EMBEDDED SYSTEMS (UNIT-III)

INTRODUCTION TO EMBEDDED SYSTEM

- An embedded system is a system that has software embedded into computer-hardware, which makes a system dedicated for an application (s) or specific part of an application or product or part of a larger system.
- An embedded system is one that has a dedicated purpose software embedded in a computer hardware.
- It is a dedicated computer based system for an application(s) or product. It may be an independent system or a part of large system. Its software usually embeds into a ROM (Read Only Memory) or flash.
- It is any device that includes a programmable computer but is not itself intended to be a general purpose computer.
- Embedded Systems are the electronic systems that contain a microprocessor or a microcontroller, but we do not think of them as computers – the computer is hidden or embedded in the system.

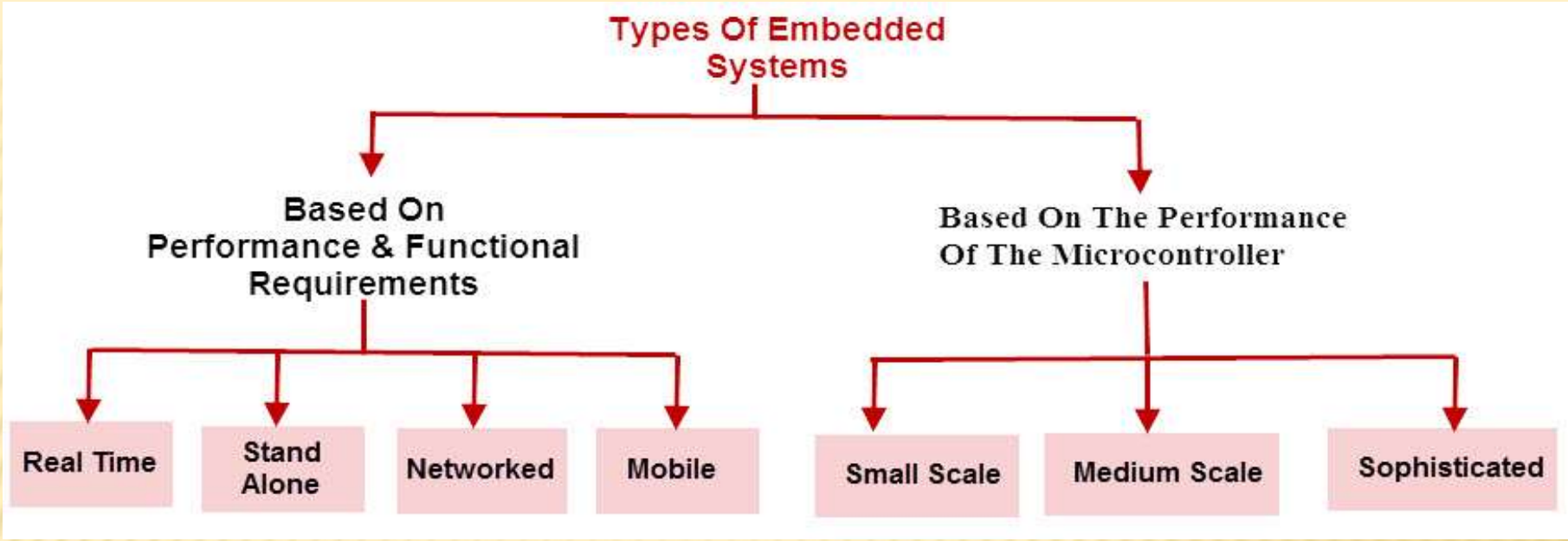

Examples of Embedded Systems
www.TheEngineeringProjects.com

EMBEDDED SYSTEMS VS GENERAL COMPUTING SYSTEMS

Criteria	General Purpose Computer	Embedded System
Contents	A system which is a combination of a generic hardware and a General Purpose Operating System for executing a variety of applications	A system which is a combination of special purpose hardware and embedded OS/firmware for executing a specific set of applications.
OS	It contains a general purpose operating system (GPOS)	It may or not contain an operating system for functioning
Alterations	Alterations Applications are alterable by the user.	Applications are not-alterable by the user
Key factors	Performance is key factor	Application specific requirements are key factors
Power Consumption	More	Less
Response Time	Not Critical	Critical for some applications
Execution	Need not be deterministic	Deterministic for certain types of ES like ‘Hard Real Time’ systems.

HISTORY AND CLASSIFICATION OF EMBEDDED SYSTEMS

- In the earliest years of computers in 1930 – 40s, computers were sometimes dedicated to a single purpose task.
- One of the first recognizably modern embedded system was the Apollo Guidance Computer, developed by Charles Stark Draper at the MIT Instrumentation Laboratory.
- Since these early applications in the 1960s, embedded systems have come down in price and there has been a dramatic rise in processing power and functionality. The first microprocessor for example, the Intel 4004 was designed for calculators and other small systems but still required many external memory and support chips.
- By the mid-1980s, most of the common previously external system components had been integrated into the same chip as the processor and this modern form of the microcontroller allowed an even more widespread use, which by the end of the decade were the norm rather than the exception for almost all electronics devices



Stand Alone Embedded Systems

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

Real Time Embedded Systems

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

Networked Embedded Systems

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

Mobile Embedded Systems

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

Small Scale Embedded Systems

These types of embedded systems are designed with a single 8 or 16-bit microcontroller, that may even be activated by a battery. For developing embedded software for small scale embedded systems, the main programming tools are an editor, assembler, cross assembler and integrated development environment (IDE). IDE will contains an EDITOR,COMPLIER,LINKER,DEBUGGER,SIMULATOR ,etc. IDEs are different for different family of processors/controllers. For Example KEIL MICRO VISION3 IDE used for all family members of 8051 microcontroller, since it contains compiler C51

Medium Scale Embedded Systems

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

Sophisticated Embedded Systems

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

MAJOR APPLICATION AREAS

1. **Consumer electronics:** Camcorders, cameras, etc
2. **Household appliances:** Television, DVD players, washing machine, fridge, microwave oven, etc.

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

PURPOSE OF EMBEDDED SYSTEMS

1) Data Collection/Storage/Representation

- Embedded systems designed for the purpose of data collection performs acquisition of data from the external world.
- Data collection is usually done for storage, analysis, manipulation and transmission.
- Data can be either analog (continuous) or digital (discrete).
- Embedded systems with analog data capturing techniques collect data directly in the form of analog signal whereas embedded systems with digital data collection mechanism converts the analog signal to the digital signal using analog to digital (A/D) converters and then collects the binary equivalent of the analog data.
- If the data is digital, it can be directly captured without any additional interface by digital embedded systems.

❖ A digital camera is a typical example of an embedded system with data collection/storage/representation of data.

❖ Images are captured and the captured image may be stored within the memory of the camera. The captured image can also be presented to the user through a graphic LCD unit.

2)Data Communication

❖ Embedded data communication systems are deployed in applications from complex satellite communication systems to simple home networking systems.

❖ The transmission is achieved either by a wire-line medium or by a wire-less medium. Data can either be transmitted by analog means or by digital means

❖ The data collecting embedded terminal itself can incorporate data communication units like Wireless modules (Bluetooth, ZigBee, Wi-Fi, EDGE, GPRS, etc.) or wire-line modules (RS-232C, USB, TCP/IP, PS2,etc).

❖ Network hubs, routers, switches, etc. are typical examples of dedicated data transmission embedded systems.

3)Data (Signal) Processing

- Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, synthesis, audio video codec, transmission applications, etc.
- A digital hearing aid is a typical example of an embedded system employing data processing.
- Digital hearing aid improves the hearing capacity of hearing impaired person

4) Monitoring

- Almost all embedded products coming under the medical domain are with monitoring functions only.
- Electro cardiogram machine (ECG) is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.
- Other examples with monitoring function are digital CRO, digital multimeters, and logic analyzers.

5) Control

- A system with control functionality contains both sensors and actuators.
- Sensors are connected to the input port for capturing the changes in environmental variable or measuring variable.
- The actuators connected to the output port are controlled according to the changes in the input variable.
- Air conditioner system used in our home to control the room temperature to a specified limit is a typical example for ES for CONTROL purpose.

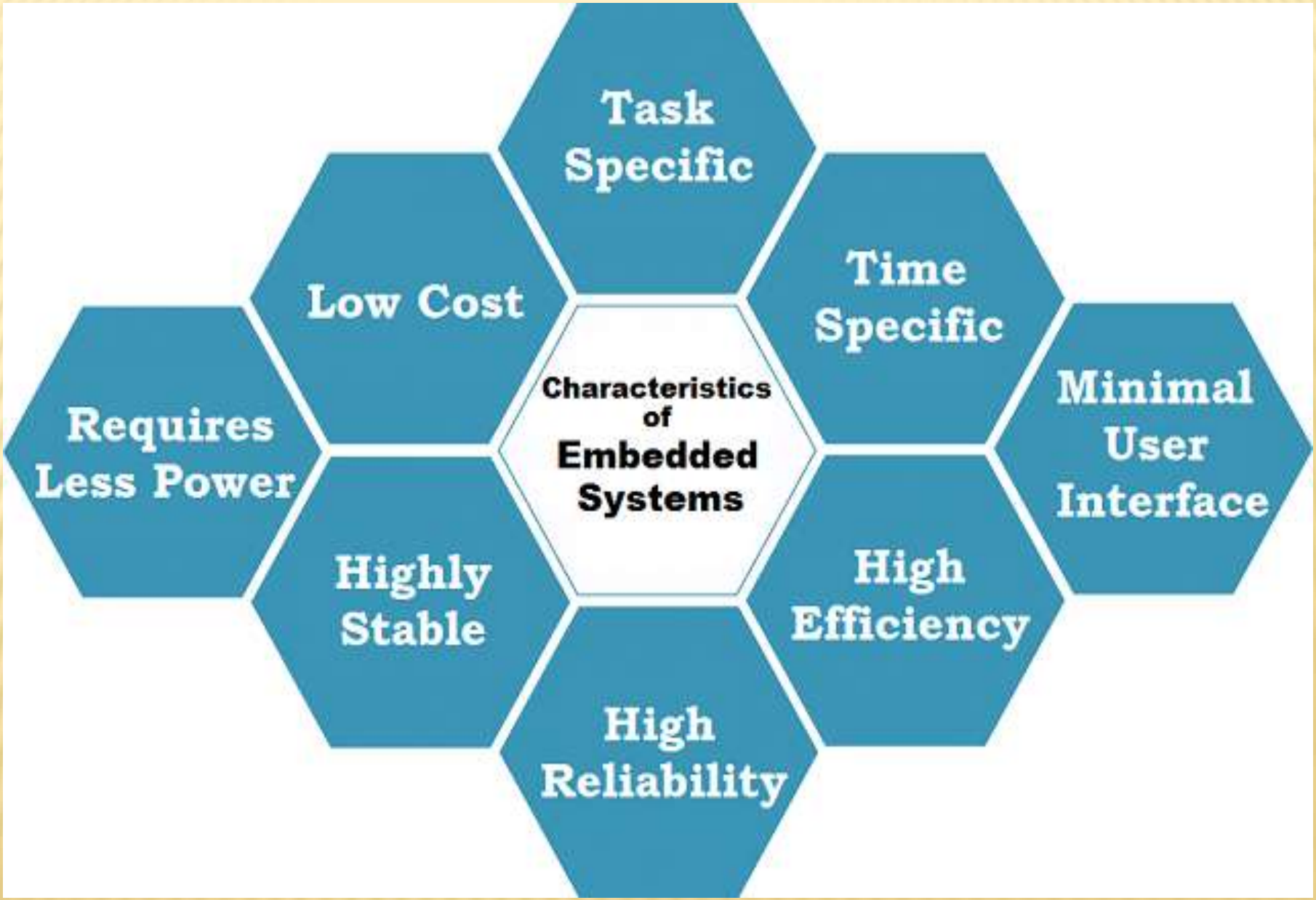
6) Applications specific user interface

- Buttons, switches, keypad, lights, speakers, display units, etc. are application-specific user interfaces.
- Mobile phone is an example of application specific user interface.
- In mobile phone the user interface is provided through the keypad, graphic LCD module, system speaker, vibration alert, etc

CHARACTERISTICS OF EMBEDDED SYSTEMS

- All Embedded Systems are task specific. They do the same task repeatedly /continuously over their lifetime. An mp3 player will function only as an mp3 player.
- Embedded systems are created to perform the task within a certain time frame. It must therefore perform fast enough. A car's brake system, if exceeds the time limit, may cause accidents.
- They have minimal or no user interface (UI). A fully automatic washing machine works on its own after the programme is set and stops once the task is over.
- Some embedded systems are designed to react to external stimuli and react accordingly. A thermometer, a GPS tracking device.
- Embedded systems are built to achieve certain efficiency levels. They are small sized, can work with less power and are not too expensive

- Embedded systems are built to achieve certain efficiency levels. They are small sized, can work with less power and are not too expensive.
- Embedded systems cannot be changed or upgraded by the users. Hence, they must rank high on reliability and stability. They are expected to function for long durations without the user experiencing any difficulties.
- Microcontroller or microprocessors are used to design embedded systems.
- Embedded systems need connected peripherals to attach input & output devices.
- The hardware of an embedded-system is used for security and performance. The Software is used for features.


Task Specific
Low Cost
Time Specific
Requires Less Power
Characteristics of Embedded Systems
Minimal User Interface
Highly Stable
High Efficiency
High Reliability

QUALITY ATTRIBUTES OF EMBEDDED SYSTEMS

The **operational quality attributes** represent the relevant quality attributes related to the embedded system when it is in the operational mode or 'online' mode. The important quality attributes coming under this category are listed below:

a) Response
b)Throughput
c) Reliability
d) Maintainability
e) Security
f) Safety

A) Response

- Response is a measure of quickness of the system.
- It gives you an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.
- Ex. An embedded system deployed in flight control application should respond in a Real Time manner.
- Any response delay in the system will create potential damages to the safety of the flight as well as the passengers.
- It is not necessary that all embedded systems should be Real Time in response.
- For example, the response time requirement for an electronic toy is not at all time-critical

b) Throughput

- Throughput deals with the efficiency of system.
- It can be defined as rate of production or process of a defined process over a stated period of time.
- The rates can be expressed in terms of units of products, batches produced, or any other meaningful measurements.
- In case of card reader like the ones used in buses, throughput means how much transactions the Reader can perform in a minute or hour or day.
- Throughput is generally measured in terms of ‘Benchmark’. A ‘Benchmark’ is a reference point by which something can be measured.
- Benchmark can be a set of performance criteria that a product is expected to meet or a standard product that can be used for comparing other products of the same product line.

c)Reliability

- Reliability is a measure of how much percentage you rely upon the proper functioning of the system or what is the % susceptibility of the system to failure.
- Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR) are the terms used in defining system reliability.
- MTBF gives the frequency of failures in hours/weeks/months.
- MTTR specifies how long the system is allowed to be out of order following a failure.
- For an embedded system with critical application need, it should be of the order of minutes.

d)Maintainability

- Maintainability deals with support and maintenance to the end user or client in case of technical issues and product failures or on the basis of a routine system check up.
- Reliability and maintainability are considered as two complementary disciplines. A more reliable system means a system with less corrective maintainability requirements and vice versa.

Maintainability can be classified into two types:

1. Scheduled or Periodic Maintenance (Preventive Maintenance) An inkjet printer uses ink cartridges, which are consumable components and as per the printer manufacturer the end use should replace the cartridge after each ‘n’ number of printouts to get quality prints.
2. Maintenance to Unexpected Failures (Corrective Maintenance) If the paper feeding part of the printer fails the printer fails to print and it requires immediate repairs to rectify this problem. Hence it is obvious that maintainability is simply an indication of the availability of the product for use. In any embedded system design, the ideal value for availability is expressed as

$$\boldsymbol{Ai = MTBF/(MTBF + MTTR)}$$

Where Ai=Availability in the ideal condition,

MTBF=Mean Time Between Failures,

MTTR= Mean Time To Repair

e)Security

- ‘Confidentially’, ‘Integrity’, and ‘Availability’ are three major measures of information security.
- ‘Confidentially’ deals with the protection of data and application from unauthorized disclosure.
- ‘Integrity’ deals with the protection of data and application from unauthorized modification.
- ‘Availability’ deals with protection of data and application from unauthorized users.
- Certain embedded systems have to make sure they conform to the security measures.
- Ex. An electronic safety Deposit Locker can be used only with a pin number like a password.

f)Safety: Safety deals with the possible damages that can happen to the operators, public and the environment due to the breakdown of an embedded system or due to the emission of radioactive or hazardous materials from the embedded products.

- The breakdown of an embedded system may occur due to a hardware failure or a firmware failure.
- Safety analysis is a must in product engineering to evaluate the anticipated damages and determine the best course of action to bring down the consequences of the damages to an acceptable level.

NON OPERATIONAL ATTRIBUTES

- The quality attributes that needs to be addressed for the product 'not' on the basic of operational aspects are grouped under this category. The important quality attributes coming under this category are listed below:

a)Testability & Debug-ability
b) Evolvability
c) Portability
d) Time to prototype and market
e) Per unit and total cost

a)Testability & Debug-ability

- Testability deals with how easily one can test his/her design, application and by which means he/she can test it.
- For an embedded product, testability is applicable to both the embedded hardware and firmware.
- Debug-ability is a means of debugging the product as such for figuring out the probable sources that create unexpected behaviour in the total system.
- Debug-ability has two aspects in the embedded system development context, namely, hardware level debugging and firmware level debugging.
- Hardware debugging is used for figuring out the issues created by hardware problems whereas firmware debugging is employed to figure out the probable errors that appear as a result of flaws in the firmware.

b)Evolvability

- Evolvability is a term which is closely related to Biology.
- Evolvability is referred as the non-heritable variation.
- For an embedded system, the quality attribute ‘Evolvability’ refers to the ease with which the embedded product (including firmware and hardware) can be modified to take advantage of new firmware or hardware technologies.

C)Portability

- Portability is a measure of ‘system independence’.
- An embedded product can be called portable if it is capable of functioning in various environments, target processors/controllers and embedded operating systems.
- A standard embedded product should always be flexible and portable

d)Time-to-Prototype and Market

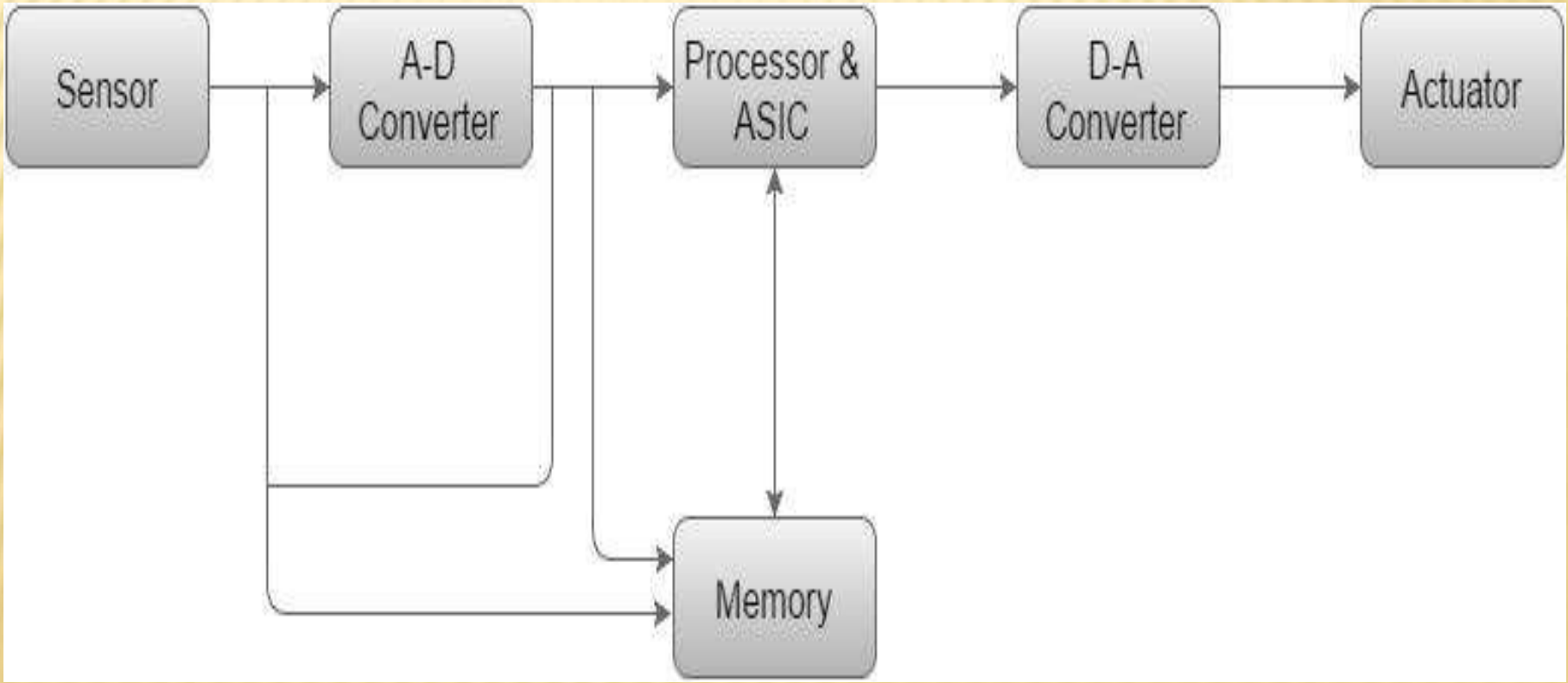
- Time-to-market is the time elapsed between the conceptualization of a product and the time at which the product is ready for selling (for commercial product) or use (for non-commercial products).
- The commercial embedded product market is highly competitive and time to market the product is a critical factor in the success of a commercial embedded product.
- Product prototyping helps a lot in reducing time-to-market.

e)Per Unit Cost and Revenue

- Cost is a factor which is closely monitored by both end user (those who buy the product) and product manufacturer (those who build the product).
- Cost is a highly sensitive factor for commercial products.
- Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.
- When the product is introduced in the market, for the initial period the sales and revenue will be low.

- There won’t be much competition when the product sales and revenue increase.
- During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.

BASIC STRUCTURE OF AN EMBEDDED SYSTEM



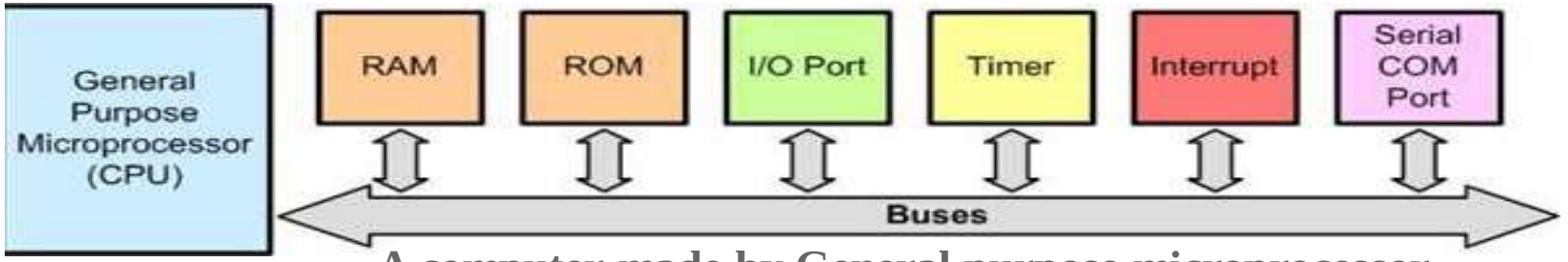
EMBEDDED SYSTEM DESIGN
UNIT 1: ARM 32 BIT MCU'S

M.A. HIMAYATH SHAMSHI
Dept Of Electronics and communication

Why ARM 32 Bit MCU's?

- **Arm technology is in 190 billion devices and 95% of today's smart phones**
- ARM stands for Advanced RISC (reduced instruction set computer) machine.
- ARM started life as part of Acorn makers of the BCC computer and now designs chips for Apple iPad.
- The first ARM was established at Cambridge University in 1978.
- The Acorn group computers have developed the first ARM commercial RISC processor in 1985.
- ARM was founded and very popular in 1990. The ARM using more than 98% of the mobile phones in 2007 and 10 billion processors are shipped in 2008. ARM is the latest technology which replaced by microcontrollers and microprocessors. Basically ARM is a 16 bit/ 32 bit Processors or Controllers.
- ARM is the heart of advanced digital products like mobile phones automotive systems digital cameras and home networking and wireless technologies.
- ARM is the most popular processor, particularly used in portable devices due to its low power consumption and reasonable performance.
- ARM has got better performance when compared to other processors.
- The ARM processor is basically consisting of low power consumption and low cost. It is very easy to use ARM for quick and efficient application developments so that is the main reason why ARM is most popular.

The History of ARM & Microcontrollers



A computer made by General purpose microprocessor

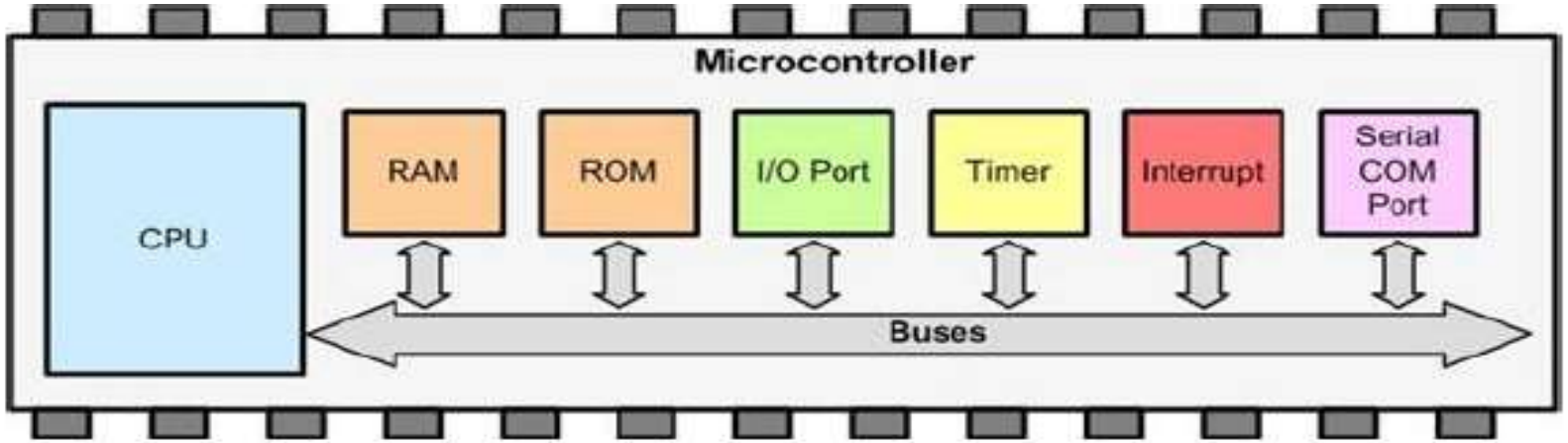


Figure 1- 2: Simplified View of the Internal Parts of Microcontrollers (SOC)

- Herein, the different parts of a system, including CPU, RAM, ROM, and I/Os, were put together on a single IC chip and it was called *microcontroller*
- Cheaper & Small and widely used in many devices
- SOC (System on Chip) and MCU (Micro Controller Unit) are other names used to refer to microcontrollers

Types of Computers

Points	Desktop Computers	Servers	Embedded Systems
Examples	PCs, tablets, desktop computers and Types of Computers laptops, are general purpose computers	used as web hosts, database servers, and in any application in which we need to process a huge amount of data such as weather Forecasting	the Kindle, digital camera, vacuum cleaner, mp3 player, mouse, keyboard, and printer
Uses	used to play games, read and edit articles, and do any other task just by running the proper application programs	Used in any application in which we need to process a huge amount of data such as weather forecasting	In embedded system devices, the software application and hardware are embedded together and are **designed to do a specific task**

Points	Desktop Computers	Servers	Embedded Systems
Made of	Microprocessor	Servers are made of microprocessors but, multiple processors are usually used in each server. Both servers and desktop computers are connected to several embedded system devices such as mouse, keyboard, disk controller, Flash stick memory and so on	In most cases embedded systems run a fixed program (software application) and contain a microcontroller

Brief History of ARM

- **The ARM came out of a company called Acorn Computers in United Kingdom in the 1980s**
- Professor Steve Furber of Manchester University worked with Sophie Wilson to define the ARM architecture and instructions
- The VLSI Technology Corp. produced the first ARM chip in 1985 for Acorn Computers and was designated as Acorn RISC (Reduced Instruction Set Architecture) Machine (ARM)
- Unable to compete with x86 (8088, 80286, 80386, …) PCs from IBM and other personal computer makers, the Acorn was forced to push the ARM chip into the single-chip microcontroller market for embedded products
- That is when Apple Corp. got interested in using the ARM chip for the PDA (personal digital assistants) products
- **This renewed interest in the chip led to the creation of a new company called**

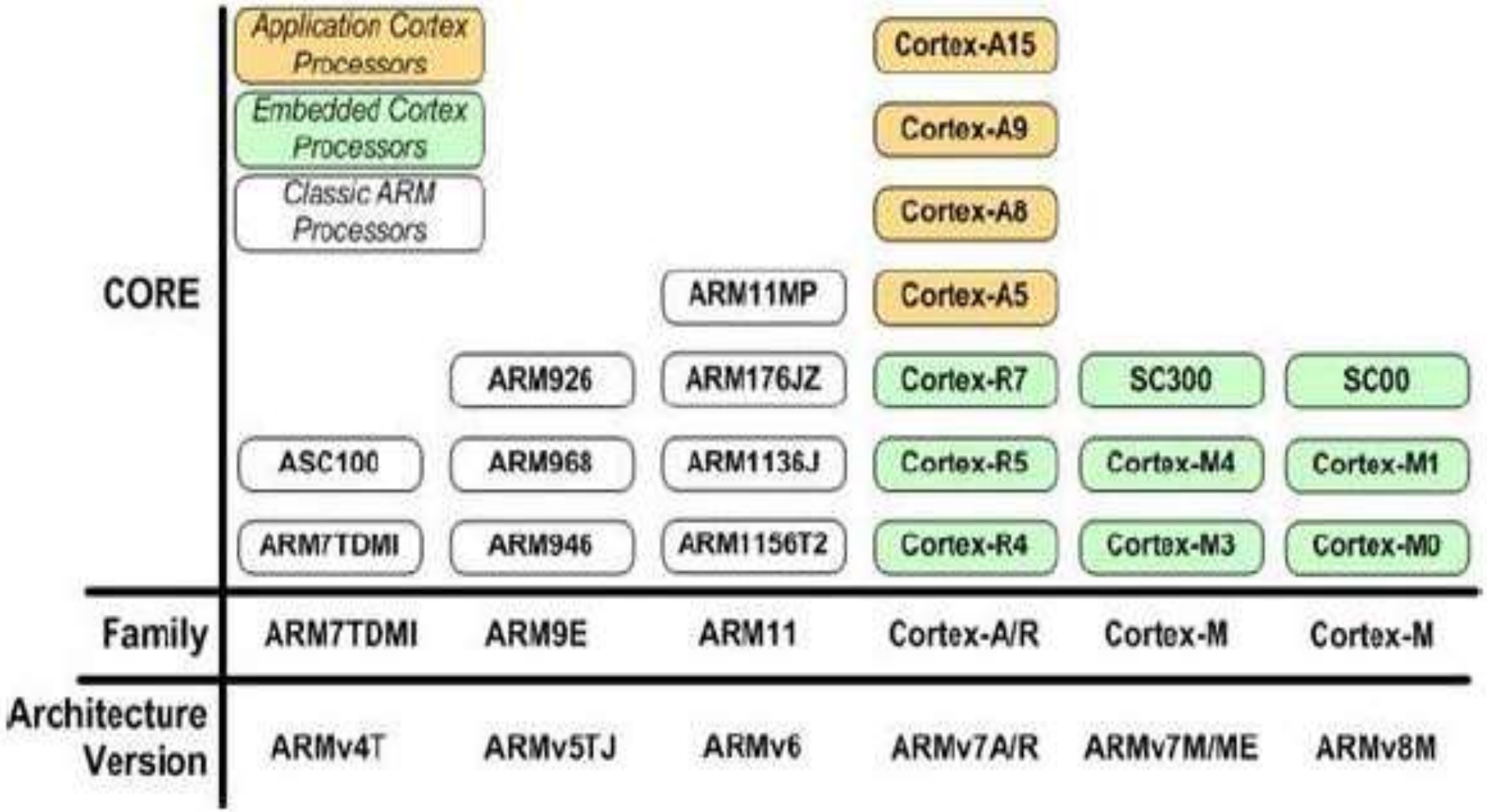
ARM (Advanced RISC Machine)

- ARM is a performance based processor

ARM Nomenclature

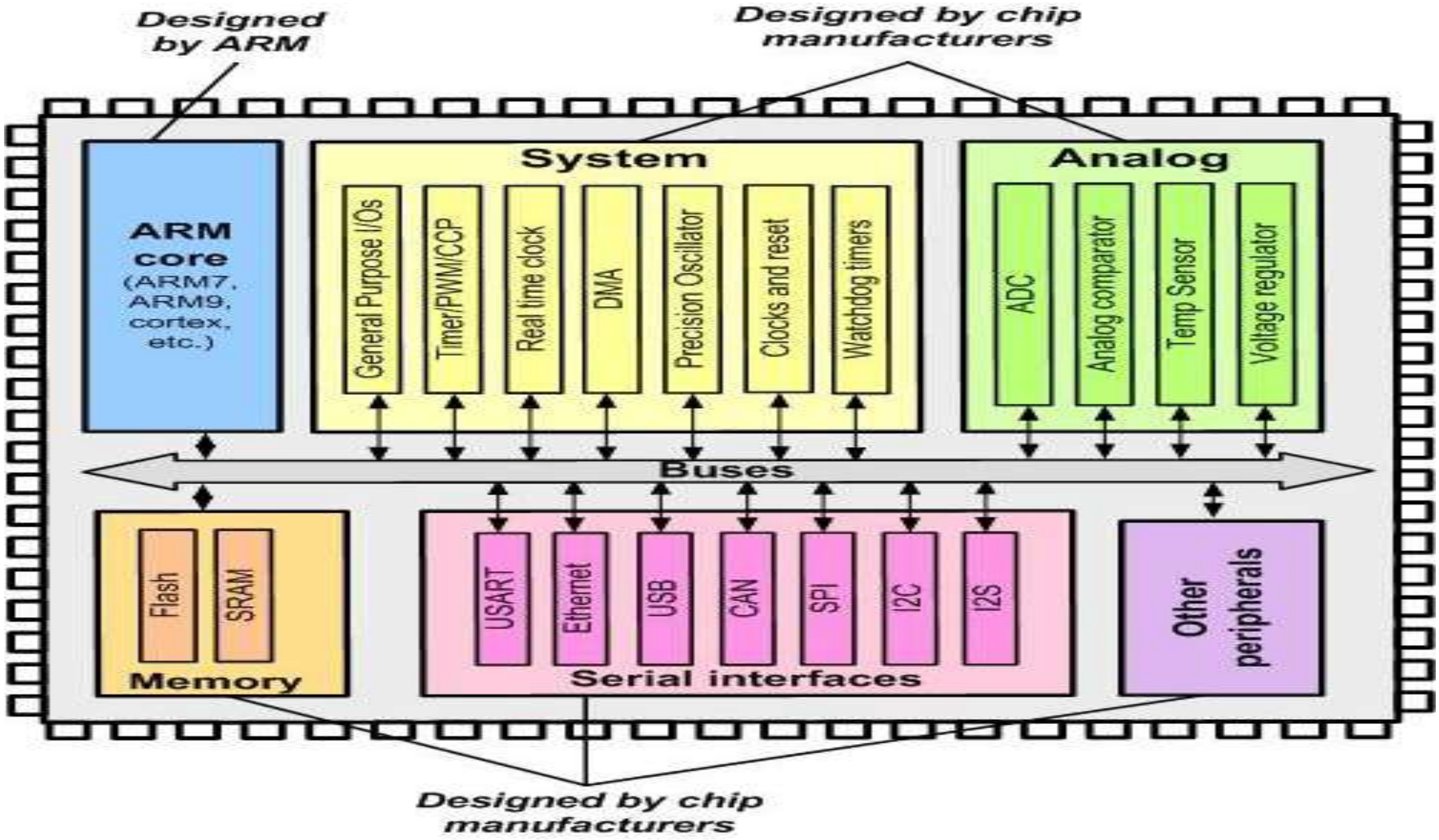
- A R M x y z T D M I E J F S (Example: ARM7-TDMI-S)
  - x - Series
  - y - MMU
  - z - Cache
  - T - Thumb
  - D - Debugger
  - M - Multiplier
  - I - Embedded In-Circuit Emulator (ICE) macrocell
  - E - Enhanced Instructions for DSP
  - J - JAVA acceleration by Jazelle
  - F - Floating-point
  - S - Synthesizable version

ARM FAMILY AND ARCHITECTURE



ARM : ONE CPU, MANY PERIPHERALS

- ARM has defined the details of architecture, registers, instruction set, memory map , and timing of the ARM CPU and holds the copyright to it.
- The various design houses and semiconductor manufacturers license the IP (intellectual property) for the CPU and can add their own peripherals as they please.
- It is up to the licensee (design houses and semiconductor manufactures) to define the details of peripherals such as I/O ports, serial port UART, timer, ADC, SPI, DAC, I2C, and so on.
- As a result while the CPU instructions and architecture are same across all the ARM chips made by different vendors, their peripherals are not compatible.
- That means if you write a program for the serial port of an ARM chip made by TI (Texas Instrument), the program might not necessarily run on an ARM chip sold by NXP.
- This is the only drawback of the ARM microcontroller.
- The good news is the IDE (integrated development environment) such as Keil (see www.keil.com)or IAR (see www.IAR.com) do provide peripheral libraries for chips from various vendors and make the job of programming the peripherals much easier.
- It must be noted that in recent years ARM provides the IP for some peripherals such as UART and SPI, but unlike the CPU architecture, its adoption is not mandatory and it is up to the chip manufacturer whether to adopt it or not.
- This is in contrast to the Cold fire microcontroller from Freescale, in which the Freescale defines the architecture and peripherals, fabricates, sells, and supports the chip.


Designed by ARM
Designed by chip manufacturers
ARM core (ARM7, ARM9, cortex, etc.)
System
General Purpose I/Os
Timer/PWM/CCP
Real time clock
DMA
Precision Oscillator
Clocks and reset
Watchdog timers
Analog
ADC
Analog comparator
Temp Sensor
Voltage regulator
Buses
Flash
SRAM
Memory
USART
Ethernet
USB
CAN
SPI
I2C
I2S
Serial interfaces
Other peripherals
Designed by chip manufacturers

ARM VENDORS

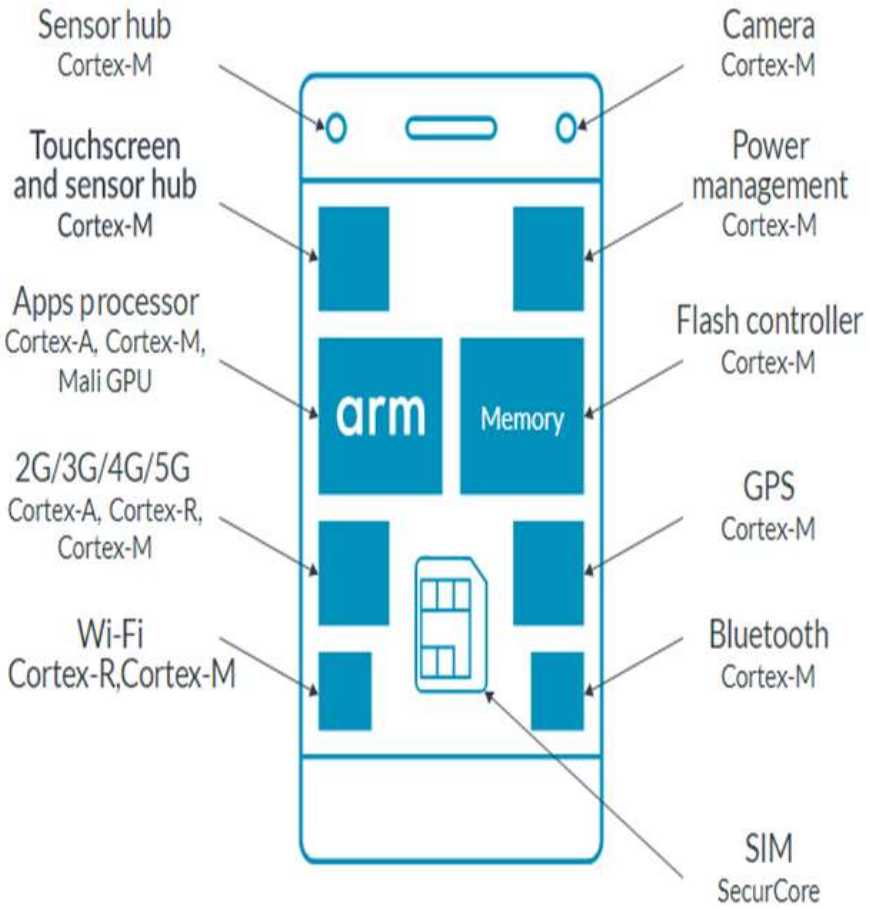
Actel	Analog Devices	Atmel
Broadcom	Cypress	Ember
Dust Networks	Energy	Freescale
Fujitso	Nuvoton	NXP
Renesas	Samsung	ST
Toshiba	Texas Instruments	Triad Semiconductor

ARM Architecture

The Arm architecture is one of the most popular processor architectures in the world. Billions of Arm-based devices are shipped every year.

The following table describes the three architecture profiles: A, R, and M:

A-Profile (Applications)	R-Profile (Real-Time)	M-Profile (Microcontroller)
High performance	Targeted at systems with real-time requirements	Small, highly power-efficient devices
Designed to run a complex operating system, such as Linux or Windows	Commonly found in networking equipment, and embedded control systems	Found at the heart of many IoT devices



This example smartphone contains the following processor types:

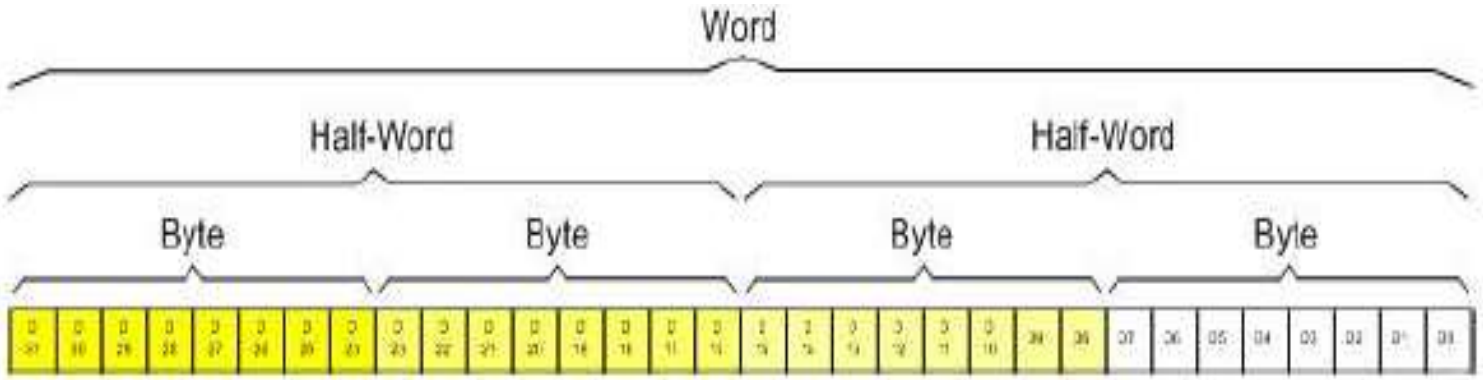
- An A-profile processor as the main CPU running a rich OS like Android
- A cellular modem, based on an R-profile processor, provides connectivity
- Several M-profile processors handle operations like system power management
- The SIM card uses Secure Core, an M-profile processor with additional security features. Secure Core processors are commonly used in smart cards
- Architecture means functional specification
- Arm Architecture implies functional specification of a processer i.e., how the processor will behave
- It is like a contract between hardware & the software. The architecture describes what functionality the software can rely on the hardware to provide

The architecture specifies:

Instruction set	The function of each instruction How that instruction is represented in memory (its encoding)
Register set	How many registers there are The size of the registers The function of the registers Their initial state
Exception model	The different levels of privilege The types of exceptions What happens on taking or returning from an exception
Memory model	How memory accesses are ordered How the caches behave, when and how software must perform explicit maintenance
Debug, trace, and profiling	How breakpoints are set and triggered What information can be captured by trace tools and in what format

Need to Study

- CPUs use registers to store data temporarily in arithmetic & logic operations.
- To program in Assembly language, it necessitates to understand the registers and architecture of a given CPU and the role they play in processing data
- All of ARM registers are 32-bit wide
- These range from the MSB (most-significant bit) D31 to the LSB (least-significant bit) D0
- With a 32-bit data type, any data larger than 32 bits must be broken into 32-bit chunks before it is processed
- size of the ARM is often referred as word. This is in contrast to x86 CPU in which word is defined as 16-bit.
- In ARM the 16-bit data is referred to as half-word. Therefore ARM supports byte, half-word (two byte), and word (four bytes) data types.



ARM POWERED PRODUCTS

RISC and CISC Processors:

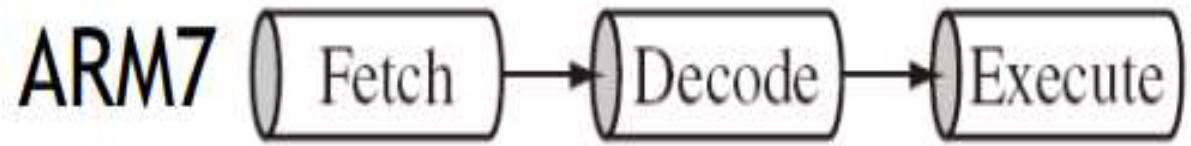
RISC	CISC
RISC stands for Reduced Instruction Set Computer	CISC stands for Complex Instruction Set Computer
Hardware plays major role in RISC processors	Software plays major role in CISC processors
RISC processors use single clock to execute an instruction	CISC processors use multiple clocks for execution.
Memory-to-memory access is used for data manipulations is RISC processors	intermediate registers are used for data manipulation
In RISC processors, single word instructions are given as inputs	In CISC processors, instructions of variable lengths are given as input, based upon the task to be performed
More lines of code and large memory footprint	High code density
Compact, uniform instructions and hence facilitate pipelining	Many addressing modes and long instructions
Allow effective compiler optimization	Often require manual optimization of assembly code for embedded systems
These machines provided a variety of instructions that may perform very complex tasks, such as string searching	These computers tended to provide somewhat fewer and simpler instructions.

Differences between Von Neumann and Harvard architecture:

VON NEUMANN(Princeton)	HARVARD ARCHITECTURE
Same memory holds data, instructions	Separate memories for data and instructions
A single set of address/data buses between CPU and memory	Two sets of address/data buses between CPU and memory
Single memory fetch operation	Harvard allows two simultaneous memory Fetches
The code is executed serially and takes more clock cycles	The code is executed in parallel
Not exactly suitable for DSP	Most DSPs use Harvard architecture for streaming data: • greater memory bandwidth; • more predictable bandwidth
There is no exclusive Multiplier	It has MAC (Multiply Accumulate)
No Barrel Shifter is there	Barrel Shifter help in shifting and rotating operations of the data
The programs can be optimized in lesser size	The program tend to grow big in size
Used in conventional processors found in PCs and Servers, and embedded systems with only control functions.	Used in DSPs and other processors found in latest embedded systems and Mobile communication systems, audio, speech, image processing systems

INTRODUCTION TO ARM

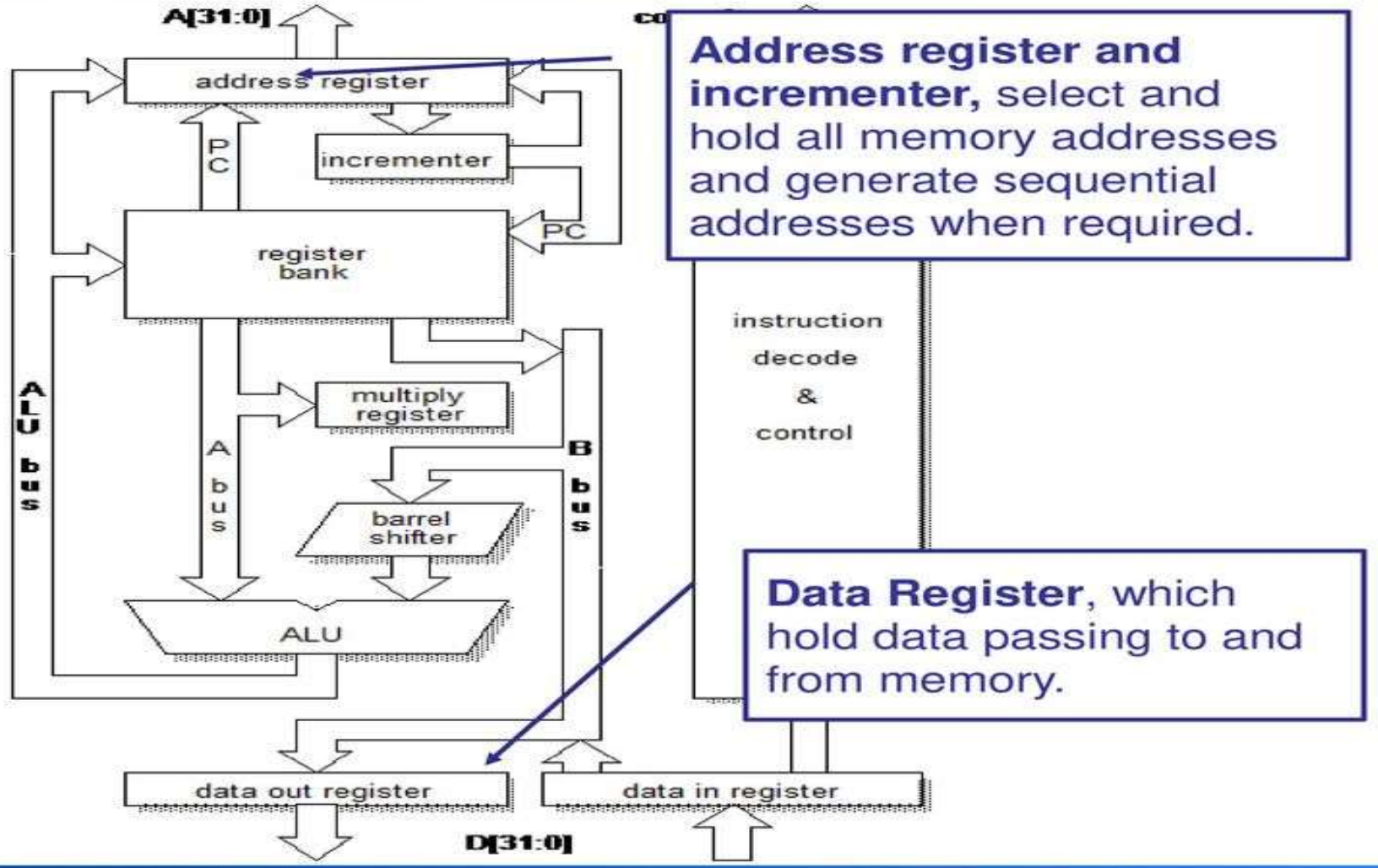
- 32 bit microcontroller
- 32 bit ALU
- 32 bit Data bus
- It is 4 times powerful than 8051
- It will do 32bit operations in one cycle
- 32 bit fixed instruction length (Most important characteristic of RISC Architecture)
- Most of ARM implements 3 – instruction sets
- 1) 32 – bit ARM instructions set
- 2) 16 – bit Thumb instruction set
- 3) 8 – bit Jazelle instruction set, specially used to implement the concepts of JAVA byte code
- 32 bit Address bus (it can address 4GB of memory i.e. 2^32 = 4GB)
- Van Neuman model (common memory for program and data)
- 3-stage pipeline



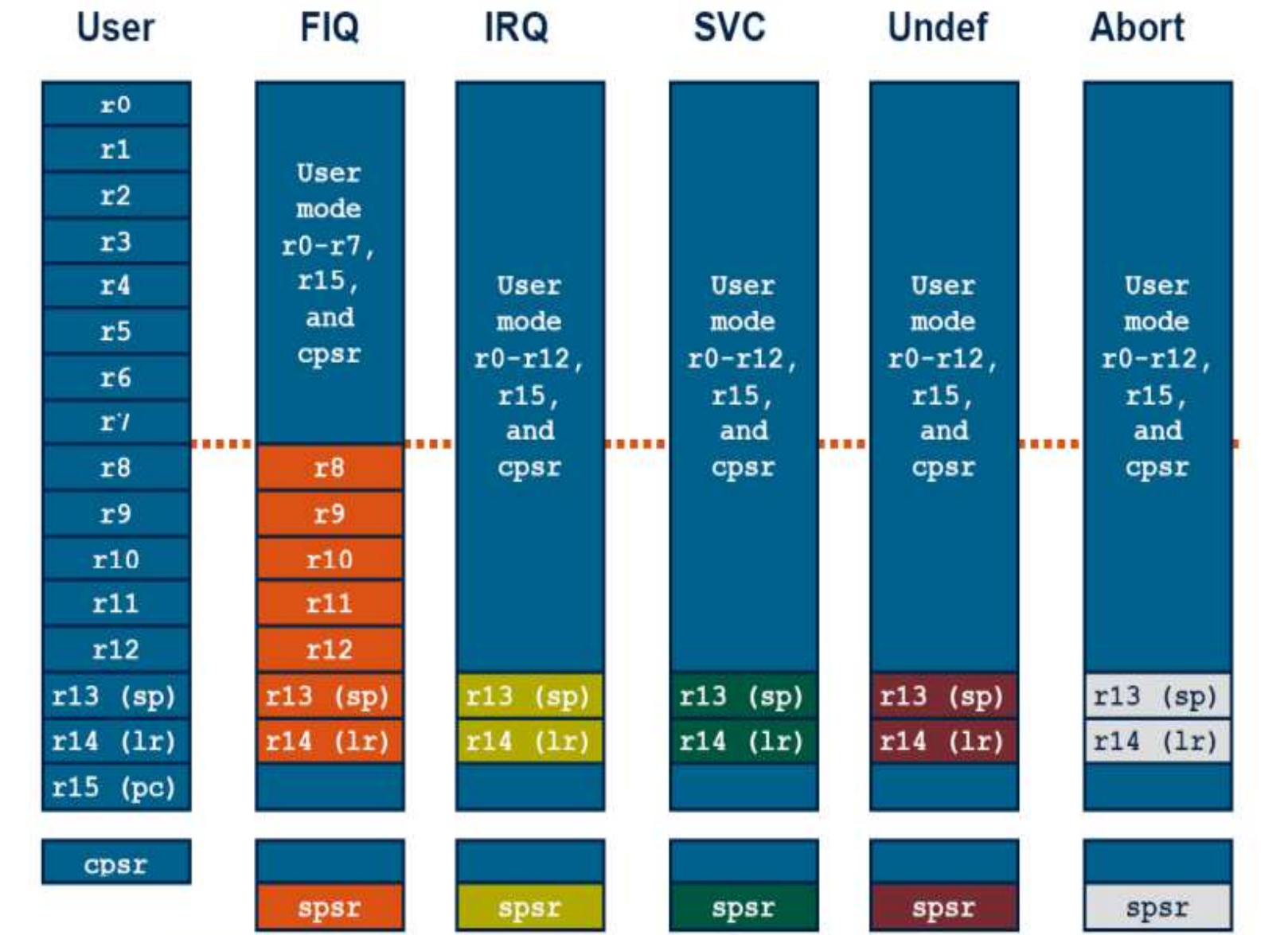
- 37 Registers of 32bits each, at a time 16 are available(R0 to R15)
- Load – Store Architecture
- 7 – operating modes
- 7 – interrupts/ Exceptions
- 7- Addressing modes and 3 – data formats (8,16,32 bits)
- ARM 7 and ARM9 supports 300 MIPS when the die size is 0.13micrometer
- ALU operations are register based arithmetic and logic unit RALU
- RALU performs Add ,sub, reverse sub, multiply and multiplying with accumulate operation
- It has a barrel shifter which performs multibit left or right shifts and rotate
- Extensive debug facilities like embedded In circuit emulator ICE, Real time debug RT and on chip Joint text action group JTAG interface units exist JTAG provides direct high speed access to the MCU’S internal units (registers and control units)

ARM ORGANIZATION AND IMPLEMENTATION

3-stage pipeline ARM organization



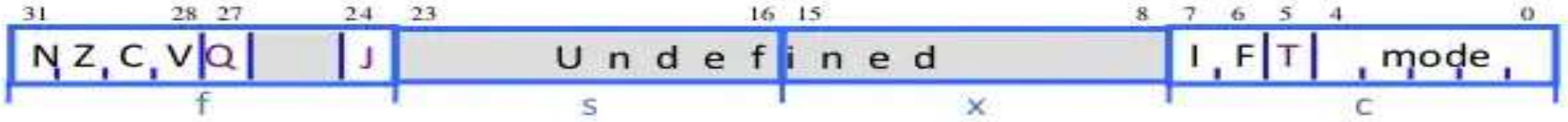
Register set of ARM



- 37 Registers of 32bits each, at a time 16 are available(R0 to R15)
- Program Counter
- CPSR
- SPSR(5)(Hold the status of the CPSR)
- GPRs (30)
- The processor mode decides which bank is accessible
- R13 (in all modes) is the OS stack pointer, but it can be used as a general purpose register when not required for stack operations.
- R14(the link Register) holds the return address form a subroutine entered when you use the branch with link (BL) instruction
- R15 is the program counter and holds the current program address (actually, it always points eight bytes ahead of the current instruction in ARM state and four bytes ahead of the current instruction in Thumb state

PROGRAM STATUS REGISTER

Program Status Registers

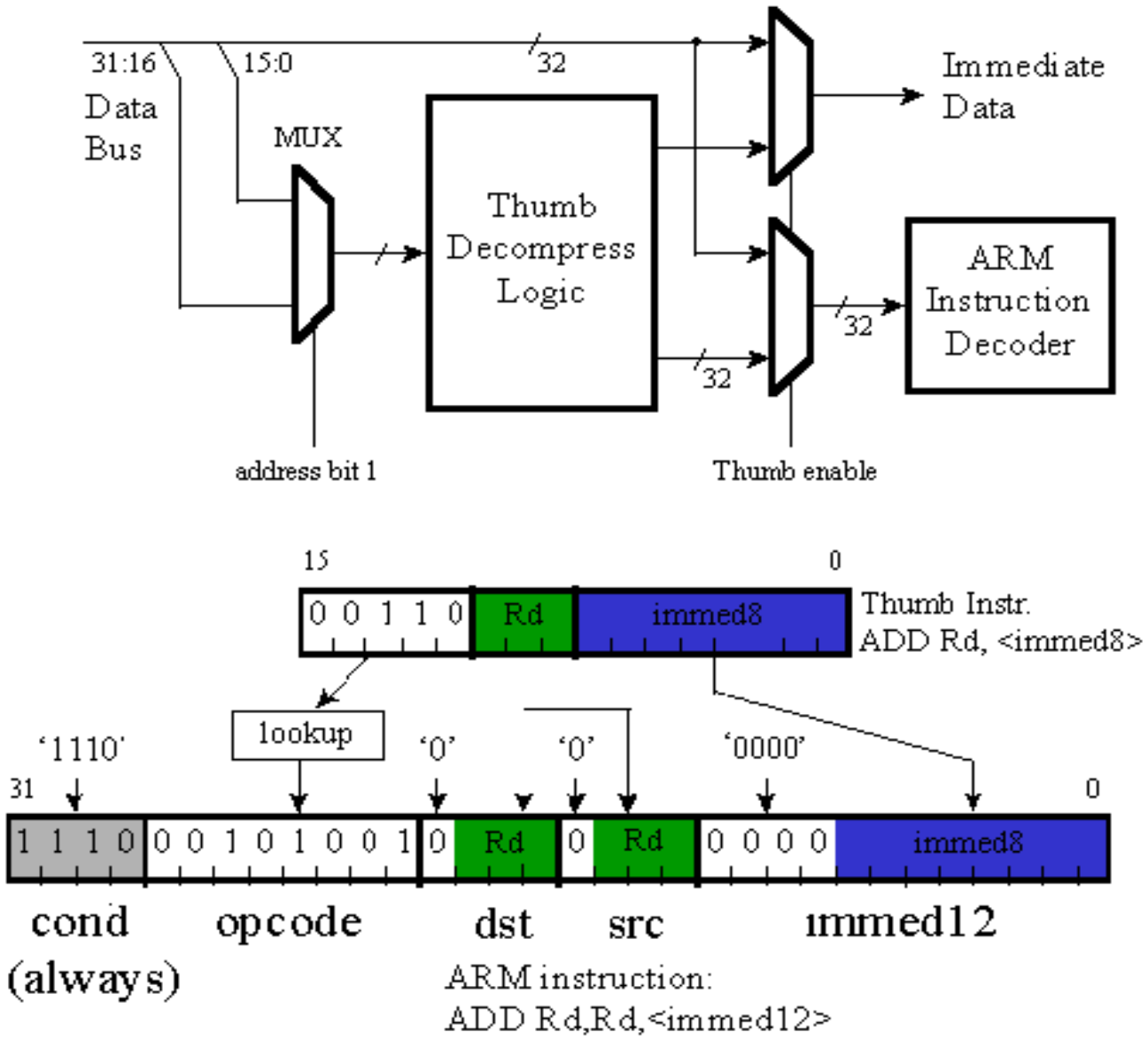


- Condition code flags
  - N = Negative result from ALU
  - Z = Zero result from ALU
  - C = ALU operation Carried out
  - V = ALU operation overflowed
- Sticky Overflow flag - Q flag
  - Architecture 5TE/J only
  - Indicates if saturation has occurred
- J bit
  - Architecture 5TEJ only
  - J = 1: Processor in Jazelle state
- Interrupt Disable bits.
  - I = 1: Disables the IRQ.
  - F = 1: Disables the FIQ.
- T Bit
  - Architecture xT only
  - T = 0: Processor in ARM state
  - T = 1: Processor in Thumb state
- Mode bits
  - Specify the processor mode

ARM PROCESSOR MODES

MODE	Function	Privilege	MODE[4:0]
Abort(ABT)	Entered on a memory access exception	Yes	10111
Fast interrupt request(FIQ)	Entered on an FIQ interrupt exception (High priority)	Yes	10001
Interrupt request(IQ)	Entered on an IRQ interrupt exception (Low priority)	Yes	10010
Supervisor(SVC)	Entered on reset or when a Supervisor Call instruction is executed	Yes	10011
System(SYS)	Mode in which the OS runs, sharing the register view with User mode	Yes	11111
Undefined(UND)	Entered when an undefined instruction executed	Yes	11011
User(USR)	Mode in which most programs and applications run	No	10000

THUMB ARCHITECTURE AND FEATURES



- increases the clock rate up to 40MHZ
- Expanded cache of 8KB
- Thumb is a combination of new instruction set with 16- bit long instruction format
- A hardware logic unit Translates thumb instruction to regular Full 32bit length ARM instruction
- Thumb will improve compiled code density by about 25% to 35%
- In system with 16 bit wide memory an ARM7 would run an application compiled into thumb faster then if it had been compiled 32- bit ARM code
- In Thumb, the 16-bit opcodes have less functionality. For example, only branches can be conditional, and many opcodes are restricted to accessing only half of all of the CPU's general-purpose registers.
- The shorter opcodes give improved code density overall

The Thumb instruction set

- The Thumb instruction set is a subset of the most commonly used 32-bit ARM instructions.
- Thumb instructions are each 16 bits long, and have a corresponding 32-bit ARM instruction that has the same effect on the processor model.
- Thumb instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and Thumb states.
- On execution, 16-bit Thumb instructions are transparently decompressed to full 32-bit ARM instructions in real time, without performance loss.
- Thumb has all the advantages of a 32-bit core:
  - 32-bit address space
  - 32-bit registers
  - 32-bit shifter, and *Arithmetic Logic Unit* (ALU)
  - 32-bit memory transfer.
- Thumb therefore offers a long branch range, powerful arithmetic operations, and a large address space.
- Thumb code is typically 65% of the size of ARM code, and provides 160% of the performance of ARM code when running from a 16-bit memory system.

- Thumb, therefore, makes the ARM7TDMI core ideally suited to embedded applications with restricted memory bandwidth, where code density and footprint is important.
- The availability of both 16-bit Thumb and 32-bit ARM instruction sets gives designers the flexibility to emphasize performance or code size on a subroutine level, according to the requirements of their applications.
- For example, critical loops for applications such as fast interrupts and DSP algorithms can be coded using the full ARM instruction set then linked with Thumb code.
- ARM and Thumb are two different instruction sets supported by ARM cores Thumb mode **allows for code to be smaller, and can potentially be faster if the target has slow memory**. This brings very high code density, since Thumb instructions are half the width of ARM instructions.

THUMB PROGRAMMERS MODEL

When operating in the 16-bit Thumb state, the application encounters a slightly different set of registers. Figure 1 compares the programmer's model in that state to the same model in the 32-bit *ARM state*.

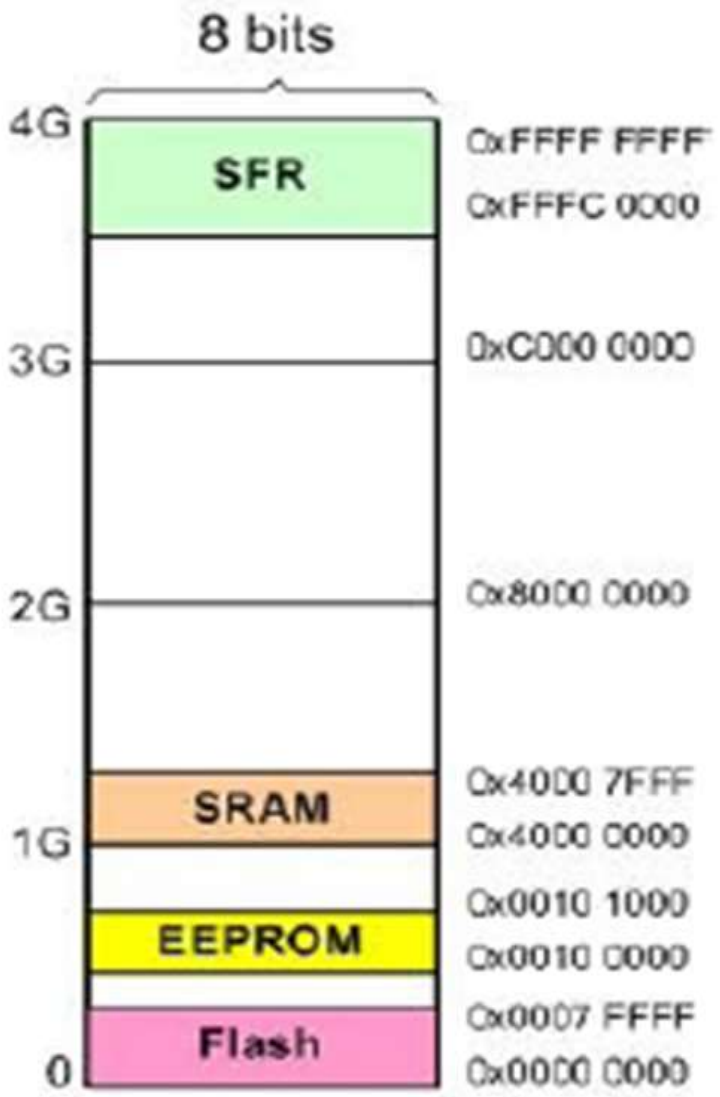
In the ARM state, 17 registers are visible in user mode. One additional register—a saved copy of Current Program Status Register (**CPSR**) that's called **SPSR** (Saved Program Status Register)—is for exception mode only.

12 registers accessible in Thumb state are exactly the same physical 32-bit registers accessible in ARM state. Thus data can be passed between software running in the ARM state and software running in the Thumb state via registers R0 through R7.

The biggest register difference involves the **SP** register. The Thumb state has unique stack mnemonics (**PUSH, POP**) that don't exist in the ARM state. These instructions assume the existence of a stack pointer, for which **R13** is used. They translate into load and store instructions in the ARM state.

Figure 1 ARM vs. Thumb programmer's models

MEMORY SPACE ALLOCATION IN ARM



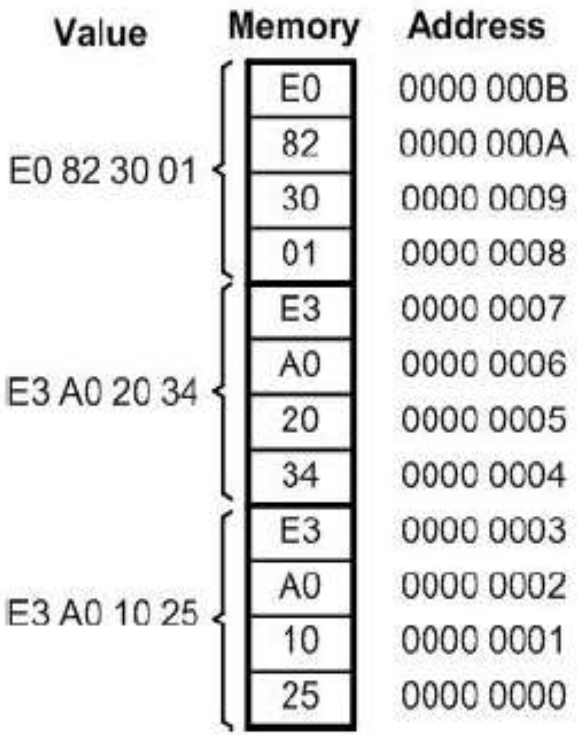
: An Example of ARM Memory Allocation

The ARM has 4GB of directly accessible memory space. This memory space has addresses 0 to 0xFFFFFFFF. The 4GB of memory space can be divided into five sections. They are as follows:

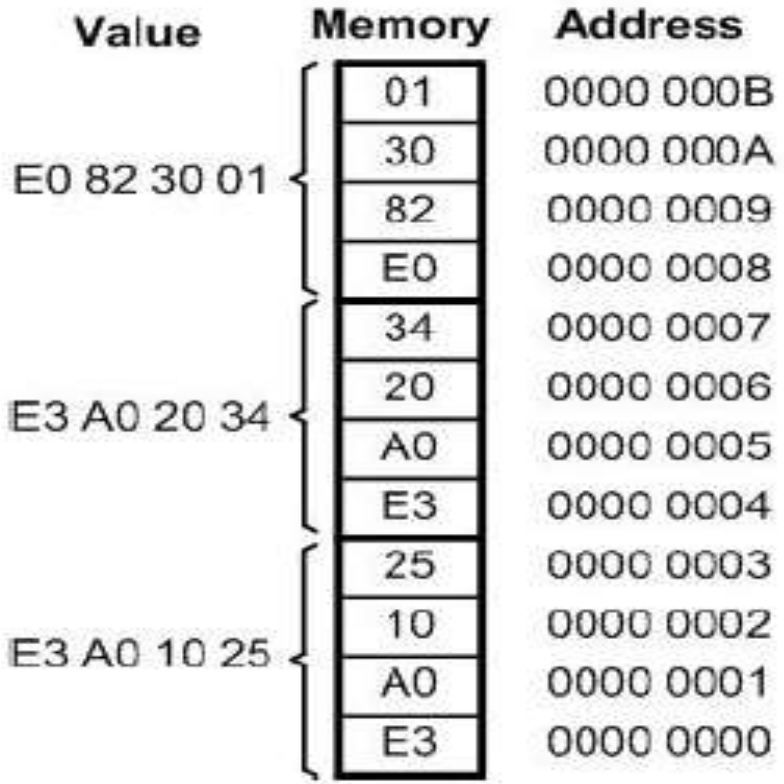
- On-chip peripheral & IO Registers
- On chip data SRAM
- On-chip EEPROM
- On-chip Flash ROM
- Off-chip DRAM
- **On-chip peripheral & IO Registers**
  - This area is dedicated to general purpose I/O (GPIO) and special function registers (SFRs) of peripherals such as timers, serial communication, ADC, and so on. In other words, ARM uses memory-mapped I/O
- **On chip data SRAM**
  - Used by the CPU for data variables & stack
  - It is a volatile memory & its content are lost if power to the chip is cut OFF

- **On chip data Flash ROM**
  - Used by the CPU for program code
  - The on-chip Flash ROM is programmed and erased in block size i.e., 8 16, 32 or 64 bytes
- **On chip EEPROM**
  - It is considered to be memory that one can add externally to the chip
  - Used most often for critical system data that must not be lost if power is cut off
  - The on-chip EEPROM is byte programmable and erasable

- All the instructions of the ARM are 32-bit wide. In other words, every instruction of ARM is fixed at 32-bit which is one of the most important characteristics of RISC architecture
- In cases where there is no need for all the 32-bit, the ARM adds zeros to make the instruction fixed at 32-bit
- 32-bit word operates separately on each byte of a word that will be divided into four 8-bits called byte0, byte1, byte2, byte3
- Processor can operate on the words as per initialization for their alignment memory addresses
- A word alignment can be Big endian and Little endian



Little Endian : LSB has Lowest address



Big Endian : LSB has Highest address

ARM Instruction Formats

- **ARM instructions are 32 bits wide, and Thumb instructions are 16 wide**. Thumb mode allows for code to be smaller and can potentially be faster if the target has slow memory
- The Thumb instruction format is designed to compete with the 8- and 16-bit microcontrollers and increase code density
- The ARM CPU uses the **tri-part instruction format for most instructions**. One of the most common format is:

  Instruction destination,source1,source2

- Depending on the instruction
  - the source2 can be a register, immediate (constant) value, or memory
  - The destination is often a register or read/write memory
- When programming the registers of the ARM microcontroller with an immediate value, the following points should be noted:
  - **put # in front of every immediate value**
  - **If we want to present a number in hex, we put a 0x in front of i**t. If we put nothing in front of a number, it is in decimal
  - For example,
  - "MOV R1,#50", R1 is loaded with 50 in decimal,
  - whereas in "MOV R1,#0x50", R1 is loaded with 50 in hex (80 in decimal)

ADDRESSING MODES OF ARM

1) **Immediate Addressing mode**, Ex: MOV Ro,#45H

ADD Ro , R1, #45H (R0 = R1 + 45H)

2) **Register Addressing mode**, Ex: MOV R0 , R1

ADD R0,R1,R2 (R0 = R1+R2)

3) **Direct Addressing mode**, Ex: LDR R0, Address (R0 = (Memory location Address))

STR R0, Address (R0 → Memory location Address)

4) **Indirect Addressing mode**, Ex: LDR R0, [R1]

STR R0, [R1]

5**) Register Relative Addressing mode** in this AM Address is given by register + displacement

i) Normal register relative AM Ex: LDR R0, [R1, #80H] (R0 = R1Address + 80H) , R1 will not be changed

ii) Pre index register relative AM Ex: LDR R0, [R1, #80H]! (R0 = R1Address + 80H) , R1 will be changed

iii) Post index register relative AM Ex: LDR R0, [R1], #08H (R0 = R1, After the operation R1 will be changed)

6**) Base index AM,**

i) Normal Base index AM Ex: LDR R0, [R1,R2] R0 = R1+R2 (R1 will not change)

ii) pre index Base index AM Ex: LDR R0, [R1,R2]! R0 = R1+R2 (R1 will change to new location)

iii) post index Base index AM Ex: LDR R0, [R1], R2 R0 = R1 (R1 will be changed to the new location given by R2)

7) Base with scaled index AM: In this addressing case on the base register is done with logical operators

i) normal base with scale index AM Ex: LDR R0, [R1,R2, LSL #3] R0 = R1+R2 (logical left shift R2 by 3) (R1 will not change)

ii) Pre index Base with scale index AM Ex: LDR R0, [R1,R2, LSL #3]! R0 = R1+ R2 (Logical left shift R2 by 3) R1 will be changed

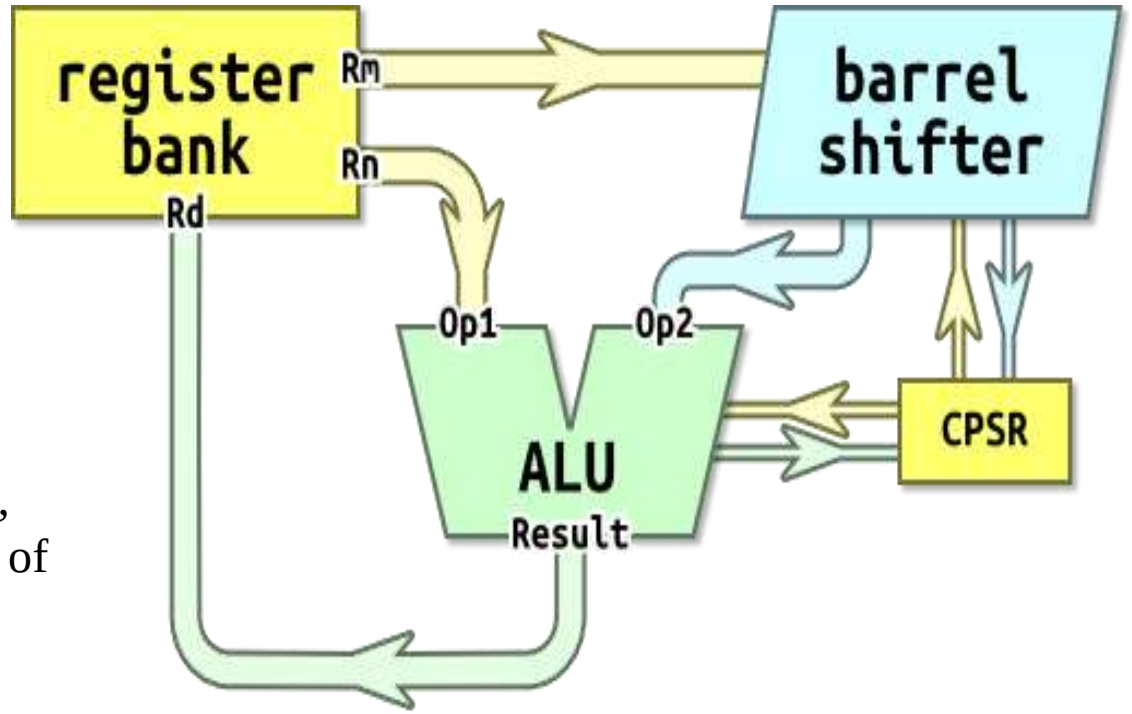
iii) Post index Base with scale index AM Ex: LDR R0, [R1], LSL #3 R0 = [R1] (After performing this operation R1 will be changed to new location that is R1 = R1 + R2 (Logical left shift R2 by 3))

ORGANIZATION

ARM has a three-address format:
•Rd — destination register
•Rn — source register
•Rm — source register

Eg: ADD R0,R1,R2

Rn is used directly but Rm is passed through the barrel shifter, A functional unit which can shift and rotate values. The result of This is called operand2



ARM INSTRUCTION SET

1) Data Processing Instructions
2) Load Store Instructions
3) Branch Instructions
4) Status register access Instructions

❖ **Data Processing Instructions :** ARM performs computations on Data which are in registers only

MOV & MVN SYNTAX: <operation>{cond}{S} Rd,Operand2

MOV Destination, source Ex: MOV R11, R2 (if R2 = 0xFFF00000 then R11 = 0xFFF00000)

MVN Destination, source Ex : MVN R11, R2 (if R2 = 0xFFF00000 then R11 = 0x000FFFFF)

❖ **The Barrel shifter instruction** (shift and rotate)

i) Logical shift left (LSL) left shift by 1 is equal to multiplication by 2

Ex: LSL R1, #8

If R1 = 0xEF00DE12

Then R1 = 0x00DE1200

ii) Logical shift right (LSR) right shift by 1

Ex: LSR R2, #5

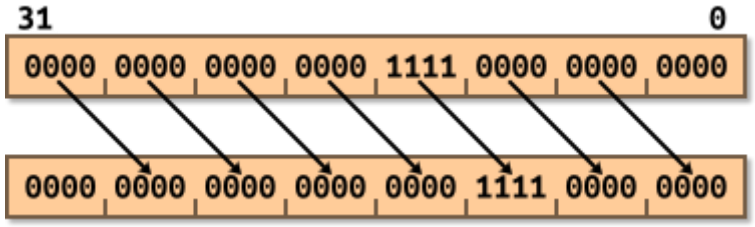
If R2 = 0x0456123F then after right shift

R2 = 0x0022B091

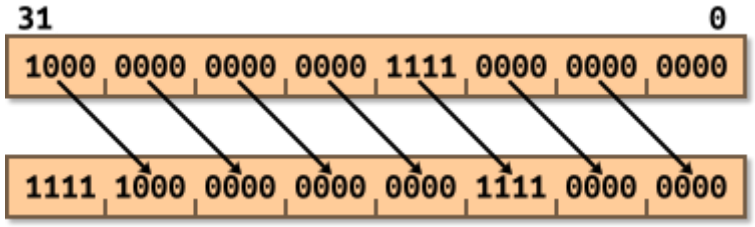
Equivalent to unsigned division by a power of 2

iii) Arithmetic shift right (ASR) right shift by 1 , this type of function is used for sign extension of data bcos for –ve numbers MSB is 1 and for +ve numbers MSB is 0

Ex: ASR R1, R5 If R1 = 0xEF00DE12 & R5 = 5 then R1 = FEF00DE1 (As MSB = 1)



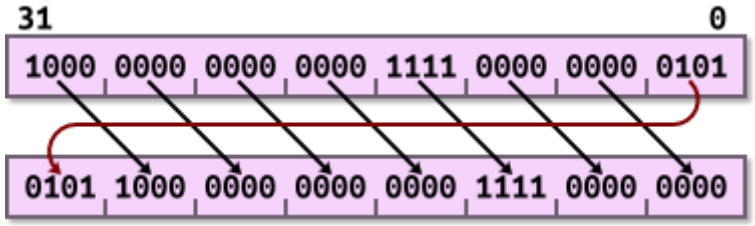
Arithmetic Shift Right by 4, positive value.



Arithmetic Shift Right by 4, negative value.

signed division by a power of 2.

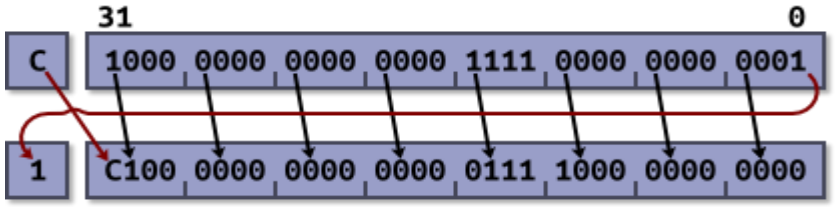
iv) Rotate Right (ROR) in this instruction the LSBs coming out of register fed back to MSB and the last bit out is also available in CF. So there is no instruction to rotate left , Ex: ROR R2, R6 If R6 = 28 this means R2 is rotated right by 28 times



if we need to rotate left by n-bits can be achieved by rotating right by (32- n)

Ex: ROR R2, R6 If R6 = 28 this means R2 is rotated right by 28 times . If we want to rotate R2 left then (32 – n) = 32- 28 = 4

V) Rotate Right Extended [RRX]



Arithmetic Instructions

<operation>{cond}{S} Rd,Rn,Operand2

ADD – Add

Rd := Rn + Operand2

ADC – Add with Carry

Rd := Rn + Operand2 + Carry

SUB – Subtract

Rd := Rn − Operand2

SBC – Subtract with Carry

Rd := Rn − Operand2 − NOT(Carry)

RSB – Reverse Subtract

Rd := Operand2 − Rn

RSC – Reverse Subtract with Carry

Rd := Operand2 − Rn − NOT(Carry)

EXAMPLES

- ADD r0, r1, r2
  - R0 = R1 + R2
- SUB r5, r3, #10
  - R5 = R3 − 10
- RSB r2, r5, #0xFF00
  - R2 = 0xFF00 − R5

Logical Instructions

<operation>{cond}{S} Rd,Rn,Operand2

AND – logical AND

Rd = Rn AND Operand2

EOR – Exclusive OR

Rd = Rn EOR Operand2

ORR – Logical OR

Rd = Rn OR Operand2

BIC – Bitwise Clear

Rd = Rn AND NOT Operand2

EXAMPLES

AND r8, r7, r2

R8 = R7 & R2

ORR r11, r11, #1

R11 |= 1

EOR r11, r11, #1

R11^ = 1

BIC r11, r11, #1

R11& = ~1

COMPARE INSTRUCTIONS

- <operation>{cond} Rn,Operand2
- CMP – compare
- Flags set to result of (Rn − Operand2).
- CMN – compare negative
- Flags set to result of (Rn + Operand2).
- TST – bitwise test
- Flags set to result of (Rn AND Operand2).
- TEQ – test equivalence
- Flags set to result of (Rn EOR Operand2).

Examples:

•CMP r0, #42
  •Compare R0 to 42.
•CMN r2, #42
  •Compare R2 to -42.
•TST r11, #1
  •Test bit zero.
•TEQ r8, r9
  •Test R8 equals R9.
•SUBS r1, r0, #42
  •Compare R0 to 42, with result.

- Comparisons produce no results – they just set condition codes. Ordinary instructions will also set condition codes if the “S” bit is set. The “S” bit is implied for comparison instructions.

❖ CMP is like SUB.
❖ CMN is like ADD – subtract of a negative number is the same as add.
❖ TST is like AND.
❖ TEQ is like EOR – Exclusive- OR of identical numbers gives result of zero.

MULTIPLY INSTRUCTION

<operation>{cond}{S} Rd, Rm, Rs {, Rn}

- MUL – *Multiply*
  - Rd := Rm × Rs
- MLA – *Multiply with Accumulate*
  - Rd := Rn + (Rm × Rs)

The multiply instructions produce the same result for both signed and unsigned values.

Single Register Data Transfer

LDR

Rd := value at <address>

STR

value at <address> := Rd

{size} is specified to transfer bytes or half-words:

<operation>B
unsigned byte
<operation>SB
signed byte
<operation>H
unsigned half-word
<operation>SH
signed half-word

EXAMPLE

- LDR r0,[r1]
  - Load word addressed by R1 into R0.
- LDRB r0,[r1]
  - The same as above but loads a byte.

❖Load And Store Instruction

```
;assume R5 = 0x40000200
LDR     R7,[R5] ;load R7 with the contents of locations
                ;0x40000200-0x40000203
```

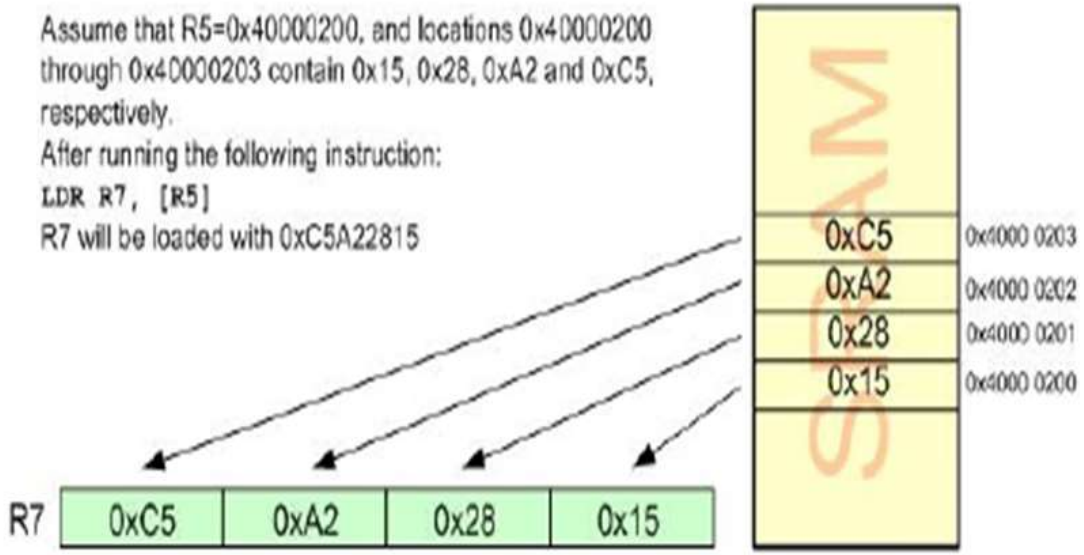


Figure : Executing the LDR Instruction

STR Rx,[Rd] instruction

```
STR     Rx,[Rd] ;store register Rx into locations pointed to by Rd
```

The STR instruction tells the CPU to store (copy) the contents of the GPR to a base address location pointed to by the Rd register. since GPR is 32-bit wide (4-byte) we need four consecutive memory locations to store the contents of GPR. The memory locations must be writable such as SRAM.

The "STR R3,[R6]" instruction will copy the contents of R3 into locations pointed to by R6. Locations 0x40000200 through 0x40000203 of the SRAM memory will have the contents of R3 since R6 = 0x40000200.

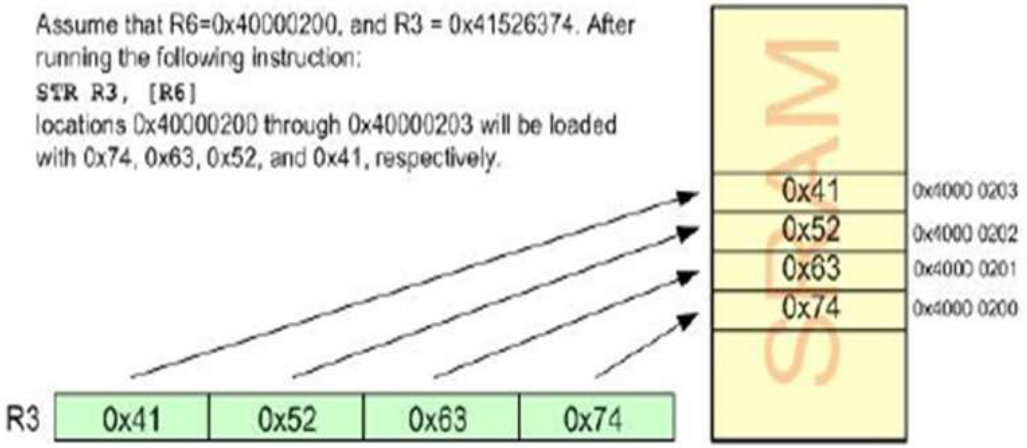


Figure : Executing the STR Instruction

MULTIPLE REGISTER DATA TRANSFER

<operation>{cond} Rn{!}, <reglist>

- LDM
  - reglist := values at Rn
- STM
  - values at Rn := reglist

<mode> controls how Rn is incremented:

- <op>IA – Increment after.
- <op>IB – Increment before.
- <op>DA – Decrement after.
- <op>DB – Decrement before.

<reglist> is the list of registers to load or store. It can be a comma-separated list or an Rx-Ry style range.

- LDMIA r0, {r3,r7}

Load words addressed by R0 into R3 and R7.
Increment After each load.

- LDMIA r0, {r3-r7}

Load words addressed by R0 into R3, R4, R5, R6 and R7.
Increment After each load.

- STMDB r1!, {r6-r8}

Store R6,R7,R8 into words addressed by R1.
Write back the final address into R1.
Decrement Before each store.

BRANCHING INSTRUCTIONS

<operation>{cond} <address>

•B – *Branch*

•PC := <address>

•BL – *Branch with Link*

•R14 := address of next instruction, PC := <address>

How do we return from the subroutine which BL invoked?

MOV pc, r14

- **Branching forward, to skip over some code:**

```
.     ……      ; some code here
      B fwd     ; jump to label 'fwd'
      ……..     ; more code here
fwd
```

- **Branching backwards, creating a loop:**

```
back
   ...          ; more code here
   B back       ; jump to label 'back'
```

we implement control structures like for and while loops

Branch instructions are used to alter control flow.

- **Using BL to call a subroutine:**

```
.....
.....
BL  calc       ; call 'calc'
.....          ; returns to here
.....

calc           ; function body
   ADD r0, r1, r2 ; do some work here
   MOV pc, r14    ; PC = R14 to return
```

CONDITIONAL EXECUTION

- A beneficial feature of the ARM architecture is that instructions can be made to execute conditionally. This is common in other architectures' branch or jump instructions but ARM allows its use with most mnemonics.
- The condition is specified with a two-letter suffix, such as EQ or CC, appended to the mnemonic. The condition is tested against the current processor flags and if not met the instruction is treated as a no-op. This feature often removes the need to branch, avoiding pipeline stalls and increasing speed. It can also increase code density.
- By default the data processing instructions do not affect the condition code flags but can be made to by suffixing S. The comparison instructions CMP, TST, and co. do this implicitly.

EXAMPLE : Looping

The following code fragment is a do-while loop which runs until the counter in R1 hits zero, at which point the condition code NE (not equal to zero) controlling the branch becomes false.

```
 MOV   r1, #10
loop
    ...
    SUBS  r1, r1, #1
    BNE   loop
```

CONDITION CODES

Code	Suffix	Description Flags
0000	EQ	Equal / equals zero Z
0001	NE	Not equal !Z
0010	CS / HS	Carry set / unsigned higher or same C
0011	CC / LO	Carry clear / unsigned lower !C
0100	MI	Minus / negative N
0101	PL	Plus / positive or zero !N
0110	VS	Overflow V
0111	VC	No overflow !V
1000	HI	Unsigned higher C and !Z
1001	LS	Unsigned lower or same !C or Z
1010	GE	Signed greater than or equal N == V
1011	LT	Signed less than N != V
1100	GT	Signed greater than !Z and (N == V)
1101	LE	Signed less than or equal Z or (N != V)
1110	AL	Always (default) any

Branch	Interpretation	Normal uses
B	Unconditional	Always take this branch
BAL	Always	Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC	Carry clear	Arithmetic operation did not give carry-out
BLO	Lower	Unsigned comparison gave lower
BCS	Carry set	Arithmetic operation gave carry-out
BHS	Higher or same	Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation; no overflow occurred
BVS	Overflow set	Signed integer operation; overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

Examples of Conditional Instructions

```
CMP   r0, #5          ; if (a == 5)
MOVEQ r0, #10
BLEQ  fn              ;   fn(10)
```

(Assume a is in R0. Compare R0 to 5. The next two instructions will be executed only if the compare returns Equal. They move 10 into R0, then call ‘fn’ (branch with link, BL).)

Set the flags, then use various condition codes:

```
CMP   r0, #0   ; if (x <= 0)
MOVLE r0, #0   ;   x = 0;
MOVGT r0, #1   ; else x = 1;
```

Use conditional compare instructions:

```
 CMP   r0, #'A' ; if (c == 'A'
CMPNE r0, #'B' ;  || c == 'B')
MOVEQ r1, #1   ;   y = 1;
```

A sequence which doesn’t use conditional execution:

```
    CMP   r3, #0
    BEQ   next
    ADD   r0, r0, r1
    SUB   r0, r0, r2
next
...
```

By transforming the sequence with conditional execution the BEQ instruction can be removed:

```
CMP   r3, #0
ADDNE r0, r0, r1
SUBNE r0, r0, r2
...
```

Thumb instruction set

Mnemonic	Instruction	Example	ARM-code equivalent
ADC	Add with Carry	`ADC Rd,Rs`	`ADCS Rd,Rd,Rs`
ADD	Add	`ADD Rd,Rs,Rn`	`ADDS Rd,Rs,Rn`
AND	AND	`AND Rd,Rs`	`ANDS Rd,Rd,Rs`
ASR	Arithmetic Shift Right	`ASR Rd,Rs`	`MOVS Rd,Rd,ASR Rs`
B	Unconditional branch	`B label`	`B label`
BCC	Conditional branch	`BCC label`	`BCC label`
BIC	Bit Clear	`BIC Rd,Rs`	`BICS Rd,Rd,Rs`
BL	Branch and Link	`BL label`	`BL label`
BX	Branch and Exchange	`BX Hs`	`BX Hs`
CMN	Compare Negative	`CMN Rd,Rs`	`CMN Rd,Rs`
CMP	Compare	`CMP Rd,#Offset8`	`CMP Rd,#Offset8`
EOR	EOR	`EOR Rd,Rs`	`EORS Rd,Rd,Rs`
LDMIA	Load multiple	`LDMIA Rb!,{Rlist}`	`LDMIA Rb!,{Rlist}`
LDR	Load word	`LDR Rd,[PC,#Imm]`	`LDR Rd,[PC,#Imm]`
LDRB	Load byte	`LDRB Rd,[Rb,Ro]`	`LDRB Rd,[Rb,Ro]`
LDRH	Load halfword	`LDRH Rd,[Rb,#Imm]`	`LDRH Rd,[Rb,#Imm]`
LSL	Logical Shift Left	`LSL Rd,Rs,#Offset5`	`MOVS Rd,Rs,LSL#Offset5`

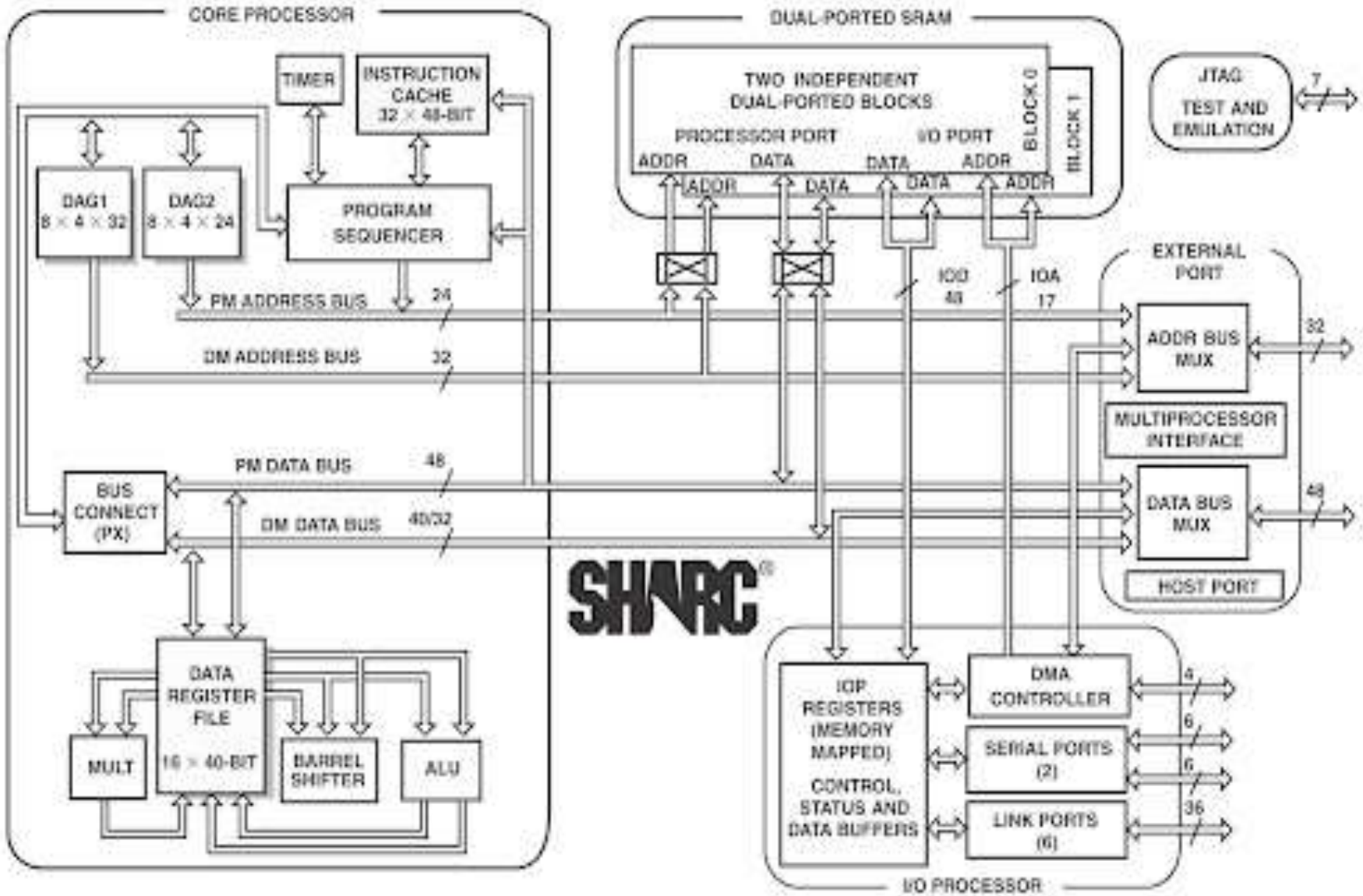
Thumb instruction set (Cont.)

LDRSB	Load sign-extended byte	`LDRSB Rd,[Rb,Ro]`	`LDRSB Rd,[Rb,Ro]`
LDRSH	Load sign-extended halfword	`LDRSH Rd,[Rb,Ro]`	`LDRSH Rd,[Rb,Ro]`
LSR	Logical Shift Right	`LSR Rd,Rs`	`MOVS Rd,Rd,LSR Rs`
MOV	Move register	`MOV Rd,#Offset8`	`MOVS Rd,#Offset8`
MUL	Multiply	`MUL Rd,Rs`	`MULS Rd,Rs,Rd`
MVN	Move NOT register	`MVN Rd,Rs`	`MVNS Rd,Rs`
NEG	Negate	`NEG Rd,Rs`	`RSBS Rd,Rs,#0`
ORR	OR	`ORR Rd,Rs`	`ORRS Rd,Rd,Rs`
POP	Pop registers	`POP {Rlist}`	`LDMIA R13!,{Rlist}`
PUSH	Push registers	`PUSH {Rlist}`	`STMDB R13!,{Rlist}`
ROR	Rotate Right	`ROR Rd,Rs`	`MOVS Rd,Rd,ROR Rs`
SBC	Subtract with Carry	`SBC Rd,Rs`	`SBCS Rd,Rd,Rs`
STMIA	Store Multiple	`STMIA Rb!,{Rlist}`	`STMIA Rb!,{Rlist}`
STR	Store word	`STR Rd,[Rb,Ro]`	`STR Rd,[Rb,Ro]`
STRB	Store byte	`STRB Rd,[Rb,Ro]`	`STRB Rd,[Rb,Ro]`
STRH	Store halfword	`STRH Rd,[Rb,Ro]`	`STRH Rd,[Rb,Ro]`
SWI	Software Interrupt	`SWI Value8`	`SWI Value8`

SHARC Processor:

Features of SHARC processor:

1. SHARC stands for **Super Harvard Architecture Computer**
2. The **ADSP-21060 SHARC chip** is made by Analog Devices, Inc.
3. It is a **32-bit signal processor** made mainly for **sound, speech, graphics, and imaging applications**.
4. It is a high-end digital signal processor designed with **RISC techniques**.
5. Number formats:
   i. **32-bit Fixed Format**
      Fractional/Integer
      Unsigned/Signed
   ii. **Floating Point**
      32-bit single-precision IEEE floating-point data format
      40-bit version of the IEEE floating-point data format.
      16-bit shortened version of the IEEE floating-point data format.
6. **32 Bit floating point, with 40 bit extended floating point capabilities**.
7. Large on-chip memory.
8. Ideal for scalable multi-processing applications.
9. Program memory can store data.
10. Able to simultaneously read or write data at one location and get instructions from another place in memory.

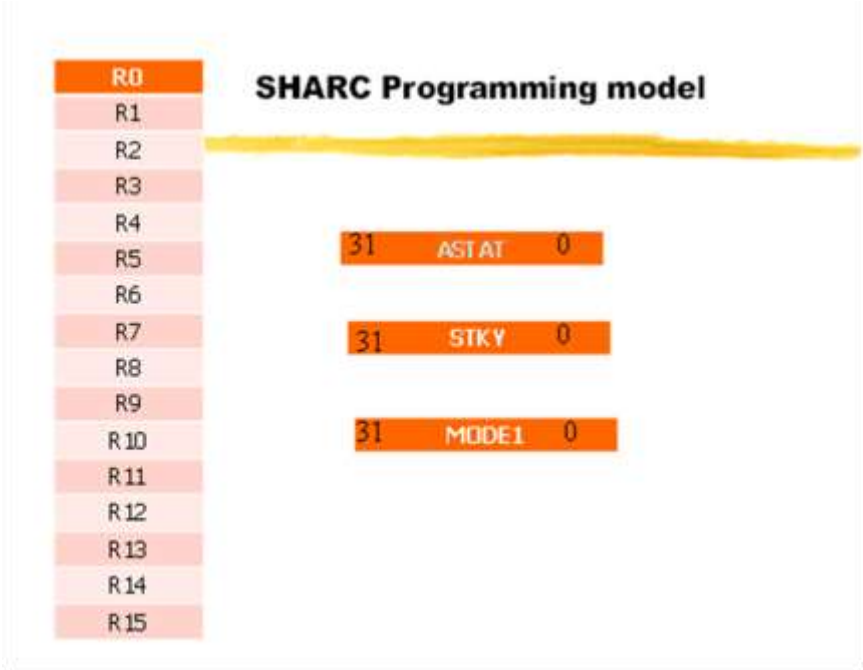

CORE PROCESSOR
TIMER
INSTRUCTION CACHE 32 × 48-BIT
DAG1 8 × 4 × 32
DAG2 8 × 4 × 24
PROGRAM SEQUENCER
PM ADDRESS BUS
24
DM ADDRESS BUS
32
PM DATA BUS
48
BUS CONNECT (PX)
DM DATA BUS
40/32
DATA REGISTER FILE
16 × 40-BIT
MULT
BARREL SHIFTER
ALU
SHARC
DUAL-PORTED SRAM
TWO INDEPENDENT DUAL-PORTED BLOCKS
BLOCK 0
BLOCK 1
PROCESSOR PORT
I/O PORT
ADDR
DATA
IOD
48
IOA
17
JTAG TEST AND EMULATION
7
EXTERNAL PORT
ADDR BUS MUX
32
MULTIPROCESSOR INTERFACE
DATA BUS MUX
48
HOST PORT
IOP REGISTERS (MEMORY MAPPED)
CONTROL, STATUS AND DATA BUFFERS
DMA CONTROLLER
4
SERIAL PORTS (2)
6
6
LINK PORTS (6)
36
I/O PROCESSOR

11. 2 buses
    Data memory bus.
    Program bus.
12. Either two separate memories or a single dual-port memory
13. High-Speed Floating Point Capability
14. The **SHARC supports floating, extended-floating and non-floating point**.
15. No additional clock cycles for floating point computations.
16. Data automatically truncated and zero padded when moved between 32-bit memory and internal registers

SHARC PROCESSOR PROGRAMMING MODEL:

Programming model gives the registers details. The following registers are used in SHARC processors for various purposes:

- Register files: R0-R15 (aliased as F0-F15 for floating point)
- Status registers.
- Loop registers.
- Data address generator registers(DAG1 and DAG2)
- Interrupt registers.
- 16 primary registers (R0-R15)
- 16 alternate registers (F0-F15)
- each register can hold 40 bits
- R0 – R15 are for Fixed-Point Numbers
- F0 – F15 are for Floating-Point Numbers



STATUS REGISTERS

- ASTAT: arithmetic status.
- STKY: sticky.
- MODE 1: mode 1.

The **Arithmetic Status Register (ASTAT)** information. Use the Core | Status| ASTAT menu function to access the ASTAT register display window

Bit	Name	Definition
0	AZ	ALU result zero or floating-point underflow
1	AV	ALU overflow
2	AN	ALU result negative
3	AC	ALU fixed-point carry
4	AS	ALU X input sign (ABS and MANT operations)
5	AI	ALU floating-point invalid operation
6	MN	Multiplier result negative
7	MV	Multiplier overflow
8	MU	Multiplier floating-point underflow
9	MI	Multiplier floating-point invalid operation
10	AF	ALU floating-point operation
11	SV	Shifter overflow
12	SZ	Shifter result zero
13	SS	Shifter input sign
14	14-17	Reserved
18	BTF	Bit test flag for system registers
19	FLG0	FLAG0 value
20	FLG1	FLAG1 value
21	FLG2	FLAG2 value
22	FLG3	FLAG3 value
23	Reserved	
24-31	CACC (Compare Accumulation) bits	

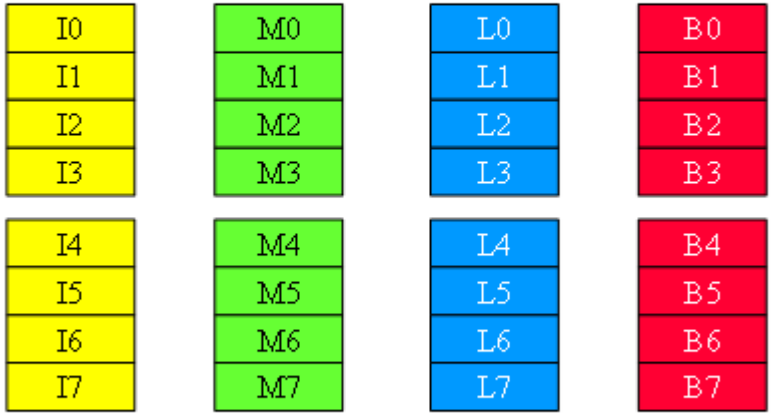
The **Sticky Status Register** (STKY) maintains -sticky+ versions of some of the ASTAT bits and circular buffer overflow. Also, it contains flags on stack overflow and underflow for the PC, status, and loop address and loop counter stacks. Use the Core | Status| STKY menu function to access the STKY register display window

Bit	Name	Definition
0	AUS	ALU floating-point underflow
1	AVS	ALU floating-point overflow
2	AOS	ALU fixed-point overflow
3-4		Reserved
5	AIS	ALU floating-point invalid operation
6	MOS	Multiplier fixed-point overflow
7	MVS	Multiplier floating-point overflow
8	MUS	Multiplier floating-point underflow
9	MIS	Multiplier floating-point invalid operation
10-16		Reserved
17	CB7S	DAG1 circular buffer 7 overflow
18	CB15S	DAG2 circular buffer 15 overflow
19-20		Reserved
21	PCFL	PC stack full
22	PCEM	PC stack empty
23	SSOV	Status stack overflow
24	SSEM	Status stack empty
25	LSOV	Loop address stack and loop counter stack overflow
26	LSEM	Loop address stack and loop counter stack overflow
27	27-31	Reserved

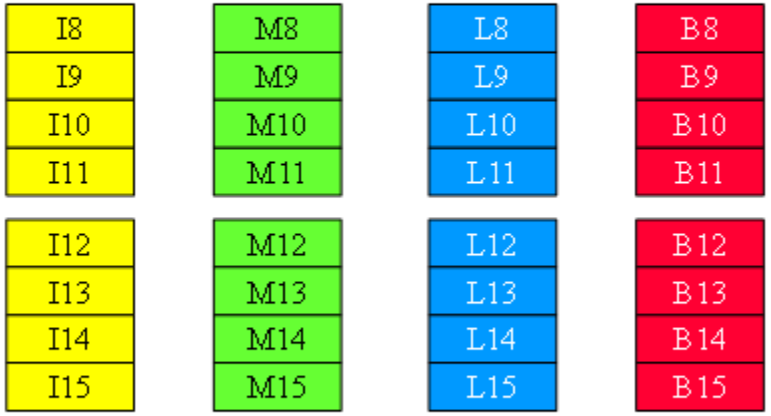
The **MODE1 register** is a 32-bit control register that enables various operating modes

Bit	Name	Definition
0		Reserved
1	BR0	Bit-reverse for I0 (uses DMS0 only)
2	SRCU	Alternate register select for computation units
3	SRD1H	DAG1 alternate register select (7-4)
4	SRD1L	DAG1 alternate register select (3-0)
5	SRD2H	DAG2 alternate register select (15-12)
6	SRD2L	DAG2 alternate register select (11-8)
7	SRRFH	Register File alternate select for R(15-8)
8-9		Reserved
10	SRRFL	Register File alternate select for R(7-0)
11	NESTM	Interrupt Nesting Enable
12	IRPTEN	Global Interrupt Enable
13	ALUSAT	Enable ALU saturation (full scale in fixed point)
14		Reserved
15	TRUNC	1 = Floating-point truncation, 0 = round to nearest
16	RND32	1 = Round floating-point to 32 bits, 0 = round to 40 bits
17-31		Reserved

DAG1 registers



DAG2 registers



The Data Address Generator 1 (DAG1) registers generate data memory addresses.
There are 8 sets of 32 bit registers in DAG1:
Index Registers I0-I7
Modify Registers M0-M7
Base Registers B0-B7
Length Registers L0-L7
Each register set (I, M, B, and L registers) has an alternate set.
The alternate registers divide into 2 groups:
a lower half (0-3) and an upper half (4-7). A bit in the MODE1 register controls the selection of primary or alternate status for each group.

Multifunction computations or instruction level parallel processing:

Can issue some computations in parallel:

- dual add-subtract;
- fixed-point multiply/accumulate and add, subtract, average
- floating-point multiply and ALU operation
- multiplication and dual add/subtract

Pipelining in SHARC processor:

Instructions are processed in three cycles:

- ❖ Fetch instruction from memory
- ❖ Decode the opcode and operand
- ❖ Execute the instruction

- SHARC supports delayed and non-delayed branches
- Specified by bit in branch instruction
- 2 instruction branch delay slot
- Six Nested Levels of Looping in Hardware

Bus Architecture:

Twin Bus Architecture:

- ❖ 1 bus for Fetching Instructions
- ❖ 1 bus for Fetching Data

Improves multiprocessing by allowing more steps to occur during each clock

Addressing modes provided by DAG in SHARC Processor:

1. The Simplest addressing mode
2. Absolute address
3. post modify with update mode
4. base-plus-offset mode
5. Circular Buffers
6. Bit reversal addressing mode

IO DEVICES AND INTERFACING AND NETWORKS UNIT-- II

Timers/Counters

Interrupt controller

DMA

A/D &D/A

Displays

Keyboards

CAN, I2C, SPI, USB, IrDA

RS485, RS232

Bluetooth, Zigbee

- I/O is very much architecture/system dependent
- I/O requires cooperation between – processor that issues I/O command (read, write etc.) buses that provide the interconnection between processor, memory and I/O devices
- I/O controllers that handle the specifics of control of each device and interfacing devices that store data or signal events

I/O operations

- **Specific I/O instructions** – I/O instruction specifies both the device number and a command (or an address where the I/O device can find a series of commands) Example: Intel x86 (IN and OUT between EAX register and an I/O port whose address is either an immediate or in the DX register)

Memory-mapped I/O

- Portions of address space devoted to I/O devices (read/write to these addresses transfer data or are used to control I/O devices)
- Memory ignores these addresses
- In both cases, only the O.S. can execute I/O operations or read/write data to memory-mapped locations

Data transfer to/from I/O device

- Can be done either by – Using the CPU to transfer data from (to) the device to (from memory.
- Can be done either via polling (programmed I/O operation) or interrupt
- Slow operation

Using DMA (direct-memory address)

- Having long blocks of I/O go through the processor via load-store is totally inefficient
- DMA (direct memory address) controller: – specialized “processor” for transfer of blocks between memory and I/O devices w/o intervention from CPU
- Has registers set up by CPU for beginning memory address and count
- DMA device interrupts CPU at end of transfer
- DMA device is a master for the bus
- DMA stands for Direct Memory Access.
- It is designed by Intel to transfer data at the fastest rate.
- It allows the device to transfer the data directly to/from memory without any interference of the CPU.

- Using a DMA controller, the device requests the CPU to hold its data, address and control bus, so the device is free to transfer data directly to/from the memory.
- The DMA data transfer is initiated only after receiving HLDA signal from the CPU.

Following is the sequence of operations performed by a DMA −

- Initially, when any device has to send data between the device and the memory, the device has to send DMA request (DRQ) to DMA controller.
- The DMA controller sends Hold request (HRQ) to the CPU and waits for the CPU to assert the HLDA.
- Then the microprocessor tri-states all the data bus, address bus, and control bus. The CPU leaves the control over bus and acknowledges the HOLD request through HLDA signal.

- Now the CPU is in HOLD state and the DMA controller has to manage the operations over buses between the CPU, memory, and I/O devices.
- DMA CONTROLLER 8257/8237
- It has four channels which can be used over four I/O devices.
- Each channel has 16-bit address and 14-bit counter.
- Each channel can transfer data up to 64kb.
- Each channel can be programmed independently.
- Each channel can perform read transfer, write transfer and verify transfer operations.
- It generates MARK signal to the peripheral device that 128 bytes have been transferred.
- It requires a single phase clock.
- Its frequency ranges from 250Hz to 3MHz.
- It operates in 2 modes, i.e., **Master mode** and **Slave mode**

Timer/Counter

A **timer** is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a **stopwatch**. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.

A **counter** is a device that stores (and sometimes displays) the number of times a particular event or process occurred, with respect to a clock signal. It is used to count the events happening outside the microcontroller. In electronics, counters can be implemented quite easily using register-type circuits such as a flip-flop.

Timer	Counter
The register incremented for every machine cycle.	The register is incremented considering 1 to 0 transition at its corresponding to an external input pin (T0, T1).
Maximum count rate is 1/12 of the oscillator frequency.	Maximum count rate is 1/24 of the oscillator frequency.
A timer uses the frequency of the internal clock, and generates delay.	A counter uses an external signal to count pulses.

- An interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
- Whenever an interrupt occurs, the controller completes the execution of the current instruction and starts the execution of an **Interrupt Service Routine** (ISR) or **Interrupt Handler**.
- ISR tells the processor or controller what to do when the interrupt occurs.
- The interrupts can be either hardware interrupts or software interrupts.

Hardware Interrupt

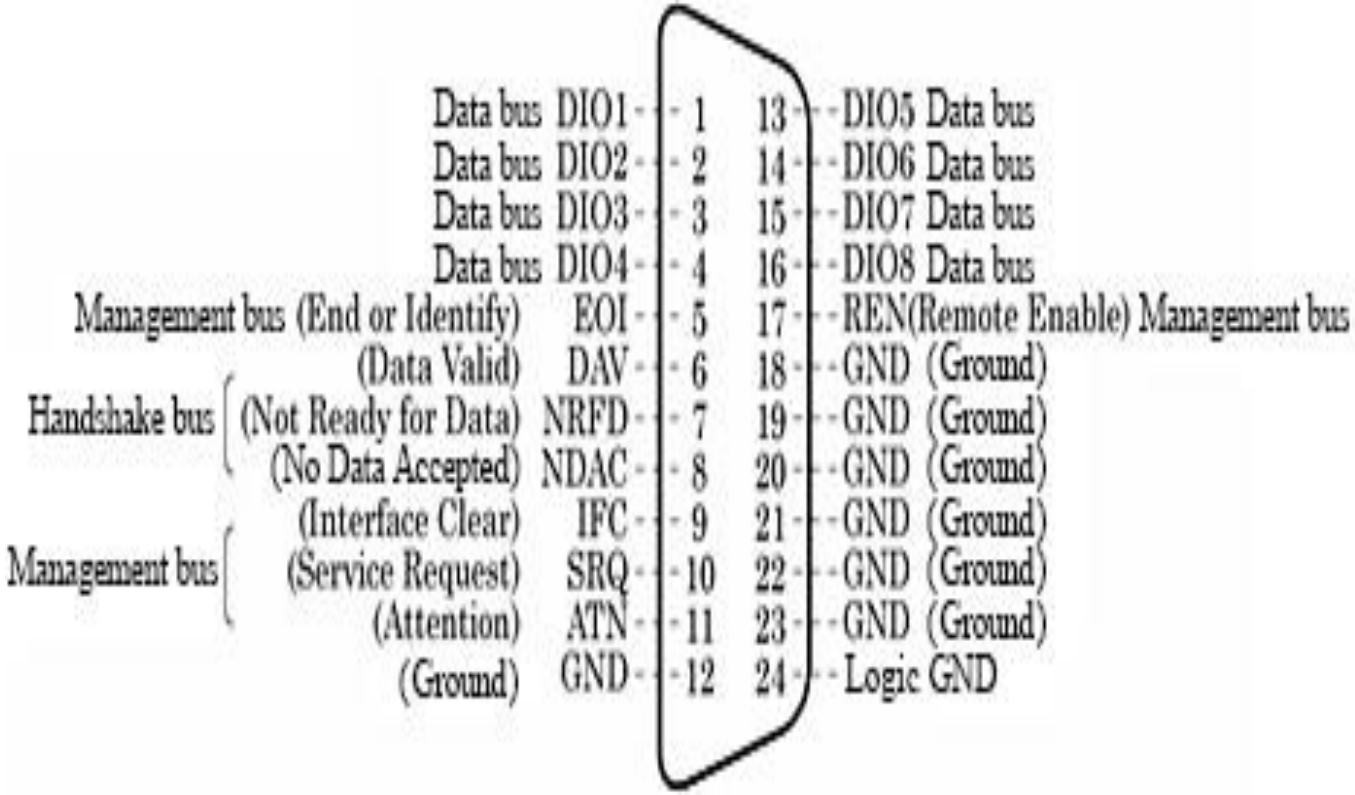
- A hardware interrupt is an electronic alerting signal sent to the processor from an external device, like a disk controller or an external peripheral.
- For example, when we press a key on the keyboard or move the mouse, they trigger hardware interrupts which cause the processor to read the keystroke or mouse position.

Software Interrupt

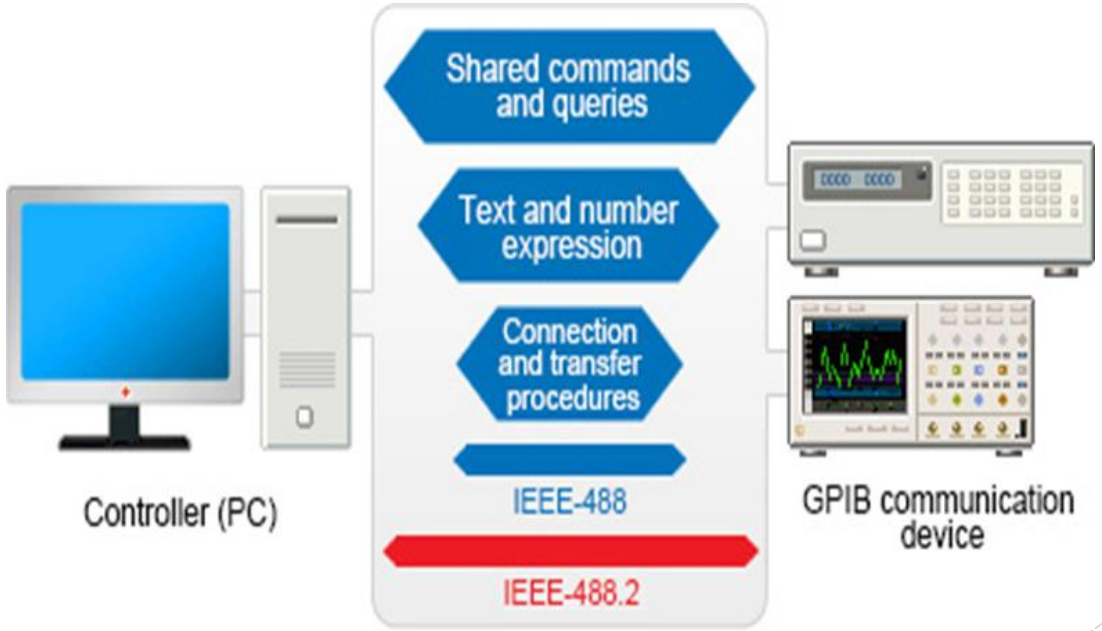
- A software interrupt is caused either by an exceptional condition or a special instruction in the instruction set which causes an interrupt when it is executed by the processor.
- For example, if the processor's arithmetic logic unit runs a command to divide a number by zero, to cause a divide-by-zero exception, thus causing the system to abandon the calculation or display an error message. Software interrupt instructions work similar to subroutine calls.

I/O DEVICES INTERFACING

General purpose digital interface system that can be used to transfer data between two or more devices (GPIB)



- Up to 15 devices may be connected to one bus
- Total bus length may be up to 20 m and the distance between devices may be up to 2 m
- Communication is digital (as opposed to analog) and messages are sent one byte (8 bits) at a time
- Message transactions are hardware handshake
- Data rates may be up to 1 Mbyte/sec



Universal Serial Bus (USB)

- A Universal Serial Bus (USB) is **a common interface that enables communication between devices and a host controller such as a personal computer (PC) or smartphone**.
- It connects peripheral devices such as digital cameras, mice, keyboards, printers, scanners, media devices, external hard drives and flash drives.
- One of the greatest features of the USB is hot swapping.
- This feature allows a device to be removed or replaced without the past prerequisite of rebooting and interrupting the system.
- Older ports required that a PC be restarted when adding or removing a new device.
- Rebooting allowed the device to be reconfigured and prevented electrostatic discharge (ESD), an unwanted electrical current capable of causing serious damage to sensitive electronic equipment such as integrated circuits.

- Another USB feature is the use of direct current (DC).
- In fact, several devices use a USB power line to connect to DC current and do not transfer data.
- Example devices using a USB connector only for DC current include a set of speakers, an audio jack and power devices like a miniature refrigerator, coffee cup warmer or keyboard lamp.
- USB Version 1 allowed for two speeds: 1.5 Mb/s (megabits per second) and 12 Mb/s, which work well for slow I/O devices.
- USB Version 2 allows up to 480 Mb/s and is backward compatible with slower USB devices.
- The first USB version 3 (USB 3.0 or SuperSpeed USB) was released in 2008, and allowed for a speed of 500 Mb/s. In 2013 and 2017, two new USB version 3 were released: USB 3.1 and USB 3.2, which allowed for 1.21 Gb/s and 2.42 Gb/s, respectively.

Pin Configuration

The typical Type-A USB connector is used in various applications. These USBs include 4 pins that are given below.

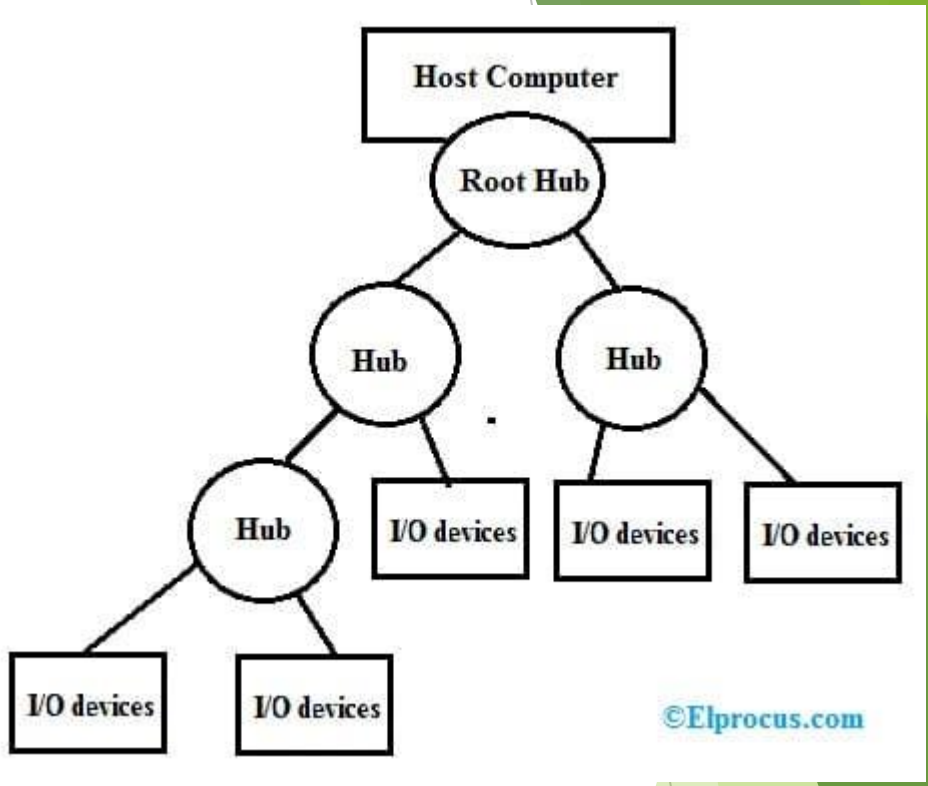
This type of USB is observed mostly in connecting various devices to PC because it is the typical four-pin USB connector. This connector is taller and narrower including 4-pins arranged within a box.

The pins of Type A USB are indicated with color wires to perform a particular function.

- **Pin1 (VBUS):** It is a red color wire, used for providing power supply.
- **Pin2 (D-):** It is a differential pair pin available in white color, used for connectivity of USB.
- **Pin3 (D+):** It is a differential pair pin available in green color, used for connectivity of USB.
- **Pin4 (GND):** It is a Ground pin, available in black color.

In the above pins, both the D+ & D- pins indicate the transfer of data. When a ‘1’ is sent across the wires, then the D+ line will have positive flow, and if ‘0’ is sent then the reverse happens.

- Once various I/O devices are connected through USB to the computer then they all are structured like a tree.
- In this USB structure, every I/O device will make a point-to-point connection to transmit data through the serial transmission format.
- I/O devices are connected to the computer through USB which is called as a hub.
- The Hub within the architecture is the connecting point between both the I/O devices as well as the computer.
- The root hub in this architecture is used to connect the whole structure to the hosting computer.
- The I/O devices in this architecture are a keyboard, mouse, speaker, camera, etc.



Infrared Data Association (IrDA)

- Infrared Data Association (IrDA) is **a protocol suite designed to provide wireless, line-of-sight connectivity between devices**.
- IrDA, as available on Microsoft Windows, provides core services similar to those exposed by Transmission Control Protocol (the TCP part of TCP/IP).
- Infrared communication is a low-cost method of providing wireless, point-to-point communication between two devices
- IrDA (Infrared Data Association) is **an industry standard for wireless communication with infrared light**.
- Many laptops sold today are equipped with an IrDA-compatible transceiver that enables communication with other devices, such as printers, modems, LANs, or other laptops.
- The Infrared Data Association (IrDA) **defines physical specifications communication protocol standard for the short-range exchange of data over infrared light**, for uses such as personal area networks (PANs)

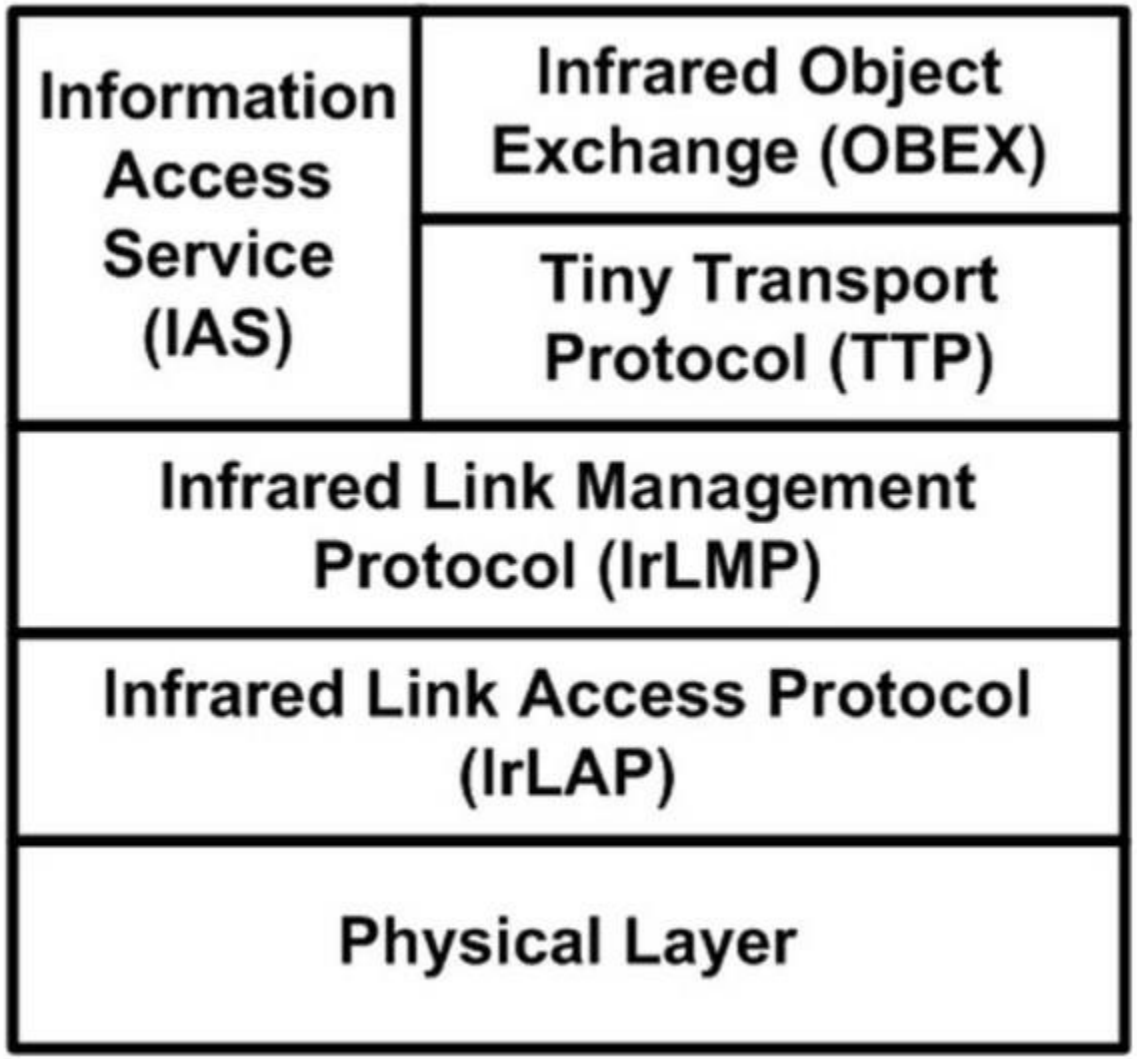
Characteristics	IrDA-Data	Bluetooth
Physical Media	Infrared	RF (2.4 GHz)
Communications Range	Up to at least 1m	10cm to 100m
Connection Type, Direction	Point-to-Point, Narrow Angle (30 degrees)	Multipoint, Omni-directional
Maximum Data Rate	4Mbps (16Mbps on the way)	1Mbps (aggregate)

- IR, or infrared, communication is **a common, inexpensive, and easy to use wireless communication technology**.
- IR light is very similar to visible light, except that it has a slightly longer wavelength.
- This means IR is undetectable to the human eye - perfect for wireless communication.

IrDA Protocol Layers:

There are different IrDA protocol layers are there

- Application Layer
- Session Layer
- IrLMIAS
- IrTinyTP
- IrLMP
- Physical Layer



Physical Layer:

- This layer has an ability for accessing half duplex or alternating directions duplex access.
- It provides a value 1 m or 10 cm(For low power LED).
- **Different Modes:** Synchronous PPM, Synchronous Serial ,Asynchronous Serial

- (a) Layer 1—physical
- (b) Data link layer
  - 2a—IrLAP (link access protocol)
  - 2b—IrLMP (link management protocol)
- (c) Layer 3-4—transport layer
  - tiny TP (transport protocol) or
  - IrLMIAS (link management information access service protocol)
- (d)Layer 5—session
  - IrLAN(for Infrared LAN access)
  - IrBus(for access to serial bus by joysticks, keyboard, mice, and game ports)
  - IrMC(IrDA mobile communication and telephony protocol)
  - IrTran (IrDA transport protocol for image or file transfers)
  - IrComm [IrDA communication protocol by emulating serial (for example RS232C COM) or parallel port]
  - IrOBEX (object exchange) Supports security by encryption and decryption at transmitter and receiver, respectively

Communicates and exchanges binary data by establishing a client–server network between two IR devices

- (e) Layers 6 and 7—security and application software layers as specified by the IrDA Alliance Sync (PIM), object push (PIM), or binary file transfer

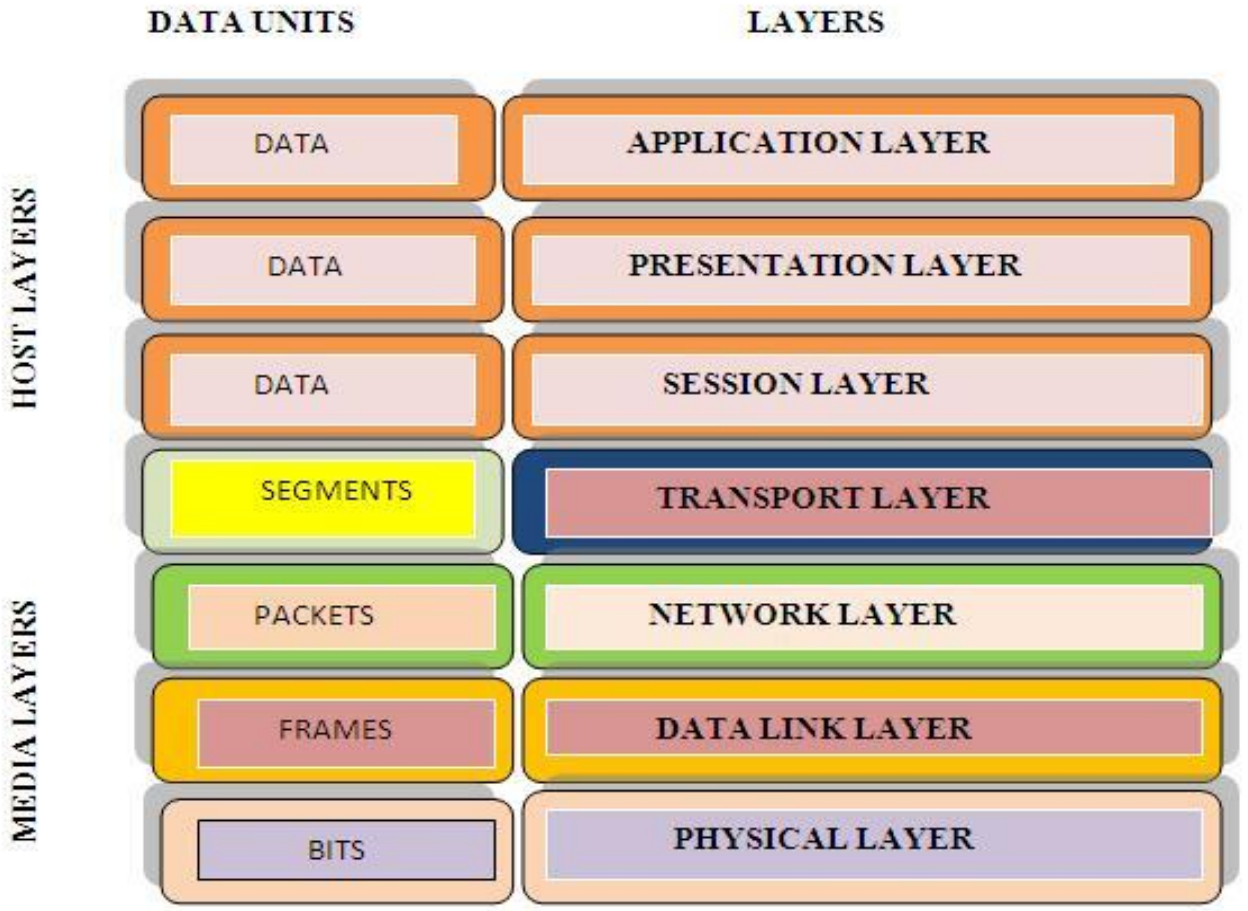
Networks

(A) CAN
(B) I2C
(C) SPI
(D) RS485
(E) RS432
(F) Bluetooth
(G) Zigbee

- **Controller Area Network (CAN) is another type of serial communications protocol that was developed within the automotive industry to allow a number of electronic units on a single vehicle to share essential control data**. A vehicle nowadays uses many microcontrollers for autonomous control systems.
- The CAN bus is primarily used in **embedded systems**, and as its name implies, is a network technology that provides fast communication among microcontrollers up to real-time requirements
- The CAN communication protocol is **a carrier-sense, multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP)**. CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message.
- Examples of CAN devices include **engine controller (ECU), transmission, ABS, lights, power windows, power steering, instrument panel**, and so on.

CAN architecture

- The CAN protocol uses the existing OSI reference model to transfer data between the nodes connected in a network.
- The OSI reference model represents a set of seven layers where data passes through during communication between connected devices.
- The seven-layered structure of the OSI model is reliable and widely used in several communication protocols.



Application layer

It serves as a window for users and application processes to access network services. The common functions of the layers are resource sharing, remote file access, network management, electronic messages and so on.

Presentation layer

The most important function of this layer is defining data formats such as ASCII text, EBCDIC text BINARY, BCD and JPEG. It acts as a translator for data into a format used by the application layer at the receiving end of the station.

Session layer

It allows to establishing, communicating and terminating sessions between processes running on two different devices performing security, name recognition and logging.

Transport layer

The transport layer ensures that messages are delivered error-free, in sequence, and without loss or duplication. It relieves the higher layer from any concern with the transfer of data between them and their peers.

Network layer

It provides end to end logical addressing system so that a packet of data can be routed across several layers and establishes, connects and terminates network connections.

Data link layer

It packages raw data into frames transferred from physical layer. This layer is responsible for transferring frames from one device to another without errors. After sending the frame it waits for the acknowledgement from receiving device. Data link layer has two sub layers:

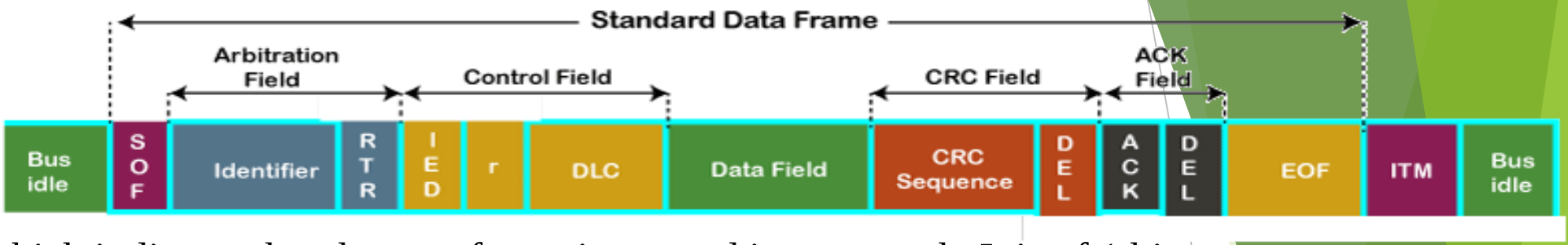
MAC (Medium Access Control) layer: It performs frame coding, error detection, signaling, serialization and de-serialization.

LLC (Logical Link Control) layer: The LLC sub layer provides multiplexing mechanisms that make it possible for several network protocols (IP, Decnet and Appletalk) to coexist within a multipoint network and to be transported over the same network medium. It performs the function of multiplexing protocols transmitted by MAC layer while transmitting and decoding when receiving and providing node-to-node flow and error control.

Physical layer

- The physical layer transmits bit from one device to another and regulates the transmission of bit streams.
- It defines the specific voltage and the type of cable to be used for transmission protocols.
- It provides the hardware means of sending and receiving data on a carrier defining cables, cards and physical aspects
- CAN protocol uses lower two layers of OSI i.e. physical layer and data link layer. The remaining five layers that are communication layers are left out by BOSCH CAN specification for system designers to optimize and adapt according to their needs.

CAN Framing



SOF: start of frame, which indicates that the new frame is entered in a network. It is of 1 bit.

Identifier: A standard data format defined under the CAN 2.0 A specification uses an 11-bit message identifier for arbitration. Basically, this message identifier sets the priority of the data frame.

RTR: Remote Transmission Request, which defines the frame type, whether it is a data frame or a remote frame. It is of 1-bit.

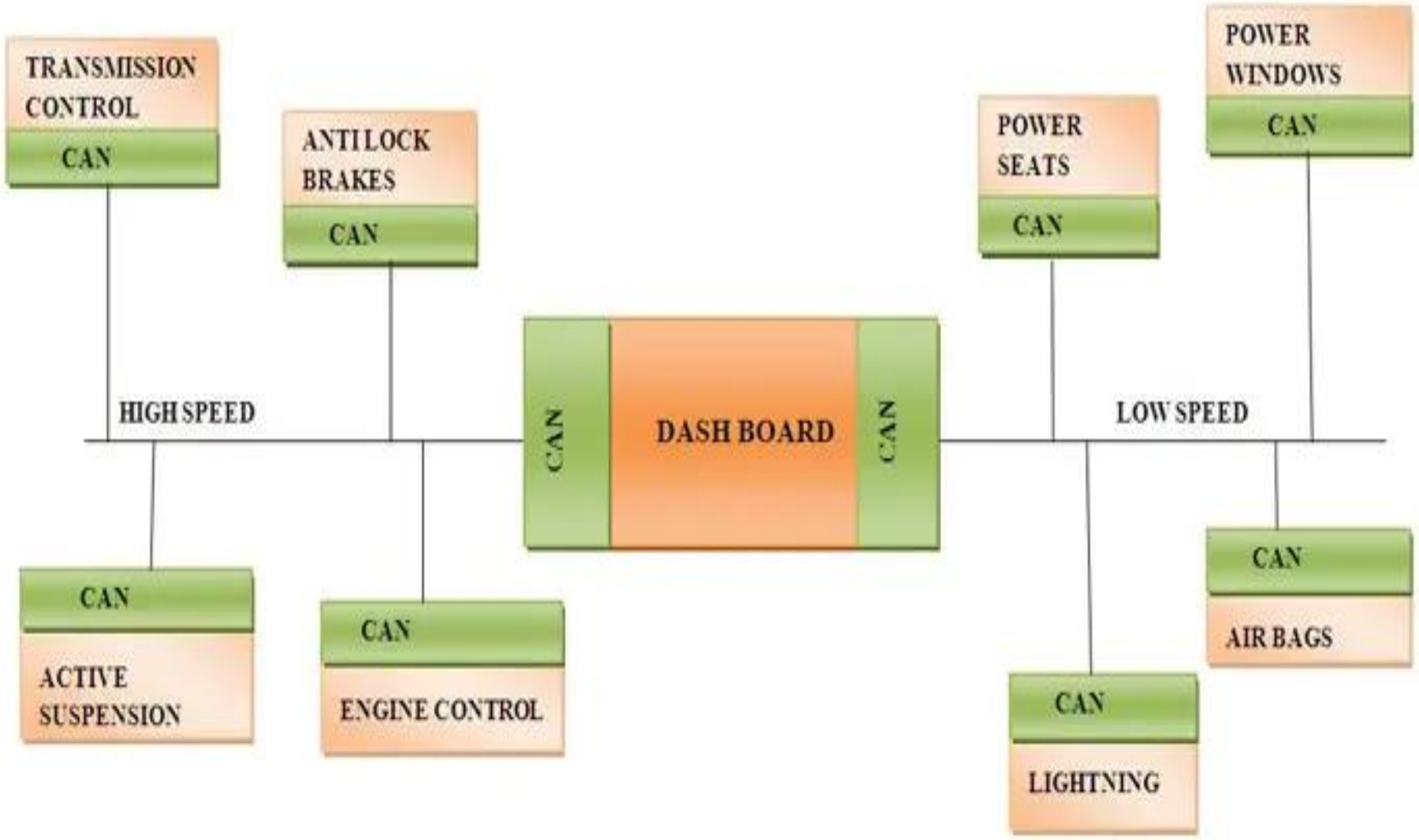
Control field: It has user-defined functions.

- **IDE:** Identifier extension. A dominant IDE bit defines the 11-bit standard identifier, whereas recessive IDE bit defines the 29-bit extended identifier.
- **DLC:** Data Length Code, which defines the data length in a data field. It is of 4 bits.
- **Data field:** The data field can contain upto 8 bytes.

CRC field: The data frame also contains a cyclic redundancy check field of 15 bit, which is used to detect the corruption if it occurs during the transmission time. The sender will compute the CRC before sending the data frame, and the receiver also computes the CRC and then compares the computed CRC with the CRC received from the sender. If the CRC does not match, then the receiver will generate the error.

ACK field: This is the receiver's acknowledgment. In other protocols

EOF: End of frame. It contains 7 consecutive recessive bits known End of frame.

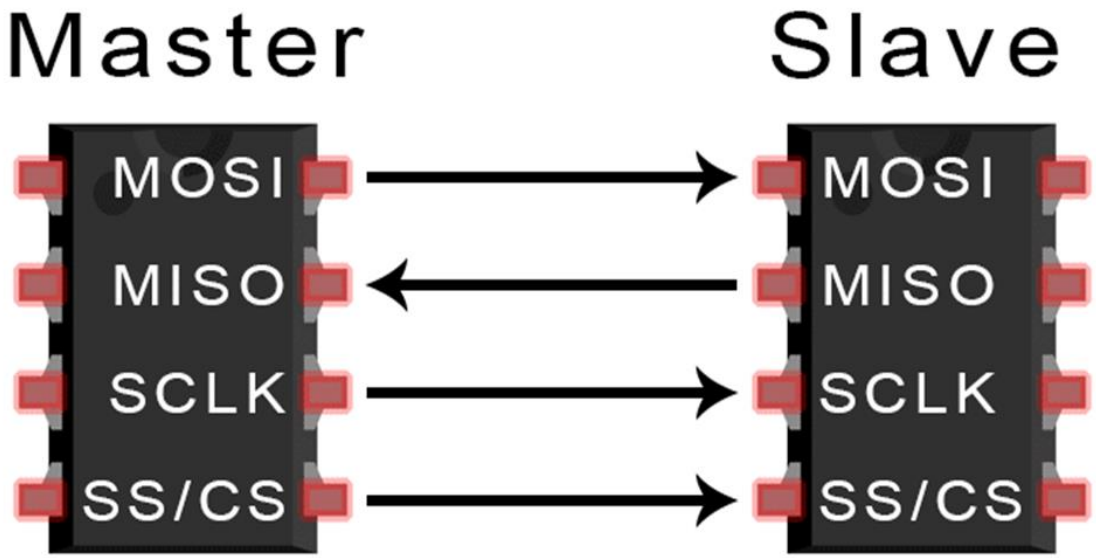

TRANSMISSION CONTROL
CAN
ANTI LOCK BRAKES
CAN
POWER SEATS
CAN
POWER WINDOWS
CAN
HIGH SPEED
CAN
DASH BOARD
CAN
LOW SPEED
CAN
ACTIVE SUSPENSION
CAN
ENGINE CONTROL
CAN
LIGHTNING
CAN
AIR BAGS

SPI COMMUNICATION PROTOCOL

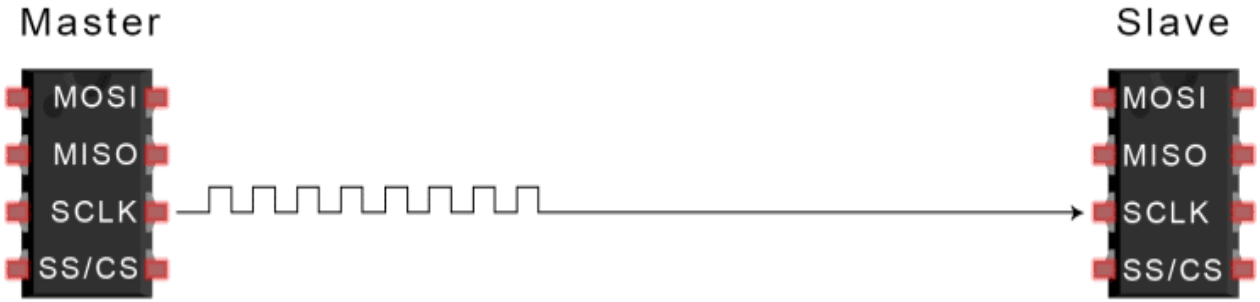
- When you connect a microcontroller to a sensor, display, or other module
- do you ever think about how the two devices talk to each other?
- What exactly are they saying?
- How are they able to understand each other?
- SPI is a common communication protocol used by many different devices.
- For example, SD card reader modules, RFID card reader modules, and 2.4 GHz wireless transmitter/receivers all use SPI to communicate with microcontrollers.
- One unique benefit of SPI is the fact that data can be transferred without interruption.
- Any number of bits can be sent or received in a continuous stream.
- Devices communicating via SPI are in a master-slave relationship.
- The master is the controlling device (usually a microcontroller),
- while the slave (usually a sensor, display, or memory chip) takes instruction from the master.
- The simplest configuration of SPI is a single master, single slave system, but one master can control more than one slave

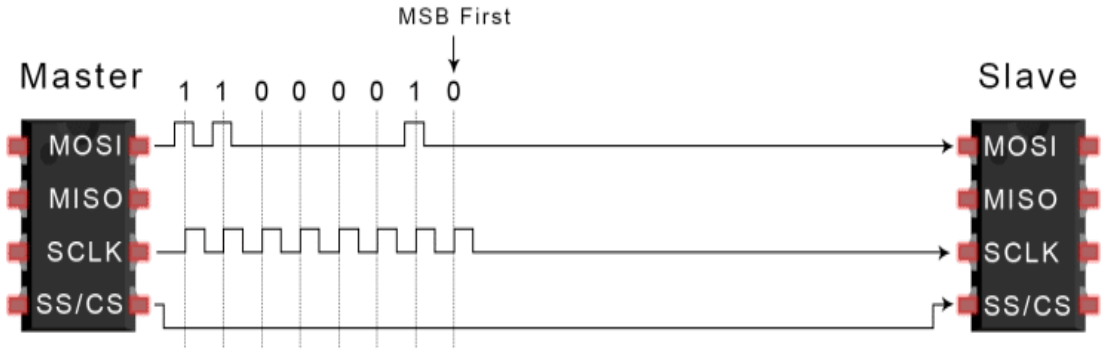
Wires Used	4
Maximum Speed	Up to 10 Mbps
Synchronous or Asynchronous?	Synchronous
Serial or Parallel?	Serial
Max # of Masters	1
Max # of Slaves	Theoretically unlimited*

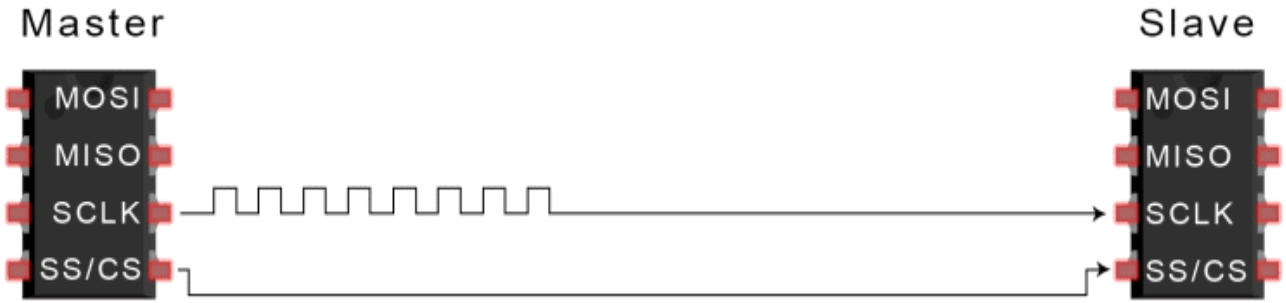
- **MOSI (Master Output/Slave Input)** – Line for the master to send data to the slave.
- **MISO (Master Input/Slave Output)** – Line for the slave to send data to the master.
- **SCLK (Clock)** – Line for the clock signal.
- **SS/CS (Slave Select/Chip Select)** – Line for the master to select which slave to send data to.

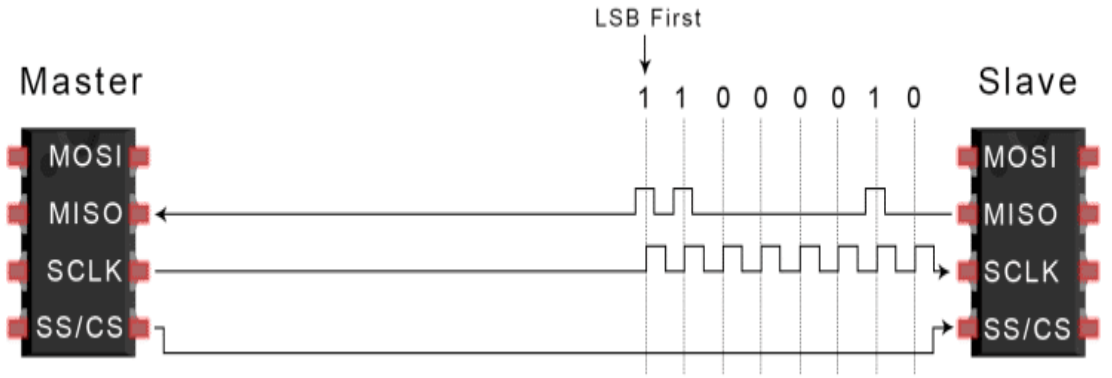


STEPS OF SPI DATA TRANSMISSION









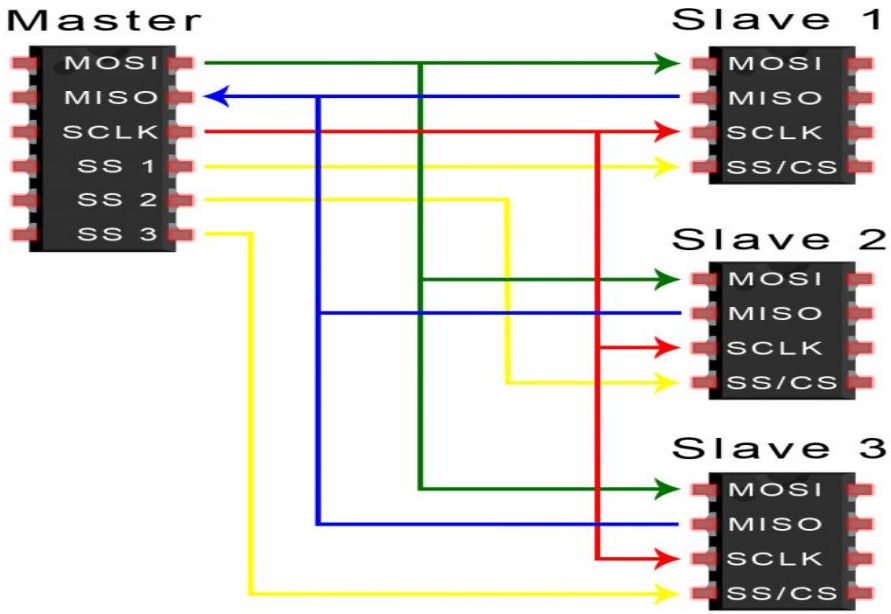
ADVANTAGES

- No start and stop bits, so the data can be streamed continuously without interruption
- No complicated slave addressing system like I2C
- Higher data transfer rate than I2C (almost twice as fast)
- Separate MISO and MOSI lines, so data can be sent and received at the same time

DISADVANTAGES

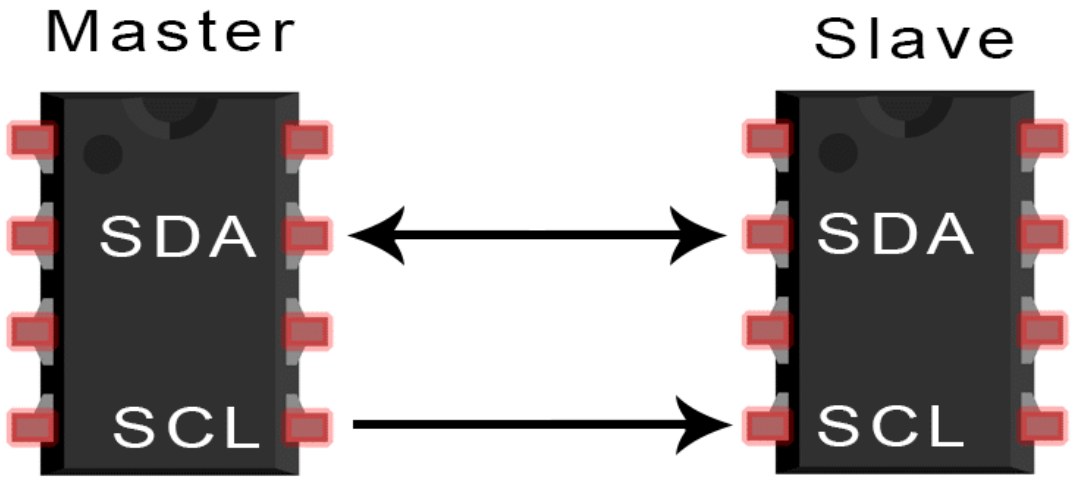
- Uses four wires (I2C and UARTs use two)
- No acknowledgement that the data has been successfully received (I2C has this)
- No form of error checking like the parity bit in UART
- Only allows for a single master

MULTIPLE SLAVES



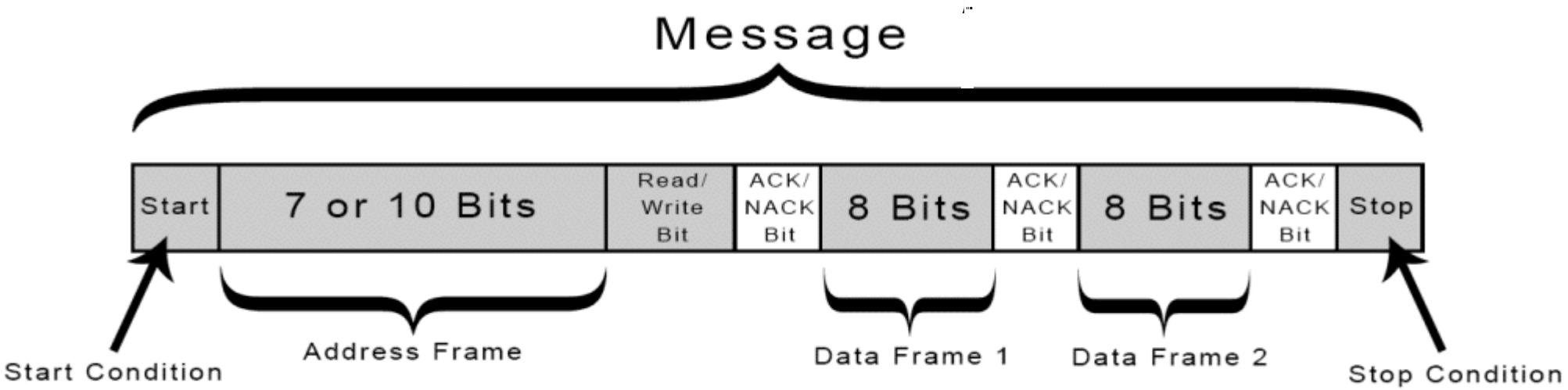
INTRODUCTION TO I2C COMMUNICATION

- I2C combines the best features of SPI and UARTs.
- With I2C, you can connect multiple slaves to a single master (like SPI) and you can have multiple masters controlling single, or multiple slaves.
- This is really useful when you want to have more than one microcontroller logging data to a single memory card or displaying text to a single LCD



HOW I2C WORKS

- With I2C, data is transferred in *messages*.
- Messages are broken up into *frames* of data.
- Each message has an address frame that contains the binary address of the slave, and one or more data frames that contain the data being transmitted.
- The message also includes start and stop conditions, read/write bits, and ACK/NACK bits between each data frame:



Start Condition: The SDA line switches from a high voltage level to a low voltage level *before* the SCL line switches from high to low

Stop Condition: The SDA line switches from a low voltage level to a high voltage level *after* the SCL line switches from low to high.

Address Frame: A 7 or 10 bit sequence unique to each slave that identifies the slave when the master wants to talk to it.

Read/Write Bit: A single bit specifying whether the master is sending data to the slave (low voltage level) or requesting data from it (high voltage level).

ACK/NACK Bit: Each frame in a message is followed by an acknowledge/no-acknowledge bit. If an address frame or data frame was successfully received, an ACK bit is returned to the sender from the receiving device.

STEPS OF I2C DATA TRANSMISSION

1. The master sends the start condition to every connected slave by switching the SDA line from a high voltage level to a low voltage level *before* switching the SCL line from high to low
2. The master sends each slave the 7 or 10 bit address of the slave it wants to communicate with, along with the read/write bit
3. Each slave compares the address sent from the master to its own address. If the address matches, the slave returns an ACK bit by pulling the SDA line low for one bit. If the address from the master does not match the slave's own address, the slave leaves the SDA line high.
4. The master sends or receives the data frame
5. After each data frame has been transferred, the receiving device returns another ACK bit to the sender to acknowledge successful receipt of the frame.
6. To stop the data transmission, the master sends a stop condition to the slave by switching SCL high before switching SDA high.

ADVANTAGES

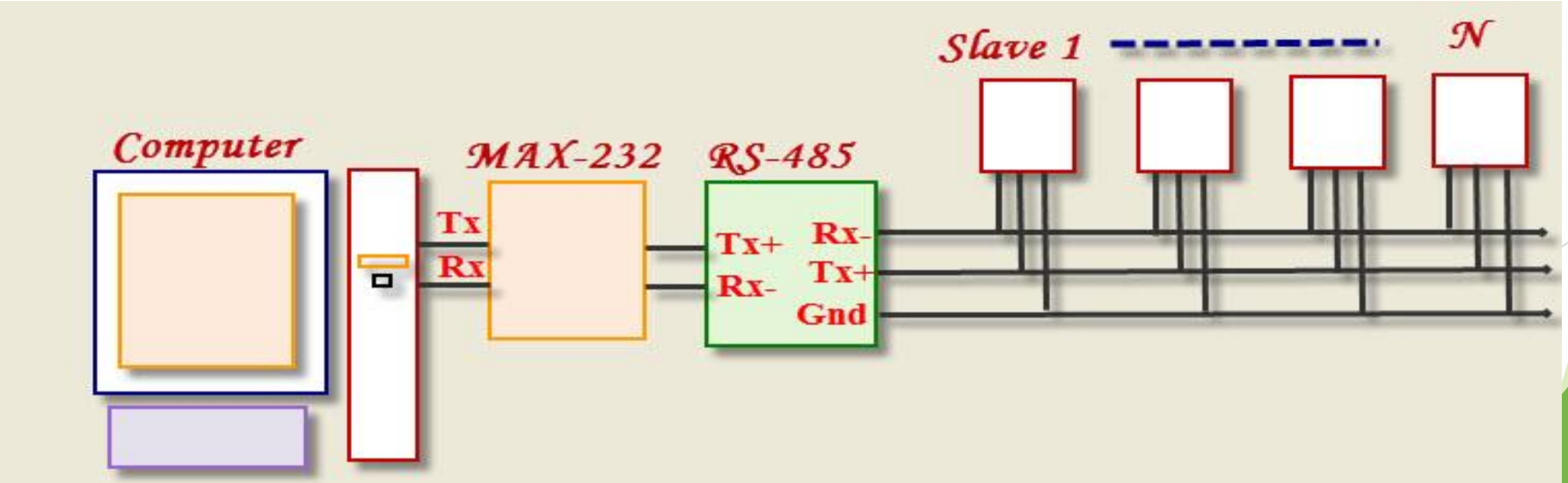
- Only uses two wires
- Supports multiple masters and multiple slaves
- ACK/NACK bit gives confirmation that each frame is transferred successfully
- Hardware is less complicated than with UARTs
- Well known and widely used protocol

DISADVANTAGES

- Slower data transfer rate than SPI
- The size of the data frame is limited to 8 bits
- More complicated hardware needed to implement than SPI

RS-485 Protocol

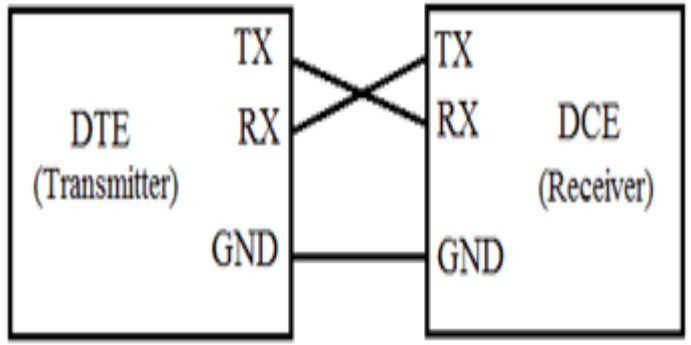
- The Rs-485 is one of the serial communication protocol that is used to send and receive data bit-by-bit sequentially through a serial cable.
- It consists of four wires: Tx+, Tx_, Rx+ and Rx- as a twisted pair cable.
- The RS-485 protocol allows multiple slave devices (EEPROM, ADC) to communicate at a time with a master device (microcontroller or any other controller).
- The RS-485 device consist of 32 drivers that allow 32 devices to communicate at a time.
- It is used to communicate high-speed data rates (1Mbps) and long-distance (1KM).
- The operating voltage of the RS -485 device is -7 to 12v. The RS-485 protocol configured in two ways: “two-wire” or “four wire”.



RS232

RS232 is a **standard protocol used for serial communication**, it is used for connecting computer and its peripheral devices to allow serial data exchange between them. As it obtains the voltage for the path used for the data exchange between the devices

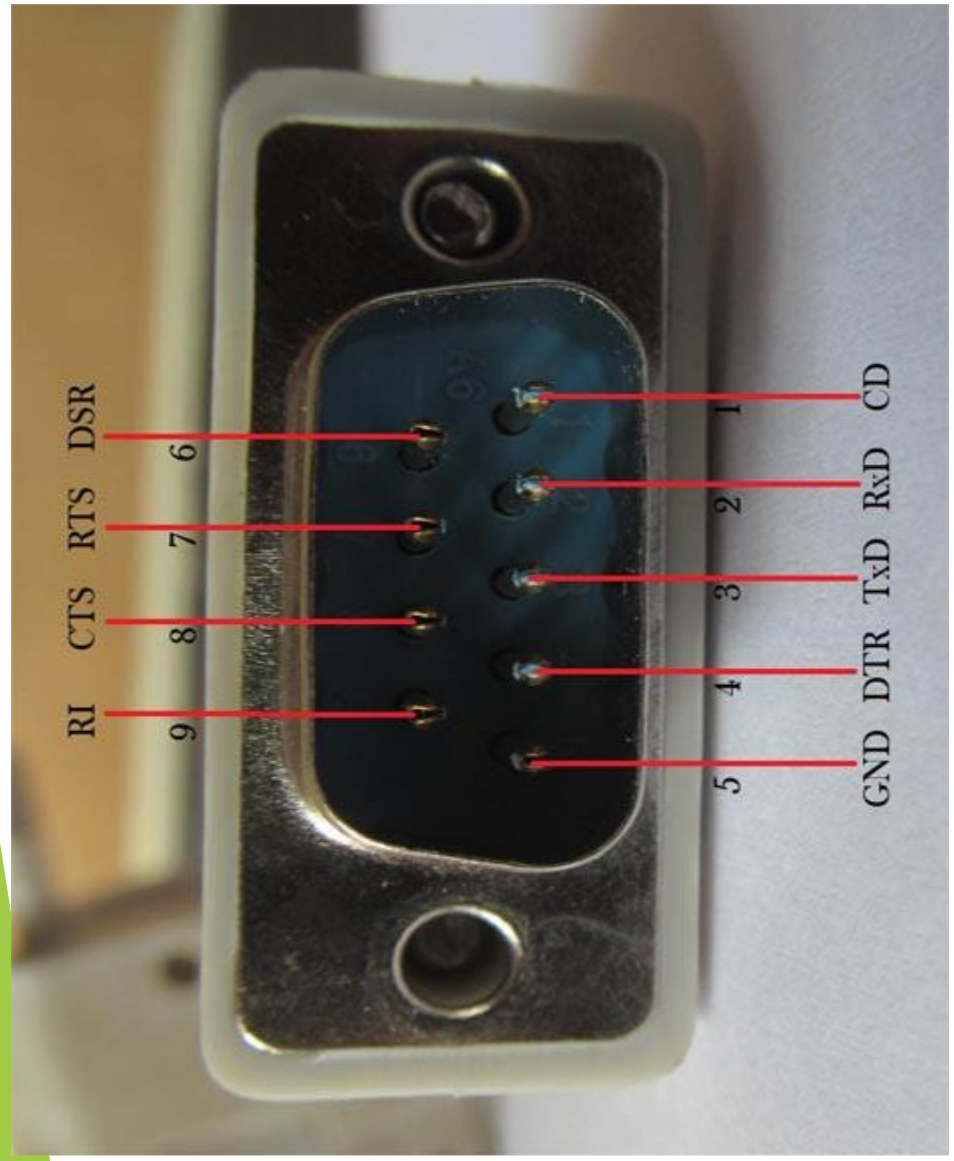
Universal Asynchronous Data Receiver &Transmitter (UART) used in connection with RS232 for transferring data between printer and computer. The microcontrollers are not able to handle such kind of voltage levels, connectors are connected between RS232 signals. These connectors are known as the **DB-9 Connector** as a serial port and they are of two type's **Male connector (DTE) & Female connector (DCE).**

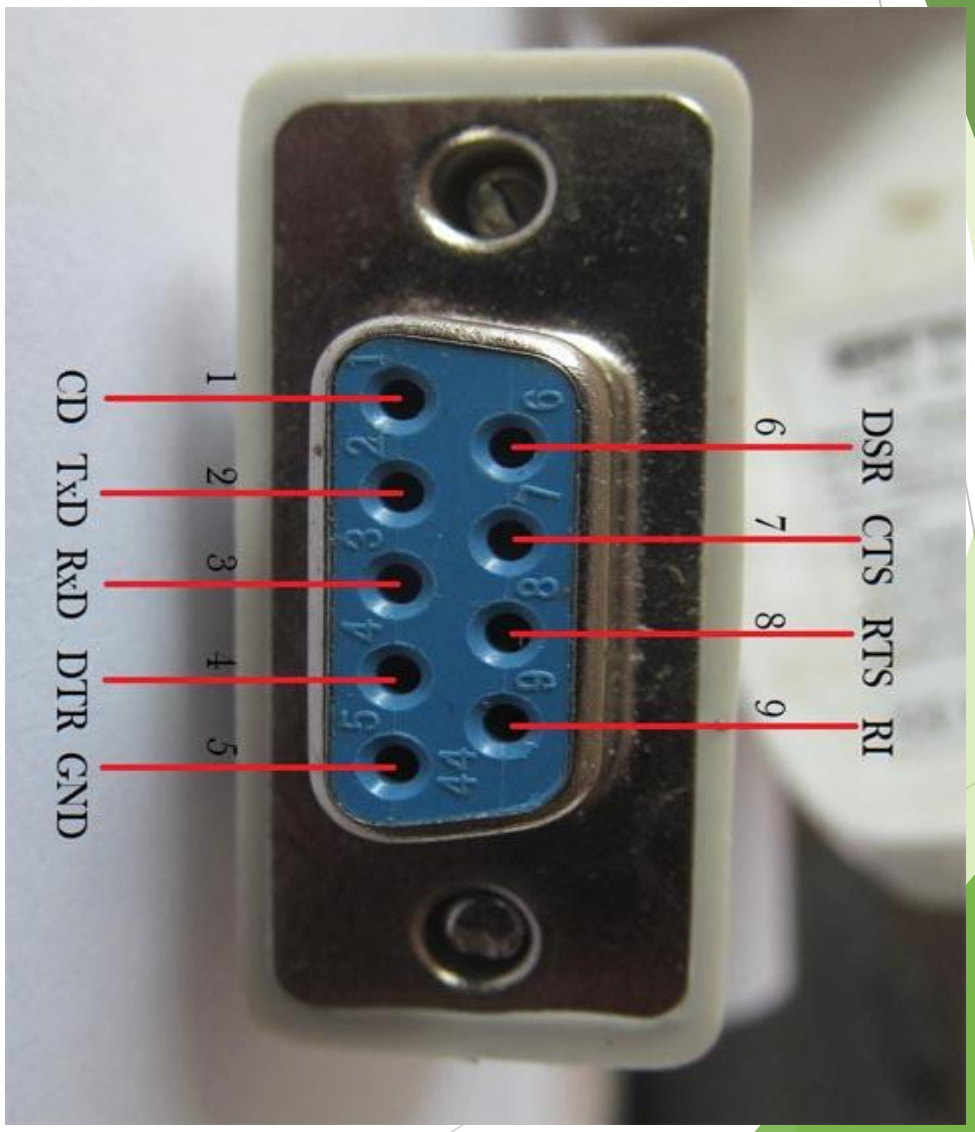


RS232 works on the two-way communication that exchanges data to one another.

- There are two devices connected to each other, **(DTE) Data Transmission Equipment& (DCE) Data Communication Equipment** which has the pins like **TXD, RXD, and RTS& CTS**.
- Now from **DTE** source, the **RTS** generates the *request to send* the data.
- Then from the other side **DCE**, the **CTS,** clears the path for receiving the data.
- After clearing a path, it will give a signal to **RTS** of the **DTE** source to send the signal.
- Then the bits are transmitted from **DTE** to **DCE**.

- Now again from **DCE** source, the request can be generated by **RTS**
- Now **CTS** of **DTE** sources clears the path for receiving the data and gives a signal to send the data.
- This is the whole process through which data transmission takes place.

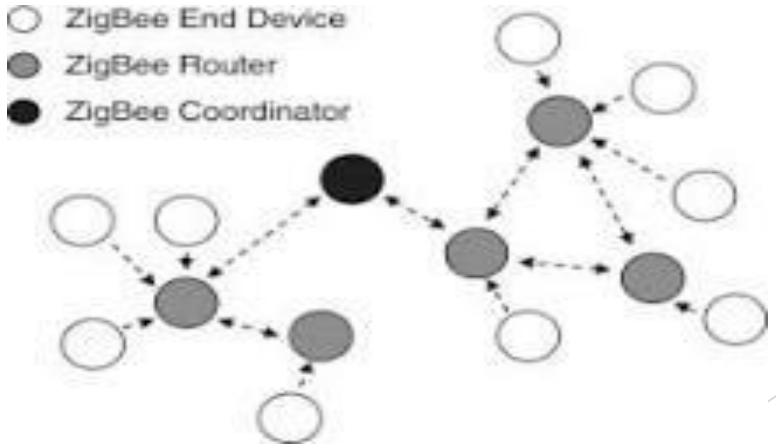




Zigbee

- Zigbee is a wireless technology developed as an open global market connectivity standard to address the unique needs of low-cost, low-power wireless IoT data networks.
- The Zigbee connectivity standard operates on the IEEE 802.15.4
- ZigBee protocol operates globally on a single frequency of 2.4 GHz.
- ZigBee offers wireless range of 70m indoors and 400m outdoors.
- It offers networking flexibility to covers homes of all size by offering support for multiple networks like point-to-point, point-to-multipoint mesh-networks.
- There are 3 types of Zigbee

1. ZigBee Coordinator (ZC It communicates with routers. This device is used for connecting the devices.)
2. ZigBee Router (ZR It is used for passing the data between devices.)
3. ZigBee End Device (ZED It is the device that is going to be controlled.)



General Characteristics of Zigbee Standard:

- Low Power Consumption
- Low Data Rate (20- 250 kbps)
- Short-Range (75-100 meters)
- Network Join Time (~ 30 msec)
- Support Small and Large Networks (up to 65000 devices (Theory); 240 devices (Practically))
- Low Cost of Products and Cheap Implementation (Open Source Protocol)
- Extremely low duty cycle.
- 3 frequency bands with 27 channels.

Operating Frequency Bands (Only one channel will be selected for use in a network):

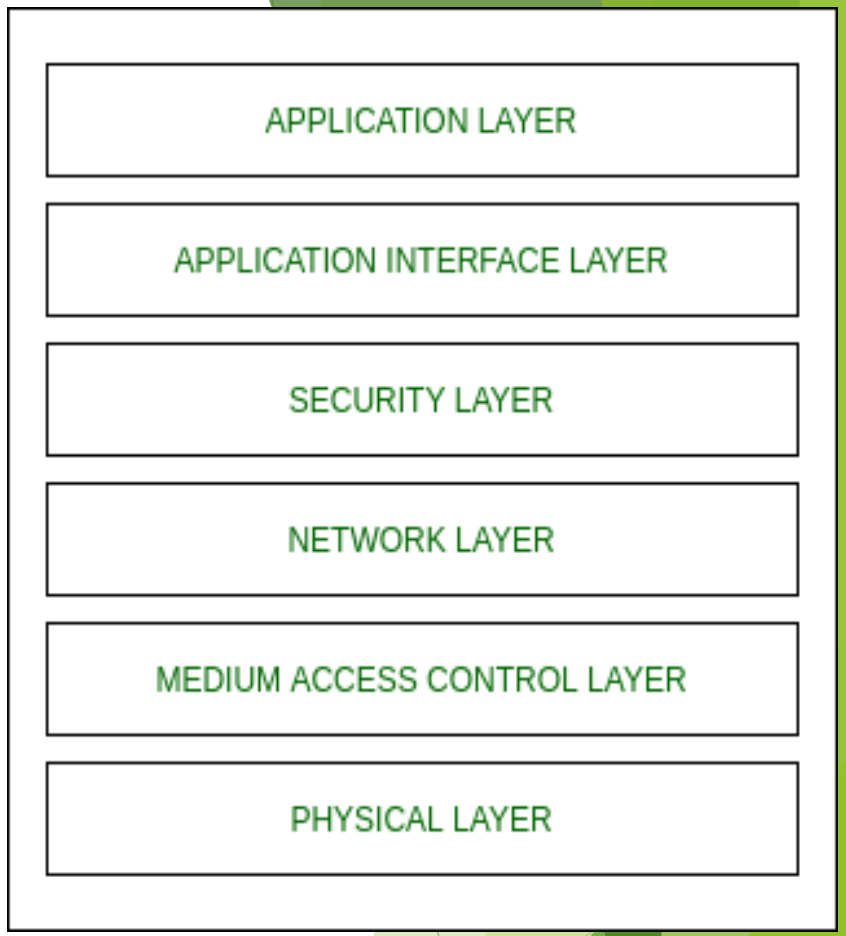
1. **Channel 0**: 868 MHz (Europe)
2. **Channel 1-10**: 915 MHz (the US and Australia)
3. **Channel 11-26**: 2.4 GHz (Across the World)

Zigbee Network Topologies:

- **Star Topology** (ZigBee Smart Energy): Consists of a coordinator and several end devices, end devices communicate only with the coordinator.
- **Mesh Topology** (Self Healing Process): Mesh topology consists of one coordinator, several routers, and end devices.
- **Tree Topology**: In this topology, the network consists of a central node which is a coordinator, several routers, and end devices. the function of the router is to extend the network coverage.

Architecture of Zigbee

- **Physical layer:** The lowest two layers i.e the physical and the MAC (Medium Access Control) Layer are defined by the IEEE 802.15.4 specifications. The Physical layer is closest to the hardware and directly controls and communicates with the Zigbee radio. The physical layer translates the data packets in the over-the-air bits for transmission and vice-versa during the reception.
- **Medium Access Control layer (MAC layer):** The layer is responsible for the interface between the physical and network layer. The MAC layer is also responsible for providing PAN ID and also network discovery through beacon requests.
- **Network layer:** This layer acts as an interface between the MAC layer and the application layer. It is responsible for mesh networking.
- **Application layer:** The application layer in the Zigbee stack is the highest protocol layer and it consists of the application support sub-layer and Zigbee device object. It contains manufacturer-defined applications.

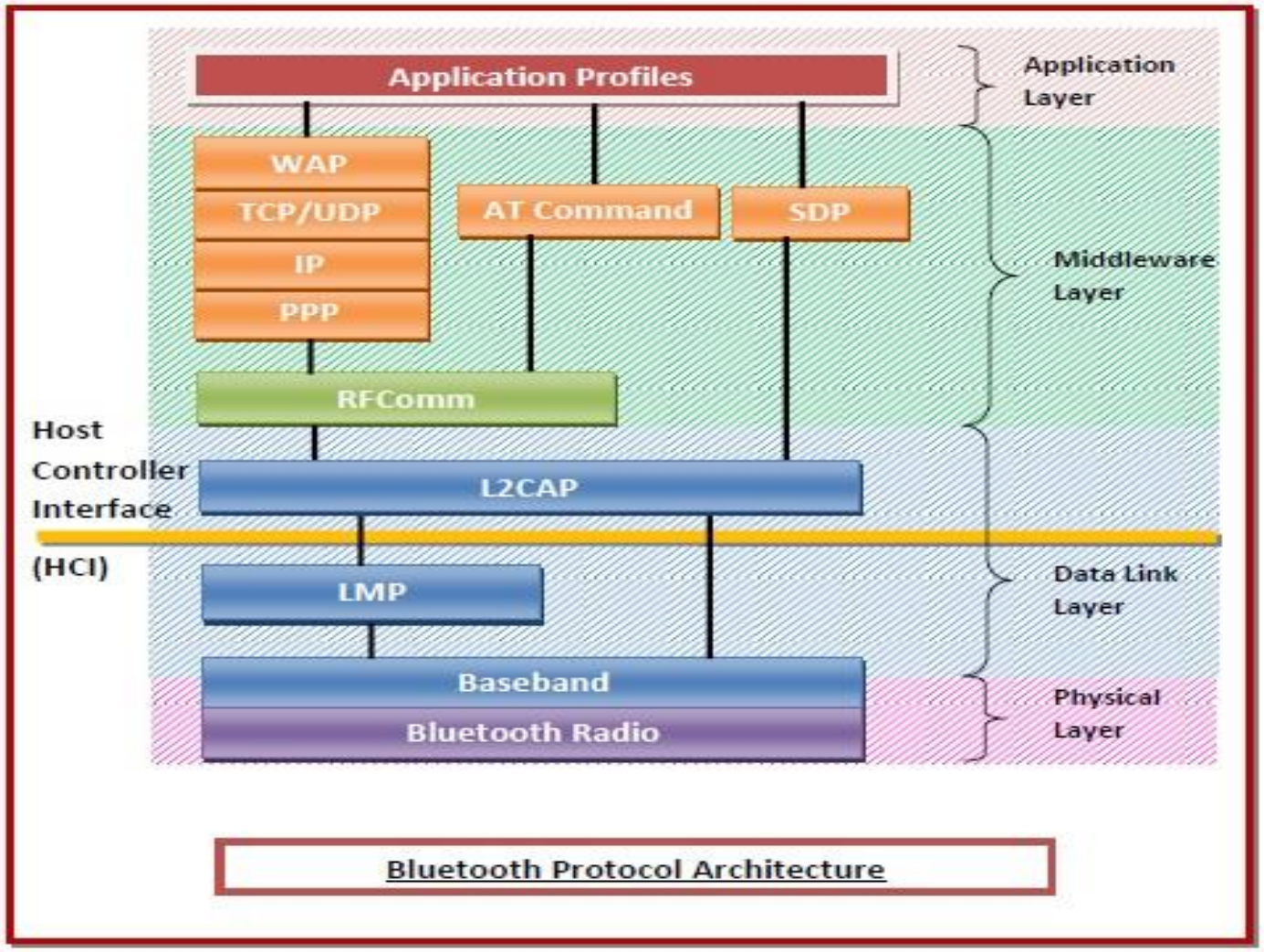


Zigbee Applications:

1.Home Automation
2.Medical Data Collection
3.Industrial Control Systems
4.meter reading system
5.light control system

Bluetooth

- Bluetooth is **a standardized protocol for sending and receiving data via a 2.4GHz wireless link**.
- It's a secure protocol, and it's perfect for short-range, low-power, low-cost, wireless transmissions between electronic devices.



- **Physical Layer** − This includes Bluetooth radio and Baseband (also in the data link layer.
  - **Radio** − This is a physical layer equivalent protocol that lays down the physical structure and specifications for transmission of radio waves. It defines air interface, frequency bands, frequency hopping specifications, and modulation techniques.
  - **Baseband** − This protocol takes the services of radio protocol. It defines the addressing scheme, packet frame format, timing, and power control algorithms.
- **Data Link Layer** − This includes Baseband, Link Manager Protocol (LMP), and Logical Link Control and Adaptation Protocol (L2CAP).
  - **Link Manager Protocol (LMP)** − LMP establishes logical links between Bluetooth devices and maintains the links for enabling communications. The other main functions of LMP are device authentication, message encryption, and negotiation of packet sizes.
  - **Logical Link Control and Adaptation Protocol (L2CAP)** − L2CAP provides adaption between upper layer frame and baseband layer frame format. L2CAP provides support for both connection-oriented as well as connectionless services.
- **Middleware Layer** − This includes Radio Frequency Communications (RF Comm) protocol, adopted protocols, SDP, and AT commands.
  - **RF Comm** − It is short for Radio Frontend Component. It provides a serial interface with WAP.
  - **Adopted Protocols** − These are the protocols that are adopted from standard models. The commonly adopted protocols used in Bluetooth are Point-to-Point Protocol (PPP), Internet Protocol (IP), User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and Wireless Application Protocol (WAP).

- **Service Discovery Protocol (SDP)**− SDP takes care of service-related queries like device information so as to establish a connection between contending Bluetooth devices.
- **AT Commands** − AT tension command set.

❖ **Applications Layer** − This includes the application profiles that allow the user to interact with the Bluetooth applications.

Working Principal

❖ Bluetooth devices are categorised into a master device and slave devices.
❖ In the simplest method, time division multiplexing is used for master – slave communications.
❖ Time slots of 625 µsec are defined the master starts transmitting in odd slots while the slaves start transmitting in even slots.
❖ Length of frames can be 1, 3 or 5 slots.
❖ Each frame is associated with 126-bits overhead for access code and header, as well as a 250 µsec/hop setting time.

Bluetooth Usage

Usage of Bluetooth can be broadly categorized into three areas −

- **Access Points for Data and Voice** − Real-time voice and data transmissions are provided by Bluetooth by connecting portable and stationary network devices wirelessly.
- **Cable replacement** − Bluetooth replaces the need for a large number of wires and cables of wired networks. The connections can be made instantly and are retained even when the devices are not within range. The range of the devices is typically 10m. However, the range can be extended by using amplifiers.
- **Ad hoc networking** − Ad hoc networks are formed impromptu by the network devices bypassing the need for a central access point like a router. Bluetooth networks are ad hoc in nature since a Bluetooth enabled device can form an instant connection with another Bluetooth enabled device as soon as it comes into range.

Bluetooth Applications

- In laptops, notebooks and wireless PCs
- In mobile phones and PDAs (personal digital assistant).
- In printers.
- In wireless headsets.
- In wireless PANs (personal area networks) and even LANs (local area networks)
- To transfer data files, videos, and images and MP3 or MP4.
- In wireless peripheral devices like mouse and keyboards.
- In data logging equipment.
- In the short-range transmission of data from sensors devices to sensor nodes like mobile phones.

TYPICAL EMBEDDED SYSTEMS (UNIT-IV)

CORE OF THE EMBEDDED SYSTEM

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

1) General purpose and Domain Specific Processors
    a) Microprocessors
    b) Microcontroller
    c) Digital Signal Processors
2) Application Specific Integrated Circuits(ASIC)
3) Programmable logic devices(PLD's)
4) Commercial off-the-shelf components (COTs)

Almost 80% of the embedded systems are processor/ controller based. The processor may be microprocessor or a microcontroller or digital signal processor, depending on the domain and application.

a)Microprocessor

A microprocessor is a silicon chip representing a central processing unit. A microprocessor is a dependent unit and it requires the combination of other hardware like memory, timer unit, and *interrupt* controller, etc. for proper functioning.

DIFFERENCES BETWEEN MP AND MC

Microprocessor	Microcontroller
Microprocessors are multitasking in nature. Can perform multiple tasks at a time. For example, on computer we can play music while writing text in text editor.	Single task oriented. For example, a washing machine is designed for washing clothes only.
RAM, ROM, I/O Ports, and Timers can be added externally and can vary in numbers.	RAM, ROM, I/O Ports, and Timers cannot be added externally. These components are to be embedded together on a chip and are fixed in numbers.

Microprocessor	Microcontroller
Designers can decide the number of memory or I/O ports needed	Fixed number for memory or I/O makes a microcontroller ideal for a limited but specific task.
External support of external memory and I/O ports makes a microprocessor-based system heavier and costlier.	Microcontrollers are lightweight and cheaper than a microprocessor.
External devices require more space and their power consumption is higher.	A microcontroller-based system consumes less power and takes less space.

VON-NEUMANN ARCHITECTURE VS HARVARD ARCHITECTURE

Von-Neumann Architecture	Harvard Architecture
Single memory to be shared by both code and data.	Separate memories for code and data.
Processor needs to fetch code in a separate clock cycle and data in another clock cycle. So it requires two clock cycles.	Single clock cycle is sufficient, as separate buses are used to access code and data.
Higher speed, thus less time consuming.	Slower in speed, thus more time-consuming.
Simple in design.	Complex in design.

CISC	RISC
Larger set of instructions. Easy to program	Smaller set of Instructions. Difficult to program.
Simpler design of compiler, considering larger set of instructions.	Complex design of compiler.
Many addressing modes causing complex instruction formats.	Few addressing modes, fix instruction format.
Instruction length is variable.	Instruction length varies.
Higher clock cycles per second.	Low clock cycle per second.
Emphasis is on hardware.	Emphasis is on software.
Control unit implements large instruction set using micro-program unit.	Each instruction is to be executed by hardware.
Slower execution, as instructions are to be read from memory and decoded by the decoder unit.	Faster execution, as each instruction is to be executed by hardware.
Pipelining is not possible.	Pipelining of instructions is possible, considering single clock cycle.

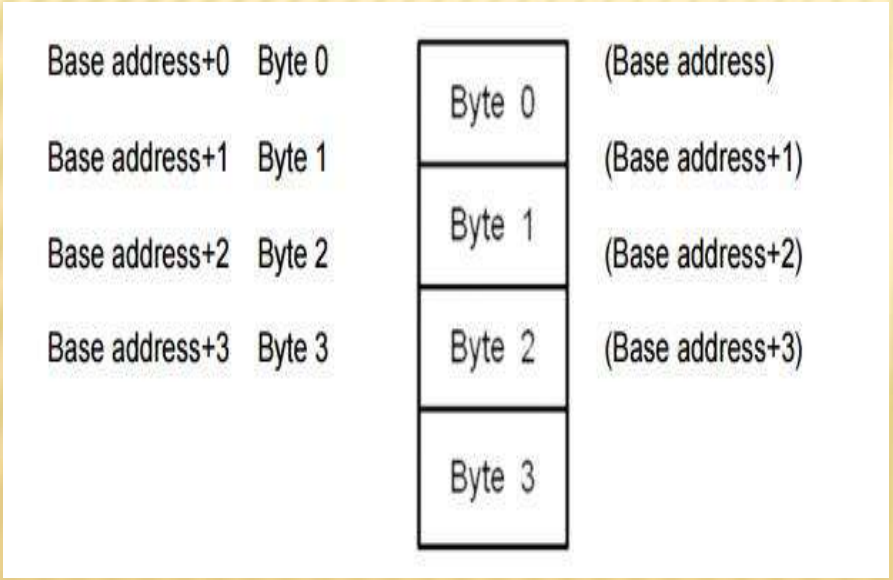
Endiannes

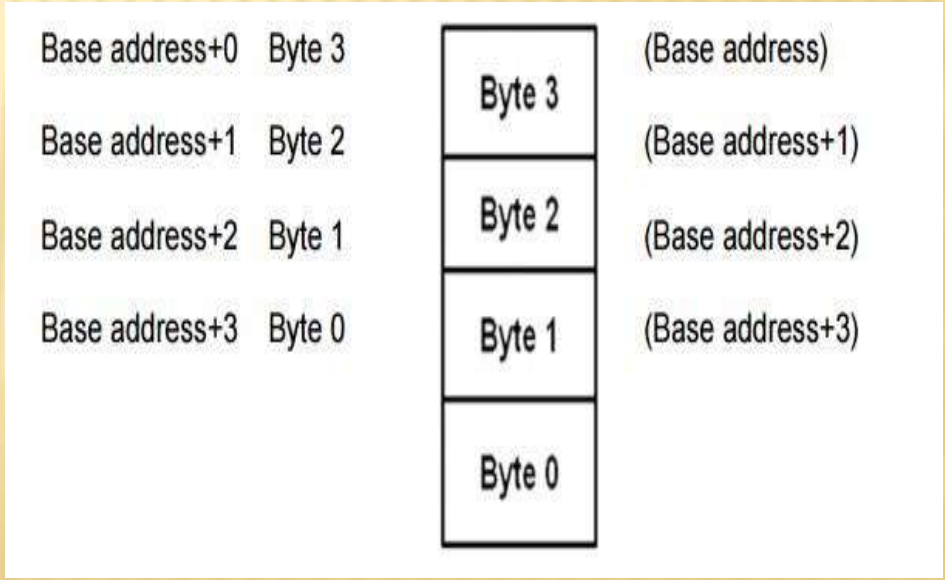
Endianness specifies the order which the data is stored in the memory by processor operations in a multi byte system. Based on Endiannes processors can be of two types:

Little Endian Processors

Big Endian Processors

Little-endian means lower order data byte is stored in memory at the lowest address and the higher order data byte at the highest address. For e.g, 4 byte long integer Byte3, Byte2, Byte1, Byte0 will be store in the memory as follows:





Big-endian means the higher order data byte is stored in memory at the lowest and the lower order data byte at the highest address. For e.g. a 4 byte integer Byte3, Byte2, Byte1, Byte0 will be stored in the memory as follows:

MICROCONTROLLERS.

- A microcontroller is a highly integrated chip that contains a CPU, scratch pad RAM, special and general purpose register arrays, on chip ROM/FLASH memory for program storage, timer and interrupt control units and dedicated I/O ports.
- Texas Instrument's TMS 1000 Is considered as the world's first microcontroller.
- Some embedded system application require only 8 bit controllers whereas some requiring superior performance and computational needs demand 16/32 bit controllers
- The instruction set of a microcontroller can be RISC or CISC.
- Microcontrollers are designed for either general purpose application requirement or domain specific application requirement.

Digital Signal Processors

- DSP are powerful special purpose 8/16/32 bit microprocessor designed to meet the computational demands and power constraints of today's embedded audio, video and communication applications
- DSP are 2 to 3 times faster than general purpose microprocessors in signal processing applications. This is because of the architectural difference between DSP and general purpose microprocessors.
- DSPs implement algorithms in hardware which speed up the execution where as general purpose processors implement the algorithm in software and the speed of execution depends primarily on the clock for the processor.

DSP includes following key units:

Program memory: It is a memory for storing the program required by DSP to process the data.

Data memory: It is a working memory for storing temporary variables and data/signal to be processed.

Computational engine: It performs the signal processing in accordance with the stored program memory computational engine incorporated many specialized arithmetic units and each of them operates simultaneously to increase the execution speed. It also includes multiple hardware shifters for shifting operands and saves execution time.

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

Examples: Audio video signal processing, telecommunication and multimedia applications.

SOP(Sum of Products) calculation, convolution, FFT(Fast Fourier Transform), DFT(Discrete Fourier Transform), etc are some of the operation performed by DSP.

Application Specific Integrated Circuits. (ASIC)

- ASICs is a microchip design to perform a specific and unique applications.
- Because of using single chip to integrate several functions there by reduces the system development cost.
- Most of the ASICs are proprietary (which having some trade name) products, it is referred as Application Specific Standard Products(ASSP)
- As a single chip ASIC consumes a very small area in the total system. Thereby helps in the design of smaller system with high capabilities or functionalities.
- The developers of such chips may not be interested in revealing the internal detail of it .

Programmable logic devices(PLD's)

- A PLD is an electronic component. It used to build digital circuits which are reconfigurable.
- A logic gate has a fixed function but a PLD does not have a defined function at the time of manufacture.
- PLDs offer customers a wide range of logic capacity, features, speed, voltage characteristics.
- PLDs can be reconfigured to perform any number of functions at any time.
- A variety of tools are available for the designers of PLDs which are inexpensive and help to develop, simulate and test the designs.
- PLDs having following two major types.

1) CPLD(Complex Programmable Logic Device):

- CPLDs offer much smaller amount of logic up to 1000 gates.

2) FPGAs(Field Programmable Gate Arrays):

- It offers highest amount of performance as well as highest logic density, the most features.

Advantages of PLDs :-

- PLDs offer customer much more flexibility during the design cycle.
- PLDs do not require long lead times for prototypes or production parts because PLDs are
- already on a distributors shelf and ready for shipment.
- PLDs can be reprogrammed even after a piece of equipment is shipped to a customer

Commercial off-the-shelf components(COTs)

- A Commercial off the Shelf product is one which is used 'as-is'.
- The COTS components itself may be develop around a general purpose or domain specific processor or an ASICs or a PLDs.
- The major advantage of using COTS is that they are readily available in the market, are chip and a developer can cut down his/her development time to a great extent
- The major drawback of using COTS components in embedded design is that the manufacturer of the COTS component may withdraw the product or discontinue the production of the COTS at any time if rapid change in technology occurs.

Advantages of COTS:

Ready to use

Easy to integrate

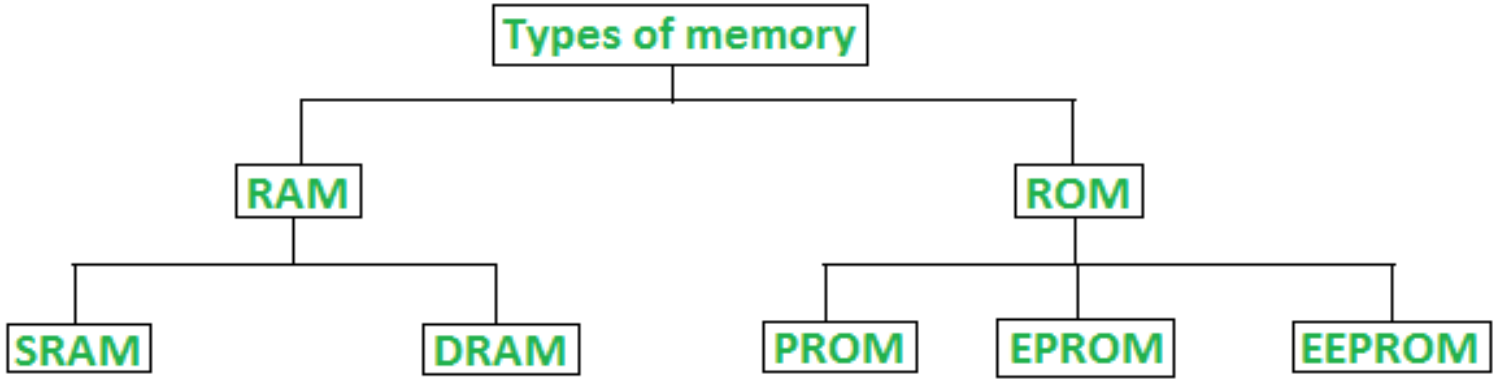
Reduces development time

Disadvantages of COTS:

- No operational or manufacturing standard (all proprietary)
- Vendor or manufacturer may discontinue production of a particular COTS product
- EXAMPLE: Hardware units of remote controlled toy car
- control units including RF circuitry
- High performance, high frequency microwave electronics.

MEMORY

Memory is an important part of processor/controller based embedded system.Some of the processors/controllers contains built in memory and this memory is referred as on-chip memory and off-chip memory.



Classification of computer memory

1. Random Access Memory (RAM)

It is also called as *read write memory* or the *main memory* or the *primary memory*. The programs and data that the CPU requires during execution of a program are stored in this memory. It is a volatile memory as the data loses when the power is turned off.

RAM is further classified into two types- *SRAM (Static Random Access Memory)* and *DRAM (Dynamic Random Access Memory)*.

DRAM	SRAM
1. Constructed of tiny capacitors that leak electricity.	1.Constructed of circuits similar to D flip-flops.
2.Requires a recharge every few milliseconds to maintain its data.	2.Holds its contents as long as power is available.
3.Inexpensive.	3.Expensive.
4. Slower than SRAM.	4. Faster than DRAM.
5. Can store many bits per chip.	5. Can not store many bits per chip.
6. Uses less power.	6.Uses more power.
7.Generates less heat.	7.Generates more heat.
8. Used for main memory.	8. Used for cache.

Difference between SRAM and DRAM

2. Read Only Memory (ROM)

- Stores crucial information essential to operate the system, like the program essential to boot the computer. It is not volatile. Always retains its data.
- Used in embedded systems or where the programming needs no change.
- Used in calculators and peripheral devices.
- ROM is further classified into 4 types- *ROM*, *PROM*, *EPROM*, and *EEPROM*.

Types of Read Only Memory (ROM)

PROM (Programmable read-only memory) – It can be programmed by user. Once programmed, the data and instructions in it cannot be changed.

EPROM (Erasable Programmable read only memory) – It can be reprogrammed. To erase data from it, expose it to ultra violet light. To reprogram it, erase all the previous data.

EEPROM (Electrically erasable programmable read only memory) – The data can be erased by applying electric field, no need of ultra violet light. We can erase only portions of the chip.

RAM	ROM
1. Temporary Storage.	1. Permanent storage.
2. Store data in MBs.	2. Store data in GBs.
3. Volatile.	3. Non-volatile.
4.Used in normal operations.	4. Used for startup process of computer.
5. Writing data is faster.	5. Writing data is slower.

Difference between RAM and ROM

MEMORY SHADOWING

- A technique used to increase a computer's speed by using high-speed RAM memory in place of slower ROM memory (RAM is about three times as fast as ROM).
- On PCs, for example, all code to control hardware devices, such as keyboards, is normally executed in a special ROM chip called the *BIOS ROM*.
- However, this chip is slower than the general-purpose RAM that comprises main memory.
- Many PC manufacturers, therefore, configure their PCs to copy the BIOS code into RAM when the computer boots.
- The RAM used to hold the BIOS code is called *shadow RAM*.

Memory Selection

- Selection of suitable memory is very much essential step in high performance applications,
- because the challenges and limitations of the system performance are often decided upon the type of memory architecture.
- Systems memory requirement depend primarily on the nature of the application that is planned to run on the system.
- Memory performance and capacity requirement for low cost systems are small whereas memory throughput can be the most critical requirement in a complex, high performance system.
- Following are the factors that are to be considered while selecting the memory devices,

- ✓ Speed
- ✓ Data storage size and capacity
- ✓ Bus width
- ✓ Latency
- ✓ Power consumption
- ✓ Cost

SENSORS & ACTUATORS

Sensor

- A Sensor is used for taking Input. It is a transducer that converts energy from one form to another for any measurement or control purpose. Ex. Temperature sensor
- **Actuator** Actuator is used for output. It is a transducer that may be either mechanical or electrical which converts signal to corresponding physical actions. Ex. LED (Light Emitting Diode)
- **LED** is a p-n junction diode and contains a **CATHODE** and **ANODE** For functioning the anode is connected to +ve end of power supply and cathode is connected to –ve end of power supply. The maximum current flowing through the **LED** is limited by connecting a **RESISTOR** in series between the power supply and **LED**.
- There are two ways to interface an LED to microprocessor/microcontroller:

- **The Anode of LED is connected to the port pin and cathode to Ground :** In this approach the port pin sources the current to the LED when it is at logic high.
- **The Cathode of LED is connected to the port pin and Anode to Vcc :** In this approach the port pin sources the current to the LED when it is at logic low. Here the port pin sinks the current and the LED is turned ON when the port pin is at Logic low .

COMMUNICATION INTERFACES

For any embedded system, the communication interfaces can broadly classified into:

1) On board Communication Interfaces

These are used for internal communication of the embedded system i.e: communication between different components present on the system.

Common examples of onboard interfaces are:

- Inter Integrated Circuit (I2C)
- Serial Peripheral Interface (SPI)
- Universal Asynchronous Receiver Transmitter (UART)
- 1-Wire Interface
- Parallel Interface commands.

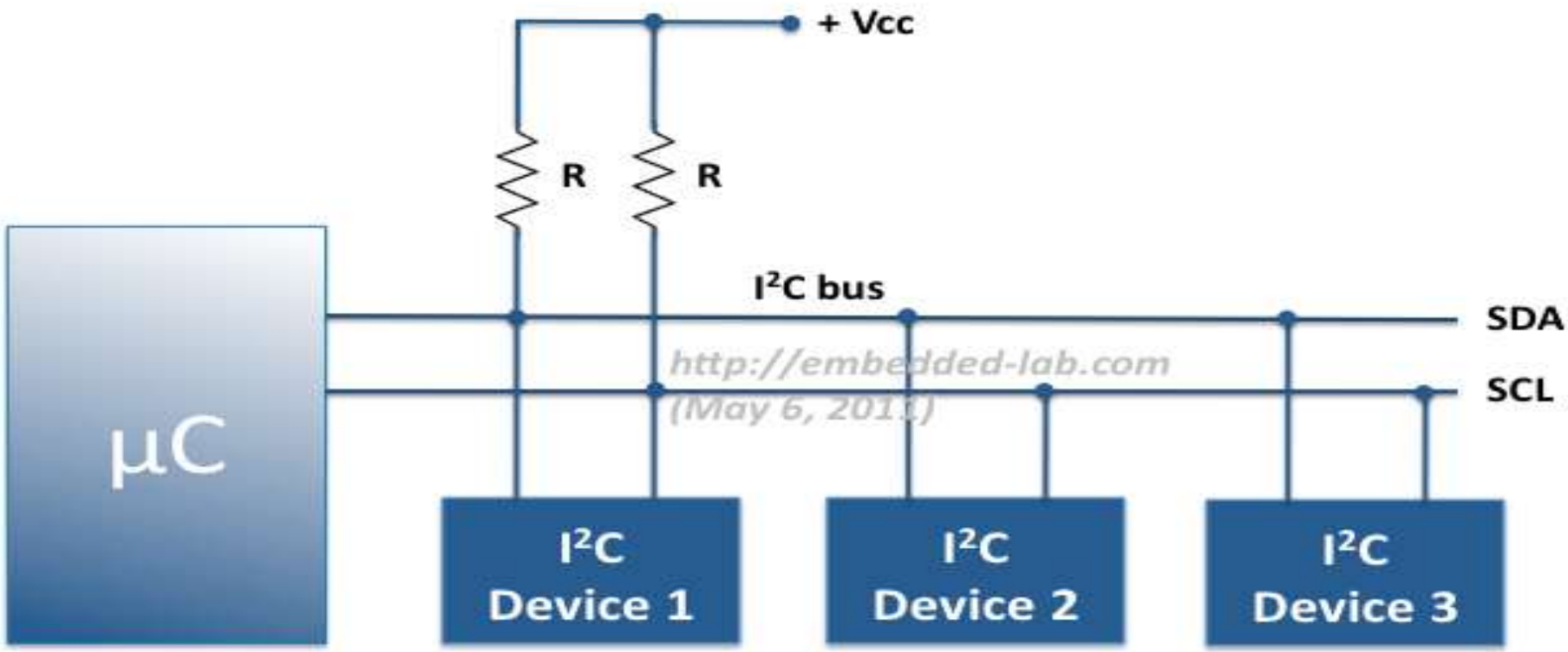
Example :**Inter Integrated Circuit (I2C)**

- It is synchronous
- Bi-directional, half duplex , two wire serial interface bus
- Developed by Phillips semiconductors in 1980

It comprises of two buses :

- Serial clock –SCL
- Serial Data – SDA
- SCL generates synchronization clock pulses
- SDA transmits data serially across devices
- I2C is a shared bus system to which many devices can be connected
- Devices connected by I2C can act as either master or slave

- The master device is responsible for controlling communication by initiating/ terminating data transfer.
- Devices acting as slave wait for commands from the master and respond to those



Multiple devices on common I²C bus

SERIAL PERIPHERAL INTERFACE(SPI) BUS

- Communication Protocol Developed By Motorola
- Four Wire Protocol
- Serial Interface
- Master-Slave Approach
- Synchronous- Data clocked with Clock Signal
- Data Rate-10mbps

SPI Protocol Specifies 4 Signal Wires.

1. Master Out Slave In (MOSI)
2. Master In Slave Out (MISO)
3. Serial Clock (SCLK)
4. Slave Select (SS)

Advantages

- Full Duplex Communication
- Higher Throughput than I2C
- Not Limited to 8 bit words in case of bit transferring
- Arbitrary choice of message size, content and Purpose
- Low Power

Disadvantages

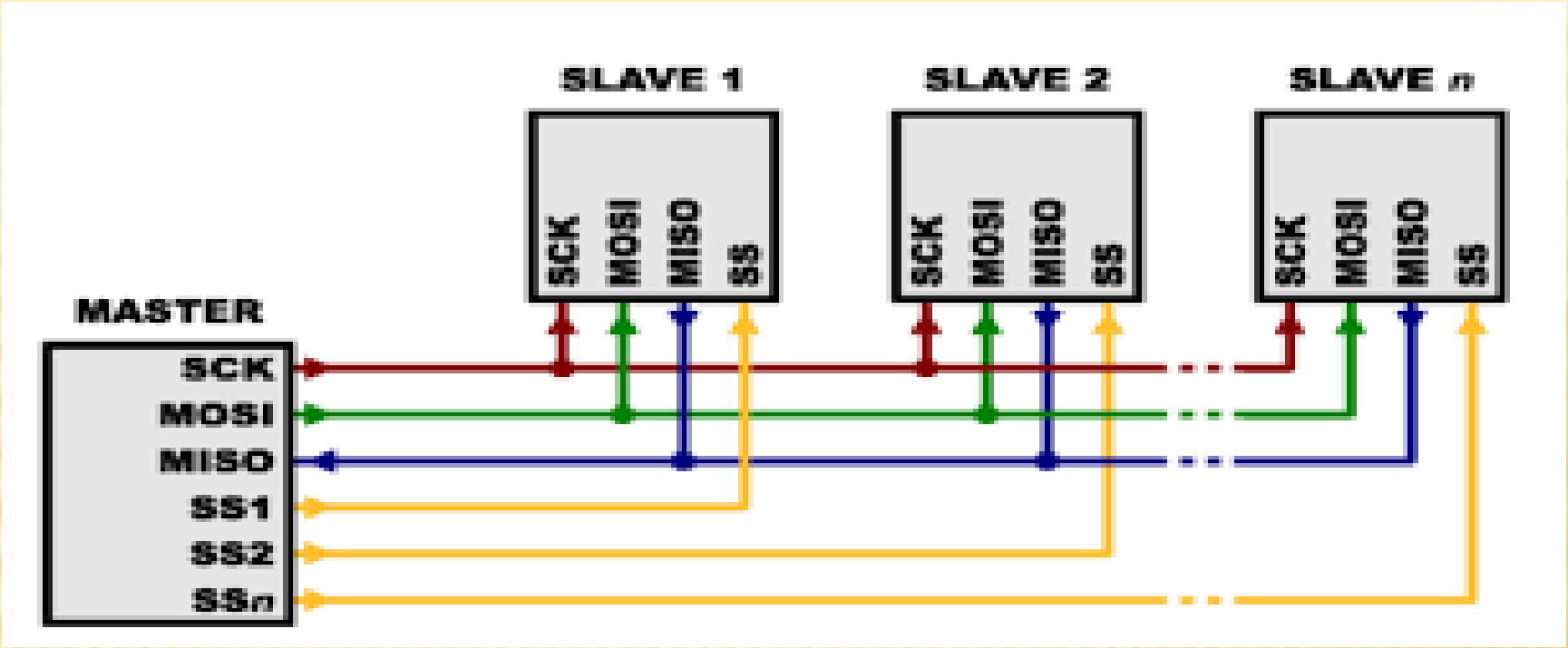
- Requires more pins than I2C
- No hardware flow control
- No Slave Acknowledgement
- Multi Master Difficult to Implement
- Short Distance

How Do They Communicate

- Communication Initiated by Master only
- Master Configures the clock – Frequency less than equal to maximum frequency Slave Support
- Master Selects Slave – By Pulling chip select(SS) of particular Slave-peripheral to Low State

SOME OTHER COMMUNICATION INTERFACE

1. UART
2. 1-WIRE INTERFACE
3. PARALLEL INTERFACE



2)External or Peripheral Communication Interfaces

These are used for external communication of the embedded system i.e: communication of different components present on the system with external or peripheral components/devices.

Common examples of external interfaces are:

RS-232 C & RS-485,Universal Serial Bus (USB), IEEE 1394 (Firewire),Infrared (IrDA), Bluetooth, Wi-Fi, Zig -Bee, General Packet Radio Service (GPRS)

Example: RS-232 C & RS-485

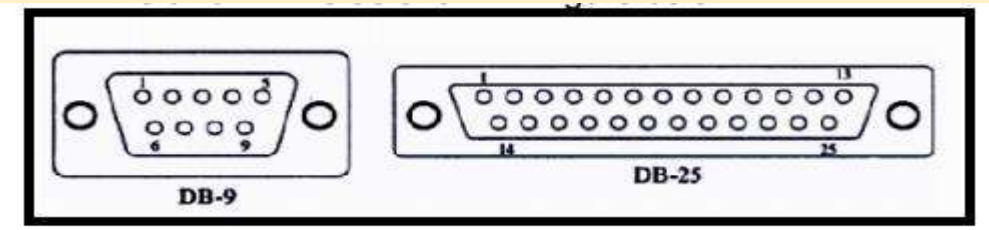
It is wired, asynchronous, serial, full duplex communication

RS 232 interface was developed by EIA (Electronic Industries Associates) In early 1960s

RS 232 is the extension to UART for external communications

RS-232 logic levels use:+3 to +25 volts to signify a "Space" (Logic 0) and-3 to -25 volts to signify a "Mark" (logic 1).RS 232 supports two different types of connectors :

RS 232 interface is a point to point communication interface and the devices involved are called as Data Terminating Equipment (DTE) And Data Communications Equipment (DCE).Embedded devices contain UART for serial transmission and generate signal levels as per TTL/CMOS logic.



DB 9 and DB 25 as shown in figure below

A level translator IC (like Max 232) is used for converting the signal lines from. UART to RS 232 signal lines for communication. The vice versa is performed on the receiving side. Converter chips contain converters for both transmitters and receivers

RS 232 is used only for point to point connections

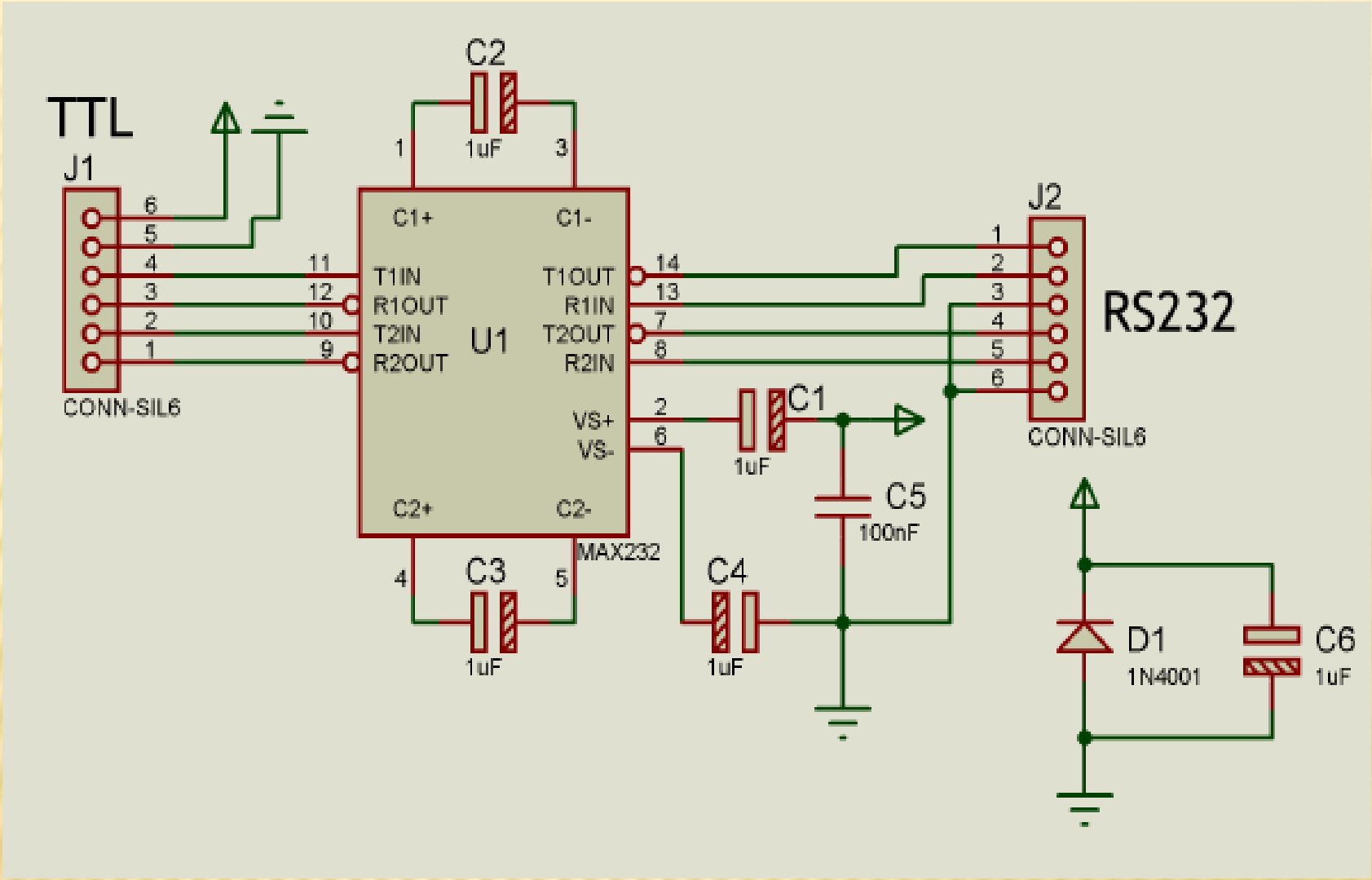
It is susceptible to noise and hence is limited to short distances only

RS 422 is another serial interface from EIA.

It supports multipoint connections with 1 transmitter and 10 receivers.

It supports data rates up to 100Kbps and distance up to 400 ft

RS 485 is enhanced version of RS 422 and supports up to 32 transmitters and 32Receivers.


TTL
J1
CONN-SIL6
C2
1uF
C1+
C1-
T1IN
R1OUT
T2IN
R2OUT
T1OUT
R1IN
T2OUT
R2IN
U1
VS+
VS-
C2+
C2-
MAX232
C3
1uF
C4
1uF
C1
1uF
C5
100nF
J2
CONN-SIL6
RS232
D1
1N4001
C6
1uF

EMBEDDED FIRMWARE (UNIT – V)

EMBEDDED FIRMWARE

Embedded firmware

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

1.Write the program in high level languages like Embedded C/C++ using an Integrated Development Environment.

- The IDE contains an editor, compiler, linker, debugger, simulator, etc.
- IDEs are different for different family of processors/controllers. For example, keil micro vision3 IDE is used for all family members of 8051 microcontroller, Since it contains the generic 8051 compiler C51.

- 2. Write the program in Assembly language using the instructions supported by your application's target processor/controller
- The instruction set for each family of processor/controller is different and the program written in either of the methods given above should be converted into a processor understandable machine code before loading it into the program memory.
- The process of converting the program written in either a high level language or processor/controller specific Assembly code to machine readable binary code is called 'HEX File Creation'.
- If the program is written in Embedded C/C++ using an IDE, the cross compiler included in the IDE converts it into corresponding processor/controller understandable 'HEX File'.
- If you are following the Assembly language based programming technique, you can use the utilities supplied by the processor/controller vendors to convert the source code into 'HEX File'.
- Also third party tools are available, which may be free of cost, for this conversion.

The other system components refer to the components/circuits/ICs which are necessary for the proper functioning of the embedded system.

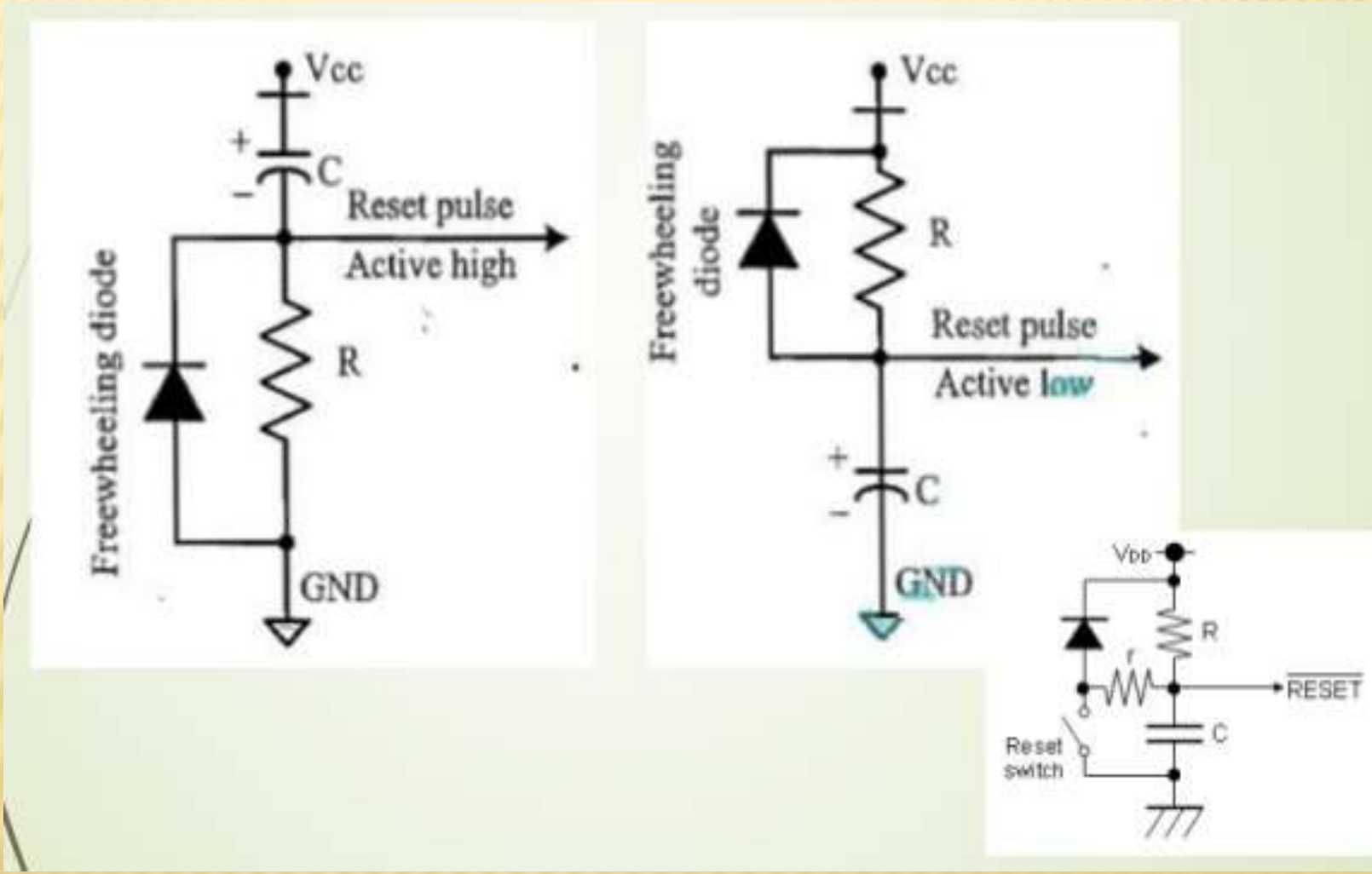
1) Reset Circuit
2) Brown-out Protection Circuit
3) Oscillator Unit
4) Real- Time Clock (RTC)
5) Watchdog Timer are examples of circuits/ICs which are essential for the proper functioning of the processor/controllers.

The reset circuit

- is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON. The reset signal brings the internal registers and the different hardware systems of the processor/controller to a known state and starts the firmware execution from the reset vector

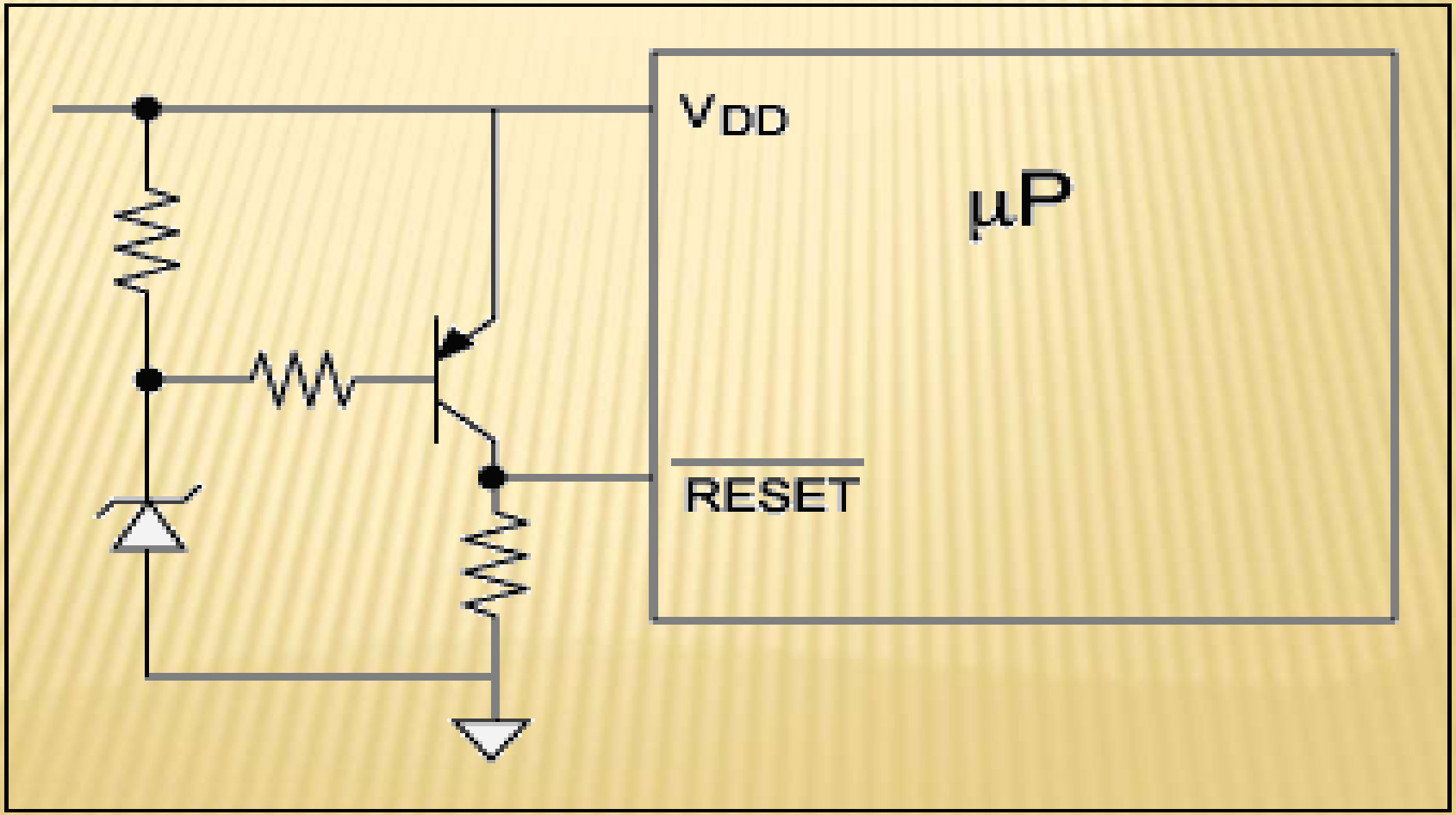
- The reset signal can be either active high (The processor undergoes reset when the reset pin of the processor is at logic high) or active low (The processor undergoes reset when the reset pin of the processor is at logic low).
- Since the processor operation is synchronized to a clock signal, the reset pulse should be wide enough to give time for the clock oscillator to stabilize before the internal reset state starts.
- Some microprocessors/controllers contain built-in internal reset circuitry and they don’t require external reset circuitry.
- Figure illustrates a resistor capacitor based passive reset circuit for active high and low configurations.
- The reset pulse width can be adjusted by changing the resistance value R and capacitance value C.

RC BASED RESET CIRCUIT

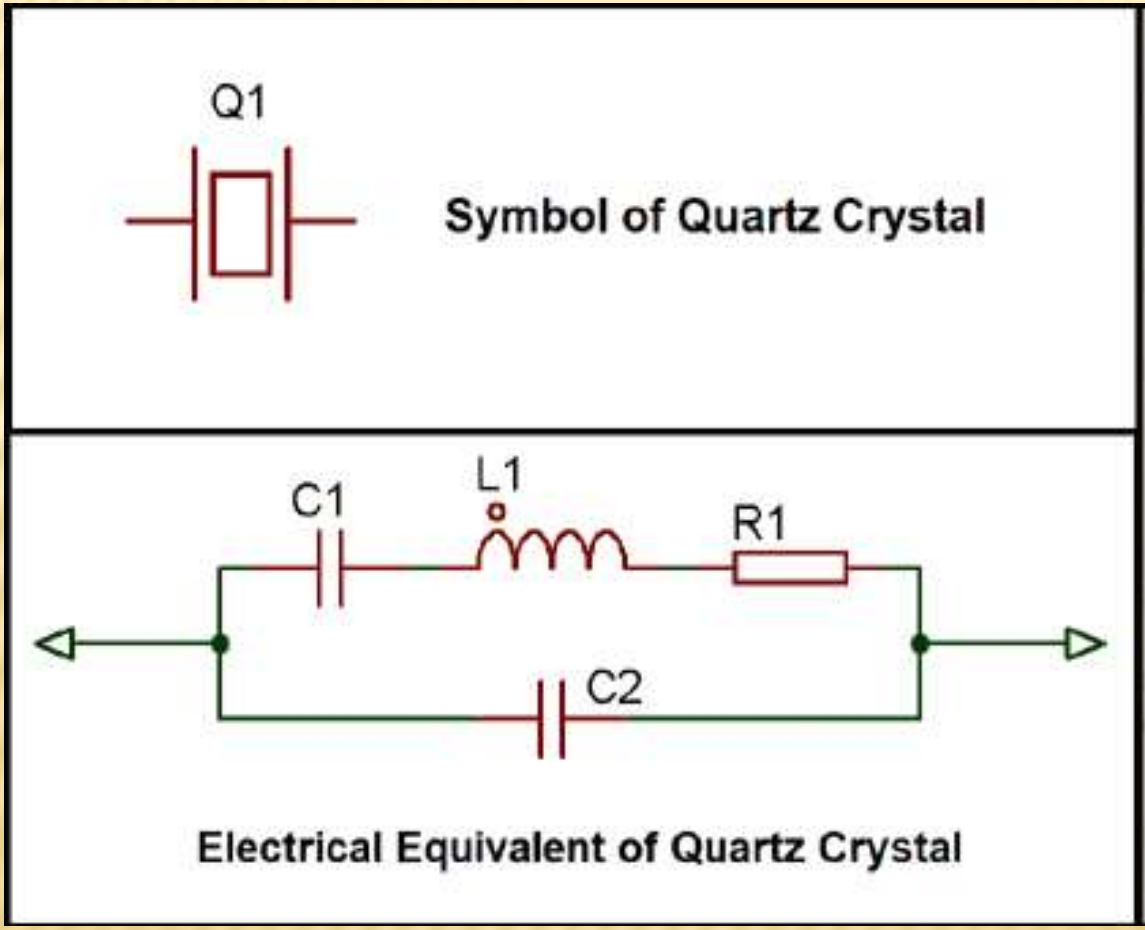


The brown-out protection

circuit prevents the processor/controller from unexpected program execution behavior when the supply voltage to the processor/controller falls below a specified voltage.



The Oscillator unit generates clock signals for synchronizing the operations of the processor.



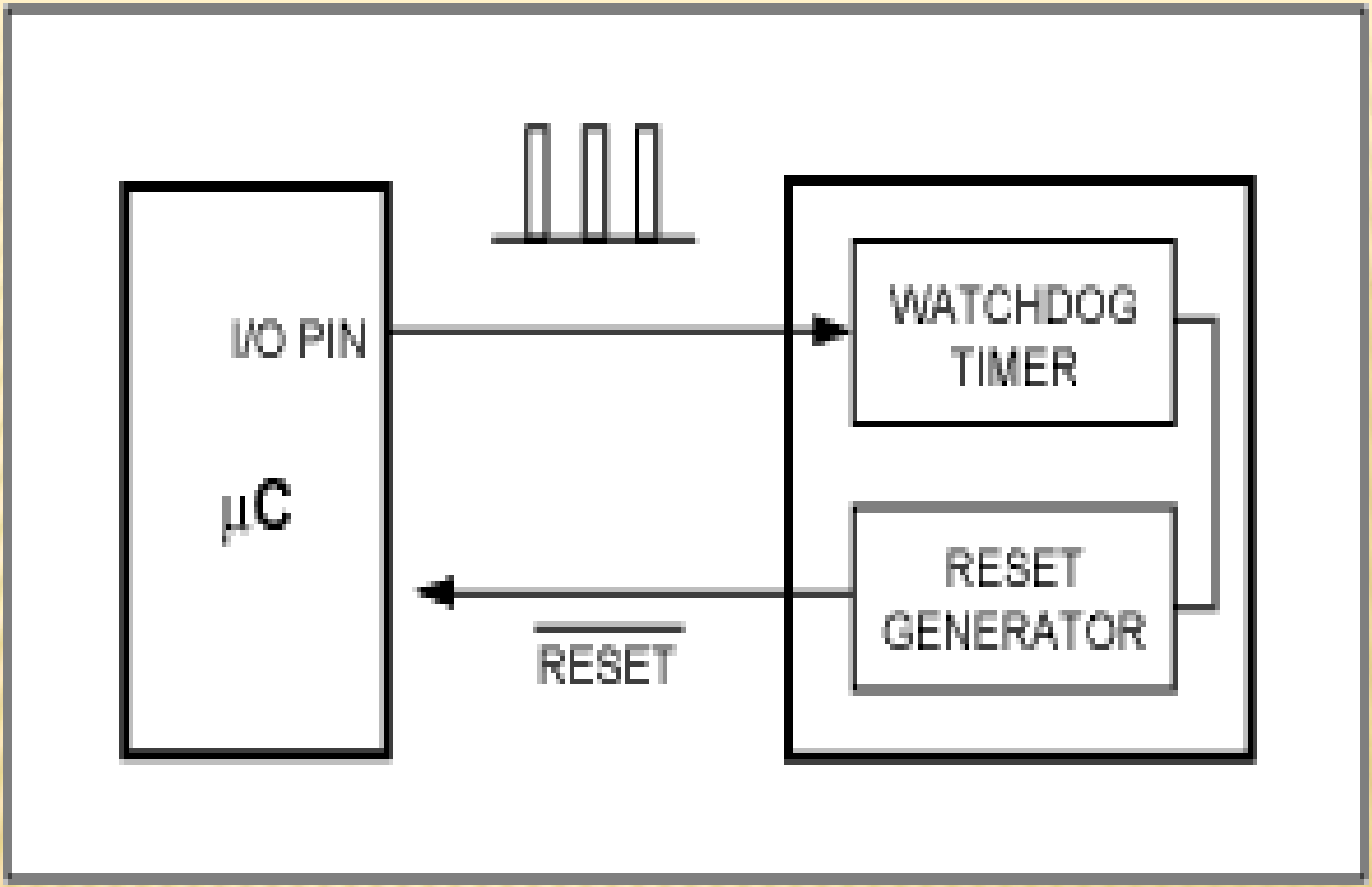
Real-Time Clock (RTC)

- is a system component responsible for keeping track of time. RTC holds information like current time (in hours, minutes and seconds) in 12 hour/24 hour format, date, month, year, day of the week, etc. and supplies timing reference to the system.
- RTC is intended to function even in the absence of power.
- The RTC chip contains a microchip for holding the time and date related information and backup battery cell for functioning in the absence of power, in a single IC package.
- The RTC chip is interfaced to the processor or controller of the embedded system.
- For Operating System based embedded devices, a timing reference is essential for synchronizing the operations of the OS kernel


VCC
GND
CLK
DAT
RST
MH-Real-Time
Clock Modules - 2
Lithium Cell
CR2032
3V

A watchdog

- is to monitor the firmware execution and reset the system processor/microcontroller when the program execution hangs up or generates an Interrupt in case the execution time for a task is exceeding the maximum allowed limit.
- If the firmware execution doesn't complete due to malfunctioning, within the time required by the watchdog to reach the maximum count, the counter will generate a reset pulse and this will reset the processor (if it is connected to the reset line of the processor).
- Most of the processors implement watchdog as a built-in component and provides status register to control the watchdog timer (like enabling and disabling watchdog functioning) and watchdog timer register for writing the count value.
- If the processor/controller doesn't contain a built in watchdog timer, the same can be implemented using an external watchdog timer IC circuit.


I/O PIN
µC
WATCHDOG TIMER
RESET GENERATOR
RESET

EMBEDDED FIRMWARE DESIGN APPROACH

- The firmware design approach for embedded product is purely dependent on the complexity of the functions to be performed etc.

Two basic approaches are used for embedded firmware design.

1) Conventional procedural based firmware design. (Super loop model)
2) Operating system based design.

THE SUPER LOOP BASED APPROACH

- The super loop based firmware development approach is adopted for applications that are not time critical and where the response time is not so important.
- It is similar to a conventional procedural programming where the code is executed task by task.
- The task listed at the top of the program code is executed first and the task below the top are executed after completing the first task.

- In a multiple task based system, each task is executed in serial in this approach. The execution flow for this will be.

1) Configure the common parameters and perform initialisation for various hardware components memory, registers,etc.
2) Start the first task and execute it
3) Execute the second task
4) Execute the next task
5) :
6) :
7) Execute the last defined task
8) Jump back to the first task and fallow the same task

- Almost all tasks in embedded application are non-ending and are repeated infinitely throughout the operation.
- This repetition is achieved by using an infinite loop like WHILE(1) {} loop.
- Since the tasks are running inside the loop, the only way to come out of the loop is either a hardware reset or an interrupt assertion.

- The super loop based design doesn’t require an operating system.
- This type of design is deployed in low cost embedded products and products where response time is not critical.
- The major drawback of this approach is that any failure in any part of a single task will affect the total system.
- If the program hangs up at some point while executing a task
- It will remain there forever and ultimately the product stops functioning.
- Another major drawback of the super loop design approach is the lack of real timeliness.
- If the number of tasks to be executed within the application increases, the time at which each task is repeated also increases.

THE EMBEDDED OPERATING SYSTEM BASED APPROACH

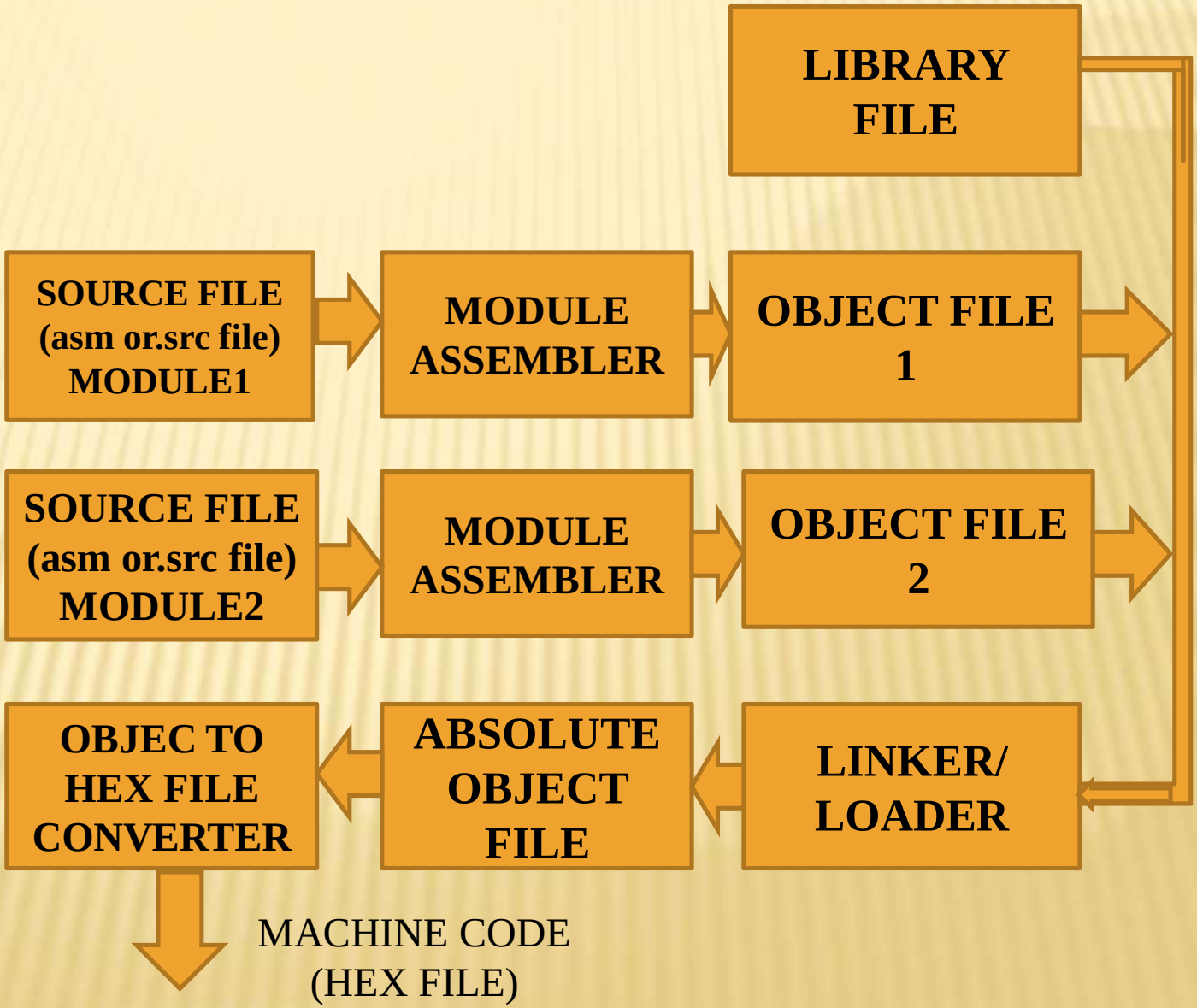
- The operating system based approach contains operating system, which can be either GPOS or RTOS to host the user written application firmware.
- GPOS based design is very similar to a conventional PC based application development where the device contains an OS(Windows/Unix/Linux, etc for desktop PCs) and you will be creating and running user application on top of it.
- OS based applications also requires **DRIVER SOFTWARE** for different hardware present on the board to communicate with them.
- RTOS based design approach is employed in embedded products demanding real time response.
- RTOS responds in a timely and predictable manner to events.
- RTOS contains a real time kernel responsible for performing pre-emptive multitasking, scheduler for scheduling tasks, multiple threads etc.

- A RTOS allows flexible scheduling of system resources like the CPU and memory and offers some way to communicate between tasks. Some examples of RTOS are,
- Windows CE, pSOS, VxWorks , ThreadX, MicroC/OS-II, Embedded Linux, Symbian etc.
- Most of the mobile phones are built around the popular RTOS Symbian

EMBEDDED FIRMWARE DEVELPOMENT LANGUAGE

- You can use either a target processor/controller specific language (generally known as Assembly language or low level language).
- A target processor/controller independent language like C, C++, JAVA, etc. Commonly known as high level languages.
- A combination of Assembly and high level language.

ASSEMBLY TO MACHINE LANGUAGE CONVERSION PROCESS



HIGH LEVEL TO MACHINE LANGUAGE CONVERSION PROCESS