

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

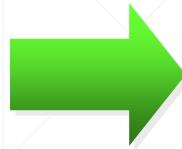
Module 1

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



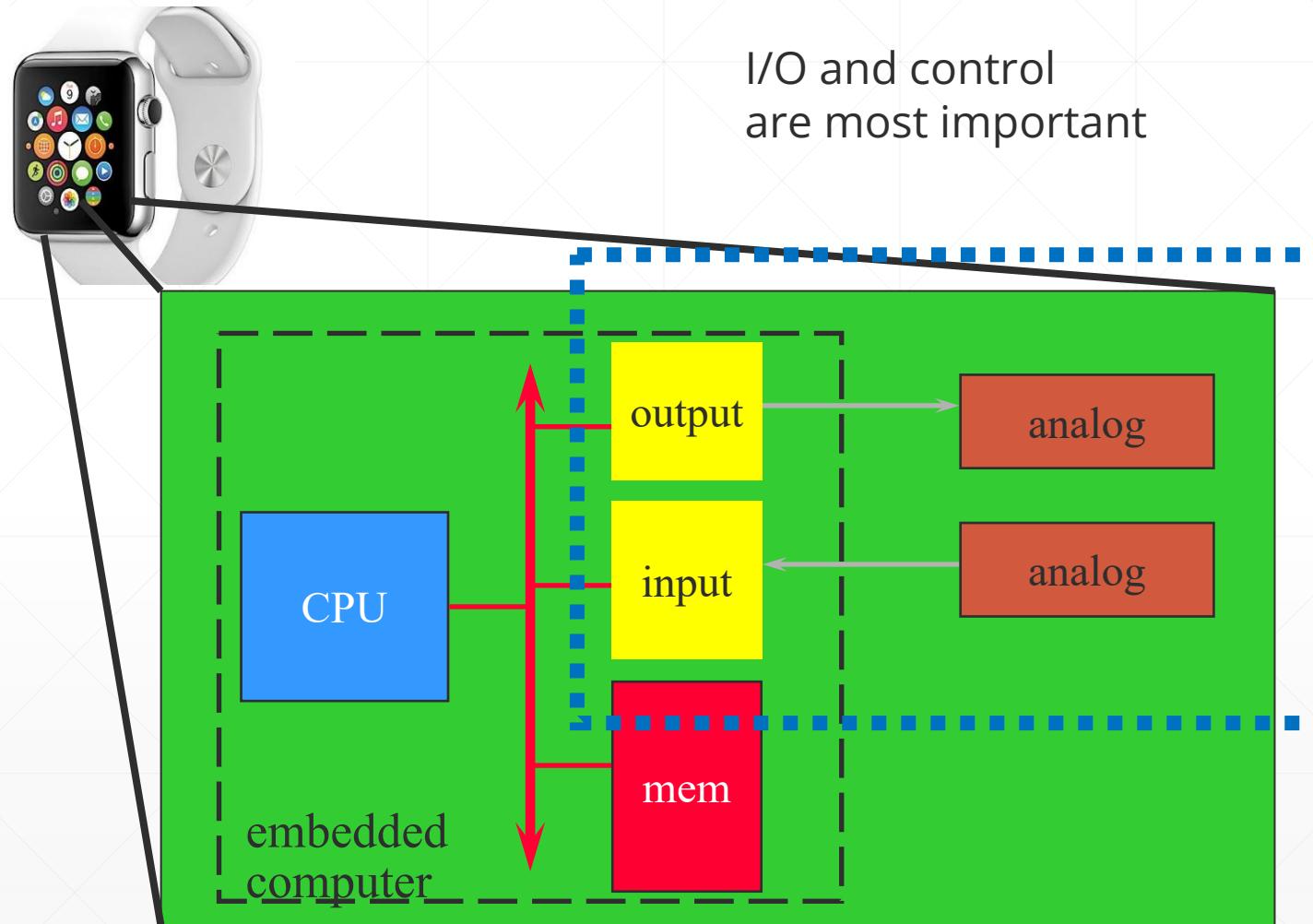
What Is an Embedded System?

- Information processing systems embedded 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

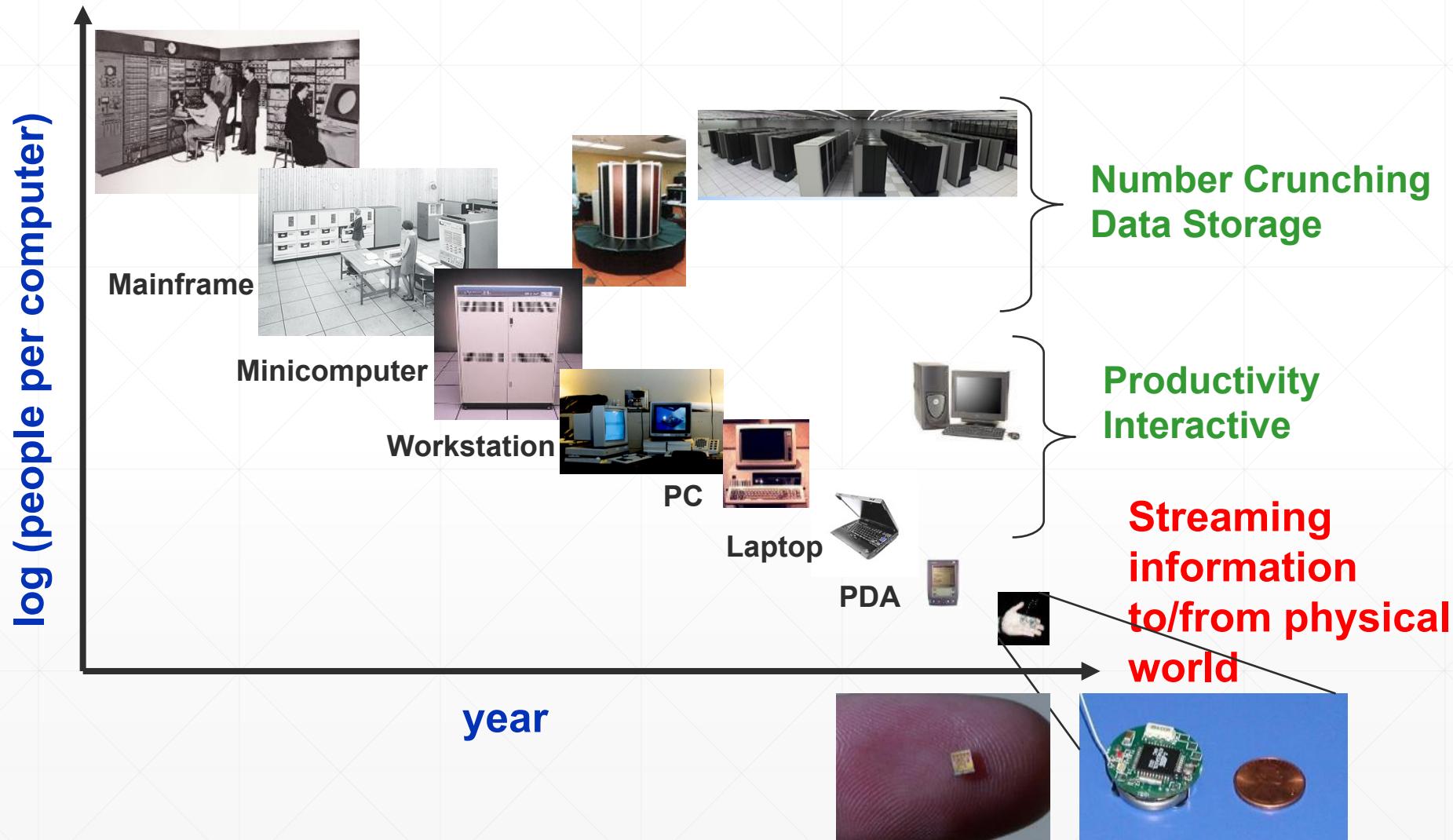
- A product, e.g., video decoder, may be implemented using pure hardware or microprocessor + embedded software
 - A chip may be implemented using pure logic gates or a microprocessor + peripheral logic + software (SoC)
- An embedded system may or may not have OS
 - Simple systems may be implemented by a single program that runs continuously
 - Systems that need to control and respond to many activities may require an OS for management

Embedded = Smart

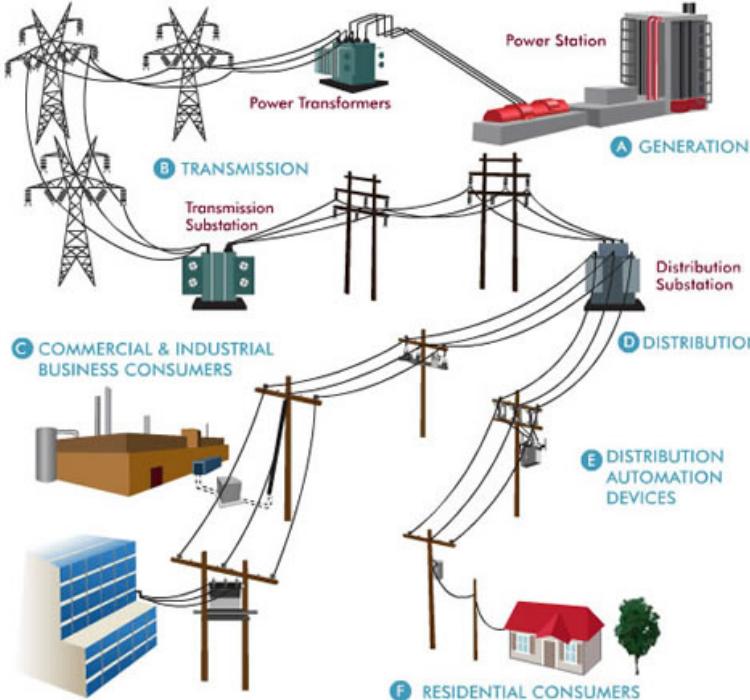
- Computers embedded into objects
 - Augment objects with programmatic control, communication, **sensing**, and **actuation**
- Let the world know you:
 - Make physical objects/phenomena accessible to digital world
- Let you know the world:
 - Give intelligence/life to physical objects so that they can 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)

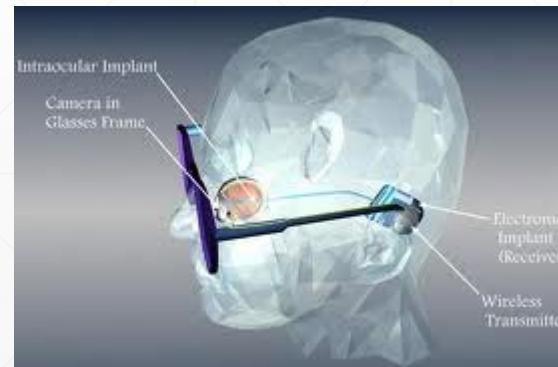
Future Embedded Systems



Smart Glasses



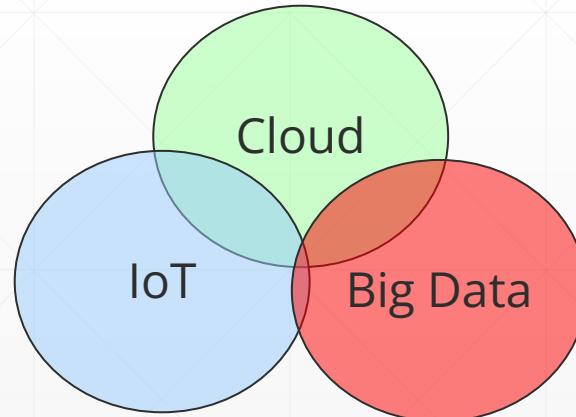
Self-driving Cars



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

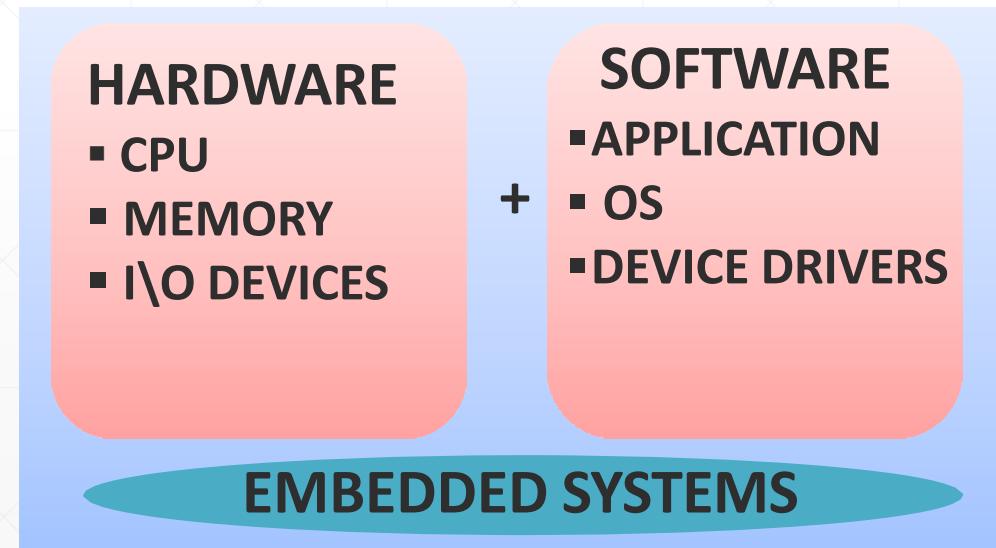
Embedded Systems

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

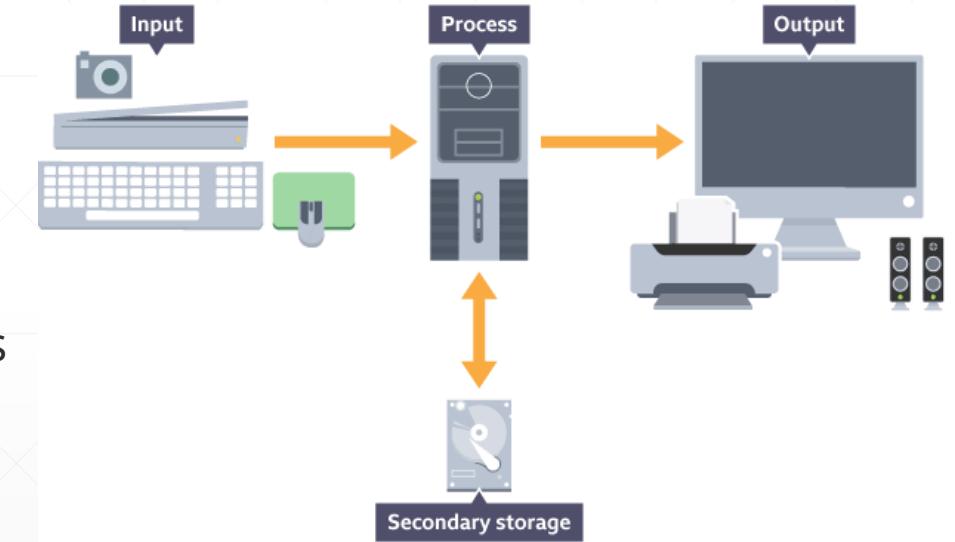
- Every embedded system consists of custom-built hardware built around a Central Processing Unit (CPU)
- Memory hardware also contains memory chips onto which the software is loaded.



General Purpose Systems

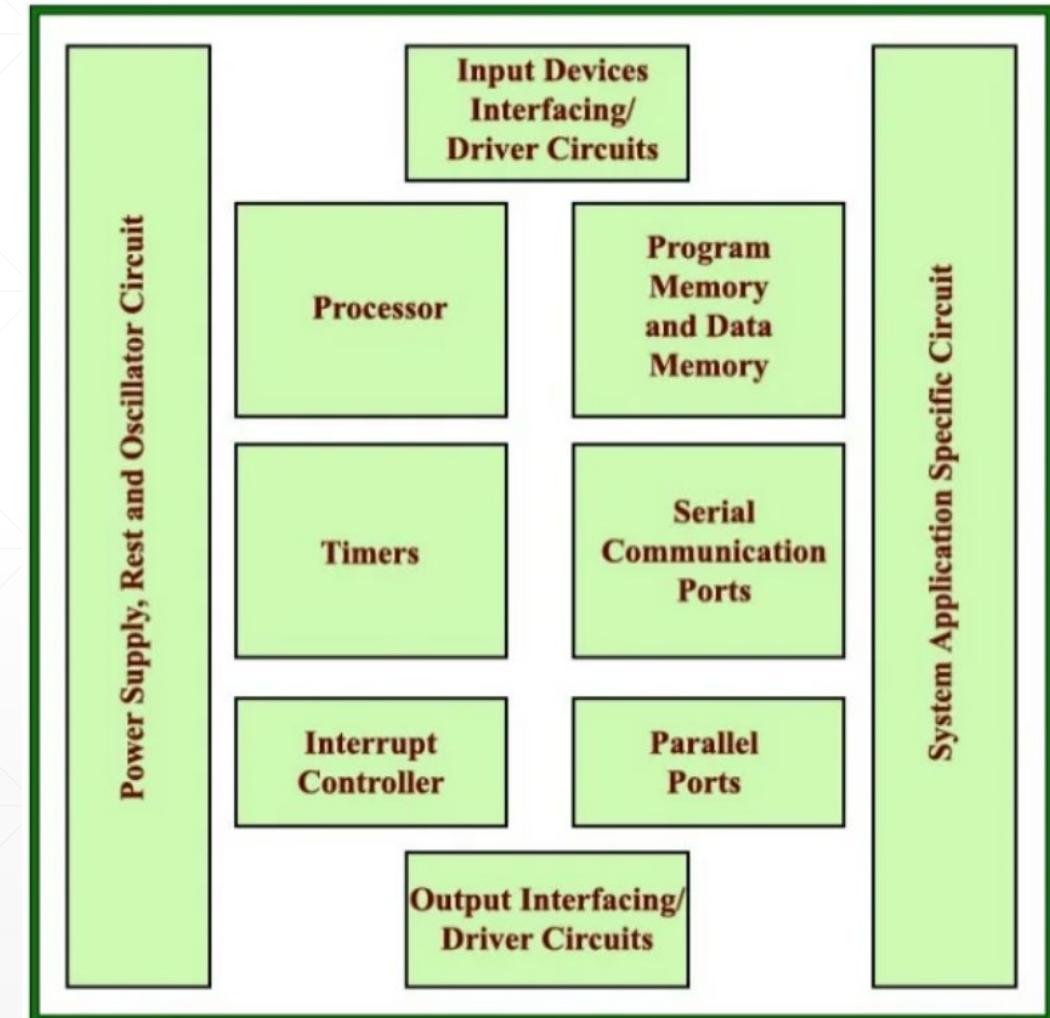
General Purpose System : Computer

- 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

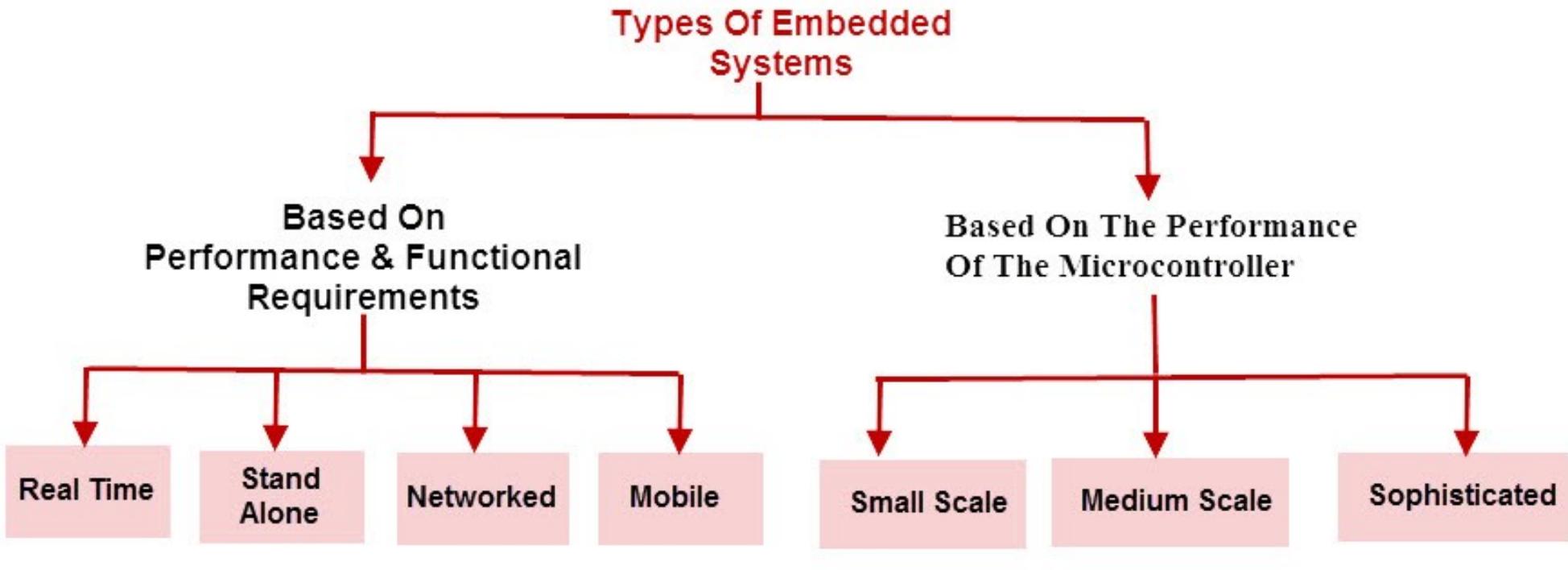


Embedded Systems

- 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

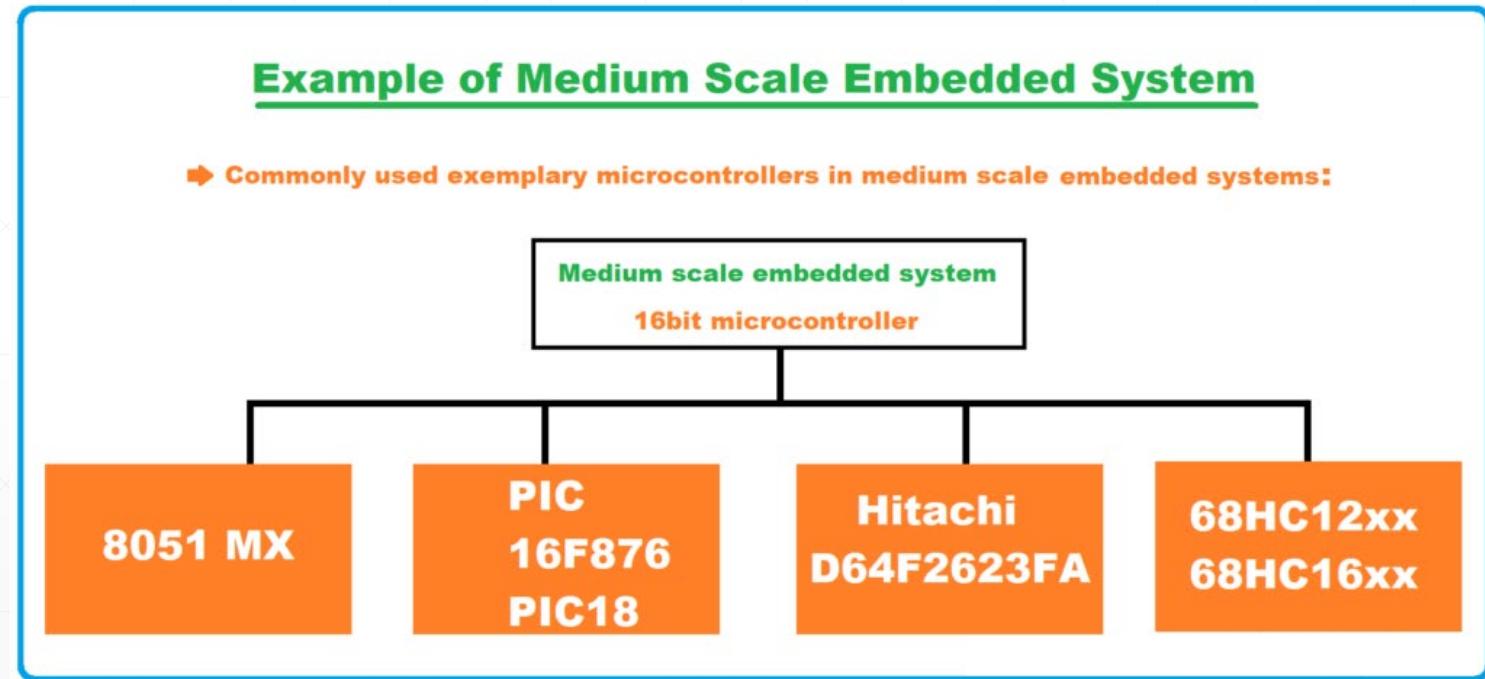
- **Small scale embedded systems:** Use 8-bit or 16-bit microcontrollers and have minimum hardware and software. They are so small and require little power they may be powered by a battery.
- **Medium scale embedded systems:** Use either one or a few 16 bit or 32-bit microcontrollers, RISCs or DSPs. 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 programming language is used in such embedded systems
- Examples
 - Bluetooth headphones
 - Digital pedometer

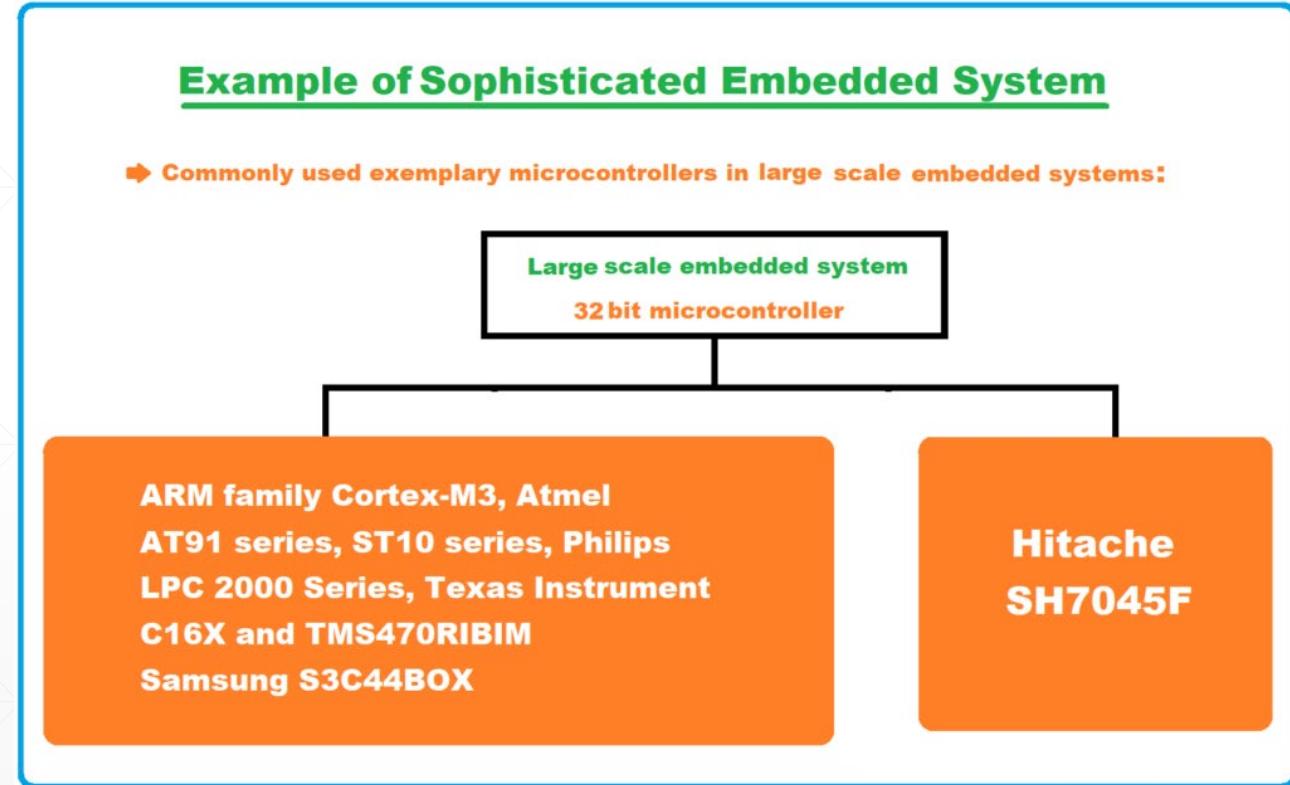
Embedded Systems - Classification

- 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

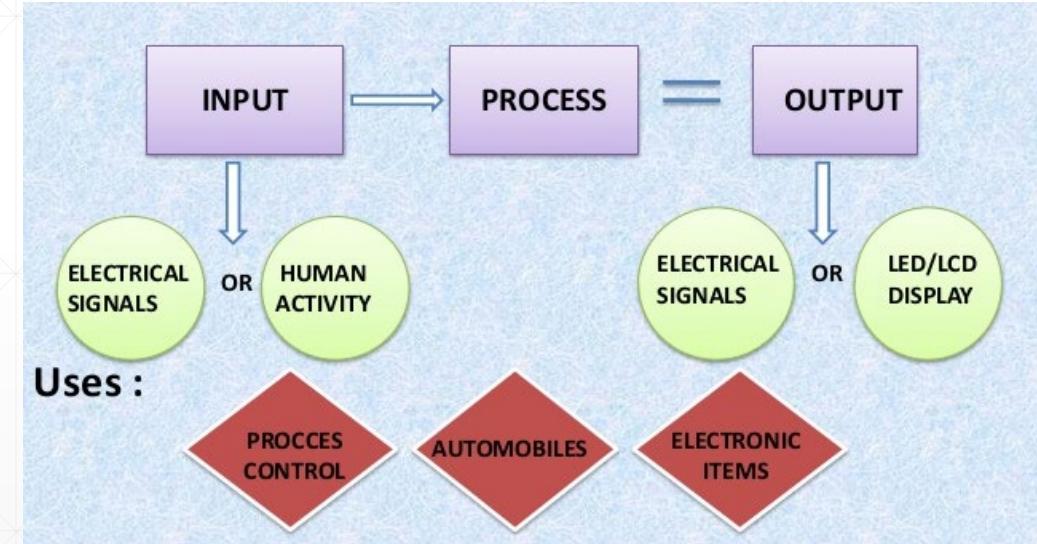
- Sophisticated embedded systems:
- Sophisticated embedded systems, as the name suggests are highly advanced and developed in terms of hardware and software.
- Examples : Medical Systems, Vehicles, etc



Embedded Systems - Classification

Stand alone embedded systems

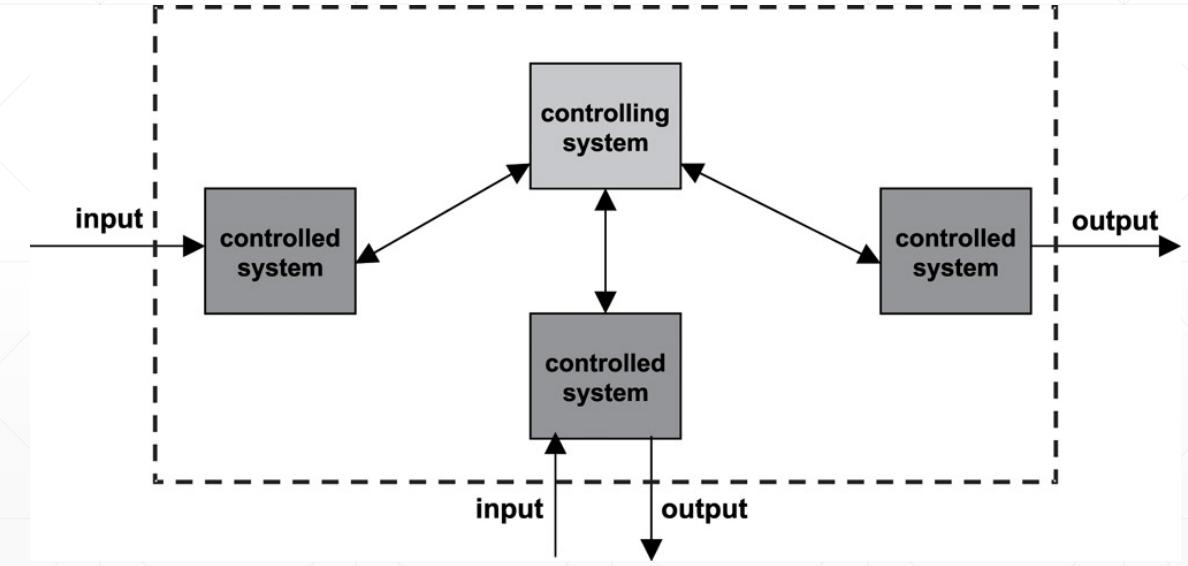
- Do not require a host system like a computer, it works by itself.
- It takes the input from the input ports either analog or digital and processes, calculates and converts the data and gives the resulting data through the connected device



Embedded Systems - Classification

Real time embedded systems

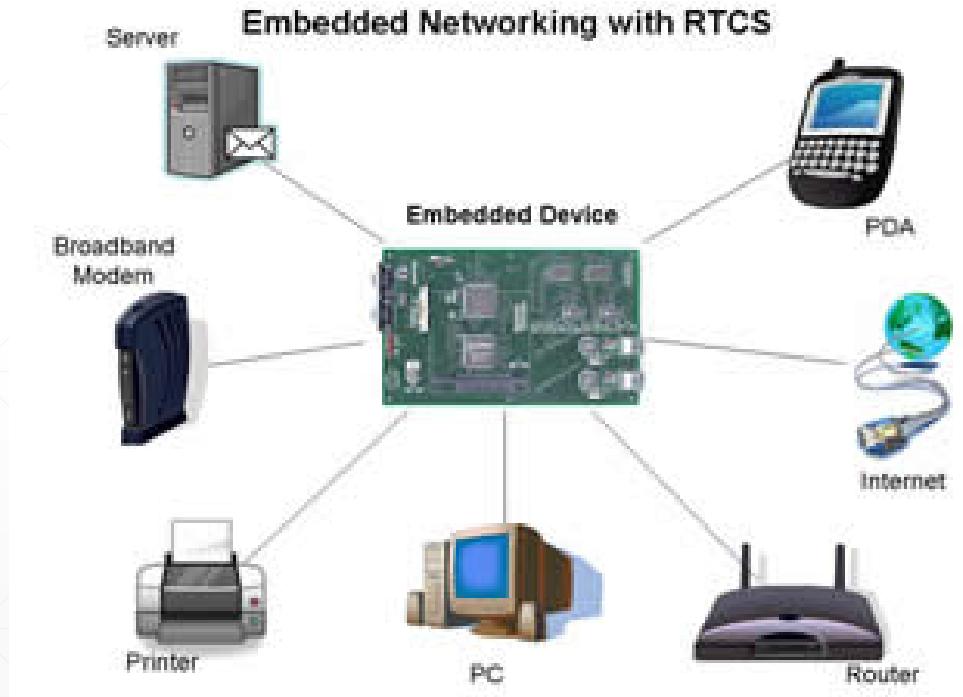
- Give the required o/p in a particular time.
- These types of embedded systems follow the time deadlines for completion of a task.
- Classified into two types such as soft and hard real time systems.



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 system applications.



Embedded Systems - Classification

Mobile embedded systems

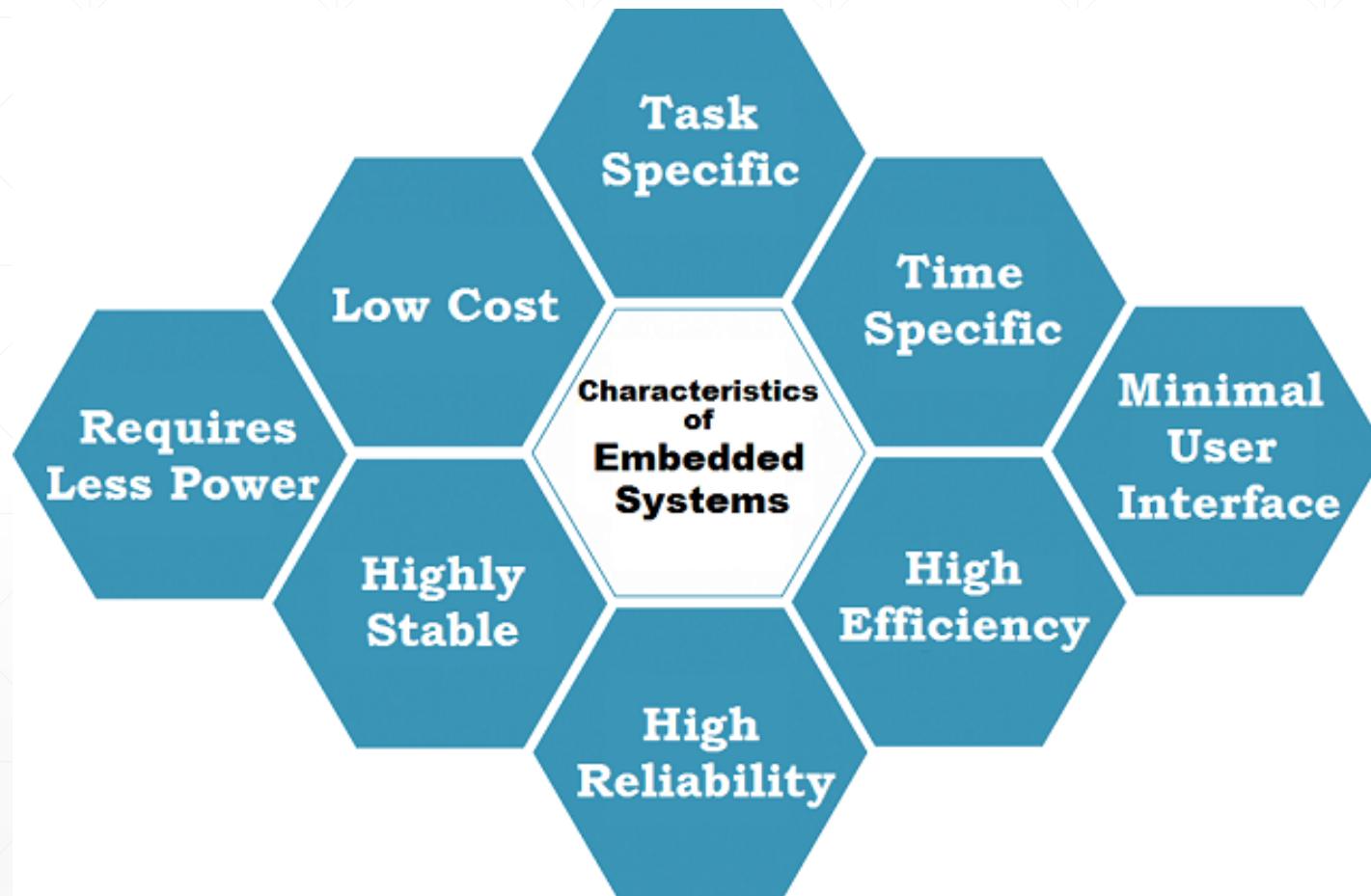
- Used in portable embedded devices like cell phones, mobiles, digital cameras, mp3 players and personal digital assistants, etc.
- The basic limitation of these devices is having limited resources.



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

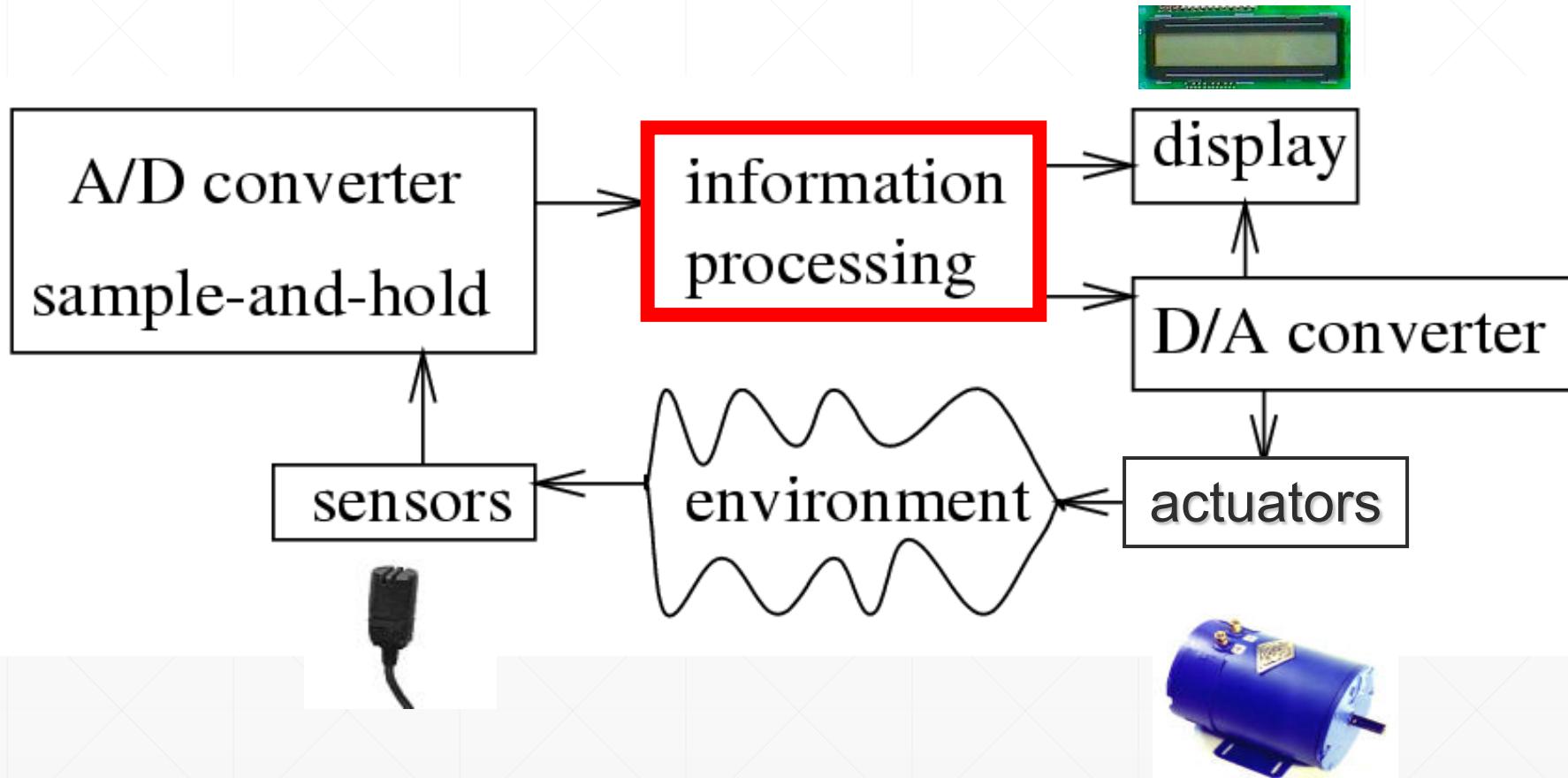
Components of Embedded Systems

- 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

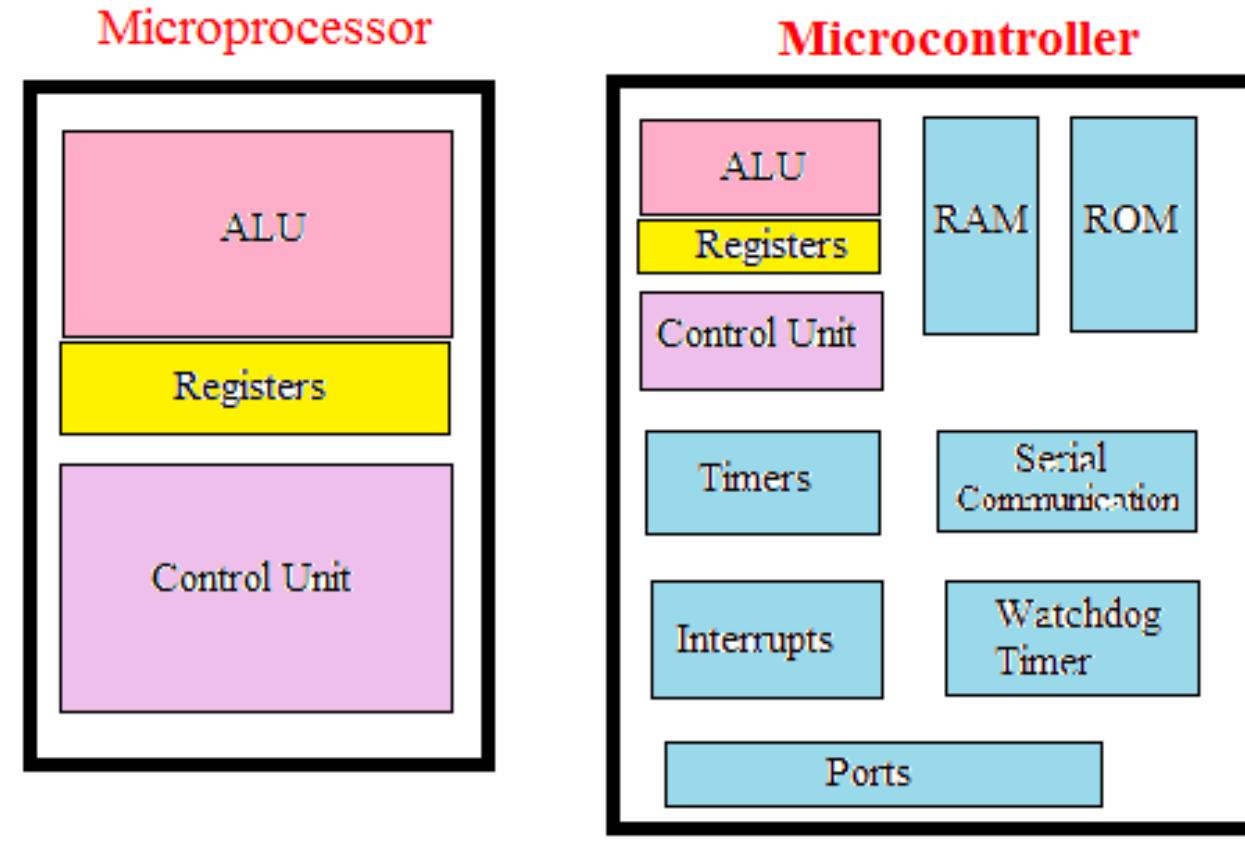
Embedded Systems: Simplified Block Diagram



Microprocessor vs Microcontroller

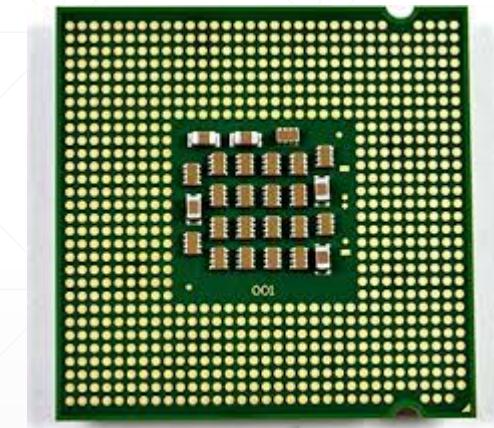
	Microprocessor	Microcontroller
1	Microprocessor acts as a heart of computer system.	Microcontroller acts as a heart of embedded system.
2	It is a processor in which memory and I/O output component is connected externally.	It is a controlling device in which memory and I/O output component is present 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



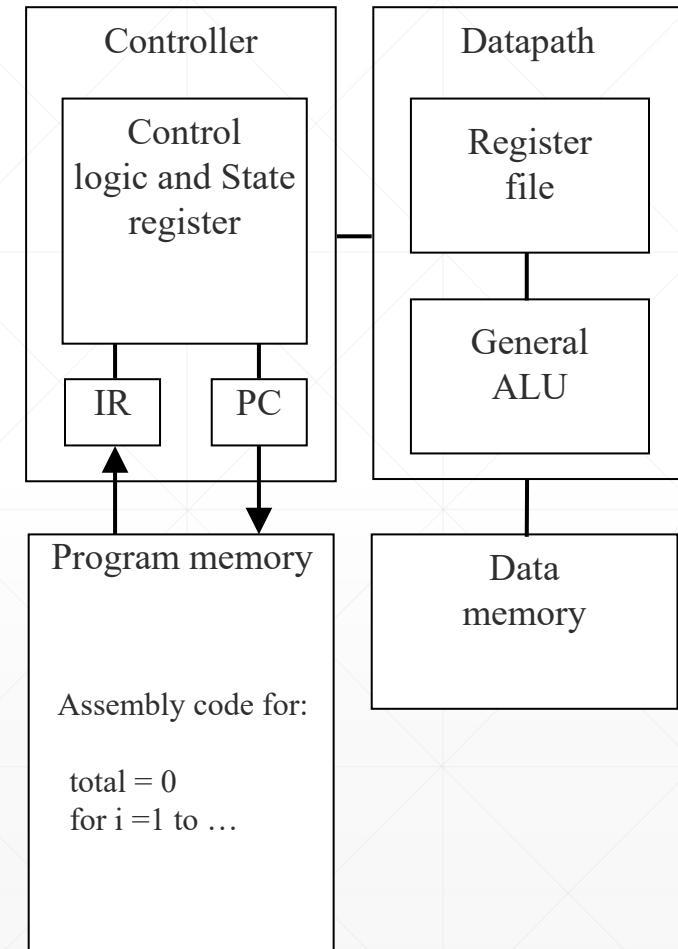
Processors

- 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)



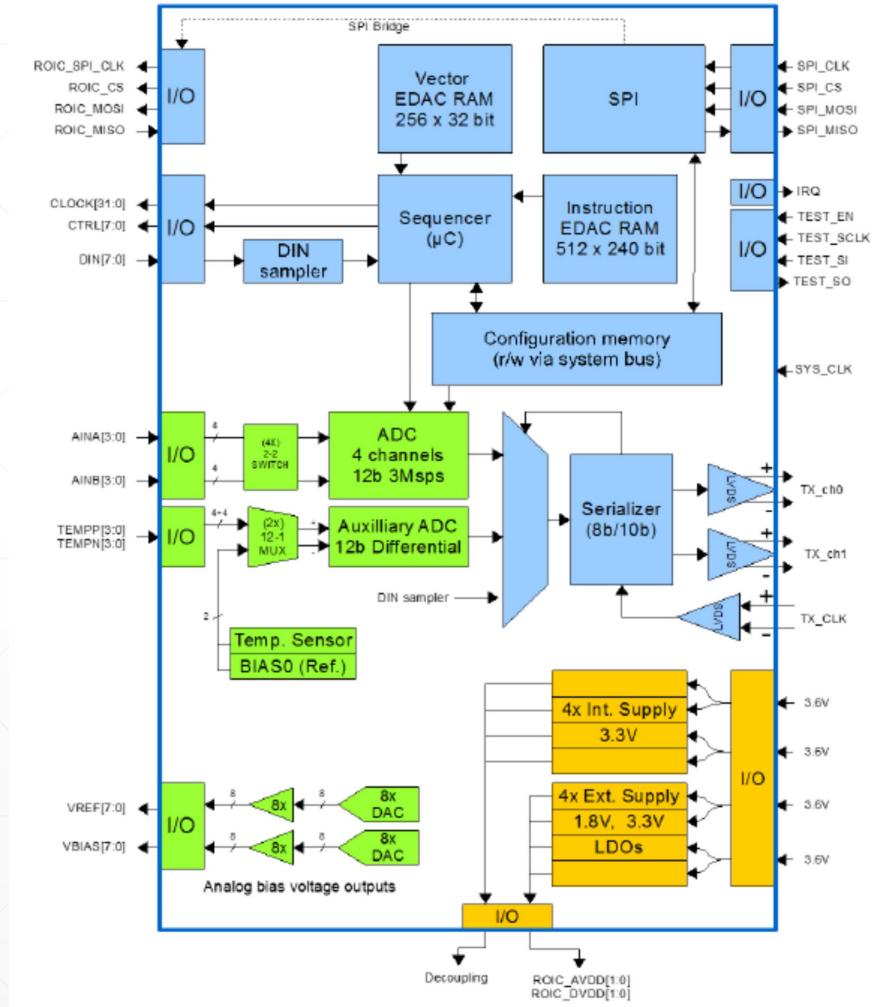
General-purpose processors

- 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

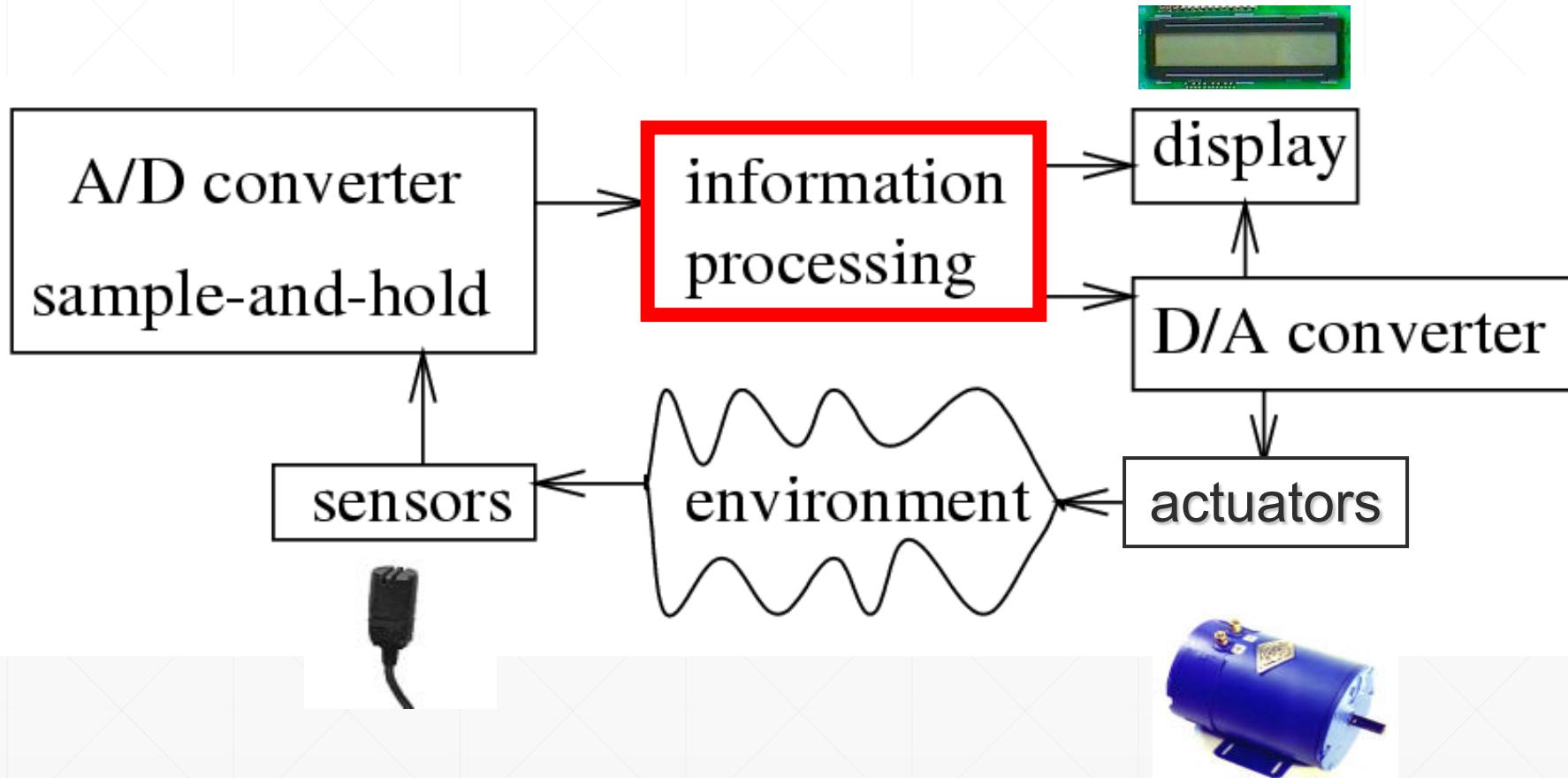
ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

Module 1

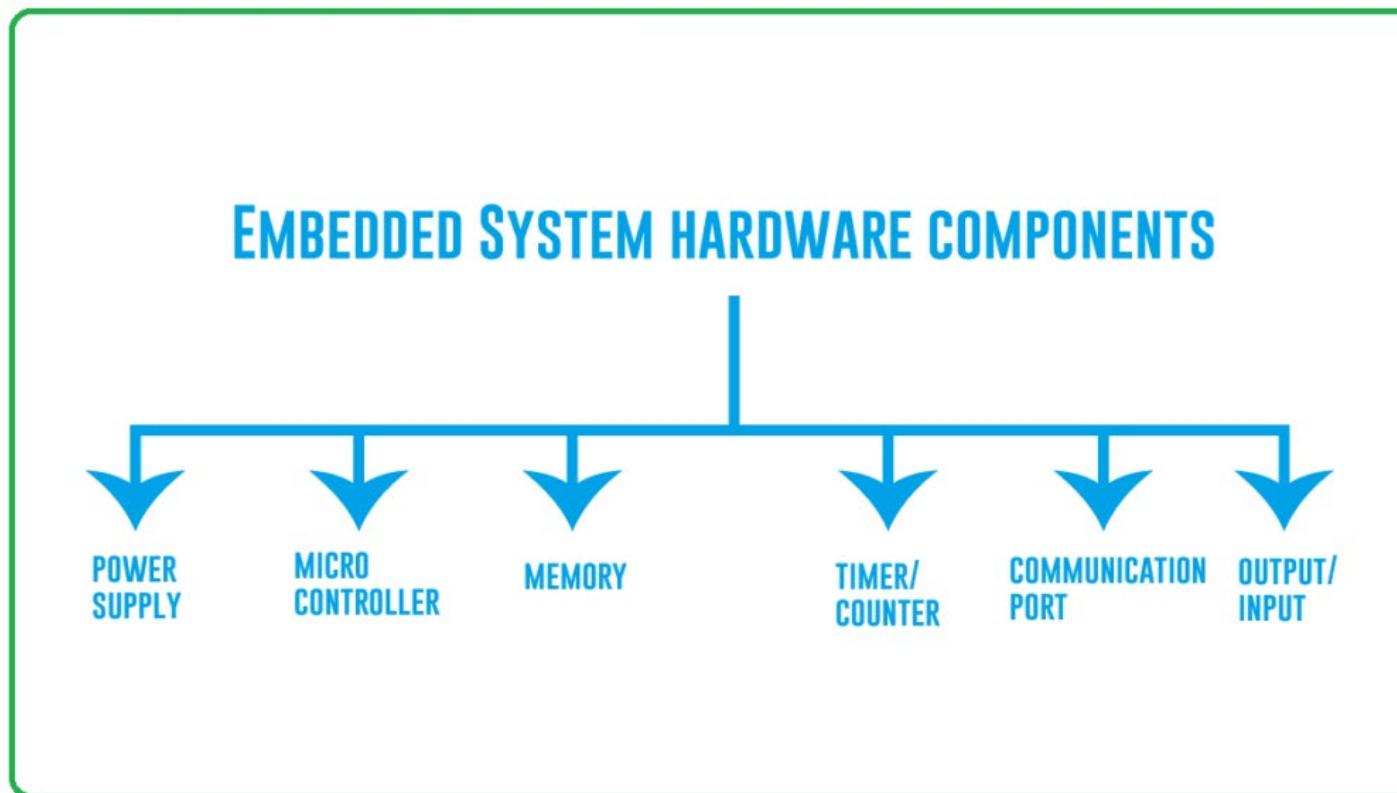
Introduction to Embedded System

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

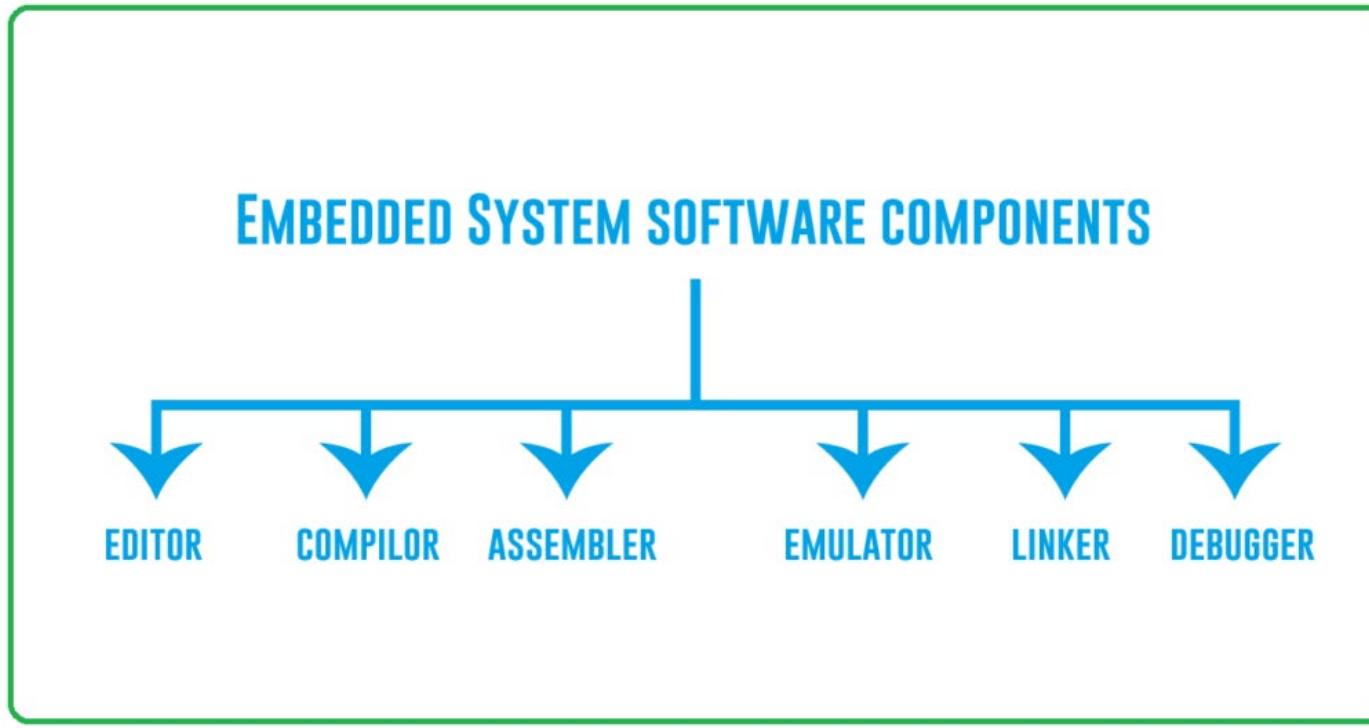
Components of Embedded Systems - Hardware

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

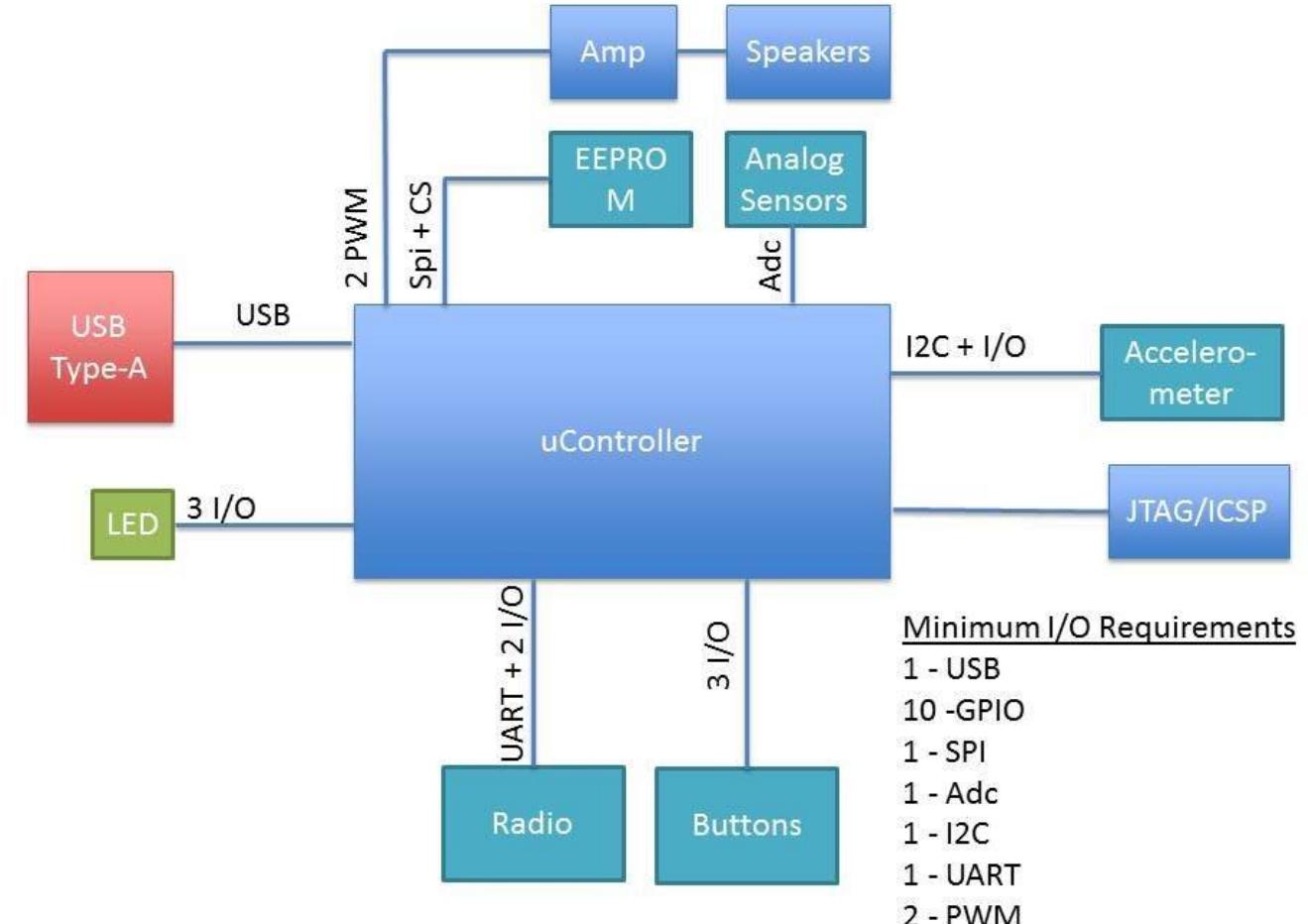
Step 1 - Make a list of required hardware interfaces

- 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 contains an 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 this machine. 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 port is 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.

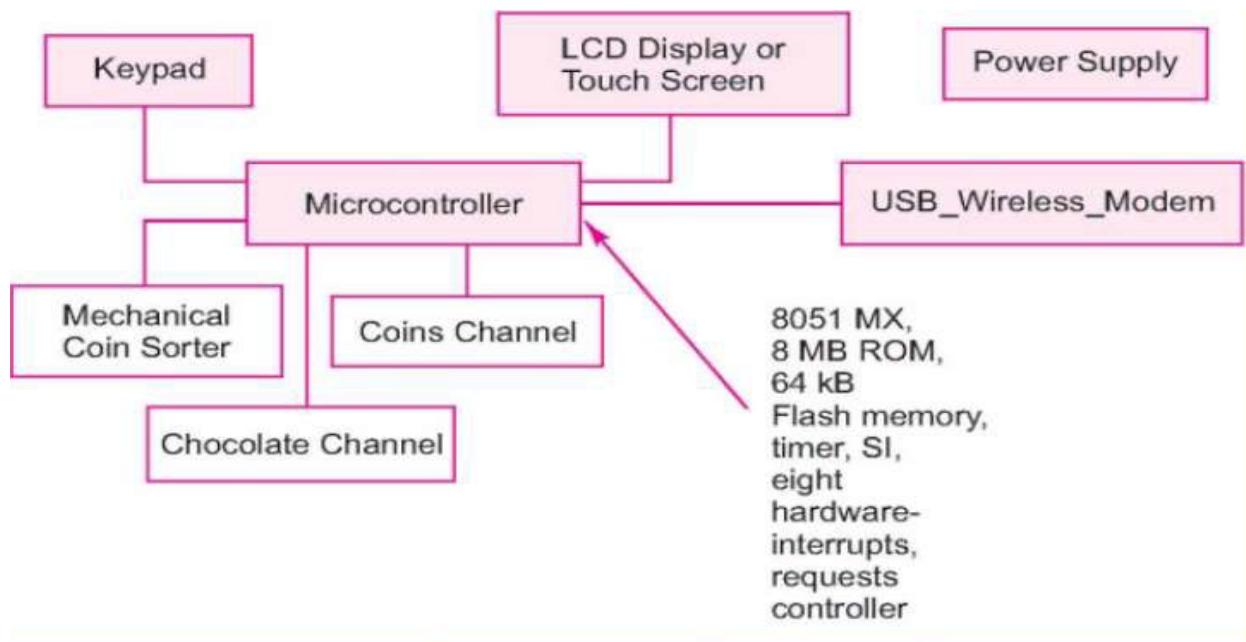


Figure 1. Block Diagram of ACVM.

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 amount of 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 LCD's right-hand bottom side. ACVM has a 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.

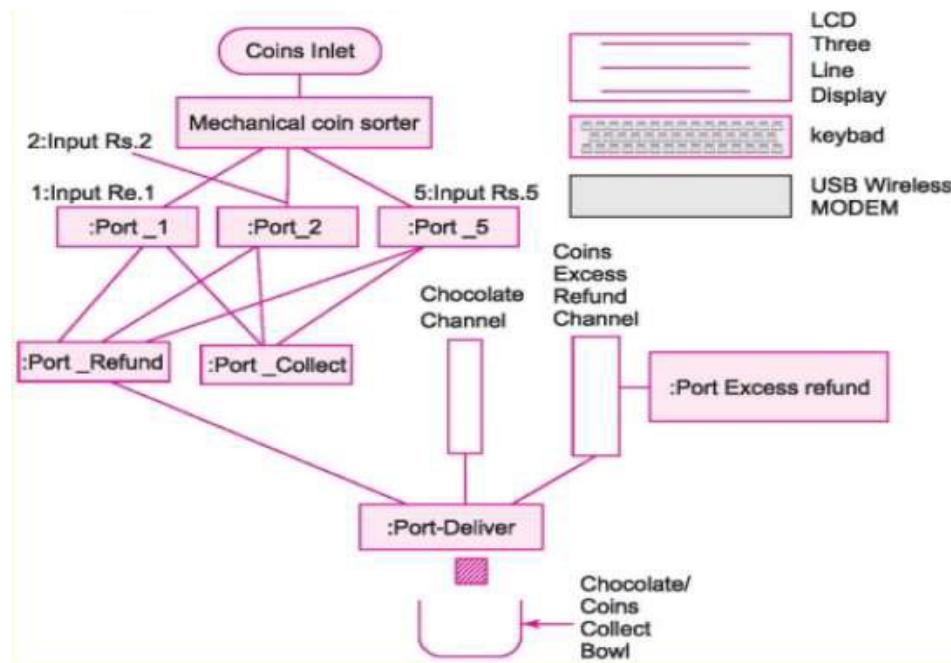


Figure 2. Basic System of an ACVM

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 of data 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.

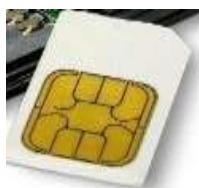


Figure 3. Contact based smart card

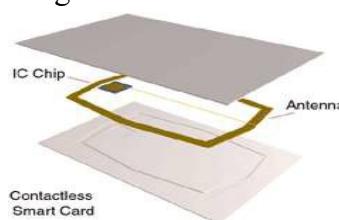


Figure 4. Contactless smart card

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_App) 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_App 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 screen or 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.

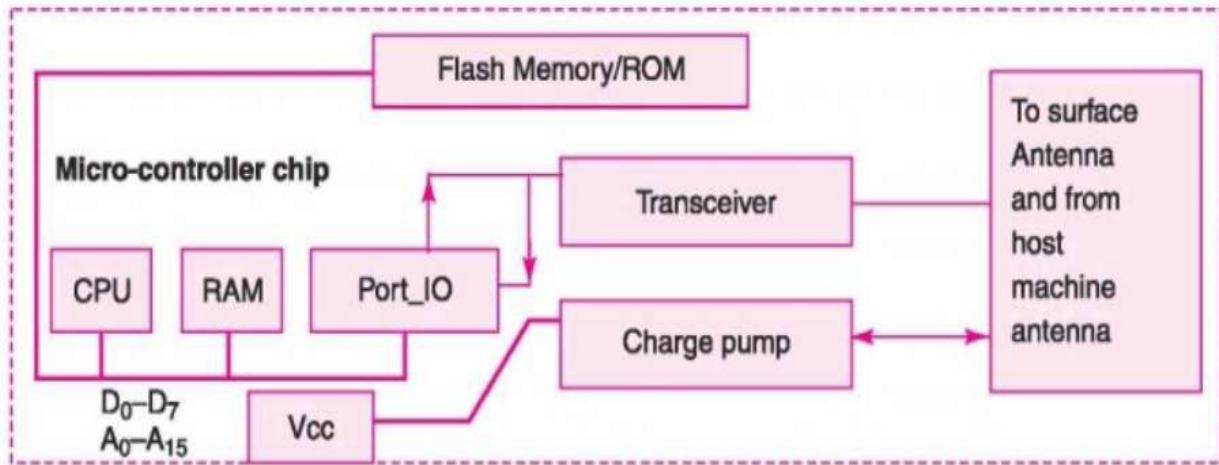


Figure 6. Smart Card Hardware System

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 physical addresses, which are different from the logical addresses used in the program. The EEPROM or Flash 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 extracts the 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-Hill publication, 2008.

Embedded Systems

ECE 4010

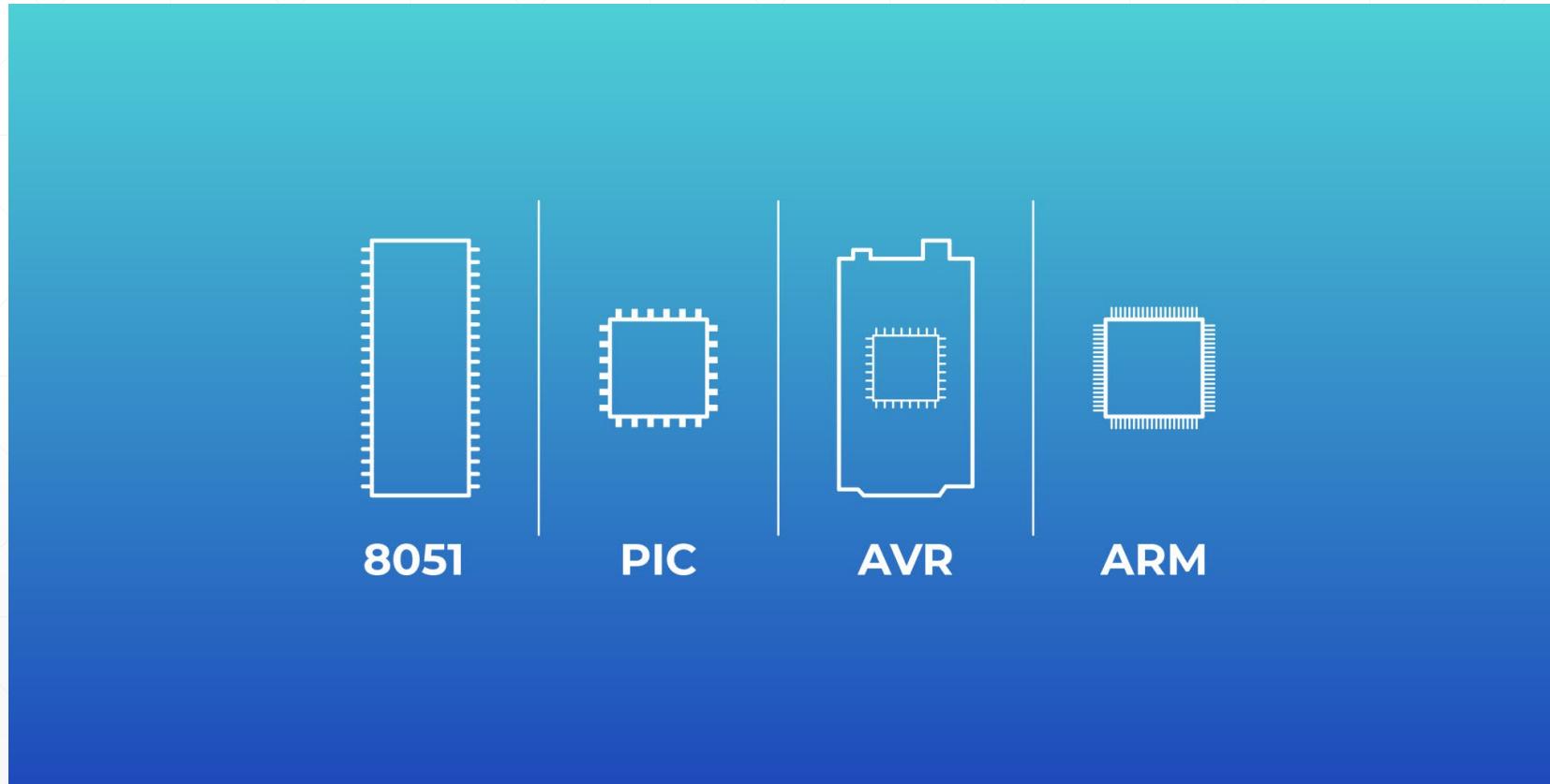
- Abhishek Joshi
SEEE, VIT Bhopal

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 widely used in client electronic devices like smartphones, tablets, multimedia players and other mobile devices, such as wearables
- Because of their reduced 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, Cypress, 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

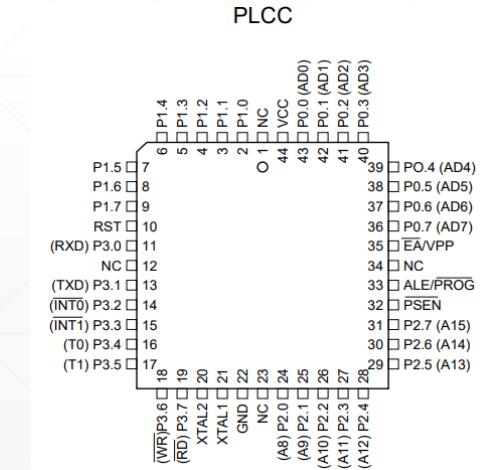
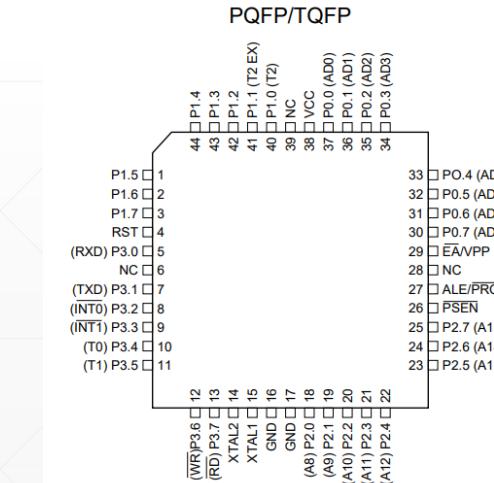
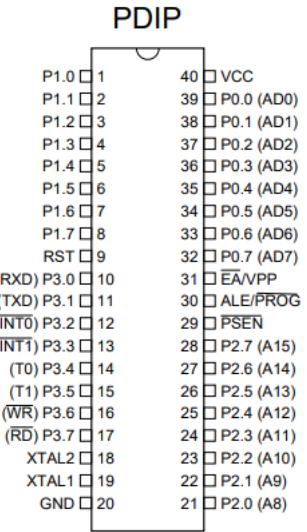
P1.0	1	40	VCC
P1.1	2	39	P0.0 (AD0)
P1.2	3	38	P0.1 (AD1)
P1.3	4	37	P0.2 (AD2)
P1.4	5	36	P0.3 (AD3)
P1.5	6	35	P0.4 (AD4)
P1.6	7	34	P0.5 (AD5)
P1.7	8	33	P0.6 (AD6)
RST	9	32	P0.7 (AD7)
(RXD) P3.0	10	31	EA/VPP
(TXD) P3.1	11	30	ALE/PROG
(INT0) P3.2	12	29	PSEN
(INT1) P3.3	13	28	P2.7 (A15)
(T0) P3.4	14	27	P2.6 (A14)
(T1) P3.5	15	26	P2.5 (A13)
(WR) P3.6	16	25	P2.4 (A12)
(RD) P3.7	17	24	P2.3 (A11)
XTAL2	18	23	P2.2 (A10)
XTAL1	19	22	P2.1 (A9)
GND	20	21	P2.0 (A8)

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.

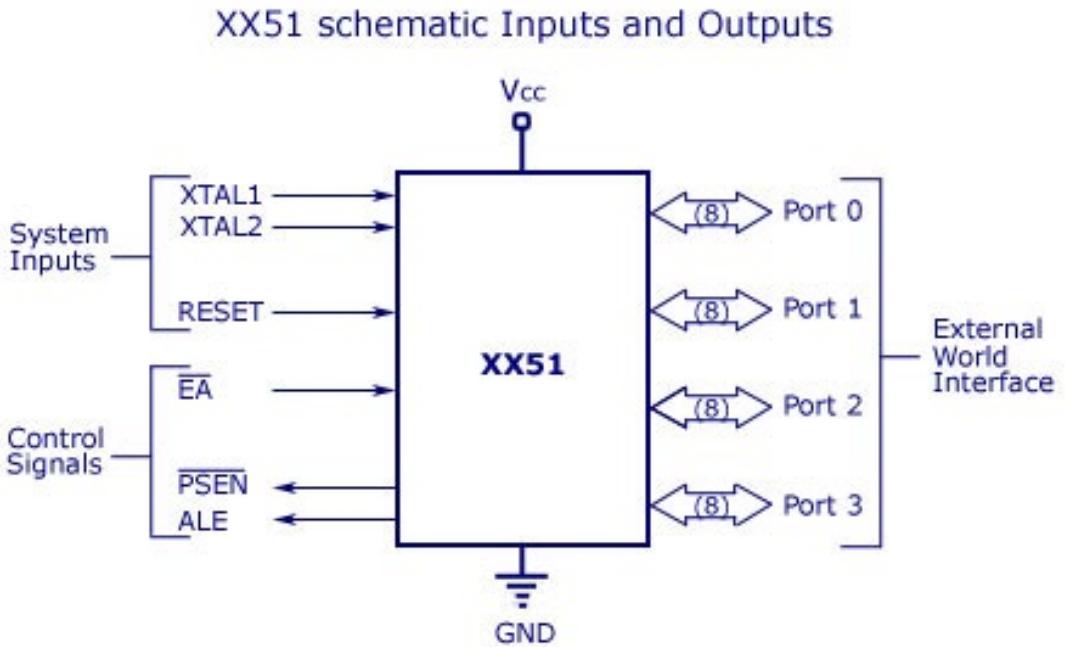
Microcontroller - 8051 - Packaging

- 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



Microcontroller - 8051 - Architecture

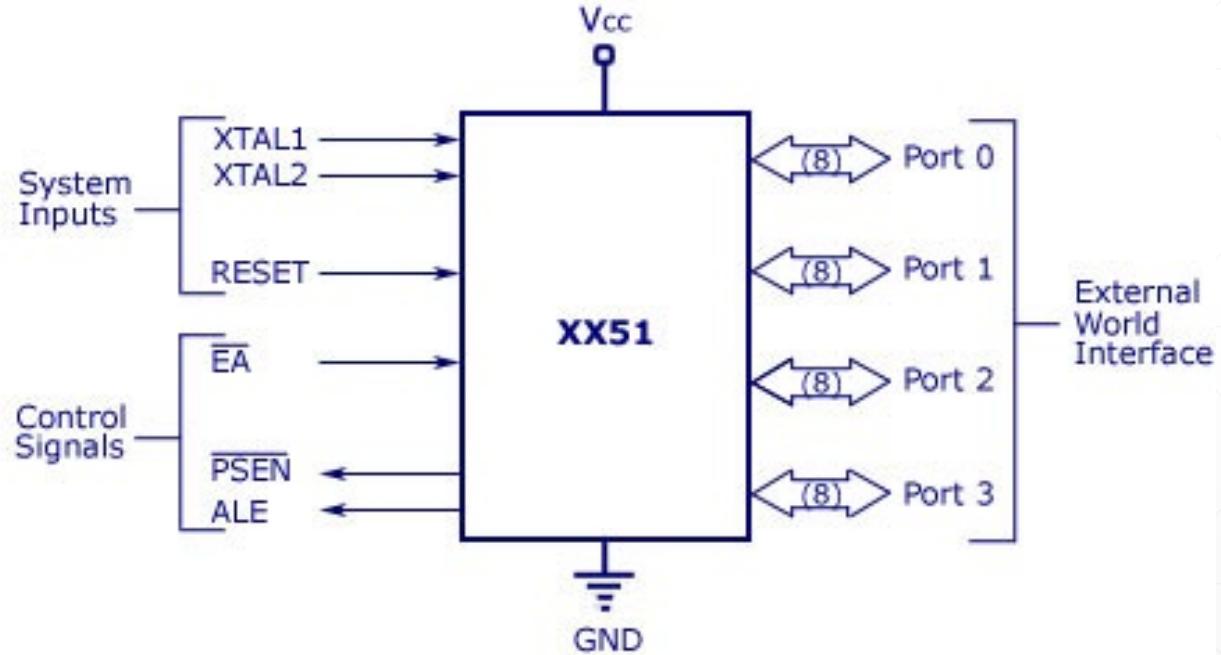
- 8051 Architecture - internal architecture, pin configuration, program memory and data memory organization
- The basic architecture remains same for the MCS-51 family
- In general, all microcontrollers in MCS- 51 family are represented by XX51, where XX can take values like 80, 89 etc.



Microcontroller – 8051 - Inputs

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

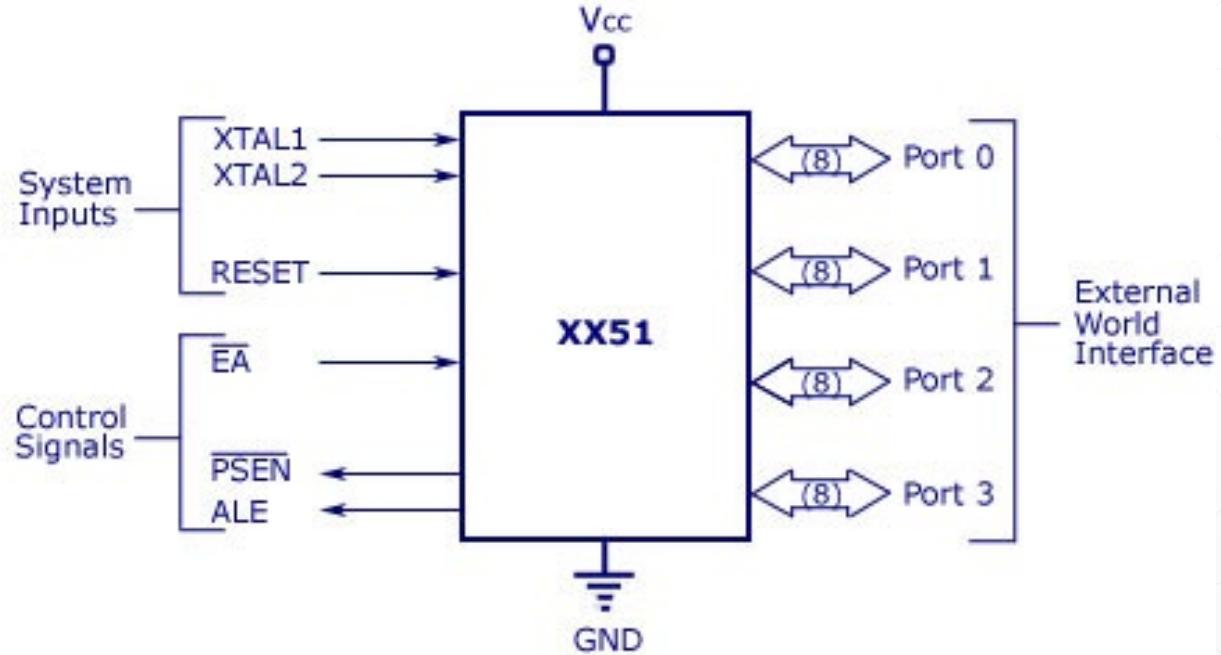
XX51 schematic Inputs and Outputs



Microcontroller – 8051 – Control Signals

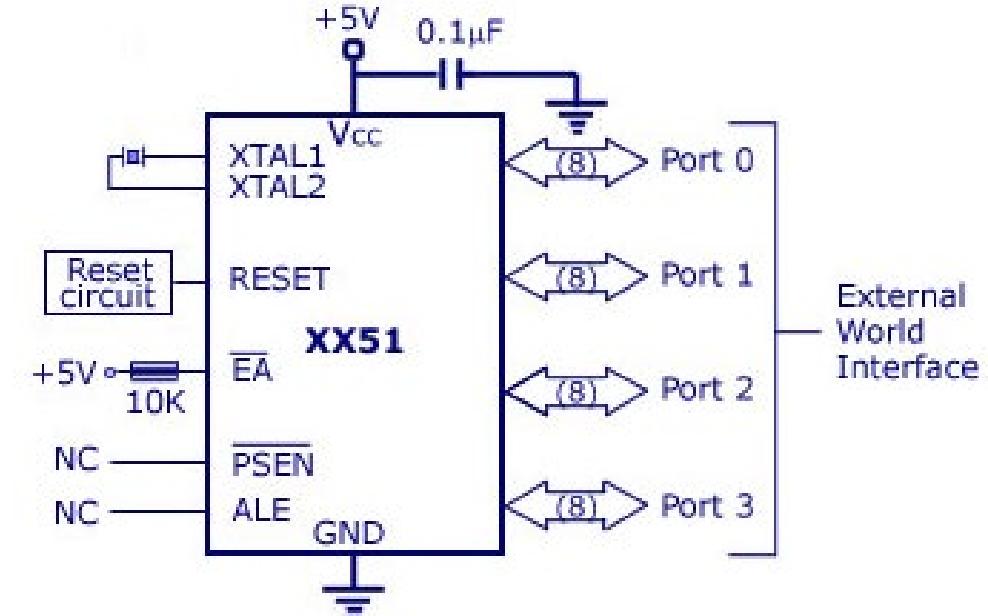
- 3 Control Signals – Used for external memory interface
- **EA** – External Access
- **PSEN** - Program Store Enable
- **ALE** - Address Latch Enable

XX51 schematic Inputs and Outputs



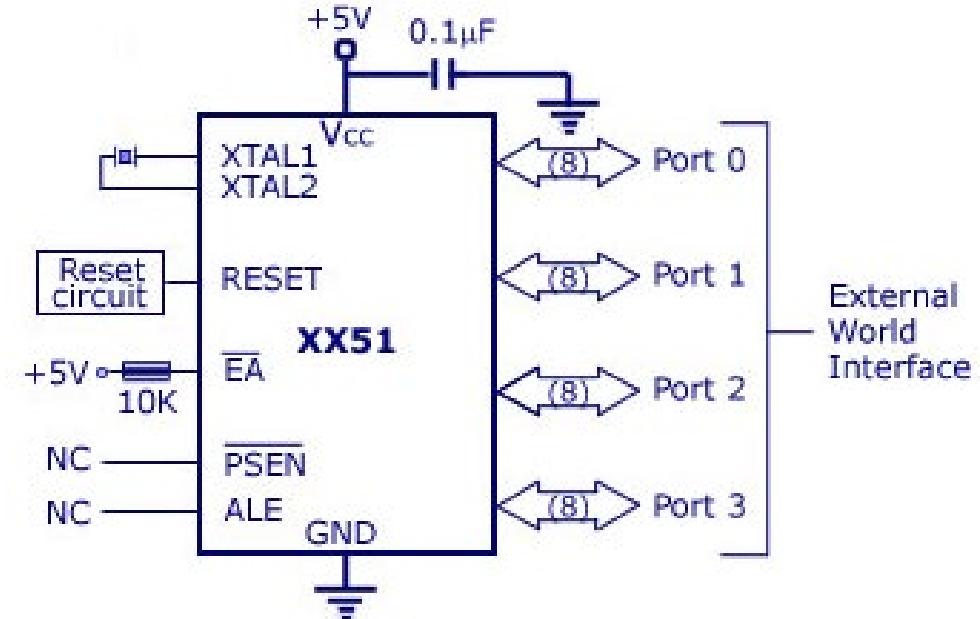
Microcontroller - 8051 - Control Signals

- 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

		Vcc	40
1	P1.0	[AD0]P0.0	39
2	P1.1	[AD1]P0.1	38
3	P1.2	[AD2]P0.2	37
4	P1.3	[AD3]P0.3	36
5	P1.4	[AD4]P0.4	35
6	P1.5	[AD5]P0.5	34
7	P1.6	[AD6]P0.6	33
8	P1.7	[AD7]P0.7	32
9	Reset	[VPP]EA	31
10	P3.0[RxD]	[PROG]ALE	30
11	P3.1[TxD]	PSEN	29
12	P3.2[INT0]	[A15]P2.7	28
13	P3.3[INT1]	[A14]P2.6	27
14	P3.4[T0]	[A13]P2.5	26
15	P3.5[T1]	[A12]P2.4	25
16	P3.6[WR]	[A11]P2.3	24
17	P3.7[RD]	[A10]P2.2	23
18	XTAL2	[A9]P2.1	22
19	XTAL1	[A8]P2.0	21
20	Vss		

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

1	P1.0	Vcc	40
2	P1.1	[AD0]P0.0	39
3	P1.2	[AD1]P0.1	38
4	P1.3	[AD2]P0.2	37
5	P1.4	[AD3]P0.3	36
6	P1.5	[AD4]P0.4	35
7	P1.6	[AD5]P0.5	34
8	P1.7	[AD6]P0.6	33
9	Reset	[AD7]P0.7	32
10	P3.0[RxD]	[VPP]EA	31
11	P3.1[TxD]	[PROG]ALE	30
12	P3.2[INT0]	PSEN	29
13	P3.3[INT1]	[A15]P2.7	28
14	P3.4[T0]	[A14]P2.6	27
15	P3.5[T1]	[A13]P2.5	26
16	P3.6[WR]	[A12]P2.4	25
17	P3.7[RD]	[A11]P2.3	24
18	XTAL2	[A10]P2.2	23
19	XTAL1	[A9]P2.1	22
20	Vss	[A8]P2.0	21

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**: For interfacing an external crystal to provide system clock

		Vcc	40
1	P1.0	[AD0]P0.0	39
2	P1.1	[AD1]P0.1	38
3	P1.2	[AD2]P0.2	37
4	P1.3	[AD3]P0.3	36
5	P1.4	[AD4]P0.4	35
6	P1.5	[AD5]P0.5	34
7	P1.6	[AD6]P0.6	33
8	P1.7	[AD7]P0.7	32
9	Reset	[VPP]EA	31
10	P3.0[RxD]	[PROG]ALE	30
11	P3.1[TxD]	PSEN	29
12	P3.2[INT0]	[A15]P2.7	28
13	P3.3[INT1]	[A14]P2.6	27
14	P3.4[T0]	[A13]P2.5	26
15	P3.5[T1]	[A12]P2.4	25
16	P3.6[WR]	[A11]P2.3	24
17	P3.7[RD]	[A10]P2.2	23
18	XTAL2	[A9]P2.1	22
19	XTAL1	[A8]P2.0	21
20	Vss		

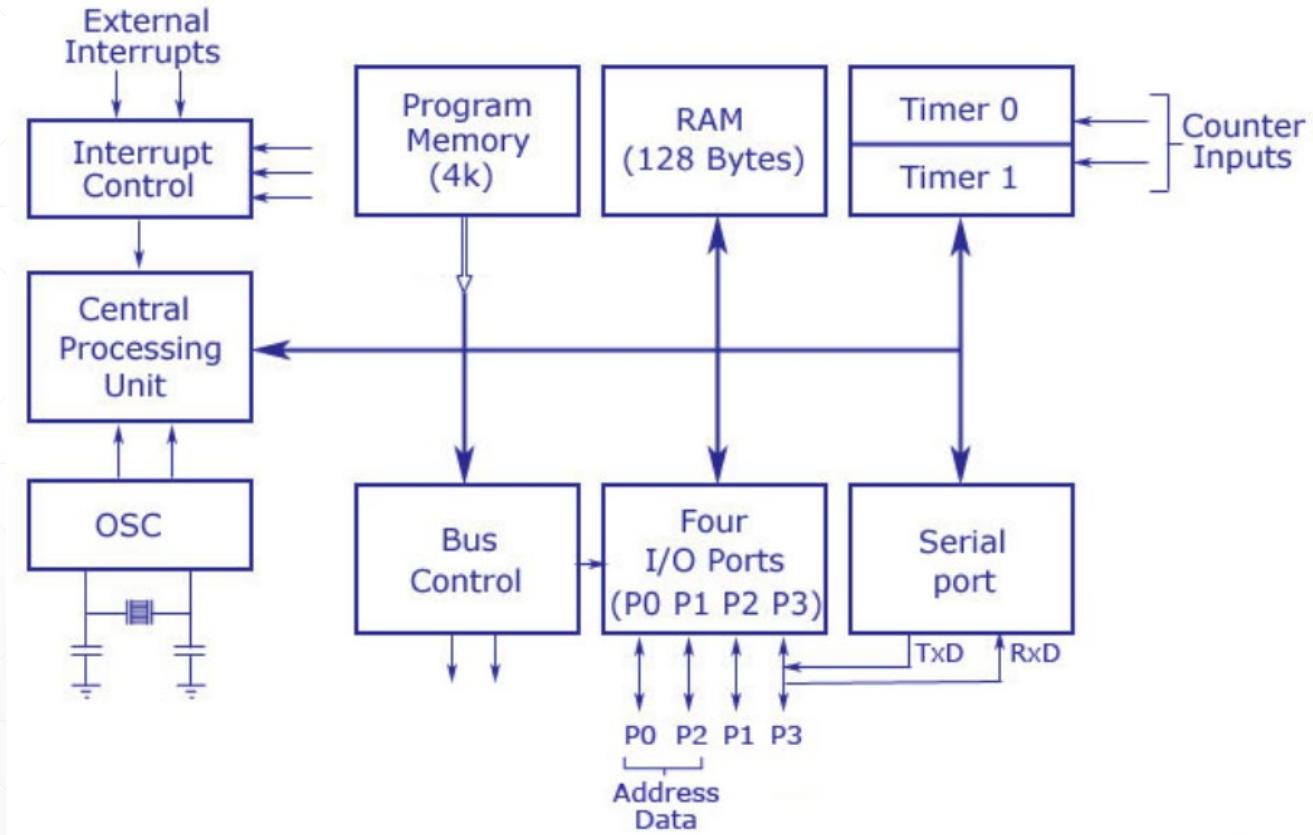
Microcontroller – 8051 – Ports and Pins

- 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

			Vcc
1	P1.0	[AD0]P0.0	40
2	P1.1	[AD1]P0.1	39
3	P1.2	[AD2]P0.2	38
4	P1.3	[AD3]P0.3	37
5	P1.4	[AD4]P0.4	36
6	P1.5	[AD5]P0.5	35
7	P1.6	[AD6]P0.6	34
8	P1.7	[AD7]P0.7	33
9	Reset	[VPP]EA	32
10	P3.0[RxD]	[PROG]ALE	31
11	P3.1[TxD]	PSEN	30
12	P3.2[INT0]	[A15]P2.7	29
13	P3.3[INT1]	[A14]P2.6	28
14	P3.4[T0]	[A13]P2.5	27
15	P3.5[T1]	[A12]P2.4	26
16	P3.6[WR]	[A11]P2.3	25
17	P3.7[RD]	[A10]P2.2	24
18	XTAL2	[A9]P2.1	23
19	XTAL1	[A8]P2.0	22
20	Vss		21

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

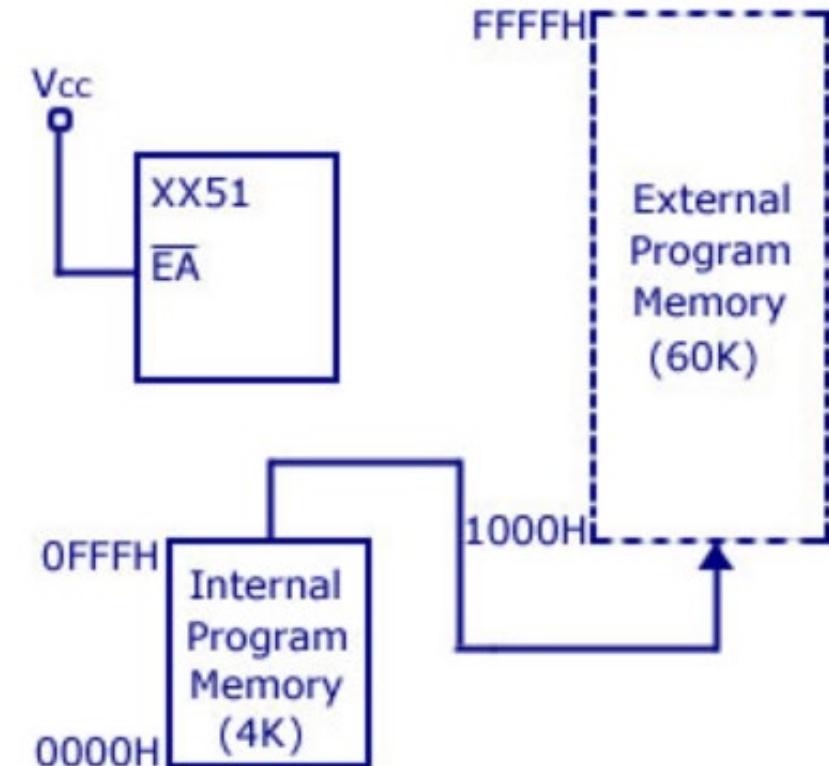
Microcontroller – 8051 – Program Memory

- 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 - FFFFFF
16M	1000000	0 - FFFFFFFF

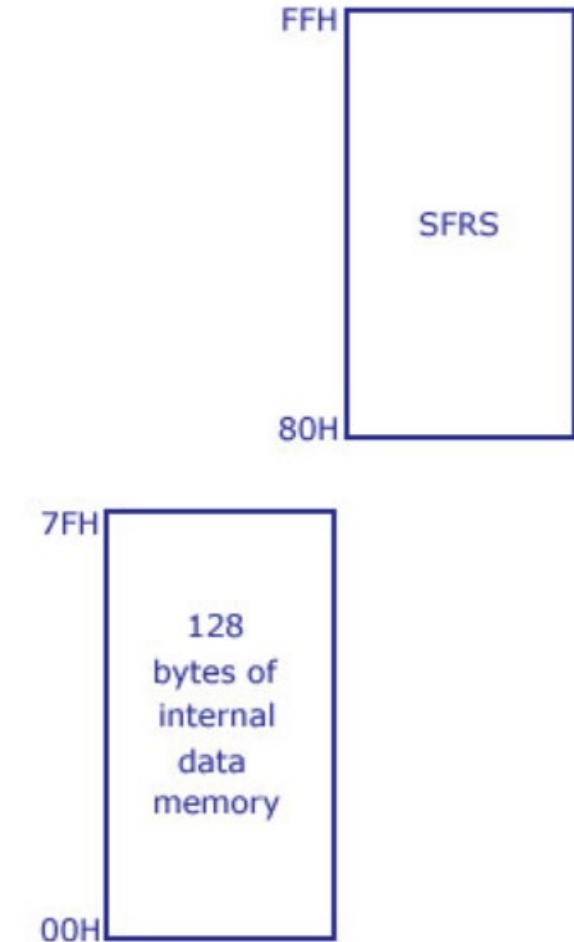
Microcontroller – 8051 – Program Memory

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



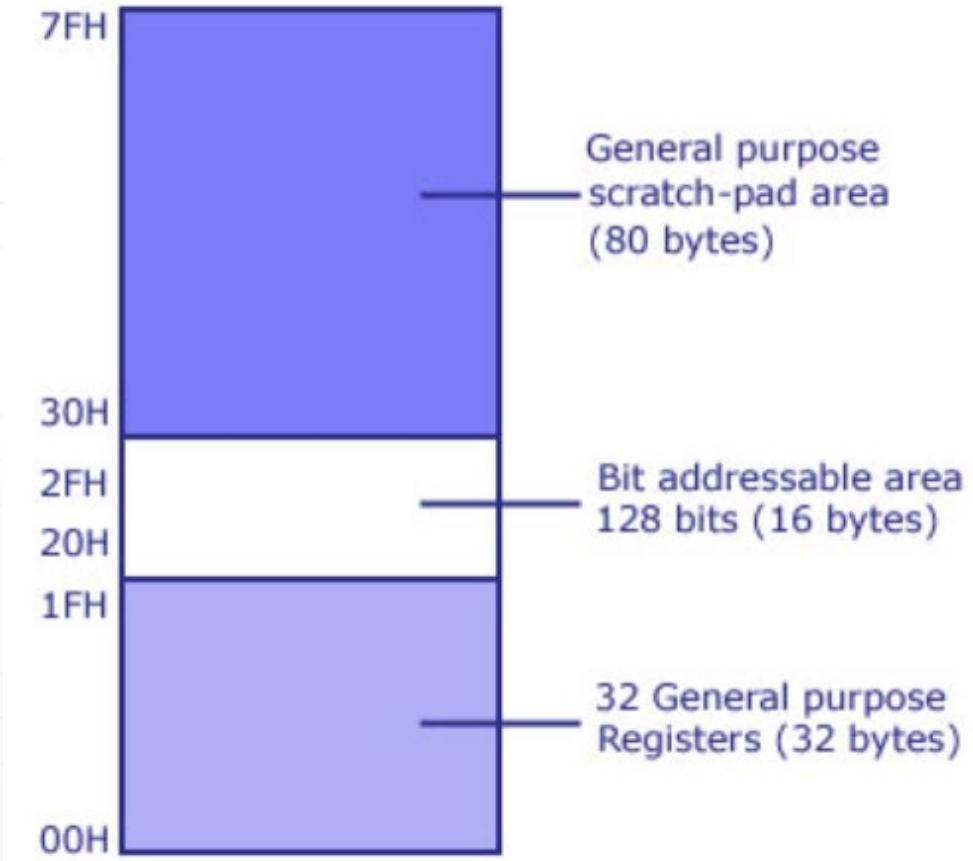
Microcontroller – 8051 – Data Memory

- 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)



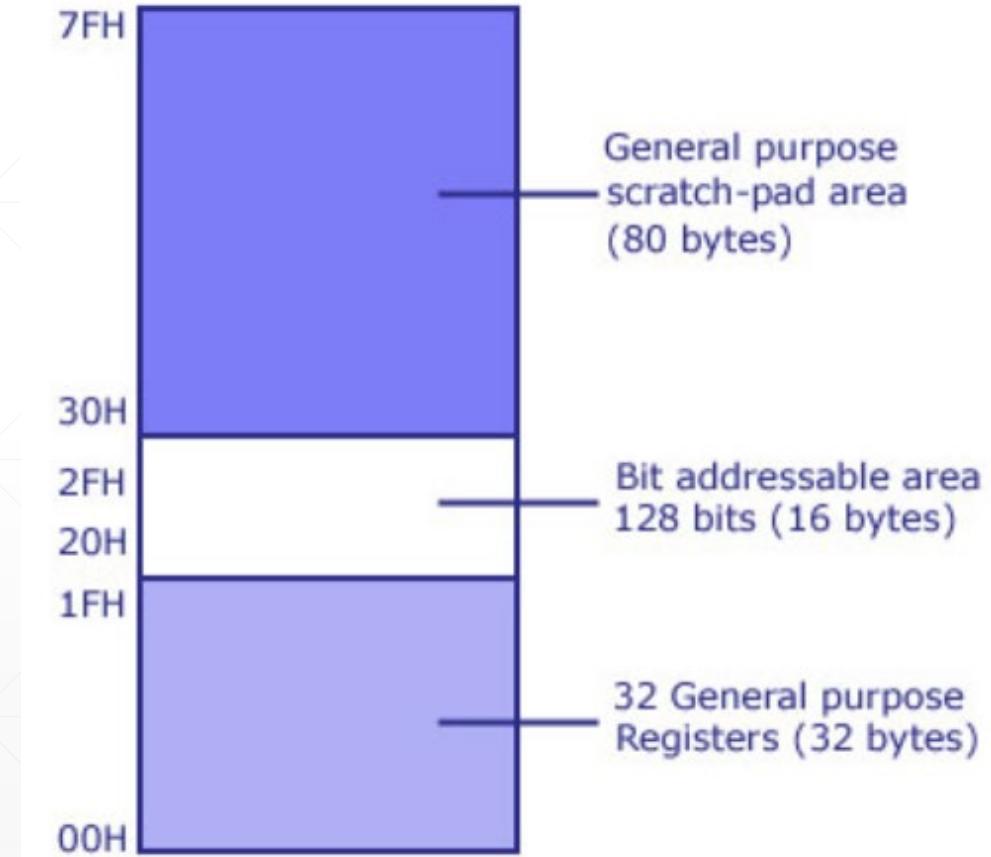
Microcontroller – 8051 – Data Memory

- 8051 – Internal Data Memory – 128 bytes
- 3 separations/divisions of the internal data memory:
 - Register banks
 - Bit addressable area
 - Scratch pad area.



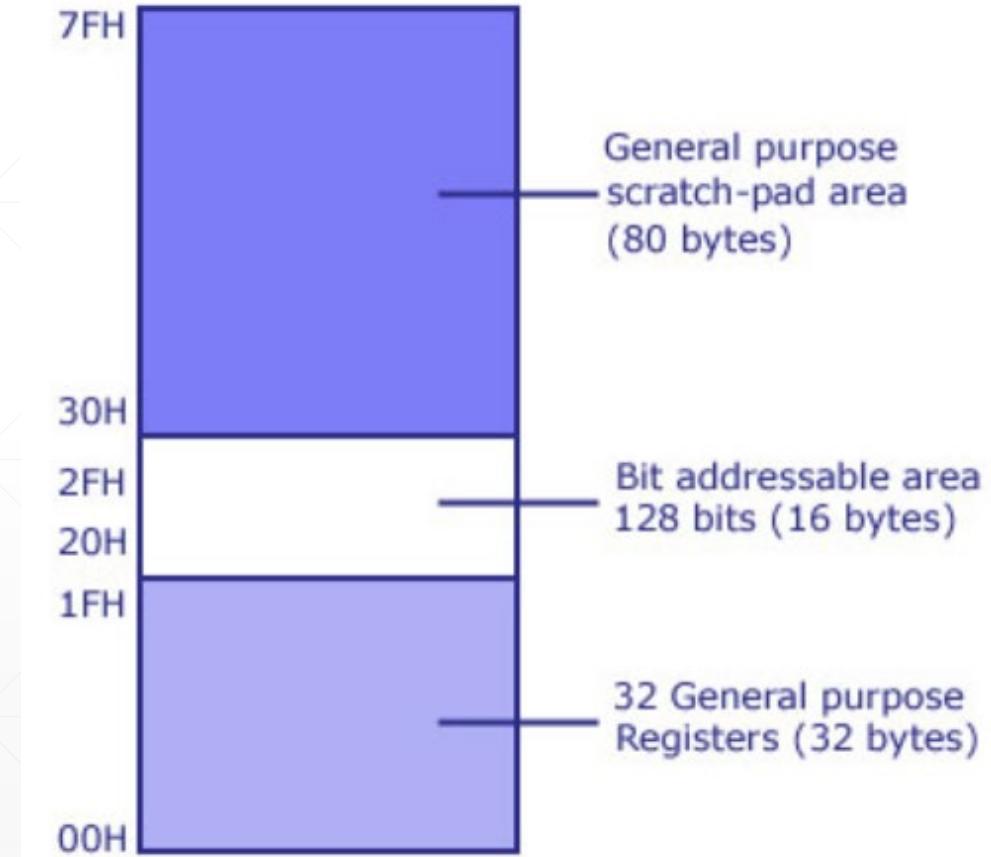
Microcontroller – 8051 – Data Memory

- 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



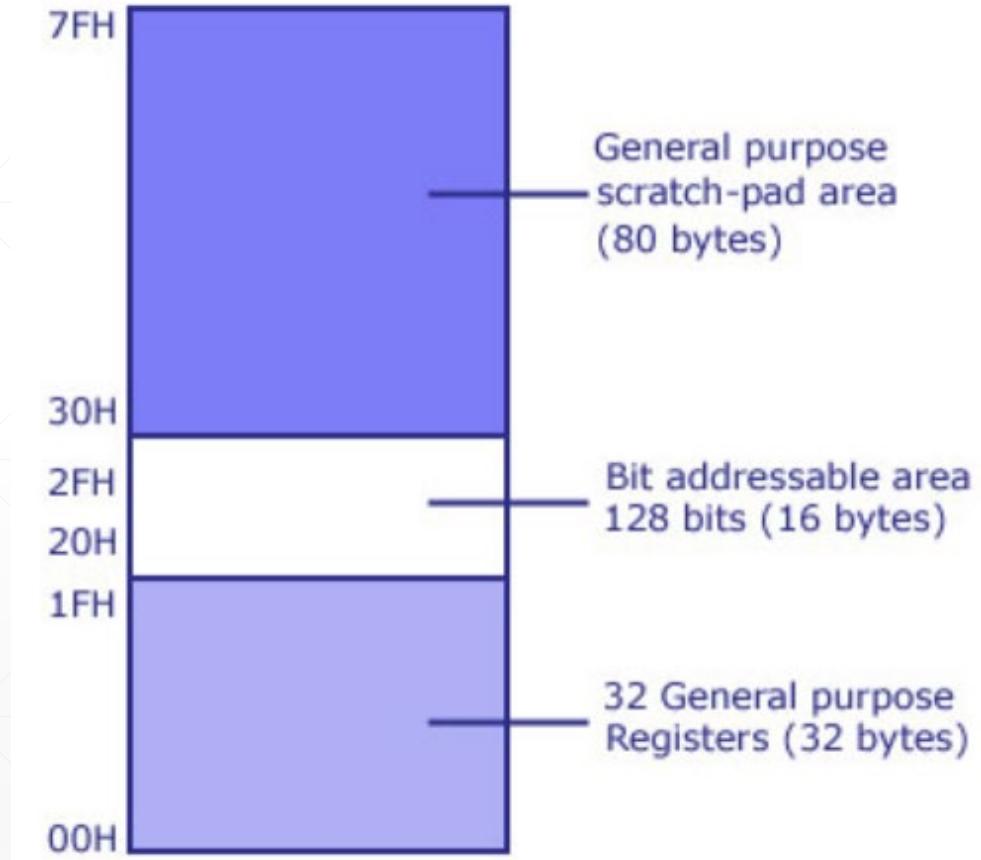
Microcontroller – 8051 – Data Memory

- 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.



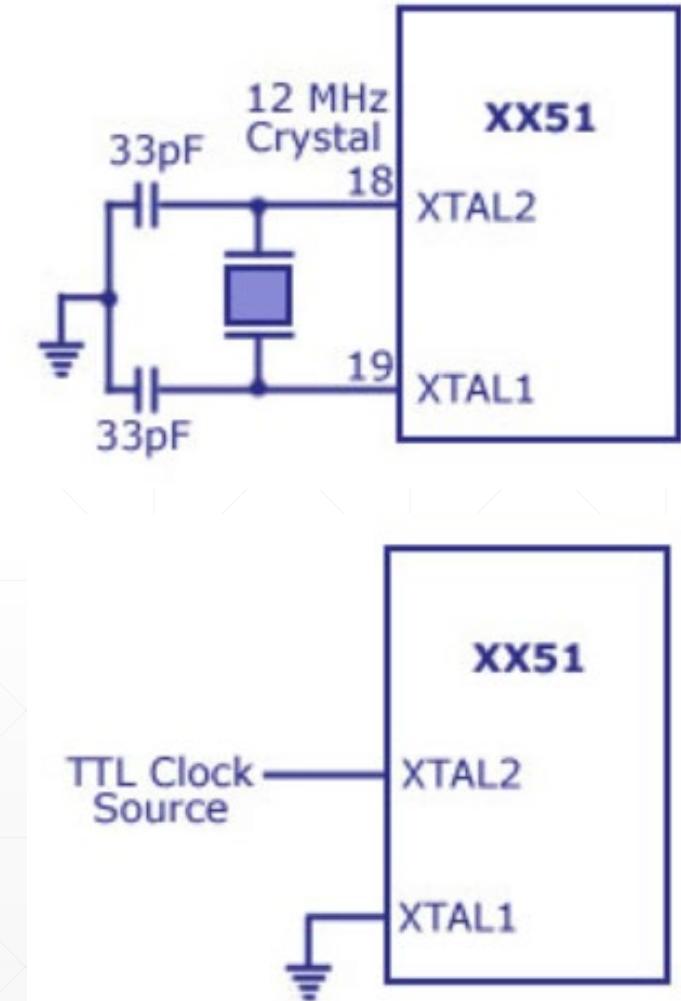
Microcontroller – 8051 – Data Memory

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



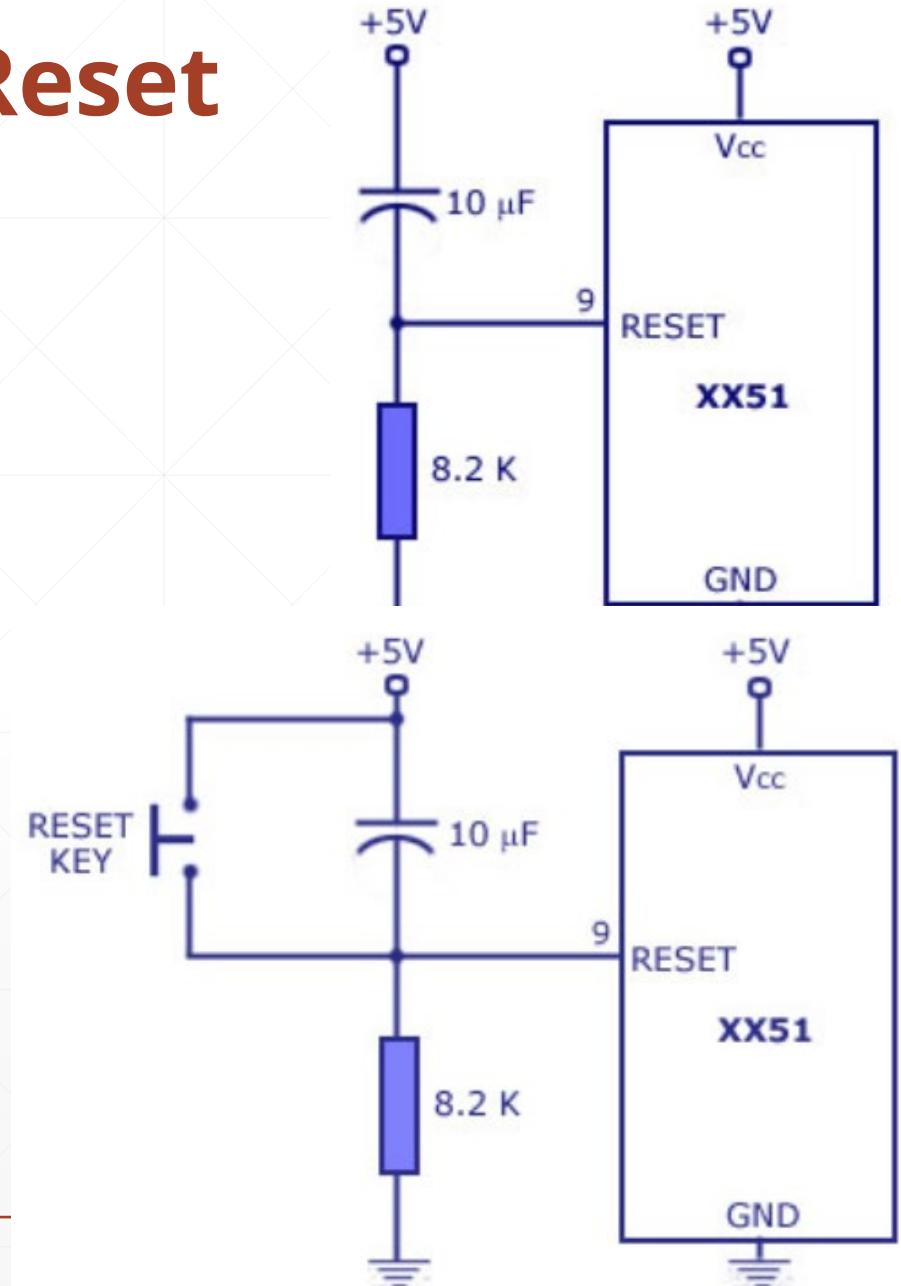
Microcontroller – 8051 – Clock

- 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



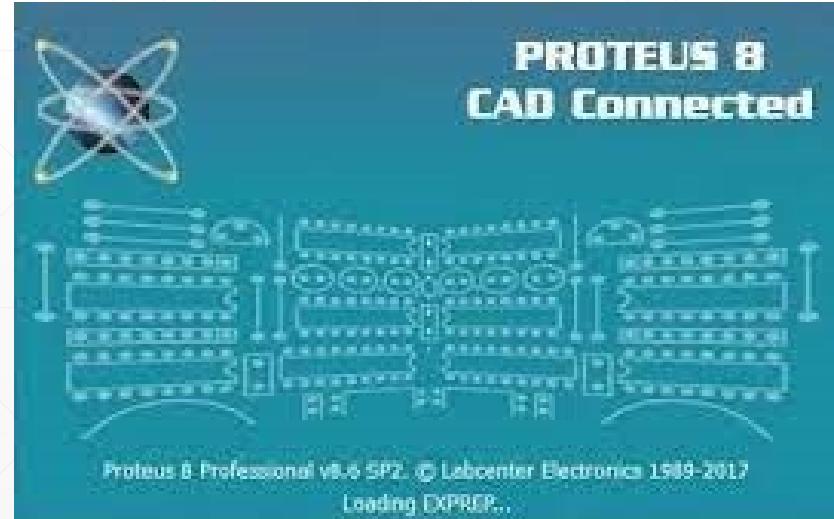
Microcontroller – 8051 – Reset

- 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 or 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

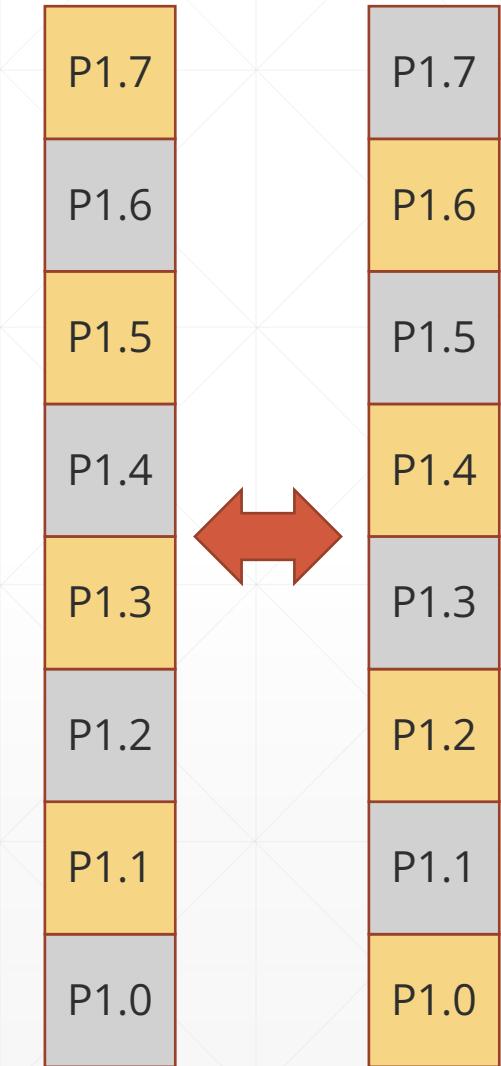
8051 PROGRAMMING IN C – Data Types

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

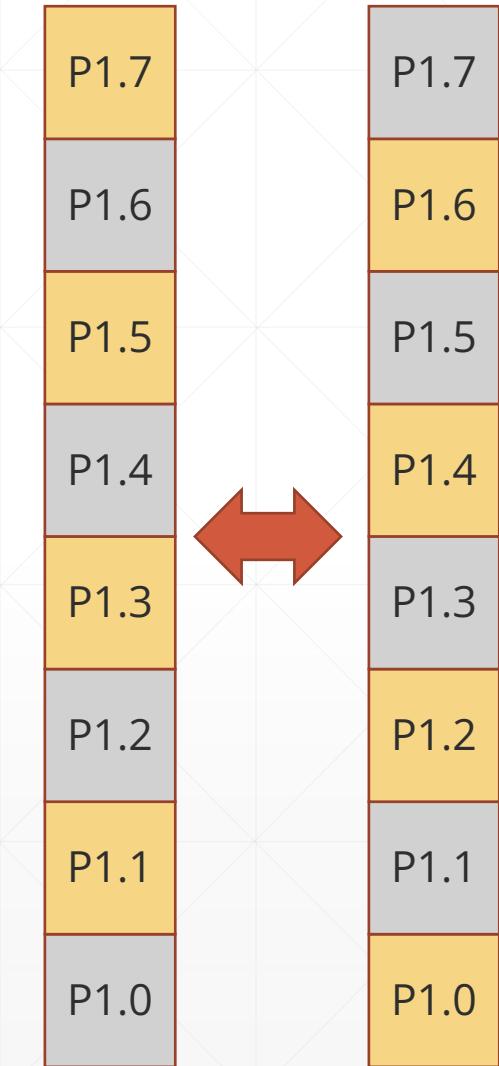
Write an 8051 C program to toggle all the bits of P1 continuously.



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

$$\text{Cycle} = 12 / 12 \text{ Mhz} = 1\mu\text{s}$$

$$Y * 1\mu\text{s} = 100 \text{ ms}$$

$$Y = 100 \text{ ms} / 1\mu\text{s}$$

$$Y = 10000$$

```
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 from 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

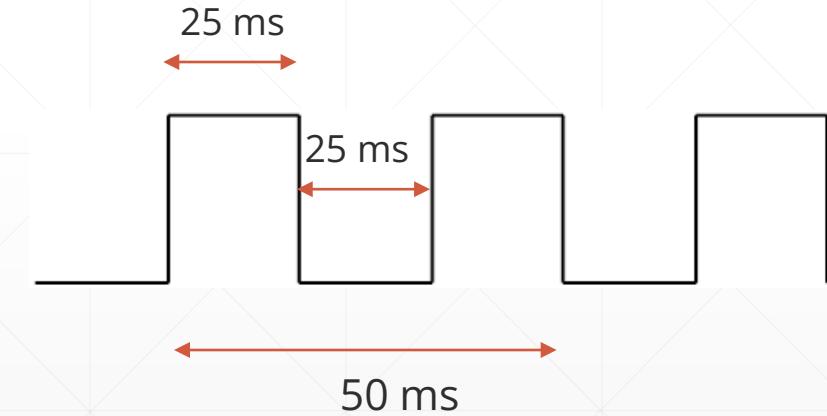
Frequency = 20Hz, Time = $1/20 = 50\text{ms}$

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

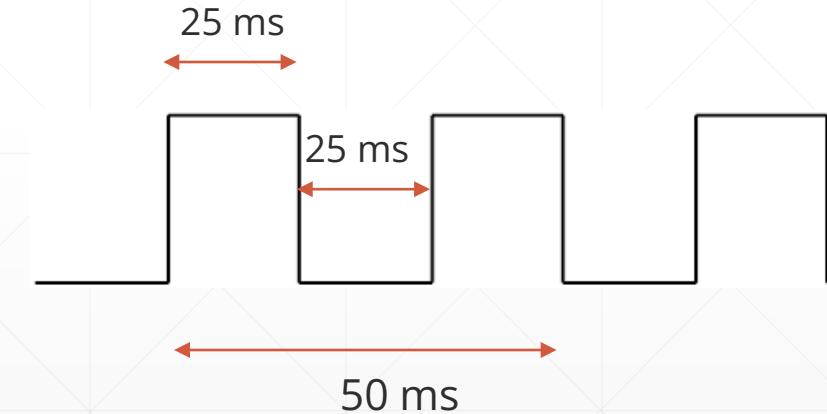
Timer 0, Mode 1

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 = $12/11.0592\text{MHz}$

Time for 1 instru = $1.085 \mu\text{s}$

Delay of 25ms = $25/1.085 \mu\text{s} = 23037$

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

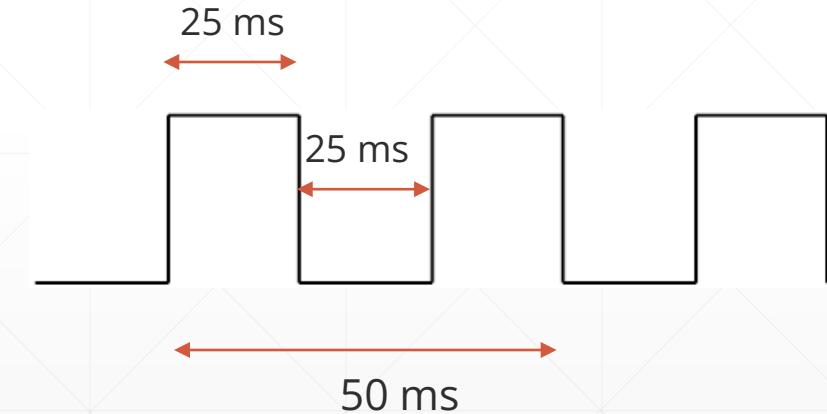
Frequency = 20Hz, Time = $1/20 = 50\text{ms}$

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

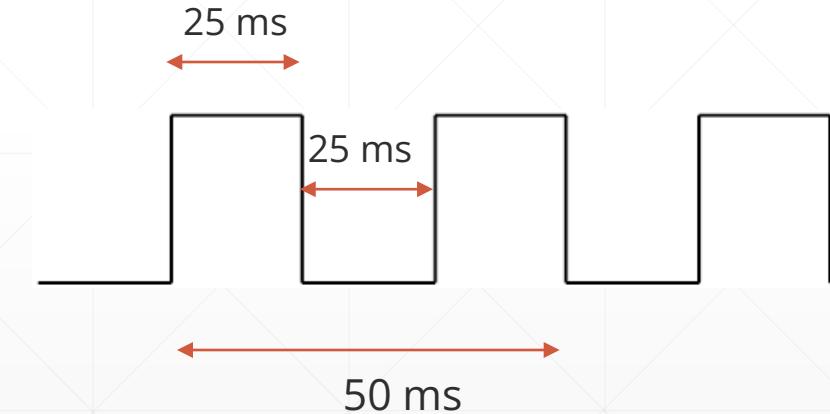
Timer 0, Mode 1

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 = $12/11.0592\text{MHz}$

Time for 1 instru = $1.085 \mu\text{s}$

Delay of 25ms = $25/1.085 \mu\text{s} = 23037$

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 elsewhere 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, $V_{cc} = 5$ volts
- Step size of n bit ADC =
$$\frac{V_{cc}}{\text{Number of steps}}$$

Analog to Digital Converter (ADC)

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

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

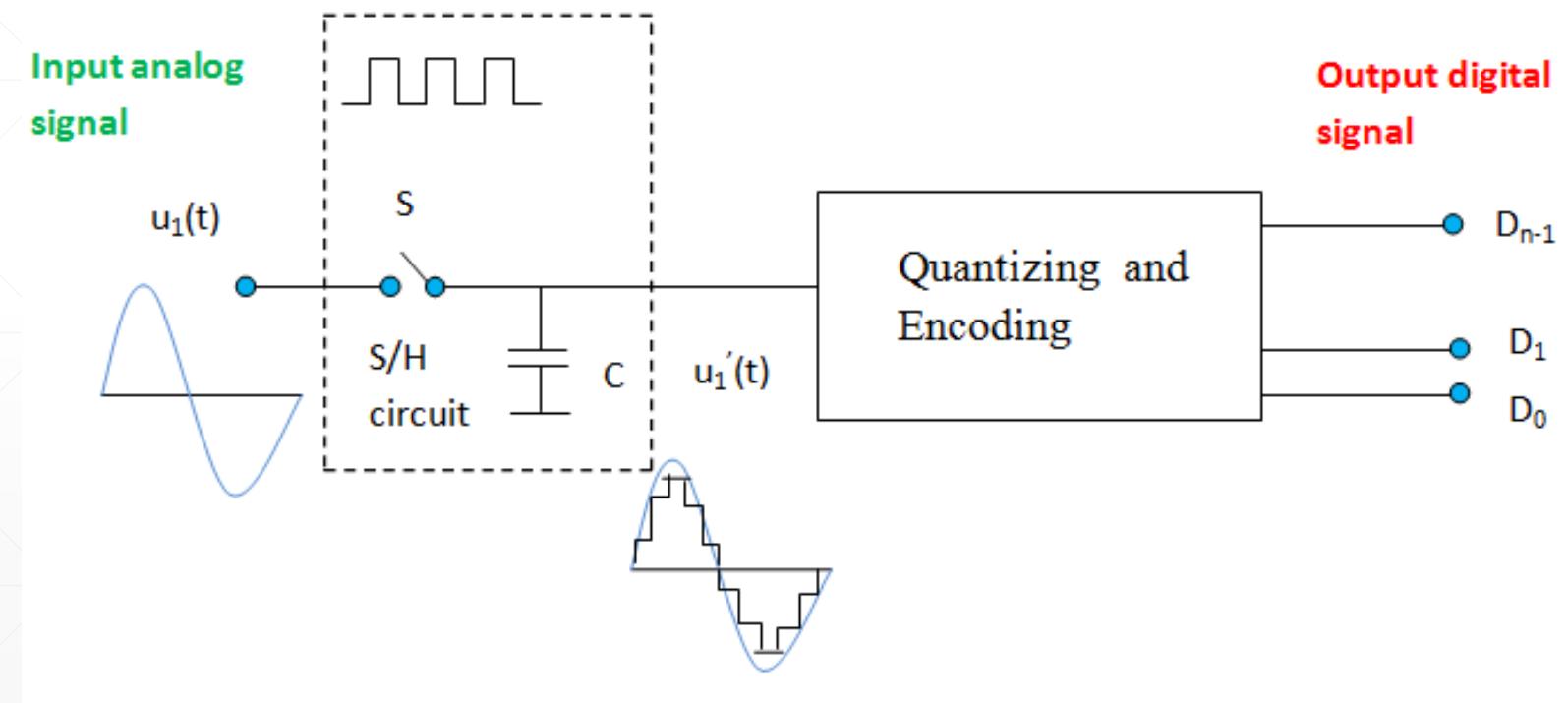
- Step size (resolution) is the smallest change that can be discerned by an ADC

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

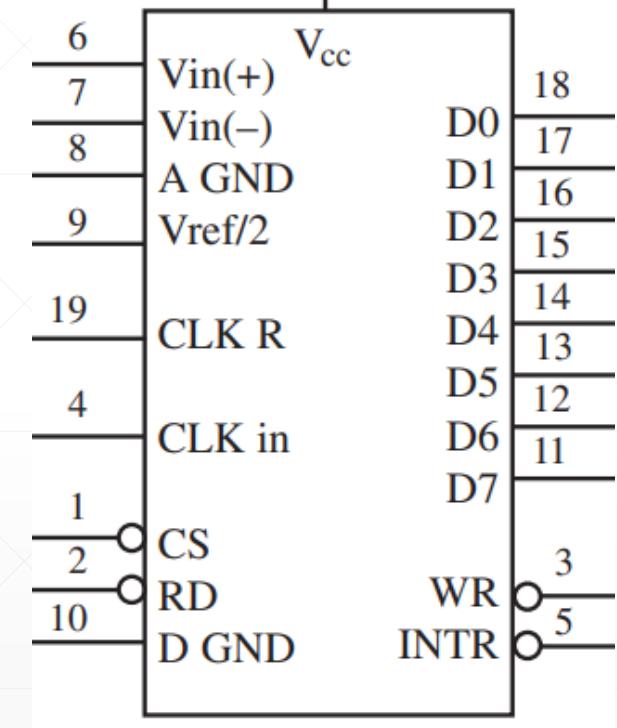
- An 8-bit digital system is used to convert an analog signal to digital signal for a data acquisition system.
- The voltage range for the conversion is 0-10 V.
- Find the resolution of the system and the value of the least significant bit

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

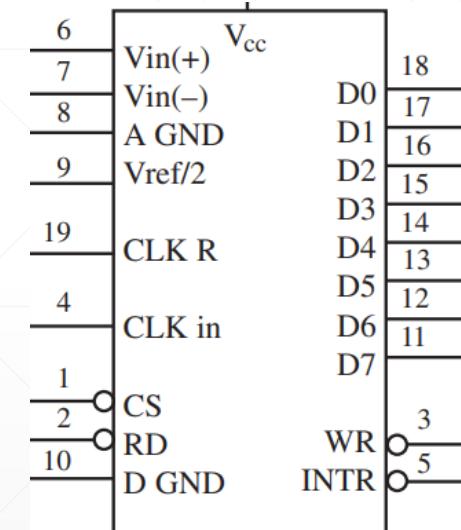
ADC - 0804

- 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 μ s



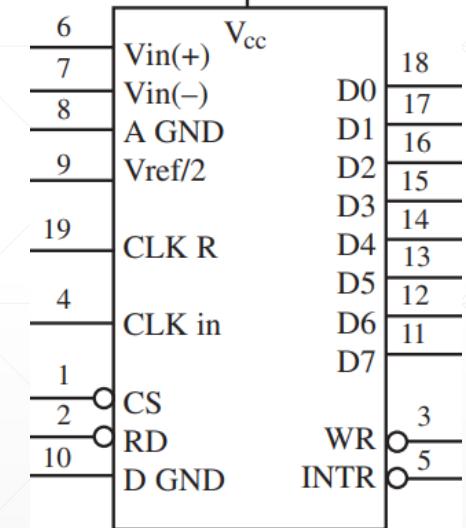
ADC - 0804

- **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

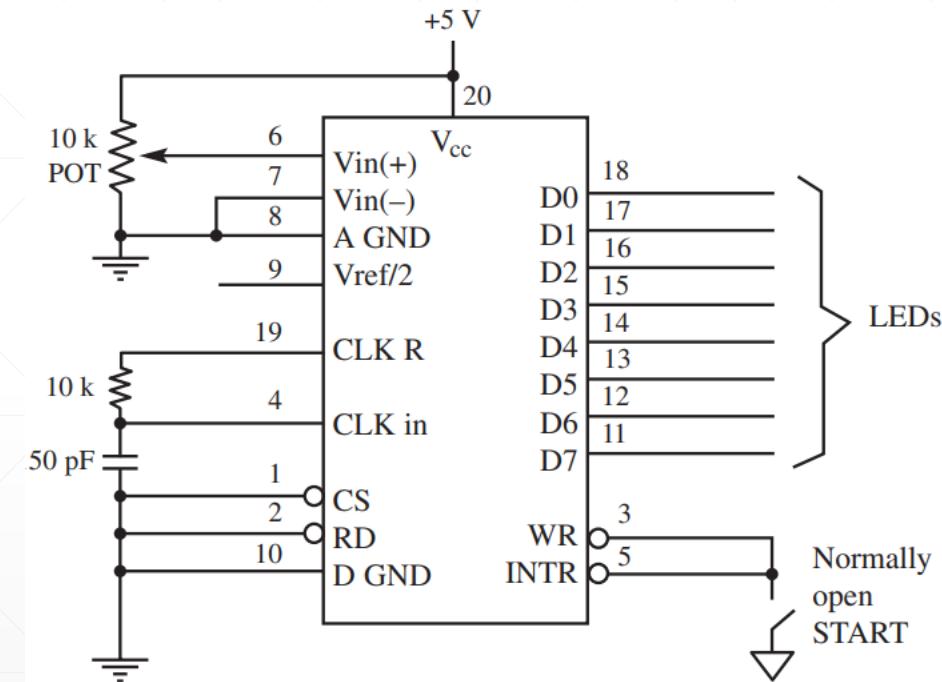
- **WR** – Write – Start of Conversion
 - This is an active-low input used to inform the ADC0804 to start the conversion process
 - If CS = 0 when WR makes a **low-to-high** transition
 - ADC0804 starts converting the analog input value of Vin to an 8-bit digital number
 - The amount of time it takes 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:
$$f = \frac{1}{1.1 RC}$$



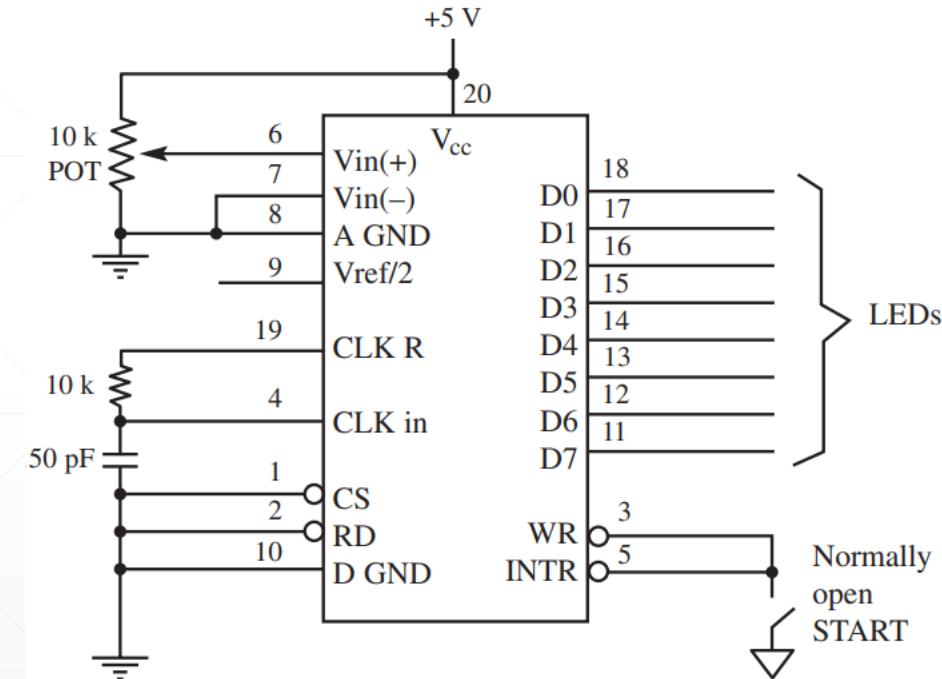
ADC - 0804

- **CLK IN and CLK R**

- In that case, the clock frequency is determined by the equation:

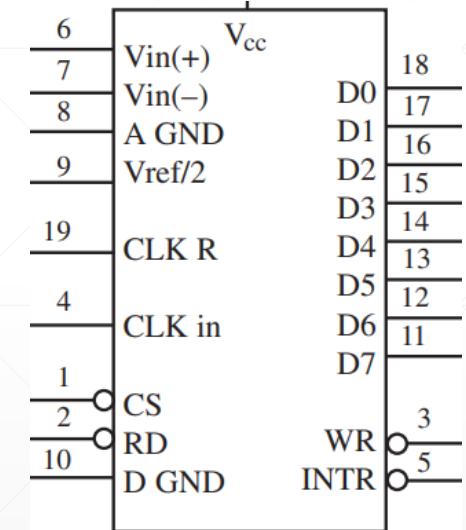
- $f = \frac{1}{1.1 RC}$

- Typical values are $R = 10K$ ohms and $C = 150\text{ pF}$
- $f = 606\text{ kHz}$
- In that case, the conversion time is $110\text{ }\mu\text{s}$



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



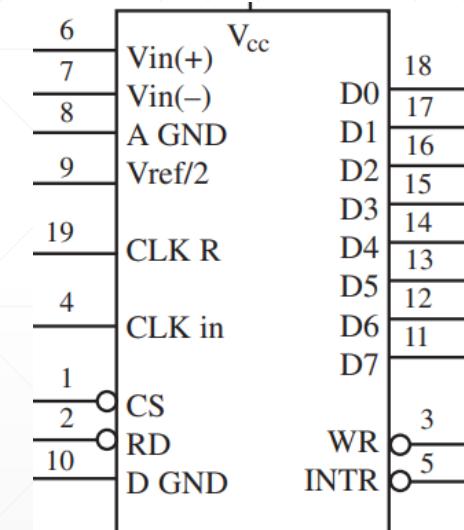
ADC - 0804

- **Vin (+) and Vin (-)**

- These are the differential analog inputs
- $V_{in} = V_{in\ (+)} - V_{in\ (-)}$
- Often the $V_{in\ (-)}$ pin is connected to ground
- $V_{in\ (+)}$ pin is used as the analog input to be converted to digital

- **VCC**

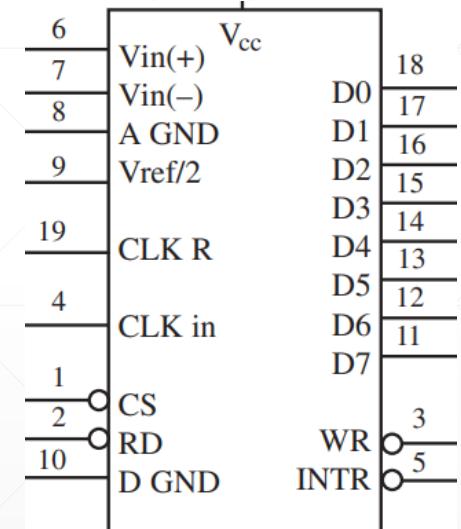
- This is the +5 V power supply
- It is also used as a reference voltage when $V_{ref/2}$ input (pin 9) is open



ADC - 0804

- **Vref/2**

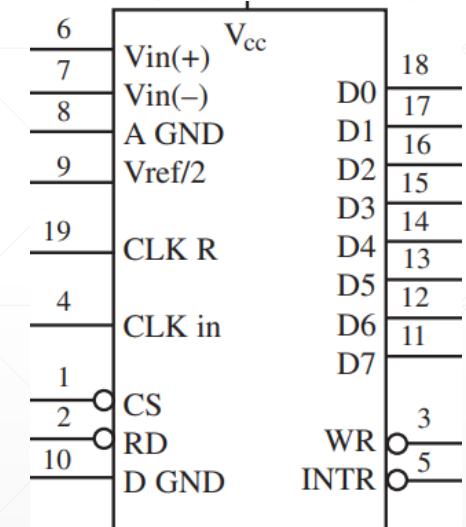
- Pin 9 is an input voltage used for the reference voltage
- If this pin is open (not connected), the analog input voltage for the ADC0804 is in the range of 0 to 5 V
- However, there are many applications where the analog input applied to Vin needs to be other than the 0 to +5 V range
- Vref /2 is used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

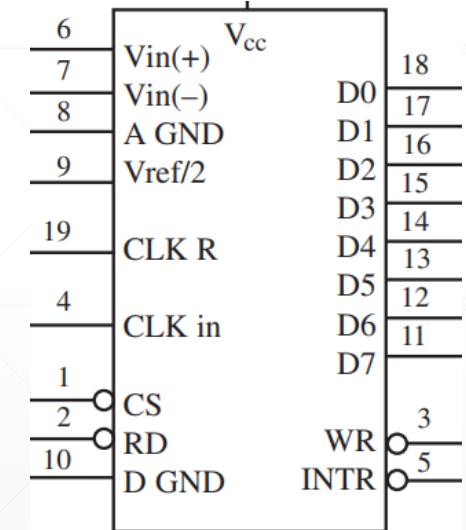
- D0–D7 (D7 is the MSB) are the digital data output pins since ADC0804 is a parallel ADC chip
- The converted data is accessed only when CS = 0 and RD is forced low
- To calculate the output voltage, use the following formula
- $$D_{out} = \frac{Vin}{Step\ Size}$$
- D_{out} = digital data output (in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



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

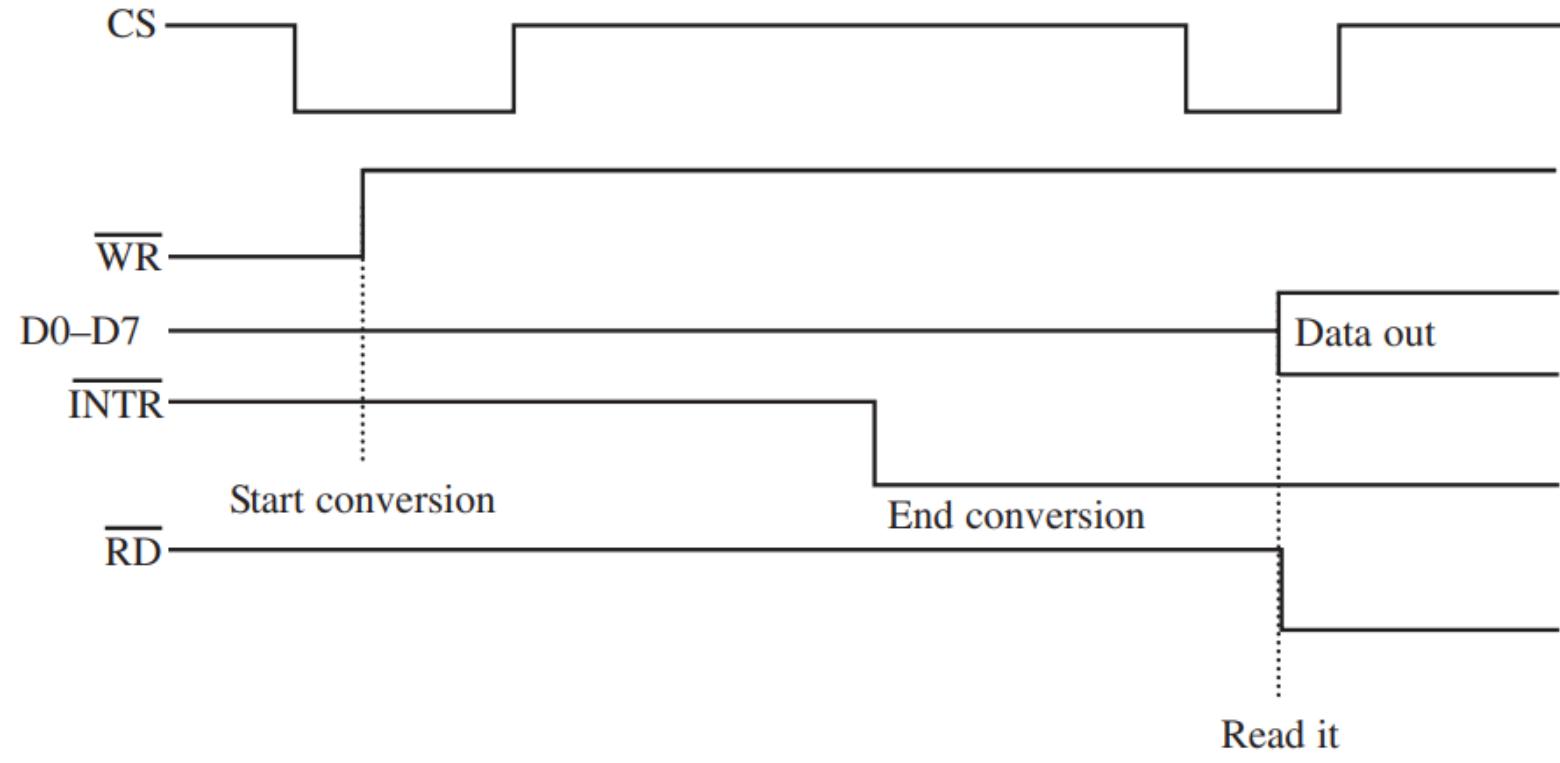


ADC – 0804 – Conversion Steps

- **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-to-low pulse to the RD pin to get the data out of the ADC0804 IC chip

ADC – 0804 – Conversion Steps

- Analog to Digital Conversion ADC0804

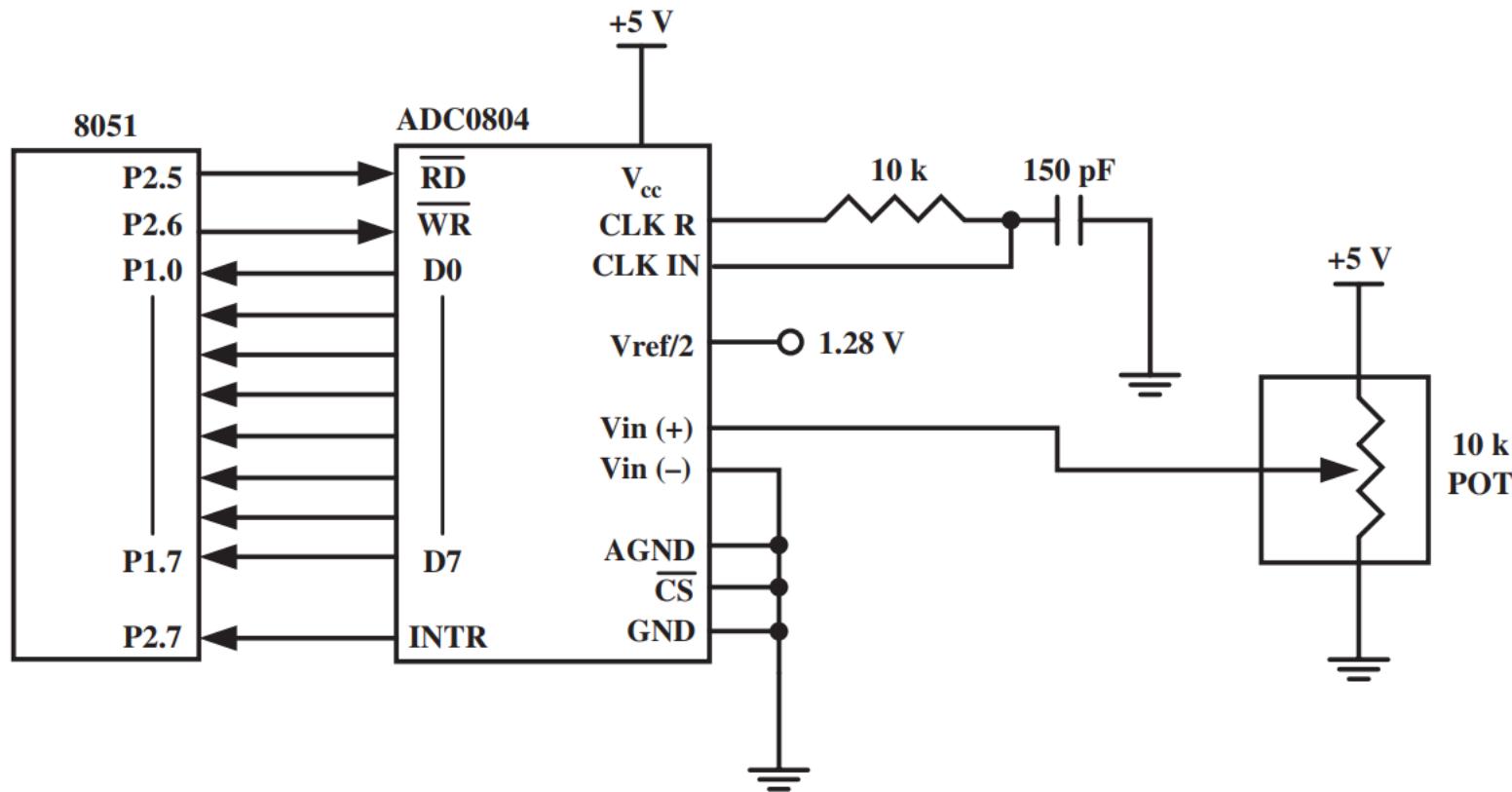


ADC – 0804 – Clock

- **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**

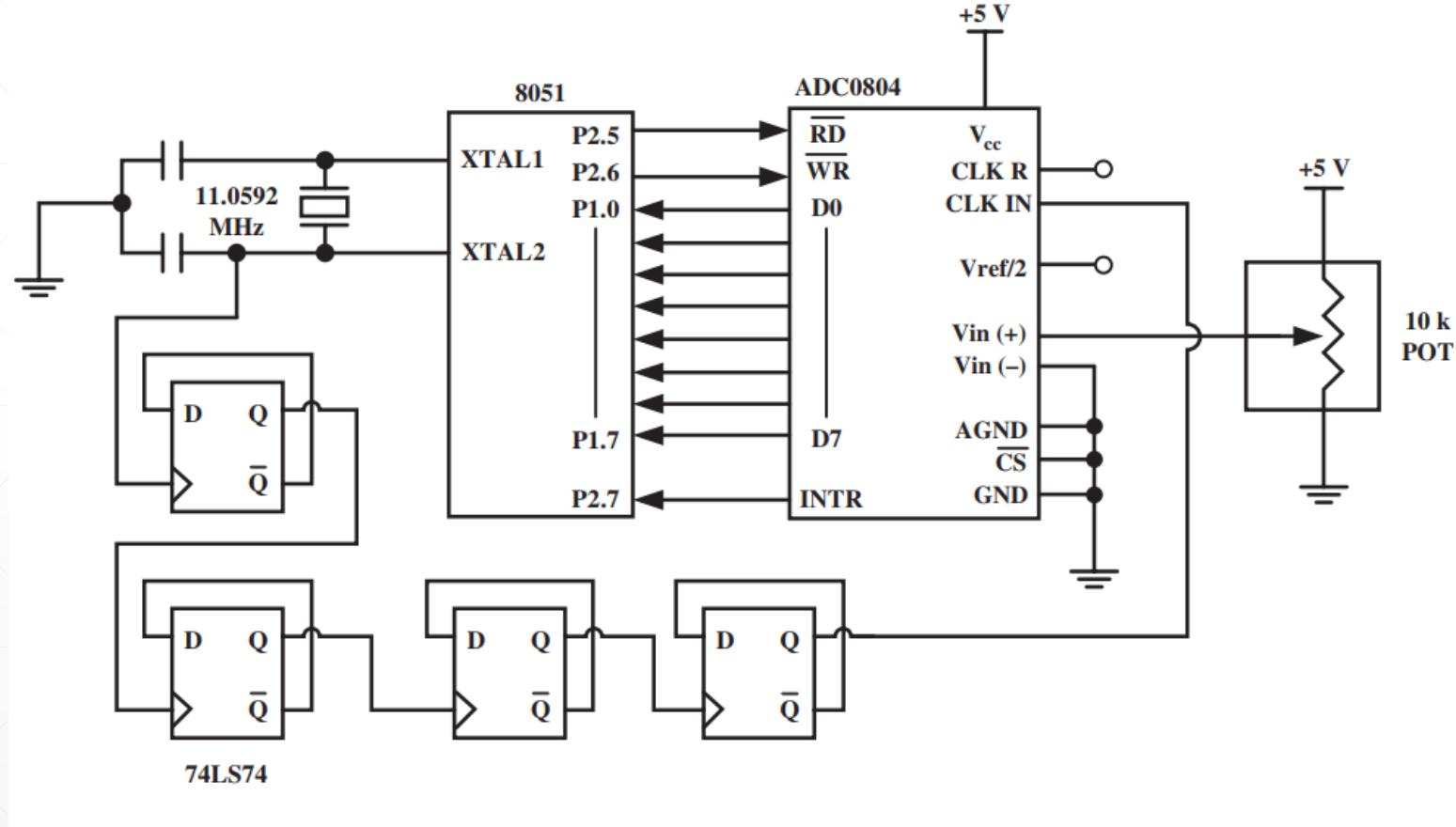
ADC – 0804 – Clock

- Internal clock to the ADC0804



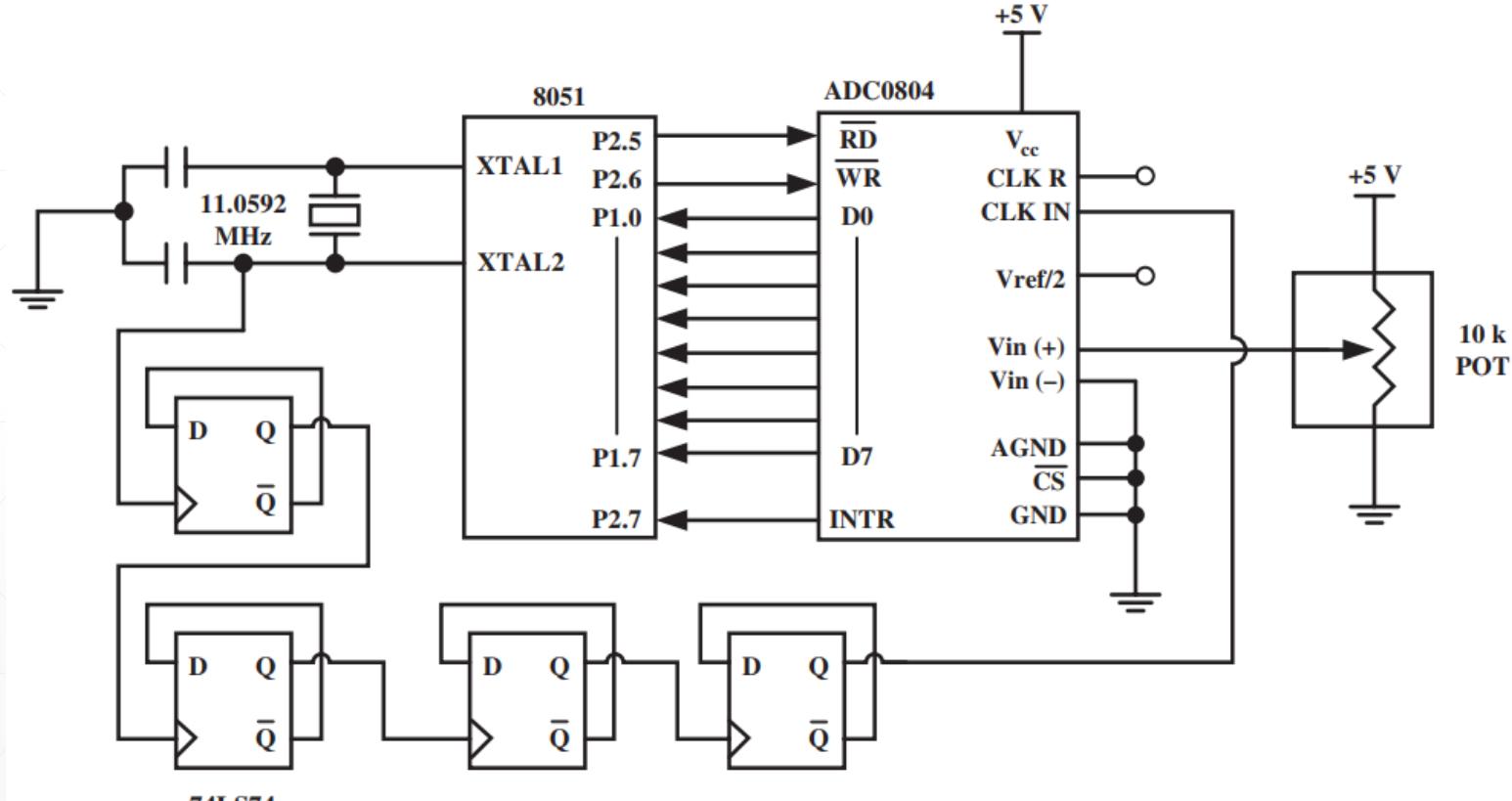
ADC - 0804 - Clock

- External clock to the ADC0804



ADC - 0804 - Clock

- External clock to the ADC0804



ADC – 0804 – Programming

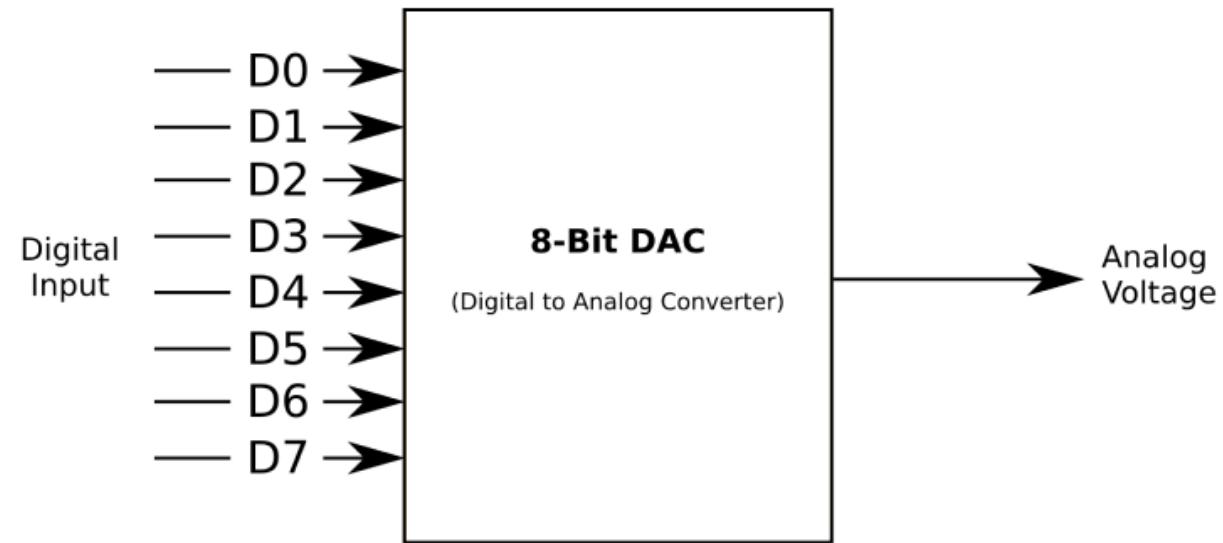
```
#include <reg51.h>

sbit RD = P2^5;
sbit WR = P2^6;
sbit INTR = P2^7;
sfr MYDATA = P1;

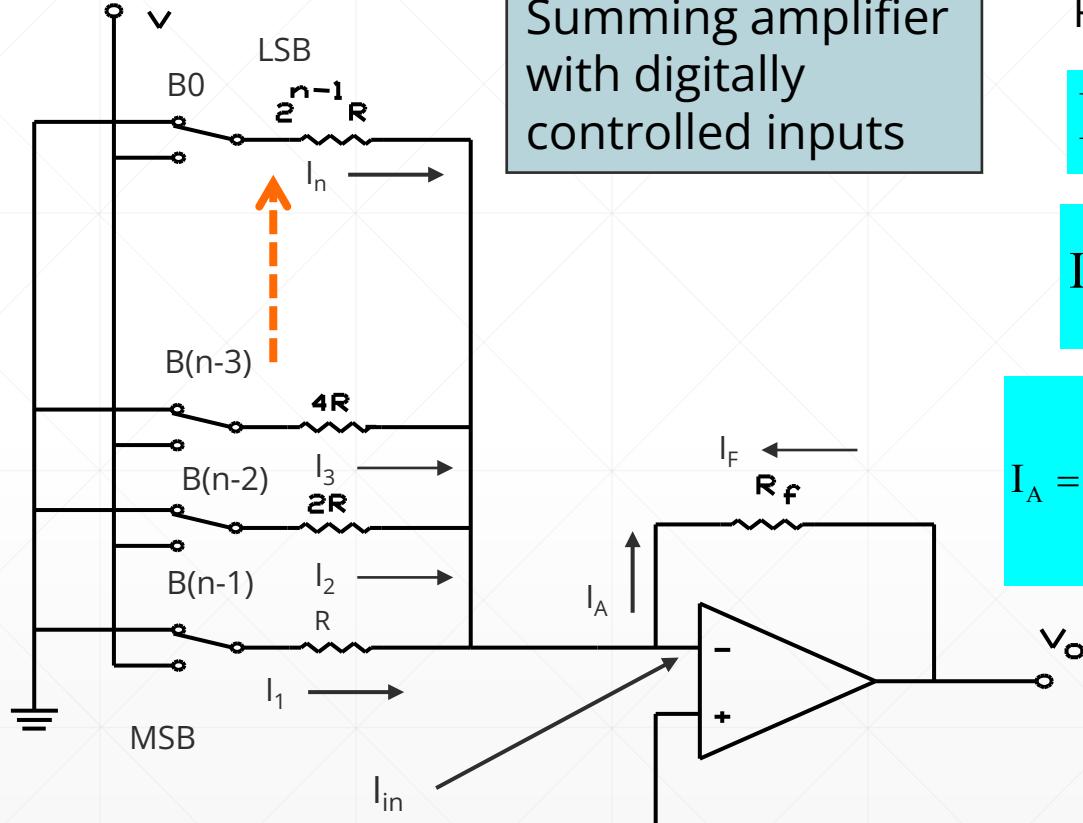
void main() {
    unsigned char value;
    MYDATA = 0xFF; //make P1 as Input
    INTR = 1; //make INTR as Input
    RD = 1; WR = 1; //set RD and WR Input
    while (1){
        WR = 0;
        WR = 1; //send WR L to H pulse
        while (INTR == 1); //wait for EOC
        RD = 0 ; //send RD H to L pulse
        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



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

ECE4010

Rules of Ideal OP AMPS $I_{in} = 0, Z_{in} = \text{infinity}$

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

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

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

Formula for output V

$$V_o = -I_F \cdot R_F$$

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

Binary Weighted resistor - DAC

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

$$V_o = -R_F \sum_{i=1}^n \frac{B(n-i) \cdot V}{2^{i-1} R} \quad n=4$$

$$V_o = -R \cdot \left[\frac{B_3 \cdot V}{2^{1-1} R} + \frac{B_2 \cdot V}{2^{2-1} R} + \frac{B_1 \cdot V}{2^{3-1} R} + \frac{B_0 \cdot V}{2^{4-1} R} \right]$$

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

$$V_o = -\left[1 + \frac{1}{2} + 0 + \frac{1}{8} \right] = -[1 + 50 + 1.2] = -51.2 \text{ V}$$

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

Embedded Systems

ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

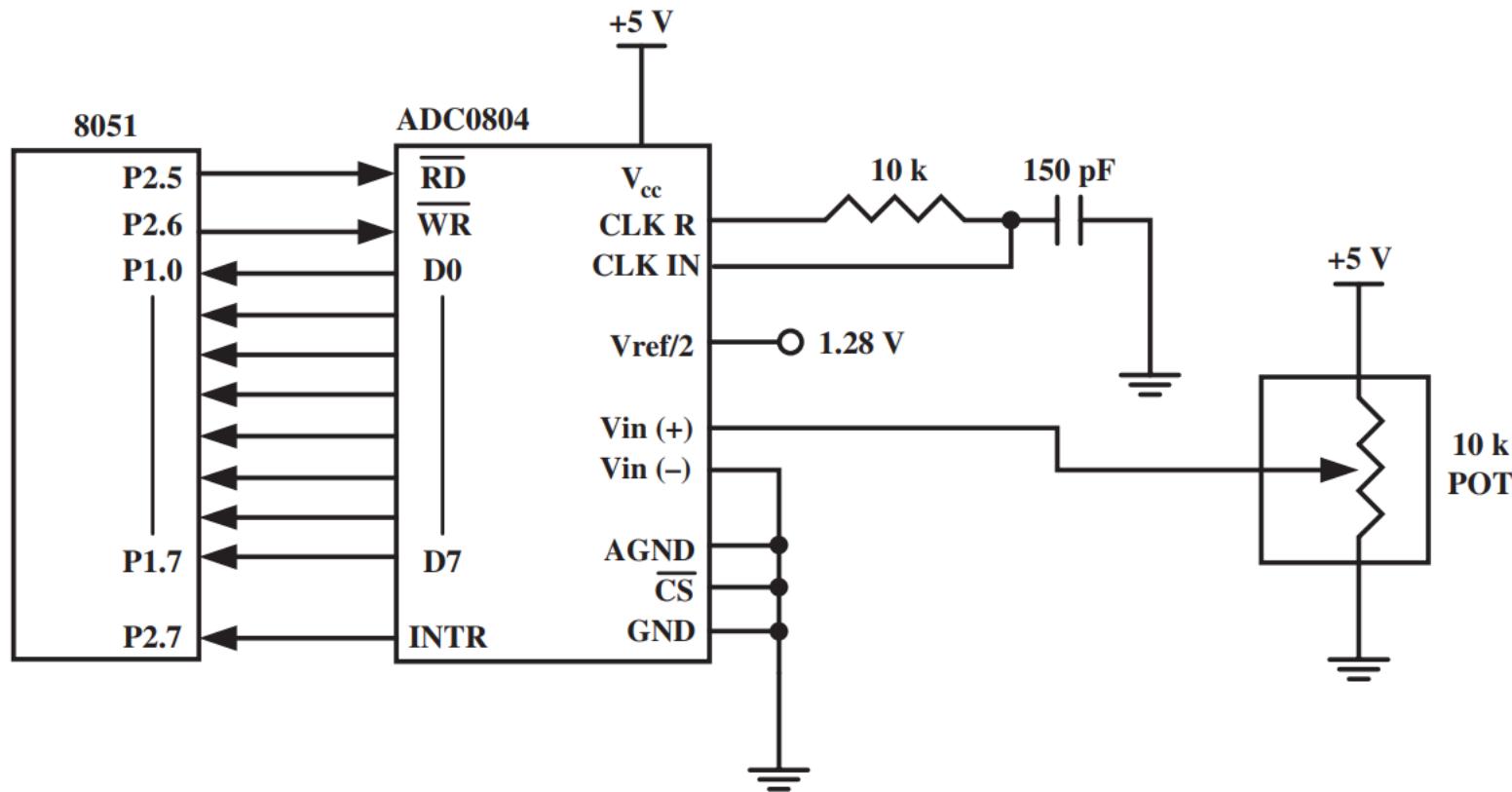
8051 PROGRAMMING IN C

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**

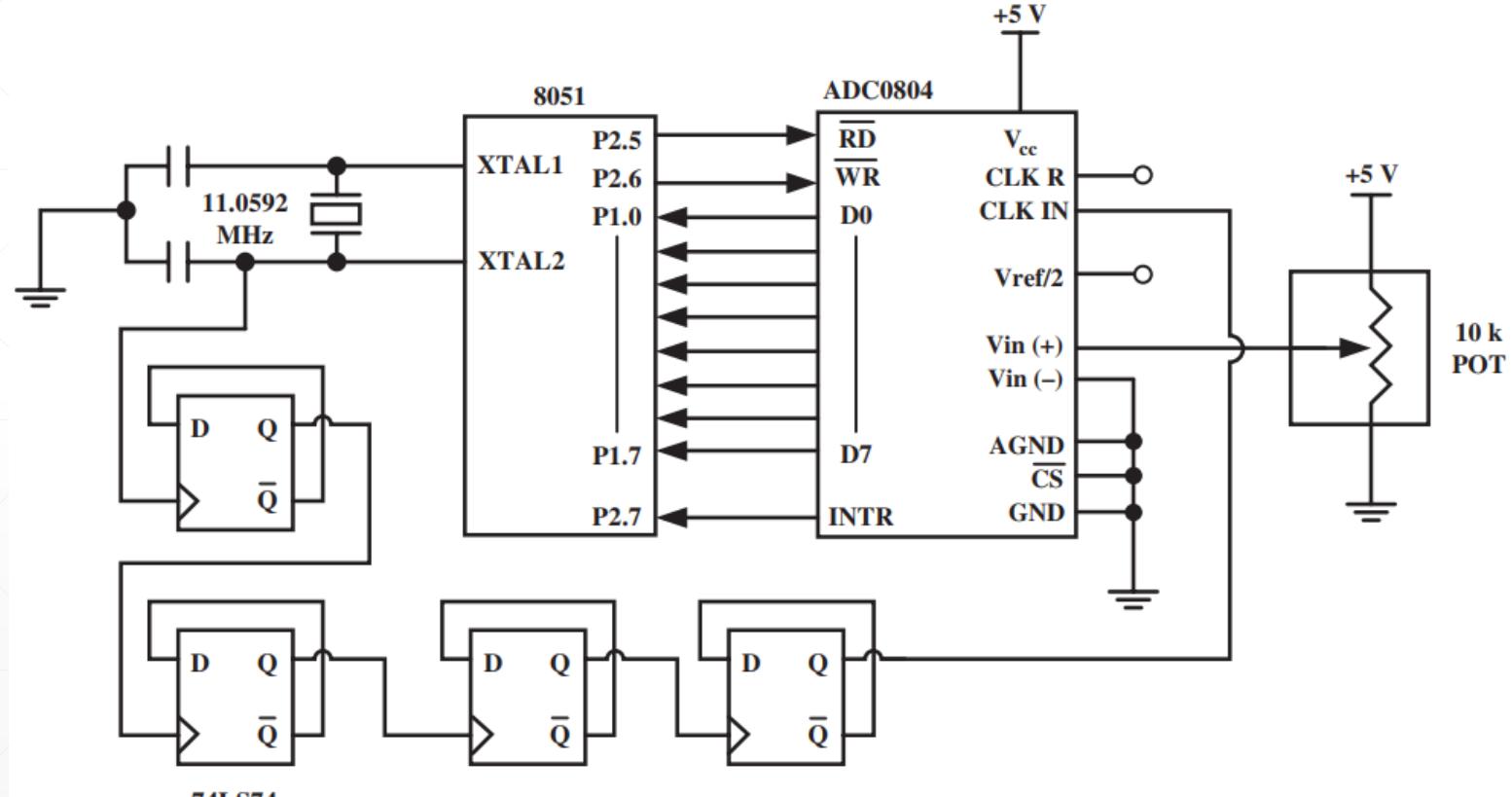
ADC – 0804 – Clock

- Internal clock to the ADC0804



ADC - 0804 - Clock

- External clock to the ADC0804



ADC – 0804 – Programming

```
#include <reg51.h>

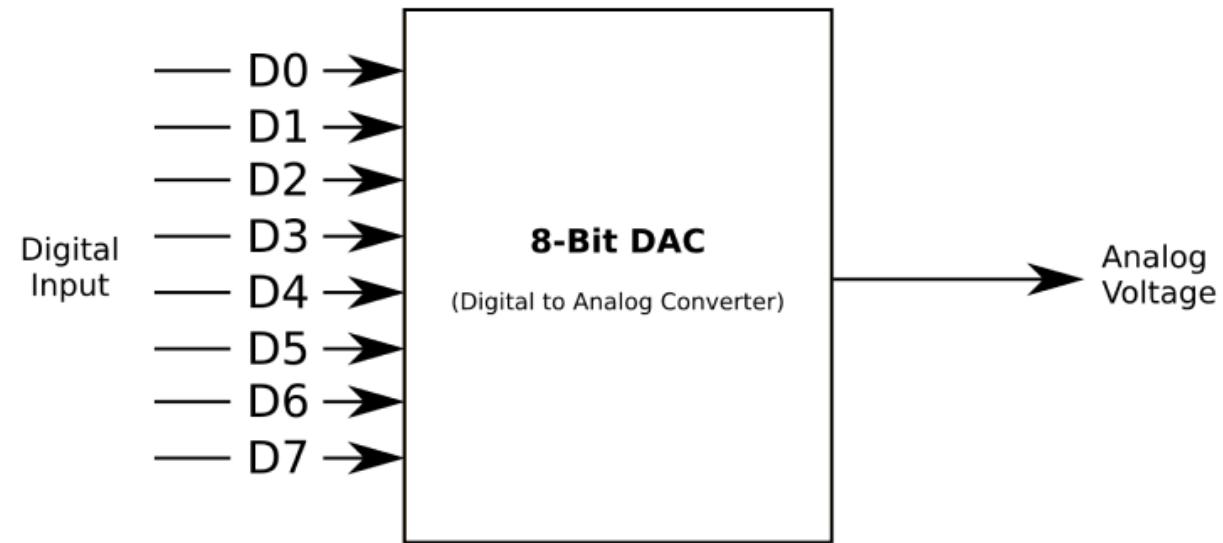
sbit RD = P2^5;
sbit WR = P2^6;
sbit INTR = P2^7;
sfr MYDATA = P1;

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
}

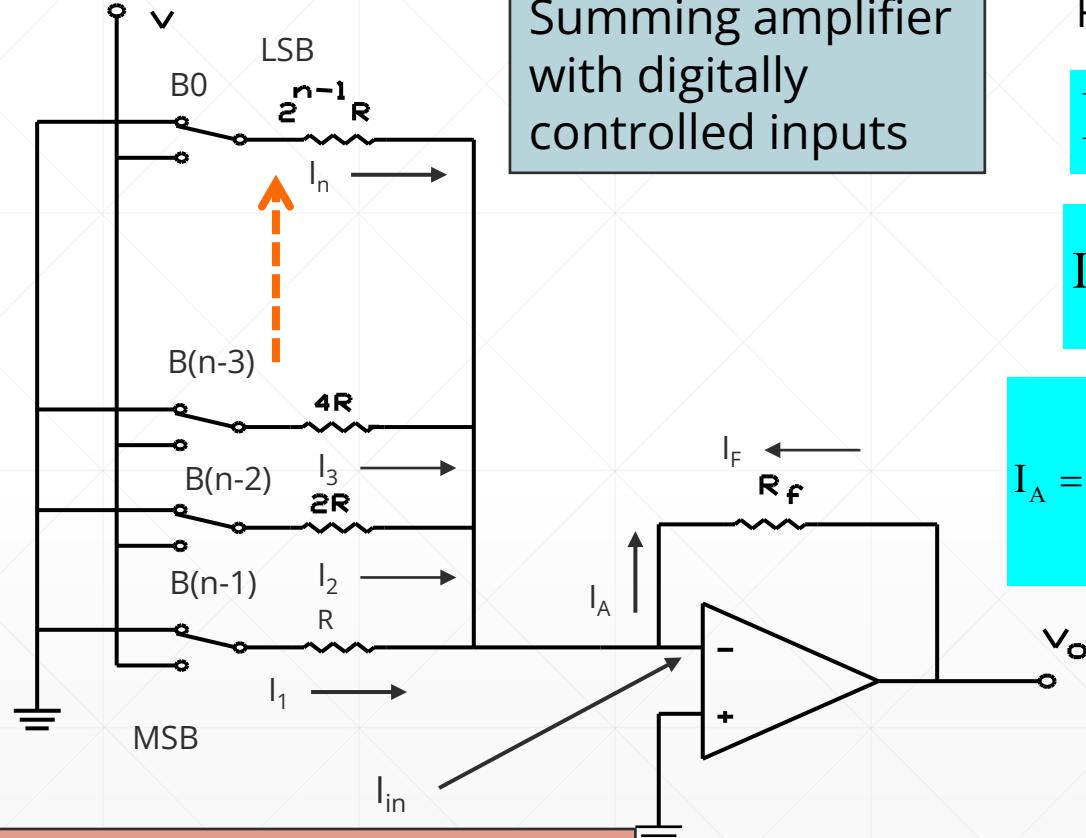
while (1){
    WR = 0;
    WR = 1; //send WR L to H pulse
    while (INTR == 1); //wait for EOC
    RD = 0 ; //send RD H to L pulse
    Value = MYDATA; //read value
    RD = 1; //reading complete
}
```

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



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

ECE4010

Rules of Ideal OP AMPS $I_{in} = 0, Z_{in} = \text{infinity}$

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

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

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

Formula for output V

$$V_o = -I_F \cdot R_F$$

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

Binary Weighted resistor - DAC

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

$$V_o = -R_F \sum_{i=1}^n \frac{B(n-i) \cdot V}{2^{i-1} R} \quad n=4$$

$$V_o = -R \cdot \left[\frac{B_3 \cdot V}{2^{1-1} R} + \frac{B_2 \cdot V}{2^{2-1} R} + \frac{B_1 \cdot V}{2^{3-1} R} + \frac{B_0 \cdot V}{2^{4-1} R} \right]$$

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

$$V_o = -\left[1 + \frac{1}{2} + 0 + \frac{1}{8} \right] = -[1 + 50 + 1.2] = -51.2 \text{ V}$$

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

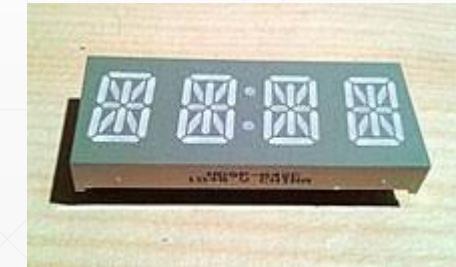
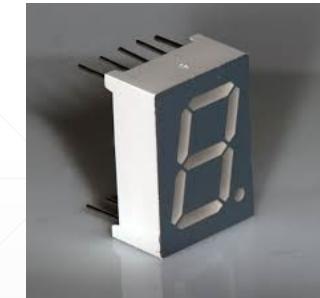
8051 - Input Output Peripherals

- **Input**
 - Keypad
-
- **Output**
 - LCD



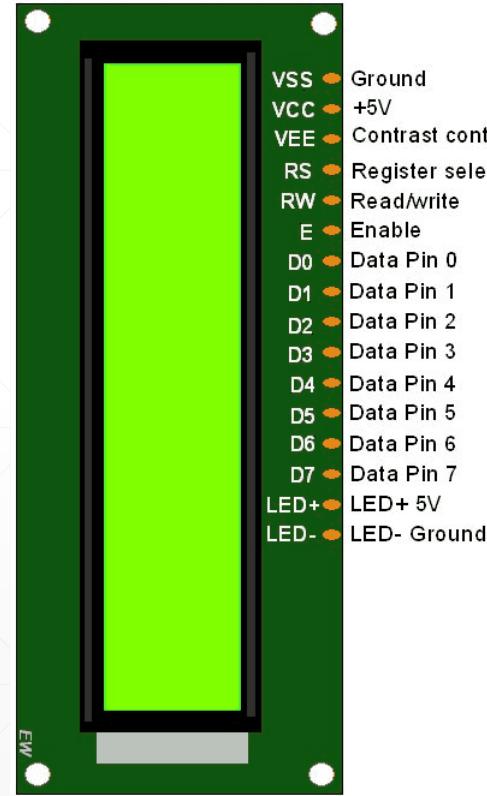
8051 - Output Peripheral - LCD

- 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

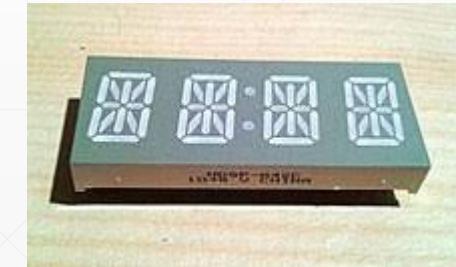
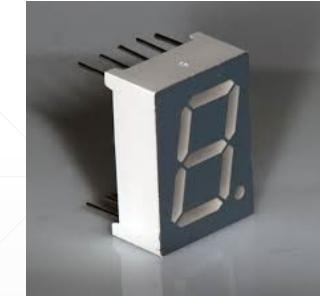
8051 - Input Output Peripherals

- **Input**
 - Keypad
-
- **Output**
 - LCD



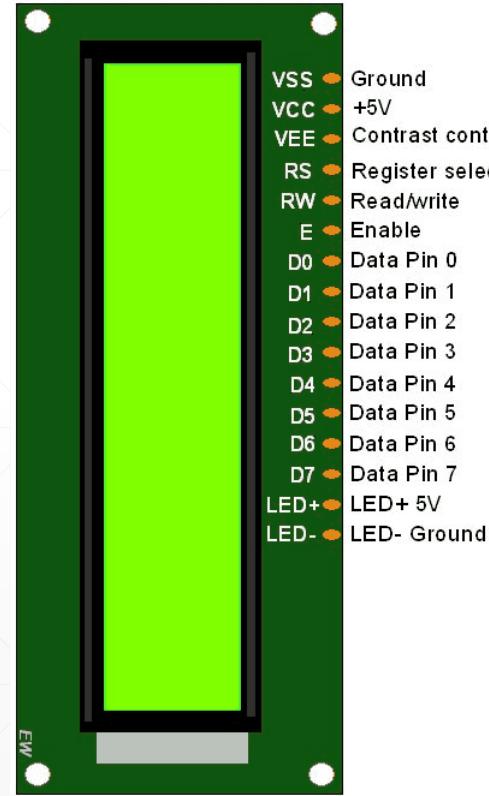
8051 - Output Peripheral - LCD

- 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

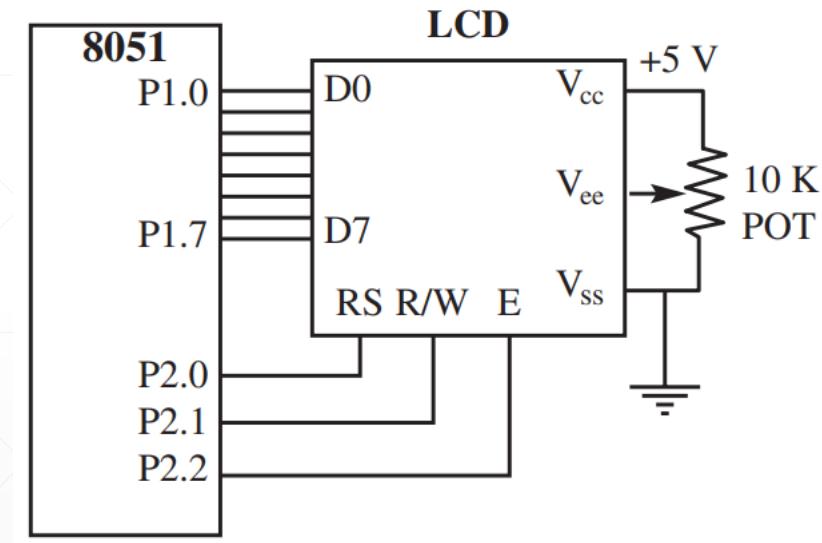
- Instruction command codes (**Hex**) that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor
- We also use RS = 0 to check the busy flag bit to see if the LCD is ready to receive information
- The busy flag is D7 and can be read when R/W = 1

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(0xOE);
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');

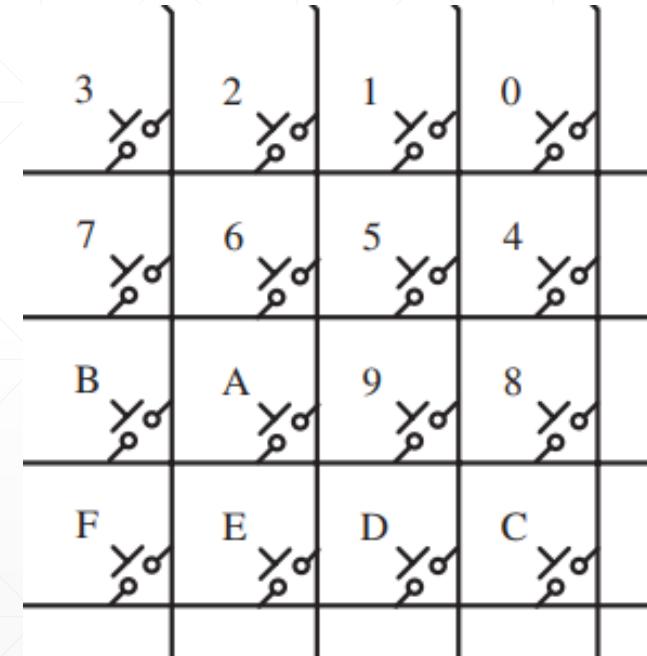
}

```
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;
}
```

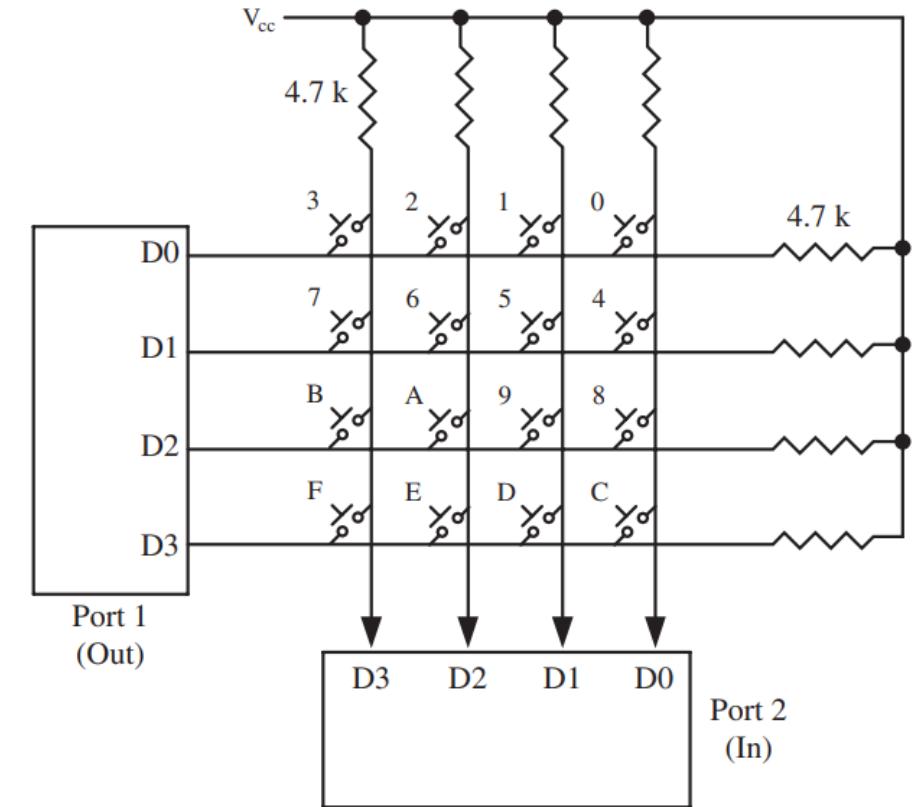
8051 - Input Peripheral - Keypad

- 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 8×8 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



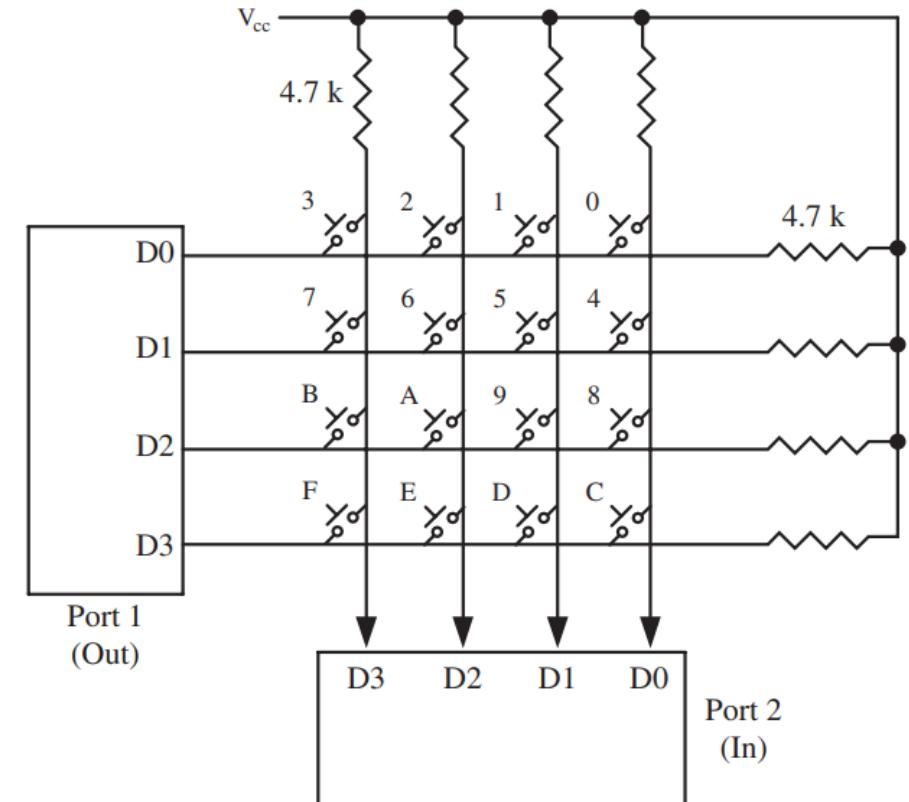
8051 - Input Peripheral - Keypad

- 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



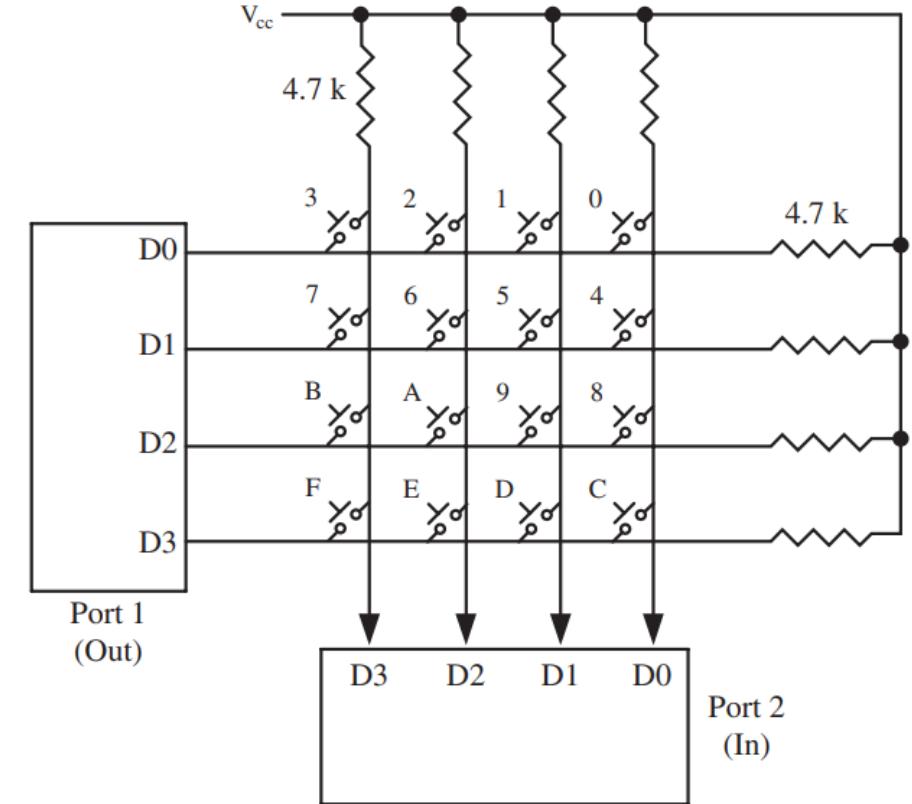
8051 - Input Peripheral - Keypad

- 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**



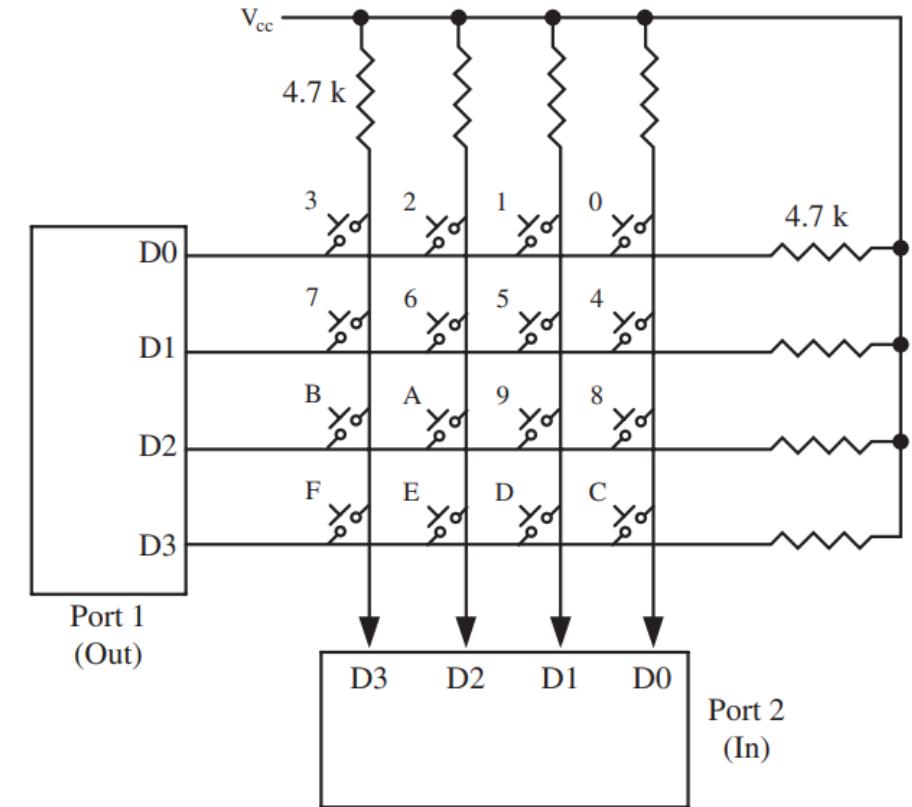
8051 - Input Peripheral - Keypad

- 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

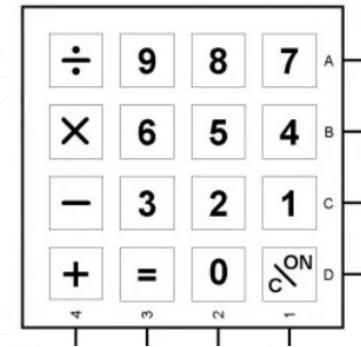
```
//Keypad Connections    char Read_Keypad()
sbit R1 = P1^0;           {
sbit R2 = P1^1;           C1=1; C2=1; C3=1; C4=1;
sbit R3 = P1^2;           R1=0; R2=1; R3=1; R4=1;
sbit R4 = P1^3;           if(C1==0){Delay(100);while(C1==0);return
sbit C1 = P1^4;           '7';}
sbit C2 = P1^5;           if(C2==0){Delay(100);while(C2==0);return
sbit C3 = P1^6;           '8';}
sbit C4 = P1^7;           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  
'-';}
```

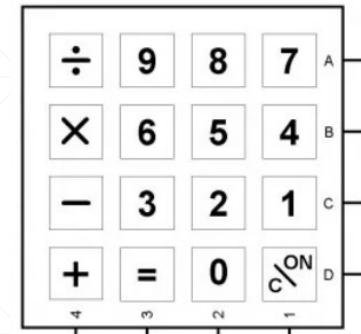
R1=1; R2=1; R3=1; **R4=0**;

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

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

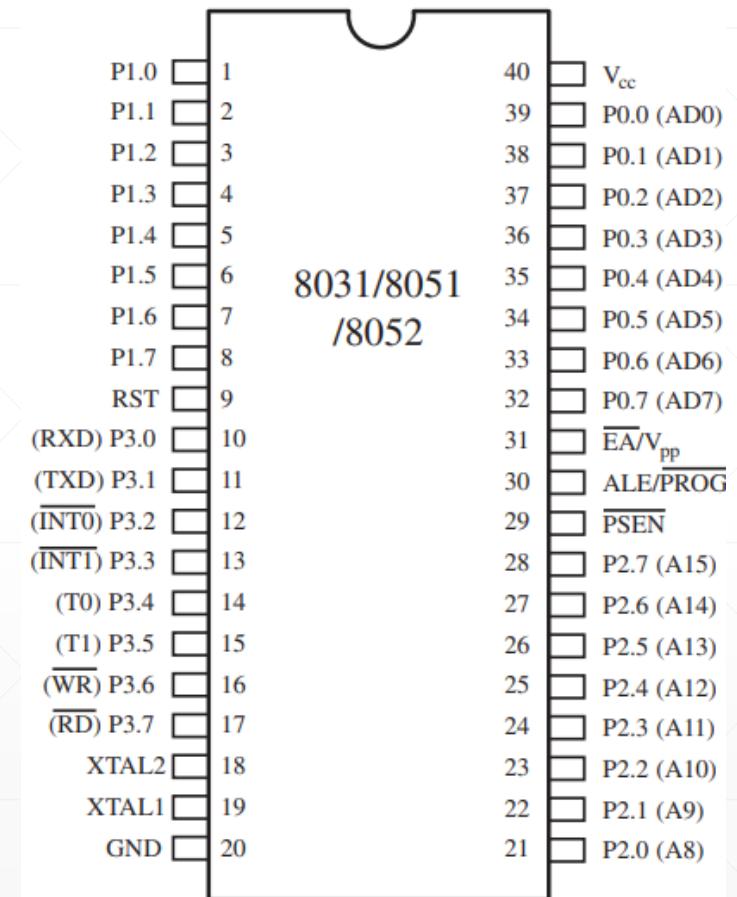
```
if(C3==0){Delay(100);while(C3==0);return  
'=';}
```

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



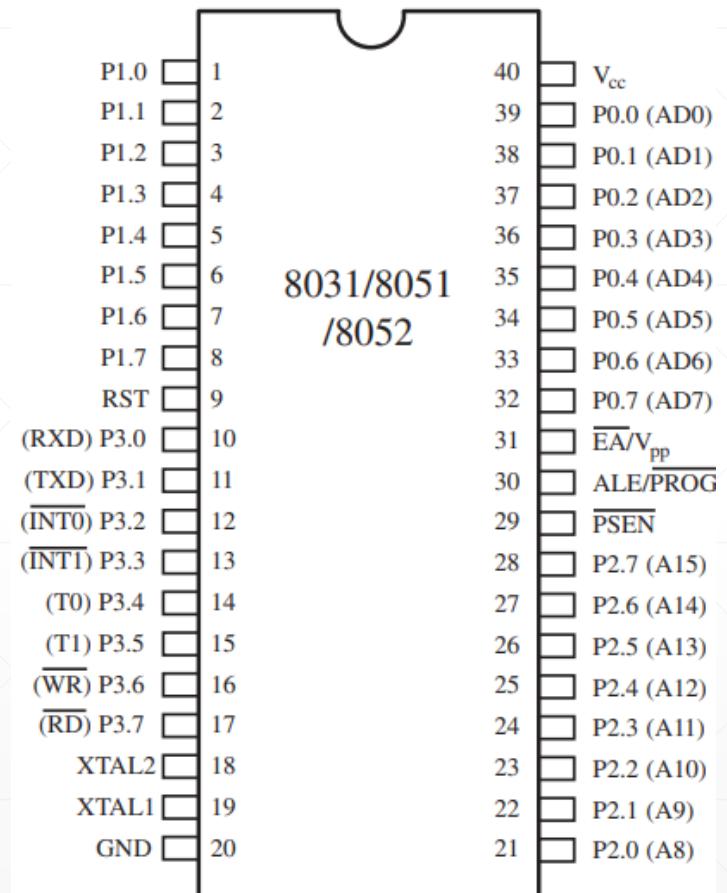
8051 – Memory Interfacing

- 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



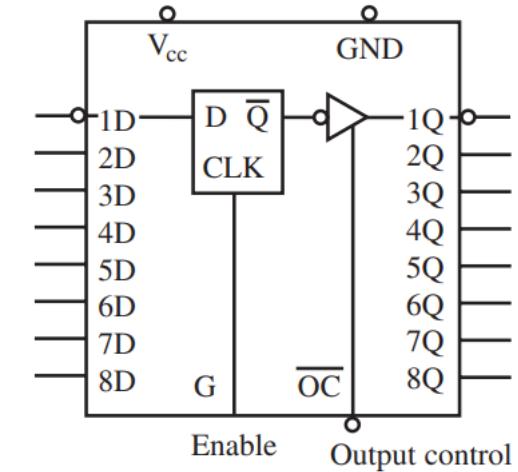
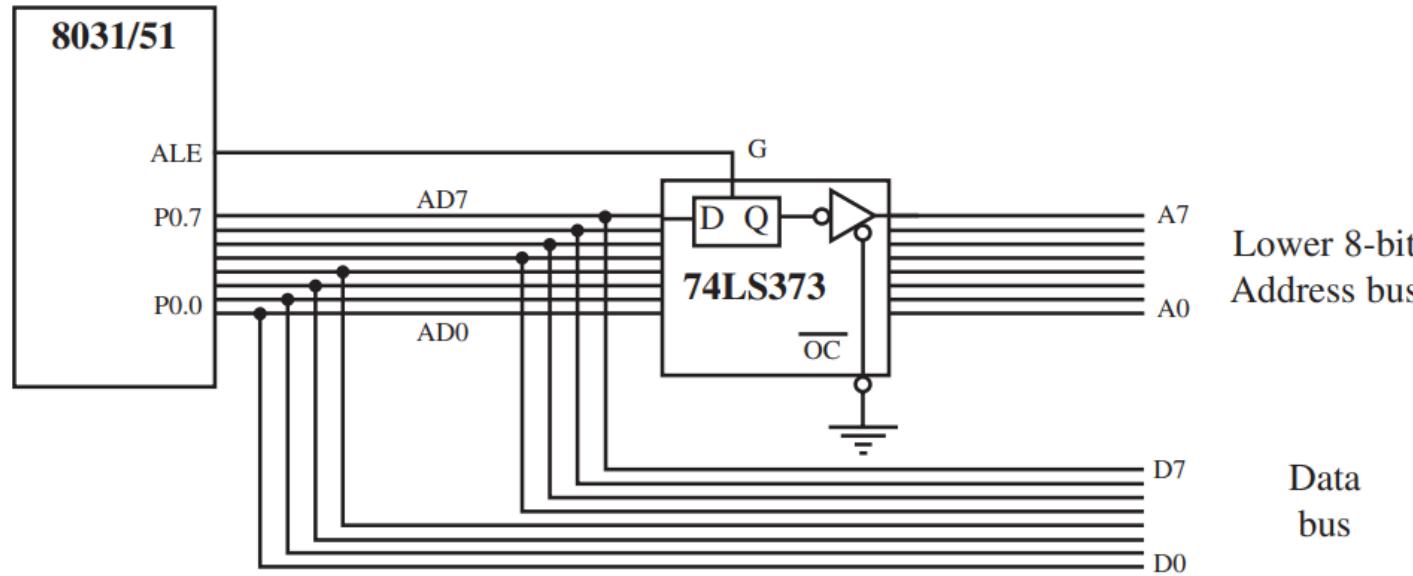
8051 – Memory Interfacing

- 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**

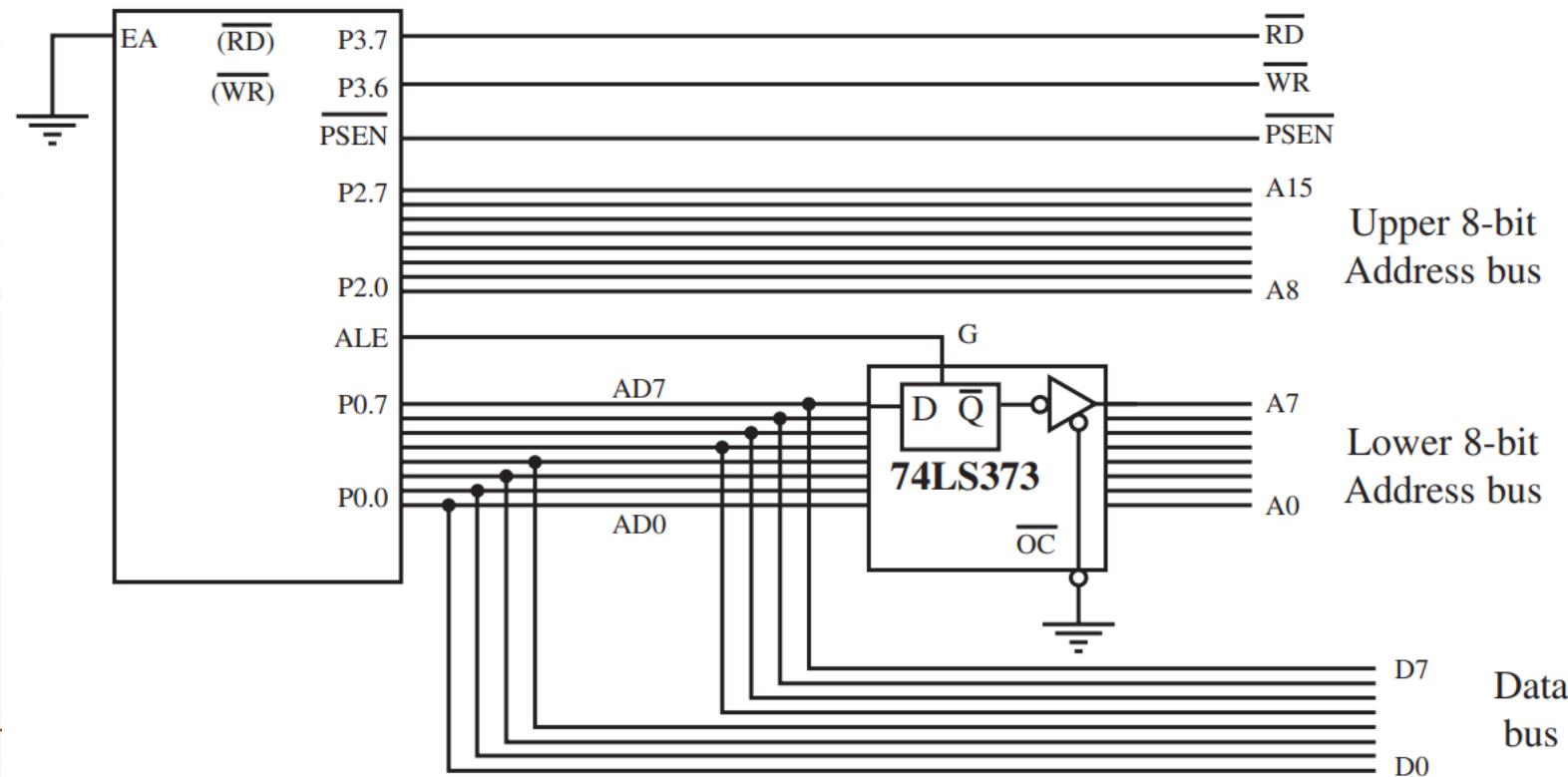


Function Table

Output control	Enable		Output
	G	D	
L	H	H	H
L	H	L	L
L	L	X	X
H	X	X	Z

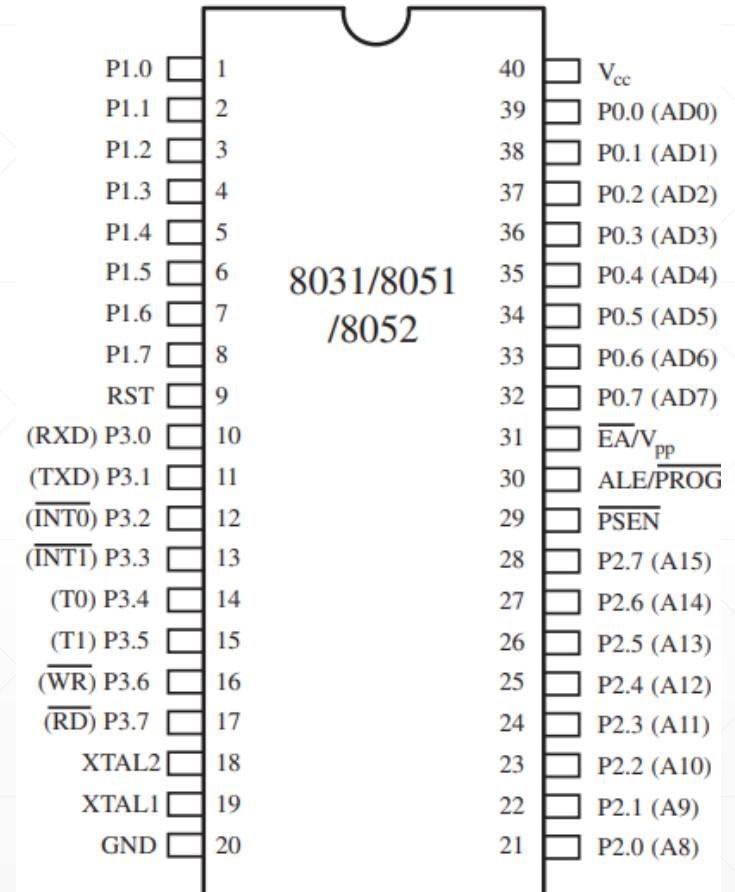
8051 - Memory Interfacing

- 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



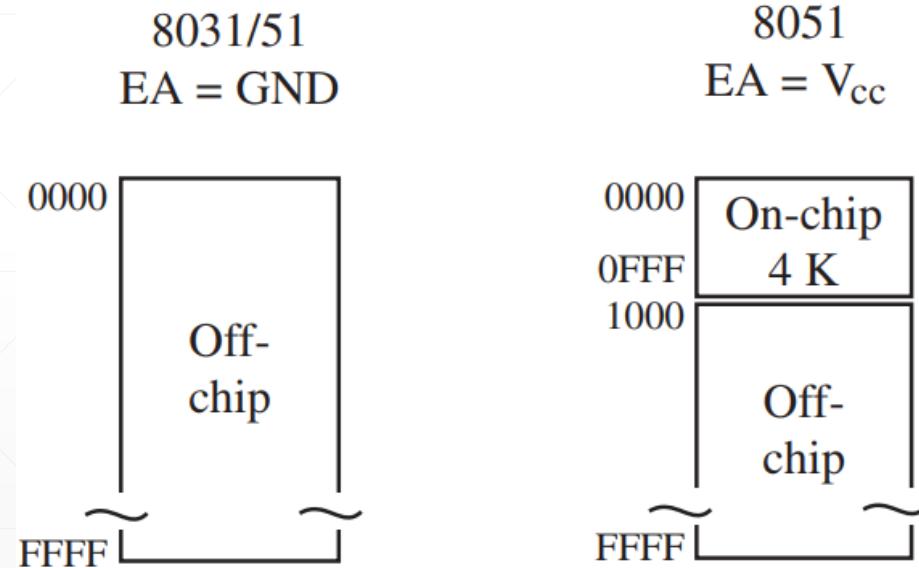
8051 - Memory Interfacing

- 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

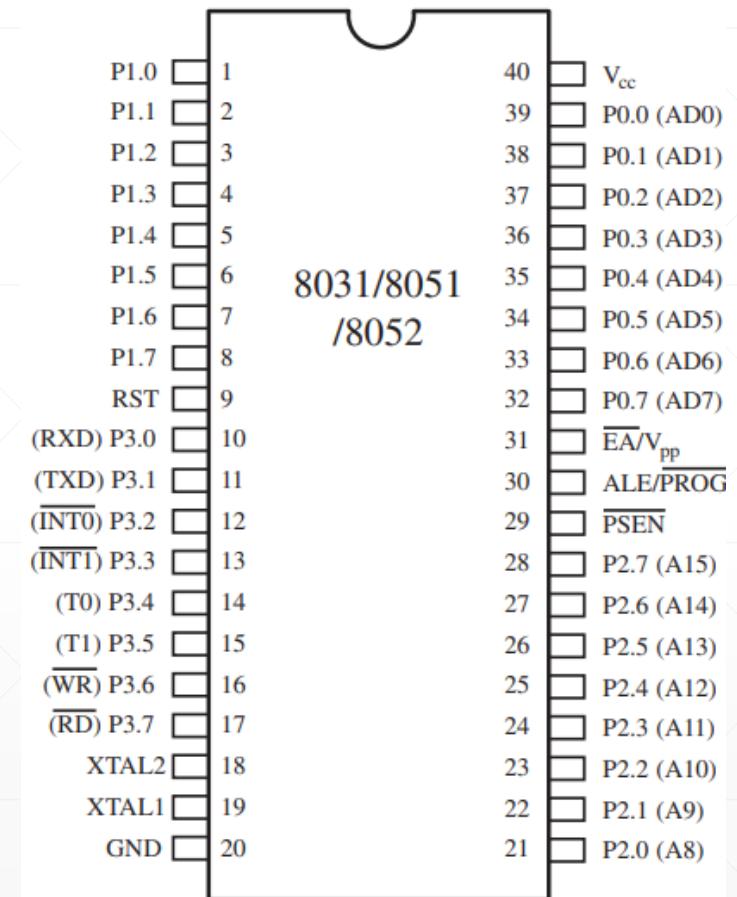
ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

8051 PROGRAMMING IN C

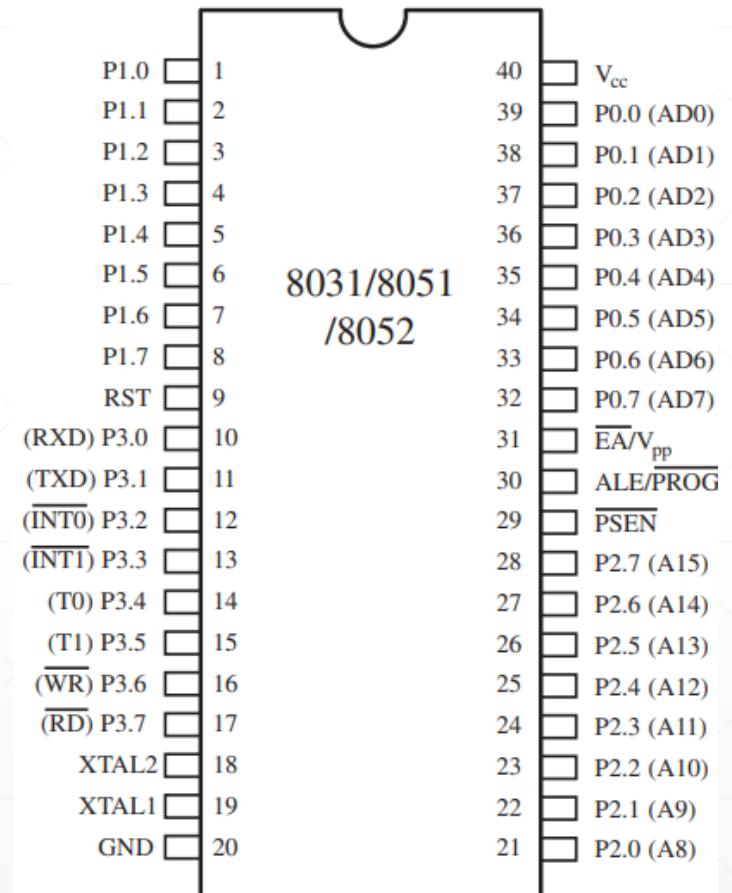
8051 – Memory Interfacing

- 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



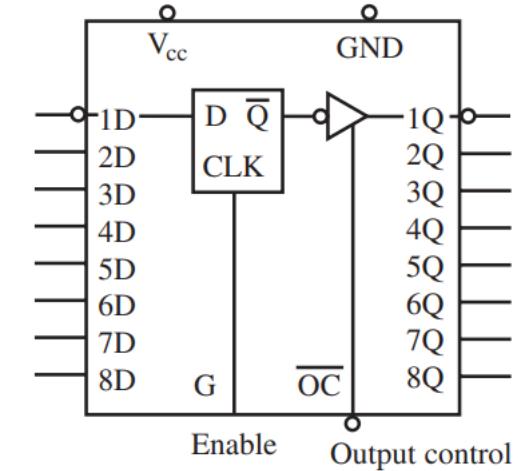
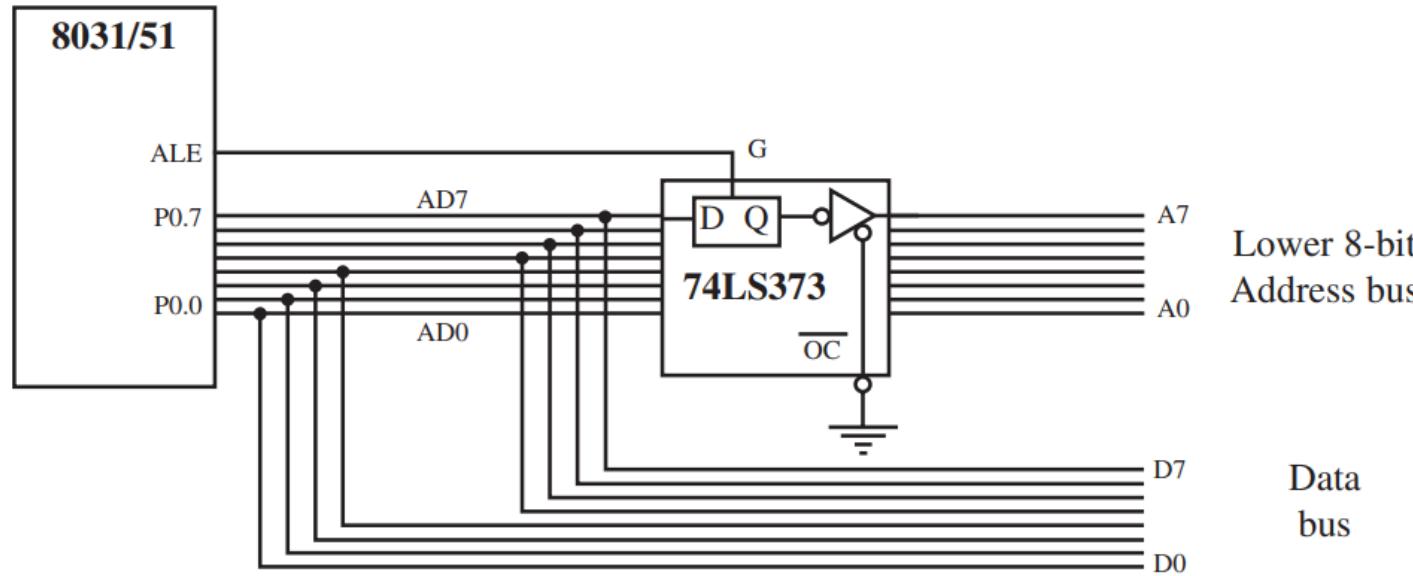
8051 – Memory Interfacing

- 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**

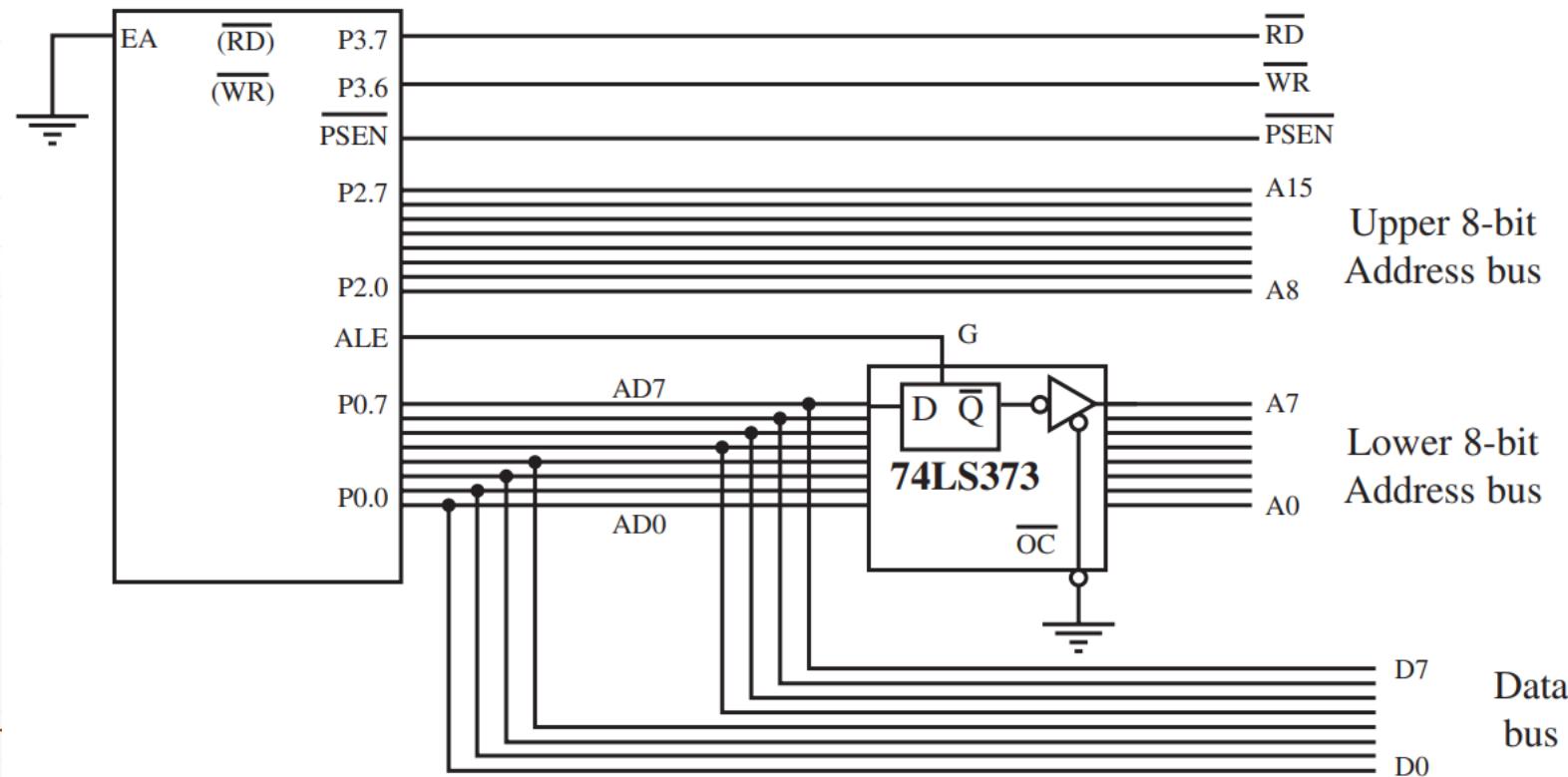


Function Table

Output control	Enable		Output
	G	D	
L	H	H	H
L	H	L	L
L	L	X	X
H	X	X	Z

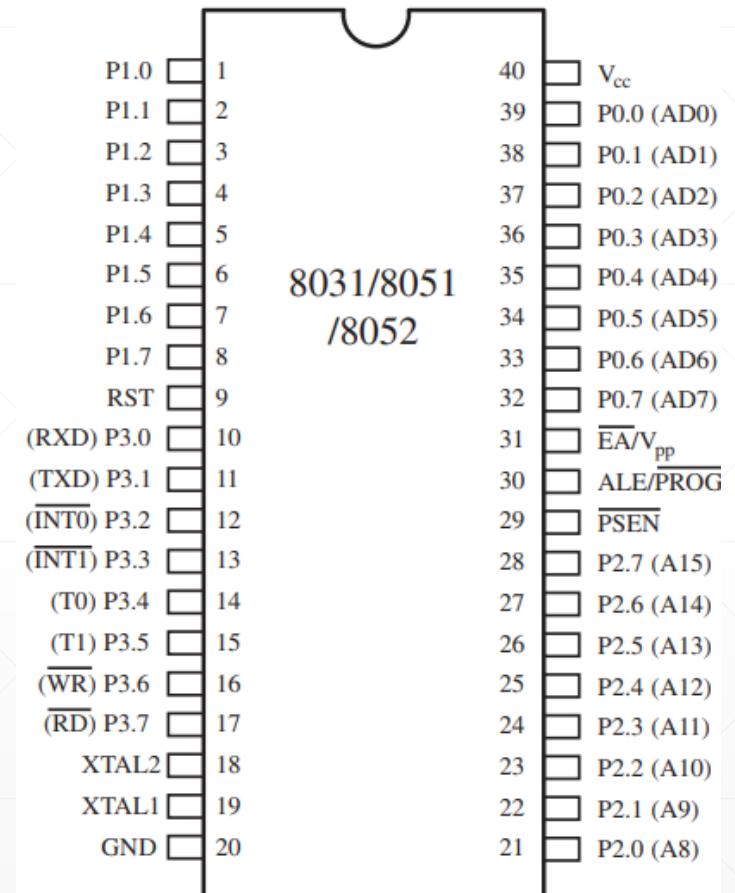
8051 - Memory Interfacing

- 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



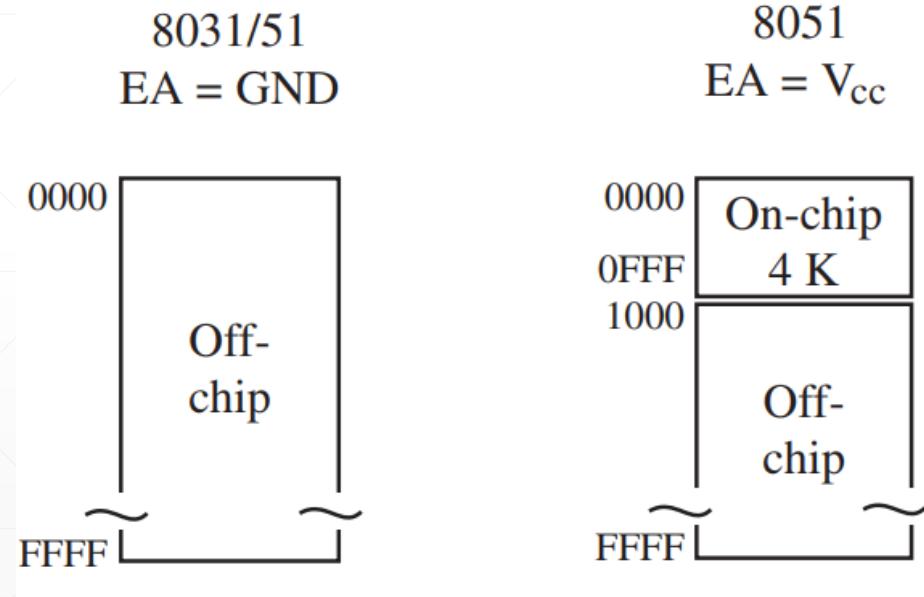
8051 - Memory Interfacing

- 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

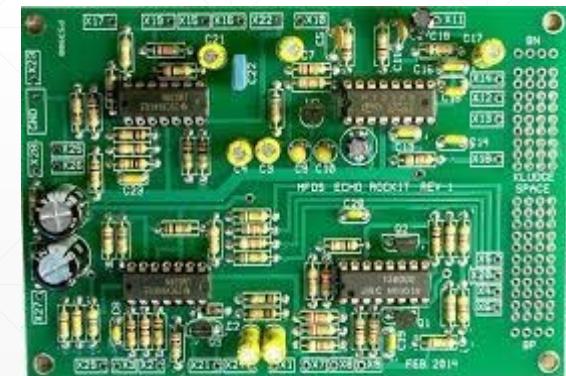
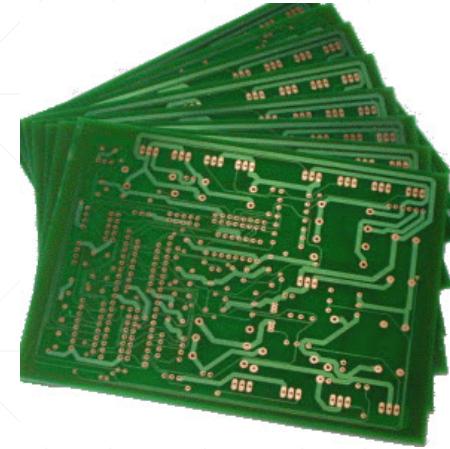
- ROM space allocation



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?

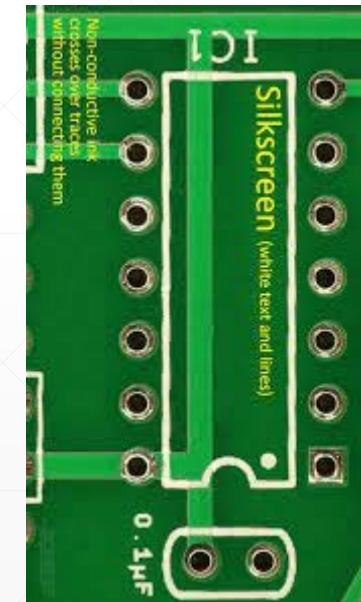
Solder Mask

- 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".

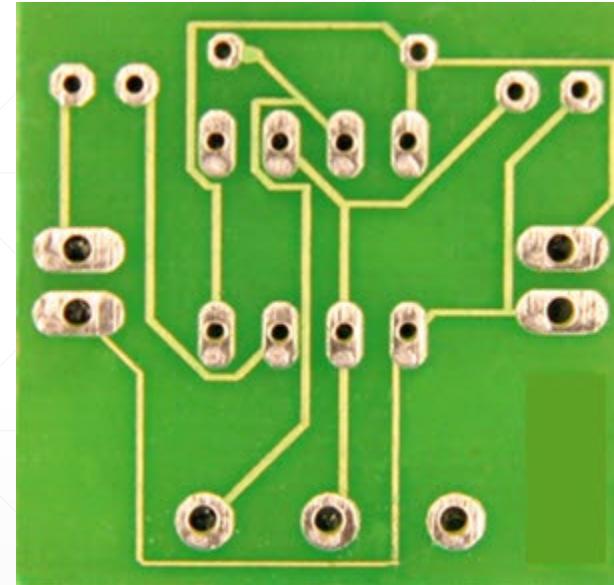
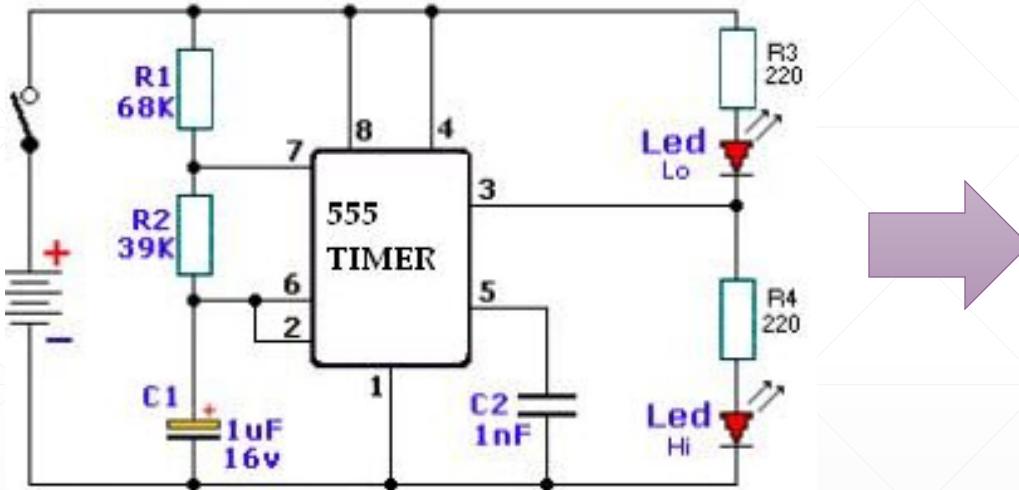


Silkscreen

- 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



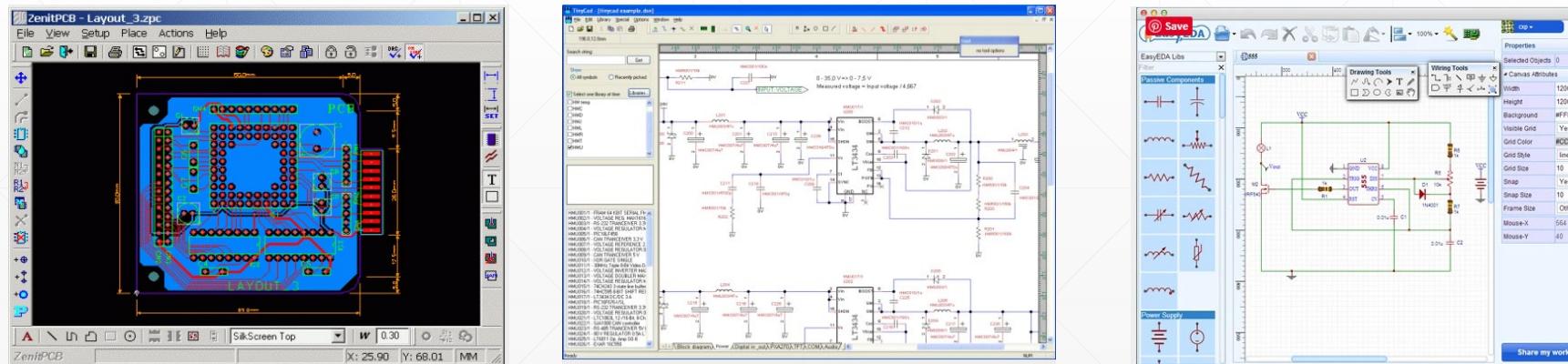
PCB Design Process



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

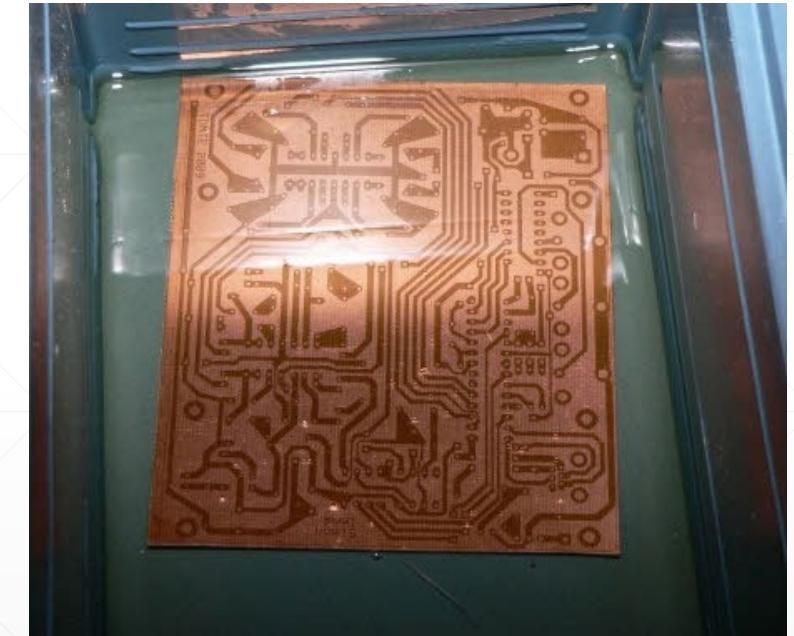
- The printed circuit layout can be printed in different ways on PCBs like a manual pen, dry transfers, pen plotters, and printers. The laser printers are a better way to print the layouts on printed circuit boards.



PCB Design Process

Step 6: Stripping and Etching

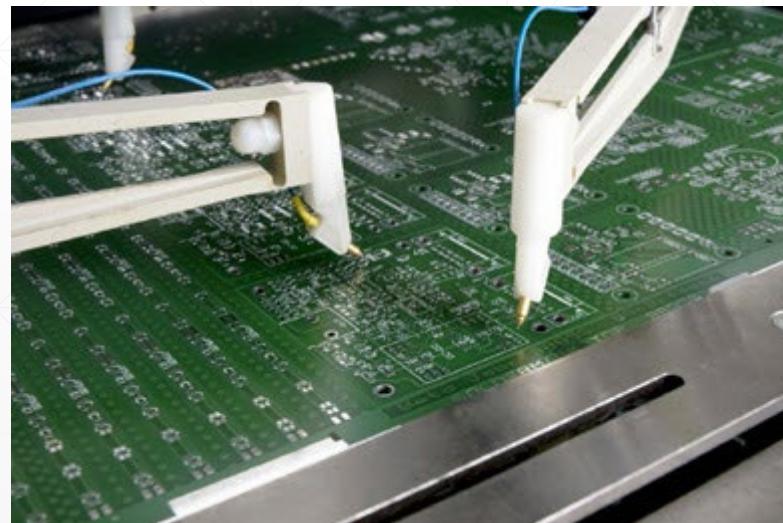
- This process involves removing the unwired copper on the PCBs by using different types of chemicals like ferric chloride, ammonium per-sulfate, etc.
- Make the solvent by mixing 1% of sodium hydroxide and 10 grams of sodium hydroxide pellets to one liter of water and mix it until everything is dissolved.



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

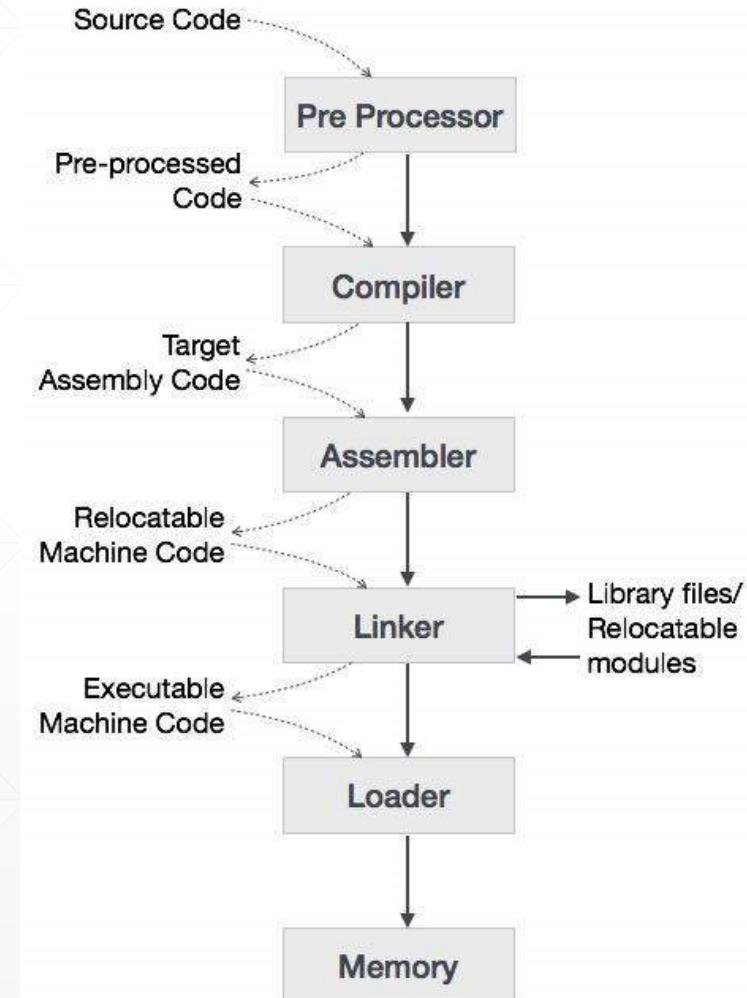
- There are many embedded software languages used in the industry such as C, C++, Java and Ada. But C-Programming is the most widely used language for embedded software design.
- C-Programming has benefits for both low level hardware interactions and high-level software language features.
- This provides 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.

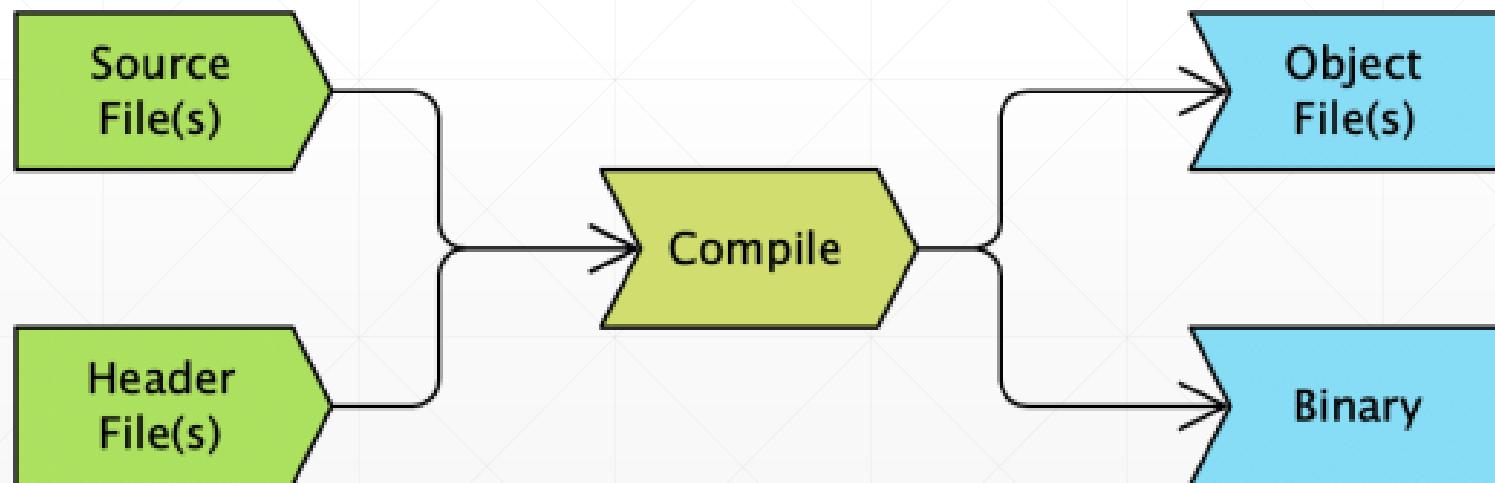
- User writes a program in C language (high-level language).
- The C compiler, compiles the program and translates it to assembly program (low-level language).
- An assembler then translates the assembly program into machine code (object).
- A linker tool is used to link all the parts of the program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

A software that translates the computer code written in high-level programming language to machine language

Helps to convert the high-level source code into machine understandable machine code

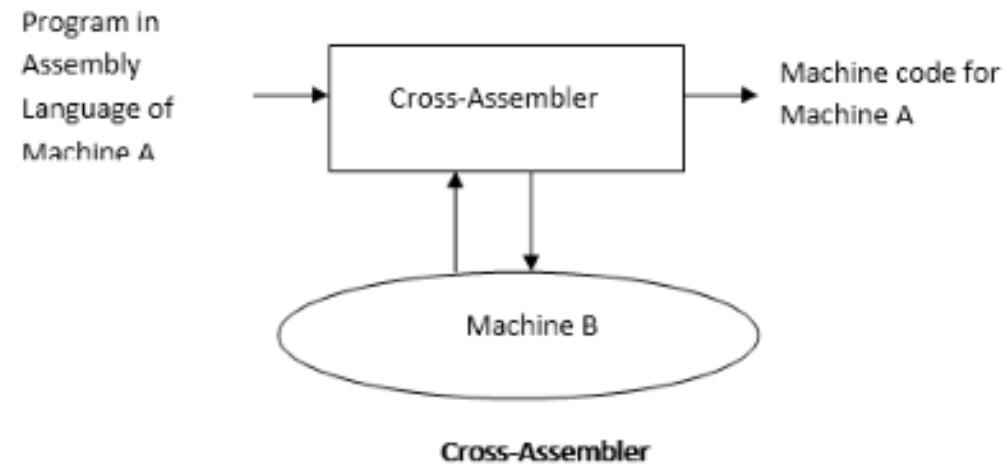
CROSS COMPILER

A software that can create executable code for platforms other than the one on which the compiler is running

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.



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

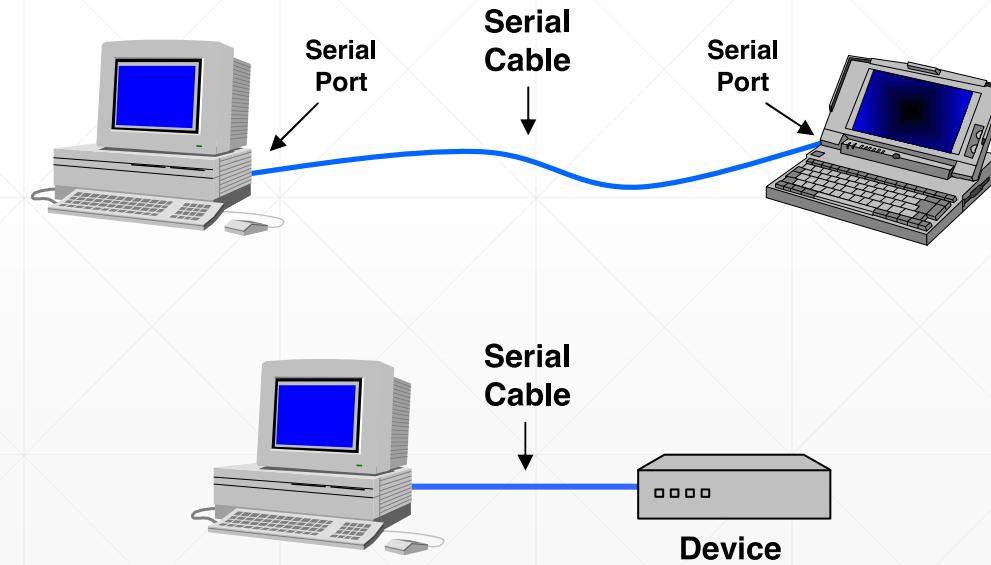
Embedding Network Technologies

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+

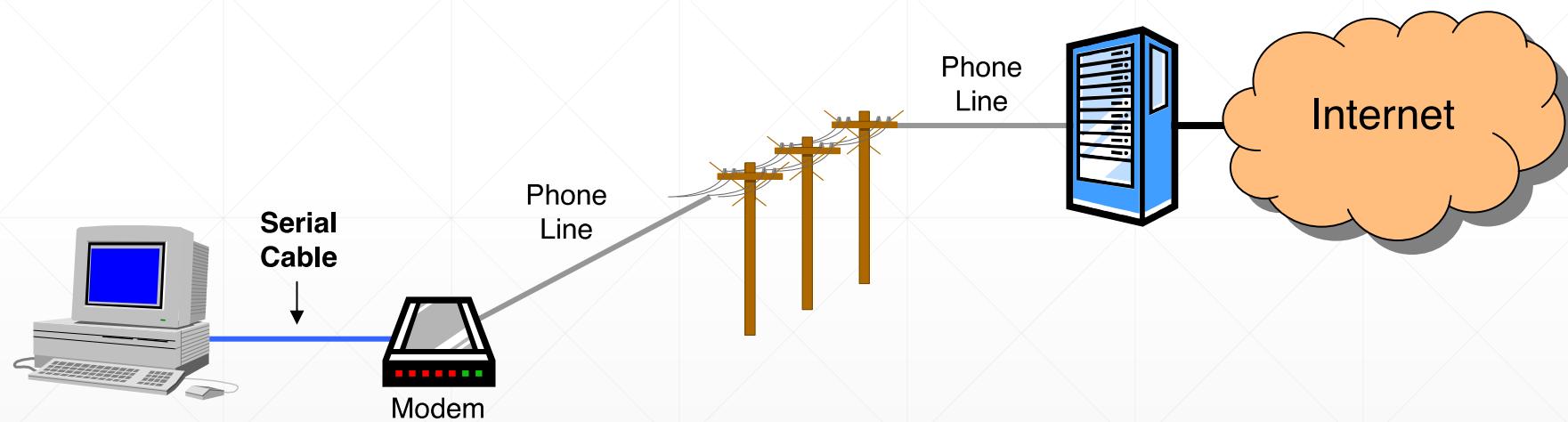
UART Use Cases

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



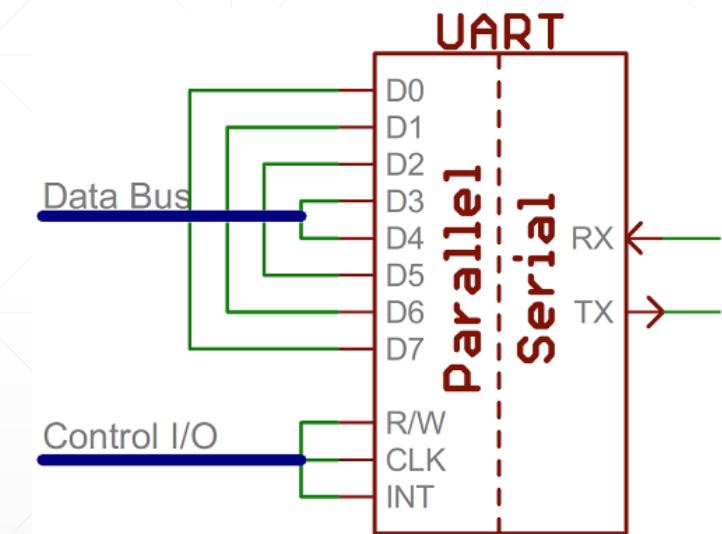
UART Use Cases

- Used to be commonly used for internet access

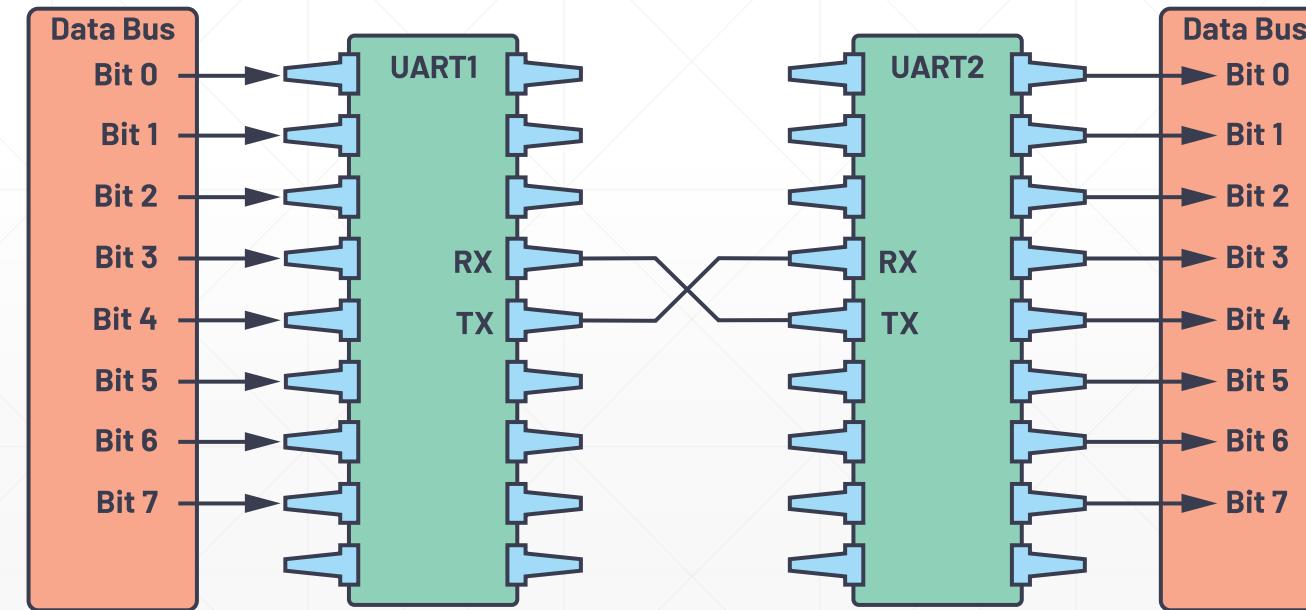
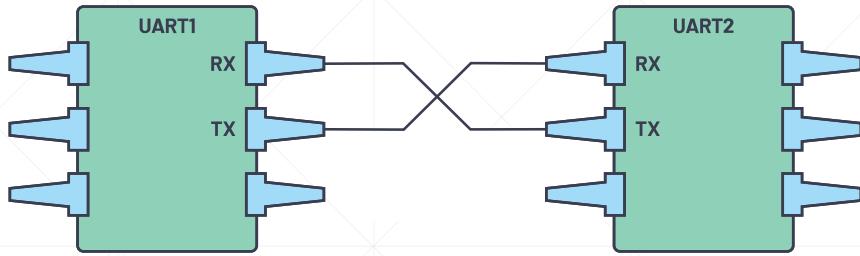


UART

- 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



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



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



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



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



UART – Data Transmission

Stop Bits

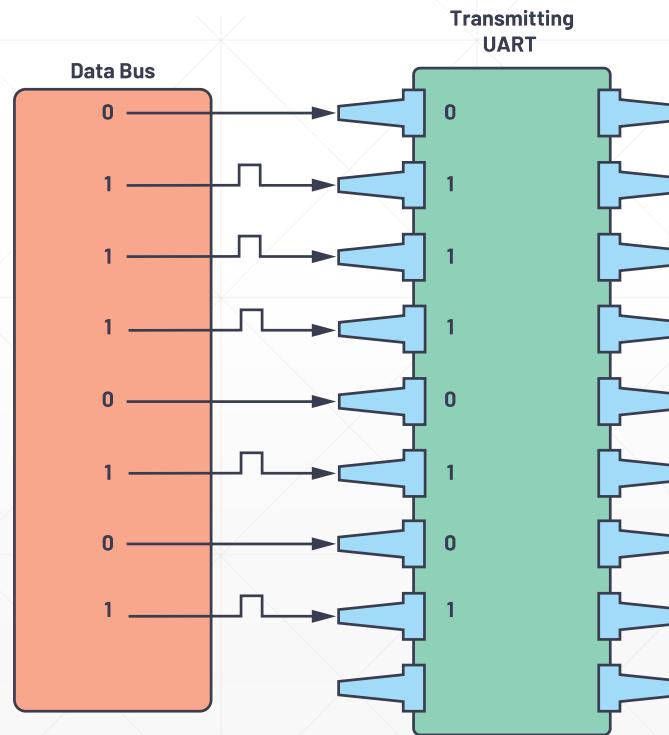
- To signal the end of the data packet, the sending UART drives the data transmission line from a **low voltage to a high voltage** for one (1) to two (2) bit(s) duration



UART - Data Transmission - Steps

Step 1

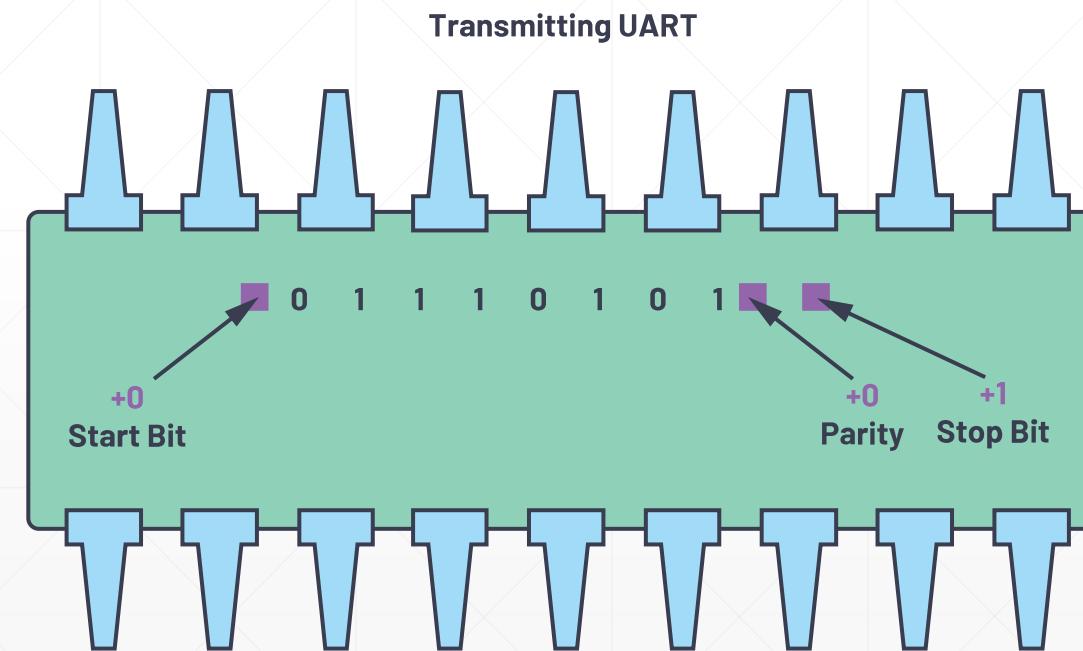
- The transmitting UART receives data in parallel from the data bus



UART - Data Transmission - Steps

Step 2

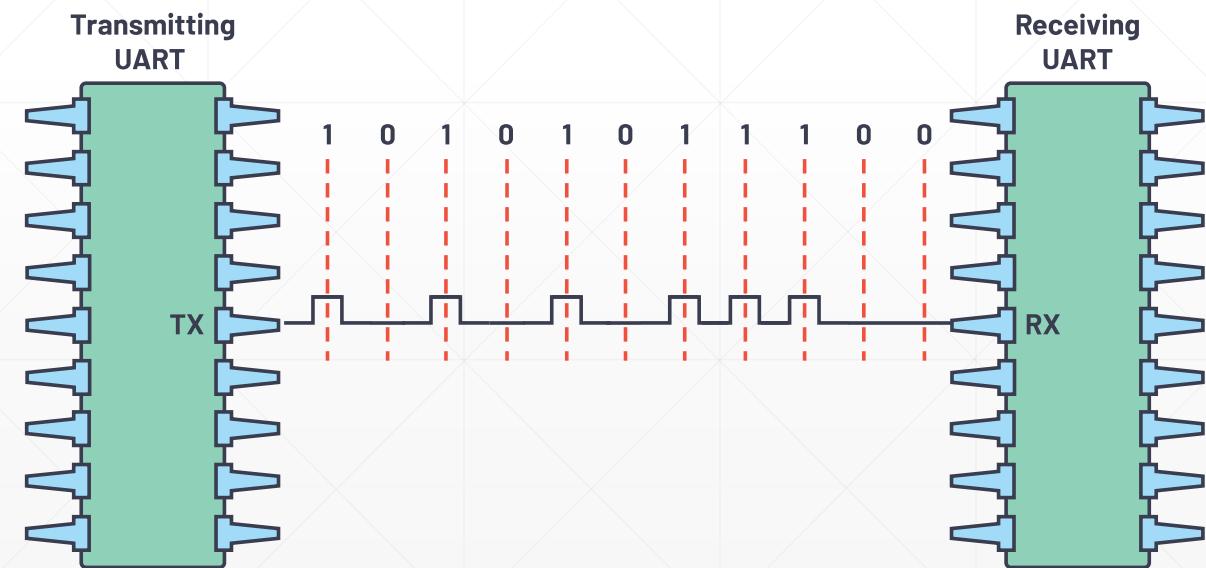
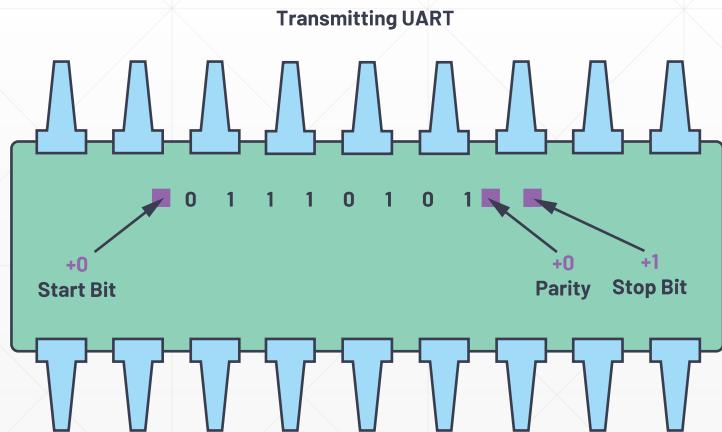
- The transmitting UART adds the start bit, parity bit, and the stop bit(s) to the data frame



UART - Data Transmission - Steps

Step 3

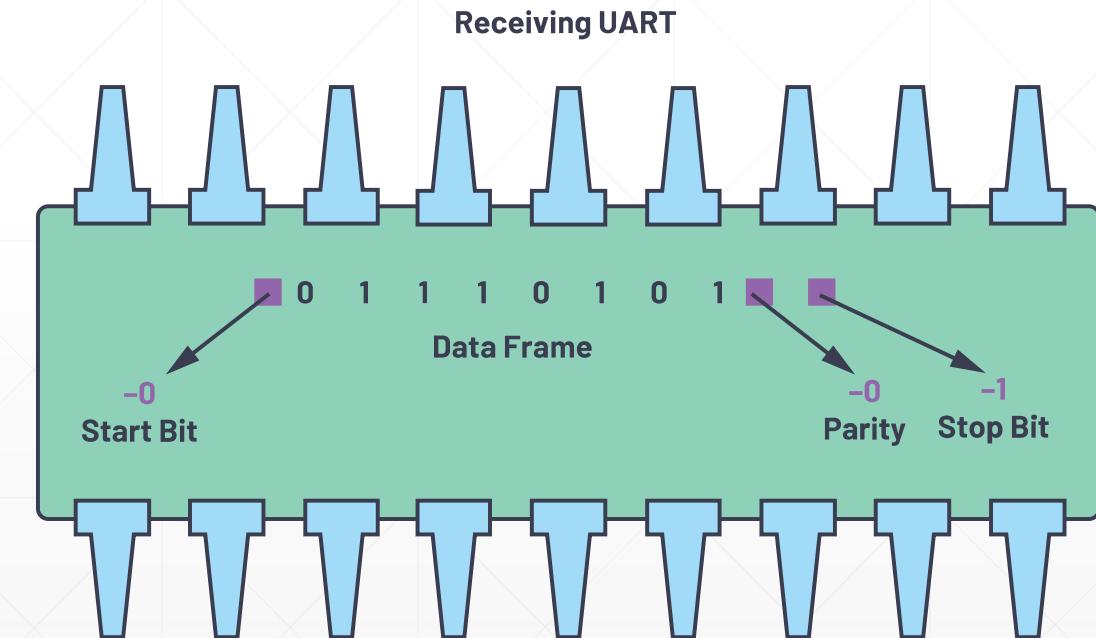
- The entire packet is sent serially starting from start bit to stop bit from the transmitting UART to the receiving UART
- The receiving UART samples the data line at the preconfigured baud rate



UART - Data Transmission - Steps

Step 4

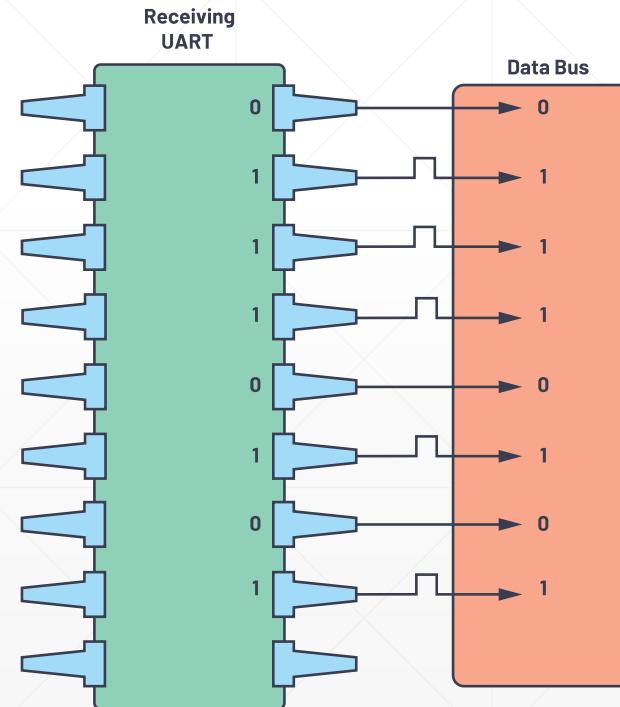
- The receiving UART discards the start bit, parity bit, and stop bit from the data frame



UART - Data Transmission - Steps

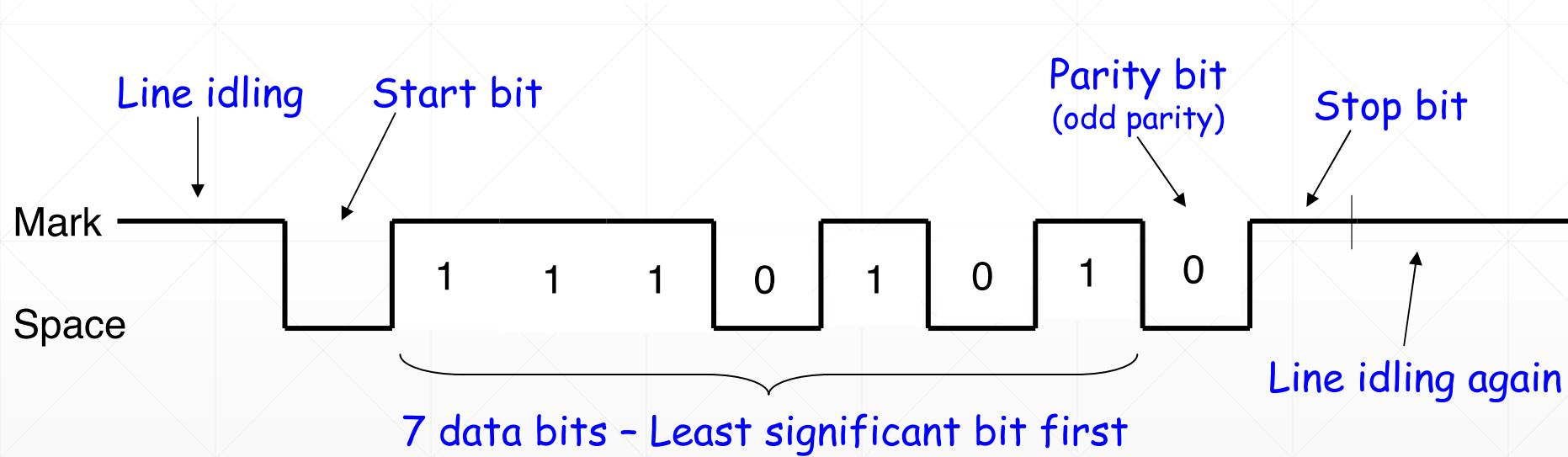
Step 5

- 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

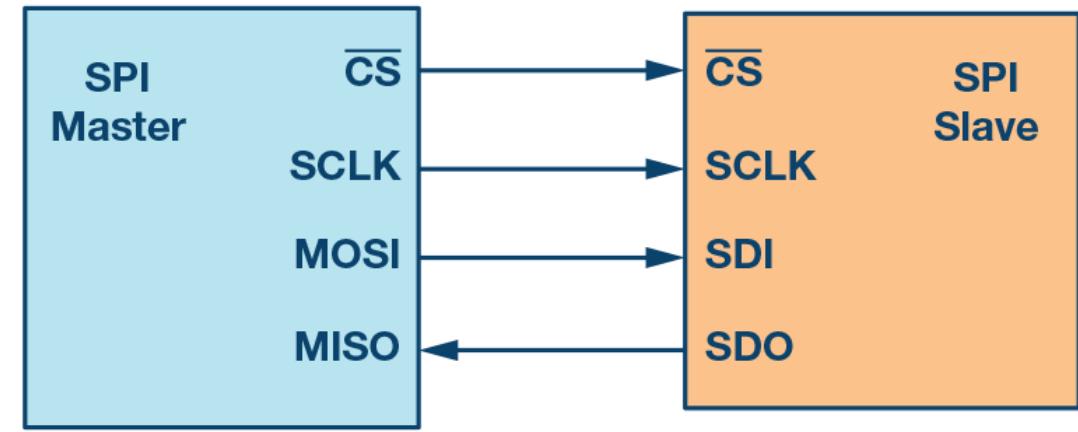
- Advantages
 - Hardware complexity is low
 - As this is one to one connection between two devices, software addressing is not required
 - It has parity bit to check quality
- Limitations
 - The data frame size is limited
 - It is suitable for communication between only two devices
 - Proper baud rate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

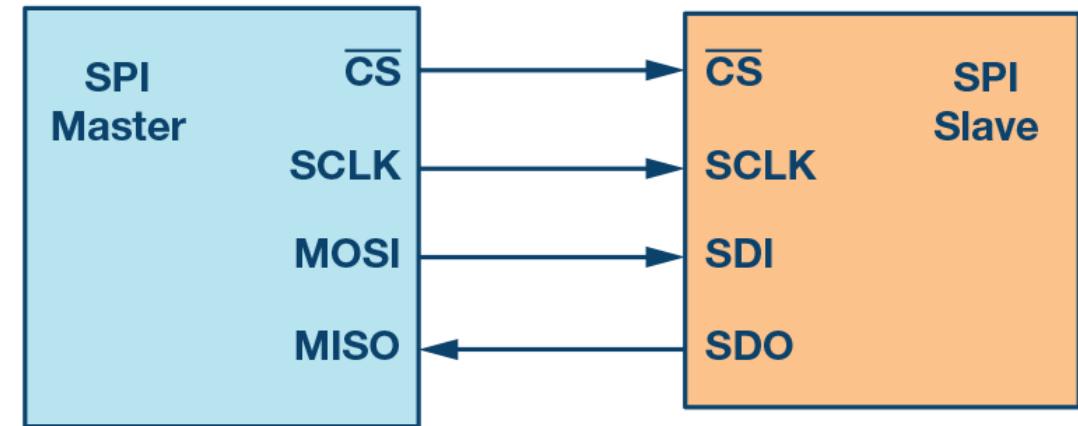
- 4-wire SPI devices have four signals:
 - Clock (SPI CLK, SCLK)
 - Chip select (CS)
 - Master out, slave in (MOSI)
 - Master in, slave out (MISO)



- 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

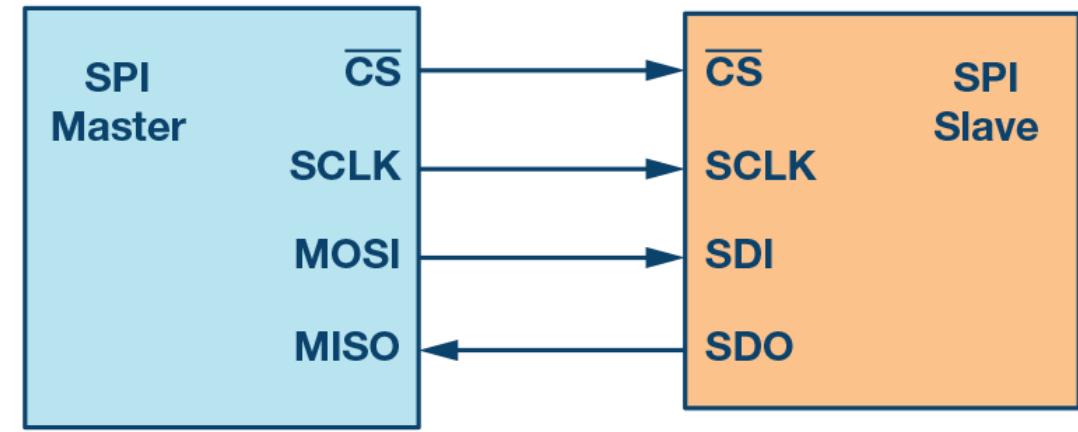
ECE 4010

- Abhishek Joshi
SEEE, VIT Bhopal

Embedding Network Technologies

Serial peripheral interface (SPI)

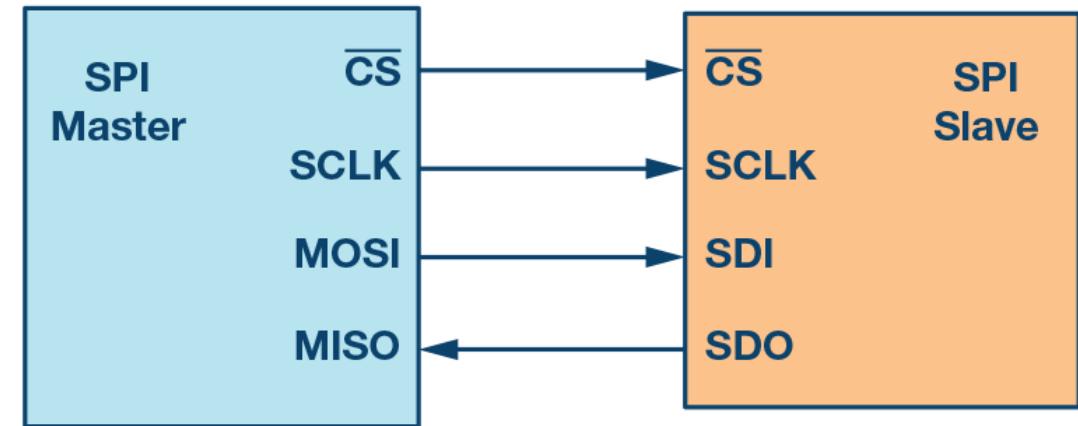
- 4-wire SPI devices have four signals:
 - Clock (SPI CLK, SCLK)
 - Chip select (CS)
 - Master out, slave in (MOSI)
 - Master in, slave out (MISO)



- 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)

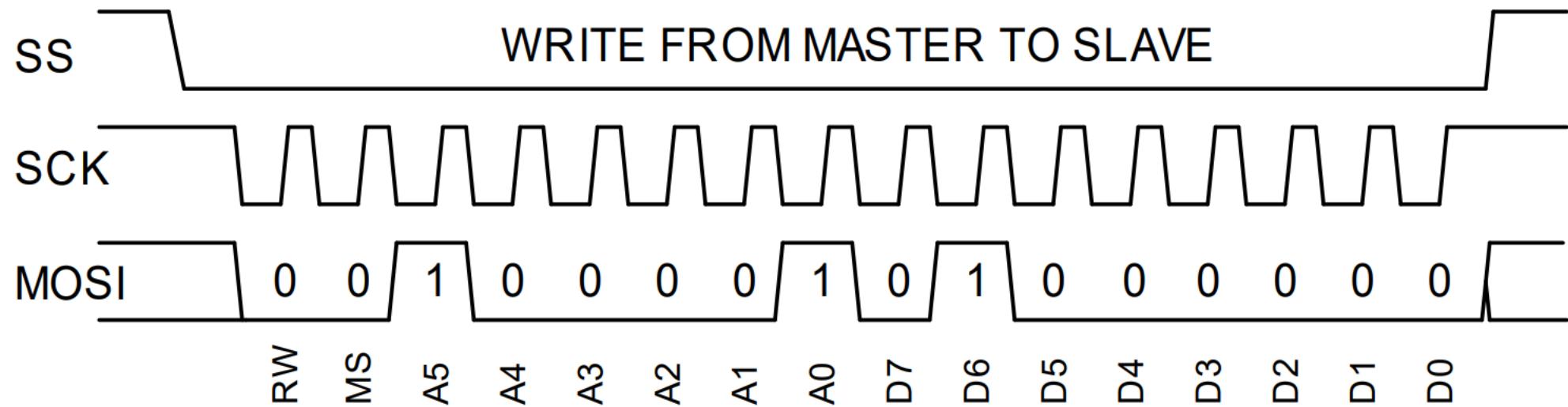
Serial peripheral interface (SPI)

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)

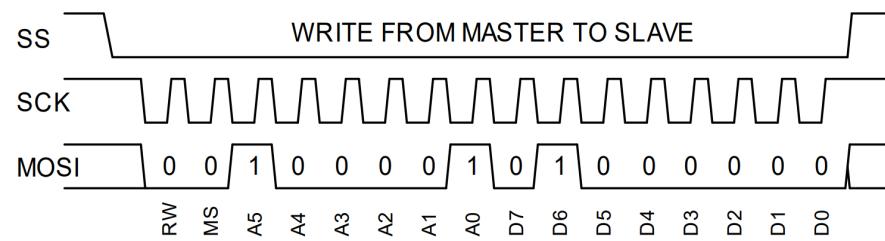
Timing Diagram – Writing (Master Side)



Serial peripheral interface (SPI)

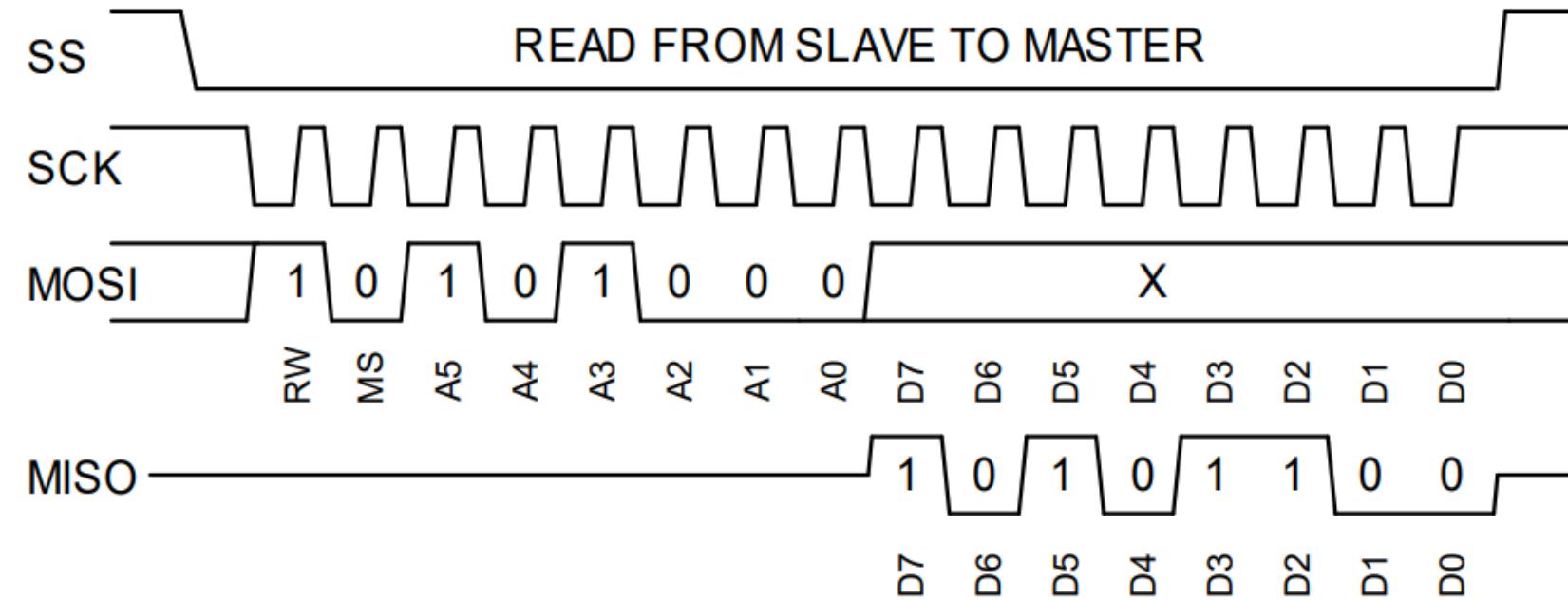
Timing Diagram - Writing (Master Side)

- 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)

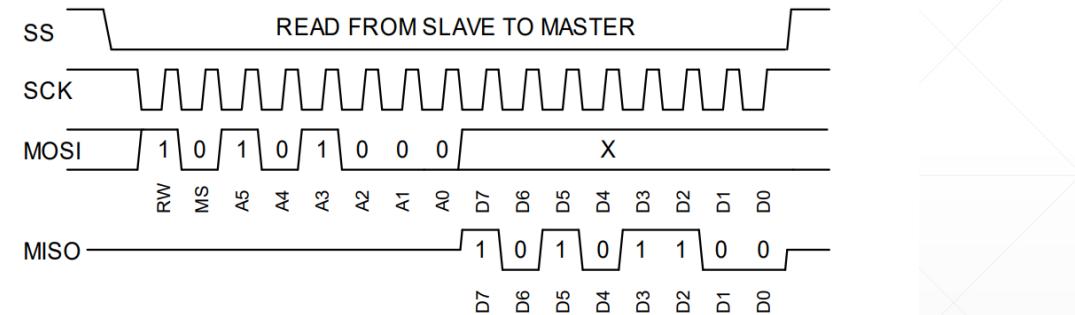
Timing Diagram – Reading (Master Side)



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)

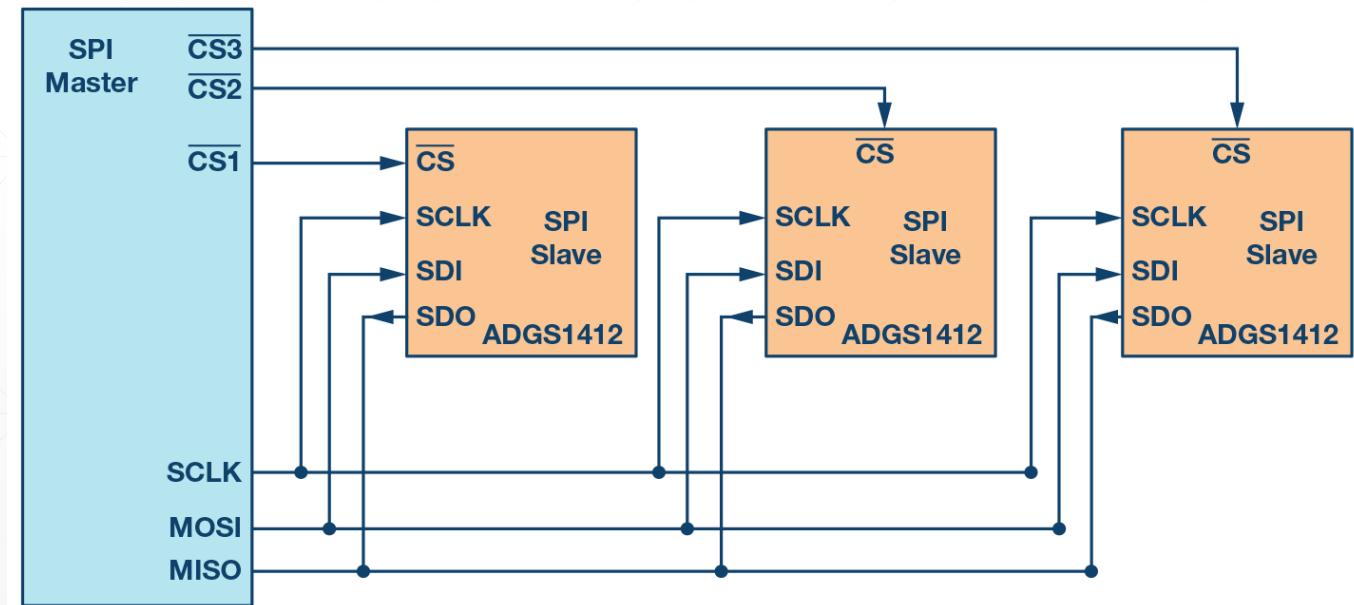


Serial peripheral interface (SPI)

Multi-Slave Configuration

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

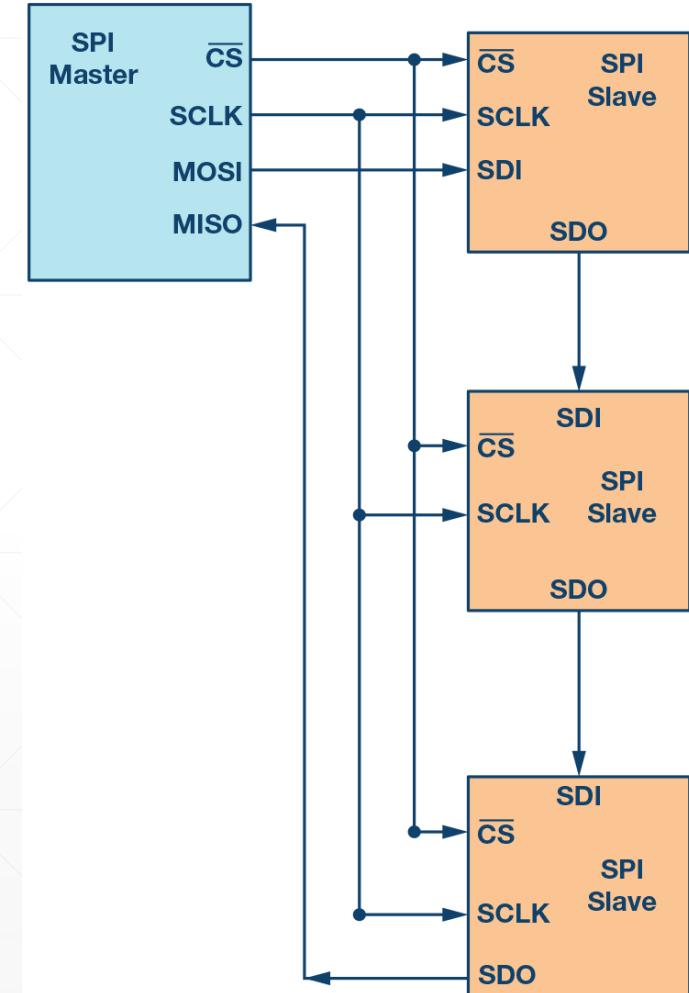
- Regular Mode
 - Individual CS signal
 - Same MOSI, MISO and SCLK



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)

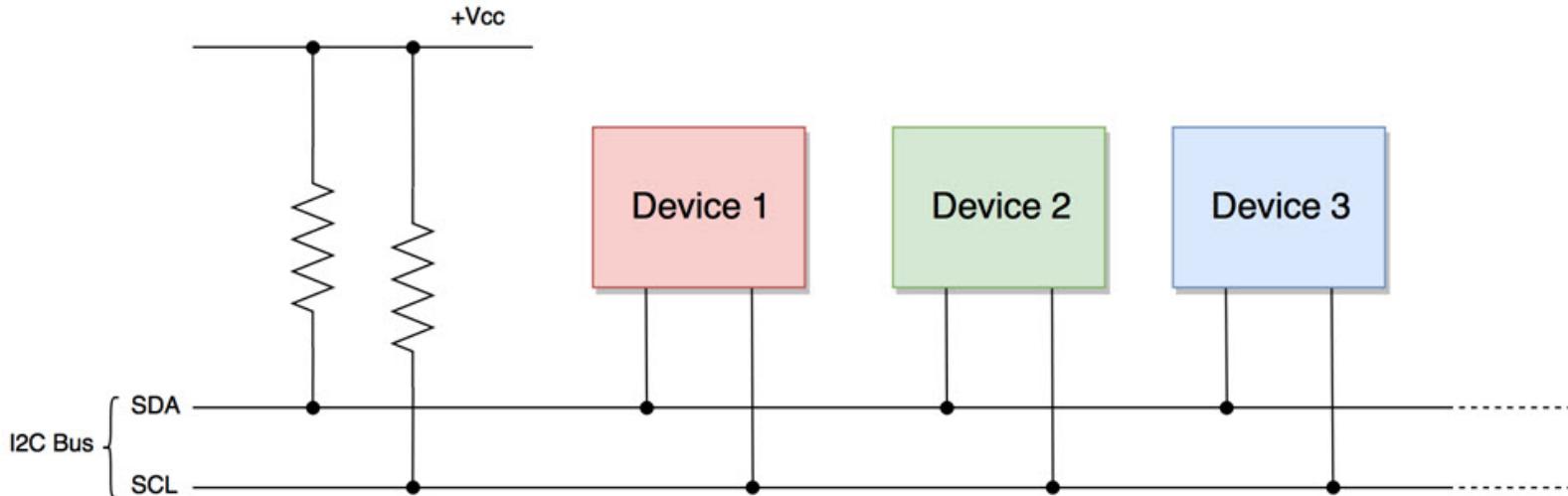
- Developed by Philips Semiconductors
- Transfer of data between a central processor and multiple ICs on the same circuit board using just two common wires
- Owing to its simplicity, it is widely adopted for communication between microcontrollers and sensor arrays, displays, IoT devices, EEPROMs etc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

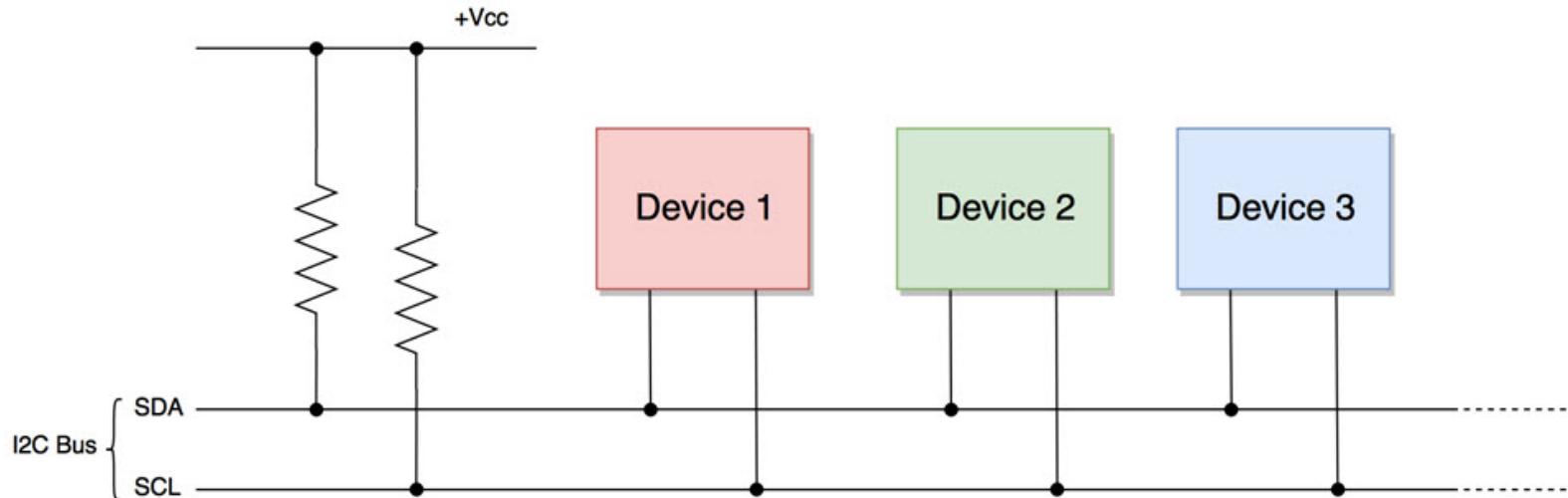
I2C Protocol - Hardware

- 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



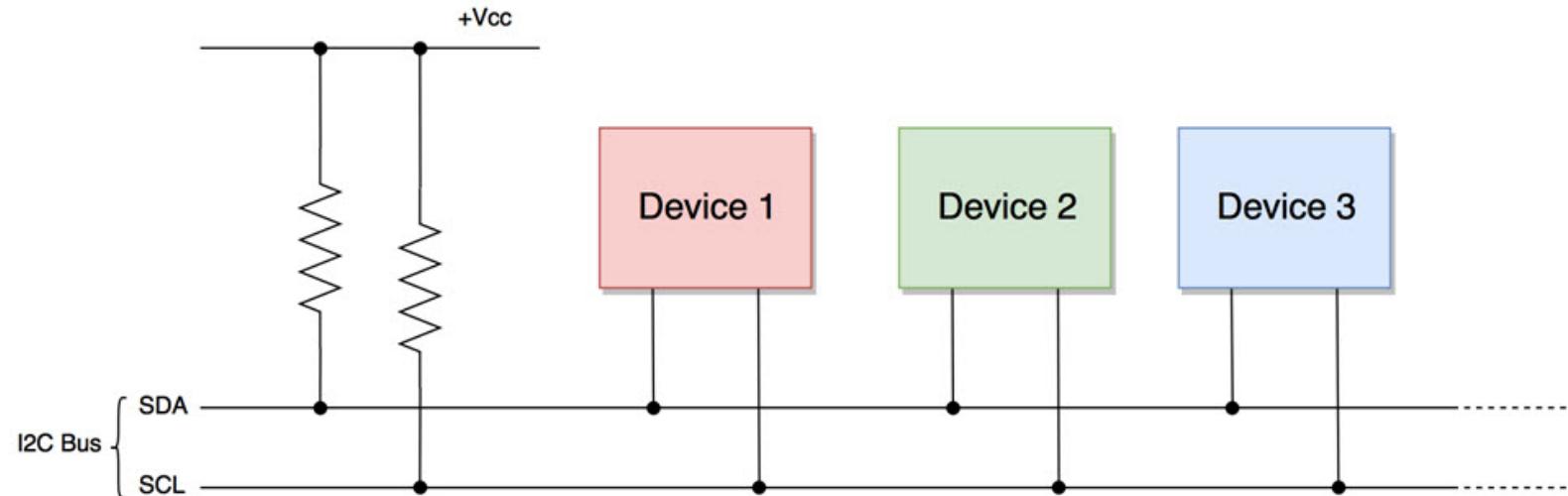
I2C Protocol - Hardware

- 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



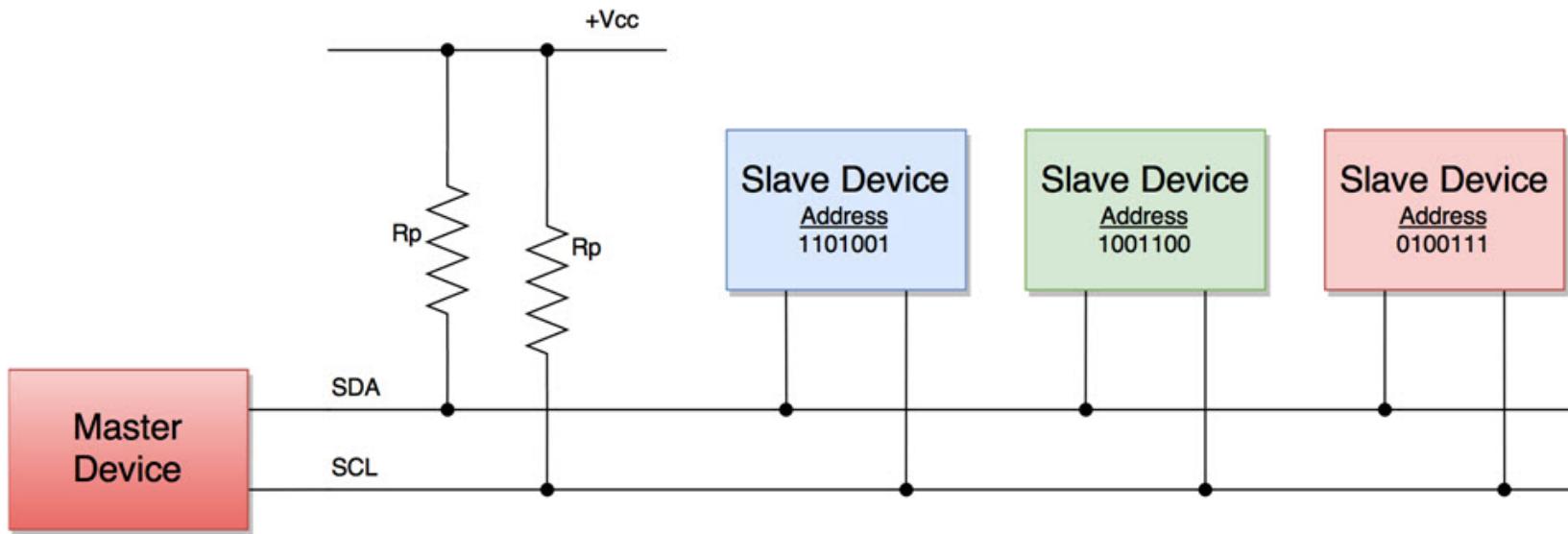
I2C Protocol - Hardware

- 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**



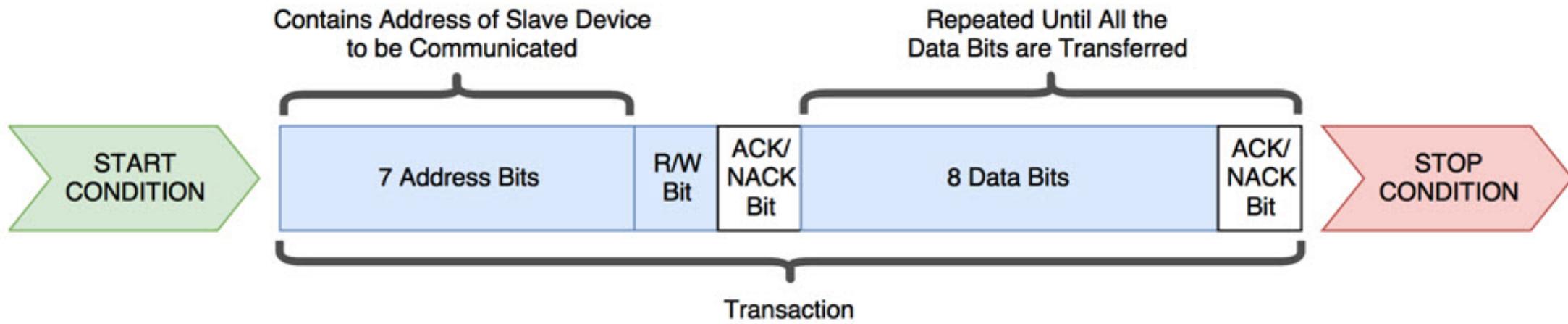
I2C Protocol - Hardware

- 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

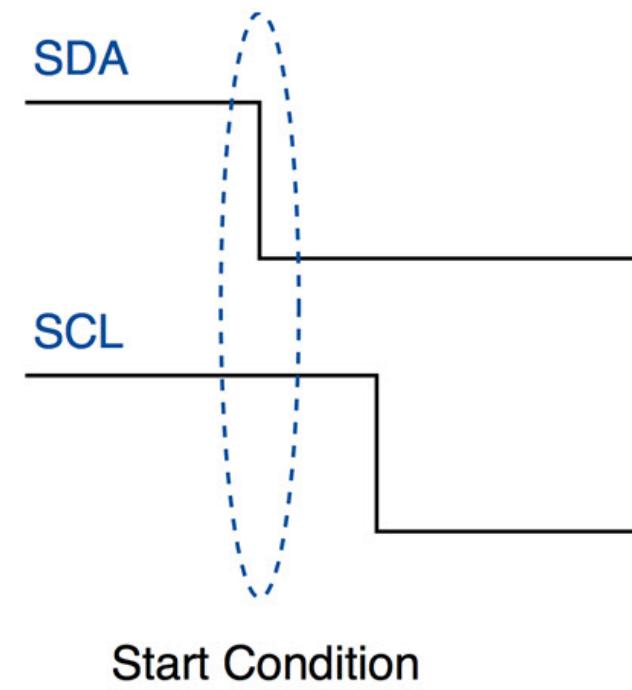
- Data is transferred between the master device and slave devices through a single SDA data line
- Via patterned sequences of 0's and 1's (bits)
- Each sequence of 0's and 1's is termed as a transaction



I2C Protocol – Data Transfer

Start Condition

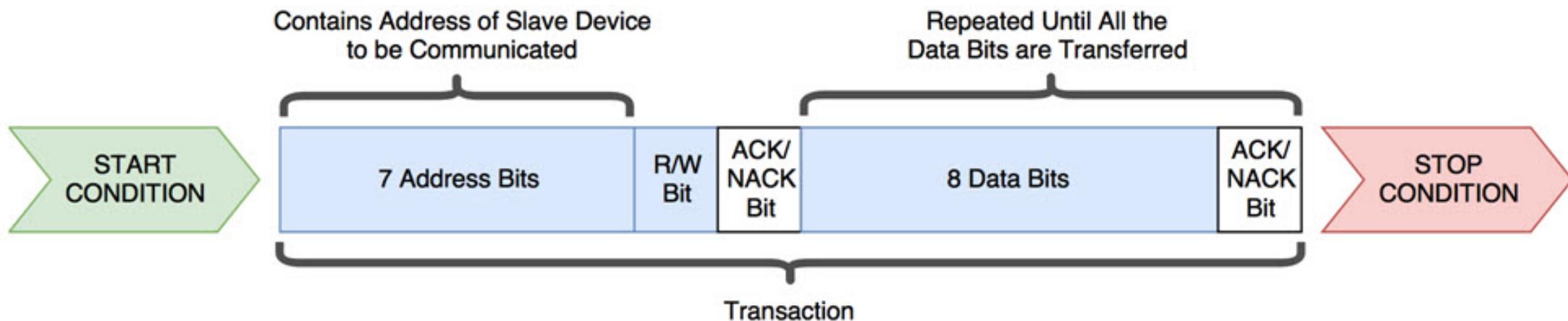
- When master device/IC decides to start a transaction
 - switches the SDA line from **high voltage to a low voltage**
 - before the SCL line switches from **high to low**



I2C Protocol – Data Transfer

Address Block

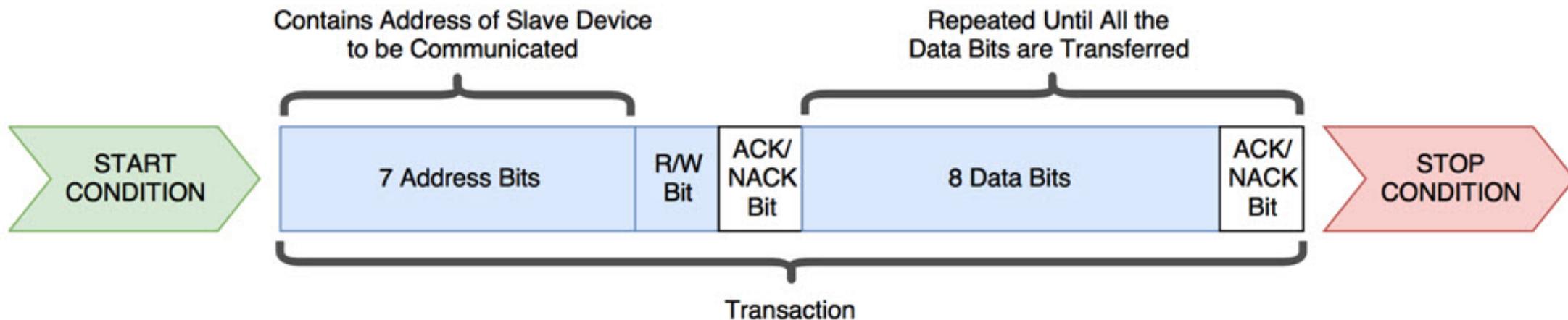
- 7 bits and are filled with the address of slave device to/from which the master device needs send/receive data
- All the slave devices on the I2C bus compare these address bits with their address



I2C Protocol – Data Transfer

Read/Write Bit

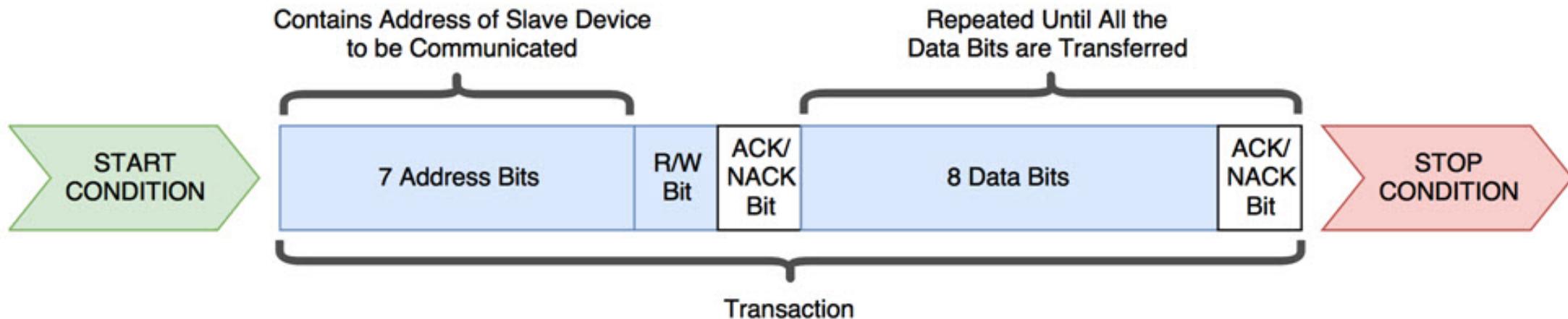
- This bit specifies the direction of data transfer
- Master device/IC needs to send data to a slave device – 0
- Master device/IC needs to receive data to a slave device – 1



I2C Protocol – Data Transfer

ACK/NACK Bit

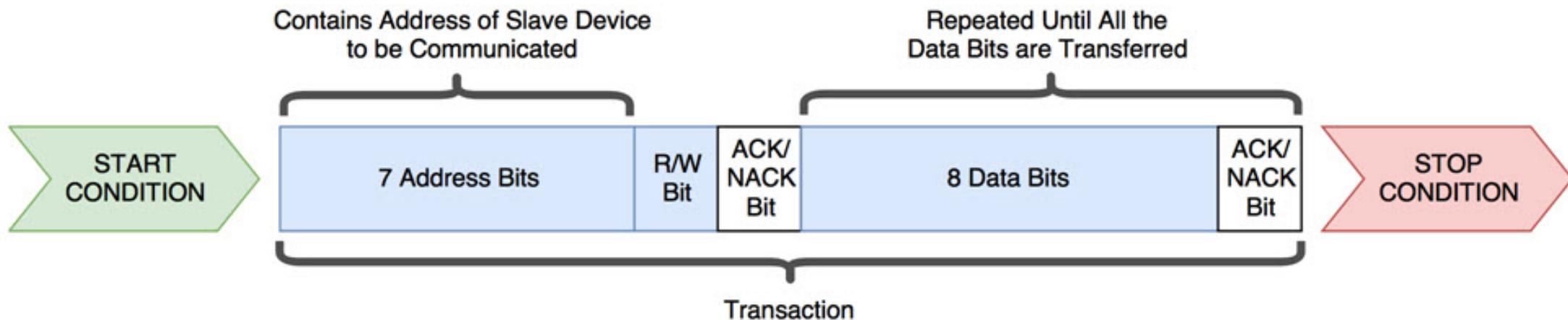
- If the address of any slave device = address broadcasted by the master device, the value of this bit is set to '0' by the slave device
- Else – 1(Default)



I2C Protocol – Data Transfer

Data Block

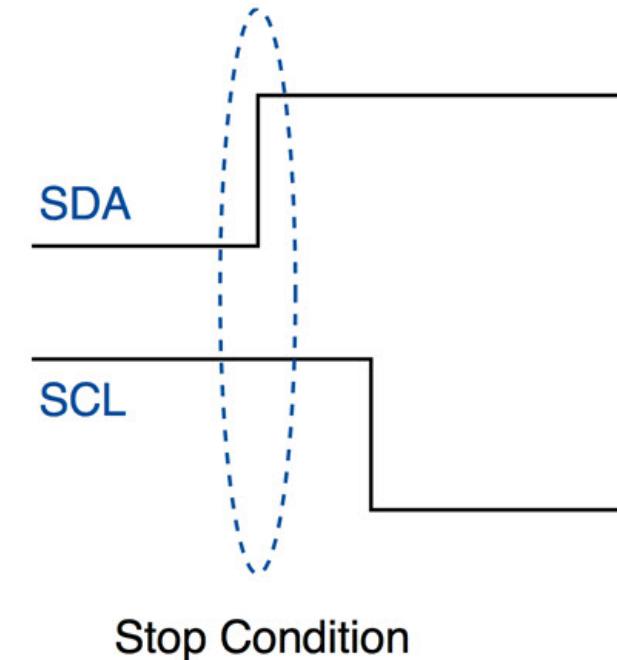
- 8 bits
- This block is followed by an ACK/NACK bit
- 0 by the receiver – Receives data, else stays at 1 (Default)



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**



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

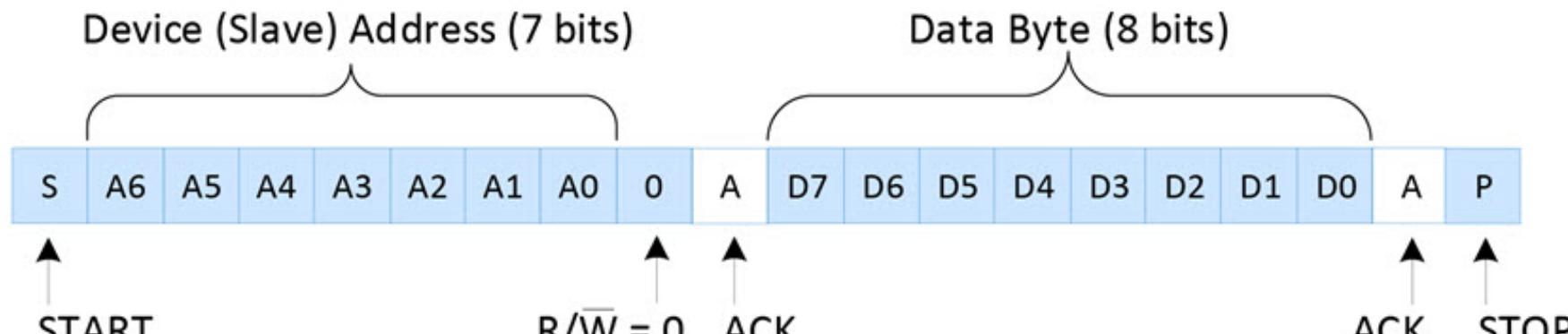
Write into Slave device



Master Controls SDA Line



Slave Controls SDA Line



I2C Protocol – Data Transfer

Reading from Slave device

 Master Controls SDA Line

 Slave Controls SDA Line

