will need to run
- Get an order of magnitude feel for how much processing power will be needed.
- The amount of computing power required will be one of the biggest requirements for the architecture and frequency of the microcontroller

# Choosing microcontroller for an Embedded System - Steps

**Step 3 - Select the architecture**

- Using the information from steps 1 and 2 an engineer should be able to start getting an idea of the architecture that will be needed
- Can the application get by with eight-bit architectures? How about 16 bits? Does it require a 32-bit Arm core?
- Between the application and the required software algorithms these questions will start to converge on a solution
- **Don't forget that microcontroller selection can be an iterative process**

# Choosing microcontroller for an Embedded System - Steps

**Step 3 - Select the architecture**

- **Don’t forget that microcontroller selection can be an iterative process**
- You may select a 16-bit part in this step but then in a later step find that a 32-bit Arm part works better
- This step is simply to start getting an engineer to look in the right direction

# Choosing microcontroller for an Embedded System - Steps

**Step 4 - Identify Memory Needs**

- Flash and RAM are two very critical components of any microcontrollers
- Making sure that you don’t run out of program space or variable space is undoubtedly of highest priority
- You can always start with more and then later move to a more constrained part within the same chip family
- Using the software architecture and the communication peripherals included in the application, an engineer can estimate how much flash and RAM will be required for the application

# Choosing microcontroller for an Embedded System - Steps

**Step 5 - Start searching for microcontrollers**

- Now that there is a better idea of what the required features of the microcontroller will be the search can begin
- One place that can be a good place to start is with a microcontroller supplier
- Most silicon providers have a search engine that allows you to enter your peripheral sets, I/O and power requirements and it will narrow down the list of parts that match the criteria

# Choosing microcontroller for an Embedded System - Steps

**Step 6 - Examine Costs and Power Constraints**

- At this point the selection process has revealed a number of potential candidates
- This is a great time to examine the power requirements and cost of the part
- If the device will be powered from a battery and mobile, then making sure the parts are low-power is precarious
- If it doesn’t meet power requirements, then keep weeding the list down until you have a select few. Don’t forget to examine the piece price of the processor either

# Choosing microcontroller for an Embedded System - Steps

**Step 7 - Check part availability**

- With the list of potential parts in hand, now is a good time to start checking on how available the part is
- Some of the things to keep in mind are what the lead times for the part?
- What are your requirements for availability?
- Then there is a question of how new the part is and whether it will be around for the duration of your product life cycle

# Choosing microcontroller for an Embedded System - Steps

**Step 8 - Select a development kit**

- One of the best parts of selecting a new microcontroller is finding a development kit to play with and learn the inner working of the controller
- Once an engineer has settled their heart on the part they want to use they should research what development kits are available
- If a development kit isn't available then the selected part is most likely not a good choice and they should go back a few steps and find a better part

# Choosing microcontroller for an Embedded System - Steps

**Step 9 - Investigate compilers and tools**

- The selection of the development kit nearly solidifies the choice of microcontroller
- The last consideration is to examine the compiler and tools that are available
- Most microcontrollers have a number of choices for compilers, example code and debugging tools
- It is important to make sure that all the necessary tools are available for the part

# Choosing microcontroller for an Embedded System - Steps

**Step 10 - Start Experimenting**

- Even with the selection a microcontroller nothing is set in stone
- Take advantage by building up test circuits and interfacing them to the microcontroller
- Choose high risk parts and get them working on the development kit
- It may be that you discover the part you thought would work great has some unforeseen issue that would force a different microcontroller to be selected
- **In any event, early experimentation will ensure that you made the right choice and that if a change is necessary, the impact will be minimal!**

**ECS 4010 Introduction to Embedded Systems**

## 1. Case Study I: Automatic Chocolate Vending Machine (ACVM)

An ACVM contains a Coin insertion slot and Keypad (on the top of the machine) to insert the coin according to the possible denomination like 2, 5 rupees. Then after the coin is inserted, the system directs each coin to the particular port like port 2 and port 5 (coin sorter). It also containsan LCD unit on the top of the machine to display menus, text entered into the ACVM and pictograms, welcome, thank and other messages. Graphic interactions are also available on thismachine. The displays in the ACVM also show the current time and date. The delivery slot in the ACVM is used to collect the chocolate and coins (if refunded). The internet connection portis provided so that the owner can know the status of the ACVM sales from a remote location. The block diagram of an ACVM is shown in Figure 1 given below.

**Figure 1. Block Diagram of ACVM.**

### 1.1 ACVM Specifications

#### 1.1.1 ACVM Hardware

The heart of an ACVM is a Microcontroller or ASIP (Application Specific Instruction Set Processor). A RAM is used for storing temporary variables and the stack, and a ROM for application codes, and the RTOS codes for scheduling the tasks. It also has flash memory for storing user preferences, contact data, user address, a user date of birth, user identification code and answers to frequently asked questions (FAQs). Timer and Interrupt controller are also needed to control the process of ACVM. It has a TCP/IP port (Internet broadband connection) to the ACVM for remote control and for providing the system status reports to the owner. It also has an ACVM specific hardware and a power supply.

### 1.1.2 ACVM Software

Software is required to handle the following:
Read input from keypad, display text/graphics, control coins reader, and control delivery port (to deliver the chocolate). In addition to these, we also need the TCP/IP stack communication for remote control, and an RTOS (say, MUCOS), to run the ACVM software.

## 1.2 ACVM Requirements

The purpose of ACVM is to build a system from which children can automatically purchase the chocolates, and the payment is by inserting the coins to the appropriate denomination coin-slot.

### 1.2.1 Inputs

Coin slot to insert the coins of different denominations and the keypad to enter the user commands.

### 1.2.2 Signals, events and Notifications

An interrupt is generated at each port after the coin is received in the coin slot. Each port interrupt starts an Interrupt Service Routine (ISR), which increases value of amount collected by corresponding rupees (1, 2, 5 or 10). A notification is generated for each selection in the menu.

### 1.2.3 Outputs

The display is used to show the GUIs, time and date, advertisements, welcome and thanks messages. Chocolate and signal (IPC) to the system that subtracts the cost from the value of amount collected.

### 1.2.4 Functions of the system

A child (user) sends commands to the ACVM using a GUI (graphic user interface). GUI consists of the LCD and keypad units. At first, the child inserts the coins ( Task_Collect through Port_Collect) for the cost of chocolate and the machine delivers the chocolate in the delivery slot. If the coins are not inserted as per the cost of chocolate for a reasonable amount of time, then all coins are refunded (Task_Refund through Port_Refund). If the inserted coins amount is more than the cost of chocolate, the excess amount is refunded along with chocolate (Task_ExcessRefund through Port_ExcessRefund). If the chocolate is of different rupees, then the port is assigned to each rupee, and then the interrupt is sent to the corresponding port (Task_ReadPorts through Port_Read). After that chocolate is delivered through the delivery slot (Task_Deliver through Port_Deliver). The coins for the chocolates purchased collect inside the machine in a collector channel (Task_Collect), so that owner can get the money, again through appropriate commands using the GUI (Task_Display). USB wireless modem enables communication through Internet to the ACVM system owner.

### 1.2.5 Design metrics

The design of the system is measured in terms of four design metrics and are explained as follows

- **Power Dissipation:** Maximum (tolerance) amount of heat it can generate while working as required by mechanical units, display units and computer system.
- **Performance:** Based on assumption, one chocolate will be delivered in two minutes and 256 chocolates before next filling of chocolates into the machine.
- **Process Deadlines:** Machine waits for maximum 30s for the coins and it should deliver the chocolate within 60s.

- **User Interfaces(UI):** Graphics at LCD or touchscreen display on LCD, and commands by children or machine owner through fingers on keypad or touch screen, form the UI in the ACVM.

Apart from these metrics, the manufacturing and engineering cost is also considered for the design metrics.

**1.2.6 Test and validation conditions**

The test and validation conditions are expressed to check whether all user commands function correctly and all the graphic displays and menus appear as per the program. Then each task should be tested with test inputs, and it should be tested for 60 users per hour.

**1.2.7 Basic system of ACVM**

The flow diagram of an ACVM is shown in figure 2. ACVM system consists of a slot into which a child inserts the coins for buying the chocolate. Whenever a coin is inserted, a mechanical system directs each coin of value Rs 1 or 2 or 5 to port -1, port -2, port -5 respectively. When a port is receiving a coin, the port generates an interrupt. The interrupt signal is sent by the corresponding read ports for reading the coin value at the ports and also to increase the amountof chocolate. The machine should have an LCD, keypad and touchscreen. Let the interface port be called port-display. The time and date appear in the LCDs right-hand bottom side. ACVM hasa port-deliver where the buyer collects the chocolate from the bowl.The customer also receives the full refund or excess amount at the bowl. It should also be possible to reprogram and relocate the codes in the system ROM or flash ROM whenever the following happens:

a) The price of chocolate increases.
b) The messages lines or menus need to be changed.
c) Machine features change.

The function of the ACVM system is as already described in Section 1.2.4.

**Figure 2. Basic System of an ACVM**

With respect to the hardware the following will be required. The 8051 can be used as the microcontroller and MUCOS the RTOS used in the ACVM. ACVM specific hardware is required to sort the coins of different denomination using coin sorter and the main Power supply needed is 220V 50Hz or 110V 60Hz. Internal circuits need a supply of 5V 50mA for electronics and 12V, 2A for mechanical systems. By programming the 8051 timer, the 1s resolution timer is obtained. Flash memory of ROM and RAM is used for storing the temporary variables and stack. 8 MB ROM is needed for application codes and RTOS codes.

## 2. Case study II: Smart Card

Smart card is one of the most used embedded system today. It is used for credit card, debit bank card, e-wallet card, identification card, medical card (for history and diagnosis details) and card for some new innovative applications. Smart card improves the convenience and security of any transaction. It provides a tamper-proof storage of user and account identity. Smart card systems have proven to be more reliable than other machine-readable cards, like magnetic stripe and barcode, etc. It also provides vital components of system security for the exchange ofdata throughout virtually any type of network. It is also a cost effective method. The smart cards are used today in various applications, including healthcare, banking, entertainment, and transportation. Smart card provides the security by added features that benefit all these applications.

### 2.1 Smart Cards

Smart cards are plastic cards embedded with a microprocessor/ microcontroller or memory chip that stores and transacts data. The smart card is differentiated into two types based on application as follows:

- Identification and process based smart card
- Identification based smart card

**Identification and process based smart cards** are of two types. They are:

**Contact based smart cards**- In the Contact based cards, the chip is attached to the materials itself as shown in Figure 3.

**Contactless smart cards** – This type of smart card is not attached directly to the system. Example is USB smart card RFID as shown in figure 4.

**Figure 3. Contact based smart card**

**Figure 4. Contactless smart card**

**Identification based smart card**

An example for this is RFID tags(Figure 5).

**Figure 5. RFID tags**

## 2.1 Smart Card Specifications

### 2.2.1 Embedded hardware components

Hardware components needed for smart card are: Microcontroller or ASIP, RAM for storing temporary variables and Stack; OTP ROM for application codes and RTOS codes for scheduling the tasks; Flash for storing user data, user address, user identification codes, card number and expiry date; Timer and interrupt controller for controlling purpose; a carrier frequency generating circuit and ASK modulator for modulating the carrier frequency according to the system. Finally, an Interfacing circuit for the IOs is needed. Then charge pumps for delivering power to the antenna for transmission and for the system circuits are also needed.

### 2.2.2 Embedded software components

Software components needed for the smart card system are Boot-up, system initialization and embedded system features. It needs a secure three layered file system called Smart card secure file system. This Smart card secure file system is needed for storing the files. Connection establishment and termination is provided by TCP/IP port connection. Then cryptographic algorithm is used for the added features like host connection. OS is stored in the protected part of the ROM. Host and card authentication are also needed for the smart card. Optimum code size and multidimensional array are needed to save the data.

### 2.2.3 Smart Card system

**requirementsPurpose:**

It enables authentication and verification of card and card holder by a host. Smart card enables GUI at host machine to interact with the cardholder/user for the required transactions. For example, financial transactions with a bank or credit card transactions.

**Inputs**

IO port is required to receive header and messages for the smart card system.

**Internal Signals, Events and Notifications:**

On power up, radiation is generated that gives the power supply to smart card (to activate the card). On activation a reset_Task is initiated which initialises the necessary timers and creates the tasks (task_ReadPort, task_PW, task_Appl) necessary to perform other functions. The task_ReadPort is responsible for sending & receiving the messages and for starting & closing applications. The task_PW is responsible for handling the passwords. The task_Appl runs the actual application.

**Output**

Headers and messages are transmitted through antenna at Port_IO.

**Control panel**

No control panel is at the card. The control panel and GUIs are at the host machine (for example, at a ATM credit card reader).

**Function of the system**

The functions are as follows: First the card is inserted at a host machine. Then the radiations from the host activate a charge pump at the card. The charge pump powers the SoC(System- On Chip). On power up, system reset signals reset_Task to start. All transactions between cardholder/user now takes place through GUIs at the host control panel (screen or touch screenor LCD display panel).

**Design metrics**

The following are the design metrics used for the smart card case study.

- **Power Dissipation:** Maximum (tolerance) amount of heat it can generate while working must be less.
- **Code size:** The optimum system memory needed should not exceed 64 KB memory.
- **Limited use of data types:** The multidimensional arrays, long 64-bit integer and floating points are the data types used in smart card. The smart card should support a limited use of error handlers, exceptions, signals, serialization, debugging and profiling mechanisms.
- **File management:** There is either a fixed length file management or a variable length file management with each file with a predefined offset. The file system stores the data using a three layer mechanism (explained below) .
- **Microcontroller hardware:** It generates distinct coded physical addresses for the program and logical addresses for the data. It is a protected, once-writable memory space.
- **Validity:** System is embedded with expiry date, after which the card authorization through the hosts is disabled.
- **Extendability:** The system expiry date is extendable by transactions and authorization of master control unit.
- **User Interfaces:** At host machine, graphics at LCD or touch screen display on LCD and commands for card holder (card user) transactions (within 1s) are the user interface requirements.

Apart from these metrics the manufacturing and engineering cost is also considered for the design metrics.

**Test conditions and validations**

It must be tested on different host machine versions for fail-proof card-host communication.

### 2.2.4 Smart Card hardware

A smart Card hardware system is shown in Figure 6 given below. It consists of a plastic card in ISO standard and it is an embedded SoC.

**Figure 6. Smart Card Hardware System**

The CPU in the hardware locks certain section of memory and protects 1 kB or more data from modification and access by any external source or instruction outside that memory ( protecting the data). Another way of protecting is to allow the CPU to access through the physicaladdresses, which are different from the logical addresses used in the program. The EEPROM orFlash memory is needed to store P.I.N. (Personal Identification Number). It is also used to store the unblocking P.I.N., access condition for the data files, card-user data, application generated data, applications non-volatile data, and invalidation lock to invalidate the card after the expiry date or server instruction. The ROM in smart card contains a Fabrication key (unique security key for each card), Personalization key (this key is inserted after the testing phase and it preserves the fabrication key; after that the RTOS only uses logical address), RTOS codes, application codes and Utilization lock. Then the need of RAM is to store the run time temporary variables. The chip-supply system voltage is extracted by a charge pump I/O system. It extractsthe charge from the signals from the host and then it generates regulated voltage to card chip, memory and I/O system. The I/O system of the chip and host interact through asynchronous UART at 9.6 k or 106 k or 115.2 k baud/s.

### 2.2.5 Smart Card Software

Smart Cards are the most used systems today in the area of secure SoC Systems. It needs a cryptographic software. Embedded system in the card needs a special feature in its OS over and above the MS-DOS and UNIX system features. The special features needed are a Protected environment where the OS is stored, i.e., in a protected part of ROM. It needs a restricted runtime environment and every method, class and runtime library in the OS should be scalable. It requires an optimum code size and a limited use of data types and multidimensional

arrays. It needs a three-layered file system for the data. One for the master file containing the header (a header means file status, access conditions, and the file lock). Second is a dedicated file used to hold a file grouping and headers of the immediate successor. A third file (called the elementary file) is used to hold the file header and its file data. There may be either fixed or variable length file management with each file predefined with offset. It should have classes for network, sockets, connections, datagrams etc.

## 3. Summary

In this module two case studies have been discussed in detail - Automatic Chocolate Vending Machine and smart card.

## 4. References


1. Raj Kamal, "Embedded Systems - Architecture,Programming and Design", McGraw-Hillpublication,2008.

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Module 1

# Introduction to Embedded System

# Microcontroller Families

Most popular Microcontroller Families for an Embedded Engineer:

# Microcontroller - 8051

- 8-bit family of microcontroller is developed by the Intel in the year 1981
- This microcontroller was moreover referred as “system on a chip” since it has
- 128 bytes of RAM, 4Kbytes of a ROM, 2 Timers, 1 Serial port, and 4 ports on a single chip
- The CPU can also work for 8bits of data at a time since 8051 is an 8-bit processor
- Most manufacturers contain put 4Kbytes of ROM even though the number of ROM can be exceeded up to 64 K bytes

# Microcontroller - ARM

- ARM processor is also one of a family of CPUs based on the RISC (reduced instruction set computer) architecture developed by Advanced RISC Machines (ARM)
- ARM processors are wide used in client electronic devices like smartphones, tablets, multimedia players and other mobile devices, such as wearable
- Because of their reduced to the instruction set, they need fewer transistors, which enable a smaller die size of the integrated circuitry (IC)
- The ARM processors, smaller size means less difficulty and lower power expenditure makes them fit for increasingly miniaturized devices

# Microcontroller - Comparison

| | 8051 | PIC | AVR | ARM |
|---|---|---|---|---|
| Bus width | 8-bit for standard core | 8/16/32-bit | 8/32-bit | 32-bit mostly also available in 64-bit |
| Communication Protocols | UART, USART,SPI,I2C | PIC, UART, USART, LIN, CAN, Ethernet, SPI, I2S | UART, USART, SPI, I2C, (special purpose AVR support CAN, USB, Ethernet) | UART, USART, LIN, I2C, SPI, CAN, USB, Ethernet, I2S, DSP, SAI (serial audio interface), IrDA |
| Speed | 12 Clock/instruction cycle | 4 Clock/instruction cycle | 1 clock/ instruction cycle | 1 clock/ instruction cycle |
| Memory | ROM_ SRAM_ FLASH | SRAM_ FLASH | Flash_ SRAM, EEPROM | Flash-SDRAM, EEPROM |
| ISA | CISC | Some feature of RISC | RISC | RISC |
| Memory Architecture | Von Neumann architecture | Harvard architecture | Modified | Modified Harvard architecture |
| Power Consumption | medium | Low | Low | Low |
| Families | 8051 variants | PIC16,PIC17, PIC18, PIC24, PIC32 | Tiny, Atmega, Xmega, special purpose AVR | ARMv4,5,6,7 and series |
| Community | Vast | Moderate | Very Good | Vast |
| Manufacturer | NXP, Atmel, Silicon Labs, Dallas, Cyprus, Infineon, etc. | Microchip | Atmel | Apple, Nvidia, Qualcomm, Samsung Electronics, and TI etc. |
| Cost (as compared to features provide) | too Low | medium | medium | Low |
| Other Feature | Known for its Standard | Cheap | Cheap- effective | High-speed operation Vast |
| Popular Microcontrollers | AT89C51, P89v51, etc. | PIC18fXX8, PIC16f88X, PIC32MXX | Atmega8, 16, 32, Arduino Community | LPC2148, ARM Cortex-M0 to ARM Cortex-M7, etc. |

# Microcontroller - Comparison

- 8051 - if one desires a low-cost controller with basic functions then 8051 can fulfill. It will be of nice use for small scale projects
- ARM - if one wants fast computing, a large number of timers and ADC's then ARM will be fit
- AVR - one among the most popular classes of the controller. Low cost, large number of library files, used in many robotic applications. Best for the beginners
- PIC - Cheap, employed in refrigerators and low budget comes. Not advisable because of its low community support

# Microcontroller - Comparison

- 8051
  - Home appliances
  - Light sensing and controlling devices
  - Temperature sensing and control devices
  - Fire detection and safety devices such as home security systems
  - Other devices like calculator, etc.
- ARM
  - Used in smart devices with internet connectivity
  - Used in medical devices such as MRI Machines, ultrasound machines
  - Used in automotive industry, space and aerospace

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Module 2

# Building Embedded Hardware & Software

# Module 2 - Building Embedded Hardware & Software

- Hardware
  - ADC, DAC
  - Sensors and Actuators
  - Interfacing Memory devices
  - Interfacing IO Devices
- Software
  - Compilers, Linker, Runtime Library
  - Pre-processor Workflow-Compiler Tool chains
  - Firmware, Middleware
  - In-Circuit Debuggers, Logic Analyzer, and Integrated Development Environment

# Microcontroller - 8051

- 8-bit family of microcontroller is developed by the Intel in the year 1981
- This microcontroller was moreover referred as “system on a chip” since it has
- 128 bytes of RAM, 4Kbytes of a ROM, 2 Timers, 1 Serial port, and 4 ports on a single chip
- The CPU can also work for 8bits of data at a time since 8051 is an 8-bit processor
- Most manufacturers contain put 4Kbytes of ROM even though the number of ROM can be exceeded up to 64 K bytes

# Microcontroller – 8051 – AT89C51

- Manufacturer : ATMEL
- Low-power, high-performance 8-bit μP
- 4K Bytes of Flash Memory
- Fully Static Operation: 0 Hz to 24 MHz
- 128 x 8-bit Internal RAM
- 32 Programmable I/O Lines
- Two 16-bit Timer/Counters
- Six Interrupt Sources
- Programmable Serial Channel
- Low-power Idle and Power-down Modes

# Microcontroller – 8051 – History

- Intel first produced a µC in 1976 under the name MCS-48 – **8 bit**
- Later in **1980** improved version - **MCS-51** – **8 bit**
- 8051 belongs to the **MCS-51** family of microcontrollers by Intel
- Many other semiconductor manufacturers released microcontrollers under their own brand name but using the MCS-51 core
- Global companies and giants in semiconductor industry like Microchip, Zilog, **Atmel**, Philips, Siemens released products under their brand name
- The specialty was that all these devices could be programmed using the same MCS-51 instruction sets
- They basically differed in support device configurations like improved memory, presence of an ADC or DAC etc.

# Microcontroller – 8051 - Packaging

- Popular packaging for 8051
  - Dual Inline Package (40 pins)
  - Plastic Leaded Chip Carrier (40 pins)
  - TQFP (Thin Quad Flat Package), PQFP (Plastic Quad Flat Package) (44 pins)

- 40 pin DIP Package

PDIP

PQFP/TQFP

PLCC

# Microcontroller – 8051 - Architecture

- 8051 Architecture - internal architecture, pin configuration, program memory and data memory organization
- The basic architecture remains same for the MCS-51 family
- In general, all microcontrollers in MCS- 51 family are represented by XX51, where XX can take values like 80, 89 etc.

# Microcontroller – 8051 - Inputs

- 3 System Inputs
- VCC, GND -
- XTAL – Crystal clock circuit
- RESET - to initialize µC to default/ desired values and to make a new start

# Microcontroller – 8051 – Control Signals

- 3 Control Signals – Used for external memory interface
- **EA** – External Access
- **PSEN** - Program Store Enable
- **ALE** - Address Latch Enable

# Microcontroller – 8051 – Control Signals

- Functional µC if no external memory requirement
- EA – **High**
- PSEN - **NC**
- ALE – **NC**

- 0.1 micro farad decoupling capacitor connected to Vcc (to avoid HF oscillations at input)

# Microcontroller – 8051 – Ports

- 4 Ports
- Port 0, 1, 2 and 3
- Used for external interfacing of devices like DAC, ADC, 7 segment display, LED etc.
- Each port has 8 I/O lines, and they all are bit programmable

# Microcontroller – 8051 – Ports and Pins

- Pin **40** – VCC
- Pins **32-39**: **Port 0** (P0.0 to P0.7)
  - In addition to serving as I/O port, lower order address and data bus signals are multiplexed with this port
  - To serve the purpose of external memory interfacing
  - This is a True bidirectional I/O port
  - External pull up resistors are required to function this port as I/O

# Microcontroller – 8051 – Ports and Pins

- **Pin 31**: EA / External Access input
  - Used to enable or disallow external memory interfacing. If there is no external memory requirement, this pin is pulled high by connecting it to Vcc
- **Pin 30**: ALE aka Address Latch Enable
  - Used to demultiplex the address-data signal of port 0 (for external memory interfacing.)
- **Pin 29**: PSEN or Program Store Enable
  - Used to read signal from external program memory

# Microcontroller – 8051 – Ports and Pins

- Pins **21-28**: **Port 2** (P2.0 to P2.7)
  - Serves as I/O port
  - Higher order address bus signals are multiplexed with this quasi-bidirectional port
- Pin **20**: Vss - represents ground (0V)
- Pins **18 and 19: F**or interfacing an external crystal to provide system clock

# Microcontroller – 8051 – Ports and Pins

- Pins **10-17**: **Port 3** (P3.0 to P3.7)
  - Serves as I/O port
  - Serves functions like interrupts, timer input, control signals for external memory interfacing RD and WR , serial communication signals RxD and TxD etc.
- Pin **9**: Reset pin
- Pins **1-8**: **Port 1** (P1.0 to P1.7)
  - Unlike other ports, this port does not serve any other functions
  - It is a quasi-bidirectional port

# Microcontroller – 8051 – Architecture

- The system bus connects all the support devices with the central processing unit
- 8-bit data bus and a 16-bit address bus
- Program memory, ports, data memory, serial interface, interrupt control, timers, and the central processing unit are all interfaced together through the system bus

# Microcontroller – 8051 – Memory

- 8051 – Harvard Architecture
- Harvard architecture treats program memory and data memory as separate entities
- Princeton architecture treats address memory and data memory as a single unit
- 8051 has two memories : Program memory and Data memory

# Microcontroller – 8051 – Program Memory

- 8051 – Program Memory – 4K
- If needed an external memory can be added (by interfacing ) of size 60K maximum
- Total Program Memory – 64K
- 64K = 4K + 60K
- FFFF = (0000-0FFF) + (1000-FFFF)

| Bytes | Hex | Address range |
|---|---|---|
| 16 | 10 | 0 - F |
| 256 | 100 | 0 - FF |
| 1K | 400 | 0 - 3FF |
| 4K | 1000 | 0 - FFF |
| 64K | 10000 | 0 - FFFF |
| 1M | 100000 | 0 - FFFFF |
| 16M | 1000000 | 0 - FFFFFF |

# Microcontroller – 8051 – Program Memory

- 8051 – Program Memory – 4K
- If needed an external memory can be added (by interfacing ) of size 60K maximum
- Total Program Memory – 64K
- 64K = 4K + 60K
- FFFF = (0000-0FFF) + (1000-FFFF)

# Microcontroller – 8051 – Data Memory

- 8051 – Data Memory – 128 bytes
- If needed an external data memory can be added (by interfacing ) of size 64K maximum
- Total Data Memory
  - 64K (External) + 128 (Internal)

# Microcontroller – 8051 – Data Memory

- 8051 – Internal Data Memory – 128 bytes

- 3 separations/divisions of the internal data memory:
  - Register banks
  - Bit addressable area
  - Scratch pad area.

# Microcontroller – 8051 – Data Memory

- Register banks
  - 4 Register banks
  - #0,#1, #2 and #3
  - Each bank has 8 registers which are designated as R0,R1…R7
  - At a time only one register bank is selected for operations and the registers inside the selected bank are accessed using mnemonics R0..R1..
  - Registers are used to store data or operands during executions

# Microcontroller – 8051 – Data Memory

- Bit addressable area
  - Usually used to store bit variables
  - The bit addressable area is formed by the 16 bytes next to register banks
  - Designated from address 20H to 2FH (total 128 bits)
  - Bit addressable area is mainly used to store bit variables from application program
  - Like status of an output device like LED or Motor (ON/OFF) etc.

# Microcontroller – 8051 – Data Memory

- Scratch-pad Area
  - Upper 80 bytes
  - Area is from 30H to 7FH
  - Includes stack too

# Microcontroller – 8051 – Clock

- Generally, a quartz crystal is used to make the clock circuit
- In some cases, external clock sources are used
- Clock frequency limits (maximum and minimum) may change from device to device
- Standard practice is to use 12MHz frequency
- If serial communications are involved then its best to use 11.0592 MHz frequency

# Microcontroller – 8051 – Reset

- 8051 can be reset in two ways
  - Power-on reset – which resets the 8051 when power is turned ON
  - Manual reset – in which a reset happens only when a push button is pressed manually
- During a reset operation :- Program counter is cleared, and it starts from 00H
- Register bank #0 is selected as default, Stack pointer is initialized to 07H, all ports are written with FFH

# Microcontroller – 8051 – Programming Tools

- Keil 8051 IDE
  - Embedded C

- Proteus Simulator
  - AT89C51

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# 8051 PROGRAMMING IN C

- Compilers produce hex files that is downloaded to ROM of microcontroller
  - The size of hex file is the main concern
  - Microcontrollers have limited on-chip ROM
  - Code space for 8051 is limited to 64K bytes
- C programming is less time consuming, but has larger hex file size
- The reasons for writing programs in C
  - It is easier and less time consuming to write in C than Assembly
  - C is easier to modify and update
  - C code is portable to other microcontroller with little of no modification

# 8051 PROGRAMMING IN C – Data Types

- A good understanding of C data types for 8051 can help programmers to create smaller hex files
  - Unsigned char
  - Signed char
  - Unsigned int
  - Signed int
  - Sbit (single bit)
  - Bit and sfr

# 8051 PROGRAMMING IN C – Data Types

**Unsigned char**

- The character data type is the most natural choice
- Unsigned char is an 8-bit data type in the range of 0 – 255 (00 – FFH)
- C compilers use the signed char as the default if we do not put the keyword unsigned

- One of the most widely used data types for the 8051
- Counter value
- ASCII characters

# 8051 PROGRAMMING IN C – Data Types

Write an 8051 C program to send values 00 – FF to port P1

```
#include <reg51.h>
void main(void)
{
unsigned char z;
for (z=0;z<=255;z++)
      P1=z;
}
```

# 8051 PROGRAMMING IN C – Data Types

Write an 8051 C program to toggle all the bits of P1 continuously.

# 8051 PROGRAMMING IN C – Data Types

Write an 8051 C program to toggle all the bits of P1 continuously.

```
#include <reg51.h>
void main(void)
{
        for (;;)
        {
                p1=0xAA; p1=0x55;
        }
}
```

# 8051 PROGRAMMING IN C – Data Types

**Signed char**

- The signed char is an 8-bit data type
- Use the MSB D7 to represent – or +
- Give us values from –128 to +127
- We should stick with the unsigned char  unless the data needs to be represented as signed numbers
- Temperature

# 8051 PROGRAMMING IN C – Data Types

Write an 8051 C program to send values of –4 to +4 to port P1.

```
#include <reg51.h>
void main(void)
{
char mynum[]={+1,-1,+2,-2,+3,-3,+4,-4};
unsigned char z;
for (z=0;z<=8;z++)
P1=mynum[z];
}
```

# 8051 PROGRAMMING IN C – Data Types

**Unsigned int**

- Takes a value in the range of 0 to 65535 (0000 – FFFFH)
- Define 16-bit variables such as memory addresses
- Set counter values of more than 256
- Since registers and memory accesses are in 8-bit chunks, the misuse of int variables will result in a larger hex file

**Signed int**

- Signed int is a 16-bit data type
- Use the MSB D15 to represent – or +
- We have 15 bits for the magnitude of the number from –32768 to +32767

# 8051 PROGRAMMING IN C – Data Types

Write an 8051 C program to toggle bit D0 of the port P1 (P1.0) 50,000 times.

```
#include <reg51.h>
sbit MYBIT=P1^0;
void main(void)
{
unsigned int z;
for (z=0;z<=50000;z++){
      MYBIT=0;  MYBIT=1;
}}
```

*sbit* keyword allows access to the single bits of the SFR registers

# 8051 PROGRAMMING IN C – Data Types

- The bit data type allows access to single bits of bit-addressable memory spaces 20 – 2FH
- To access the byte-size SFR registers, we use the sfr data type

| Data Type | Size in Bits | Data Range/Usage |
|---|---|---|
| unsigned char | 8-bit | 0 to 255 |
| (signed) char | 8-bit | -128 to +127 |
| unsigned int | 16-bit | 0 to 65535 |
| (signed) int | 16-bit | -32768 to +32767 |
| sbit | 1-bit | SFR bit-addressable only |
| bit | 1-bit | RAM bit-addressable only |
| sfr | 8-bit | RAM addresses 80 – FFH only |

# 8051 PROGRAMMING IN C – Time Delay

- There are **two ways** to create a time delay in 8051 C
- Using the 8051 timer
- Using a simple for loop
- Be mindful of **three factors** that can affect the accuracy of the delay
- The 8051 design
  - The number of machine cycle
  - The number of clock periods per machine cycle
- The crystal frequency connected to the X1 – X2 input pins
- Compiler choice
  - C compiler converts the C statements and functions to Assembly language instructions
  - Different compilers produce different code

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# 8051 PROGRAMMING IN C – Time Delay (Software)

Write an 8051 C program to toggle bits of P1 continuously forever with some delay.

```
#include <reg51.h>
void main(void){
unsigned int x;
for (;;)                    //repeat forever
{
p1=0x55;
for (x=0;x<40000;x++);  //delay size unknown
p1=0xAA;
for (x=0;x<40000;x++);
}}
```

We must use the oscilloscope to measure the exact duration

# 8051 PROGRAMMING IN C – Time Delay (Software)

Write an 8051 C program to toggle bits of P1 ports continuously with a 100 ms.
Crystal Frequency = 12Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
while (1)
{
p1=0x55;
MSDelay(100);
p1=0xAA;
MSDelay(100);
}}
```

```
void MSDelay(unsigned int
itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<Number;j++);
}
```

# 8051 PROGRAMMING IN C – Time Delay (Software)

Write an 8051 C program to toggle bits of P1 ports continuously with a 100 ms. Crystal Frequency = 12Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
Time for 1 Instruction
Cycle = 12 / 12 Mhz = 1µs

Y * 1µs = 100 ms

Y = 100 ms/ 1µs

Y = 10000
```

```
void MSDelay(unsigned int
itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<Number;j++);
}
```

# 8051 PROGRAMMING IN C – Time Delay (Software)

Write an 8051 C program to toggle bits of P1 ports continuously with a 100 ms.
Crystal Frequency = 12Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
while (1)
{
p1=0x55;
MSDelay(100);
p1=0xAA;
MSDelay(100);
}}
```

```
void MSDelay(unsigned int
itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<1000;j++);
}
```

# 8051 PROGRAMMING IN C – Time Delay (Software)

Write an 8051 C program to toggle bits of P1 ports continuously with a 20 ms.
Crystal Frequency = **11.0592 Mhz**, 8051 – 1 instruction cycle = 12 clock cycles

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
while (1)
{
p1=0x55;
MSDelay(100);
p1=0xAA;
MSDelay(100);
}}
```

```
void MSDelay(unsigned int
itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<Number;j++);
}
```

# 8051 PROGRAMMING IN C – Input Output Ports

Write an 8051 C program to get a byte of data form P1, wait 1/2 second, and then send it to P2.

```
#include <reg51.h>
void MSDelay(unsigned int);
void main(void)
{
------ Logic--------------
}
```

```
------Logic-----------
unsigned char mybyte;
P1=0xFF;      //make P1 input port
while (1)
{
mybyte=P1;    //get a byte from P1
MSDelay(500);
P2=mybyte;    //send it to P2
}
```

# 8051 PROGRAMMING IN C – Bit Addressing

Write an 8051 C program to toggle only bit P2.4 continuously without disturbing the rest of the bits of P2.

```
#include <reg51.h>
sbit mybit=P2^4;

void main(void){
while (1){
mybit=1;
mybit=0;
}}
```

Ports P0 – P3 are bit- addressable and we use *sbit* data type to access a single bit of P0 - P3

Use the **Px^y** format, where x is the port 0, 1, 2, or 3 and y is the bit 0 – 7 of that port

# 8051 PROGRAMMING IN C – Bit Addressing + Delay (Software)

Write an 8051 C program to toggle only bit P2.4 every 100ms without disturbing the rest of the bits of P2.
Crystal Frequency = 12Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
#include <reg51.h>
sbit mybit=P2^4;
void MSDelay(unsigned int);

void main(){
while (1){
mybit= ~mybit;
MSDelay(100);
}}
```

```
void MSDelay(unsigned int
itime)
{
unsigned int i,j;
for (i=0;i<itime;i++)
for (j=0;j<1000;j++);
}
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
Frequency = 20Hz, Time = 1/20 = 50ms
Time = 50ms = 25ms (ON) + 25ms (OFF)

Delay of 25 ms

Timer 0, Mode 1

(Use 8051 hardware ref manual for registers)
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
Delay of 25 ms

Timer 0, Mode 1

TMOD – Set timer/counter + mode
TCON – TR0, TF0
TR0 – Start timer
TF0 – Timer overflow
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
#include <reg51.h>
sbit mybit = P2^4;
void MSDelay(void);

void main(){
      TMOD = 0x01;
      while(1){
            mybit = ~mybit;
            MSDelay();
      }}
```

```
void MSDelay()
{
      TL0 = Lower Byte;
      TH0 = Upper Byte;
      TR0 = 1;
      while (TF0 ==0);
      TR0 = 0;
      TF0 = 0;
}
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592MHz, 8051 – 1 instruction cycle = 12 clock cycles

```
Delay Calculation

XTAL = 11.0592 MHz
Time for 1 instru = 12/11.0592MHz
Time for 1 instru = 1.085 µs
Delay of 25ms = 25/1.085 µs = 23037
Count = 65536 – 23037 = 42499
42499 = A5D1 (Hex)
TH0 = A5, TL0 = D1
```

```
void MSDelay()
{
        TL0 = D1;
        TH0 = A5;
        TR0 = 1;
        while (TF0 ==0);
        TR0 = 0;
        TF0 = 0;
}
```

# 8051 PROGRAMMING IN C –Interrupts

- Interrupts are useful in many cases wherein the process simply wants to continue doing its main job and other units(timers or external events) seek its attention when required
- In other words, the microcontroller, need not monitor the timers, the serial communication or the external pins P3.2 and P3.3
- Whenever an event related to these units occur, it is informed to the microcontroller with the help of interrupts

# 8051 PROGRAMMING IN C –Interrupts

- A single microcontroller can serve several devices by two ways:
- Polling
  - The microcontroller continuously monitors the status of all the devices.
  - Whenever any device needs the service, it provides the service and moves on to the next device until everyone is serviced.
  - This will be done in an infinite loop.
- Interrupts
  - Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
  - Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.
  - The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
Frequency = 20Hz, Time = 1/20 = 50ms
Time = 50ms = 25ms (ON) + 25ms (OFF)

Delay of 25 ms

Timer 0, Mode 1

(Use 8051 hardware ref manual for registers)
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
Delay of 25 ms

Timer 0, Mode 1

TMOD – Set timer/counter + mode
TCON – TR0, TF0
TR0 – Start timer
TF0 – Timer overflow
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592Mhz, 8051 – 1 instruction cycle = 12 clock cycles

```
#include <reg51.h>
sbit mybit = P2^4;
void MSDelay(void);

void main(){
      TMOD = 0x01;
      while(1){
            mybit = ~mybit;
            MSDelay();
      }}
```

```
void MSDelay()
{
      TL0 = Lower Byte;
      TH0 = Upper Byte;
      TR0 = 1;
      while (TF0 ==0);
      TR0 = 0;
      TF0 = 0;
}
```

# 8051 PROGRAMMING IN C –Delay (Hardware)

Write an 8051 C program to generate frequency of 20Hz on Pin P2.4
Crystal Frequency = 11.0592MHz, 8051 – 1 instruction cycle = 12 clock cycles

```
Delay Calculation

XTAL = 11.0592 MHz
Time for 1 instru = 12/11.0592MHz
Time for 1 instru = 1.085 µs
Delay of 25ms = 25/1.085 µs = 23037
Count = 65536 – 23037 = 42499
42499 = A5D1 (Hex)
TH0 = A5, TL0 = D1
```

```
void MSDelay()
{
        TL0 = D1;
        TH0 = A5;
        TR0 = 1;
        while (TF0 ==0);
        TR0 = 0;
        TF0 = 0;
}
```

# 8051 PROGRAMMING IN C –Interrupts

- Interrupts are useful in many cases wherein the process simply wants to continue doing its main job and other units(timers or external events) seek its attention when required
- In other words, the microcontroller, need not monitor the timers, the serial communication or the external pins P3.2 and P3.3
- Whenever an event related to these units occur, it is informed to the microcontroller with the help of interrupts

# 8051 PROGRAMMING IN C –Interrupts

- A single microcontroller can serve several devices by two ways:
- Polling
  - The microcontroller continuously monitors the status of all the devices.
  - Whenever any device needs the service, it provides the service and moves on to the next device until everyone is serviced.
  - This will be done in an infinite loop.
- Interrupts
  - Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal
  - Upon receiving an interrupt signal, the microcontroller interrupts whatever it is doing and serves the device.
  - The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

# 8051 PROGRAMMING IN C –Interrupt Structure

- 8051 Microcontroller has six interrupt sources

| Interrupt | ROM Location(Hex) | Pin | Flag Clearing | Interrupt no. in C |
|---|---|---|---|---|
| Reset | 0000 | 9 | Auto | -- |
| External HW Interrupt 0 (INT0) | 0003 | P3.2(12) | Auto | **0** |
| Timer 0 Interrupt(TF0) | 000B | - | Auto | **1** |
| External HW Interrupt 1 (INT1) | 0013 | P3.3(13) | Auto | **2** |
| Timer 1 Interrupt(TF1) | 001B | - | Auto | **3** |
| Serial Com Interrupt(RI and TI) | 0023 | - | Program SW | **4** |

# 8051 PROGRAMMING IN C –Interrupt Structure

- Reset vector has just 3 bytes allocated to it. It can hold a jump instruction to the location where the main program is stored
- The other interrupts have 8 bytes allocated to each of them, hence a small Interrupt service routine(ISR) can be placed here
- However, if the ISR needs to larger in length, it has to placed else where and the allocated 8 bytes need to have the code that simple redirects the control to the ISR

# 8051 PROGRAMMING IN C –Interrupt Structure

- INT0 and INT1 are external interrupts on P3.2 and P3.3 respectively. These can be configured to be low level triggered or edge triggered interrupt sources
- TF0 and TF1 are timer overflow interrupts for timer 0 and 1 respectively
- The Serial COM Interrupt can be configured to trigger upon transmit or receipt of a byte during serial communication

# 8051 PROGRAMMING IN C –Interrupt Enable/Disable

- When the MCU is reset, all the interrupts are disabled
- In 8051 Interrupt Enable (EA) Register is used to enable or disable the interrupt

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EA | - | ET2 | ES | ET1 | EX1 | ET0 | EX0 |

- **EA**: Global interrupt-controlled bit

# 8051 PROGRAMMING IN C – ISR examples

- **External Interrupt 0**

```
void ex_0() interrupt 0
{
        <Body of ISR>
}
```

- **Timer 0**

```
void timer0() interrupt 1
{
        <Body of ISR>
}
```

- **External Interrupt 1**

```
void ex_1() interrupt 2
{
        <Body of ISR>
}
```

- **Timer 1**

```
void timer1() interrupt 3
{
        <Body of ISR>
}
```

# 8051 PROGRAMMING IN C –Interrupt Examples - Timer

Blinking an LED using Timer interrupt in 8051

```
#include <reg51.h>
sbit mybit = P2^3;
void main()
{
    TMOD = 0x01;
    TH0 = 0xFC;
    TL0 = 0x66;
    IE = 0x82; // enable interrupt
    TR0 = 1;   //start timer
    while(1);
}
```

```
void timer0(void) interrupt 1
//interrupt number 1 for Timer
0
{
      mybit = ~mybit;
//Toggles the LED on interrupt

      TH0=0xFC;
      TL0=0x66;
}
```

# 8051 PROGRAMMING IN C –Interrupt Examples – External Interrupt

- Use both the external interrupts INT0(P3.2) and INT1(P3.1)

- A low signal on these pins will generates the corresponding interrupts causing it to execute the respective ISR's

- INTO ISR will increment a counter
- INT1 will decrement the same counter

- The main program will just display this 8-bit interrupt counter on LED's connected to P0

# 8051 PROGRAMMING IN C –Interrupt Examples – External Interrupt

```
#include<reg51.h>
#define LEDs P0
unsigned char count=0;

void ext_int_0 () interrupt 0
{
    count++;
}

void ext_int_1 () interrupt 2
{
   count--;
}
```

```
void main()
{
    P3 |= 0x0c;  // Configure the INT0 &
INT1 pins as Inputs
    EX0 = 1;     // Enable INT0
    EX1 = 1;     // Enable INT1
    EA  = 1;     // Enable Global Interrupt
bit

    while(1)
    {
      LEDs = count;
    }}
```

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# Sensors

- A **sensor** is a device that receives a stimulus and responds with an electrical signal
- The terms **sensor** and **detector** are synonyms
- Different sensors are IR sensors, proximity sensors, temperature sensors, tilt sensors, accelerometers, ultrasonic sensors, RADAR, SONAR, etc.

# Sensors

- The **stimulus** is the quantity, property, or condition that is received and converted into electrical signal
- Examples of stimuli are light intensity and wavelength, sound, force, acceleration, distance, rate of motion, and chemical composition
- When we say “electrical”, we mean a signal which can be channeled, amplified, and modified by electronic devices
- The term sensor should be distinguished from transducer
- The transducer is a converter of any one type of energy or property into another type of energy or property, whereas the sensor converts it into electrical signal

# Sensors - Characteristics

- Range
- Span
- Accuracy
- Precision
- Sensitivity
- Linearity
- Hysteresis
- Resolution
- Repeatability

# Sensors - Types

- Temperature Sensors
- Pressure Sensors
- Flow Sensors
- Level Sensors
- Imaging sensors
- Noise Sensors
- Air Pollution Sensors
- Proximity and displacement Sensors
- Infrared Sensors
- Moisture and Humidity Sensors
- Speed Sensors

# Actuators

- The actuator is a device that transforms a certain form of energy into motion
- As their mechanism converts energy into motion, we can categorize them based on energy sources:
  - **Pneumatic**: use compressed air for generating motion.
  - **Hydraulic**: use the liquid for generating motion.
  - **Thermal**: use a heat source for generating motion.
  - **Electric**: use external energy sources such as batteries or other types of electric energy to generate motion.

# Analog to Digital Converter (ADC)

- Analog-to-digital converters are among the most widely used devices for data acquisition
- Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous)
- Temperature, pressure (wind or liquid), humidity, and velocity are a few examples of physical quantities that we deal with every day
- Sensors for temperature, velocity, pressure, light, and many other natural quantities produce an output that is voltage (or current)
- Therefore, we need an analog-to-digital converter to translate the analog signals to digital numbers so that the microcontroller can read and process them

# Analog to Digital Converter (ADC)

- Analog-to-digital converter translates analog signals to digital numbers so that the microcontroller can read and process them
- An ADC has n-bit resolution where n can be 8, 10, 12, 16 or 24 bits
- The **higher-resolution ADC** provides a **smaller step size**, where step size is the smallest change that can be discerned by an ADC
- For $n$ bit ADC, number of steps = $2^n$
- Considering the supply voltage for ADC, Vcc = 5 volts
- Step size of $n$ bit ADC = $\frac{Vcc}{Number\ of\ steps}$

# Analog to Digital Converter (ADC)

- For $n$ bit ADC, number of steps = $2^n$
- Considering the supply voltage for ADC, Vcc = 5 volts
- Step size of $n$ bit ADC = $\frac{Vcc}{Number\ of\ steps}$

| *n*-bit | Number of Steps | Step Size (mV) |
|---|---|---|
| 8 | 256 | 5/256 = 19.53 |
| 10 | 1024 | 5/1024 = 4.88 |
| 12 | 4096 | 5/4096 = 1.2 |
| 16 | 65536 | 5/65536 = 0.076 |

- Step size (resolution) is the smallest change that can be discerned by an ADC

# Analog to Digital Converter (ADC)

- In addition to resolution, conversion time is another major factor in judging an ADC
- Conversion time is defined as the time it takes for an ADC to convert the analog input to a digital (binary) number
- The ADC chips are either parallel or serial
- In **parallel ADC**, we have eight or more pins dedicated to bringing out the binary data, but in serial ADC we have only one pin for data out

# Analog to Digital Conversion

**3** steps involved

- Sampling
- Quantization
- Encoding

# Resolution Example - HW

- An 8-bit digital system is used to convert an analog signal to digital signal for a data acquisition system.
- The voltage range for the conversion is 0-10 V.
- Find the resolution of the system and the value of the least significant bit

# Resolution Example - HW

- The 8-bit converter of the previous example is replaced with a 12 bit system
- Compute the resolution and the value of the least significant bit

# ADC - 0804

- The ADC0804 IC is an 8-bit parallel ADC in the family of the ADC0800 series from National Semiconductor
- It works with +5 V and has a resolution of 8 bits
- In the ADC0804, the conversion time varies depending on the clocking signals applied to the **CLK IN** pin
- But it cannot be faster than 110 μs

# ADC - 0804

- **CS** – Chip Select
  - Chip select is an **active-low input** used to activate the ADC0804 chip
- **RD** – Read
  - This is an **input** signal and is **active low**
  - The ADC converts the analog input to its binary equivalent and holds it in an internal register
  - RD is used to get the converted data out of the ADC0804 chip
  - When CS = 0, if a **high-to-low** pulse is applied to the RD pin
  - the 8-bit digital output shows up at the D0–D7 data pins
  - The RD pin is also referred to as **Output Enable (OE)**

# ADC - 0804

- **WR** – Write – Start of Conversion
  - This is an active-low input used to inform the ADC0804 to start the conversion process
  - If CS = 0 when WR makes a **low-to-high** transition
  - ADC0804 starts converting the analog input value of Vin to an 8-bit digital number
  - The amount of time it takes to convert varies depending on the CLK IN and CLK R values
  - When the data conversion is complete, the INTR pin is forced low by the ADC0804

# ADC - 0804

- **CLK IN and CLK R**
  - CLK IN is an input pin connected to an external clock source when an external clock is used for timing
  - 0804 has an internal clock generator
  - To use the internal clock generator (also called self-clocking) of the ADC0804, the CLK IN and CLK R pins are connected to a capacitor and a resistor
  - In that case, the clock frequency is determined by the equation:
  - $f = \frac{1}{1.1\,RC}$

# ADC - 0804

- **CLK IN and CLK R**
  - In that case, the clock frequency is determined by the equation:
  - $f = \frac{1}{1.1\,RC}$
  - Typical values are R = 10K ohms and C = 150 pF
  - f = 606 kHz
  - In that case, the conversion time is 110 μs

# ADC - 0804

- **INTR** – Interrupt – End of Conversion
  - This is an output pin and is active low
  - It is a normally high pin
  - When the conversion is finished, it goes low to signal the CPU that the converted data is ready to be picked up
  - After INTR goes low, we make CS = 0
  - Send a high-to-low pulse to the RD pin to get the data out of the ADC0804 chip

# ADC - 0804

- **Vin (+) and Vin (−)**
  - These are the differential analog inputs
  - Vin = Vin (+) − Vin (−)
  - Often the Vin (−) pin is connected to ground
  - Vin (+) pin is used as the analog input to be converted to digital
- **VCC**
  - This is the +5 V power supply
  - It is also used as a reference voltage when Vref/2 input (pin 9) is open

# ADC - 0804

- **Vref/2**
  - Pin 9 is an input voltage used for the reference voltage
  - If this pin is open (not connected), the analog input voltage for the ADC0804 is in the range of 0 to 5 V
  - However, there are many applications where the analog input applied to Vin needs to be other than the 0 to +5 V range
  - Vref /2 is used to implement analog input voltages other than 0 to 5 V
  - For example, if the analog input range needs to be 0 to 4 V, Vref/2 is connected to 2 V

# ADC - 0804

- **D0-D7**
  - D0–D7 (D7 is the MSB) are the digital data output pins since ADC0804 is a parallel ADC chip
  - The converted data is accessed only when CS = 0 and RD is forced low
  - To calculate the output voltage, use the following formula
  - $Dout = \frac{Vin}{Step\ Size}$
  - Dout = digital data output (in decimal)
  - Vin = analog input voltage
  - step size (resolution) is the smallest change which is for ADC0804 = (2 × Vref/2)/256

# ADC - 0804

- **A Ground, D Ground**
  - These are the input pins providing the ground for both the analog signal and the digital signal
  - These are the input pins providing the ground for both the analog signal and the digital signal
  - The reason that we have two ground pins is to isolate the analog Vin signal
  - Such isolation contributes to the accuracy of the digital data output
  - In our discussion, both are connected to the same ground; however, in the real world of data acquisition, the analog and digital grounds are handled separately

# ADC – 0804 – Conversion Steps

- **Analog to Digital Conversion ADC0804**
- Make CS = 0 and send a low-to-high pulse to pin WR to start the conversion
- Keep monitoring the INTR pin If INTR is low, the conversion is finished, and we can go to the next step. If INTR is high, keep polling until it goes low
- After the INTR has become low, we make CS = 0 and send a high-tolow pulse to the RD pin to get the data out of the ADC0804 IC chip

# ADC – 0804 – Conversion Steps

- **Analog to Digital Conversion ADC0804**

# ADC – 0804 – Clock

- **Clock source for ADC0804**
- The speed at which an analog input is converted to the digital output depends on the speed of the CLK input
- According to the ADC0804 data sheets, the typical operating frequency is approximately 640 kHz at 5 V
- Two ways of providing clock to the ADC0804
  - **Internal Clock**
  - **External Clock**

# ADC – 0804 – Clock

- Internal clock to the ADC0804

# ADC – 0804 – Clock

- External clock to the ADC0804

# ADC – 0804 – Clock

- External clock to the ADC0804

# ADC – 0804 – Programming

```
#include <reg51.h>
sbit RD = P2^5;
sbit WR = P2^6;
sbit INTR = P2^7;
sfr MYDATA = P1;
void main() {
unsigned char value;
MYDATA = 0xFF;   //make P1 as Input
INTR = 1;        //make INTR as Input
RD = 1; WR = 1;  //set RD and WR Input
while (1){
WR = 0;
WR = 1; //send WR L to H pulse
while (INTR ==1); //wait for EOC
RD = 0 ; //send RD H to L pulse
Value = MYDATA; //read value
RD = 1; //reading complete
}}
```

# Digital to Analog Converter

- Digital input to analog output
- Mainly used in the output stage
- DAC can reconstruct sampled data into an analog signal with precision
- 2 types
  - Binary Weighted resistor
  - R-2R ladder

# Binary Weighted resistor - DAC

Summing amplifier with digitally controlled inputs

Rules of Ideal OP AMPs $I_{in} = 0$, $Z_{in}$ = infinity

$$I_A = I_1 + I_2 + I_3 \ldots + I_n$$

$$I_1 = \frac{V}{R}, \quad I_2 = \frac{V}{2R}, \quad I_3 = \frac{V}{4R}, \ldots \quad I_n = \frac{V}{(2^{n-1})R}$$

$$I_A = B(n-1)\cdot\left(\frac{V}{R}\right) + B(n-2)\cdot\left(\frac{V}{2R}\right) + B(n-3)\cdot\left(\frac{V}{4R}\right) \ldots + B0\cdot\left(\frac{V}{(2^{n-1})R}\right)$$

Formula for output V

$$V_0 = -I_F \cdot R_F$$

$$V_0 = -R_F \sum_{i=1}^{n} \frac{B(n-i)\cdot V}{2^{i-1}R}$$

B0, B(n-3), B(n-2), ....B(n-1) take on values of 1 or 0 depending of the digital output controlling switch

# Binary Weighted resistor - DAC

- For the binary-weighted resistor DAC, find the output when the input word is 1101, V = 10 Vdc, $R_f = R$

$$V_0 = -R_F \sum_{i=1}^{n} \frac{B(n-i)\cdot V}{2^{i-1}R}$$

n=4

B(4-1)=B3=1 MSB
B(4-2)=B2=1
B(4-3)=B1=0
B(4-4)=B0=1 LSB

$$V_o = -R \cdot \left[ \frac{B3 \cdot V}{2^{1-1}R} + \frac{B2 \cdot V}{2^{2-1}R} + \frac{B1 \cdot V}{2^{3-1}R} + \frac{B0 \cdot V}{2^{4-1}R} \right]$$

$$V_o = -\frac{R}{R} \cdot \left[ \frac{1 \cdot V}{2^0} + \frac{1 \cdot V}{2^1} + \frac{0 \cdot V}{2^2} + \frac{1 \cdot V}{2^3} \right]$$

$$V_o = -\left[ 1 + \frac{1}{2} + 0 + \frac{1}{8} \right] = -[1 + 0.5 + 0 + 1.2] = -5.1 \quad .2 \quad 6$$

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# ADC – 0804 – Clock

- **Clock source for ADC0804**
- The speed at which an analog input is converted to the digital output depends on the speed of the CLK input
- According to the ADC0804 data sheets, the typical operating frequency is approximately 640 kHz at 5 V
- Two ways of providing clock to the ADC0804
  - **Internal Clock**
  - **External Clock**

# ADC – 0804 – Clock

- Internal clock to the ADC0804

# ADC – 0804 – Clock

- External clock to the ADC0804

# ADC – 0804 – Programming

```
#include <reg51.h>
sbit RD = P2^5;
sbit WR = P2^6;
sbit INTR = P2^7;
sfr MYDATA = P1;
void main() {
unsigned char value;
MYDATA = 0xFF;   //make P1 as Input
INTR = 1;        //make INTR as Input
RD = 1; WR = 1;  //set RD and WR Input
while (1){
WR = 0;
WR = 1; //send WR L to H pulse
while (INTR ==1); //wait for EOC
RD = 0 ; //send RD H to L pulse
Value = MYDATA; //read value
RD = 1; //reading complete
}}
```

# Digital to Analog Converter

- Digital input to analog output
- Mainly used in the output stage
- DAC can reconstruct sampled data into an analog signal with precision
- 2 types
  - Binary Weighted resistor
  - R-2R ladder

# Binary Weighted resistor - DAC

Summing amplifier with digitally controlled inputs

Rules of Ideal OP AMPs $I_{in} = 0$, $Z_{in}$ = infinity

$$I_A = I_1 + I_2 + I_3 \ldots + I_n$$

$$I_1 = \frac{V}{R}, \quad I_2 = \frac{V}{2R}, \quad I_3 = \frac{V}{4R}, \ldots \quad I_n = \frac{V}{(2^{n-1})R}$$

$$I_A = B(n-1)\cdot\left(\frac{V}{R}\right) + B(n-2)\cdot\left(\frac{V}{2R}\right) + B(n-3)\cdot\left(\frac{V}{4R}\right) \ldots + B0\cdot\left(\frac{V}{(2^{n-1})R}\right)$$

Formula for output V

$$V_0 = -I_F \cdot R_F$$

$$V_0 = -R_F \sum_{i=1}^{n} \frac{B(n-i)\cdot V}{2^{i-1}R}$$

B0, B(n-3), B(n-2), ....B(n-1) take on values of 1 or 0 depending of the digital output controlling switch

# Binary Weighted resistor - DAC

- For the binary-weighted resistor DAC, find the output when the input word is 1101, V = 10 Vdc, $R_f = R$

$$V_0 = -R_F \sum_{i=1}^{n} \frac{B(n-i)\cdot V}{2^{i-1}R}$$

n=4

B(4-1)=B3=1 MSB
B(4-2)=B2=1
B(4-3)=B1=0
B(4-4)=B0=1 LSB

$$V_o = -R\cdot\left[\frac{B3\cdot V}{2^{1-1}R} + \frac{B2\cdot V}{2^{2-1}R} + \frac{B1\cdot V}{2^{3-1}R} + \frac{B0\cdot V}{2^{4-1}R}\right]$$

$$V_o = -\frac{R}{R}\cdot\left[\frac{1\cdot V}{2^0} + \frac{1\cdot V}{2^1} + \frac{0\cdot V}{2^2} + \frac{1\cdot V}{2^3}\right]$$

$$V_o = -\left[1 + \frac{1}{2} + 0 + \frac{1}{8}\right] = -[1 + 0.5 + 0 + 1.2] = -5.1 \quad .2\ 6$$

# 8051 – Input Output Peripherals

- **Input**
- Keypad

- **Output**
- LCD

# 8051 – Output Peripheral - LCD

- In recent years, the LCD is finding widespread use replacing LEDs (seven-segment LEDs or other multisegmented LEDs)
- LCDs have ability to display numbers, characters, and graphics.
- This contrasts with LEDs, which are limited to numbers and a few characters.

# 8051 – Output Peripheral - LCD

- **16x2 LCD**
- 16 Columns x 2 Rows
- Operating voltage 4.7V to 5.3V
- Pixel box of each character is 5×8 pixel
- 14 Pins
- 16 Pins

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **VCC, VSS, VEE**
- Vcc = Supply Voltage = 5V
- Vss = Ground = 0V
- VEE = Controlling LCD contrast

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **RS – Register Select**
- There are two important registers inside LCD
  - Command (Code/Control) Register
  - Data Register
- If RS = 0, Command Register
  - Allowing the user to send a command such as clear display or cursor at home
- If RS = 1, Data Register
  - Allowing the user to send a command such as clear display or cursor at home

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **R/W – Read/Write**
- R/W input allows the user to write information to the LCD or read information from it
- R/W = 1 when reading
- R/W = 0 when writing

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **E - enable**
- The enable pin is used by the LCD to latch information presented to its data pins
- When data is supplied to data pins, a high-to-low pulse must be applied to this pin for the LCD to latch in the data present at the data pins
- This pulse must be a minimum of 450 ns wide

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **D0–D7**
- The 8-bit data pins, D0–D7, are used to send information to the LCD
- Or they can be used to read the contents of the LCD's internal registers
- To display letters and numbers, we send ASCII codes for the letters A–Z, a–z, and numbers 0–9 to these pins while making RS = 1

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- Instruction command codes **(Hex)** that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor

- We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information

- The busy flag is D7 and can be read when R/W = 1

| | |
|---|---|
| 1 | Clear display screen |
| 2 | Return home |
| 4 | Decrement cursor (shift cursor to left) |
| 6 | Increment cursor (shift cursor to right) |
| 5 | Shift display right |
| 7 | Shift display left |
| 8 | Display off, cursor off |
| A | Display off, cursor on |
| C | Display on, cursor off |
| E | Display on, cursor blinking off |
| F | Display on, cursor blinking |
| 10 | Shift cursor position to left |
| 14 | Shift cursor position to right |
| 18 | Shift the entire display to the left |
| 1C | Shift the entire display to the right |
| 80 | Force cursor to beginning of 1st line |
| C0 | Force cursor to beginning of 2nd line |
| 38 | 2 lines and 5x7 matrix |

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# 8051 – Input Output Peripherals

- **Input**
- Keypad

- **Output**
- LCD

# 8051 – Output Peripheral - LCD

- In recent years, the LCD is finding widespread use replacing LEDs (seven-segment LEDs or other multisegmented LEDs)
- LCDs have ability to display numbers, characters, and graphics.
- This contrasts with LEDs, which are limited to numbers and a few characters.

# 8051 – Output Peripheral - LCD

- **16x2 LCD**
- 16 Columns x 2 Rows
- Operating voltage 4.7V to 5.3V
- Pixel box of each character is 5×8 pixel
- 14 Pins
- 16 Pins

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **VCC, VSS, VEE**
- Vcc = Supply Voltage = 5V
- Vss = Ground = 0V
- VEE = Controlling LCD contrast

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **RS – Register Select**
- There are two important registers inside LCD
  - Command (Code/Control) Register
  - Data Register
- If RS = 0, Command Register
  - Allowing the user to send a command such as clear display or cursor at home
- If RS = 1, Data Register
  - Allowing the user to send a command such as clear display or cursor at home

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **R/W – Read/Write**
- R/W input allows the user to write information to the LCD or read information from it
- R/W = 1 when reading
- R/W = 0 when writing

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **E - enable**
- The enable pin is used by the LCD to latch information presented to its data pins
- When data is supplied to data pins, a high-to-low pulse must be applied to this pin for the LCD to latch in the data present at the data pins
- This pulse must be a minimum of 450 ns wide

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- **D0–D7**
- The 8-bit data pins, D0–D7, are used to send information to the LCD
- Or they can be used to read the contents of the LCD's internal registers
- To display letters and numbers, we send ASCII codes for the letters A–Z, a–z, and numbers 0–9 to these pins while making RS = 1

| Pin | Symbol | I/O | Description |
|---|---|---|---|
| 1 | $V_{ss}$ | -- | Ground |
| 2 | $V_{cc}$ | -- | +5 V power supply |
| 3 | $V_{ee}$ | -- | Power supply to control contrast |
| 4 | RS | I | RS = 0 to select command register, RS = 1 to select data register |
| 5 | R/W | I | R/W = 0 for write, R/W = 1 for read |
| 6 | E | I | Enable |
| 7 | DB0 | I/O | The 8-bit data bus |
| 8 | DB1 | I/O | The 8-bit data bus |
| 9 | DB2 | I/O | The 8-bit data bus |
| 10 | DB3 | I/O | The 8-bit data bus |
| 11 | DB4 | I/O | The 8-bit data bus |
| 12 | DB5 | I/O | The 8-bit data bus |
| 13 | DB6 | I/O | The 8-bit data bus |
| 14 | DB7 | I/O | The 8-bit data bus |

# 8051 – Output Peripheral - LCD

- Instruction command codes **(Hex)** that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor

- We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information

- The busy flag is D7 and can be read when R/W = 1

| | |
|---|---|
| 1 | Clear display screen |
| 2 | Return home |
| 4 | Decrement cursor (shift cursor to left) |
| 6 | Increment cursor (shift cursor to right) |
| 5 | Shift display right |
| 7 | Shift display left |
| 8 | Display off, cursor off |
| A | Display off, cursor on |
| C | Display on, cursor off |
| E | Display on, cursor blinking off |
| F | Display on, cursor blinking |
| 10 | Shift cursor position to left |
| 14 | Shift cursor position to right |
| 18 | Shift the entire display to the left |
| 1C | Shift the entire display to the right |
| 80 | Force cursor to beginning of 1st line |
| C0 | Force cursor to beginning of 2nd line |
| 38 | 2 lines and 5x7 matrix |

# 8051 – Output Peripheral - LCD

- Sample connection of 16x2 LCD with 8051 Microcontroller

- P1 = Data Lines for LCD
- P2.0 = RS
- P2.1 = R/W
- P2.2 = E

# 8051 – Output Peripheral – LCD - Code

```
#include<reg51.h>

sfr ldata = 0x90; //P1=LCD data pins

sbit rs = P2^0;

sbit rw = P2^1;

sbit en = P2^2;

void main()

{

Logic

}
```

**Logic**

```
lcdcmd(0x38);

MSDelay(250);

lcdcmd(0x0E);

MSDelay(250);

lcdcmd(0x01);

MSDelay(250);

 lcdcmd(0x06);

MSDelay(250);

lcdcmd(0x86); //line 1,position 6

MSDelay(250);
```

| | |
|---|---|
| 1 | Clear display screen |
| 2 | Return home |
| 4 | Decrement cursor (shift cursor to left) |
| 6 | Increment cursor (shift cursor to right) |
| 5 | Shift display right |
| 7 | Shift display left |
| 8 | Display off, cursor off |
| A | Display off, cursor on |
| C | Display on, cursor off |
| E | Display on, cursor blinking off |
| F | Display on, cursor blinking |
| 10 | Shift cursor position to left |
| 14 | Shift cursor position to right |
| 18 | Shift the entire display to the left |
| 1C | Shift the entire display to the right |
| 80 | Force cursor to beginning of 1st line |
| C0 | Force cursor to beginning of 2nd line |
| 38 | 2 lines and 5x7 matrix |

# 8051 – Output Peripheral – LCD - Code

Logic

```
line 1,position 6

MSDelay(250);

lcddata('V');

 MSDelay(250);

 lcddata('I');

 MSDelay(250);

 lcddata('T');

}
```

```
void lcdcmd (unsigned char value)

{

ldata = value; // put the value on the pins

 rs = 0;

 rw = 0;

 en = 1; // strobe the enable pin

 MSDelay(1);

 en = 0;

 return;

}
```

```
void lcddata (unsigned char value)

{

ldata = value; // put the value on the pins

 rs = 1;

 rw = 0;

 en = 1; // strobe the enable pin

 MSDelay(1);

 en = 0;

 return;

}
```

# 8051 – Input Peripheral - Keypad

- At the lowest level, keyboards are organized in a matrix of rows and columns
- The CPU accesses both rows and columns through ports; therefore, with two 8-bit ports, an 8 x 8 matrix of keys can be connected to a microprocessor
- When a key is pressed, a row and a column make a contact; otherwise, there is no connection between rows and columns

# 8051 – Input Peripheral - Keypad

- 4 x 4 matrix connected to two ports
- The rows are connected to an output port
- The columns are connected to an input port
- If no key has been pressed, reading the input port will yield 1s for all columns
- If all the rows are grounded and a key is pressed, one of the columns will have 0

# 8051 – Input Peripheral - Keypad

- To detect a pressed key, the microcontroller grounds all rows by providing 0 to the output latch and then it reads the columns
- If the data read from the columns is D3–D0 = 1111, no key has been pressed and the process continues until a key press is detected
- If one of the column bits has a zero, this means that a key press has occurred
- If D3–D0 = 1101, this means that a key in the **D1 column has been pressed**

# 8051 – Input Peripheral - Keypad

- After a key press is detected, the microcontroller will go through the process of identifying the key
- Starting with the top row, the microcontroller grounds it by providing a low to row D0 only; then it reads the columns
- If the data read is all 1s, no key in that row is activated and the process is moved to the next row

# 8051 – Input Peripheral - Keypad

- It grounds the next row, reads the columns, and checks for any zero
- This process continues until the row is identified

# 8051 – Input Peripheral – Keypad - Code

```
//Keypad Connections

sbit R1 = P1^0;

sbit R2 = P1^1;

sbit R3 = P1^2;

sbit R4 = P1^3;

sbit C1 = P1^4;

sbit C2 = P1^5;

sbit C3 = P1^6;

sbit C4 = P1^7;
```

```
char Read_Keypad()

{

 C1=1; C2=1; C3=1; C4=1;

 R1=0; R2=1; R3=1; R4=1;

 if(C1==0){Delay(100);while(C1==0);return '7';}

if(C2==0){Delay(100);while(C2==0);return '8';}

if(C3==0){Delay(100);while(C3==0);return '9';}

if(C4==0){Delay(100);while(C4==0);return '/';}

R1=1; R2=0; R3=1; R4=1;

 if(C1==0){Delay(100);while(C1==0);return '4';}

if(C2==0){Delay(100);while(C2==0);return '5';}

if(C3==0){Delay(100);while(C3==0);return '6';}

if(C4==0){Delay(100);while(C4==0);return 'X;}
```

# 8051 – Input Peripheral – Keypad - Code

```
R1=1; R2=1; R3=0; R4=1;

 if(C1==0){Delay(100);while(C1==0);return '1';}

if(C2==0){Delay(100);while(C2==0);return '2';}

if(C3==0){Delay(100);while(C3==0);return '3';}

if(C4==0){Delay(100);while(C4==0);return '-';}
```

```
R1=1; R2=1; R3=1; R4=0;

 if(C1==0){Delay(100);while(C1==0);return 'C';}

if(C2==0){Delay(100);while(C2==0);return '0';}

if(C3==0){Delay(100);while(C3==0);return '=';}

if(C4==0){Delay(100);while(C4==0);return '+;}
```

# 8051 – Memory Interfacing

- EA Pin
  - EA = **1**, program code is stored in the on-chip ROM
  - EA = **0**, program code is stored in the external ROM
- P0 and P2
  - P0 provides the lower 8-bit addresses A0–A7
  - P2 provides the upper 8-bit addresses A8–A15
  - P0 is also used to provide the 8-bit data bus D0–D7
- ALE Pin
- PSEN Pin

# 8051 – Memory Interfacing

- EA Pin
  - EA = **1**, program code is stored in the on-chip ROM
  - EA = **0**, program code is stored in the external ROM
- P0 and P2
  - P0 provides the lower 8-bit addresses A0–A7
  - P2 provides the upper 8-bit addresses A8–A15
  - P0 is also used to provide the 8-bit data bus D0–D7
  - P0.0–P0.7 are used for both the address and data paths
  - This is called address/data multiplexing in chip design

# 8051 – Memory Interfacing

- To know when P0 is used for Data and when P0 is used for address?
- ALE Pin
  - **ALE = 0**, the 8051 uses P0 for the **data path**
  - **ALE = 1**, the 8051 uses P0 for the **address path**

**Funtion Table**

| Output control | Enable G | Enable D | Output |
|---|---|---|---|
| L | H | H | H |
| L | H | L | L |
| L | L | X | Q0 |
| H | X | X | Z |

# 8051 – Memory Interfacing

- Normally, ALE = 0, and P0 is used as a data bus, sending data out or bringing data in
- Whenever the 8051 wants to use P0 as an address bus, it puts the addresses A0–A7 on the P0 pins and activates ALE = 1 to indicate that P0 has the addresses

# 8051 – Memory Interfacing

- PSEN Pin
  - Program Store Enable
  - **PSEN** connected to the **OE pin of a ROM** containing the program code
  - In other words, to access external ROM containing program code, the 8051 uses the PSEN signal
- EA
  - EA = 0, 8051 fetches opcode from external ROM by using PSEN
  - EA = 1, 8051 uses its internal (on-chip) ROM

# 8051 – Memory Interfacing

- ROM space allocation

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# 8051 PROGRAMMING IN C

# 8051 – Memory Interfacing

- EA Pin
  - EA = **1**, program code is stored in the on-chip ROM
  - EA = **0**, program code is stored in the external ROM
- P0 and P2
  - P0 provides the lower 8-bit addresses A0–A7
  - P2 provides the upper 8-bit addresses A8–A15
  - P0 is also used to provide the 8-bit data bus D0–D7
- ALE Pin
- PSEN Pin

# 8051 – Memory Interfacing

- EA Pin
  - EA = **1**, program code is stored in the on-chip ROM
  - EA = **0**, program code is stored in the external ROM
- P0 and P2
  - P0 provides the lower 8-bit addresses A0–A7
  - P2 provides the upper 8-bit addresses A8–A15
  - P0 is also used to provide the 8-bit data bus D0–D7
  - P0.0–P0.7 are used for both the address and data paths
  - This is called address/data multiplexing in chip design

# 8051 – Memory Interfacing

- To know when P0 is used for Data and when P0 is used for address?
- ALE Pin
  - **ALE = 0**, the 8051 uses P0 for the **data path**
  - **ALE = 1**, the 8051 uses P0 for the **address path**

**Funtion Table**

| Output control | Enable G | Enable D | Output |
|---|---|---|---|
| L | H | H | H |
| L | H | L | L |
| L | L | X | Q0 |
| H | X | X | Z |

# 8051 – Memory Interfacing

- Normally, ALE = 0, and P0 is used as a data bus, sending data out or bringing data in
- Whenever the 8051 wants to use P0 as an address bus, it puts the addresses A0–A7 on the P0 pins and activates ALE = 1 to indicate that P0 has the addresses

# 8051 – Memory Interfacing

- PSEN Pin
  - Program Store Enable
  - **PSEN** connected to the **OE pin of a ROM** containing the program code
  - In other words, to access external ROM containing program code, the 8051 uses the PSEN signal
- EA
  - EA = 0, 8051 fetches opcode from external ROM by using PSEN
  - EA = 1, 8051 uses its internal (on-chip) ROM

# 8051 – Memory Interfacing

- ROM space allocation

# Building Embedded Hardware

# PCB: Printed Circuit Board

- These boards are made up of special materials that do not conduct electricity such as fiber and glass.
- The circuits are designed on the boards with copper tracks instead of wires for the conduction of electricity between the electronic components.
- The printed circuit boards used in all electronic products such as automotives, wireless devices, Robotic applications, etc.
- Offers quick functioning, access, control, monitoring and precise and exact results when compared to other wiring methods-based devices

# What Are The Parts of a PCB?

## Substrate

- The first, and most important, is the substrate, usually made of fiberglass.
- Fiberglass is used because it provides a core strength to the PCB and helps resist breakage.
- Think of the substrate as the PCB's "skeleton".

## Copper Layer

- Depending on the board type, this layer can either be copper foil or a full-on copper coating.
- Regardless of which approach is used, the point of the copper is still the same — to carry electrical signals to and from the PCB, much like your nervous system carries signals between your brain and your muscles.

# What Are The Parts of a PCB?

## Solder Mask

- The third piece of the PCB is the solder mask, which is a layer of polymer that helps protect the copper so that it doesn't short-circuit from coming into contact with the environment.
- In this way, the solder mask acts as the PCB's "skin".

## Silkscreen

- The final part of the circuit board is the silkscreen.
- The silkscreen is usually on the component side of the board used to show part numbers, logos, symbols switch settings, component reference and test points.
- The silkscreen can also be known as legend or nomenclature

# PCB Design Process

# PCB Design Process

## Step 1: Design the PCB Circuit with a Software

- Draw the schematic circuit diagram with the PCB layout software such as CAD software, Eagle and Multisim software.
- It is also possible to change the circuit design's position and then to modify it according to your convenience and requirement.

# PCB Design Process

## Step 2: Film generation

- The film is generated from the finalized circuit board diagram of the PCB layout software which is sent to the manufacturing unit where the negative image or mask is printed out on a plastic sheet.

## Step 3: Select Raw Material

- The bulk of the printed circuit board is made with an unbreakable glass or fiberglass having copper foil bonded unto one or both the sides of the board.
- The PCBs made from unbreakable paper phenolic with a bonded copper foil are less expensive and are often used in household electrical devices.

# PCB Design Process

## Step 4: Preparing Drill Holes

- Machines and carbide drills are used to put holes on the printed circuit board.
- There are two types of machines available to drill the PCBs;
  - Hand machines
  - CNC machines.
- The hand machines require human intervention or effort to drill the holes, whereas CNC machines are computer-based machines that work-based on the machine timetables or programs that run both automatic as well as manually.

# PCB Design Process

## Step 5: Apply Image

- The printed circuit layout can be printed in different ways on PCBs like a manual pen, dry transfers, pen plotters, and printers. The laser printers are a better way to print the layouts on printed circuit boards.

# PCB Design Process

## Step 6: Stripping and Etching

- This process involves removing the unwired copper on the PCBs by using different types of chemicals like ferric chloride, ammonium per-sulfate, etc.
- Make the solvent by mixing 1% of sodium hydroxide and 10 grams of sodium hydroxide pellets to one liter of water and mix it until everything is dissolved.

# PCB Design Process

## Step 7: Testing

- After finishing the manufacturing process of the Printed Circuit Board, the Board undergoes a testing process to check whether the PCB is working properly. Nowadays many automatic testing equipments are available for the high-volume testing of the PCBs.

# Building Embedded Software

# Software Engineer Tools

There are five main development tools that allows software engineers to get started on development and testing of their applications

- **Simulators**, software that imitates or intended to hardware's behavior without the actual hardware.
- **Emulators**, the hardboard platform that imitates the operation of your intended system.
- **Compilers**, it's the software that allows developers to pre-executable code for their intended architecture.
- **Installers**, a software-hardware combination that allows compiled executable programs to be installed onto a platform.
- **Debuggers**, a hardware-software solution that allows programmers to test and validate their executable programs.
- These tools are important because developing hardware takes time and it's expensive.

# Software Engineer Tools

- The software engineer's role includes the design of  the software product, along with its validation of the hardware platform.
- A complete hardware design and documentation usually finishes first, with software following.
- Simulators and emulators will allow validation of design to occur before the arrival of hardware.
- Compilers, Installers, and debuggers will provide a quick development of features for your embedded target. And an easy way to fix off our bugs.
- Prototyping software fast is important, but not at the cost of quality.

# High quality code

- Write a code that is maintainable, testable, portable, robust, efficient and consistent.
- A common misconception is that the advanced platform a general programmer might use.
- That the advanced hardware will make up for poorly written software.
- This is not true, especially from an embedded perspective. Because even the best hardware cannot make up for buggy, slow, or inefficient embedded code.
- By putting time into architecting your software, writing code effectively. Making it modular, you can extend the lifetime and re-usability of this pieces, of your various projects and platforms.

# Embedded Software

- There are many embedded software languages use in the industry such as C, C++, Java and Ada. But C-Programming is the most widely used language for embedded software design.
- C-Programming has benefits for both low level hardware interactions and high-level software language features.
- This provides portability across different embedded platforms.
- Typical embedded engineers actually write a form of C called Embedded C.

# Embedded C

- Embedded C differs from C because it puts a focus on the following features.
  - Efficient memory management
  - Timing centric operations
  - Direct hardware/IO control
  - Code size constraints
  - Optimized execution
- You can think of Embedded C as optimum features in minimum space and time.

# Language Processing System

- The hardware understands a language, which humans cannot understand.
- We write programs in high-level language, which is easier for us to understand and remember.
- These programs are then fed into a series of tools and OS components to get the desired code that can be used by the machine. This is known as Language Processing System.

# Language Processing System

How a program using C compiler is executed on a host machine.

- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program (low-level language).
- An assembler then translates the assembly program into machine code (object).
- A linker tool is used to link all the parts of the program together for execution (executable machine code).
- A loader loads all of them into memory and then the program is executed.

# Compilers

- A compiler is a program that translates code from one language (such as C, C++, or Rust) and outputs it into another language (such as assembly, object code, or machine code).
- A compiler can be used to create an executable binary directly or can generate intermediate files that can be used during other software construction steps, such as linking.
- The name "compiler" is primarily used for programs that translate the source code from a high-level programming language to a low-level language (e.g., assembly language or machine code).
- The most common compilers used for embedded systems are GCC, Clang, Keil, and IAR
- A compiler transforms those input files into object files (usually ending in .o)

# Compilation Process

- The inputs to a compiler are the source files and header files that define our program, as well as an array of optional flags that can be used to control the behavior of the compiler.
- A compiler transforms those input files into object files (usually ending in .o) or an executable program (usually ending in .bin, .hex, .out, .exe, .elf; sometimes there is no ending).

# Cross Compiler

- If the compiled program can run on a system having different CPU or operating system than the system on which the compiler compiled the program, then that compiler is known as a cross-compiler.

| COMPILER | CROSS COMPILER |
| --- | --- |
| A software that translates the computer code written in high-level programming language to machine language | A software that can create executable code for platforms other than the one on which the compiler is running |
| Helps to convert the high-level source code into machine understandable machine code | A type of compiler that can create executable code for different machines other than the machine it runs on |

# Cross Assembler

- A cross assembler takes the conversion process a step further by allowing you to generate machine code for a different processor than the one the compiler is run on.
- Cross assemblers are generally used to develop programs which are supposed to run on game consoles, appliances which are not able to run a development environment.

Cross-Assembler

# Linker

- Linker is a program in a system which helps to link an object modules of program into a single object file.
- It takes object modules from assembler as input and forms an executable file as output for loader.
- Linking is performed at both compile time, when the source code is translated into machine code and load time, when the program is loaded into memory by the loader.
- Linking is performed at the last step in compiling a program.

# Preprocessor

- The preprocessor is a part of the compiler which performs preliminary operations (conditionally compiling code, including files etc...) to your code before the compiler sees it.
- Preprocessor directives change the text of the source code and the result is a new source code without these directives.
- Before the actual compilation of every C program it is passed through a Preprocessor. The Preprocessor looks through the program trying to find out specific instructions called Preprocessor directives that it can understand. All Preprocessor directives begin with the # (hash) symbol.
- e.g. #include <stdio.h> , #define

# Make Files

- A Makefile is the set of instructions that you use to tell compiler how to build your program and are a simple way to organize code compilation.
- This is present with other files in the parent folder of source code.
- The make utility requires a file, **makefile**, which defines set of tasks to be executed.
- You may have used make to compile a program from source code. Most open source projects use make to compile a final executable binary, which can then be installed using **make install**.

# Compiler Tool chains

- A tool chain is a set of tools (chain of tools - compiler, linker, debugger, standard-libraries etc.) that are used to create an executable
- Tool chains are used in the embedded world for cross-compiling, which means creating a program on a host which will eventually run on a different kind of target platform - therefore there is a need to create it with a specific compiler, linker, debugger etc.

Popular tool chains

- GCC - GNU Compiler Collection - a free software collection of compilers, can be set up to cross compile.
- ARMCC - ARM Compiler tool chain – ARM processors are widely used in smartphones e.g., Qualcomm snapdragon processors
- Intel tool chain – proprietary high performance tool chain for Intel processors

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# Serial Bus Examples

| | S/A | Type | Duplex | #Devices | Speed (kbps) | Distance (ft) | Wires |
|---|---|---|---|---|---|---|---|
| RS232 | A | Peer | Full | 2 | 20 | 30 | 2+ |
| RS422 | A | Multi-drop | Half | 10 | 10000 | 4000 | 1+ |
| RS485 | A | Multi-point | Half | 32 | 10000 | 4000 | 2 |
| I2C | S | Multi-master | Half | ? | 3400 | <10 | 2 |
| SPI | S | Multi-master | Full | ? | >1000 | <10 | 3+ |
| Microwire | S | Master/slave | Full | ? | >625 | <10 | 3+ |
| 1-Wire | A | Master/slave | half | ? | 16 | 1000 | 1+ |

# UART Use Cases

- PC serial port is a UART
- Serializes data to be sent over serial cable
  - De-serializes received data

# UART Use Cases

- Used to be commonly used for internet access

# UART

- Universal Asynchronous Receiver/Transmitter
- Hardware that translates between parallel and serial forms
- Commonly used in conjunction with communication standards such as EIA, RS-232, RS-422 or RS-485

# UART – Hardware Connections

# UART

- For UART and most serial communications, the **baud rate** needs to be set the same on both the transmitting and receiving device
- The **baud rate** is the rate at which information is transferred to a communication channel
- In the serial port context, the set baud rate will serve as the maximum number of bits per second to be transferred

| **Wires** | 2 |
|---|---|
| Speed (Baud Rate) | **9600**, 19200, 38400, 57600, **115200**, 230400, 460800, 921600, 1000000, 1500000 |
| Methods of Transmission | Asynchronous |
| Maximum Number of Masters | 1 |
| Maximum Number of Slaves | 1 |

# UART

- The **UART** interface **does not use a clock signal** to synchronize the transmitter and receiver devices
- UART transmits data **asynchronously**
- Instead of a clock signal, the transmitter generates a bitstream based on its clock signal
- While the receiver is using its internal clock signal to sample the incoming data
- The point of synchronization is managed by having **the same baud rate** on both devices

# UART – Data Transmission

- The mode of transmission is in the form of a **packet**
- A packet consists of a start bit, data frame, a parity bit, and stop bits

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission

**Start Bit**

- UART data transmission line is normally held at a high voltage level when it's not transmitting data
- To start the transfer of data, the transmitting UART pulls the transmission line from high to low for one (1) clock cycle
- When the receiving UART detects the high to low voltage transition, it begins reading the bits in the data frame at the frequency of the baud rate

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission

## Data Frame

- The data frame contains the actual data being transferred
- It can be five (5) bits up to eight (8) bits long if a parity bit is used
- If no parity bit is used, the data frame can be nine (9) bits long
- In most cases, the data is sent with the least significant bit first

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission

## Parity

- Parity describes the evenness or oddness of a number
- The parity bit is a way for the receiving UART to tell if any data has changed during transmission
- Bits can be changed by electromagnetic radiation, mismatched baud rates, or long-distance data transfers

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission

**Parity**

- After the receiving UART reads the data frame, it counts the number of bits with a value of 1 and checks if the total is an even or odd number
- If the parity bit is a 0 (even parity), the 1 or logic-high bit in the data frame should total to an even number
- If the parity bit is a 1 (odd parity), the 1 bit or logic highs in the data frame should total to an odd number

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission

**Parity**

- When the parity bit matches the data, the UART knows that the transmission was free of errors
- But if the parity bit is a 0, and the total is odd, or the parity bit is a 1, and the total is even, the UART knows that bits in the data frame have changed

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission

## Stop Bits

- To signal the end of the data packet, the sending UART drives the data transmission line from a **low voltage to a high voltage** for one (1) to two (2) bit(s) duration

| Start Bit ( 1 bit ) | Data Frame ( 5 to 9 Data Bits ) | Parity Bits ( 0 to 1 bit ) | Stop Bits ( 1 to 2 bits ) |
|---|---|---|---|

# UART – Data Transmission - Steps

## Step 1

- The transmitting UART receives data in parallel from the data bus

# UART – Data Transmission - Steps

## Step 2

- The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame

# UART – Data Transmission - Steps

## Step 3

- The entire packet is sent serially starting from start bit to stop bit from the transmitting UART to the receiving UART
- The receiving UART samples the data line at the preconfigured baud rate

# UART – Data Transmission - Steps

## Step 4

- The receiving UART discards the start bit, parity bit, and stop bit from the data frame

# UART – Data Transmission - Steps

## Step 5

- The receiving UART converts the serial data back into parallel and transfers it to the data bus on the receiving end

# UART – Protocol - Example

- Send the ASCII letter ‘W’ (1010111)

# UART – Summary

- UART stands for Universal Asynchronous Receiver/Transmitter
- It uses two lines viz. TxD and RxD for transmit and receive functions
- It is asynchronous communication
- Hence data rate should be matched between devices wanting to communicate
- It supports data rate of about 230 to 460 Kbps (maximum)
- It supports distance of about 50 feet
- It is also known as RS232 interface

# UART – Summary

- Advantages
  - Hardware complexity is low
  - As this is one to one connection between two devices, software addressing is not required
  - It has parity bit to check quality
- Limitations
  - The data frame size is limited
  - It is suitable for communication between only two devices
  - Proper baud rate should be selected
  - Low data rate

# Serial peripheral interface (SPI)

- One of the most widely used interfaces between microcontroller and peripheral ICs
  - sensors, ADCs, DACs, shift registers, SRAM, and others
- Synchronous, full duplex master-slave-based interface
- The data from the master or the slave is synchronized on the rising or falling clock edge
- Both master and slave can transmit data at the same time
- The SPI interface can be either 3-wire or 4-wire
- Popular 4-wire SPI interface

# Serial peripheral interface (SPI)

- 4-wire SPI devices have four signals:
  - Clock (SPI CLK, SCLK)
  - Chip select (CS)
  - Master out, slave in (MOSI)
  - Master in, slave out (MISO)

- Device that generates the clock signal is called the master
- Data transmitted between the master and the slave is synchronized to the clock generated by the master

# Serial peripheral interface (SPI)

- SPI interfaces can have only one master and can have one or multiple slaves
- CS signal from the master is used to select the slave

- CS - Active low signal and is pulled high to disconnect the slave from the SPI bus
- When multiple slaves are used, an individual chip select signal for each slave is required from the master
- MOSI and MISO are the data lines

# Serial peripheral interface (SPI)

## Data Transmission

- To begin SPI communication, master must send the clock signal and select the slave by enabling the CS signal
- Usually, chip select is an active low signal; hence, the master must send a logic 0 on this signal to select the slave
- SPI is a full-duplex interface; both master and slave can send data at the same time via the MOSI and MISO lines respectively
- During SPI communication, the data is simultaneously transmitted (shifted out serially onto the MOSI/SDO bus) and received (the data on the bus (MISO/SDI) is sampled or read in)

# Serial peripheral interface (SPI)

## Data Transmission

- The serial clock edge synchronizes the shifting and sampling of the data
- The SPI interface provides the user with flexibility to select the rising or falling edge of the clock to sample and/or shift the data
- Refer to the device data sheet to determine the number of data bits transmitted using the SPI interface

# Serial peripheral interface (SPI)

## Clock Polarity and Clock Phase

- Master can select the clock polarity and clock phase

| SPI Mode | CPOL | CPHA | Clock Polarity in Idle State | Clock Phase Used to Sample and/or Shift the Data |
|---|---|---|---|---|
| 0 | 0 | 0 | Logic low | Data sampled on rising edge and shifted out on the falling edge |
| 1 | 0 | 1 | Logic low | Data sampled on the falling edge and shifted out on the rising edge |
| 2 | 1 | 1 | Logic high | Data sampled on the falling edge and shifted out on the rising edge |
| 3 | 1 | 0 | Logic high | Data sampled on the rising edge and shifted out on the falling edge |

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# Serial peripheral interface (SPI)

- 4-wire SPI devices have four signals:
  - Clock (SPI CLK, SCLK)
  - Chip select (CS)
  - Master out, slave in (MOSI)
  - Master in, slave out (MISO)

- Device that generates the clock signal is called the master
- Data transmitted between the master and the slave is synchronized to the clock generated by the master

# Serial peripheral interface (SPI)

- SPI interfaces can have only one master and can have one or multiple slaves
- CS signal from the master is used to select the slave

- CS - Active low signal and is pulled high to disconnect the slave from the SPI bus
- When multiple slaves are used, an individual chip select signal for each slave is required from the master
- MOSI and MISO are the data lines

# Serial peripheral interface (SPI)

## Data Transmission

- To begin SPI communication, master must send the clock signal and select the slave by enabling the CS signal
- Usually, chip select is an active low signal; hence, the master must send a logic 0 on this signal to select the slave
- SPI is a full-duplex interface; both master and slave can send data at the same time via the MOSI and MISO lines respectively
- During SPI communication, the data is simultaneously transmitted (shifted out serially onto the MOSI/SDO bus) and received (the data on the bus (MISO/SDI) is sampled or read in)

# Serial peripheral interface (SPI)

## Data Transmission

- The serial clock edge synchronizes the shifting and sampling of the data
- The SPI interface provides the user with flexibility to select the rising or falling edge of the clock to sample and/or shift the data
- Refer to the device data sheet to determine the number of data bits transmitted using the SPI interface

# Serial peripheral interface (SPI)

## Timing Diagram – Writing (Master Side)

# Serial peripheral interface (SPI)

## Timing Diagram – Writing (Master Side)

- Slave select signal must first be forced low by the master
- Series of 2 times eight clock pulses are issued by the master
- The first eight bits (MOSI)
  - **Write (0),** Read (1)
  - Fixed 0
  - Six bits of address
  - This is the address to be used by the slave device to select one of the internal locations for writing, not the address of the slave on the bus

- Next eight bits – Data (MOSI)

# Serial peripheral interface (SPI)

## Timing Diagram – Reading (Master Side)

# Serial peripheral interface (SPI)

## Timing Diagram – Reading (Master Side)

- Slave select signal must first be forced low by the master
- Series of 2 times eight clock pulses are issued by the master
- The first eight bits (MOSI)
  - Write (0), **Read (1)**
  - Fixed 0
  - Six bits of address
  - This is the address to be used by the slave device to select one of the internal locations for writing, not the address of the slave on the bus
- Next eight bits – Data (MISO)

# Serial peripheral interface (SPI)

## Multi-Slave Configuration

- SPI – Multiple Slaves, Single Master Mode
- Regular Mode or Daisy Chain mode

- Regular Mode
  - Individual CS signal
  - Same MOSI, MISO and SCLK

# Serial peripheral interface (SPI)

## Multi-Slave Configuration

- Daisy Chain Mode
  - Common CS signal
  - Common SCLK and MOSI signals
  - The data from the master is directly connected to the first slave and that slave provides data to the next slave and so on
  - As data is propagated from one slave to the next, the number of clock cycles required to transmit data is proportional to the slave position in the daisy chain

# Serial peripheral interface (SPI)

**Advantages**

- Support full-duplex communication
- Better signal integrity, supporting high-speed applications
- The slave device does not need to address

**Limitations**

- Short transmission distance
- There is no flow control specified, and no acknowledgement mechanism
- More Pin ports are occupied
- The slave device does not need to address
- No form of error check unlike in UART (using parity bit)
- Only 1 master

# I2C Protocol

**Inter-Integrated Circuits (I2C)**

- Developed by Philips Semiconductors
- Transfer of data between a central processor and multiple ICs on the same circuit board using just two common wires
- Owing to its simplicity, it is widely adopted for communication between microcontrollers and sensor arrays, displays, IoT devices, EEPROMs etc
- Type of synchronous serial communication protocol

# I2C Protocol - Features

- Only two common bus lines (wires) are required to control any device/IC on the I2C network
- No need of prior agreement on data transfer rate like in UART communication
- The data transfer speed can be adjusted whenever required
- Simple mechanism for validation of data transferred
- Uses 7-bit addressing system to target a specific device/IC on the I2C bus
- I2C networks are easy to scale
- New devices can simply be connected to the two common I2C bus lines

# I2C Protocol - Hardware

- Two Lines - Serial Clock Line (SCL) and Serial Data Line (SDA)
- The data to be transferred is sent through the SDA wire
- The data is synchronized with the clock signal from SCL
- All the devices/ICs on the I2C network - same SCL and SDA lines

# I2C Protocol - Hardware

- Devices connected to the I2C bus - either masters or slaves
- At any instant of time only a single master stays active on the I2C bus
- It controls the SCL clock line and decides what operation is to be done on the SDA data line

# I2C Protocol - Hardware

- All the devices that respond to instructions from this master device are slaves
- For differentiating between multiple slave devices connected to the same I2C bus, each slave device is physically assigned a **permanent 7-bit address**

# I2C Protocol - Hardware

- When a master device wants to transfer data to or from a slave device, it specifies this particular slave device address on the SDA line and then proceeds with the transfer
- All the other slave devices doesn't respond unless their address is specified by the master device on the SDA line

# I2C Protocol – Data Transfer

- Data is transferred between the master device and slave devices through a single SDA data line
- Via patterned sequences of 0's and 1's (bits)
- Each sequence of 0's and 1's is termed as a transaction

# I2C Protocol – Data Transfer

## Start Condition

- When master device/IC decides to start a transaction
  - switches the SDA line from **high voltage to a low voltage**
  - before the SCL line switches from **high to low**

# I2C Protocol – Data Transfer

## Address Block

- 7 bits and are filled with the address of slave device to/from which the master device needs send/receive data
- All the slave devices on the I2C bus compare these address bits with their address

# I2C Protocol – Data Transfer

**Read/Write Bit**

- This bit specifies the direction of data transfer
- Master device/IC needs to send data to a slave device – 0
- Master device/IC needs to receive data to a slave device – 1

# I2C Protocol – Data Transfer

## ACK/NACK Bit

- If the address of any slave device = address broadcasted by the master device, the value of this bit is set to ‘0’ by the slave device
- Else – 1(Default)

# I2C Protocol – Data Transfer

**Data Block**

- 8 bits
- This block is followed by an ACK/NACK bit
- 0 by the receiver – Receives data, else stays at 1 (Default)

# I2C Protocol – Data Transfer

## Stop Condition

- Master device switches the SDA line from **low to high** before the SCL line switches from **high to low**

Stop Condition

# I2C Protocol – Data Transfer

**Write into Slave device**

- The master device sends the start condition
- The master device sends the 7 address bits which corresponds to the slave device to be targeted
- The master device sets the Read/Write bit to ‘0’. which signifies a write
- Two scenarios are possible
  - If no slave device matches with the address sent by the master device
  - If a slave device exists with the same address as the one specified by the master device

# I2C Protocol – Data Transfer

## Write into Slave device

- If **no slave device matches** with the address sent by the master device
  - The next ACK/NACK bit stays at ‘1’ (default)
  - This signals the master device that the slave device identification is unsuccessful
  - The master clock will end the current transaction by sending a Stop condition or a new Start condition
- If **a slave device exists** with the same address as the one specified by the master device
  - The slave device sets the ACK/NACK bit to ‘0’
  - Which signals the master device that a slave device is successfully targeted

# I2C Protocol – Data Transfer

**Write into Slave device**

- If a slave device is successfully targeted, the master device now sends 8 bits of data
- If the data is successfully received by the slave device, it sets the ACK/NACK bit to ‘0’ which signals the master device to continue
- The previous two steps are repeated until all the data is transferred
- After all the data is sent to the slave device, the master device sends the Stop condition which signals all the slave devices that the current transaction has ended

# I2C Protocol – Data Transfer

## Write into Slave device

# I2C Protocol – Data Transfer

## Reading from Slave device

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# I2C Protocol – Data Transfer

## Write into Slave device

# I2C Protocol – Data Transfer

## Reading from Slave device

# Controlled Area Network (CAN)

- Developed by **Robert Bosch** in around 1986
- Standard designed to allow the microcontroller and other devices to communicate with each other without any host computer
- The node can be a sensor, microcontroller, or a gateway
- The CAN is a message-based protocol
- There is no need for node identification in the CAN network
- It is a serial half-duplex and asynchronous type of communication protocol
- Automobiles

# Controlled Area Network (CAN)

- Henry Ford got the idea to add lights and an electric horn to his cars in 1915, electricity flowed from the battery through switches to the lights and other devices
- By the 1960s there were thousands of heavy wires running throughout every vehicle
- Every bit of extra weight reduces a vehicle's fuel efficiency

# Controlled Area Network (CAN)

- Early 1980s, cars had more and more ECUs (electronic control units) inside them
- Companies of Germany were looking for a type of bus communication system that could be used as a communication system between multiple ECUs and vehicle systems
- They searched the market but couldn’t find exactly what they needed

# Controlled Area Network (CAN)

- German companies began developing the **CAN** in partnership with Mercedes-Benz, Intel, and several universities in Germany
- In 1986, Bosch introduced the CAN standard

# Controlled Area Network (CAN)

# Controlled Area Network (CAN)

- German companies began developing the **CAN** in partnership with Mercedes-Benz, Intel, and several universities in Germany
- In 1986, Bosch introduced the CAN standard

# Controlled Area Network (CAN)

- Devices on a CAN bus are called “nodes”
- Each node consists of a CPU, CAN controller, and a transceiver
- All nodes can send and receive data, but not at the same time

- Nodes cannot send data directly to each other. Instead, they send their data onto the network
- Where it is available to any node to which it has been addressed

# Controlled Area Network (CAN)

# Controlled Area Network (CAN)

- CAN bus uses two dedicated wires for communication
- The wires are called CAN high and CAN low
- When the CAN bus is in idle mode, both lines carry 2.5V
- When data bits are being transmitted, the CAN high line goes to 3.75V and the CAN low drops to 1.25V
- Generating a 2.5V differential between the lines

# Controlled Area Network (CAN)

- With CAN, all data are sent in frames, and there are four types
- **Data frames** transfer data to one or many receiver nodes
- **Remote frames** ask for data from other nodes
- **Error frames** report errors
- **Overload frames** report overload conditions

- There are two variants of the message length: standard and extended

# Controlled Area Network (CAN)

**Standard CAN data message**

**Extended CAN data message**

# Controlled Area Network (CAN)

**Standard CAN data message**

| Field | Bits | Description |
|---|---|---|
| SOF | 1 | This bit marks the start of a message. It synchronizes the nodes after an idle period. |
| Identifier | 11 | The CAN 11-bit identifier data field sets the message priority. Lower values mean higher priorities. |
| RTR | 1 | Remote Transmission Request. This bit is dominant when information is requested by another node. All nodes will receive the request, but the identifier determines the desired node. |
| IDE | 1 | The Identifier Extension bit indicates that a standard CAN identifier (not an extended one) is being transmitted. |
| R0 | 1 | Reserved for future use. |

# Controlled Area Network (CAN)

| Field | Bits | Description |
|---|---|---|
| DLC | 4 | The Data Length Code contains the number of bytes in the transmission. |
| Data | 0 - 64 | The actual data being transmitted. |
| CRC | 16 | The 16-bit (15 bits plus delimiter) cyclic redundancy check (CRC) contains the checksum (number of bits transmitted) |
| ACK | 2 | When a node successfully receives a message, is ACKnowledges it by overwriting this overwrites this bit with a dominant bit. On the other hand, if a node finds an error in a message, it allows this bit to remain recessive and ignores the message. ACK slot and ACK delimiter are each one bit long. |

# Controlled Area Network (CAN)

| Field | Bits | Description |
|---|---|---|
| EOF | 7 | End Of Frame is a 7-bit field that denotes the end of every CAN frame (message). |
| IFS | 3+ | Inter Frame Space (IFS) is the time that the controller needs to move a frame (message) into position in the buffer area. Note that IFS contains a minimum of three consecutive recessive (1) bits. After three recessive bits have passed, when a dominant bit is detected, it becomes the SOF bit of the next frame. |

# Controlled Area Network (CAN)

- If multiple nodes transmit at the same time, they start a simultaneous arbitration
- The node with the lowest message ID number gets priority
- Dominant bits overwrite recessive bits on the CAN bus
- The message identifier can be **11-bit** (Standard CAN, 2048 different message identifiers)
- Or **29 bit** in length (Extended CAN, 537 million different message identifiers).

# Controlled Area Network (CAN)

**Advantages:**

- It support multi master and multicast features.
- The CAN bus has maximum length of 40 meters.
- The CAN provides the ability to work in different electrical environment.
- The CAN reduces wiring since it is a distributed control, and this ensures enhancing of the system performance.
- It has standard bus in distributed network.

**Limitations:**

- It has high software expenditure.
- Undesirable interaction more probable.

# Universal Serial Bus (USB)

- USB cables and connectors create an interface as a way for computers and peripheral devices to connect with each other and transfer data
- There are numerous USB connector types that have been used to interface the USB 1.1/2.0 and USB 3.0 protocols
- Most commonly used connectors include USB Standard-A, USB Standard-B, USB Mini-B, USB Micro-B, and USB Type-C

# Universal Serial Bus (USB)

- **USB Type-A**: Is the most widely used connector type. It is primarily used on host controllers in computers and hubs and is more commonly used as a downstream connection
- **USB Type-B**: is mainly used for connecting USB peripheral devices including printers and compact devices like mobile phones. It is commonly used as an upstream connection.
- **USB Type-C:** is an advanced connector type that uses a reversible design, and is intended to replace other connectors in the hopes of there being one cable to function with a variety of different devices.

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **Full Speed USB (USB 1.1)**
- The first USB specification, USB 1.0, was introduced in 1996
- Supported a Low-Speed transfer rate at 1.5 Mbps
- Specification was later revised in 1998 to USB 1.1, also known as Full-Speed USB
- This updated specification supports a bandwidth of 12 Mbps and power levels up to 2.5W
- USB connectors supporting this spec include USB Type-A and USB Type-B

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **High-Speed USB (USB 2.0)**
- In 2001, the USB 2.0 specification was introduced
- USB 2.0, also known as High-Speed USB, supports a transfer rate of 480 Mbps and is backwards compatible with USB 1.1
- USB 2.0 also uses the same USB Type-A and USB Type-B cables and connectors, as well as the same software interfaces as USB 1.1
- But substantially increases the support for higher bandwidth peripheral devices, such as video camera devices

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **SuperSpeed USB (USB 3.x)**
- The USB 3.0 specification, also known as SuperSpeed USB, was first released in 2008
- USB 3.0 supports transfer rates up to 5 Gbps and power levels up to 4.5W
- Ten times faster and twice as powerful as USB 2.0
- SuperSpeed USB is also backwards compatible with its predecessors and is supported on cable and connector types including USB Type-A and USB Type-B.

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **SuperSpeed USB (USB 3.x)**
- In 2013, SuperSpeed USB 3.1 was introduced to reflect its support for transfer rates up to 10 Gbps by using a dual-lane operation within a USB Type-C connector
- In 2017, USB 3.2 was released, further increasing the signaling rate
- This revision supports USB transfer rates up to 20 Gbps
- **USB4**

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **USB4**
- The USB4 specification was released in 2019
- It offers users some of the most robust features and capabilities, including the ability to transfer data up to 40 Gbps
- USB4 permits the highest USB bandwidth available with multiple data and display protocols to efficiently share the maximum aggregate bandwidth over the bus
- USB4 also has backwards compatibility with USB 3.2, USB 2.0

# Universal Serial Bus (USB)

- The USB systems consist of a host, which is typically a personal computer (PC) and multiple peripheral devices connected to them through a tiered-star topology
- This topology may also include hubs that allow additional to the connection points to the USB protocol system
- The host itself contains two components, the host controller and the root hub

# Universal Serial Bus (USB)

- The host controller is a hardware chipset with the software driver layer that is responsible for these tasks:
  - Detect attachment and removal of USB devices
  - Manage data flow between host and devices
  - Provide and manage power to attached devices
  - Monitor activity on the bus

- In USB protocol at least one host controller is present in a host and it is possible to have more than one host controller
- Each controller allows the connection of up to 127 devices with the use of external USB hubs

# Universal Serial Bus (USB)

- The root of the hub is an internal hub that connects to the host controller(s) and acts as the first interface layer to the USB in a system
- Currently, on your PC, there are multiple USB ports. These ports are part of the root hub in your PC

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- When a USB device is the first connected to a host using USB protocol, the USB enumeration process is initiated
- Enumeration is the process of exchanging information between the device and the host that includes learning about the device
- Additionally, enumeration includes assigning an address to the device, reading descriptors (which are data structures that provide information about the device), and assigning and loading a device driver

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- When a USB device is the first connected to a host using USB protocol, the USB enumeration process is initiated
- Enumeration is the process of exchanging information between the device and the host that includes learning about the device
- Additionally, enumeration includes assigning an address to the device, reading descriptors (which are data structures that provide information about the device), and assigning and loading a device driver

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- When this process is complete, the device is ready for the transfer data to the host
- Two files are affiliated with the enumeration and the loading of a driver
- .INF and .SYS
- They exist on the host side

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- **.INF:**
- A text file that contains all the information necessary to install a device
- Such as driver names and locations, Windows registry information, and driver version information
- ***.SYS:*** The driver needed to communicate effectively with the USB device

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

1. The Device Connected to Host
2. The Host resets device then requests Device Descriptor
3. The device responds to request and Host sets a new address
4. The Host requests Device Descriptor using new address and the Device responds to it
5. Then the host locates and reads INF file
6. The INF specifies the Device Driver
7. The Driver loaded on host
8. Then the Host selects the configuration

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# Universal Serial Bus (USB)

- USB cables and connectors create an interface as a way for computers and peripheral devices to connect with each other and transfer data
- There are numerous USB connector types that have been used to interface the USB 1.1/2.0 and USB 3.0 protocols
- Most commonly used connectors include USB Standard-A, USB Standard-B, USB Mini-B, USB Micro-B, and USB Type-C

# Universal Serial Bus (USB)

- **USB Type-A**: Is the most widely used connector type. It is primarily used on host controllers in computers and hubs and is more commonly used as a downstream connection
- **USB Type-B**: is mainly used for connecting USB peripheral devices including printers and compact devices like mobile phones. It is commonly used as an upstream connection.
- **USB Type-C:** is an advanced connector type that uses a reversible design, and is intended to replace other connectors in the hopes of there being one cable to function with a variety of different devices.

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **Full Speed USB (USB 1.1)**
- The first USB specification, USB 1.0, was introduced in 1996
- Supported a Low-Speed transfer rate at 1.5 Mbps
- Specification was later revised in 1998 to USB 1.1, also known as Full-Speed USB
- This updated specification supports a bandwidth of 12 Mbps and power levels up to 2.5W
- USB connectors supporting this spec include USB Type-A and USB Type-B

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **High-Speed USB (USB 2.0)**
- In 2001, the USB 2.0 specification was introduced
- USB 2.0, also known as High-Speed USB, supports a transfer rate of 480 Mbps and is backwards compatible with USB 1.1
- USB 2.0 also uses the same USB Type-A and USB Type-B cables and connectors, as well as the same software interfaces as USB 1.1
- But substantially increases the support for higher bandwidth peripheral devices, such as video camera devices

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **SuperSpeed USB (USB 3.x)**
- The USB 3.0 specification, also known as SuperSpeed USB, was first released in 2008
- USB 3.0 supports transfer rates up to 5 Gbps and power levels up to 4.5W
- Ten times faster and twice as powerful as USB 2.0
- SuperSpeed USB is also backwards compatible with its predecessors and is supported on cable and connector types including USB Type-A and USB Type-B.

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **SuperSpeed USB (USB 3.x)**
- In 2013, SuperSpeed USB 3.1 was introduced to reflect its support for transfer rates up to 10 Gbps by using a dual-lane operation within a USB Type-C connector
- In 2017, USB 3.2 was released, further increasing the signaling rate
- This revision supports USB transfer rates up to 20 Gbps
- **USB4**

# Universal Serial Bus (USB)

- Over the years, there have been multiple USB revisions and specifications that have been introduced
- **USB4**
- The USB4 specification was released in 2019
- It offers users some of the most robust features and capabilities, including the ability to transfer data up to 40 Gbps
- USB4 permits the highest USB bandwidth available with multiple data and display protocols to efficiently share the maximum aggregate bandwidth over the bus
- USB4 also has backwards compatibility with USB 3.2, USB 2.0

# Universal Serial Bus (USB)

- The USB systems consist of a host, which is typically a personal computer (PC) and multiple peripheral devices connected to them through a tiered-star topology
- This topology may also include hubs that allow additional to the connection points to the USB protocol system
- The host itself contains two components, the host controller and the root hub

# Universal Serial Bus (USB)

- The host controller is a hardware chipset with the software driver layer that is responsible for these tasks:
  - Detect attachment and removal of USB devices
  - Manage data flow between host and devices
  - Provide and manage power to attached devices
  - Monitor activity on the bus

- In USB protocol at least one host controller is present in a host and it is possible to have more than one host controller
- Each controller allows the connection of up to 127 devices with the use of external USB hubs

# Universal Serial Bus (USB)

- The root of the hub is an internal hub that connects to the host controller(s) and acts as the first interface layer to the USB in a system
- Currently, on your PC, there are multiple USB ports. These ports are part of the root hub in your PC

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- When a USB device is the first connected to a host using USB protocol, the USB enumeration process is initiated
- Enumeration is the process of exchanging information between the device and the host that includes learning about the device
- Additionally, enumeration includes assigning an address to the device, reading descriptors (which are data structures that provide information about the device), and assigning and loading a device driver

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- When this process is complete, the device is ready for the transfer data to the host
- Two files are affiliated with the enumeration and the loading of a driver
- .INF and .SYS
- They exist on the host side

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

- **.INF:**
- A text file that contains all the information necessary to install a device
- Such as driver names and locations, Windows registry information, and driver version information
- ***.SYS:*** The driver needed to communicate effectively with the USB device

# Universal Serial Bus (USB)

**How does a USB device connect with Host Controller?**

1. The Device Connected to Host
2. The Host resets device then requests Device Descriptor
3. The device responds to request and Host sets a new address
4. The Host requests Device Descriptor using new address and the Device responds to it
5. Then the host locates and reads INF file
6. The INF specifies the Device Driver
7. The Driver loaded on host
8. Then the Host selects the configuration

# Universal Serial Bus (USB)

- A USB interface has several layers of protocol
- Most of the time the lower-level layers are single handled by host controller IC
- The end designer needs to work on higher level layers
- Every communication protocol involves the exchange of packets
- These packets generally contain information related to:
  - Controlling the data exchange
  - Data exchange in the form of actual payload
  - Error detection and correction through status check up

# Universal Serial Bus (USB)

- Data on the USBus is transmitted LSBit first
- USB Packet Field:
- **Sync**
  - All packets must start with a sync field
  - Field is 8 bits long at low and full speed or 32 bits long for high speed
  - Used to synchronize the clock of the receiver with that of the transmitter
- **PID**

# Universal Serial Bus (USB)

- **PID**
  - PID stands for Packet ID
  - This field is used to identify the type of packet that is being sent
  - There are 4 bits to the PID
  - To insure it is received correctly, the 4 bits are complemented and repeated
  - Making an 8 bit PID in total

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $PID_0$ | $PID_1$ | $PID_2$ | $PID_3$ | $nPID_0$ | $nPID_1$ | $nPID_2$ | $nPID_3$ |

| Group | PID Value | Packet Identifier |
|---|---|---|
| Token | 0001 | OUT Token |
| | 1001 | IN Token |
| | 0101 | SOF Token |
| | 1101 | SETUP Token |
| Data | 0011 | DATA0 |
| | 1011 | DATA1 |
| | 0111 | DATA2 |
| | 1111 | MDATA |
| Handshake | 0010 | ACK Handshake |
| | 1010 | NAK Handshake |
| | 1110 | STALL Handshake |
| | 0110 | NYET (No Response Yet) |
| Special | 1100 | PREamble |
| | 1100 | ERR |
| | 1000 | Split |
| | 0100 | Ping |

# Universal Serial Bus (USB)

- **ADDR**
  - The address field specifies which device the packet is designated for
  - Being 7 bits in length allows for 127 devices to be supported
- **ENDP**
  - The endpoint field is made up of 4 bits
  - Allowing 16 possible endpoints
- **CRC**
  - Data packets have 16 bit CRC
- **EOP**
  - End of packet
  - Single Ended Zero (SE0) for approximately 2 bit times followed by a J for 1 bit time

# Universal Serial Bus (USB)

## USB Packets

- **Token packets**
- These packets are only sent by host
- The packet structure contains a PID byte
- 11 bits of address
- 5-bit CRC

| Sync | PID | Device Address | End Point | CRC 5-Bit | EOP |
|---|---|---|---|---|---|

# Universal Serial Bus (USB)

## USB Packets

- **Data packets**
- Two types of data packets are there, Data0 and Data1
- Each capable of transmitting up to 1024 bytes of data

| Sync | PID | Data | CRC 16-Bit | EOP |
|---|---|---|---|---|

- Maximum data payload size for low-speed devices is 8 bytes
- Maximum data payload size for high-speed devices is 1024 bytes
- Data must be sent in multiples of bytes

# Universal Serial Bus (USB)

## USB Packets

- **Handshake Packets**
- Three type of handshake packets which consist simply of the PID

| Sync | PID | EOP |
|---|---|---|

- **ACK** - Acknowledgment that the packet has been successfully received
- **NAK** - Reports that the device temporary cannot send or received data
- **STALL** - The device finds its in a state that it requires intervention from the host

# Universal Serial Bus (USB)

## USB Packets

- **Start of Frame Packets**
- SOF packet consisting of an 11-bit frame number is sent by the host
- every 1ms ± 500ns on a full speed bus (12 Mbit/s)
- every 125 μs ± 0.0625 μs on a high speed bus (480 Mbit/s)

| | | | | | |
|---|---|---|---|---|---|
| Sync | PID | Frame number | End Point | CRC | EOP |

# Wireless Communication Protocols

- There is a wide range of wireless technologies that enable the connectivity, each with their own strengths and weaknesses
- Choosing the right technology will ensure the embedded application
  - runs smoothly
  - at the lowest cost
  - with the best power efficiency
- Some wireless technologies existed pre-IoT
- Whereas some have been designed specifically for it, but all have their own advantages and disadvantages

# Wireless Communication Protocols

**Parameters**

- Range
- Power
- Data rate
- Module cost
- Compatibility
- One/two-way data transfer
- Global coverage
- Ecosystem requirements

# Wireless Communication Protocols

- Short range wireless technologies
  - NFC
  - Bluetooth
  - BLE
  - RFID
  - ZigBee
  - Wi-Fi
- Longer range wireless technologies
  - LPWAN

# Wireless Communication Protocols

| Attribute | Bluetooth® Low Energy Technology | Wi-Fi | Z-Wave | IEEE 802.15.4 (Zigbee, Thread) | LTE-M | NB-IoT | Sigfox | LoRaWAN |
|---|---|---|---|---|---|---|---|---|
| Range | 10 m – 1.5 km | 15 m – 100 m | 30 m - 50 m | 30 m – 100 m | 1 km – 10 km | 1 km – 10 km | 3 km – 50 km | 2 km – 20 km |
| Throughput | 125 kbps – 2 Mbps | 54 Mbps – 1.3 Gbps | 10 kbps – 100 kbps | 20 kbps – 250 kbps | Up to 1 Mbps | Up to 200 kbps | Up to 100 bps | 10 kbps – 50 kbps |
| Power Consumption | Low | Medium | Low | Low | Medium | Low | Low | Low |
| Ongoing Cost | One-time | One-time | One-time | One-time | Recurring | Recurring | Recurring | One-time |
| Module Cost | Under $5 | Under $10 | Under $10 | $8-$15 | $8-$20 | $8-$20 | Under $5 | $8-$15 |
| Topology | P2P, Star, Mesh, Broadcast | Star, Mesh | Mesh | Mesh | Star | Star | Star | Star |

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# Wireless Communication Protocols

- There is a wide range of wireless technologies that enable the connectivity, each with their own strengths and weaknesses
- Choosing the right technology will ensure the embedded application
  - runs smoothly
  - at the lowest cost
  - with the best power efficiency
- Some wireless technologies existed pre-IoT
- Whereas some have been designed specifically for it, but all have their own advantages and disadvantages

# Wireless Communication Protocols

**Parameters**

- Range
- Power
- Data rate
- Module cost
- Compatibility
- One/two-way data transfer
- Global coverage
- Ecosystem requirements

# Wireless Communication Protocols

- Short range wireless technologies
  - NFC
  - Bluetooth
  - BLE
  - RFID
  - ZigBee
  - Wi-Fi
- Longer range wireless technologies
  - LPWAN

# Wireless Communication Protocols

| Attribute | Bluetooth® Low Energy Technology | Wi-Fi | Z-Wave | IEEE 802.15.4 (Zigbee, Thread) | LTE-M | NB-IoT | Sigfox | LoRaWAN |
|---|---|---|---|---|---|---|---|---|
| Range | 10 m – 1.5 km | 15 m – 100 m | 30 m - 50 m | 30 m – 100 m | 1 km – 10 km | 1 km – 10 km | 3 km – 50 km | 2 km – 20 km |
| Throughput | 125 kbps – 2 Mbps | 54 Mbps – 1.3 Gbps | 10 kbps – 100 kbps | 20 kbps – 250 kbps | Up to 1 Mbps | Up to 200 kbps | Up to 100 bps | 10 kbps – 50 kbps |
| Power Consumption | Low | Medium | Low | Low | Medium | Low | Low | Low |
| Ongoing Cost | One-time | One-time | One-time | One-time | Recurring | Recurring | Recurring | One-time |
| Module Cost | Under $5 | Under $10 | Under $10 | $8-$15 | $8-$20 | $8-$20 | Under $5 | $8-$15 |
| Topology | P2P, Star, Mesh, Broadcast | Star, Mesh | Mesh | Mesh | Star | Star | Star | Star |

# Bluetooth

- Short range wireless communications technology
- Bluetooth technology is based on **Ad-hoc technology**
- Local area network with a very limited coverage

- Part of our lives due to its immense applications
- With Bluetooth one can transfer data, audio, graphic images and video from one device to the other provided they are Bluetooth compliant
- IEEE 802.15.1 standard

# Bluetooth

- Bluetooth network consists of multiple Bluetooth users
- There are two types of network topologies in Bluetooth viz. Piconet and scatternet
- Piconet is formed by one master and one slave as well as one master and multiple slaves
- There will be maximum 7 active slaves in the piconet
- Hence there will be about 8 maximum devices communicating in a small network referred as piconet

# Bluetooth

- Combinations of multiple piconets is known as scatternet
- A device can participate in multiple piconets
- It will timeshare and need to be synchronized with the master of current piconet

# Bluetooth

| Specification | feature supported |
|---|---|
| RF Frequency | 2.4GHz |
| Transmit power | 100 mW(Max.) 1mW(Min.) |
| Data rate | About 1Mbps |
| Distance | 100 meter(max.), 1 meter(min.) |
| RF Bandwidth | 220 KHz to about 1MHz |
| Modulation type | Gaussian FSK (GFSK) |
| Number of RF carriers | 79(max.), 23(min.) |
| Topology | upto 7 links in a star configuration |

# Bluetooth

| | Bluetooth Classic | Bluetooth v4.x | Bluetooth v5.0 | Bluetooth v5.2 |
|---|---|---|---|---|
| **Data rates** | 1 Mbps | 1 Mbps | 2 Mbps | 2 Mbps |
| **Maximum Range** | 10 m | 30 m | 200 m | 200 m |
| **Power Consumption** | Very High | High | Low | Very Low |
| **Throughput** | 700 kbps | 300 kbps | 1400 kbps | 1400 kbps |
| **Message Capacity** | 31 bytes | 31 bytes | 255 bytes | 255 bytes |

# Bluetooth – Protocol Stack

- Bluetooth standard has a protocol stack of its own, just like the OSI and TCP/IP model for networks
- Unlike the OSI model, Bluetooth doesn't require its devices to use all the layers of the protocol stack

# Bluetooth – Protocol Stack

- Protocols in the stack can be grouped based on their role:
- Core Protocols
- Cable Replacement Protocols
- Adapted Protocols
- AT Commands

# Bluetooth – Protocol Stack

## Core Protocols

- Core Protocols are those that are ad hoc protocols essential for the functioning of a large number of Bluetooth's features
- **Bluetooth Radio**
  - Lays down the physical structure and specifications for transmission of radio waves
  - Similar to the physical layer protocols in the OSI and TCP/IP models
  - Defines the air interface, frequency bands, frequency hopping specifications and modulation techniques

# Bluetooth – Protocol Stack

## Core Protocols

- Core Protocols are those that are ad hoc protocols essential for the functioning of a large number of Bluetooth's features
- **Baseband**
  - The baseband layer is responsible for searching other devices
  - It defines the addressing scheme, packet frame format, timing, and power control algorithms
  - Also responsible for assigning master/ slave roles

# Bluetooth – Protocol Stack

## Core Protocols

- Core Protocols are those that are ad hoc protocols essential for the functioning of a large number of Bluetooth's features
- **Link Management Protocol (LMP)**
  - Establishes and manages links that are already established
  - Also responsible for authentication and encryption of the link
- **Logical Link Control and Adaptation Protocol (L2CAP)**
  - The heart of the protocol stack and allows the communication between the upper and lower layers
  - It modifies the packages from the upper layer as required by the lower layer and vice versa

# Bluetooth – Protocol Stack

## Core Protocols

- Core Protocols are those that are ad hoc protocols essential for the functioning of a large number of Bluetooth's features
- **Link Management Protocol (LMP)**
  - Establishes and manages links that are already established
  - Also responsible for authentication and encryption of the link
- **Logical Link Control and Adaptation Protocol (L2CAP)**
  - The heart of the protocol stack and allows the communication between the upper and lower layers
  - It modifies the packages from the upper layer as required by the lower layer and vice versa

# Bluetooth – Protocol Stack

## Cable Replacement Protocols

- **Radio Frequency Communications (RFComm) Protocol**
- It provides a virtual serial port for applications that require it
- Eliminated the need for cables

# Bluetooth – Protocol Stack

## Adopted Protocols

- These protocols are adopted from other models, such as the TCP/IP model and the OSI model
- **Point-to-Point Protocol (PPP)**
  - Facilitates communication between two directly connected computers (Bluetooth devices in this case)
- **Wireless Application Protocol (WAP)**
  - Standardizes the way how wireless Bluetooth devices can become a part of a network

# Bluetooth – Protocol Stack

## AT Commands

- These can be used in sending instructions to a Bluetooth module and to change the device's settings like baud rate and name
- You can usually find information on how to configure setting using AT commands with your Bluetooth device's datasheet

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# Wireless Communication Protocols

- There is a wide range of wireless technologies that enable the connectivity, each with their own strengths and weaknesses
- Choosing the right technology will ensure the embedded application
  - runs smoothly
  - at the lowest cost
  - with the best power efficiency
- Some wireless technologies existed pre-IoT
- Whereas some have been designed specifically for it, but all have their own advantages and disadvantages

# Wireless Communication Protocols

**Parameters**

- Range
- Power
- Data rate
- Module cost
- Compatibility
- One/two-way data transfer
- Global coverage
- Ecosystem requirements

# Wireless Communication Protocols

- Short range wireless technologies
  - NFC
  - Bluetooth
  - BLE
  - RFID
  - ZigBee
  - Wi-Fi
- Longer range wireless technologies
  - LPWAN

# Wi-Fi – 802.11

- Family of wireless protocols standardized by Institute of Electrical and Electronics Engineers (IEEE)
- Specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) computer communication
- IEEE 802.11 is used in most home and office networks to allow laptops, printers, smartphones, and other devices to communicate with each other and access the Internet without connecting wires
- The base version of the standard was released in 1997 and has had subsequent amendments

# Wi-Fi – 802.11

- Family of wireless protocols standardized by Institute of Electrical and Electronics Engineers (IEEE)
- Specifies the set of media access control (MAC) and physical layer (PHY) protocols for implementing wireless local area network (WLAN) computer communication
- IEEE 802.11 is used in most home and office networks to allow laptops, printers, smartphones, and other devices to communicate with each other and access the Internet without connecting wires
- The base version of the standard was released in 1997 and has had subsequent amendments

# Wi-Fi – 802.11

- Each Wi-Fi network standard has two parameters :
- **Speed**
  - This is the data transfer rate of the network measured in Mbps
- **Frequency**
  - On what radio frequency, the network is carried on
  - Two bands of frequency for the Wi-Fi are 2.4 GHz and 5 GHz

# Wi-Fi – 802.11

- Wi-Fi routers that come with 2.4 GHz or 5 GHz are called the single-band routers
- A lot of new routers support both 2.4 GHz and 5 GHz frequency they are called dual-band routers
- The 2.4 GHz is a common Wi-Fi band, but it is also used by other appliances like Bluetooth devices, wireless phones, cameras, etc
- Because of the signal used by so many devices, the signal becomes overcrowded and speed becomes slow
- So 5 GHz comes into the picture
- The 2.4 GHz transmits data at a slower speed than 5 GHz but does have a longer range than 5 GHz

# Wi-Fi – 802.11

## The 802.11 architecture

- 802.11 networks can be used in two modes
  - Infrastructure mode
  - Ad-hoc mode

# Wi-Fi – 802.11

## The 802.11 architecture

- 802.11 networks can be used in two modes
  - Infrastructure mode
  - Ad-hoc mode

# Wi-Fi – 802.11

## The 802.11 protocol stack

- The 802.11 physical layer corresponds to the OSI physical layer, but the data link layer is split into multiple sublayers

# Wi-Fi – 802.11

## The 802.11 protocol stack

- The MAC sublayer determines which channel gets to transmit next
- LLC (Logical Link Layer), hides the differences between the varying 802.11 versions for the network

# Wi-Fi – 802.11

## 802.11 physical layer

- All 802.11 techniques use short-range radios to transmit signals in either 2.4-GHz or 5-GHz ISM frequency bands
- All 802.11 transmission methods define multiple rates
- Different rates can be used depending on the current conditions
- If the signal is weak, a low rate is used.
- If the signal is clear, the highest rate is used
- The process of adjustment is called **rate adaption**

# Wi-Fi – 802.11

## 802.11 physical layer

- Role of Bandwidth
- Bandwidth in terms of radio wave communication is the upper and lower limit of the frequency used for transmitting or receiving information. It's also measured in hertz
- wireless network operate on 2.4 Ghz with 20Mhz bandwidth
- It means the device is transmitting and receiving radio signal between 2.390 Ghz (lower limit) and 2.410 Ghz (upper limit)
- Bigger bandwidth usually means higher transfer rate, because more data can be transmitted on one unit of time

# Wi-Fi – 802.11

## 802.11 physical layer

| Standard | Frequency Band | Bandwidth | Modulation Scheme | Channel Arch. | Maximum Data Rate | Range |
|---|---|---|---|---|---|---|
| 802.11 | 2.4 GHz | 20 MHz | BPSK to 256-QAM | DSSS, FHSS | 2 Mbps | 20 m |
| b | 2.4 GHz | 21 MHz | BPSK to 256-QAM | CCK, DSSS | 11 Mbps | 35 m |
| a | 5 GHz | 22 MHz | BPSK to 256-QAM | OFDM | 54 Mbps | 35 m |
| g | 2.4 GHz | 23 MHz | BPSK to 256-QAM | DSSS, OFDM | 54 Mbps | 70 m |
| n | 2.4 GHz, 5 GHz | 24 MHz and 40 MHz | BPSK to 256-QAM | OFDM | 600 Mbps | 70 m |
| ah | 900 MHz | 1, 2, 4, 8, and 16 MHz | BPSK to 256-QAM | SC, OFDM | 40 Mbps | 1 km |

# Wi-Fi – 802.11

## 802.11 MAC Layer

- 802.11 MAC sublayer is different from the Ethernet MAC sublayer for two reasons
  - Radios are almost always half duplex
  - Transmission ranges of different stations might be different
- 802.11 uses the **CSMA/CA (CSMA with Collision Avoidance) protocol**
- CSMA/CA is similar to ethernet CSMA/CD
- It uses channel sensing and exponential backoff after collisions, but instead of entering backoff once a collision has been detected, CSMA/CA uses backoff immediately

# Wi-Fi – 802.11

## 802.11 Frame Structure

- There are three different classes of frames used in the air:
  - Data
  - Control
  - Management

# Wi-Fi – 802.11

## 802.11 Frame Structure

- The first part of frame is the *Frame Control* field, made up of 11 subfields
- **Protocol Version**: set to 00 for current versions of 802.11
- **Type**: can be one of data, control, or management, and the Subtype (e.g RTS or CTS). These are set to 10 and 0000 in binary for a normal data field

# Wi-Fi – 802.11

## 802.11 Frame Structure

- **To DS and From DS**: these bits indicate whether frames are coming or going from a network connected to the AP (the network is called the distribution system)
- **More Fragment***s*: this bit means that more fragments will follow
- **Retry**: this bit "marks a retransmission of a frame sent earlier"

# Wi-Fi – 802.11

## 802.11 Frame Structure

- **Power Management**: this bit indicates that the sender is going into power-save mode
- **More Data**: this bit indicates that the sender has additional frames for the receiver
- **Protected Frame**: this bit indicates that the frame body has been encrypted for security
- **Order**: this “bit tells the receiver that the higher layer expects the sequence of frames to arrive strictly in order”

# Wi-Fi – 802.11

## 802.11 Frame Structure

- **Duration field**: This describes how long the frame and its acknowledgements will occupy the channel (measured in microseconds)
- It's included in all frames, including control frames

# Wi-Fi – 802.11

## 802.11 Frame Structure

- The addresses to and from an AP follow the standard IEEE 802 format.
- The ***Address 1*** is the receiver, ***Address 2*** is the transmitter
- ***Address 3*** is the address of the endpoint that originally sent the frame via the AP

# Wi-Fi – 802.11

## 802.11 Frame Structure

- The *Sequence* 16-bit field numbers frames so that duplicates can be detected. T
- he first 4 bits identify the fragment
- The last 12 contain a number that's incremented on each transmission

# Wi-Fi – 802.11

**Advantages:**

- Convenience
- Mobility
- Deployment
- Expandability

**Limitations:**

- Security
- Range
- Reliability
- Speed

# Zigbee – 802.15.4

- Specification created for wireless networking
- It includes hardware and software standard design for WSN (Wireless sensor network)
  - requiring high reliability
  - low cost
  - low power
  - Scalability
  - low data rate
- ZigBee-style self-organizing ad-hoc digital radio networks were conceived in the 1990s
- IEEE 802.15.4-2003/ZigBee specification was ratified in 2004

# Zigbee – 802.15.4

- ZigBee provides ultra low consumption and efficiency

| Standard | ZigBee/IEEE 802.15.4 | Bluetooth | UWB | IEEE 802.11 b/g |
|---|---|---|---|---|
| Working frequency | 868/915 MHz, 2.4GHz | 2.4 GHz | 3.1 - 10.6 GHz | 2.4 GHz |
| Range (m) | 30 – 75+ | 10 – 30 | ~10 | 30 – 100 + |
| Data rate | 20/40/250 kbps | 1 Mbps | 100+ Mbps | 2 – 54 Mbps |
| Devices | 255 – 65k | 8 | | 50 – 200 |
| Power consumption | ~1 mW | ~40 – 100 mW | ~80 – 300 mW | ~160 mW – 600W |
| Cost ($US) | ~2 – 5 | ~4 – 5 | ~5 – 10 | ~20 – 50 |

# Zigbee – 802.15.4

- Protocol Stack

# Zigbee – 802.15.4

It defines three types of devices:

- ZigBee Coordinator
  - responsible for initializing, maintaining and controlling the network. There is one and only one per network
- ZigBee Router
  - connected to the coordinator or other routers. It can have zero or more children nodes. Participate in multi hop routing
- ZigBee End Devices
  - does not participate in routing

# Zigbee – 802.15.4

It defines three types of devices:

# Zigbee – 802.15.4

It defines three types of network topology: Star, tree and mesh

| | Pros | Cons |
|---|---|---|
| Star | 1. Easy to synchronize<br>2. Support low power operation<br>3. Low latency | 1. Small scale |
| Tree | 1. Low routing cost<br>2. Can form superframes to support sleep mode<br>3. Allow multihop communication | 1. Route reconstruction is costly<br>2. Latency may be quite long |
| Mesh | 1. Robust multihop communication<br>2. Network is more flexible<br>3. Lower latency | 1. Cannot form superframes (and thus cannot support sleep mode)<br>2. Route discovery is costly<br>3. Needs storage for routing table |

# Zigbee – 802.15.4

## Advantages of IoT with ZigBee

- Power of the mesh, devices tend to connect with every near device, that makes every node of the network reachable from every other node and expanding the network geographically, also providing self healing
- Low consumption of energy and working in the network even without the necessity of a battery (Green Power)
- And high scalability, ZigBee networks can to thousand of devices and they will communicate with each other using the best available path

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedding Network Technologies

# Zigbee – 802.15.4

- Specification created for wireless networking
- It includes hardware and software standard design for WSN (Wireless sensor network)
  - requiring high reliability
  - low cost
  - low power
  - Scalability
  - low data rate
- ZigBee-style self-organizing ad-hoc digital radio networks were conceived in the 1990s
- IEEE 802.15.4-2003/ZigBee specification was ratified in 2004

# Zigbee – 802.15.4

- ZigBee provides ultra low consumption and efficiency

| Standard | ZigBee/IEEE 802.15.4 | Bluetooth | UWB | IEEE 802.11 b/g |
|---|---|---|---|---|
| Working frequency | 868/915 MHz, 2.4GHz | 2.4 GHz | 3.1 - 10.6 GHz | 2.4 GHz |
| Range (m) | 30 – 75+ | 10 – 30 | ~10 | 30 – 100 + |
| Data rate | 20/40/250 kbps | 1 Mbps | 100+ Mbps | 2 – 54 Mbps |
| Devices | 255 – 65k | 8 | | 50 – 200 |
| Power consumption | ~1 mW | ~40 – 100 mW | ~80 – 300 mW | ~160 mW – 600W |
| Cost ($US) | ~2 – 5 | ~4 – 5 | ~5 – 10 | ~20 – 50 |

# Zigbee – 802.15.4

- Protocol Stack

# Zigbee – 802.15.4

It defines three types of devices:

- ZigBee Coordinator
  - responsible for initializing, maintaining and controlling the network. There is one and only one per network
- ZigBee Router
  - connected to the coordinator or other routers. It can have zero or more children nodes. Participate in multi hop routing
- ZigBee End Devices
  - does not participate in routing

# Zigbee – 802.15.4

It defines three types of devices:

# Zigbee – 802.15.4

It defines three types of network topology: Star, tree and mesh

| | Pros | Cons |
|---|---|---|
| Star | 1. Easy to synchronize<br>2. Support low power operation<br>3. Low latency | 1. Small scale |
| Tree | 1. Low routing cost<br>2. Can form superframes to support sleep mode<br>3. Allow multihop communication | 1. Route reconstruction is costly<br>2. Latency may be quite long |
| Mesh | 1. Robust multihop communication<br>2. Network is more flexible<br>3. Lower latency | 1. Cannot form superframes (and thus cannot support sleep mode)<br>2. Route discovery is costly<br>3. Needs storage for routing table |

# Zigbee – 802.15.4

## Advantages of IoT with ZigBee

- Power of the mesh, devices tend to connect with every near device, that makes every node of the network reachable from every other node and expanding the network geographically, also providing self healing
- Low consumption of energy and working in the network even without the necessity of a battery (Green Power)
- And high scalability, ZigBee networks can to thousand of devices and they will communicate with each other using the best available path

# NFC – Near Field Communication

- For devices in **very close range,** normally in physical contact
- One of the two communicating devices may be **passive,** meaning that it has no energy source (no batteries, no solar cell)
- Ancient relative of NFC is the anti-theft tag or security tag found in many stores

# NFC – Near Field Communication

- Ancient relative of NFC is the anti-theft tag or security tag found in many stores
- The door sensor generates a magnetic field. The tag has no batteries but can absorb field energy
- This absorption is sensed and it triggers the alarm
- On top of this principle, there is the RFID technology
- A RFID tag not only absorbs energy but also sends some information back

# NFC – Near Field Communication

- NFC goes one step further; the passive NFC device may be "intelligent“
- Meaning that it possesses a tiny computer and can do some processing
- The NFC potential is enormous and includes things like:
  - Contactless credit cards. Making payments without taking the card out of the wallet
  - Public transport fare systems, tickets in general
  - E-money stored in your cell phone, use your phone as the credit card or pre-paid card
  - Communication with medical devices

# NFC – Near Field Communication

- NFC allows communication between two "active" devices, e.g. two cell phones
- This feature makes it easy to exchange contact cards, photos etc.
- The security and convenience are very nice: the interested parties just need to touch their gadgets
- This still does not work as well as it should, because the standards are still emerging and there are incompatibilities

# NFC – Near Field Communication

- For the user, NFC is perfect. For the developer, NFC can be more complicated than initially expected
- This has "historical" roots: the NFC specifications inherited preexisting and incompatible flavors of NFC (from Philips, from Sony, and so on)
- The result is a stack with many redundant features, and an annoying multiplicity of tag types

# NFC – Near Field Communication

- There are many tag "types": Type A, Type B, Type F…
- On top of these types, we have the "technology types": Type 1, 2 e and 4
- "Type 1" adopts a subset of the features implemented by "Type A“
- Each type has a set of capabilities and limitations

# NFC – Near Field Communication

- Commercial standards like *MiFARE* implement additional features, like tag-embeddded cryptography, on top of a basic type
- The NFC protocol stack makes the TCP/IP stack look dead simple in direct comparison

# NFC – Near Field Communication

- The NFC protocol stack makes the TCP/IP stack look dead simple in direct comparison
- Each tag type offers a different link protocol, and each technology type extends the command list of the base protocol
- NFC-A offers a command set, and Type 2 extends that set

# NFC – Near Field Communication

- NFC-A offers a command set, and Type 2 extends that set
- Because of that, there are many link protocols (the square hooks)
- The transport/application protocols can hook to any link protocol

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedded Real –Time Operating Systems

# Fundamentals of Real-Time Systems

# Definition: System

- System is a mapping of a set of inputs into a set of outputs.

1. A system is an assembly of components connected together in an organized way
2. A system is fundamentally altered if a component joins or leaves it
3. It has a purpose
4. It has a degree of permanence
5. It has been defined as being of particular interest

# Example: A Real-Time Control System

- Inputs are *excitations* and outputs are corresponding *responses*
- Inputs and outputs may be digital or analog
- Inputs are associated with sensors, cameras, etc.
- Outputs with actuators, displays, etc.

# Classic Representation of RTS

- A sequence of jobs to be scheduled and performance to be predicted
- Ignores the usual fact that the input sources and hardware under control may be highly complex

# Definition: Response Time

- The time between the presentation of a set of inputs to a system and the realization of the required behavior, including the availability of all associated outputs, is called the response time of the system

- How fast and punctual does it need to be?
  - Depends on the specific real-time system
- But what is a real-time system?

# Definition: Real-Time System

- A real-time system is a computer system that must satisfy bounded response-time constraints or risk severe consequences, including failure

- A real-time system is one whose logical correctness is based on both the correctness of the outputs and their timeliness

# Definition: Failed System

- A failed system is a system that **cannot** satisfy one or more of the requirements stipulated in the system requirements specification

- Hence, rigorous specification of the system operating criteria, including timing constraints, is necessary

# Definition: Embedded System

- An embedded system is a system containing one or more computers (or processors) having a central role in the functionality of the system, but the system is not explicitly called a computer

- A real-time system may be embedded or non-embedded
- But it is always reactive
  - Task scheduling is driven by ongoing interaction with the environment

# Degrees of “Real-Time”

- All practical systems are ultimately real-time systems
- Even a batch-oriented system—for example, grade processing at the end of a semester—is real-time
- Although the system may have response times of days, it must respond within a certain time
- Even a word-processing program should respond to commands within a reasonable amount of time
- Most of the literature refers to such systems as soft real-time systems

# Soft, Hard, and Firm “Real-Time”

Definition: Soft Real-Time System

A soft real-time system is one in which performance is degraded but not destroyed by failure to meet response-time constraints

Definition: Hard Real-Time System

A hard real-time system is one in which failure to meet even a single deadline may lead to complete or catastrophic system failure

Definition: Firm Real-Time System

A firm real-time system is one in which a few missed deadlines will not lead to total failure, but missing more than a few may lead to complete or catastrophic system failure

# Example: Real-Time Classification

| System | Real-Time Classification | Explanation |
|---|---|---|
| Avionics weapons delivery system in which pressing a button launches an air-to-air missile | Hard | Missing the deadline to launch the missile within a specified time after pressing the button may cause the target to be missed, which will result in a catastrophe |
| Navigation controller for an autonomous weed-killer robot | Firm | Missing a few navigation deadlines causes the robot to veer out from a planned path and damage some crops |
| Console hockey game | Soft | Missing even several deadlines will only degrade performance |

# Where Do Deadlines Come from?

- Deadlines are based on the underlying physical phenomena of the system under control

- Punctuality is another measure related to response times
  - Particularly important in periodically sampled systems with high sampling rates (e.g., in audio and video signal processing)

- Rule of thumb for cost-effective real-time systems:
  - Process everything as slowly as possible and repeat tasks as seldom as possible

# Definition: Real-Time Punctuality

- Real-time punctuality means that every response time has an average value, $t_{\mathrm{R}}$, with upper and lower bounds of $t_{\mathrm{R}} + \varepsilon_{\mathrm{U}}$ and $t_{\mathrm{R}} - \varepsilon_{\mathrm{L}}$, respectively, and $\varepsilon_{\mathrm{U}}, \varepsilon_{\mathrm{L}} \to 0^{+}$
- In all practical systems, the values of $\varepsilon$_U and $\varepsilon$_L are nonzero, though they may be very small
- The nonzero values are due to cumulative latency and propagation-delay components (hardware/software)
- Such response times contain **jitter** within the interval $t \in [-\varepsilon_{\mathrm{L}}, +\varepsilon_{\mathrm{U}}]$ (jitter is the amount of variation in response time in ms)

# Example: Where a Response Time Comes from?

- An elevator door is automatically operated and it may have a capacitive safety edge for sensing possible passengers between the closing door blades
- Thus, the door blades can be quickly reopened before they touch the passenger and cause discomfort or even threaten the passenger's safety
- What is the required system response time from when it recognizes that a passenger is between the closing door blades and starting to reopen the door?

# Door Reopening Example Cont'd

This response time consists of five independent latency components:

| | | |
|---|---|---|
| Sensor: | $t_{\mathrm{SE_min}} = 5\ \mathrm{ms}$ | $t_{\mathrm{SE_max}} = 15\ \mathrm{ms}$ |
| Hardware: | $t_{\mathrm{HW_min}} = 1\ \mu\mathrm{s}$ | $t_{\mathrm{HW_max}} = 2\ \mu\mathrm{s}$ |
| System software: | $t_{\mathrm{SS_min}} = 16\ \mu\mathrm{s}$ | $t_{\mathrm{SS_max}} = 48\ \mu\mathrm{s}$ |
| Application software: | $t_{\mathrm{AS_min}} = 0.5\ \mu\mathrm{s}$ | $t_{\mathrm{AS_max}} = 0.5\ \mu\mathrm{s}$ |
| Door drive: | $t_{\mathrm{DD_min}} = 300\ \mathrm{ms}$ | $t_{\mathrm{DD_max}} = 500\ \mathrm{ms}$ |

Now, we can calculate the minimum and maximum values of the composite response time: $t_{\mathrm{min}} \approx 305\ \mathrm{ms},\ t_{\mathrm{max}} \approx 515\ \mathrm{ms}$

The overall response time is dominated by the door drive's response time containing the deceleration time of the moving door blades.

# Definitions: Event and Release Time

Definition: Event

Any occurrence that causes the program counter to change non-sequentially is considered a change of flow-of-control, and thus an event

Definition: Release Time

The release time is the time at which an instance of a scheduled task is ready to run, and is generally associated with an interrupt

# Taxonomy of Events

- An event can be either **synchronous or asynchronous**
  - *Synchronous* events are those that occur at predictable times in the flow-of-control
  - *Asynchronous* events occur at unpredictable points in the flow-of-control and are usually caused by external sources
- Moreover, events can be **periodic, aperiodic or sporadic**
  - A real-time clock that pulses regularly is a *periodic* event
  - Events that do not occur at regular periods are called *aperiodic*
  - Aperiodic events that tend to occur very infrequently are called *sporadic*

# Example: Various Types of Events

| Type | Periodic | Aperiodic | Sporadic |
|---|---|---|---|
| Synchronous | Cyclic code | Conditional branch | Divide-by-zero (trap) interrupt |
| Asynchronous | Clock interrupt | Regular, but not fixed-period interrupt | Power-loss alarm |

# Deterministic Systems

- For any physical system, certain states exist under which the system is considered to be out of control
- The software controlling such a system must therefore avoid these states
- In embedded real-time systems, maintaining overall control is extremely important

Definition: Deterministic System

A system is deterministic, if for each possible state and each set of inputs, a unique set of outputs and next state of the system can be determined

# Deterministic Systems Cont'd

- *Event determinism* means the next states and outputs of a system are known for each set of inputs that trigger events
- Thus, a system that is deterministic is also event deterministic
- However, event determinism may not imply determinism

- While it is a significant challenge to design systems that are completely event deterministic, it is possible to inadvertently end up with a system that is non-deterministic

# Deterministic Systems Cont'd

- Finally, if in a deterministic system the response time for each set of outputs is known, then, the system also exhibits *temporal determinism*

- A side benefit of designing deterministic systems is that guarantees can be given that
  - the system will be able to respond at any time,
  - and in the case of temporally deterministic systems, when they will respond

# CPU Utilization or Time-Loading Factor

- The final term to be defined is a critical measure of real-time system performance
- Because the CPU continues to execute instructions as long as power is applied, it will more or less frequently execute instructions that are not related to the fulfillment of a specific deadline
- The measure of the relative time spent doing *non-idle processing* indicates how much real-time processing is occurring

Definition: CPU Utilization Factor

The CPU utilization or time-loading factor, $U$, is a relative measure of the non-idle processing taking place

# CPU Utilization Zones

- A system is said to be time-overloaded if $U > 100\%$
- Systems that are too highly utilized are problematic
  - Additions, changes, or corrections cannot be made to the system without risk of time-overloading
- A utilization of 50% is common for new products
- However, 70% as a target for $U$ is one of the potentially useful results in the theory of real-time systems where tasks are periodic and independent

# Calculation of *U*

- *U* is calculated by summing the contribution of utilization factors for each task
- Suppose a system has $n \geq 1$ periodic tasks, each with an execution period of $p_i$
- If task *i* is known to have a *worst-case* execution time of $e_i$, then the utilization factor, $u_i$, for task *i* is

$$u_i = e_i / p_i$$

# Calculation of *U*

- Furthermore, the overall system utilization factor is

$$U = \sum_{i=1}^{n} u_i = \sum_{i=1}^{n} e_i / p_i$$

- In practice, the determination of $e_i$ can be difficult, in which case estimation or measuring must be used
- For aperiodic and sporadic tasks, $u_i$ is calculated by assuming a worst-case execution period

# CPU Utilization Zones Cont'd

| Utilization % | Zone Type | Typical Application |
|---|---|---|
| < 26 | Unnecessarily safe | Various |
| 26 – 50 | Very safe | Various |
| 51 – 68 | Safe | Various |
| 69 | Theoretical limit | Embedded systems |
| 70 – 82 | Questionable | Embedded systems |
| 83 – 99 | Dangerous | Embedded systems |
| 100 | Critical | Marginally stressed system |
| > 100 | Overloaded | Stressed system |

# Example: Calculation of *U*

Suppose, an individual elevator controller in a bank of elevators has the following tasks with execution periods of $p_i$ and worst-case execution times of $e_i$, $i \in [1,2,3,4]$:

Task 1: Communicate with the group dispatcher.

Task 2: Update the car position information and manage floor-to-floor runs as well as door control.

Task 3: Register and cancel car calls.

Task 4: Miscellaneous system supervisions.

| $i$ | $e_i$ | $p_i$ |
|---|---|---|
| 1 | 17 ms | 500 ms |
| 2 | 4 ms | 25 ms |
| 3 | 1 ms | 75 ms |
| 4 | 20 ms | 200 ms |

# Example: Calculation of *U*

Suppose, an individual elevator controller in a bank of elevators has the following tasks with execution periods of $p_i$ and worst-case execution times of $e_i$, $i \in [1,2,3,4]$:

Task 1: Communicate with the group dispatcher.

Task 2: Update the car position information and manage floor-to-floor runs as well as door control.

Task 3: Register and cancel car calls.

Task 4: Miscellaneous system supervisions.

| $i$ | $e_i$ | $p_i$ |
|---|---|---|
| 1 | 17 ms | 500 ms |
| 2 | 4 ms | 25 ms |
| 3 | 1 ms | 75 ms |
| 4 | 20 ms | 200 ms |

$$U = \sum_{i=1}^{4} e_i/p_i = 0.31$$

31% (Very safe zone)

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedded Real –Time Operating Systems

# Real Time Operating System

# Operating System (OS)

- Collection of programs that provides an interface between application programs and the computer system (hardware)
- Primary function is to provide application programmers with an abstraction of the system resources, such as memory, input-output and processor
- It enhances the convenience, efficiency and correctness of their use

# Operating System (OS)

OS typically provides

- multitasking
- synchronization
- Interrupt and Event Handling
- Input/Output
- Inter-task Communication
- Timers and Clocks
- Memory Management

# Operating System (OS)

- Core of the OS is the Kernel which is typically a small, highly optimised set of libraries.
- When we hear the word “Operating System” the first ones that come to our mind are those we experience/use in our day to day life
- Windows XP, Linux, Ubuntu, Windows 7, 8, 10, 11 for Computer systems,
- Android and iOS for mobiles and many more

# Operating System (OS)

- Core of the OS is the Kernel which is typically a small, highly optimised set of libraries.
- When we hear the word “Operating System” the first ones that come to our mind are those we experience/use in our day to day life
- Windows XP, Linux, Ubuntu, Windows 7, 8, 10, 11 for Computer systems,
- Android and iOS for mobiles and many more

# Real-Time Operating Systems (RTOS):

- RTOS is a subset of operating system with special features that make it suitable for building real-time computing applications also referred to as *Real-Time Systems (RTS*)
- A Real-Time OS (RTOS) is used for those applications where data processing should be done in a small quantum of time
- It is an operating system intended to serve real time applications that process data as it comes in, mostly without buffer delay.
- **RTOS** is therefore an operating system that supports real-time applications by providing logically correct result **within the deadline required**

# Real-Time Operating Systems (RTOS):

- All embedded systems are not designed with RTOS
- Embedded systems with relatively simple/small hardware/code might not require an RTOS
- Embedded systems with moderate-to-large software applications require some form of scheduling, and hence RTOS

Fig. Real time embedded system with RTOS

# RTOS Architecture:

- The architecture of an RTOS is dependent on the complexity of its deployment
- Good RTOSs are scalable to meet different sets of requirements for different applications
- For simple applications, an RTOS usually comprises only a kernel
- For more complex embedded systems, an RTOS can be a combination of various modules, including the kernel, networking protocol stacks, and other components

# RTOS Architecture:

- For more complex embedded systems, an RTOS can be a combination of various modules
- kernel
- networking protocol stacks
- other components

# RTOS Architecture:

**Kernel**

- Smallest and central component of an operating system
- Heart of every operating system
- Managing memory and devices and also to provide an interface for software applications to use the resources.
- Additional services such as managing protection of programs and multitasking

There are two RTOS architectures: **Monolithic and Microkernel**

# Monolithic RTOS

- Monolithic means “one huge stone”
- Monolithic kernel runs all operating system components in the kernel space
- Monolithic RTOS includes
  - device drivers
  - file management
  - Networking
  - graphics stack as part of the kernel space

# Monolithic RTOS

- Applications, however, run in the user space
- A single programming error in a file system, protocol stack or driver can crash the system
- In addition, any change to a driver or system file requires OS modification and recompiling

- **<u>Example: RTLinux</u>**

# Monolithic RTOS

| Advantages | Disadvantages |
|---|---|
| Process scheduling, memory management and file management all run as a single large process in the same address space, which can improve performance. | The failure of any service can crash the OS. |
| The entire OS is contained in a single static binary file, which may run faster and more reliably than dynamically linked libraries. | Adding or removing a service requires modifying and recompiling the OS. |
| | The OS's kernel services represent a large attack surface - if a service is compromised it can make the whole system vulnerable. |
| | Larger footprint. |
| | Difficult to debug and maintain. |

# Microkernel RTOS

- A microkernel RTOS is structured with a tiny kernel that provides minimal services
- The microkernel works with a team of optional cooperating processes that run outside the kernel space (in the user space)
- The microkernel itself lacks file systems and many other services normally expected of an OS
- In a microkernel, only the core RTOS kernel is granted access to the entire system
  - Improves reliability and security

# Microkernel RTOS

- All other components, including drivers and system-level components, are each contained within their own isolated process space

- Isolation prevents errors in a component from affecting other parts of the system
- The only thing that a component can crash is itself
- Such crashes can be easily detected, and the faulty component can be restarted hot—while the system is still running
- **Example: QNX Neutrino RTOS**

# Microkernel RTOS

| Advantages | Disadvantages |
|---|---|
| Fault isolation and recovery for high availability. | Requires more context switching, which can increase overhead. |
| Restart a failed system service dynamically without impacts to the kernel (no system reboot). | |
| Easy expansion—develop device drivers and OS extensions without a kernel guru and without recompiling. | |
| Easier to debug. | |
| Small footprint. | |
| Less code running in kernel space reduces attack surface and increases security. | |

# Microkernel vs Monolithic

| Category | Microkernel | Monolithic |
|---|---|---|
| Performance | Slightly slower performance due to higher number of context switches. | Better performance due to smaller number of context switches. |
| System Updates | New drivers and OS services updates can be performed with no changes to the kernel, and thus do not require an OS reboot. | New drivers and OS services updates require an OS rebuild and reboot. |
| Power Fail Recovery | Can restart any service individually, without interrupting the kernel. | Recovery of a failed service requires an OS reboot. |
| Qualification and Certification | Easier and less costly to qualify and certify the kernel. Most system updates do not require a full qualification and certification cycle but rather are limited to the updated service or driver. | Difficult and more costly to qualify and certify. System updates require a full qualification and certification cycle as the OS is generally rebuilt. |
| Maintenance | Easier and less time-consuming to maintain and troubleshoot deployed systems. Users can update, troubleshoot and reboot a service without requiring an entire OS reboot. | Challenging and more time-consuming to maintain and troubleshoot. Users require an OS reboot when performing most system updates, troubleshooting steps or a service reboot. |

# Embedded Systems

# ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

# Embedded Real –Time Operating Systems

# Real Time Operating System

# RTOS Architecture:

**Kernel**

- Smallest and central component of an operating system
- Heart of every operating system
- Managing memory and devices and also to provide an interface for software applications to use the resources.
- Additional services such as managing protection of programs and multitasking

There are two RTOS architectures: **Monolithic and Microkernel**

# Monolithic RTOS

- Monolithic means “one huge stone”
- Monolithic kernel runs all operating system components in the kernel space
- Monolithic RTOS includes
  - device drivers
  - file management
  - Networking
  - graphics stack as part of the kernel space

# Monolithic RTOS

- Applications, however, run in the user space
- A single programming error in a file system, protocol stack or driver can crash the system
- In addition, any change to a driver or system file requires OS modification and recompiling

- **<u>Example: RTLinux</u>**

# Monolithic RTOS

| Advantages | Disadvantages |
|---|---|
| Process scheduling, memory management and file management all run as a single large process in the same address space, which can improve performance. | The failure of any service can crash the OS. |
| The entire OS is contained in a single static binary file, which may run faster and more reliably than dynamically linked libraries. | Adding or removing a service requires modifying and recompiling the OS. |
| | The OS's kernel services represent a large attack surface - if a service is compromised it can make the whole system vulnerable. |
| | Larger footprint. |
| | Difficult to debug and maintain. |

# Microkernel RTOS

- A microkernel RTOS is structured with a tiny kernel that provides minimal services
- The microkernel works with a team of optional cooperating processes that run outside the kernel space (in the user space)
- The microkernel itself lacks file systems and many other services normally expected of an OS
- In a microkernel, only the core RTOS kernel is granted access to the entire system
  - Improves reliability and security

# Microkernel RTOS

- All other components, including drivers and system-level components, are each contained within their own isolated process space

- Isolation prevents errors in a component from affecting other parts of the system
- The only thing that a component can crash is itself
- Such crashes can be easily detected, and the faulty component can be restarted hot—while the system is still running
- **Example: QNX Neutrino RTOS**

# Microkernel RTOS

| Advantages | Disadvantages |
|---|---|
| Fault isolation and recovery for high availability. | Requires more context switching, which can increase overhead. |
| Restart a failed system service dynamically without impacts to the kernel (no system reboot). | |
| Easy expansion—develop device drivers and OS extensions without a kernel guru and without recompiling. | |
| Easier to debug. | |
| Small footprint. | |
| Less code running in kernel space reduces attack surface and increases security. | |

# Microkernel vs Monolithic

| Category | Microkernel | Monolithic |
|---|---|---|
| Performance | Slightly slower performance due to higher number of context switches. | Better performance due to smaller number of context switches. |
| System Updates | New drivers and OS services updates can be performed with no changes to the kernel, and thus do not require an OS reboot. | New drivers and OS services updates require an OS rebuild and reboot. |
| Power Fail Recovery | Can restart any service individually, without interrupting the kernel. | Recovery of a failed service requires an OS reboot. |
| Qualification and Certification | Easier and less costly to qualify and certify the kernel. Most system updates do not require a full qualification and certification cycle but rather are limited to the updated service or driver. | Difficult and more costly to qualify and certify. System updates require a full qualification and certification cycle as the OS is generally rebuilt. |
| Maintenance | Easier and less time-consuming to maintain and troubleshoot deployed systems. Users can update, troubleshoot and reboot a service without requiring an entire OS reboot. | Challenging and more time-consuming to maintain and troubleshoot. Users require an OS reboot when performing most system updates, troubleshooting steps or a service reboot. |

# RTOS Features:

- **Multithreading and preemptability** – The scheduler should be able to pre-empt any task in the system and allocate the resource to the thread that needs it most even at peak load.
- **Task Priority** – Preemption defines the capability to identify the task that needs a resource the most and allocates it the control to obtain the resource. In RTOS, such capability is achieved by assigning individual task with the appropriate priority level. Thus, it is important for RTOS to be equipped with this feature.
- **Reliable and Sufficient Inter Task Communication Mechanism** – For multiple tasks to communicate in a timely manner and to ensure data integrity among each other, reliable and sufficient inter-task communication and synchronization mechanisms are required. .

# RTOS Features:

- **Priority Inheritance** – Priority inheritance is a mechanism to ensure that lower priority tasks cannot obstruct the execution of higher priority tasks. RTOS should have large number of priority levels & should prevent priority inversion using priority inheritance.
- **Predefined Short Latencies** – An RTOS needs to have accurately defined short timing of its system calls. The behavior metrics are: *1) Task switching latency:* The time needed to save the context of a currently executing task and switching to another task is desirable to be short. *2) Interrupt latency*: The time elapsed between execution of the last instruction of the interrupted task and the first instruction in the interrupt handler. *3) Interrupt dispatch latency:* The time from the last instruction in the interrupt handler to the next task scheduled to run.

# Factors for selecting an RTOS (Selection criteria):

- **Run-time Performance**: Performance is the most important factor required to be considered while selecting for a RTOS**.** RTOS run time performance is governed by context switching time, interrupt latency and other kernel performance metrics. Run time performance consideration is ideal when an application's performance assessment on an RTOS is prototyping performance critical aspects on standard hardware.
- **Scalability:** Most RTOSs are scalable in which only the required code is added to the final memory footprint. Therefore, finding granular scalability in an RTOS is important because it minimizes memory usage.
- **Portability:** An application may outgrow its hardware as product requirements increase. An RTOS with appropriate capabilities can be ported between specific target systems and processor architectures.

# Factors for selecting RTOS (Selection criteria):

- **Middleware**

➢ Almost all real-time operating systems feature middleware components or third-party components that are integrated with the RTOS. You should evaluate the middleware to ensure that it has a seamless integration method. If your RTOS lacks middleware support, you may have to deal with time-consuming integration processes. Ensure that your middleware features components like TCP/IP , file systems, GUI and USB.

➢ A real-time operating system should be of premium quality and easy to navigate. Developing embedded projects is hard and time-consuming; developers should not have to struggle with real-time system-related issues that can be distracting. An RTOS should be a trusted component that any developer can count on.

➢ Good examples of real-time operating systems are the RTX (32-bit) and RTX64 (64-bit) solutions that allow you and your team to focus on adding value to your applications. This software is designed to serve as a hard real-time system that delivers output within a specified time frame to improve embedded systems' quality.

# Factors for selecting RTOS (Selection criteria):

- **Security:** For any device that is going to be connected to the internet, selecting an RTOS that has gone through security certifications and was built for security is a must. Security certifications for an RTOS are relatively new but will be an important criterion for any team that wants to make sure every component in their system is designed with security in mind. One immediate example is verifying that the RTOS has Arm's Platform Security Architecture (PSA) certification at least to level one.

- **Cost:** With new RTOS vendors emerging daily, cost should be a main consideration when selecting an RTOS. Some RTOS packages are offered as complete operating systems that include a real-time kernel, windowing systems, input and output manager, language interface libraries, a file system, networking, cross platform compilers and debuggers. Note that RTOS costs range from $70 to over $20,000. Some RTOS vendors may charge per target system based loyalties that range from $5 to $200 per unit.

# Factors for selecting RTOS (Selection criteria):

- **Unique features**: Every real-time operating system has unique features that determine how it operates to execute commands. As such, you must evaluate the features required to run your system effectively and choose an RTOS with the relevant features. A good RTOS should be scalable and feature efficient memory protection systems.

# Misconception of RTOS:

- **RTOS must be fast:** The responsiveness of an RTOS depends on its deterministic behavior and not on its processing speed. The ability of RTOS to respond to events within a timeline does not imply it is fast.
- **RTOS introduce considerable amount of overhead on CPU:** An RTOS typically only require between 1% to 4% of a CPU time.
- **All RTOS are the same:** RTOS are generally designed for 3 types of real-time systems (i.e. hard, firm & soft). In addition, they are further classified according to the types of hardware devices (e.g. 8-bit, 16-bit, 32-bit MPU) supported.

# Examples of RTOS:

- LynxOS
- OSE
- QNX (Most traditional RTOS in the market)
- **VxWorks** (Most widely adopted RTOS in the embedded industry, Used in famous NASA rover robots Spirit and Opportunity, Certified by several agencies and international standards for real time systems, reliability and security-critical applications)
- Symbian (Designed for Smartphones, Supported by ARM, x86 architecture)
- Windows CE
- RT Linux