NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
(NAAC Accredited)
(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

COURSE MATERIALS

EC 308:EMBEDDED SYSTEMS

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- ♦ Established in: 2002
- ♦ Course offered : B.Tech in Electronics and Communication Engineering

  M.Tech in VLSI
- ♦ Approved by AICTE New Delhi and Accredited by NAAC
- ♦ Affiliated to the University of Dr. A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Providing Universal Communicative Electronics Engineers with corporate and social relevance towards sustainable developments through quality education.

DEPARTMENT MISSION

1) Imparting Quality education by providing excellent teaching, learning environment.

2) Transforming and adopting students in this knowledgeable era, where the electronic gadgets (things) are getting obsolete in short span.

3) To initiate multi-disciplinary activities to students at earliest and apply in their respective fields of interest later.

4) Promoting leading edge Research & Development through collaboration with academia & industry.

PROGRAMME EDUCATIONAL OBJECTIVES

PEOI. To prepare students to excel in postgraduate programmes or to succeed in industry / technical profession through global, rigorous education and prepare the students to practice and innovate recent fields in the specified program/ industry environment.

PEO2. To provide students with a solid foundation in mathematical, Scientific and engineering fundamentals required to solve engineering problems and to have strong practical knowledge required to design and test the system.

PEO3. To train students with good scientific and engineering breadth so as to comprehend, analyze, design, and create novel products and solutions for the real life problems.

PEO4. To provide student with an academic environment aware of excellence, effective communication skills, leadership, multidisciplinary approach, written ethical codes and the life-long learning needed for a successful professional career.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOMES
EC 308

SUBJECT CODE: EC 308	
COURSE OUTCOMES	
C308.1	Ability to understand basics of embedded system and to design an embedded system product.
C308.2	Ability to understand the different standards and protocols used for communication with I/O devices.
C308.3	Ability to distinguish different ways of communication with I/O devices.
C308.4	Ability to understand basic programming concepts of Embedded Systems
C308.5	Ability to understand about inter-process communication.
C308.6	Ability to design real time embedded systems using the concepts of RTOS.

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C308.1			3									1
C308.2	2	3	3		2			2				1
C308.3	2	3	3	3	2	3	2				2	1
C308.4		3	3	3		3						1
C308.5			3		2							1
C308.6			3	3			2					1
C303	2	3	3	3	2	3	2	2			2	1

CO'S	PSO1	PSO2	PSO3
C308.1			
C308.2	3		
C308.3	3	3	2
C308.4	3	3	2
C308.5			
C308.6			

C308	3	3	2

SYLLABUS

COURSE CODE	COURSE NAME	L-T-P-C	YEAR OF INTRODUCTION
EC308	**Embedded Systems**	**3-0-0 -3**	**2016**

Prerequisite: EC206 Computer Organization, EC305 Microprocessors & Microcontrollers

Course objectives:

- To have a thorough understanding of the basic structure and design of an Embedded System
- To study the different ways of communicating with I/O devices and standard I/O interfaces.
- To study the basics of RTOS for Embedded systems.
- To study the programming concepts of Embedded Systems
- To study the architecture of System-on-Chip and some design examples.

Syllabus: Introduction to Embedded Systems, Embedded system design process, Serial and parallel communication standards and devices, Memory devices and device drivers, Programming concepts of embedded programming - Embedded C++ and embedded java, Real Time Operating Systems Micro C/OS-II.

Expected outcome:

The students will be able to:

i. Understand the basics of an embedded system
ii. Develop program for an embedded system.
iii. Design, implement and test an embedded system.

Text Books:

1. David E. Simon, An Embedded Software Primer, Pearson Education Asia, First Indian Reprint 2000.
2. Wayne Wolf, Computers as Components: Principles of Embedded Computing System Design, Morgan Kaufman Publishers - Elsevier 3ed, 2008

References:

1. Frank Vahid and Tony Givargis, Embedded Systems Design – A Unified Hardware / Software Introduction, John Wiley, 2002
2. Iyer - Embedded Real time Systems, 1e, McGraw Hill Education New Delhi, 2003
3. K.V. Shibu, Introduction to Embedded Systems, 2e, McGraw Hill Education India, 2016.
3. Lyla B. Das, Embedded Systems: An Integrated Approach, 1/e , Lyla B. Das, Embedded Systems, 2012
4. Rajkamal, Embedded Systems Architecture, Programming and Design, TMH, 2003
5. Steve Heath, Embedded Systems Design, Newnes – Elsevier 2ed, 2002
6. Tammy Noergaard, Embedded Systems Architecture, A Comprehensive Guide for Engineers and Programmers, Newnes – Elsevier 2ed, 2012

Course Plan

Module	Course content	Hours	End Sem. Exam Marks
I	Introduction to Embedded Systems– Components of embedded system hardware–Software embedded into the system – Embedded Processors - CPU architecture of ARM processor (ARM9) – CPU Bus Organization and Protocol.	4	15
	Design and Development life cycle model - Embedded system design process – Challenges in Embedded system design	3	
II	Serial Communication Standards and Devices - UART, HDLC, SCI and SPI.	3	15
	Serial Bus Protocols - I2C Bus, CAN Bus and USB Bus. Parallel communication standards ISA, PCI and PCI-X Bus.	3	
FIRST INTERNAL EXAM			
III	Memory devices and systems - memory map – DMA - I/O Devices – Interrupts - ISR – Device drivers for handling ISR – Memory Device Drivers – Device Drivers for on-board bus.	6	15
IV	Programming concepts of Embedded programming – Features of Embedded C++ and Embedded Java (basics only). Software Implementation, Testing, Validation and debugging, system-on-chip.	6	15
	Design Examples: Mobile phones, ATM machine, Set top box	1	0
SECOND INTERNAL EXAM			
V	Inter Process Communication and Synchronization -Process, tasks and threads –Shared data– Inter process communication - Signals – Semaphore – Message Queues – Mailboxes – Pipes – Sockets – Remote Procedure Calls (RPCs).	8	20
VI	Real time operating systems - Services- Goals – Structures - Kernel - Process Management – Memory Management – Device Management – File System Organization. Micro C/OS-II RTOS - System Level Functions – Task Service Functions – Memory Allocation Related Functions – Semaphore Related Functions. Study of other popular Real Time Operating Systems.	8	20
END SEMESTER EXAM			

Question Paper Pattern (End semester exam)

Maximum Marks : 100 **Time : 3 hours**

The question paper shall consist of three parts. Part A covers modules I and II, Part B covers modules III and IV, and Part C covers modules V and VI. Each part has three questions uniformly covering the two modules and each question can have maximum four subdivisions. In each part, any two questions are to be answered. Mark patterns are as per the syllabus with 100 % for theory.

QUESTION BANK

MODULE I				
Q:NO:	QUESTIONS	CO	KL	PAGE NO:
1	Define system and embedded system with example	CO1	K2	2
2	Discuss briefly challenges in embedded system design	CO1	K3	22
3	Compare RISC & CISC architecture	CO1	K3	14
4	List out classification of embedded system with example and also define embedded system.	CO1	K3	3
5	Discuss the challenges in embedded system design.	CO1	K2	22
6	Explain various types of embedded system software	CO1	K2	11
7	Discuss about CPU architecture of ARM processor	CO1	K2	16
8	Explain about embedded processors	CO1	K2	7
9	Explain the different embedded system development life cycle models	CO1	K2	21
10	Draw the neat diagram of ARM9 architecture and explain it.	CO1	K2	17
11	Explain the components of embedded system.	CO1	K2	3

MODULE II				
1	Write a note about UART	CO2	K1	25
2	Draw the diagram of **I²C** frame format. Explain each field	CO2	K2	41
3	Explain different data transfer modes used in USB bus standard	CO2	K2	35
4	Describe the various modes of serial communication	CO2	K2	27
5	Draw the HDLC frame and explain it	CO2	K2	28
6	Write a short notes on UART & HDLC	CO2	K2	28
7	Describe bus arbitration. Explain the bus arbitration scheme used in CAN bus with example.	CO2	K2	13
8	Discuss about CAN bus and USB bus	CO2	K2	34
9	Explain parallel communication standards ISA & PCI	CO2	K2	31
10	Briefly describe parallel communication standards.	CO2	K2	30
11	Discuss about serial bus protocols I2C bus and CAN bus	CO2	K2	41
MODULE III				
1	Draw the hierarchy of memory and explain	CO3	K1	50
2	2.Compare static RAM & dynamic RAM	CO3	K3	51
3	Explain about memory system organization	CO3	K2	53
4	Explain the concept of interrupt service routine	CO3	K3	61
5	Explain any four type I/O types used in embedded systems	CO3	K2	63

6	Explain the different modes in which a DMA controller transfers data between memory and peripherals	CO3	K2	54
7	Describe about interrupt driven input and output. Point out the different steps for reading port bytes received from external system.	CO3	K2	55
8	Explain about memory device drivers.	CO3	K3	66
9	Summarize memory management device driver pseudo code with examples	CO3	K2	67
10	Interpret about I/O Devices	CO3	K3	68
	MODULE IV			
1	Explain about various C programming elements	CO4	K3	75
2	Discuss about embedded programming in Java	CO4	K2	78
3	Distinguish macros and functions	CO4	K2	76
4	Describe about use of pipe and table with a diagram	CO4	K2	
5	Compare C programming and embedded C programming	CO4	K2	76
6	Discuss about array and multi-dimensional array with example	CO4	K2	77
7	Discuss the hardware and software components required for designing an ATM machine	CO4	K3	81
8	Discuss about common software tools used for testing and debugging during embedded system development. Explain	CO4	K3	83
9	Explain set top box function and its various	CO4	K2	86

	implementation on software and hardware			
10	Explain mobile phone and its configurations	CO4	K2	91
11	Summarize the features of embedded C++.Explain each one in detail.	CO4	K1	77
12	Describe about debugging and types of software debugging tools	CO4	K2	92
	MODULE V			
1	What is inter process communication?	CO5	K3	98
2	.Compare task and thread	CO5	K2	99
3	What do you mean by shared data	CO5	K3	103
4	write a short notes about semaphore	CO5	K3	104
5	What do you mean by mailbox and pipes	CO5	K2	105
6	Explain about interprocess communication	CO5	K3	102
7	Explain about remote procedure calls	CO5	K3	116
8	Explain about sockets,pipes and semaphore	CO5	K2	106
9	Explain about mailboxes ,queues and sockets	CO5	K2	111
	MODULE VI			
1	Explain about RTOS	CO6	K3	119
2	Explain services and goals of RTOS	CO6	K2	120
3	What do you mean by process management and memory management	CO6	K2	129

4	What is soft and hard real time constraints	CO6	K2	107
5	Explain in detail about RTOS ,structurs ,services and goals	CO6	K3	108
6	Explain in detail about Micro C/OS-II	CO6	K2	135
7	Explain about system level functions and task service functions	CO6	K2	132
9	Explain about memory allocated functions	CO6	K3	133

APPENDIX 1

CONTENT BEYOND THE SYLLABUS

.

.

MODULE 1

EC 308 EMBEDDED SYSTEMS

MODULE 1

Syllabus:

Module 1

- Introduction to Embedded Systems– Components of embedded system hardware– Software embedded into the system – Embedded Processors - CPU architecture of ARM processor (ARM9) – CPU Bus Organization and Protocol. Design and Development life cycle model - Embedded system design process – Challenges in Embedded system design

Module 2

- Serial Communication Standards and Devices - UART, HDLC, SCI and SPI. Serial Bus Protocols - I^{2}C Bus, CAN Bus and USB Bus. Parallel communication standards ISA, PCIand PCI-X Bus.

Module 3

- Memory devices and systems - memory map – DMA - I/O Devices – Interrupts - ISR – Device drivers for handling ISR – Memory Device Drivers – Device Drivers for on-boardbus.

Module 4

- Programming concepts of Embedded programming – Features of Embedded C++ and Embedded Java (basics only). Software Implementation, Testing, Validation and debugging, system-on- chip. Design Examples: Mobile phones, ATM machine, Set top box

Module 5

- Inter Process Communication and Synchronization -Process, tasks and threads – Shared data– Inter process communication - Signals – Semaphore – Message Queues – Mailboxes Pipes –Sockets – Remote Procedure Calls (RPCs).

Module 6

- Real time operating systems - Services- Goals – Structures - Kernel - Process Management – Memory Management – Device Management – File System Organization. Micro C/OS-II RTOS - System Level Functions – Task Service Functions – Memory Allocation Related Functions – Semaphore Related Functions. Study of other popular Real Time Operating Systems

EC 308 EMBEDDED SYSTEMS

Notes – Module 1

Embedded system- definitions

1. “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.”

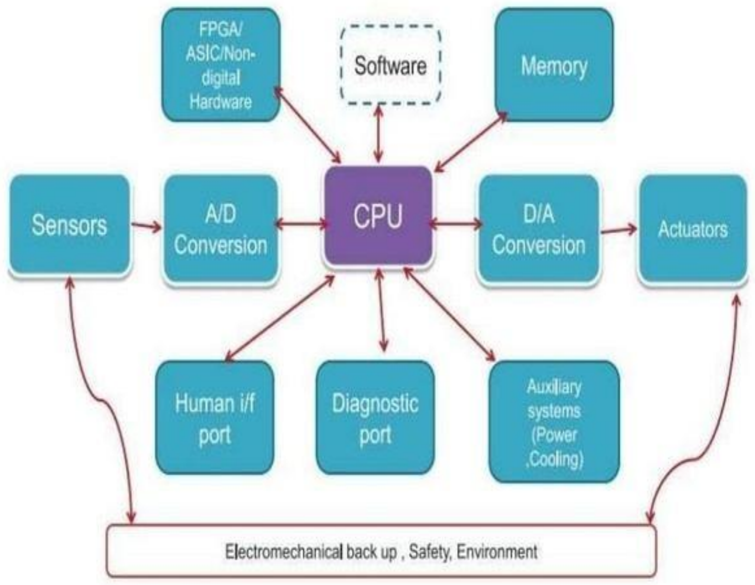
2. “An embedded system is one that has a dedicated purpose software embedded in a computer hardware.”

3. “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.”

4. “It is any device that includes a programmable computer but is not itself intended to be a general purpose computer.”

5. “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.”

BLOCK DIAGRAM OF TYPICAL EMBEDDED SYSTEMS


FPGA/ ASIC/Non-digital Hardware
Software
Memory
Sensors
A/D Conversion
CPU
D/A Conversion
Actuators
Human i/f port
Diagnostic port
Auxiliary systems (Power ,Cooling)
Electromechanical back up , Safety, Environment

Embedded systems are basically designed to regulate a physical variable or the state of some devices by sending some control signals to the Actuators/Devices connected to the output ports of the system in response to the input signals provided by the Sensors/Devices connected tothe input ports.

A typical embedded system contains a single-chip controller which acts as the master brain of the system. The controller can be a Micro-processor or a Micro-controller or a Field Programmable Gate Array device (FPGA) or a Digital Signal Processor (DSP) or an Application Specific Integrated Circuit (ASIC)/ Application specific system processor (ASSP). Key boards, push button switches etc. are examples for user interface input devices. LEDS, Liquid crystal displays (LCDS), Buzzers etc. are examples for common user interface output devices.

It's not necessary that all embedded systems should incorporate these I /O user interfaces.It depends only on the type of the application for which the embedded system is designed. Some embedded systems do not require any manual intervention for its operation. They automatically sense the variations in the input parameters through the sensors connected to the input port of thesystem.

The information from sensors is passed to the processor after signal conditioning & digitalization. Upon receiving the sensor data, the processor of the embedded system performs some pre-defined operations with the help of the firmware program embedded in the system & sends some signals to the actuators connected to the output port of the embedded system.

For an embedded system, it is the responsibility of designer to impart intelligence to the system. So, an embedded system without control algorithm in memory cannot be capable of making any decision depending on the situations as well as changes in the real world.

The Memory of the embedded when is responsible for storing the control algorithm as software program code. In most of embedded systems, this memory is a kind of Read Only Memory (ROM) and it is not available for the user for modifications which means that the memory protected from unwanted user interaction. The common types of memories used in embedded systems FROM, UV-EPROM, EE-PROM & FLASH memory. Depending on the application, the memory may vary from a few bytes to megabytes.

Sometimes the system requires temporary memory while performing arithmetic operations and this type of memory known as "Working Memory". Random Access Memory (RAM) used in most the systems as the working memory. Various types of RAM like SRAM, DRAM and NVRAM are used for this purpose. The size of the RAM also varies from few bytes to kilobytes or megabytes depending on the application.

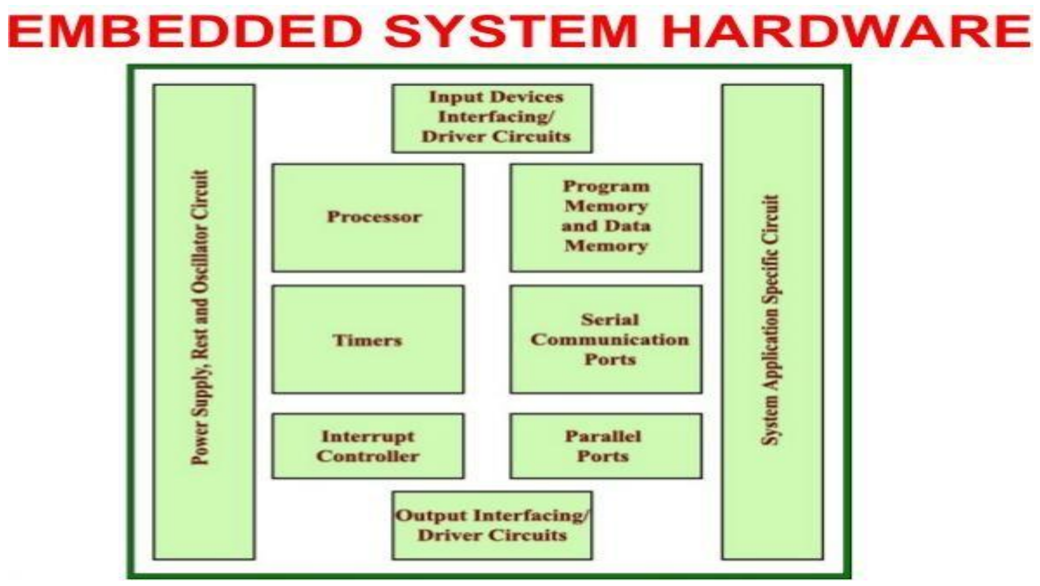
COMPONENTS OF EMBEDDED SYSTEM HARDWARE

An embedded system has three components:

1. **Hardware**.
2. **Application software**. This may perform concurrently the series of tasks or multiple tasks.
3. **Real Time Operating system (RTOS**) that supervises the application software and

providemechanism to let the processor run a process as per scheduling by following a plan to control the latencies. RTOS defines the way the system works. It sets the rules during the execution ofapplication program. A small scale embedded system may not have RTOS.

Hardware.


EMBEDDED SYSTEM HARDWARE
Input Devices Interfacing/ Driver Circuits
Power Supply, Rest and Oscillator Circuit
Processor
Program Memory and Data Memory
Timers
Serial Communication Ports
Interrupt Controller
Parallel Ports
Output Interfacing/ Driver Circuits
System Application Specific Circuit

Processor - A Processor is the heart of the Embedded System. The main criteria for the processor are the processing power needed to perform the tasks within the system. Processors can be of the following categories:

➢ General Purpose Processor (GPP)

➢ Microprocessor

➢ Microcontroller

➢ Embedded Processor

➢ Digital Signal Processor / Media Processor

➢ Application Specific System Processor (ASSP)

➢ Application Specific Instruction Processors (ASIPs)

Power Source

Most embedded systems have a **power supply of their own**. The supply has specific operation range of voltages in one of the following 4 power ranges: **5.0 V + 0.25 V; 3.3 V + 0.3 V; 2.0 V + 0.2 V and 1.5 V + 0.2 V.** Certain systems do not have a power source of their own, so they are connect to external power supply.

For e.g. **A Graphic accelerator** do not have its own power supply.

Clock Oscillator Circuit *(Clocking Units)*

The clock is an another basic unit of a system. A processor needs a clock oscillator circuit as the clock controls the time for executing an instruction. The clock controls the various clocking requirements of the CPU, system clocks and the CPU machine cycles. For processing units, a highly stable oscillator is required as the clock signal provides the synchronizing of all other system units.

System Timers & Real-Time Clocks *(RTC)*

To schedule the various system tasks and for real-time programming, a system clock or an RTC is needed. These clocks drives the timers for various timing & counting needs in a system. System clock & RTC are also used to obtain delays and time-outs. A timer circuit is usually configured as the system-clock.

Another timer circuit is suitably configured as the real-time clock *(RTC)* for periodic saving of time & date in the system. Microcontrollers has built-in internal timer circuits for counting & timing devices

Timer - Embedded systems often require mechanisms for counting the occurrence of events and for

performing tasks at regular intervals.

➢ Embedded processors are often equipped with hardware support for this functionality.

➢ Timer is a device, which counts the input at regular interval using clock pulses at its input.

➢ The count increment on each pulse and store in a register, called count register.

➢ Timer is used for generating delay and for generating waveforms with specific delay.

Serial Port - A serial port is a serial communication interface through which information transfers in

or out one bit at a time.

➢ Serial data transmission is much more common in new communication protocols due to areduction in the I/O pin count, hence a reduction in cost.

➢ Common serial protocols include UART, SPI, SCI and I2C etc.

➢ In most of the embedded systems at least two serial ports are provided.

Parallel Port - A parallel port is a type of interface found on computers or embedded systemsfor connecting peripherals.

The name refers to the way the data is sent; parallel ports send multiple bits of data at once.

Parallel ports require multiple data lines in their cables and port connectors, and tend to be largerthan contemporary serial ports.

Interrupt Controller - An interrupt is a signal to the processor emitted by hardware or softwareindicating an event that needs immediate attention.

Interrupts allow an embedded system to respond to multiple real world events in rapid time.

By managing the interaction with external systems through effective use of interrupts can dramatically improve system efficiency and the use of processing resources.

In an embedded system there are usually multiple interrupt sources.

These interrupt sources share a single pin. The sharing is controlled by a piece of hardwarecalled an interrupt controller that allows individual interrupts to be either enabled or disabled.

System Application Specific Circuit - These are the dedicated circuits for the implementationof the application of particular system. This may varies from one system to other.

Reset Circuit, Power-up Reset & Watchdog-Timer Reset

Reset can be activated by an external reset circuit that activates on power-up *(switching- on)* the system. The reset circuit is a simple circuit *(such as an RC circuit)* whose output connects to the reset pin of the processor. To reset a processor, the reset circuit should activate for a fixed period of a few clock cycles & then deactivate thereby making the processor's reset pin active and then deactivate. Reset can also be activated by any one of the following:

(i) Software instruction *(e.g. RST instruction)*

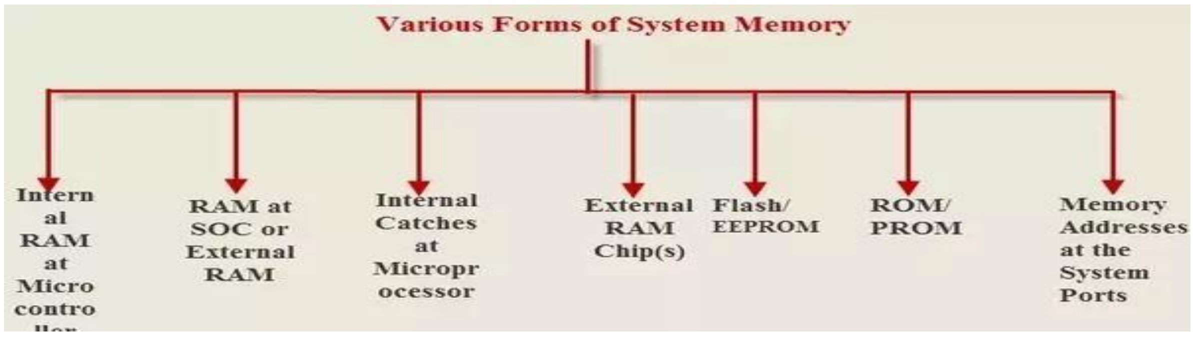
(ii) Reset after a time-out by a programmed timer known as a watchdog timer

The watchdog timer is a timing device that resets the system after a predefined time-out of a few clock cycles. A watchdog timer reset is very essential in embedded systems because it helps in rescuing the system if the system program gets stuck due to a fault. On restart, the system can function normally. Most microcontrollers have on-chip watchdog timers.

Reset means that the processor begins the processing of instructions from a starting address. That address is one that is set by default in a processor on a power-up. From that memory address *(start-up addresses),* program-instructions are fetched following the reset of the processor. In certain processors, there are two start-up addresses. One is for the power-up reset and the processor fetches the program bytes from this address upon power-up. The other one is for after the execution of a "reset‟ instruction or after a timeout such as a watchdog timer based reset. Here, processor fetches the bytes program bytes from this second address on executing the reset instruction or on the watchdog timer based reset.

Memory

An embedded system uses different types of memory modules for a wide range of taskssuch as storage of software code and instructions for hardware


Various Forms of System Memory
Internal RAM at Microcontroller
RAM at SOC or External RAM
Internal Catches at Microprocessor
External RAM Chip(s)
Flash/ EEPROM
ROM/ PROM
Memory Addresses at the System Ports

a. ROM or EPROM or Flash

1. Stores 'Application' program from where the processor fetches the instruction codes
2. Stores codes for system booting, initializing, Initial input data and Strings.
3. Stores Codes for RTOS.
4. Stores Pointers (addresses) of various service routines.

b. Internal, External and Buffer RAM - The function of these memory are

1. Storing the variables during program run,
2. Storing the stacks,
3. Storing input or output buffers for example, for speech or image .

c. EEPROM or Flash - Storing non-volatile results of processing.

d. Caches memory – Functions Assigned to the Caches

1. Storing copies of the instructions, data and branch-transfer instructions in advance fromexternal memories and
2. Storing temporarily the results in write back caches during fast processing.

EMBEDDED PROCESSORS

*1.*General Purpose Processor *(GPP)*

A GPP is a general-purpose processor with instruction set designed not specific to the applications. General purpose processors (GPP) are designed for general purpose computers such as PCs or workstations. The computation speed of a GPP is the main concern and the cost of the GPP is usually much higher than DSPs and microcontrollers. All techniques that can increase CPU speed have been applied to GPPs. For example, GPPs usually include on-chip cache and on-chip DMAs. Commonly used math operations are also supported by the on-chip hardware. GPPs are not designed for fast real-time applications. Scalar structure is common in GPPs but rarely seen in DSPs and microcontrollers.

E.g. Microprocessor; Embedded Processor.

2. Application-Specific Integrated Circuit (ASIC)

An application-specific integrated circuit is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed to run in a digital voice recorder is an ASIC. Modern ASICs often include entire microprocessors, memory blocks including ROM, RAM, EEPROM, flash memory and other large building blocks. Such an ASIC is often termed a SoC (system-on-chip). Designers of digital ASICs often use a hardware description language (HDL), such as Verilog or VHDL, to describe the functionality of ASICs.

3.* Application Specific Instruction-Set Processor *(ASIP)

An application-specific instruction set processor (ASIP) is a component used in system-on-a-chip design. The instruction set of an ASIP is tailored to benefit a specific application. This specialization of the core provides a tradeoff between the flexibility of a general purpose CPU and the performance of an ASIC. An ASIP is a processor with an instruction set designed for specific applications.

E.g. Microcontroller; Embedded microcontroller; Digital Signal Processor (DSP) and media processor; Network processor; IO processor etc.

4. Multi-core processors or multi-processor:

A multi-core processor is an integrated circuit (IC) to which two or more processors have been attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks (*see* parallel processing). A dual core set-up is somewhat comparable to having multiple, separate processors installed in the same computer, but because the two processors are actually plugged into the same socket, the connection between them is faster. In a dual core processor in practice, performance gains are said to be about fifty percent: a dual core processor is likely to be about one-and-a-half times as powerful as a single core processor

The following are important considerations when selecting a processor:

(1) Instruction set

(2) Maximum bits in an operand *(8 or 16 or 32)*

(3) Clock frequency in MHz

(4) Processing speed in MIPS *(Million Instructions Per Second)*

(5) Processor ability to solve complex algorithms

5. MICROPROCESSOR

A microprocessor is a single VLSI chip that has a CPU. It may also have some other units *(e.g., caches, floating point processing arithmetic unit, pipelining units etc.)* that are additionally present for the faster processing of instructions. The CPU is a unit that fetches & processes a set of instructions. The CPU instruction set includes instructions for data transfer operations, ALU operations, stack operations, IO operations, program control & sequencing operations. The important microprocessors used in the embedded systems are ARM, 68HCxxx, 80x86 and SPARC family of microprocessors. *A microprocessor is used as general-purpose processor when large embedded software has to be located in the external memory chips*.

6. MICROCONTROLLER *(MICRO-COMPUTER)*

A microcontroller is a single-chip VLSI unit having limited computational capabilities, possesses enhanced IO and a number of on-chip functional units. A microcontroller has processor, memory & several other hardware units *(peripherals)* in it; these form the microcomputer part of the embedded system. Choosing a microcontroller as a processing unit depends upon the application-specific features in it. Microcontrollers are particularly suited for use in embedded systems for real-time control applications with on-chip memory & peripherals/devices. A microcontroller is used when a small embedded software has to be locatedin the internal memory and when on-chip functional units such as various types of peripherals such as ports, timer, ADC, PWM etc. are required.

7. SINGLE PURPOSE PROCESSORS

(1) **Coprocessor** *(for e.g., math coprocessor, floating point processor etc.)*

(2) Graphics processor: - An image consists of a no. of pixels.

- A separate graphics processor is required for applications such as gaming**,** display fromgraphics memory and rotate an image or its segments.

(3) Pixel coprocessor: - A pixel coprocessor is required in digital cameras for operation onimages such as rotate right, rotate-left, rotate- up, rotate-down, shift to next, shift to previous.

(4) Encryption engine: - A suitable algorithm runs in this processor to encrypt data for securetransmission.

(5) Decryption engine: - A suitable algorithm runs in this processor to decrypt the encrypteddata at receiver's end.

(6) A discrete cosine transformation *(DCT)* and inverse transformation *(DCIT)* processor isrequired in speech and video processing.

(7) Protocol stack processor: - A protocol stack required before an application data is sent to anetwork.

- At the receiver's end, the protocol stack is received and application data is accepted accordingly. **(E.g. for Wi-Fi protocol)**

- MP3 decompression is done before retrieving or playing files.

(8) Network processor:

- A network processor's functions are to establish a connection, finish, send & receive acknowledgements, send and receive retransmission requests and check and correct received dataframe errors.

- The network processor's functions include all protocol related processing functions.

(9)Accelerator

- For e.g., **Java codes accelerator** is a coprocessor that accelerates computations by takingactions that are in Java programs.

(10) Controller

- E.g., DMA Controller *(Direct Memory Access),* bus control, peripheral management

(11)* CODEC *(Coder & Decoder)

- A CODEC is a processor circuit encodes input bits or decodes the encoded information into a complete set of bits or original signals. The CODEC functions as a compression & decompression unit signals such voice, speech, image or video signals

JPEG CODEC:- This is a processor for jpg compression and decompression.

MPEG CODEC:- MPEG3 CODEC is a processor for mp3 compression of audio/video data streams for storing or transmitting

EMBEDDED SOFTWARE IN A SYSTEM

The software program code (instruction codes) is the brain of an embedded system. An embedded system processor executes software that is specific to a given application. The program codes are placed in the ROM (or flash memory or PROM) for the execution of tasks when the system runs. This final "machine implementable software" is called the "ROM image" that is being embedded into the ROM similar to an "image" in an "image frame". Each program code is in bytes format & these bytes are saved at each address of the system memory (ROM). The code bytes are required at each ROM address to execute the tasks.

So, a machine implementable software file (ROM image) is similar to a table having many rows and only two columns; 1st column for memory address & 2nd column for corresponding code byte in a memory address. By changing ROM image, the same hardware platform will work differently.

1) MACHINE-CODE BASED CODING

In machine-code based coding, the programmer defines the machine code bytescorresponding each memory addresses for a program.

- Machine-code based coding is done only in specific situations because it is time consuming and the programmer must have to understand the processor instructions set and their corresponding machine codes

2) CODING IN ASSEMBLY LANGUAGE

Small programs can be coded in assembly language after understanding the processor & its instruction set. "Assembler" is the software used for converting the codes written in assemblylanguage (similar to a compiler for high level language like C, Java etc) . Assembly language coding is extremely useful for configuring devices like ports, ADC, DAC etc. But, Assembly language based programming is also very time consuming while making larger programs/ codes. Full coding in assembly may be done only for a few simple, small-scale embedded systems.

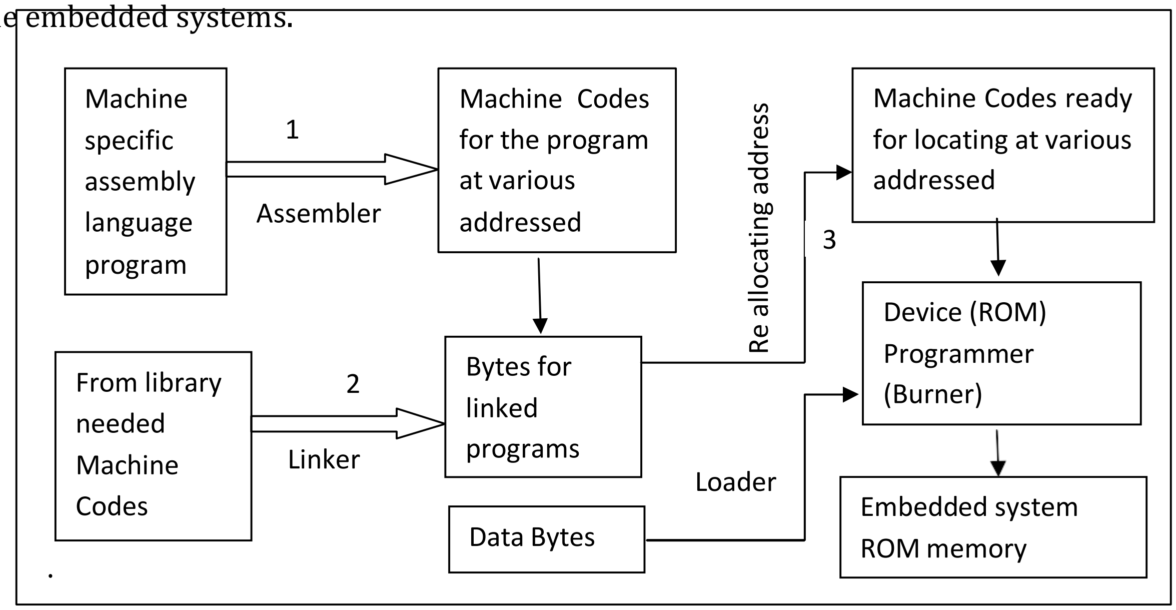

Machine specific assembly language program
1
Assembler
Machine Codes for the program at various addressed
Re allocating address
3
Machine Codes ready for locating at various addressed
From library needed Machine Codes
2
Linker
Bytes for linked programs
Device (ROM) Programmer (Burner)
Loader
Data Bytes
Embedded system ROM memory

Figure: shows the process of converting an assembly language program into machine implementable software file and then finally obtaining a ROM image file.

Assembler, Linker, Locator & Loader are the software required for the whole process.

(a) First step is called "Assembling" in which assembler software translates the assemblysoftware into the machine codes.

(b) Next step is called linking; a linker links these codes (if necessary) with the other codes takenfrom the library.

(c) For a final program, a no. of other codes are to be linked together.

For eg., there are the standard codes for delay function (eg. delay( ) in Arduino). If 'delay( )' is included in the program, the program codes for the delay( ) must link with the final assembled code. The linked file in binary is known as executable file (a file with '.EXE' extension).

(d) In embedded systems, the next step after linking is the use of a "Locator" software which locates the already fixed ROM addresses. For eg., in a memory-mapped IO scheme; IO port addresses, IO devices addresses etc. are permanently assigned to some memory locations.

- The locator software "re-allocates" the memory addresses in a linked file & creates a file with permanent memory allocation for each of the code bytes in a standard format

- Eg for such a standard file format is "Intel .HEX file format".

(e) In the next step, the "Loader" software performs the task of placing/ loading the code bytes as an "image to be placed in ROM" by finding the exact available ROM memory addresses

- For many processors, the available memory addresses may not start from "0000H".

- The "loader" finds the appropriate "start address" for the final program codes.

(f) Lastly "Programmer Device/ Equipment" takes as input the ROM image file.

For eg. (in .HEX format) & "writes" the image as byte by byte into the memory. So the process of placing the codes into ROM or flash memory is also called "Burning" into the ROM/flash.

3) CODING IN HIGH LEVEL LANGUAGE

For large software programs development, high-level language like C, C++, visual C++, Java etc. are used. 'C‟ is usually the preferred language. The programmer needs to understand only the hardware organization of the whole embedded system when coding in high level language

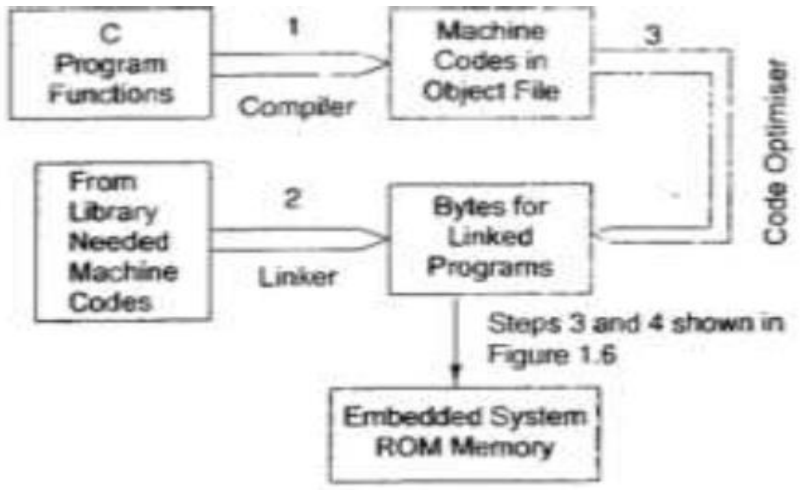

C Program Functions
1
Compiler
Machine Codes in Object File
3
Code Optimiser
From Library Needed Machine Codes
2
Linker
Bytes for Linked Programs
Steps 3 and 4 shown in Figure 1.6
Embedded System ROM Memory

Figure shows the process of converting a C program into the ROM image file.

-First, the compiler software generates the object codes (file with .OBJ extension). The compilerassembles the codes according to the processor instruction set

- Before linking, the compiler may use a “Code-Optimizer” that optimizes the codes by removing the “redundant/ unnecessary variables or steps” written in the program

- The linker links the object codes with other standard program codes in the program (similar to the linking step in assembling process). For eg, the linker includes the program codes for the pre-defined functions like printf(), delay() etc

- Codes for some standard devices & the device control management also link at this

stage;For eg., a printer device management & its driver codes

- After linking, the other steps for creating a file for ROM image are same as that discussed inprevious section of assembling process (Locating, Loading & Burning).

4) DEVICE DRIVERS

- A device driver is a software written in a high level language that controls functions/actions ofa device such as opening the device, “connecting the device to other devices”, “reading the device”, “writing to the device” & “closing the device”

- Each of the device driver software can access a parallel port, serial port, keyboard, mice, disk, network, display, file etc. at specific addresses.

- A device driver software controls 3 functions,

(i) Initializing a device by placing appropriate bits at the control register of the device.

(ii) Calling an Interrupt Service Routine upon setting up a status flag in the status register of thedevice for running the ISR.

(iii) Re-setting the status flag after an interrupt service is completed Device Manager Software

- A device manager software provide programs for

• Detecting the presence of devices connected to the system (both virtual & physical)

• Listing out the available devices connected to the system

• Initializing the devices testing the devices

• Allocates “Port Addresses” for the various physical & virtual devices (Eg. COM2, COM3 etcfor virtual serial ports in Windows)

ARM PROCESSOR

Advanced RISC machine (ARM) is the first reduced instruction set computer (RISC) processor for commercial use, which is currently being developed by ARM Holdings.

RISC:- A Reduced Instruction Set Computer (RISC) is a microprocessor that has been designedto perform a small set of instructions, with the aim of increasing the overall speed of the processor. The RISC concept first originated in the early 1970's when an IBM research team proved that 20% of instruction did 80% of the work. The RISC architecture follows the philosophy that one instruction should be performed every clock cycle.

Comparison between CISC and RISC

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.
Mainly used in normal pc‘s, workstations & servers.	Mainly used for real time applications.

ARM9

- The ARM9TDMI is a member of the ARM family of general-purpose microprocessors.
- The ARM9TDMI is targeted at embedded control applications where high performance, low die size and low power are all important.
- The ARM9TDMI supports both the 32-bit ARM and 16-bit Thumb instruction sets, allowing the user to trade off between high performance and high code density.
- ARM and Thumb are two different instruction sets supported by ARM cores with a “T” in their name. Thumb mode allows for code to be smaller, and can potentially be faster if the target has slow memory
- The ARM9TDMI supports the ARM debug architecture and includes logic to assist in both hardware and software debug.
- The ARM9TDMI supports both bidirectional and unidirectional connection to external memory systems.
- The ARM9TDMI also includes support for coprocessors.

- The ARM9TDMI processor core is implemented using a five-stage pipeline consisting of fetch, decode, execute, memory and write stages.
- The device has a Harvard architecture, and the simple bus interface eases connection to either a cached or SRAM-based memory system.

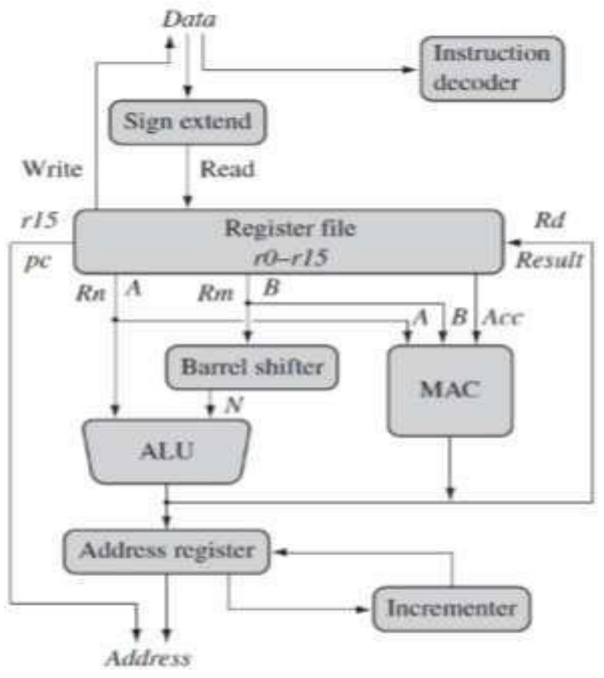
Advanced RISC Machine (ARM) Processor Fundamentals

Like all RISC processors, the ARM processor also uses an architecture known as the "Load-Store Architecture" which means ARM has only two instructions for transferring data in & outof the processor & its memory.

The two instructions are

(i) 'LOAD' Instruction:- Copy data from Memory to CPU registers in ARM core

(ii) 'STORE' Instruction :- Copy data from CPU registers of ARM core to Memory.

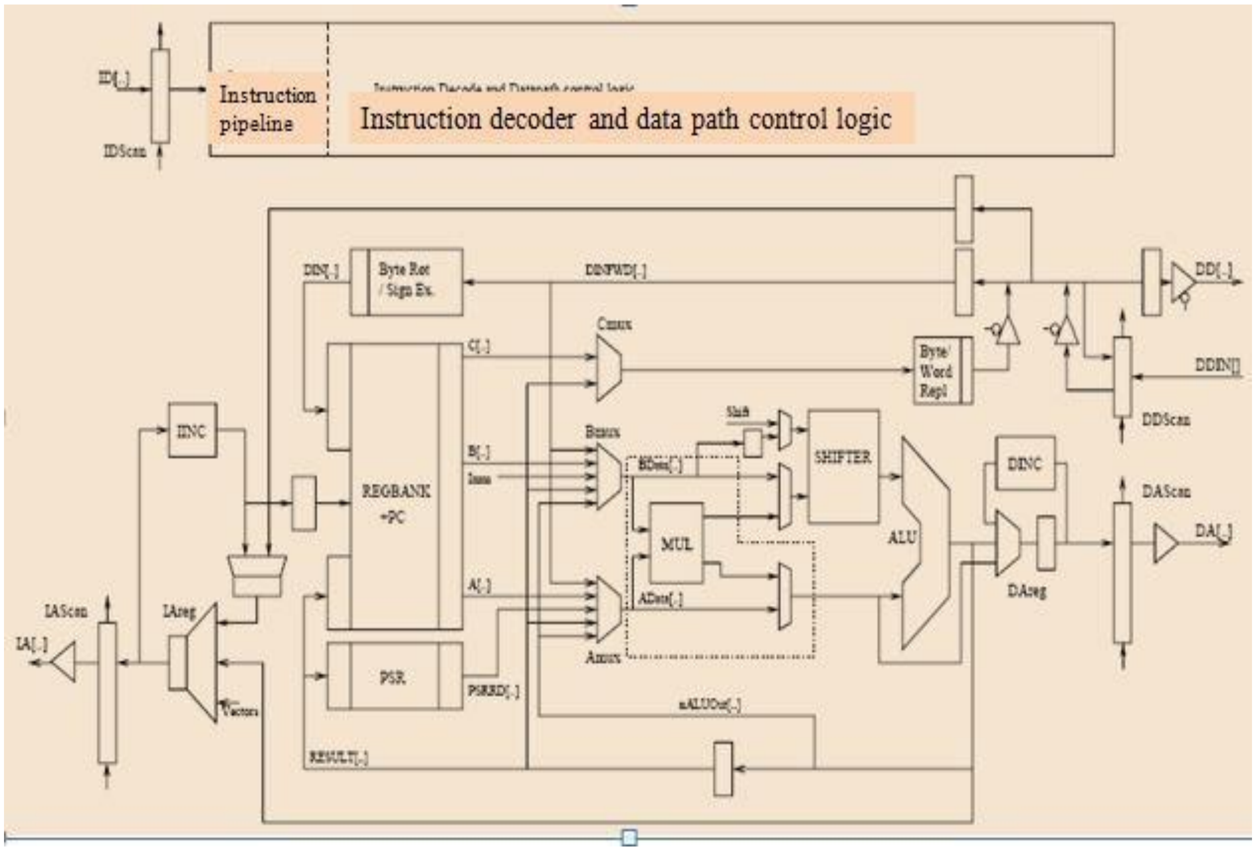


- In ARM, there are no other instructions that directly manipulate data in the memory.
- Fig. shows a Von-Neumann implementation of ARM core, in which the instructions & the data share the same bus (A Harvard implementation of ARM core has two separate buses of instructions & data)
- So, Data/ Instructions enters the processor core through the 'Data Bus' . The 'Instruction Decoder' translates instructions before they are executed.
- Data items are placed in the 'Register File', which is a CPU register bank made up of 16 general purpose registers (r0 to r15) in which each register has a size of 32-bit.
- The ARM core is a 32-bit processor in which the general purpose registers (r0 - r15) of 'Register_File' can hold only 32-bit data values. The 'Sign Extend Hardware'

converts the signed 8 or 16-bit data inputs to 32-bit data values while they read from memory

- ARM instructions have two Source Registers : ‘Rn’ & ‘Rm’.One Destination Register :‘Rd’(also called as Result Register).
- The source operands are read from the register file using internal buses ‘A_bus’ & ‘B_bus’ respectively.
- The ALU or MAC(Multiply accumulator) unit of ARM takes the source data in the registers ‘Rn’ & ‘Rm’ through A bus & B bus for computing the result.
- After passing through the functional units (ALU or MAC), the result in ‘Rd’ register is written back to the register file through the ‘Result Bus’.
- Here, the data in one of the source register ‘Rm’ can also be shifted in a ‘Barrel_Shifter’ unit before it enters the ALU . Using the combined operation of the Barrel Shifter & ALU, the ARM can generate a wide range of expressions & addresses.
- In ARM, the ALU also generates the memory addresses that is to be placed in the addressregister for broadcasting the address through the ‘Address Bus’.
- The ‘Incrementer’ is used to update the address register for a data transfer operation from sequential memory locations.

ARM 9 ARCHITECTURE



The main parts of the ARM processor are:

1. Register file: The processor has a total of 37 registers made up of 31 general 32 bit registersand 6 status registers
2. Booth Multiplier
3. Barrel shifter
4. Arithmetic Logic Unit (ALU)
5. Control Unit.

Registers

The processor has a total of 37 registers made up of 31 general 32 bit registers and 6 status registers. At any one time 16 general registers (R0 to R15) and one or two status registers are visible to the programmer. The visible registers depend on the processor mode. In all modes 16 registers, R0 to R15, are directly accessible. All registers except R15 are general purpose andmay be used to hold data or address values. Register R15 holds the Program Counter (PC). A seventeenth register (the CPSR - Current Program Status Register) is also accessible. It contains condition code flags and the current mode bits and may be thought of as an extension to the PC. R14 is used as the subroutine link register and receives a copy of R15 when a Branch and Link instruction is executed.

Program status register:- It contains the processor flags (Z, S, V and C). The modes bits alsoexist in the program status register in addition to the interrupt and fast interrupt disable bits

Some special registers: Some registers are used like the instruction register, memory data readand write register and memory address register

Multiplexers: Many multiplexers are used to control the operation of the processor buses

Arithmetic Logic Unit (ALU) :- The ALU has two 32-bits inputs. The first comes from the register file while the other comes from the shifter. ALU outputs modify the status register flags.

Booth multiplier:- The multiplier has three 32-bit inputs. All the inputs come from the register file. The multiplier output is only the 32 least significant bits of the product.

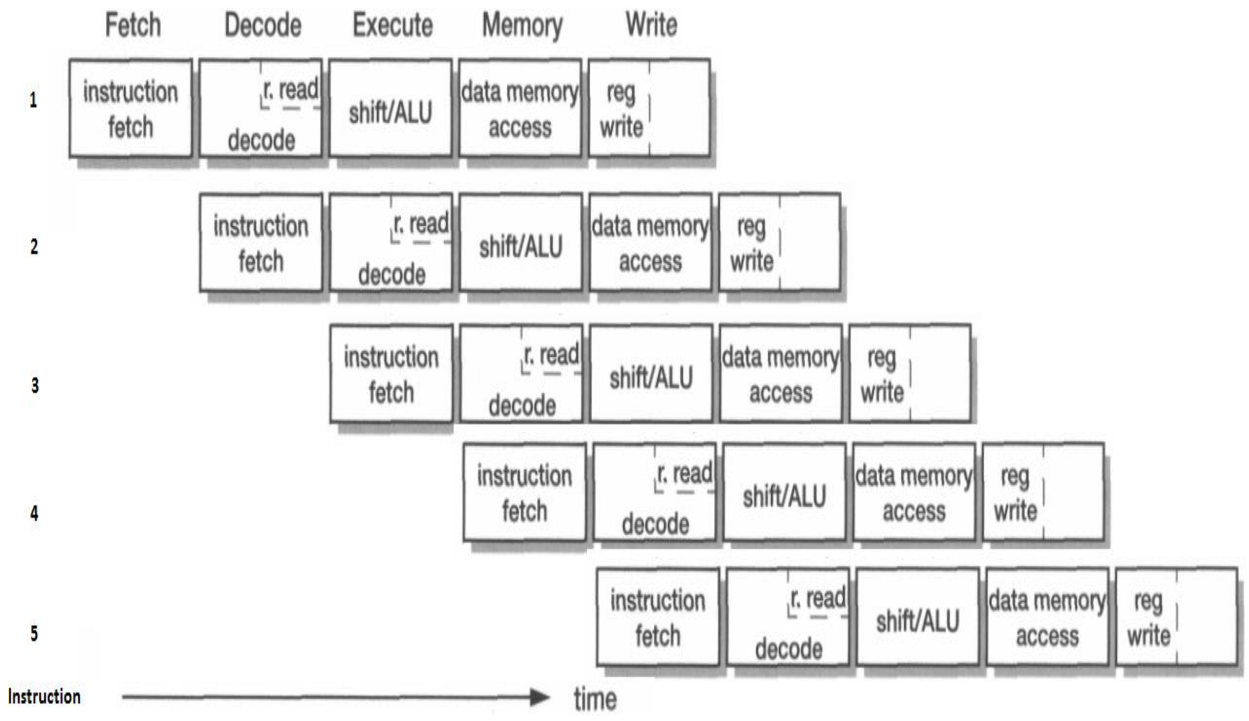
Barrel shifter:- A **barrel shifter** is a digital circuit that can **shift** a data word by a specified number of bits without the use of any sequential logic, only pure combinational logic. The barrel shifter has a 32-bit input to be shifted. This input is coming from the register file or it could be immediate data. The shifter has other control inputs coming from instruction register. Shift field in the instruction controls the operation of the barrel shifter. This field indicates the type of shift to be performed (logical left or right, arithmetic right or rotate right). The amount by which the register should be shifted is contained in an immediate field in the instruction or it could be the lower 6 bits of a register in the register file.

Control Unit :- For any microprocessor, control unit is the heart of the system. It is responsiblefor the system operation and so the control unit design is the most important part in the hole design. Control unit is usually a pure combinational circuit. The processor timing is also included in the control unit.

Pipeline implementation

The ARM9TDMI implementation uses a five-stage pipeline design. These five stages are:

1. Instruction fetch (F):- The instruction is fetched from memory and placed in the instruction pipeline
2. Instruction decode (D):- The instruction is decoded and register operands read from the register files. There are 3 operand read ports in the register file so most ARM instructions can source all their operands in one cycle
3. Execute (E):- An operand is shifted and the ALU result generated. If the instruction is a loador store, the memory address is computed in the ALU
4. Data memory access (M):- Data memory is accessed if required. Otherwise the ALU resultis simply buffered for one cycle
5. Register write (W):- The result generated by the instruction are written back to the register file, including any data loaded from memory



ARM Bus Organization -AMBA

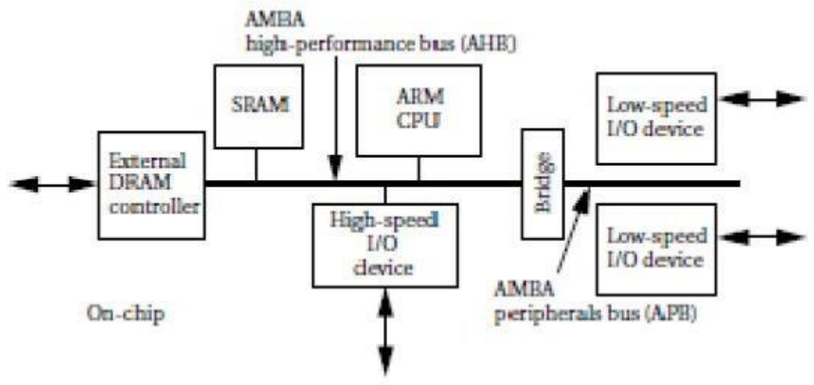


Figure: ARM Bus structure

ARM has created a separate bus specification for single-chip systems. **The AMBA[Advanced Microcontroller Bus Architecture] bus** [ARM99A] supports CPUs, memories, and peripherals integrated in a system-on-silicon.

The AMBA specification includes two buses.

The AMBA high-performance bus (AHB) is optimized for high-speed transfers and is directly connected to the CPU. It supports several high performance features: pipelining, burst transfers.

Burst mode is a temporary high-speed data transmission mode used to facilitate sequential data transfer at maximum throughput. Burst mode data transfer rate (DTR) speeds canbe approximately two to five times faster than normal transmission protocols.,

Split transactions:- the Split and Retry responses are used by slaves which require a large number of cycles to complete a transfer. These responses allow a data phase transfer to appear completed to avoid sharing the bus, but at the same time indicate that the transfer should be re- attempted when the master is next granted the bus.

The difference between them is that a SPLIT response tells the Arbiter to give priority to all other masters until the SPLIT transfer can be completed (effectively ignoring all further requests from this master until the SPLIT slave indicates it can complete the SPLIT transfer), whereas the RETRY response only tells the Arbiter to give priority to higher priority masters.

A SPLIT response is more complicated to implement than a RETRY, but has the advantage that it allows the maximum efficiency to be made of the bus bandwidth.

Multiple bus masters.

The AMBA bus specification supports multiple bus masters on the high performance ASB. Asimple two wire request and grant mechanism is implemented between the arbiter and each bus master. The arbiter ensures that only one bus master is active on the bus and also ensures that when no masters are requesting the bus, a default master is granted.

The specification also supports a shared lock signal. This allows bus masters to indicate that the current transfer is indivisible from the following transfer and prevents other bus masters fromgaining access to the bus until the locked transfers have completed. A bridge can be used to connect the AHB[AMBA High-performance Bus] that is a single clock-edge protocol, to an AMBA peripherals bus(APB). This bus is designed to be simple and easy to implement; it also consumes relatively little power. The AHB assumes that all peripherals act as slaves, simplifying the logic required in both the peripherals and the bus controller. It also does not perform pipelined operations, which simplifies the bus logic.

Embedded Product Development Life Cycle (EDLC)

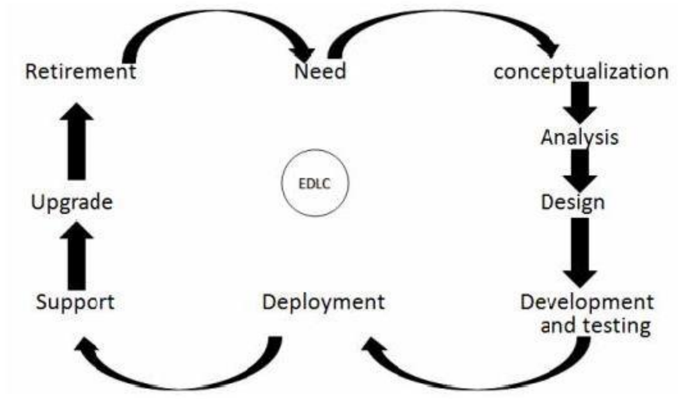


Figure : Phases of EDLC

EDLC is Embedded Product Development Life Cycle:- It is an Analysis, Design, Implementation based problem solving approach for embedded systems development.

Different Phases of EDLC: The following figure depicts the different phases in EDLC:

Need :- The need may come from an individual or from the public or from a company.

Conceptualization :- Defines the scope of concept, performs cost benefit analysis and feasibility study and prepare project management and risk management plans.

Analysis:- The product is defined in detail with respect to the inputs, processes, outputs, and interfaces at a functional level.

Design :- The design phase identifies application environment and creates an overall architecturefor the product.

Development and Testing :- Development phase transforms the design into a realizable product.

Deployment :- Deployment is the process of launching the first fully functional model of the product in the market.

Support :- The support phase deals with the operational and maintenance of the product in the production environment.

Upgrades :- Deals with the development of upgrades (new versions) for the product which is already present in the market.

Retirement/Disposal :- The retirement/disposal of the product is a gradual process. This phase is the final phase in a product development life cycle where the product is declared as discontinued from the market.

Waterfall Model

· Linear or waterfall model is one of the EDLC models which adopted in most of the olden systems.

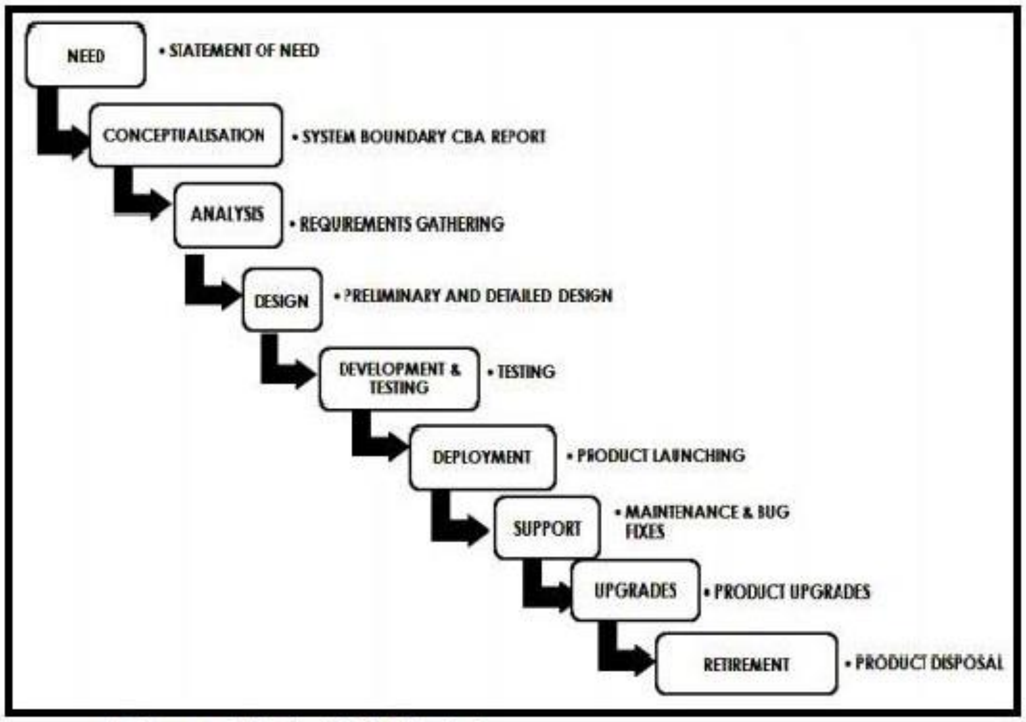


Figure: Waterfall Model

In this approach each phase of EDLC (Embedded Development Product Lifecycle) is **executed in sequence.** It establishes analysis and design with **highly structured development phases**. The execution flow is **unidirectional.** The **output of one phase serves as the input of the next phase.** All activities involved in each phase are well planned so that what should be done in the next phase and how it can be done. The feedback of each phase is available only after they are executed. It implements extensive review systems to ensure the process flow is going in the right direction. One significant feature of this model is that even **if you identify bugs in the current design the development process proceeds with the design.** The fixes for the bug are postponed till the support phase.

Advantages

Product development is rich in terms of: · Documentation, Easy project management, Goodcontrol over cost & Schedule

Drawbacks

· It assumes all the analysis can be done without doing any design or implementation

· The risk analysis is performed only once.

· The working product is available only at the end of the development phase

· Bug fixes and correction are performed only at the maintenance / support phase of the lifecycle.

Challenges in Embedded Systems

1. Amount and type of hardware needed. Optimizing various hardware elements for a particulardesign.

2. Taking into account the design metrics

Design metrics examples –power dissipation, physical size, number of gates and engineering, prototype development and manufacturing costs.

3. Optimizing the Power Dissipation:- Clock Rate Reduction and Operating Voltage Reduction

4. Disable use of certain structural units of the processor to reduce power dissipation the processor to reduce power dissipation. Control of power requirement, for example, by screen auto-brightness control

5. Process Deadlines:- Meeting the deadline of all processes in the system while keeping the memory, power dissipation, processor clock rate and cost at minimum is a challenge.

6. Flexibility and Upgradeability:- Ability to offer the different versions of a product for marketing and offering the product in advanced versions later on.

7. Reliability:- Designing reliable product by appropriate design and thorough testing, verification and validation is a challenge.

8. Testing, Verification and Validation:- Testing – to find errors and to validate that the implemented software is as per the specifications and requirements to get reliable product.

MODULE 2

Subject : EC 308 EMBEDDED SYSTEMS

Module 2

Communication Protocol

A protocol is a standard adopted, which tells the way in which the bits of a frame must be sent from a device (or controller or port or processor) to another device or system

UART (Universal Asynchronous Receiver Transmitter)

UART is a simple half-duplex, asynchronous, serial protocol. It makes a simple communication between two equivalent nodes and any node can initiate communication. Since connection is half-duplex, the two lanes of communication are completely independent. Its parameters (format, speed ...) are configurable so it is called universal. It doesn't have a clock, so it is called asynchronous.

The speed of communication (measured in bauds) is predetermined on both ends. A general rule of thumb is to use 9600 bauds for wired communication. Maximum speed is 115200. The UART implements error-detection in the form of parity bit. Parity bit is HIGH when number of 1's in the Data is odd and it is LOW when number of 1's in the Data is even.

Handshaking Signals' in UART based serial communication

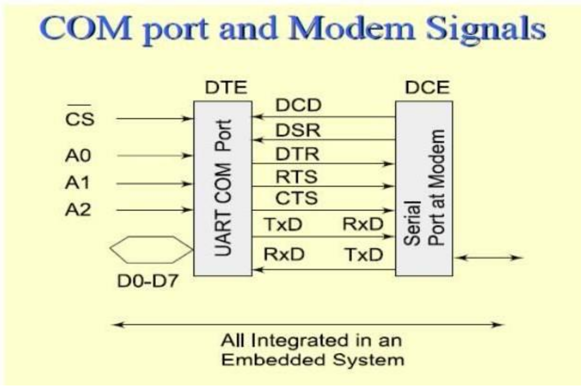


Figure : The handshaking signals of UART.

The following are the names of the five handshaking signals defined for the UART based communication between a 'Computer Terminal devices' to an external 'Modem' device

(1) DCD (Data Carrier Detect)

(2) DSR (Data Set Ready)

(3) DTR (Data Terminal Ready)

(4) RTS (Request to Send)

(5) CTS (Clear to Send)

- Two handshaking signals (DCD & DSR) are generated from the modem side
- Three handshaking signals (DTR, RTS & CTS) are generated by the terminal.

-Here, 'TxD' & 'RxD' are the data transmitting & data receiving signal lines/ pins of the 'computer terminal device.

DCD (Data Carrier Detect) :-This handshaking signal will be raised from a 'modem' towards the 'computer terminal device' to which that modem is connected. The modem device asserts a DCD signal to inform the 'terminal device' that a 'carrier signal' has been detected & the contact between this modem to an another modem has been established successfully

DSR (Data Set Ready) :- This is another handshaking signal will be raised from a 'modem' towards the 'terminal device'. When the 'modem' is turned-on after a power off (or sleep) & if it is ready to communicate, it assert DSR signal to indicate the terminal

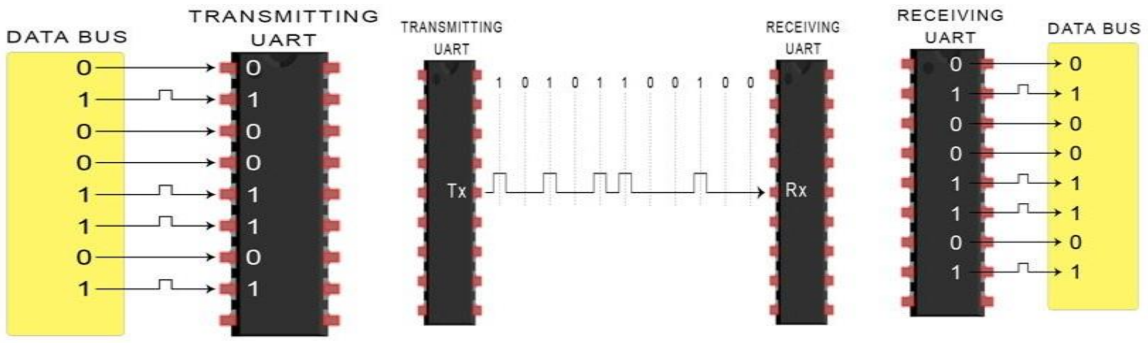
DTR (Data Terminal Ready) :- This handshaking signal will be raised from the 'terminal device' towards a 'modem' connected to it. When 'terminal' is 'turned-on', it sends DTR signal to indicate the' modem' that it is 'ready for a communication'

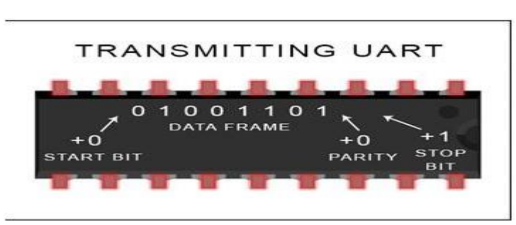
RTS (Request to Send) :- This handshaking signal will be raised by the 'terminal device'. When the terminal device has a 'Data Byte' to transmit, it assert RTS signal to indicate the modem that it has a byte of data to transmit

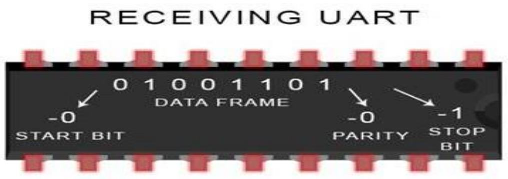
CTS (Clear to Send) :- This handshaking signal will also be raised from the terminal device's side. When the 'terminal' has room (or storage space) for storing a data that is going receive, it sends out CTS signal to modem to indicate that it can receive the data byte now

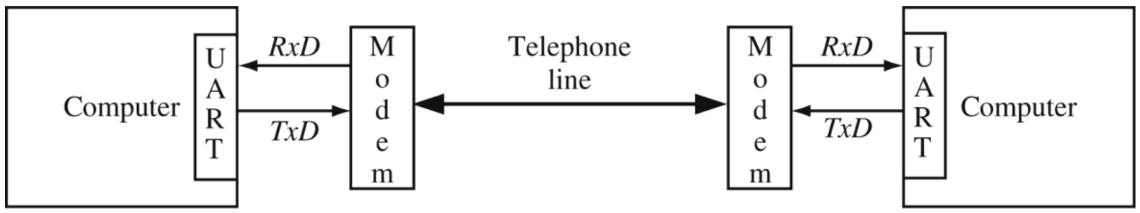
Serial Data Transmission in UART

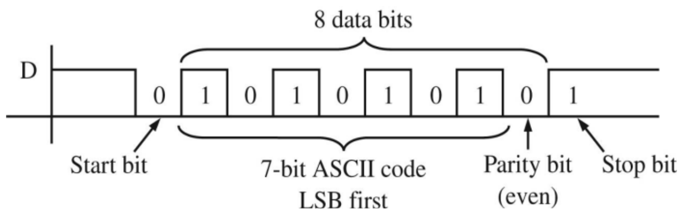
The serial transmission in UART is shown in the figure. The start and stop bits are identified based on the signal state. In between that the data transmission is obtained.











UART mode is as follows

UART having different mode of states that are

1. **Idle State**:- A line non-return to zero (NRZ) state. It means in idle state the logic state is 1 at the serial line.
2. **Byte start signalling flag bit:-** Start bit 1 to 0 transition, which receiver detect at the middle of bit interval T
3. **Data bits :-** After start bit; 8 bits transmitted on TxD line and received on RxD line during period of 8 T (receiver detect at the middle of each bit interval T), In earlier circuits, the number of data bits could also be set 5, 6 or 7 in place of 8
4. **Control or error detect bit:-** One bit- P-bit is optional. P bit can be used to detect parity error. P-bit can be used to interpret the preceding byte not as data but as address or command or parity as per the processing circuit for serial bits at receiver
5. **Byte end flag bit :-** Minimum one stop bit at Logic 1 [In earlier circuits, the number of stop bits could also be set 1½ or 2 in place of 1]
6. **Disconnected State :-** Zero (Z) state. Disconnected serial line logic state is 0

UART Frame structure

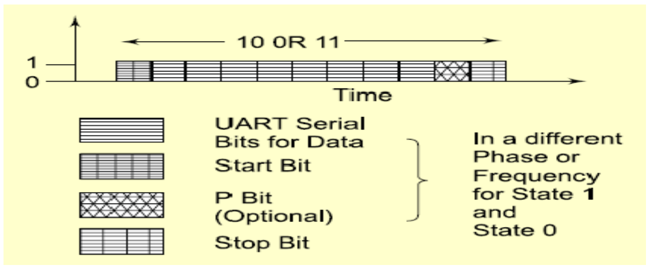

10 0R 11
1
0
Time
UART Serial Bits for Data
Start Bit
P Bit (Optional)
Stop Bit
In a different Phase or Frequency for State 1 and State 0

Fig : UART Frame structure

The frame structure of UART is shown below. A start bit at the initial position indicates that a communication is starting. Then the Data bits are follows. After that a parity bit for error checking and at last a stop bit to indicate the end of transmission.

Advantages of UART

- Only uses two wires
- No clock signal is necessary
- Has a parity bit to allow for error checking

Disadvantages of UART

- The size of the data frame is limited to a maximum of 9 bits
- Doesn’t support multiple slave or multiple master systems
- The baud rates of each UART must be within 10% of each other

HDLC (High-level Data Link Control)

HDLC is a standard protocol for the data link network. For synchronous communication between two data link layers on a network. There are two formats Standard HDLC and extended HDLC for **2^8** and **2^{16}** destination devices or systems, respectively .

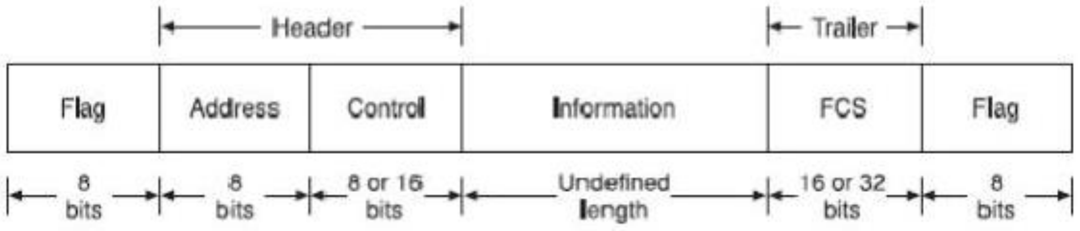


Fig : Sequence of bits in a HDLC frame

The functions of the bits in the HDLC frame structure are

1. **Flag**: - frame start with a 8 bits flag bits which indicate the starting of the frame. The Flag bits at start are "01111110".

2. **Address** - 8 bits in Standard HDLC format and 16 bits in extended format

3. **Control field** :- The control field distinguishes between the three different types of frames used in HDLC, namely information, control and unnumbered frames. The first one or two bits of the field determine the type of frame. The field also contains control information which is used for flow control and link management.

4. **Information field** :- The information field does not have a length specified by HDLC. In practice, it normally has a maximum length determined by a particular implementation. Information frames (also known as I-frames) are the only frames that carry information bits which are normally in the form of a fixed-length block of data of several kilobytes in length. All other types of frames (control and unnumbered frames) normally have an empty information field.

5. **Frame check sequence:-** The FCS field contains error-checking bits, normally 16 bit with a provision for increasing this to 32. 8 bits in Standard HDLC format and 16 bits in extended format

HDLC Frame types

□ The different types of frame are distinguished by the contents of the control field.
□ The structure of all three types of control field is shown in Figure.

Frame Type	1	2	3	4	5	6	7	8	bits
Information	0	N(S)			P	N(R)			
Supervisory	1	0	F		P	N(R)			
Unnumbered	1	1	F		P	F			

Where N(S) = Sequence number, N(R)= Receive sequence number, F=Function bits
P=poll final bit used for polling in normal response mode

Information frames

□ An I-frame is distinguished by the first bit of the control field being a binary 0.

□ The control field of an I-frame contains both a send sequence number, N(S), and a receive sequence number, N(R), which are used to facilitate flow control.

□ N(S) is the sequence number of frames sent and N(R) the sequence number of frames successfully received by the sending node prior to the present frame being sent.

□ Thus the first frame transmitted in a data transfer has send and receive sequence numbers 0,0.

□ I-frames also contain a poll/final (P/F) bit (as do other frames). This acts as a poll bit when used by a primary station and a final bit by a secondary.

□ A poll bit is set when a primary is transmitting to a secondary and requires a frame or frames to be returned in response, and the final bit is set in the final frame of a response.

Supervisory frames

□ Supervisory frames are distinguished by the first 2 bits of the control field being 1 0.

□ These frames are used as acknowledgements for flow and error control.

□ Supervisory frames contain only a receive sequence number since they relate to the acknowledgement of I-frames and not to their transmission.

□ They also contain two function bits which allow for four functions as shown in Table 2.1 which lists the supervisory commands/responses.

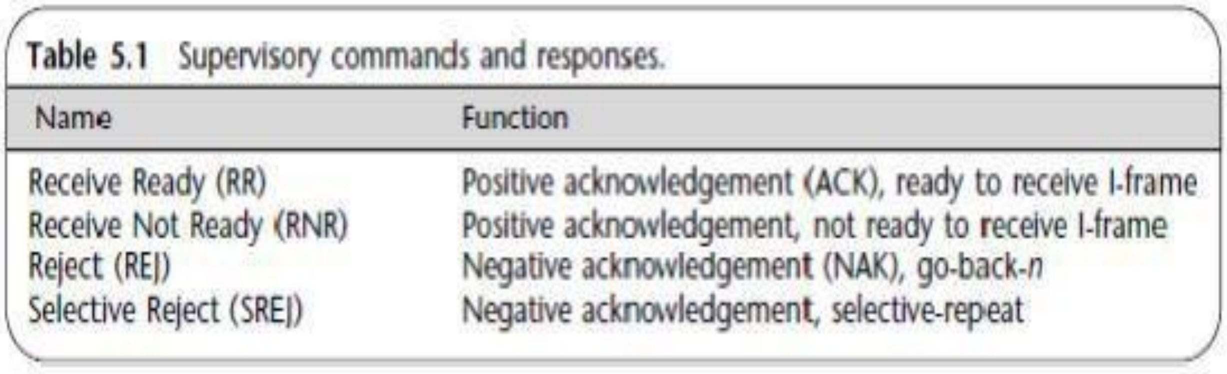


Table 5.1 Supervisory commands and responses.

Name	Function
Receive Ready (RR)	Positive acknowledgement (ACK), ready to receive I-frame
Receive Not Ready (RNR)	Positive acknowledgement, not ready to receive I-frame
Reject (REJ)	Negative acknowledgement (NAK), go-back-*n*
Selective Reject (SREJ)	Negative acknowledgement, selective-repeat

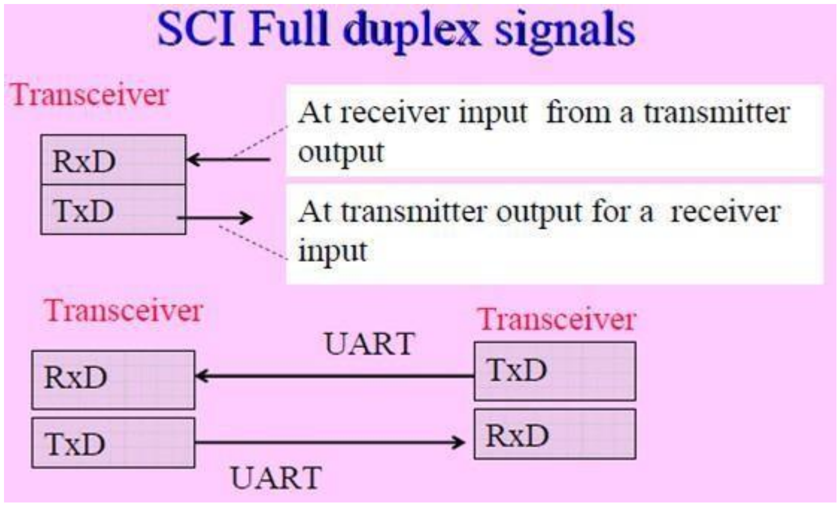
Unnumbered frames

□ Unnumbered frames do not contain any sequence numbers (hence their name) and are used for various control functions.

□ They have five function bits which allow for the fairly large number of commands and responses.

SCI Port (Serial Connect Interface)

SCI is a UART protocol based asynchronous, Full-duplex serial communication port . SCI is programmable for transmission and reception mode of communication


SCI Full duplex signals
Transceiver
RxD
TxD
At receiver input from a transmitter output
At transmitter output for a receiver input
Transceiver
RxD
TxD
UART
Transceiver
TxD
RxD
UART

Features of the SCI port of '68HC11/12' Microcontroller

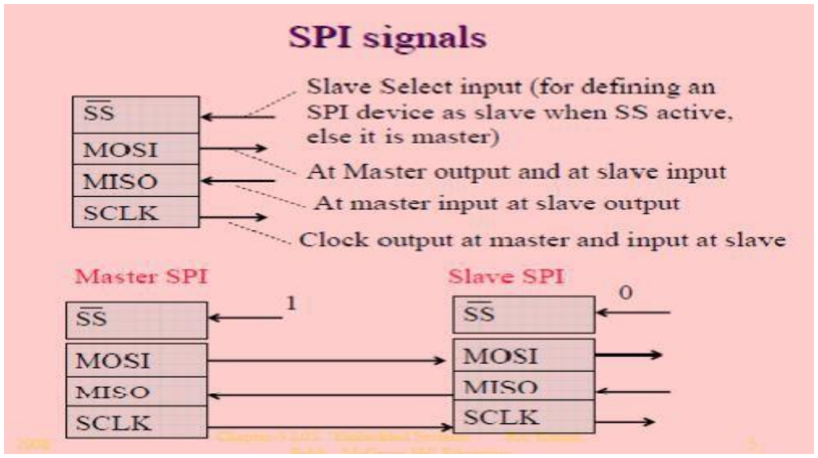
1. UART based Asynchronous Full-Duplex Communication Port

2. Programmable Baud Rate
   - A baud rate can be selected among 32 possible ones by the programming three bits called '**Rate Bits**' & other two bits called '**Pre-Scaling bits**' of the associated serial communication control registers (SCC1 & SCC2) of microcontrollers
   - Baud rates cannot be programmed separately for individual 'Serial In' (Reception) & 'Serial Out' (Transmission) signal lines
   - SCI has two control register bits, T8 and R8 for the inter processor communication in 11-bit format.

3. Programmable Wake-Up Feature for Multi-processor Communication
   - The SCI receiver of microcontroller('68HC11/12) has a 'Wake-Up feature', that can be programmed by an **RWU (Receiver Wake-Up Unavailable**) bit
   - Wake up feature enabled if RWU is set, and is disabled if RWU is reset.
   - If RWU if set, then the receiver of a slave does not interrupt by the succeeding frames.
   - Number of processors can communicate on the SCI bus using control bits RWU, R8 and T8

SPI Bus (Synchronous Peripheral Interface)

SPI is a Full-duplex Synchronous communication protocol. SPI works with Master-Slave mode. It is a 'master-slave' protocol also, in the sense that the master is the unit that generates the clock signal & initiates the data transfer. SPI is a 'Single Master Multi-Slave' system in which only one of the slave is to be enabled at a time. The signals used in SPI are listed in the figure below.


SPI signals
$\overline{SS}$
MOSI
MISO
SCLK
Slave Select input (for defining an SPI device as slave when SS active, else it is master)
At Master output and at slave input
At master input at slave output
Clock output at master and input at slave
Master SPI
Slave SPI
1
0

SCLK is the serial clock from master.

□ MOSI is the output from master.

□ MISO is the input to the master.

□ Device selection as master or slave can be done by applying a signal to input SS pin. When this pin is 0, then the device act as Master. When this pin is 1, then the device act as slave.

□ SPI is programmable for defining the occurrence of positive and negative edges within an interval of bits at serial data out or in

□ SPI is also programmable for open-drain or totem pole output from a master to a slave.

SPI I/O Registers

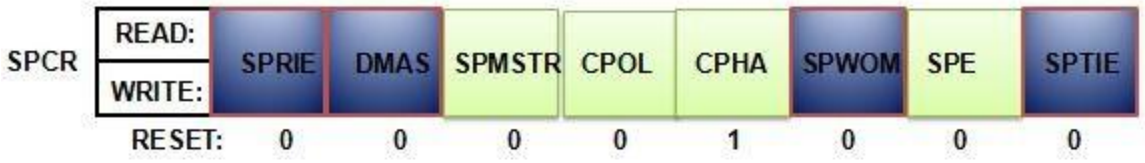
Three registers control and monitor SPI operations:

1. SPI Control Register (SPCR)
2. SPI Status and Control Register (SPSCR)
3. SPI Data Register (SPDR)

SPI Modes

- Master mode
  - Only a master SPI initiates a transmission
  - Data is shifted out via Master Out Slave In (MOSI) line
  - Data is shifted in via Master In Slave Out (MISO) line
  - Transmission ends after 8 cycles of serial clock (SPSCK)
- Slave Mode
  - Transfer synchronized to serial clock (SPSCK) from Master
  - Data is shifted in via the Master Out Slave In (MOSI) line
  - Data is shifted out via the Master In Slave

SPI Control Register

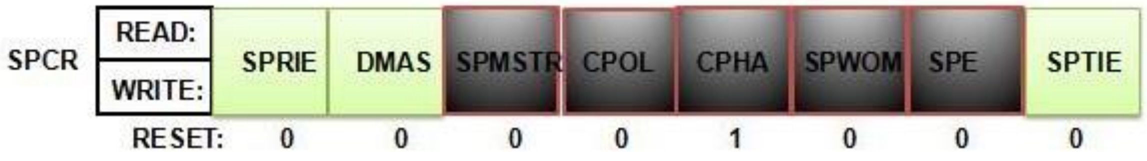


SPCR								
READ: / WRITE:	SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
RESET:	0	0	0	0	1	0	0	0

- **SPI Control Register (SPCR)**
  - SPI Master (SPMSTR):-Selects master mode or slave mode operation
    1 = Master mode, 0 = Slave mode
  - SPI Master and Slave need identical clock polarity and phase settings
  - Clock Polarity (CPOL):- Determines clock state when idle
  - Clock Phase (CPHA)
    1 = Begin capturing data on second clock cycle edge
    0 = Begin capturing data on first clock cycle edge*
    -When CPHA = 0, the SS must be disserted and reasserted between each transmitted byte
  - SPI Enable (SPE) :- 1 = SPI module enabled, 0 = SPI module disabled

Recommend disabling SPI before initializing or changing clock phase, clock polarity, or baud rate

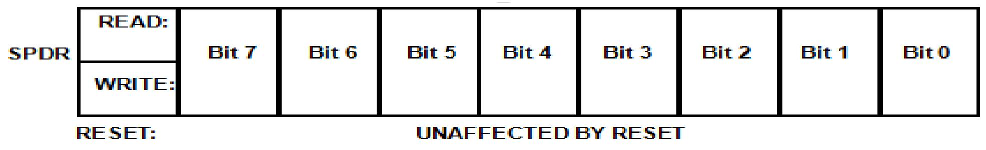
SPI Interrupts



SPCR								
READ: / WRITE:	SPRIE	DMAS	SPMSTR	CPOL	CPHA	SPWOM	SPE	SPTIE
RESET:	0	0	0	0	1	0	0	0

- SPI Receiver Interrupt Enable Bit (SPRIE) :- Interrupt generated when SPRF flag set
- SPI Transmit Interrupt Enable (SPTIE)) :- Interrupt generated when SPTE flag set
  1= Interrupt enabled
  0 = Interrupt disabled
- Direct Memory Access Select (DMAS)
  - Selects either DMA or CPU interrupt request
  - SPRIE/SPTIE bits still enable or disable interrupts

SPI Data Register

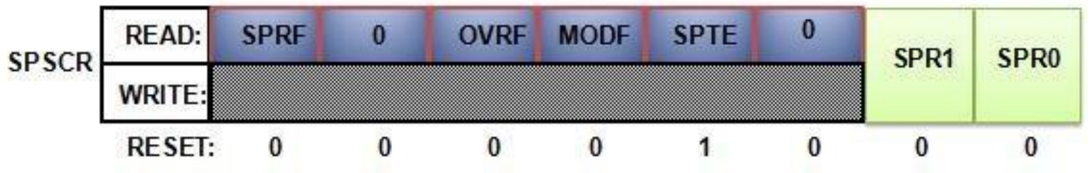


SPDR								
READ: / WRITE:	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
RESET:	UNAFFECTED BY RESET							

- SPI Data Register (SPDR)
  – Read/Write buffer for SPI data
  – Write operation
    - Writes data to transmit data register

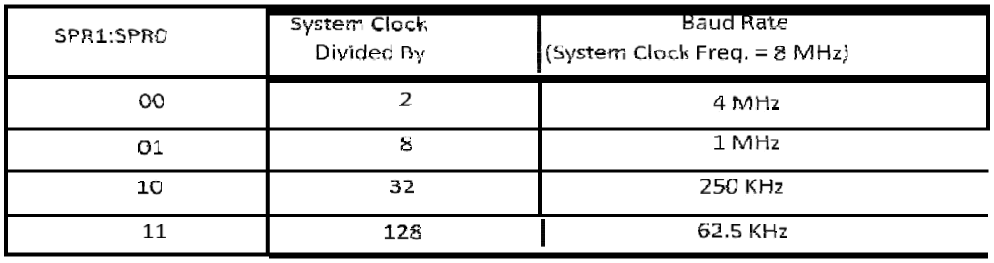
- Read operation
  - Reads data in receive data register

SPI Baud Rate



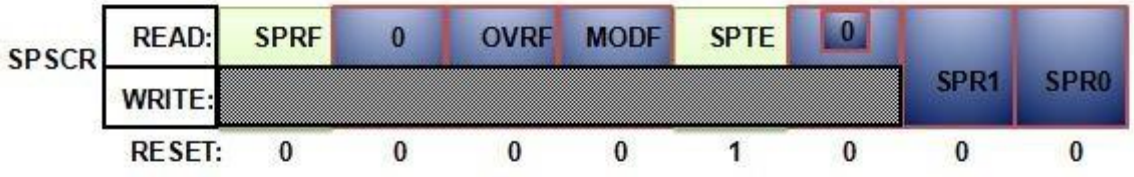
SPI Status and Control Register (SPSCR)

- SPI rate select bits (SPR1, SPR0)
  - Sets the Master SPSCK clock frequency
  - No effect in the Slave devices



SPR1:SPR0	System Clock Divided By	Baud Rate (System Clock Freq. = 8 MHz)
00	2	4 MHz
01	8	1 MHz
10	32	250 KHz
11	128	62.5 KHz

SPI Status Flags



SPI Status and Control Register (SPSCR)

SPI Receiver Full (SPRF)

- Set when a byte is shifted from shift register to the receive data register
- Cleared by reading SPSCR then reading SPDR

  1 = Receive data register full

  0 = Receive data register not full

SPI Transmitter Empty (SPTE)

- Set when a byte is transferred from SPDR to the shift register
- Cleared by reading SPDR register

  1 = Transmit data register empty

  0 = Transmit data register not empty

SPI Initialization sequence

1) Initialize SPI clock frequency (SPR1 and SPR0 in SPSCR)
2) Set clock configuration (CPOL and CPHA bits in SPSCR)
3) Select Master/Slave operation (SPMSTR in SPCR)
4) Enable interrupts if desired (SPTIE, SPRIE in SPCR)
5) Enable the SPI system (SPE in SPCR)

– Should enable Master before Slaves

Master to Slave Transfer

Simple Polled operation

1) Initialize the SPI
2) Select SS to Slave device (hardware dependent)
3) Write byte to SPDR
4) Wait for SPI Transmitter Empty Flag (SPTE)
5) Read the SPDR
6) Release SS to Slave (hardware dependent)

SERIAL BUS COMMUNICATION PROTOCOLS

The commonly used serial bus communication protocols are USB Bus., I²C Bus, CAN Bus

Universal Serial Bus (USB)

USB is used for serial transmission and reception between host and serial devices. It is an 'Asynchronous Serial Bus' between a 'USB Host Controller' & no. of interconnected 'USB peripheral Devices'. It mainly used for the networking of I/O devices like scanner, printer etc. in a PC

Features of USB protocol

(i) Easy Attaching & Detaching of USB devices
(ii) 'Bandwidth Sharing' among USB devices
(iii) Bus-Powered or Self-Powered Devices

- USB 1.0 – 1.5 Mbps(low speed) to 12 Mbps(full speed).
- USB 2.0 – 480 Mbps(high speed).
- USB 3.0 – 5 Gbps.(super speed)
- USB 3.1 - 10 Gbps.(super speed plus)

USB Host & USB Devices

In USB protocol defines two components, a 'Host' which is the 'Bus Master' & many 'Devices' which are 'Slaves'. USB host controls the USB bus & there is only one host per bus. So, a USB host is a 'controller' that can function as bus master for connecting peripheral devices such as pen drive like flash memory devices, digi cam, printer, mouse etc

In USB the host can only act as the bus master by initiating data transfers & the communication takes place between the host & a USB device. But no direct communication possible between the USB devices. By using USB protocol several peripheral Devices/ Nodes can be interconnected using a 'Host Controller' with the help of USB port 'driver' software. Up to 127 devices (including USB hubs) can be connected to a single host controller. The limitation of 127 devices is because the 'address field' in the USB protocol data packet has only 7 bits of length

Host connection to the devices or nodes

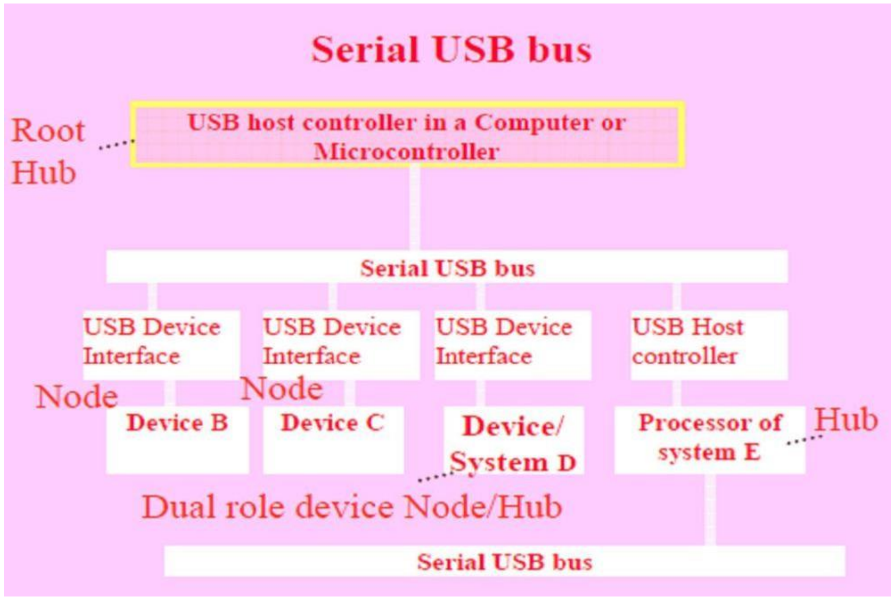
_ Using USB port driving software and host controller,

_ Host computer or system has a host controller, which connects to a root hub.

_ A hub is one that connects to other nodes or hubs.

_ A tree- like topology

Tree Topology of USB 'Hubs' & 'Nodes



A USB system consists of a USB host controller & multiple USB devices connected in a tree-fashion with the help of special devices called 'USB Hubs'. A tree-like topology forms of hub devices & nodes as follows. The 'Root Hub' can connect to the hubs & nodes at 'Level-1'. A hub at 'Level-1' can connect to other hubs & nodes at 'Level-2' and so on up to a max. of 6 levels. Only USB devices will be present at the last level of the tree.

(1) One wire carries +5 V supply (called as VBUS)

(2, 3) Two serial signal wires ('Data+' & 'Data-') forming a Twisted Pair Cable

(4) One wire for ground (GND)

The length of any USB cable is limited to '5 Meters'

USB Protocol

– USB is entirely 'host initiated protocol'.
– All USB peripheral devices are slaves that obey the protocol.
– At a time, either one device or the host can transmit data.
– When the host transmits & if there are many devices in the system, only the 'device' that is 'addressed by the host' can respond.
– If the address broadcasted by the host doesn't matches with the device's address, the USB device simply ignores the communication.

- Data Transfer through 'USB Packets'
  - Here, the data transfer is done using 'USB packets'.
  - The 'USB Data packet' starts with a 'Synchronous Code' called as a 'SYNC' pattern.
  - The clock for data transfer is encoded & inserted in the SYNC field.
  - The receiver USB device synchronizes to the transmitter by recovering the clock continuously from the SYNC field of each packet.
  - SYNC field is followed by a 'PID Field' (Packet Identifier) & 'Data Bytes' of the packet.
  - After data bytes, a CRC Field (Cyclic Redundancy Check) may present for error detection.
  - A USB packet ends with an 'End of Packet' signal
- As in any serial protocol, in USB also data is sent 'one bit at a time'.

There are four ways in which data transfer can be done in USB protocol depending on the type of data

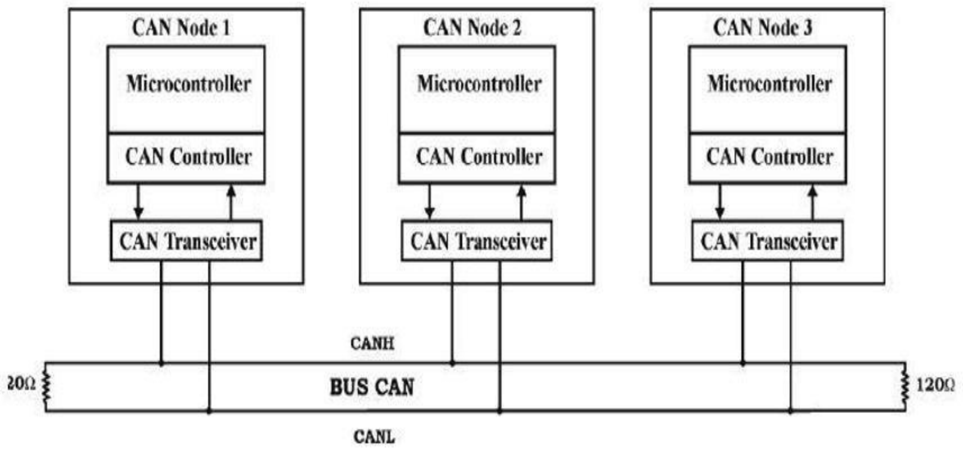
(a) Controlled data transfer
(b) Bulk data transfer
(c) Interrupt driven data transfer
(d) Iso-Synchronous transfer

- **Control Transfer**
  - A control transfer is the one & only 'bi-directional' data transfer type & all the other transfer types are 'uni-directional'.
  - A control transfer is used for the initial configuration of a USB device by the host.
  - Here, the host will do activities such as to read information about a device, set a device's address, select the settings/configurations related to the driver software etc.
- **Bulk Transfer**
  - Bulk transfers are intended for situations where the rate of data transfer isn't critical.
  - They are designed to transfer large amounts of data with error-free delivery.
  - Here, the max. rate of data transfer is not guaranteed at every instant of data transfer
- **Interrupt Transfer**
  - Interrupt transfers are meant for USB devices that needs only a 'periodic attention' from the host.
  - Low speed USB devices like computer mouse & keyboard use this type of transfer - 'No interrupts' are involved here; But the data transfer is similar to an interrupt based system which means that data should be transferred without delay.

- **Isochronous Transfer** –
  - The meaning of the term ‘isochronous’ is ‘at regular intervals’.
  - This is type of asynchronous data transmission done at regular intervals.
  - Here, sending & receiving of data is done in equal time increments.
  - Isochronous transfers have guaranteed time of data delivery.
  - But here there is no error correcting facility because as per the protocol there is no provision for the ‘automatic re-transmitting’ of data received with errors.
  - High speed real-time application like audio & video streaming can be done using this data transfer method

<u>CAN Protocol (Controller Area Network)</u>

- An automobile uses a no. of devices & embedded controllers that are located & distributed inside a car such as devices/ controllers for the brakes control, engine control, electric power control, controlling of lamps, temperature control, air conditioning, dash board display panel, cruise control etc.
- CAN, a protocol standard developed by Robert Bosch for automobile electronics, is a ‘serial bus protocol’ for interconnecting the ‘micro-controller based’ devices of a ‘distributed control area network’



- CAN-bus line usually interconnects to a CAN controller between line and host at the node. It gives the input and gets output between the physical and data link layers at the host node.
- The CAN controller has a BIU (bus interface unit consisting of buffer and driver), protocol controller, status-cum control registers, receiver-buffer and message objects.
- These units connect the host node through the host interface circuit

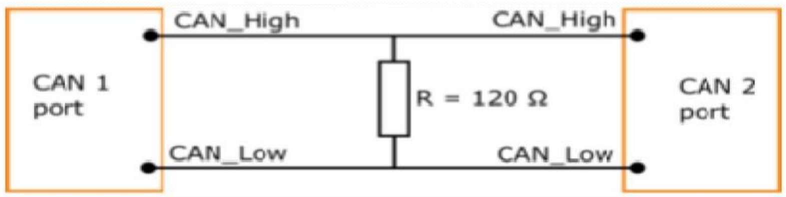
<u>Features of CAN Bus</u>

There is a CAN controller between the CAN line and the host node.

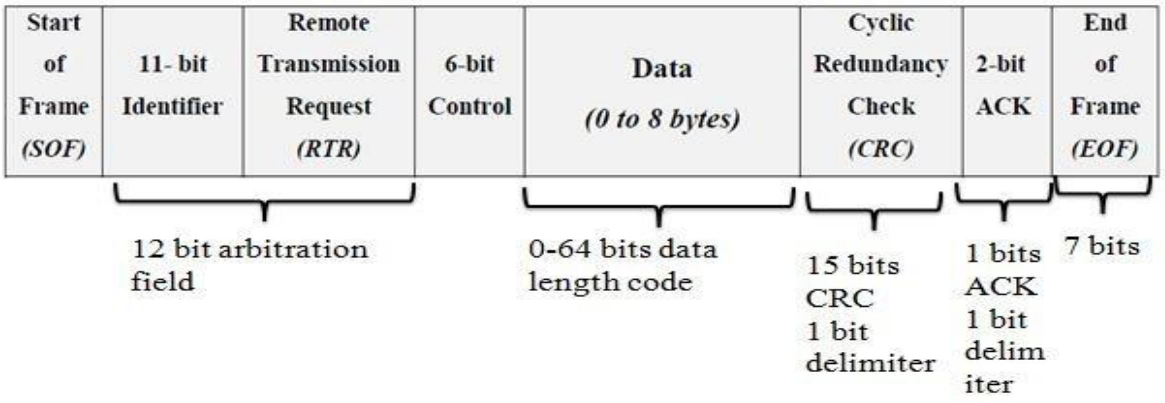
- The CAN protocol based network has only a serial line which is ‘Bi-Directional’.
- A CAN node with a CAN controller can receive or send a bit at any instance by operating at the max. data transfer rate of 1 Mbps.

– CAN bus has 'Multi-Master' & 'Multicasting' of messages as its key features.
_Protocol defined start bit followed by six fields of frame bits.
_ Data frame starts after first detecting that dominant state is not present at the CAN line with logic 1 (R state) to 0 (D state transition) for one serial bit interval

• After start bit, six fields starting from arbitration field and end with seven logic 0s end-field
• 3-bit minimum inter frame gap before next start bit (R→ D transition) occurs
• Differential Signaling in CAN Bus – There are two signal wires CAN_L & CAN_H (Low & High) are seen - This is called 'Differential Signaling' in which the effective signal on the bus is the difference of voltages in these two wires with respect to ground
• When a common noise signal appears on the two wires, they are subtracted off while taking the difference between the two bus signal voltages.
• This makes CAN bus more resistant/immune to noise voltages.



CAN Frames – CAN protocol supports four message formats
(i) Data Frame
(ii) Remote Frame
(iii) Error Frame
(iv) Overload Frame



Start of Frame *(SOF)*	11- bit Identifier	Remote Transmission Request *(RTR)*	6-bit Control	Data *(0 to 8 bytes)*	Cyclic Redundancy Check *(CRC)*	2-bit ACK	End of Frame *(EOF)*

12 bit arbitration field
0-64 bits data length code
15 bits CRC 1 bit delimiter
1 bits ACK 1 bit delimiter
7 bits

Start-of-Frame (SOF)

– The CAN 'data frame' begins with a dominant bit '0' indicating the starting of a frame

Arbitration Field 12 bits :_–

- 11-bit identifier(destination address) field & 1 bit the remote transmission request (RTR)
– The identifier & the RTR bit fields are together called as the 'Arbitration Field' .
– By programming the 'Identifier field', the priority of a message can be fixed.
– Destination device address specified in an11-bit sub-field and whether the data byte being sent is a data for the device or a request to the device in 1-bit sub-field.

- Maximum 211 devices can connect a CAN controller in case of 11-bit
- CAN Identifies the device to which data is being sent or request is being made.
- When RTR bit is at '1', it means this packet is for the device at destination address. If this bit is at '0' (dominant state) it means, this packet is a request for the data from the device.

Control Field & Data Field

- The control field consists of six bits that indicates how many bytes of data follow in the data field .

- The first bit is for the identifier‘s extension.
- The second bit is always '1'.
- The last 4 bits specify code for data Length
- The size of the data field can be zero byte to eight bytes.

Cyclic Redundancy Checksum Field (CRC) – The CRC filed enables the receiver to check if the received data bit sequence was corrupted or not.

Acknowledgement (ACK) - The two-bit acknowledgement field from receiver is used by the transmitter to consider it as an acknowledgment of a valid frame from any receiver.

. ACK = '1' and receiver sends back '0' in this slot when the receiver detects an error in the reception.

• Sender after sensing '0' in the ACK slot, generally retransmits the data frame.

• Second bit 'ACK delimiter' bit. It signals the end of ACK field.

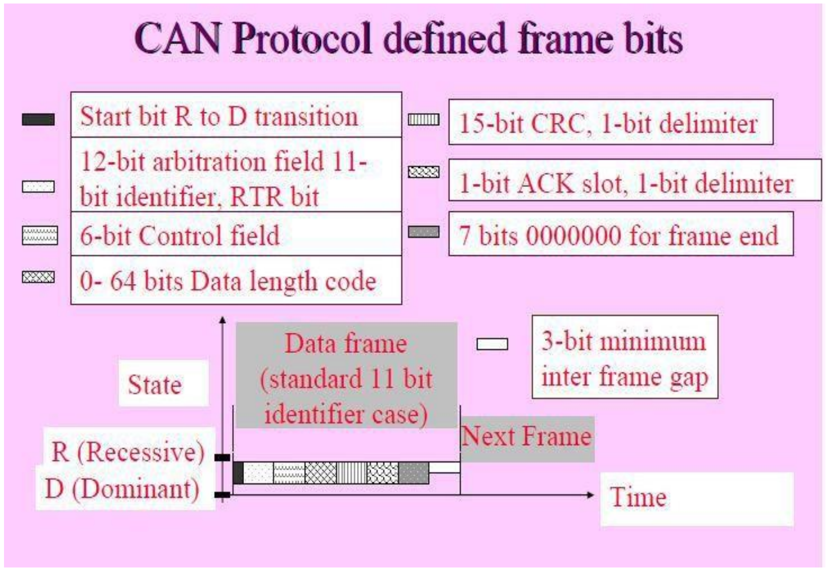
• If the transmitting node does not receive any acknowledgement of data frame within a Specified time slot, it should retransmit.

End-of-Frame (EOF) - The data message frame is terminated by seven bits as ‘0s’ indicating the end of a frame

Can protocol works in two cases

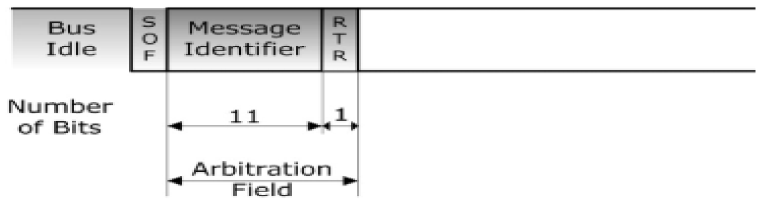
CASE 1 : Only One Node Sends Message to the Network

CASE 2 : Many Nodes Send Messages to the Network simultaneously

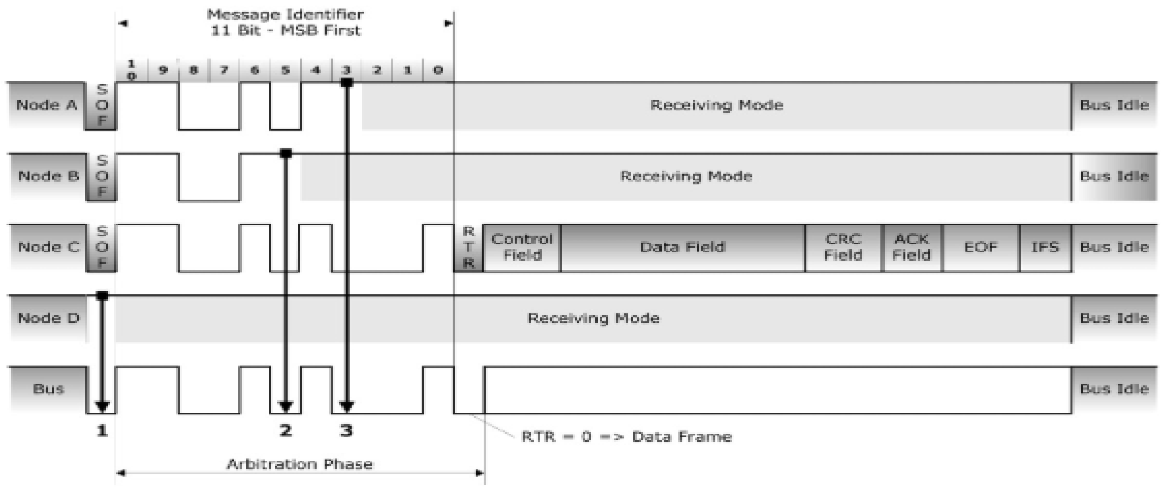

CAN Protocol defined frame bits
Start bit R to D transition
12-bit arbitration field 11-bit identifier, RTR bit
6-bit Control field
0- 64 bits Data length code
15-bit CRC, 1-bit delimiter
1-bit ACK slot, 1-bit delimiter
7 bits 0000000 for frame end
Data frame (standard 11 bit identifier case)
3-bit minimum inter frame gap
State
Next Frame
R (Recessive)
D (Dominant)
Time

Bus arbitration method

- This means that any node is allowed to access the **bus** at any time, if it is idle. If several nodes want to communicate at the same moment, the message with the highest priority wins the **bus arbitration** and gets the right to transmit.
- The message **arbitration** (the process in which two or more **CAN** controllers agree on who is to use the **bus**) is of great importance for the really available bandwidth for data transmission This may result in two or more controllers starting a message (almost) at the same time.
- **CAN** data transmissions are distinguished by a unique message identifier (11/29 bit), which also represents the message **priority**. A low message ID represents a high **priority**. High **priority** messages will gain **bus** access within shortest time even when the **bus** load is high caused by lower **priority** messages.
- The arbitration field follows right after the SOF (Start of Frame) bit and it contains of the message ID and the RTR (Remote Transmission Request) bit.
- Per definition, CAN nodes are not concerned with information about the system configuration (e.g. node address), hence CAN does not support node IDs. CAN data transmissions are distinguished by a unique message identifier (11/29 bit), which also represents the message priority. A low message ID represents a high priority.
- High priority messages will gain bus access within shortest time even when the bus load is high caused by lower priority messages.



Picture 6.1.1: Arbitration Field



Picture 6.1.2: Bus Arbitration Example

Picture 6.1.2 shows an example where three nodes in a four node CAN network try to access the bus at virtually the same time. In this example node C will win the bus access within 12 clock times. At a baud rate of 1 MBit/sec this would translate into 12 microseconds. Naturally, the bus arbitration time varies with baud rate and the message identifier length, 11 or 29 Bit.

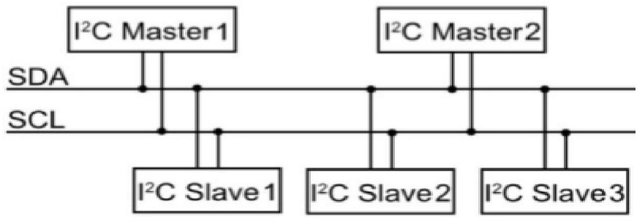
I2C (Inter-Integrated Circuit)

This is a ‘ Two-Wire’ protocol developed by Philips for interconnecting Ics. It is a ‘Serial Bus Protocol’ which is synchronous Serial Bus Communication for networking. It having half-Duplex communication. It is a ‘Multi-Master Multi-Slave’ protocol. It is also a ‘Byte sized data’ oriented protocol with ‘Acknowledgement’ signals. After each byte received, an ‘Acknowledgement’ is to be sent by the by the receiver-IC to the sender-IC.

I²C Bus communication use of only simplifies the number of connections and provides a common way (protocol) of connecting different or same type of I/O devices using synchronous serial communication. The I2C Bus has two signal lines/ wires that carry its signals, they are:

(i) SCL (serial clock):- The signal line/wire for ‘Clock’

(ii) SDA (serial data) :- The line for ‘Bidirectional serial Data’ - Both of these two signal wires are ‘bi-directional’.



Device Addresses and Master in the I²C bus

- Each device has its own 7-bit ‘Unique Address’ using which the data transfers take place.
- Master IC has processing units that can function as bus controller with bus interface circuits
- The master will usually be a ‘Microcontroller’ that can transmit & receive. These microcontrollers have ‘I²C hardware’ within the chip and SDA, SCL lines as pins.
- The master can address ‘127 slaves’ at an instance (Multiple Slaves) - Slaves will usually be ICs like - an external ROM chip from which master can only read , - a LCD module to which master can only write onto , - an external Flash Memory chip (RAM) to which master can both read & write.
- Multiple masters can also connect to the I²C bus
- But, at any instance, only one IC can become master & that master IC is the one who initiates a data transfer on SDA line by transmitting the SCL serial clock pulses

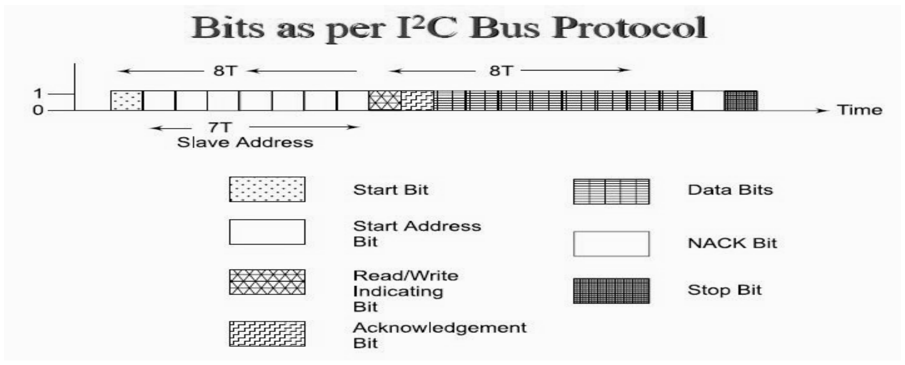
Data Frame of I2C- Synchronous Serial Bus Fields and its length

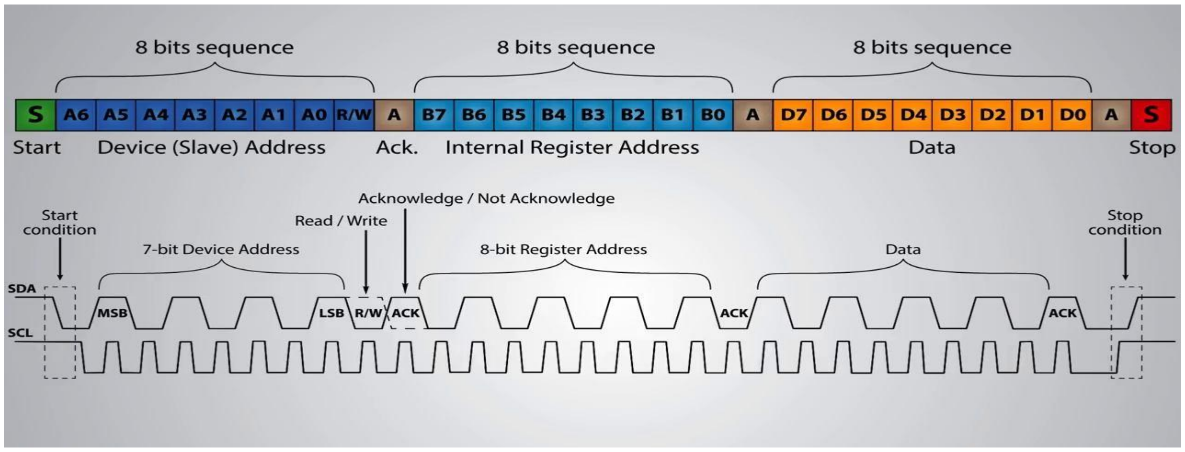
– First field of 1 bit─ Start bit similar to one in an UART

– Second field of 7 bits─ address field. It defines the slave address, which is being sent the data frame (of many bytes) by the master

– Third field of 1 control bit─ defines whether a read or write cycle is in progress

- Fourth field of 1 control bit— defines whether is the present data is an aacknowledgment (from slave)
- Fifth field of 8 bits— **I²C device data byte**
- Sixth field of 1-bit— bit NACK (negative acknowledgement) from the receiver. If active then acknowledgment after a transfer is not needed from the slave, else acknowledgement is expected from the slave
- Seventh field of 1 bit — stop bit like in an UART





Important Properties of I2C protocol

(1) I2C protocol is simple, But I2C is not as fast as SPI protocol

(2) There are 3 standards for I2C bus with the following data transfer speeds:

(i) Slow (under 100 Kbps)

(ii) Fast (400 Kbps)

(iii) High-Speed (3.4 Mbps)

(3) Commonly used I2C devices include EEPROMs, Sensor Modules, Real-Time Clock (RTC) ICs & similar peripheral chips which can have only one data line & a clock line

(4) Many standard microcontrollers such as PIC, AVR, and ARM etc. have on-chip I2C hardware module along with SDA & SCL pins

PARALLEL BUS DEVICE PROTOCOLS

A computer system connects to other sub-systems (such as of I/O devices) at high speed at very short distances (<25 cm) using a 'Parallel Bus. A parallel bus has a large no. of lines/ wires as per a 'simple parallel bus protocol'

- Eg. of parallel buses are **ISA, PCI, 'AMBA'** bus interface of **ARM** etc.

The parallel port has the facility to transfer data both in & out between the PC and the outside world. Parallel port is an inexpensive connector tool for connecting computer devices. The parallel port is often available in the motherboard of PCs. More no. of parallel ports can be added to a computer using ISA or PCI parallel port interfacing standards

ISA (Industry Standard Architecture)

ISA Bus Protocol - ISA is a commonly used 'parallel bus protocol standard' for IBM compatible 8086 processor based computer - There are two versions of ISA standards:

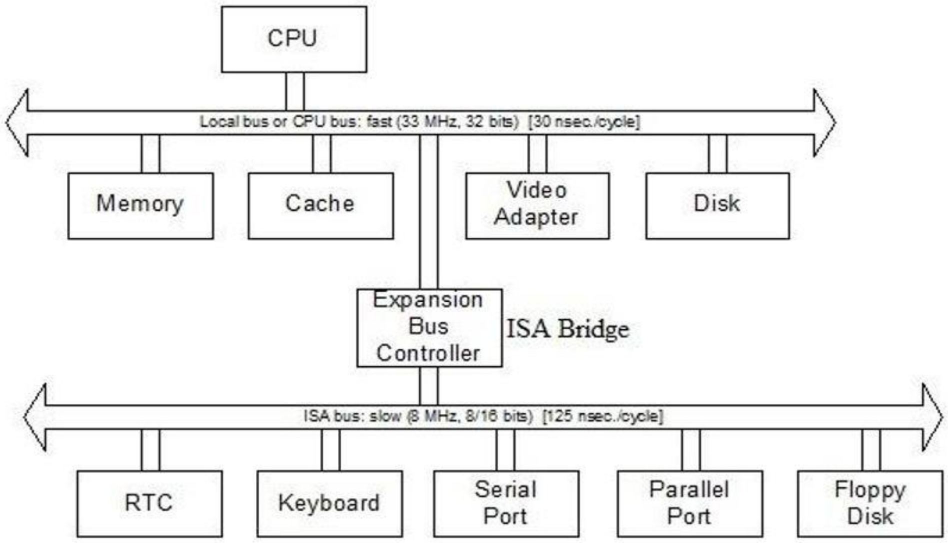
(i) 8-Bit ISA (ii) 16-Bit ISA.

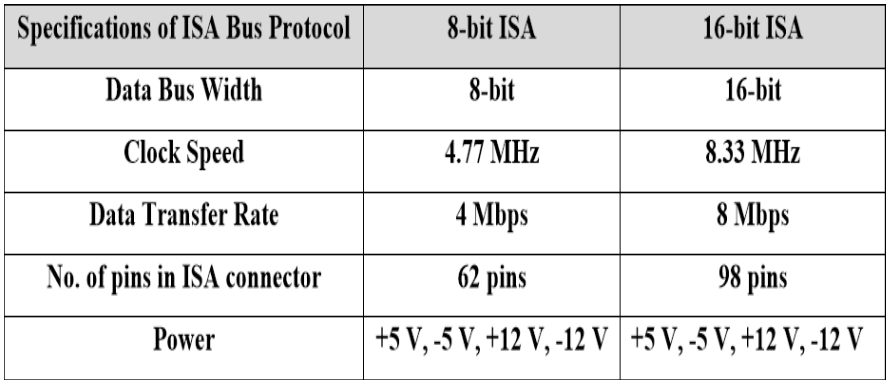
ISA is a type of 'Parallel Bus' used in PCs for adding 'Expansion Cards'. The ISA slots in a motherboard can be used to insert expansion cards to enhance the performance of a computer. An Expansion card (Expansion Board/ Daughter Card/ Interface Card/ Add-on Card) is an electronic board that is used to add extra functionality to a computer

Eg. for expansion cards are sound card, video card, graphics card, network card (for LAN), dial-up modem etc

An ISA bus can connect only to an embedded device that has 80x86 processor. ISA buses are used for connecting peripheral devices/chips as per the I/O device address range & interrupt vector addresses designed in IBM PC architecture using 80x86 processor. - Here, the 8086 processor's addressing modes & the interrupt vector address assignments are taken into account.

ISA bus based IO Port addressing on IBM PC architecture


CPU
Local bus or CPU bus: fast (33 MHz, 32 bits) [30 nsec./cycle]
Memory
Cache
Video Adapter
Disk
Expansion Bus Controller
ISA Bridge
ISA bus: slow (8 MHz, 8/16 bits) [125 nsec./cycle]
RTC
Keyboard
Serial Port
Parallel Port
Floppy Disk



Specifications of ISA Bus Protocol	8-bit ISA	16-bit ISA
Data Bus Width	8-bit	16-bit
Clock Speed	4.77 MHz	8.33 MHz
Data Transfer Rate	4 Mbps	8 Mbps
No. of pins in ISA connector	62 pins	98 pins
Power	+5 V, -5 V, +12 V, -12 V	+5 V, -5 V, +12 V, -12 V

PCI (Peripheral Component Interconnect)

A 'PCI' device is any piece of computer hardware that plugs directly into a 'PCI slot' on a computer's motherboard. PCI based Network cards, Audio cards etc are available. A PCI device or card/ board can be inserted into PCI slot of computer motherboard & it is automatically recognized & configured to work in our system (Plug-&-Play). PCI connects at high speed to other subsystems having a range of I/O devices at very short distances (<25 cm) using a parallel bus without having to implement a specific interface for each I/O device.

PCI Bus Protocol for Parallel Data Communication

PCI is an interconnection system between micro-processor & attached devices in which PCI connection slots are spaced closely for high speed operation. PCI protocol specifies the interaction between the different components of a computer. The PCI bus operates slow, when it is connected to low speed devices. But, PCI also provides good advantages for higher speed devices like audio & video streaming, gaming, high-speed modems etc.

PCI Bus

The PCI is a high performance 32-bit or 64-bit 'synchronous bus' used for parallel bus communication. PCI is implemented with multiplexed address & data bus which allows reduced pin count in the connectors. A specification version 2.1─synchronous/asynchronous throughput is up to 132/ 528 MB/s [33M× 4/ 66M× 8 Byte/s], PCI supports devices that works at '5 Volts' or '3.3 Volts'. PCI based devices/ cards have plug & play facility on PCI slots i.e PCI driver can access the hardware automatically and assigns new addresses Thus, simplified addition and deletion (attachment and detachment) of the system peripherals.

FIFO in PCI device/card:- _ Each device may use a FIFO controller with a FIFO buffer for maximum throughput.

Identification Numbers :- _ A device identifies its address space by three identification numbers,

(i) I/O port

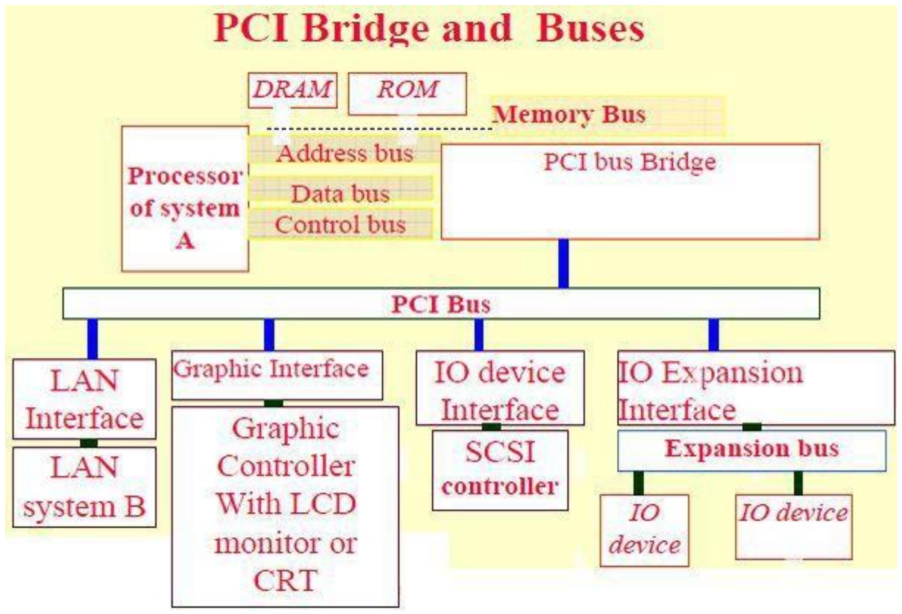
(ii) Memory locations and
(iii) Configuration registers of total 256B with a 4-byte unique ID.
Each PCI device has address space allocation of 256 bytes to access it by the host Computer

PCI device identification

_ A 16-bit register in a PCI device identifies this number to let that device auto- detect it.
_ Another 16-bit register identifies a device ID number. These two numbers let allow the device to carry out its auto-detection by its host computer.

PCI bridge

_ PCI bus interface switches a process or communication with the memory bus to PCI bus.
_ In most systems, the processor has a single data bus that connects to a switch module
_ Some processors integrate the switch module onto the same integrated circuit as the processor to reduce the number of chips required to build a system and thus the system cost.
_ Communicates with the memory through a memory bus (a set of address, control and data buses), a dedicated set of wires that transfer data between these two systems.
_ A separate I/O bus connects the PCI switch to the I/O devices.



PCI-X Bus

'PCI-X' is an extension of PCI which supports 64 bit / 100 MHz data transfers Both 'PCI' & 'PCI-X' buses doesn't follow the IBM PC architecture- ie. PCI & PIC-X buses are independent from the IBM PC architecture PCI has a speed of 133 to 533Mbps but PCI-X having more than 1066Mbps. ie 1 Gbps

PCI –X is Backward compatible with existing PCI cards. It is used in high bandwidth devices (Fiber Channel, and processors that are part of a cluster and Gigabit Ethernet). It uses maximum 264 MBps throughput, uses 8,16 , 32, or 64 bit transfers. 6U cards contain additional pins for user defined I/Os. It has a live insertion support (Hot-Swap), it supports two independent buses on the back plane (on different connectors). It also supports Ethernet, Infiniband, and Star Fabric support (Switched fabric based systems) Compact PCI (cPCI)

Advantages & Benefits of PCI

(i) Very High Speed . -The clock speed (MHz) of a PCI bus works independently of the CPU speed.

(ii) ‘Plug-&-Play’ Facility of PCI cards - PCI is auto-configurable & is fully controlled by software and not by jumpers or hardware switches on a board - This is a feature referred to as "Plug-&-Play"

(iii) Platform-Independent - Both PCI & PCI-X works independent of the CPU - Unlike the ISA, both are ‘platform-independent’(where ISA depends on the IBM PC architecture platform)

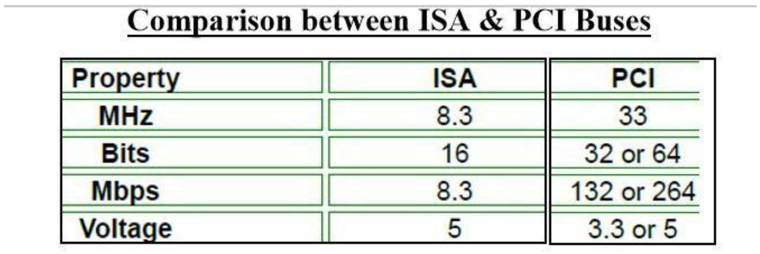
(iv) 3.3 Volts Operation - Unlike ISA, a PCI bus can operate at 3.3V which is important for the battery life on portable devices

(v) More Throughput.

(vi) Dominant & Reliable parallel bus standard - PCI bus requires less components, pins & boards, so there is a higher reliability for board-level buses

Features of PCI bus & PCI protocol

(1) Synchronous bus architecture.
(2) 64-Bit Addressing .
(3) Linear Burst Mode Data Transfer .
(4) Large Bandwidth.
(5) The PCI driver software can automatically access the hardware.



Comparison between ISA & PCI Buses

Property	ISA	PCI
MHz	8.3	33
Bits	16	32 or 64
Mbps	8.3	132 or 264
Voltage	5	3.3 or 5

MODULE 3

Subject: EC 308 EMBEDDED SYSTEMS

Module 3

MEMORY DEVICES

Memory is the storage device in a system used to store data and instructions. Memory chips do not typically have separate READ & WRITE pins. A read/ write signal pin (R/W) controls the direction of data transfer. Most memories include an 'ENABLE signal pin' that controls the tri-stating of data onto the memory's pins. The enable pin can be used to easily build large memories from multiple memory banks.

CHARACTERISTICS OF MEMORY

1. **Memory Capacity** – The basic characteristic of a memory is its 'Capacity' or 'Size' (in Kilo/Mega/Giga bits).But, several versions will be available for a memory of a same size, each with a different data width.

For e g, a 256 MB memory may be available in two versions.

i. (64Mega) X (4-bit) Array.

ii. (32 Mega) X (8-bit) Array

2. **Aspect Ratio** - Internally, the data are stored as a two-dimensional array of cells in a memory chip. The 'Height/ Width' ratio of a memory is known as its aspect ratio. The best aspect ratio depends on the amount of memory required.

The array of memory is in two dimensional, the 'n-bit address' received by the memory chip is split into a 'Row Address (r)' & a 'Column Address (c)' with 'n' = 'r' + ' c' . The row & column select a particular memory cell.

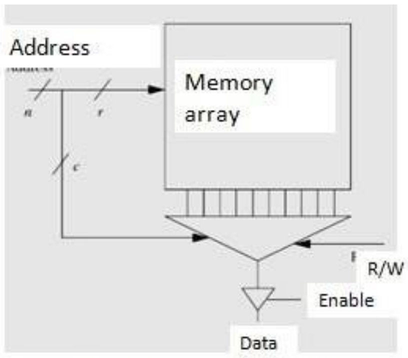


Fig. shows internal organization of a memory device

Most systems consist of two types of memory, Read-only Memory (ROM) and Random-Access Memory (RAM).

ROM :-This type of memory is non-volatile. ROM image holds the programs, operating system, and data required by the system. Following are the various types of ROM:

1. MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre programmed set of data or instructions. These kind of ROMs are known as masked ROMs. Ø MROM programming is performed during IC fabrication. It is inexpensive ROM. It is used for large scale manufacturing

Uses: 1. A finalized ROM image of system program.

2. Data, pictograms, image pixels, pixels for the fonts of a language.

2. PROM (One Time Programmable ROM –OTP ROM)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM programmer. A PROM once written is not erasable.

Uses: 1. Smart card identity number and personal information.

2. Storing boot programs.

3. ATM card or credit card or identity card.

3. EPROM (Erasable and Programmable ROM)

Used in place of masked ROM during development phase. UV Erasable and Electrically programmable by a device programmer

4. EPROM (Erasable and Programmable ROM)

The EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of re programming is flexible but slow.

Uses: 1. Storing current date and time in a machine.

2. Storing port status.

3. Storing driving, malfunctions and failure history in an automobile.

5. Flash Memory

Flash memory is a form of EEPROM in which a sector of bytes can be erased in a flash (very short duration corresponding to a single clock cycle)

Uses: 1. Storing pictures in digital camera.

2. Storing SMS, MMS messages in a mobile phone.

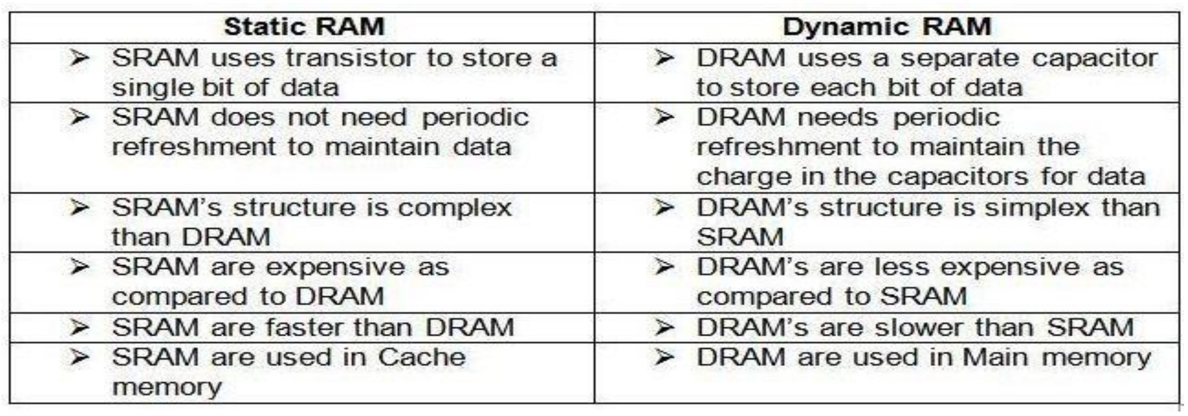
3. Storing voice compressed form in a voice recorder.

6. FACTORY-PROGRAMMED ROM - Factory-programmed ROMs are ‘already data written’ memories that user can’t alter. They ordered from the manufacturing factory with particular data written onto it as per the customer’s request.

7. FIELD-PROGRAMMABLE ROM - Field-programmable ROMs can be programmed by the user using a special hardware set-up for writing & erasing the ROM . ‘Flash memory’ is a field-programmable ROM which is ‘electrically erasable’. Flash memory uses standard system voltage for erasing & programming which allows it to be erased & re-programmed using a computer system. Early flash memories had to be erased in their entirety but modern flash devices allow memory to be erased in block.

RAM

RAM is used to hold the programs, operating system, and data required by a computer system. In embedded systems, it holds the stack and temporary variables of the programs, operating system, and data. RAM is generally volatile, (does not retain the data stored in it when the system 's power is turned off). It is used for saving the variables, stacks, process control blocks, input buffer, output buffer, decompressed format of program and data at the ROM image. Following are the various types of RAM:



Static RAM	Dynamic RAM
➢ SRAM uses transistor to store a single bit of data	➢ DRAM uses a separate capacitor to store each bit of data
➢ SRAM does not need periodic refreshment to maintain data	➢ DRAM needs periodic refreshment to maintain the charge in the capacitors for data
➢ SRAM’s structure is complex than DRAM	➢ DRAM’s structure is simplex than SRAM
➢ SRAM are expensive as compared to DRAM	➢ DRAM’s are less expensive as compared to SRAM
➢ SRAM are faster than DRAM	➢ DRAM’s are slower than SRAM
➢ SRAM are used in Cache memory	➢ DRAM are used in Main memory

1. EDO (Extended Data Out) RAM

EDO RAM is used in systems with clock rates up to 100 MHz. Zero wait state is needed between two fetches. Single cycle read or write is possible.

2. SDRAM (Synchronous DRAM)

SDRAM Synchronizes the read operation and keeps next word ready while previous one is being fetched; SDRAM is used in systems with clock rates up to 1 GHz.

3. Double-Data Rate SDRAMs (DDRs) –

Faster synchronous DRAMs known as ‘Double-Data Rate Synchronous DRAMs’ (DDR SDRAMs) or simply DDRs. DDRs use sophisticated clocking circuit techniques to perform more memory access operations per clock cycle (Typically ‘Two memory access’/ single Cycle by using both rising & falling edges of clock) .

4. RDRAM (Rambus* DRAM)

RDRAM accesses in bursts of four successive words in single fetch; used for 1 GHz + performance of the system

5. Parameterised Distributed RAM

Parameterized Distributed RAM is the RAM distributes in various system subunits. I/O units and transceiver sub units can have a slice of RAM each.

MMU & MEMORY MAPPING

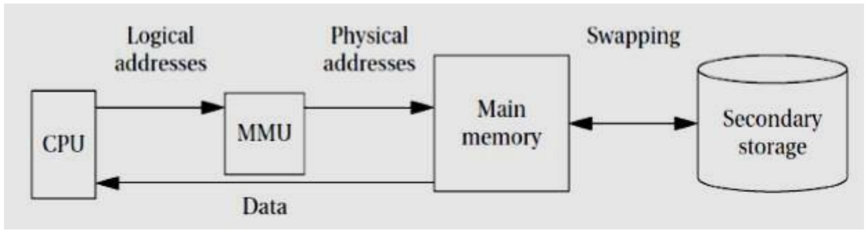
A Memory Management Unit (MMU) translates addresses between the CPU & the 'Physical Memory' (or physically present main memory area/'addresses'). This 'Address Translation Process' is known as 'Memory Mapping'. Here, the logical addresses are mapped from a logical space into a physical space. Memory Management Units (MMUs) appear as a part of the processor. MMUs allow the system software(OS) to run & manage multiple programs in a single physical memory, in which different programs resides at different address spaces.

Memory Map

A **memory map** is a massive table, in effect a database that comprises complete information about how the memory is structured in a computer system. Map to show the program and data allocation of the addresses to ROM, RAM, EEPROM or Flash in the system. It reflects the addresses available to the memory blocks and IO devises. It also reflects a description of memory and IO devices in the system hardware. It reflects memory allocation for the programs, data and IO operations by a locator program.

Virtual Memory System

MMUs are used to provide 'Virtual Addressing' shown in fig



Logical Address - The MMU accepts logical addresses from the CPU. Logical addresses (or Virtual Addresses) refer to the abstract address generated by the CPU. But do not correspond to actually present physical address locations of main memory. MMU translates logical addresses to physical addresses that correspond to actual RAM locations with the help of MMU tables. By changing the MMU's tables, we can even change the physical location at which the program resides.

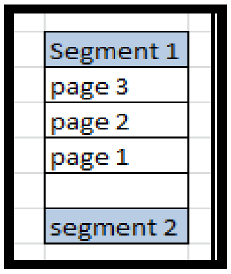
Page Fault - In a virtual memory system, the MMU keeps track of which of the logical addresses are actually resident (or present) in main memory .

- Those logical addresses that do not reside (or present) in main memory are kept on the secondary storage device (such as disk or flash memory devices).
- When the CPU requests an address that is not in main memory, the MMU generates an 'Exception' called as a 'Page Fault'(An 'Exception' is referred to as a software interrupt that detects an error condition) .
- Then an 'Exception Handling Program code' (similar to an ISR) is executed to read the requested location from the secondary storage device into main memory.
- After the required memory has been read from main memory & the MMU's tables have been updated to reflect the changes

MEMORY MAPPING THROGH ADDRESS TRANSLATION

Virtual memory is implemented either through 'Paging' or 'Segmented' schemes.

Memory Segments – memory segments are 'Large & Arbitrarily Sized Regions of Memory'. A segment is usually described by its starting address & size. Segmentation scheme allows different segments to be of different sizes.



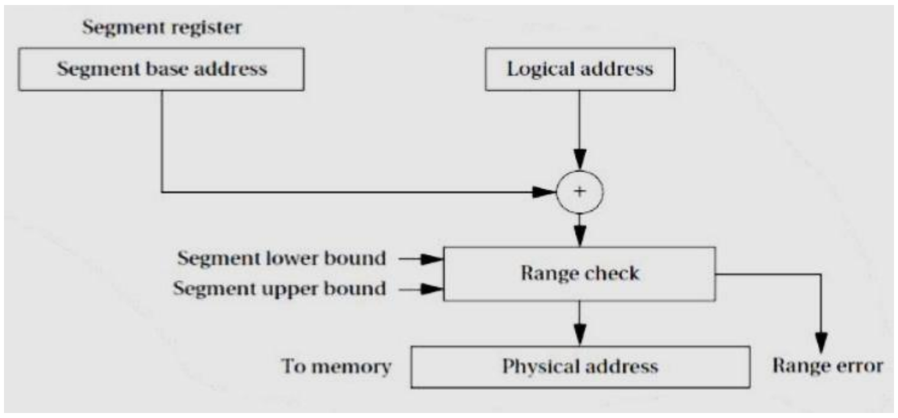
- Pages - Pages describe 'Small & Equally Sized Memory Regions'. Pages are of same size & this feature simplifies the hardware required for address translation.
- A segmented, paged scheme is created by dividing each segment into pages & using two steps for address translation.

Address Translation Schemes - There are two styles of address translation:

(i) Segmentation

(ii) Paging - The two schemes can be combined to form a Segmented, Paged addressing scheme.

Address Translation for 'Segmented Addresses' –



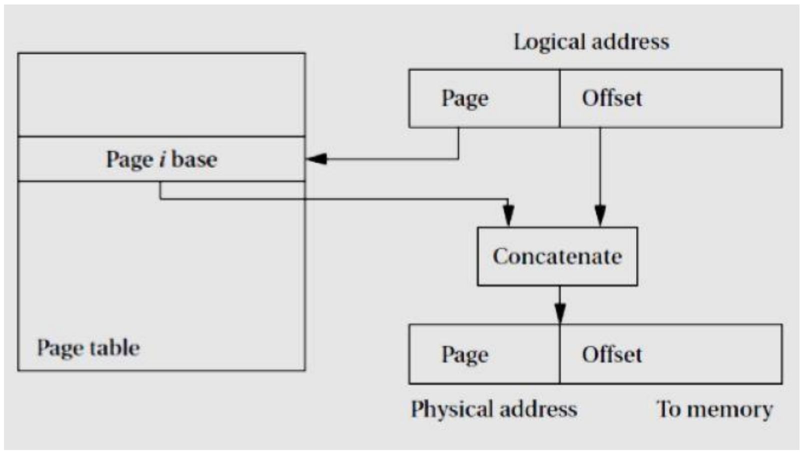
Here, the MMU maintains a 'Segment Base Register' that contains the base address (or points to the starting) of the currently active/ used memory segment. The logical address from

CPU is used only as an 'Offset' for this segment base. The 'Physical Address' is formed by adding the 'Segment Base' to the 'Offset'. (Physical Address = Segment Base Address + Offset). Segmentation schemes also check the physical address generated with the upper & lower address limit of the current segment with the help of its segment size & the offset (ie. The allowed 'address range' for the current segment is checked with the physical address generated).

Address Translation for 'Paged Addresses'

The translation of paged addresses is a simpler calculation; but requires more MMU state

As shown in fig, the logical address is divided into two sections: **a 'Page Number' & an 'Offset'**. CPU maintains table that stores the starting addresses of all pages present in the physical memory. The page table is normally kept in main memory. Here, the 'Top bits of logical address' represents a pointer to the 'Page Table'. The 'Bottom Bits' of logical address is the 'Offset' within that page. The MMU concatenates the 'Page Starting Address' from the page table with the 'Offset' to form the Physical Address. All pages have uniform size with proper page boundaries. Pages are small, typically between 512 bytes & 4 KB, so the page table is large for CPUs with a large physical memory area/ address.



TRANSLATION LOOKASIDE BUFFER (TLB)

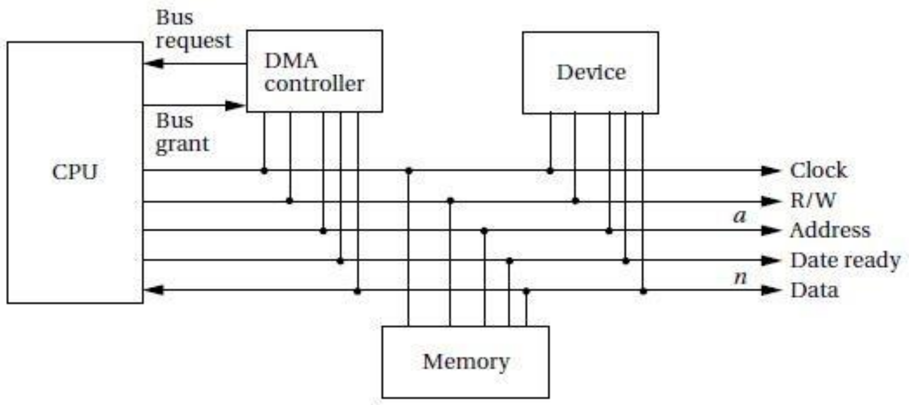
The efficiency of paged address translation can be increased by caching page translation information. A 'Cache memory' for storing 'Address Translation information' is known as a 'Translation Lookaside Buffer' (Also called as TLB cache). The MMU reads the TLB to check whether a page no. is currently in the TLB cache. If so, the MMU uses that value rather than reading from memory MMU of ARM Architecture. A MMU is an optional part of the ARM architecture & the ARM MMU supports virtual address translation.

The ARM MMU supports the following types of memory regions for address translation:

(i) a 'Section' is a 1-MB block of memory
(ii) (ii) a 'Large Page' is 64 KB
(iii) (iii) a 'Small Page' is 4 KB.

DIRECT MEMORY ACCESS (DMA)

Direct memory access (DMA) is a bus operation that allows read and write between IO devices and memory not controlled by the CPU. A DMA transfer is controlled by a ***DMA controller***, which requests control of the bus from the CPU. After gaining control, the DMA controller performs read and write operations directly between devices and memory. Figure shows the configuration of a bus with a DMA controller.



The DMA requires the CPU to provide two additional bus signals. The ***bus request*** is an input to the CPU through which DMA controllers ask for ownership of the bus. The ***bus grant*** signals that the bus has been granted to the DMA controller. The 'Bus Grant' is asserted by the CPU when the system bus is ready. The DMA controller uses the above two handshaking signals to gain control of the bus using the normal four-cycle handshake signaling scheme for system bus operation. The CPU will finish all pending bus operations before granting control of the bus to the DMA controller & thus DMA controller becomes 'Bus Master'. When CPU grants control, it stops generating (or driving) the other bus control signals such as R/W signals, Address signal etc.

DMA CONTROLLER AS BUS MASTER - Upon becoming bus master, the DMA controller has control of all bus signals such as 'R/W', 'Address' etc. If the DMA controller is bus master, it can perform read & write operations using the same bus protocol used for CPU based bus operations. After a data transfer is finished, the DMA controller returns the bus to the CPU by de-asserting (or by removing) it's 'Bus Request' signal towards the CPU. This will make the CPU to de-assert (or remove) the 'Bus Grant' signal issued earlier.

DMA REGISTERS - The CPU controls DMA operations through special control registers in the DMA controller. A typical DMA controller includes the following three control registers

(i) Starting Address Register : Specifies from where the transfer is to begin .

(ii) Length Register : Specifies the no. of words to be transferred.

(iii) Status Register : Allows the DMA controller to be operated by the CPU.

The CPU initiates a DMA based transfer by configuring the 'Starting Address' register & 'Length register' appropriately. Then CPU set a 'Start Bit' in the 'Status Register' to start

DMA transfer. During a DMA transfer CPU can't use the bus & can't do bus operations. If CPU needs the bus during a DMA operation, it waits until the DMA controller returns bus master-ship (or bus-ownership) to the CPU. But, if the CPU has instructions & data in the cache or in CPU registers, CPU can do some other useful work for some time using the data in cache/ CPU registers. After the DMA operation is complete, the DMA controller interrupts the CPU to inform that the DMA based data transfer is completed - To prevent the CPU from waiting for too long. Most DMA controllers implement 'Data Block' based data transfer that occupy the bus for only a few cycles at a time ie. the DMA transfer will be done only for a block of data at a time. After each block, DMA controller returns bus control to the CPU & then DMA controller goes to sleep for some time, after which it requests the bus again for the next block transfer.

I/O Devices

Many I/O devices are used in embedded computing systems such as

(i) Timers & Counters
(ii) 'Analog-to-Digital' & 'Digital-to-Analog' Converters
(iii) Keyboards
(iv) Light-Emitting Diodes
(v) Displays
(vi) Touch screens.

Some of the above devices are available as on-chip devices in the micro-controllers; others are implemented separately.

(i) **TIMERS & COUNTERS** - 'Timers' & 'Counters' use same logic circuit for their operation, but they differ from each other by their use. Timers/ Counters have registers to hold the 'Current Count Value' & an 'Increment Input' (or Update) signal to increment the current register value by one.

Difference between Timer & Counter

✓ In 'Timers', the counting operation is done based on a 'Periodic Clock Signal'.

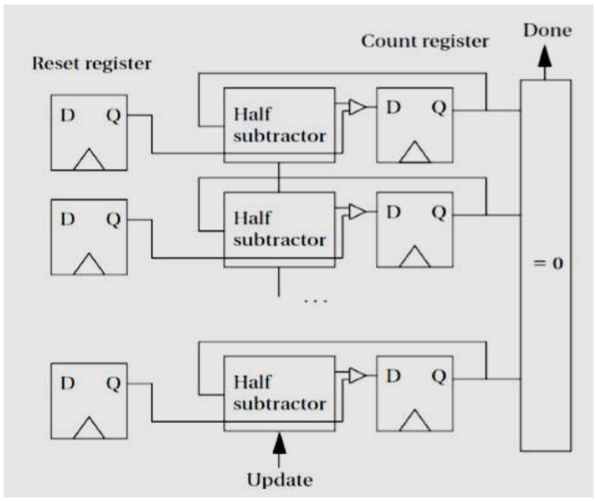
✓ For a 'Counter', it's clock input connected to an 'Aperiodic Signal' for counting the no. of occurrences of an external event.

✓ Since, both of them use the same logic, the device is often called a 'Counter/ Timer' - Fig. shows the internal logic circuits of a counter/timer.

✓ A 'counter/ timer' uses a 'Array of Half-Subtractors' to decrement the count when the count 'Update' signal is asserted.

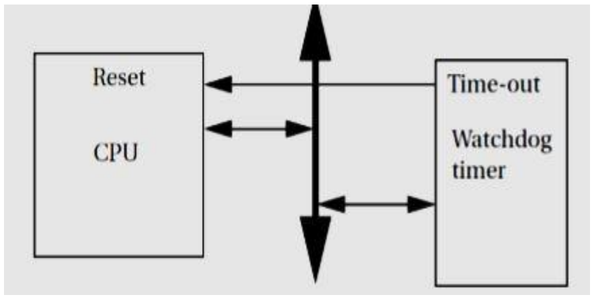
✓ A 'counter/ timer' has an 'n-bit register' called the 'Count Register' to store the current state/ value of counting.

- ✓ A combinational logic circuit block (shown as '= 0' block) checks whether the count becomes zero (Count = 0) - The 'Done' is an output signal to indicate the 'Zero Count'.
- ✓ The 'Starting Count' value of the counter/timer can be provided through the 'Reset Register' though which the 'initial count state' of the 'Count Register' is loaded.
- ✓ The reset register can be loaded with required starting values by using some other circuits of the counter/timer peripheral of a micro-controller.



Cyclic & Acyclic Modes of Operation of Counter/ Timer

Most counters provide both 'Cyclic' & 'Acyclic' modes of operation. The cyclic mode is also called as 'Auto-Reload' mode ie. once the counter reaches the 'Done' state (or completed its counting), it is automatically re-loaded with the initial count state to continue the counting process. In acyclic mode, the counter/timer waits for an explicit starting signal from the microprocessor to resume its counting.



Watchdog timer - A watchdog timer is an I/O device that is used for the internal operation of a computer system. As shown in fig, the watchdog timer output line is connected to the hardware 'Reset' pin/ line of the processor/CPU through the system bus. The CPU's software is programmed to reset the watchdog timer circuit, just before the count value of the watchdog timer reaches its 'maximum counting limit'. If the count reached the maximum counting limit without being reset by the CPU's software program assumes that either a software/ hardware problem has caused the CPU to get stuck (or halt) without resetting watchdog timer before it reached the maximum count limit. In this situation, the processor will get reset due to the watch dog timer's connection to reset pin/ line of the CPU. Here, it is assumed that either a software or hardware problem has caused the CPU to get stuck (or halt) without resetting its. Using a watch dog timer, the system is made to quickly reset to make it operational again.

A/D & D/A CONVERTERS

It is also known as ADCs & DACs that are used to interface 'Non-Digital Devices' to embedded systems. Since the analog-to-digital conversion requires complex circuitry, it requires more complex interface circuitry to the microprocessor. A/D conversion requires 'Sampling' of analog input signal to convert it to digital form. Using a control signal, the ADC takes a sample of analog signal & digitizes it.

There are several types of A/D converter circuits. Some of ADC circuits take a constant amount of conversion time for any sampled analog input signal value.

Variable-time A/D converters should provide a 'Done' signal to indicate the microprocessor that the digitally converted data value is ready to be read. An A/D interface has 'Analog Input' pins/ lines & two 'Digital Input' pins. The digital input lines are

(i) A data port in ADC that allows the converted data to be read by a micro-processor.

(ii) A 'clock' input of ADC that tells when to start the next conversion.

In D/A converters, the digital input data is continuously converted to its analog value. The D/A conversion process & the related circuits are relatively simple. So the D/A converter interface is not a complex interface

KEYBOARDS

A keyboard is basically an array of switches, but it may include some internal logic to help simplify the interface to the microprocessor. A switch uses a mechanical contact to make or break an electrical circuit. An ***encoded keyboard*** uses some code to represent which switch is currently being depressed. At the heart of the encoded keyboard is the scanned array of switches shown in Figure.

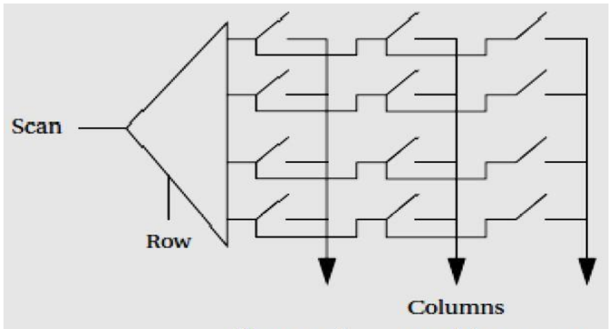
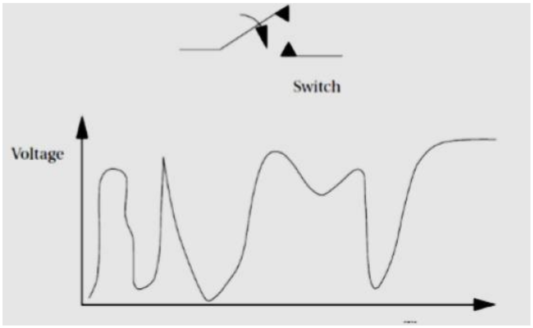


Figure: A scanned key array

The scanned keyboard array reads only one row of switches at a time. The de-multiplexer at the left side of the array selects the row to be read. When the scan input is 1, that value is transmitted to one terminal of each key in the row. If the switch is depressed, the 1 is sensed at that switch's column. Since only one switch in the column is activated, that value uniquely identifies a key. The row address and column output can be used for encoding, or circuitry can be used to give a different encoding.

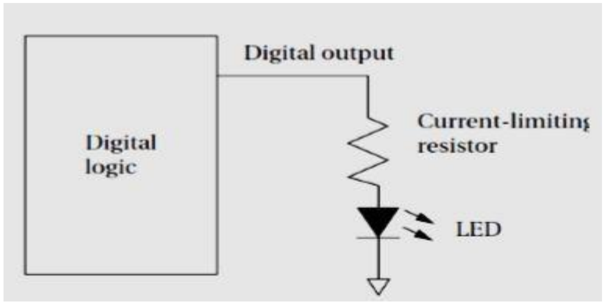
Key Bounce:-One major problem with mechanical switches is that they 'Bounce' when they are being pressed (as shown in fig). When the switch is pressed by pushing the button, the force of the press causes the contacts inside the switch to bounce several times until it get settled down .This is called 'Bouncing' problem' of a switch. If bouncing problem is not



corrected, it will appear that the switch has been pressed several times, thereby giving false inputs. A hardware circuit can be built using a 'timer' for removing the bouncing problem. Bouncing problem can be removed using software program coding also.

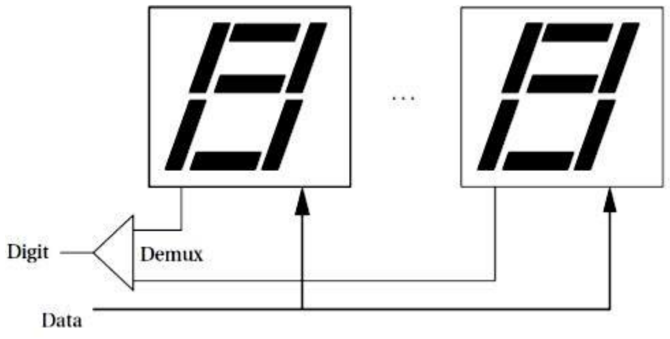
LEDs (Light Emitting Diodes)

LED is often used as a display for simple indication such as a 'power on status' Fig. shows how to connect an LED to a digital output. A current- limiting resistor is connected between the digital output pin & the LED. The presence of this resistor will create the voltage drop between the digital output voltage & the LED's on voltage of 0.7 volts (similar a forward biased junction). When the digital output goes to '0'(LOW), the effective voltage across LED is in its off region (similar to a reverse biased diode) which causes the LED is not to turn on.



DISPLAYS

A display device may be either directly driven or driven from a frame buffer. Typically, displays with a small number of elements are driven directly by logic, while large displays use a RAM frame buffer. The *n*-digit array, shown in Figure is a simple example of a display that is usually directly driven. Arrays of LEDs' can be used to make more complex displays. The n-digit array.



A 'Single-Digit Display' typically consists of 'Seven Segments'. Each segment may be either an LED or an LCD element. The digit input (as shown in fig.) is used to choose which digit position that is currently going to displayed. A 'digit' is displayed on the segment by selectively activating the LED or LCD elements of that segment based on the decoded data input value. The display's driver circuit is responsible for repeatedly scanning through the digits & presenting the current value of each to the display.

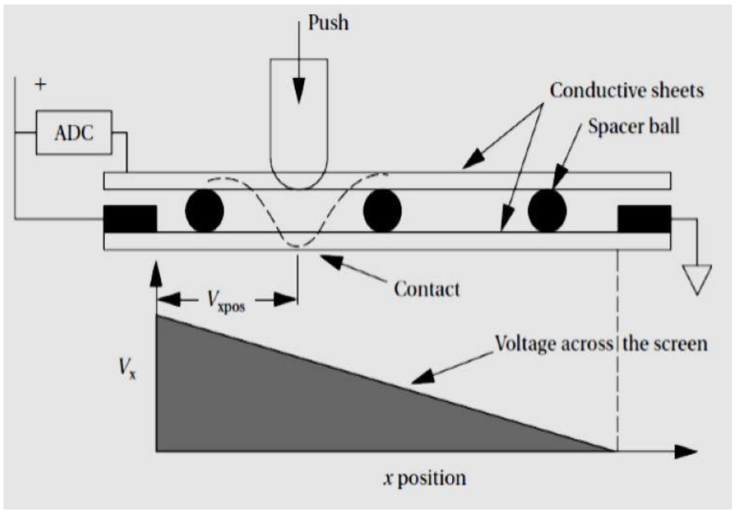
Frame Buffer - Displays with a small no. of elements can be driven directly by a decoder logic circuit. But large displays use a 'buffer memory' for storing consecutive 'display frames'. A 'Frame Buffer' is a simple & dedicated RAM that is attached to the system bus. The processor writes data into the frame buffer in the desired order of displaying i.e. The pixels in the frame buffer are written to the display in the standard order of sequentially reading the pixels .

LCDs (Liquid Crystal Displays)

Each pixel in the LCD display is formed by a single ‘Liquid Crystal’. Large displays are built using ‘array of liquid crystal pixels’. Interfacing an LCD displays to the computer system is very difficult because the array liquid crystal pixels cannot be randomly accessed. Old LCD panels were called a ‘Passive Matrix’ displays because they use a two dimensional grid of wires to access the pixels of an array. Modern LCD panels use an ‘Active Matrix’ system that puts a transistor at each pixel to control access to the pixels. Active matrix LCD displays provide a higher-quality display with a higher contrast

TOUCHSCREENS

A touch screen is an ‘input device overlaid on an output device’. The touch screen monitors & accounts the position of a touch to its surface. Since the touch screen input is overlaid on a display, the user can react to information shown on the touch screen display.



The two types of touch screens are:

(i) Resistive Type (ii) Capacitive Type

A resistive touch screen uses a two-dimensional voltmeter to sense touch position. The resistive touch screen consists of two conductive sheets separated by ‘spacer balls’. The top conductive sheet is flexible so that it can be pressed to touch the bottom sheet. A constant voltage is always applied across the length of the conducting sheet. The resistance across the length of the conducting sheet causes a ‘voltage gradient’ to appear across the length of the sheet (as shown in fig). A touch on the top sheet makes a contact between top & bottom sheets. The value of voltage gradient on the conductive sheet after a touch can be sampled though a ‘Contact Point’ is provided on the top sheet. An analog-to-digital converter is used to digitalize this sampled voltage. The position of the ‘touch’ can be sensed by measuring the results of the ADC.

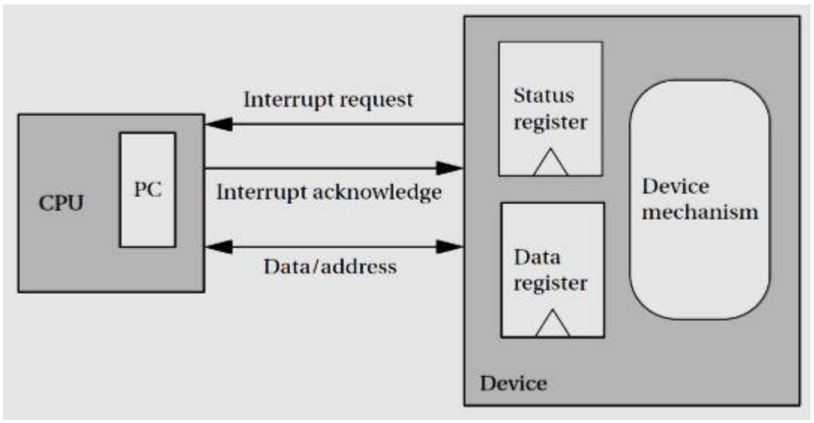
INTERRUPTS

The 'Interrupt Mechanism' allows devices to signal (or interrupt) the CPU & to force the CPU to execute a particular piece of code. When an interrupt occurs, the 'Program Counter' value is changed to point to an 'Interrupt Handler Routine' ((or an interrupt service routine, ISR) that takes care of the interrupting device. The interrupt mechanism saves the value of the PC at the instant of interruption so that the CPU can return to the program that was interrupted earlier. Interrupts allows the control flow in the CPU to change easily between different 'contexts'.

Interrupt Request : The device asserts the 'Interrupt Request Signal' when it wants service from the CPU . The logic circuitry in the device will decides when to interrupt the CPU by generating an interrupt request signal.

Interrupt Acknowledgement : The CPU asserts 'Interrupt Acknowledge Signal' when it is ready to handle the request from an interrupting device.

The CPU change the 'Program Counter (PC)' to point to the appropriate 'Interrupt Handler Program'



INTERRUPT SERVICE ROUTINE (ISR) (Interrupt Handler Program -)

ISR is a small program that the processor executes when the corresponding Interrupt is being requested. For every interrupt, there must be an interrupt service routine (ISR), When an interrupt occurs, the processor/microcontroller runs the interrupt service routine.

When an interrupt occurs

- Processor masks further external interrupts.
- Save the current context on stack
- Then executes the appropriate interrupt service routine (ISR).
- Upon return from the ISR restores the context and enables interrupts and return.

Interrupt handling mechanism

Steps in the 'Interrupt Handling Process

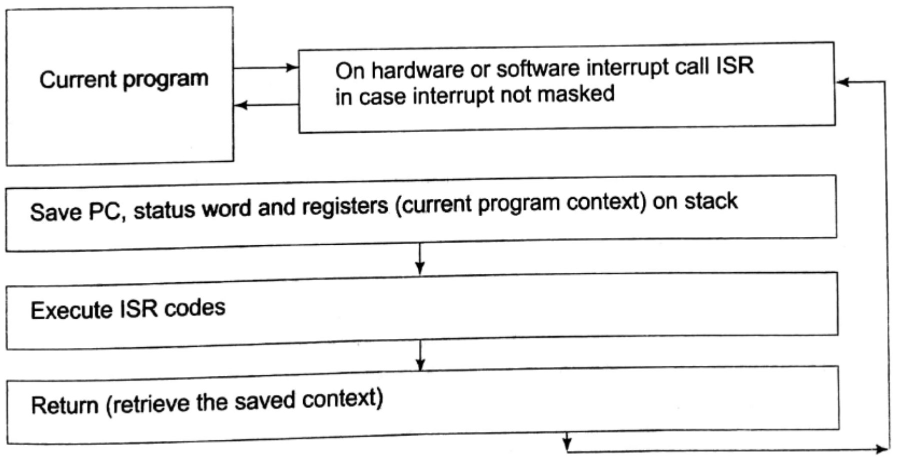
Step-1 (CPU): - The CPU checks for any pending interrupt requests just before the beginning of execution of every instruction in a program . If any interrupt requests are found in the 'Interrupt Request Register', CPU responds to the highest-priority interrupt (ie. The interrupt that has higher priority in the 'Interrupt Priority Register')

Step-2 (Device): - The device receives the acknowledgment signal from CPU . The device sends its interrupt 'vector number' to the CPU.

Step-3 (CPU): - The CPU looks up the interrupting device's 'Interrupt Handler Program Address' in the 'Interrupt Vector Table' using the vector no. as the pointer to the vector table . A 'sub-routine' like mechanism is used to save the current value of the Program Counter . The internal CPU state, such as the contents of general-purpose registers are also stored.

Step-4 (Software): -The device driver software program will save additional CPU states (or ISR program) . The device driver program performs the required operations on the interrupting device. The device driver also executes the 'Return from Interrupt' instruction.

Step-5 (CPU): - The 'Return from Interrupt' instruction re-stores the contents of the Program Counter & the other automatically saved CPU states to return execution to the program code that was interrupted earlier.


Current program
On hardware or software interrupt call ISR in case interrupt not masked
Save PC, status word and registers (current program context) on stack
Execute ISR codes
Return (retrieve the saved context)

Multiple Interrupts: (Non Nested ISR and Nested ISR)

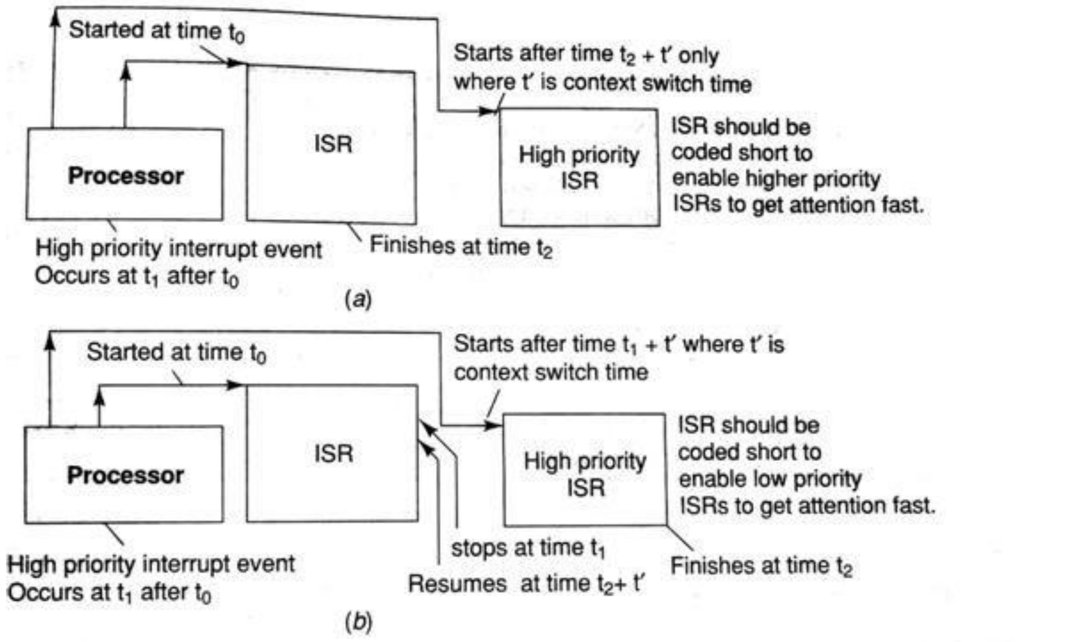
In a system with multiple interrupts, each and every interrupt may be assigned a

priority level. Here there is a possibility to occur multiple interrupts at a time. There are two types of interrupt service mechanisms for the case of multiple interrupts. Non Nested ISR and Nested ISR

Non Nested ISR:

In this case, Processors don't permit in-between routine diversion to higher priority interrupts. These processors provide auto disabling of all mask-able interrupt when ISR execution starts and auto re-enabling of all mask-able interrupt on return from ISR. These processors may also provide for auto saving the CPU registers (context) when ISR execution starts and auto return of saved values into the CPU registers on return from ISR. This help in fast transfer to pending higher priority interrupt.

Figure (a) shows multiple interrupts with diversion to higher priority interrupts only at the end of present ISR. Whenever an interrupt occurs the processor diverts the execution to the corresponding ISR (at time t_0). During the execution of this first ISR, suppose another interrupt with higher priority occurs (at time t_1)but the processor will continue the first ISR and finished it (at time t_2) then only execution will bediverted to the second ISR (at time t_2+t').


Started at time t_0
Starts after time $t_2 + t'$ only where t' is context switch time
ISR
Processor
High priority ISR
ISR should be coded short to enable higher priority ISRs to get attention fast.
High priority interrupt event Occurs at t_1 after t_0
Finishes at time t_2
(a)
Started at time t_0
Starts after time $t_1 + t'$ where t' is context switch time
ISR
Processor
High priority ISR
ISR should be coded short to enable low priority ISRs to get attention fast.
High priority interrupt event Occurs at t_1 after t_0
stops at time t_1
Resumes at time $t_2 + t'$
Finishes at time t_2
(b)

(a) Diversion to higher priority interrupts, only at the end of the present interrupt service routine (Non Nested ISR)
(b) In-between routine diversion to higher priority interrupts (Nested ISR)

Nested ISR:

In this case, processors permit in-between routine diversion to higher priority interrupts. Figure (b) shows multiple interrupts with diversion to higher priority interrupts in between ISR.

Whenever an interrupt occurs the processor diverts the execution to the corresponding ISR (at time t_0). During the execution of this first ISR, suppose another interrupt with higher priority occurs (at time t_1) then the processor will stop the first ISR (pending) and execution will be diverted to the second ISR (at time t_1+t'). After completion of second ISR (at time t_2) the pending ISR will be resumed at time t_2+t'. After finishing this first ISR, the execution will be returned to the main program.

These processors may also provide for auto saving the CPU registers (context) when ISR execution starts and auto return of saved values into the CPU registers on return from ISR. This help in fast transfer to higher priority interrupt

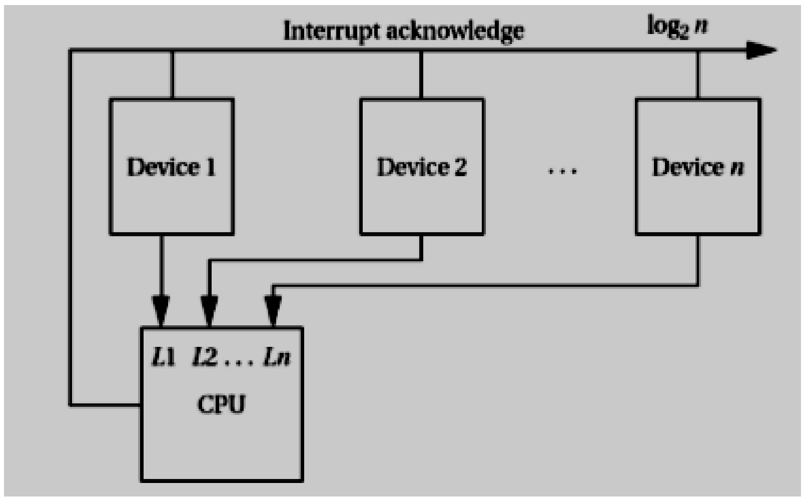
Context:- Whenever an interrupt occurs, processor stops the currently doing program and saves some values including program counter, stack pointer, status register, CPU registers etc before diverting the execution to the corresponding ISR. The saved data during the ISR call is known as context.

Interrupt Priorities & Interrupt Vectors

Computer systems usually have more than one device, so there must be some mechanism for allowing multiple devices to interrupt the CPU. There are two ways in which interrupt mechanism can be made to handle multiple devices & to provide flexibility in assigning the locations of ISRs . The two mechanisms are **'Priorities' & 'Vectors' .**'**Priorities**' determine which device is serviced first . '**Vectors'** determine what 'routine program' is used to service the interrupt.

INTERRUPT PRIORITIES

'Prioritized interrupts' mechanism allow the CPU to recognize some interrupts as more important that others ie. Prioritized interrupts allow multiple devices to interrupt the CPU simultaneously. Prioritized interrupts also allow the CPU to ignore less important interrupt requests to handle more important requests . Prioritized device interrupts are shown in fig.



The CPU provides several different interrupt request signals shown in fig. as 'L1', 'L2' … up to 'Ln''. Typically, the **'Lower-Numbered'** interrupt request lines are given '**Higher Priority'**. If the devices '1', '2' & 'n' all requested interrupts simultaneously, Device-1's request would be acknowledged because it is connected to the 'highest priority' interrupt line . Rather than providing a separate interrupt acknowledge line for each device, most CPUs use a set of acknowledgment signal lines (similar to a bus) that carry the winning interrupt's priority no. of in its binary form.

So a prioritized interrupt system with eight no. of devices requires a total of just 3 interrupt acknowledgment signal lines rather than 8 separate acknowledgment lines. The device can recognize that its interrupt request was accepted by seeing its own priority no. on the interrupt acknowledge signal lines (bus).

Most CPUs provide a relatively small no. of interrupt priority levels (up to eight levels) - More priority levels can be added using an external ‘Interrupt Controller’ module/ IC.

The interrupt priority of a device can be changed by simply connecting the device to a different interrupt request line with higher priority. Since such a connection change requires hardware modification, some other mechanism should be provided to make the system easily change interrupt priorities

Masking of interrupts

The interrupt priority mechanism must ensure that a ‘Lower-Priority Interrupt’ does not occur when a ‘Higher-Priority Interrupt’ is being handled . This process is known as ‘Masking the Interrupts’. When an interrupt is acknowledged, the priority level of that interrupt is stored in an internal register by the CPU. When a next subsequent interrupt is received, its priority is checked against the priority register. The new interrupt request is acknowledged only if it has higher priority than the currently pending interrupt.

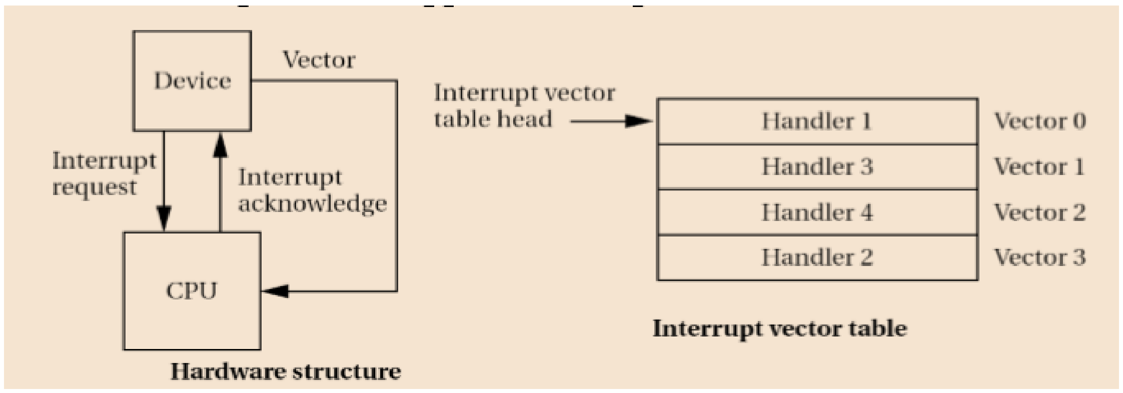
When the execution of the interrupt handler program is completed, the priority register must be reset. To ensure the resetting of the priority register, in most of the CPU architectures a specialized instruction will be available to return from interrupts Non-Maskable interrupt (NMI).

The highest-priority interrupt is normally called the ‘Non-Maskable Interrupt’. The NMI interrupts cannot be ‘masked’ or ‘turned-off’. NMI is usually reserved for interrupts caused by power failures (To handle a low power situation, a simple circuit can be used to detect a low power supply & the NMI interrupt handler program can be used to save critical variables/ states, turn-off I/O devices etc.)

INTERRUPT VECTORS

Interrupt vector mechanism allow the interrupting device to specify its handler program - Vectors provide flexibility to change the interrupt handler program that should service an interrupt request from a device - Fig. shows the hardware structure required to support interrupt vectors. Working of Interrupt Vector Mechanism - Here, extra signal lines known as the ‘interrupt vector lines’ run from the device’s side towards the CPU along with the ‘interrupt request & acknowledgement’ signal lines .

The CPU maintains a table called ‘interrupt Vector Table’ stored in the memory (as shown in the fig.). The ‘Interrupt Vector Table’ contains the starting address of different interrupt handler programs. After receiving an interrupt acknowledgement signal, the interrupting device will send its ‘Interrupt Vector Number’ through the vector lines towards the CPU. The CPU uses this vector no. as an index (or pointer) towards ‘Interrupt Vector Table’ to find the address of the interrupt handler program requested by the device.



Types of Interrupts

There are two types of interrupts: hardware interrupts and software interrupts.

1. Hardware Interrupt Sources:

Hardware sources can be internal or external for interrupt of ongoing routine and thereby diversion to corresponding ISR.

- **The internal sources** from devices differ in different processor or microcontroller or device and their versions and families
- **External sources** and ports also differ in different processors or microcontrollers

There are four possible sources of hardware interrupts as follows

a) Internal Hardware Device Sources

The Internal Hardware Device Sources are

1 Parallel Port

2. UART Serial Receiver Port - [Noise, Overrun, Frame-Error, IDLE, RDRF in 68HC11]

3. Synchronous Receiver byte Completion

4. UART Serial Transmit Port-Transmission Complete, [For example, TDRE (transmitter data register Empty]

5. Synchronous Transmission of byte completed

6. ADC Start of Conversion

7. ADC End of Conversion

8. Pulse-Accumulator overflow

9. Real Time Clock time-outs

10. Watchdog Timer Reset

11. Timer Overflow on time-out

12. Timer comparison with Output compare Registers

13. Timer capture on inputs

b) External Hardware interrupts with also sending vector address

- INTR in 8086 and 80x86 ─ The device provides the ISR Address or Vector Address or Type externally on data bus after interrupt at INTR pin

c) External Hardware Interrupts with Internal Vector Address Generation

Non-Maskable Pin─ NMI in 8086 and 80x86. Mask-able Pins (interrupt request pin) ─INT0 and INT 1 in 8051, IRQ in 68HC11.

d) Sources of interrupts due to Processor Hardware detecting Software error

Software sources for interrupt are related to processor detecting computational error during execution such as division by 0, illegal opcode or overflow (for example, multiplication of two numbers exceeding the limit)

1. Division by zero detection (or *trap*) by hardware
2. Over-flow detection by hardware
3. Under-flow detection by hardware
4. Illegal opcode detection by hardware

2. Software Interrupts(Exceptions & Traps)

- Software interrupts are triggered internally by some program instruction within the currently executing software by the master processor .

They are initiated by an event that is a result of a problem with the current instruction stream .

Basically , software interrupts are also referred to as 'Exceptions' or 'Traps'.

- **Exceptions** - Exceptions are internally generated software interrupts triggered by errors that are detected by the master processor during software execution such as illegal math operations like a 'Divide-by-Zero' condition ,'Invalid Op-codes'(Fetch & Decode of invalid instructions), Overflow after an arithmetic operation etc.
- **Traps** - Traps are software interrupts specifically generated via a assembly language instruction known as the 'interrupt instruction' of the master processor
- Eg.'SWI' instruction in 8085, 'INTn' instruction of 8086.

DEVICE DRIVERS

Most embedded hardware requires some type of software initialization and management. A device driver is a function used by a high-level language programmer, which does the interaction with the device hardware, sends control commands to the device, communicates data to the device and runs the codes for reading device data.

Device drivers are the software libraries that initialize the hardware, and manage access to the hardware by higher layers of software. Device drivers are the ‘Communication Facilitator’ program between the Hardware & the Operating System & Application Layers.

TYPES OF DEVICE DRIVERS –

Device drivers are typically considered either

(i) **Architecture-Specific Device Drivers -** A device driver that is architecture-specific manages the hardware that is integrated into the master processor’s architecture (On-chip hardware modules). Device drivers that initialize & enable components within a master processor such as ‘On-chip Memory’, on-chip memory managers (MMU) & different hardware modules such as Interrupt Controller etc.

(ii) **Generic Device Drivers** - A device driver that is generic manages hardware that is located on the board & not integrated onto the master processor’s on-chip architecture. The generic driver manages hardware units on an embedded board that is not specific to a particular processor

It can be configured to run on a variety of architectures that contain the related board hardware for which the driver is written. It include code that initializes & manages access to the major components of the embedded board, including board buses (I2C, PCI controller Modules/ ICs etc), ‘Off-chip Memory’ .

FUNCTIONALITIES OF DEVICE DRIVERS –

Regardless of the type of device driver or the hardware it manages, all device drivers are generally made up of some around ten basic functions. Each of the driver functions requires program code that interfaces directly to the hardware & to higher layers of software . Device driver code typically runs in supervisory mode (Software running in supervisory mode having more access privileges than software running in user mode).

The **ten basic functions** of device drivers are

(1) Hardware Startup : Initialization of the hardware upon power-on or reset

(2) Hardware Shutdown :Configuring hardware into its power-off state.

(3) Hardware Disable :Allowing other software programs to disable hardware.

(4) Hardware Enable :Allowing other software programs to enable hardware.

(5) Hardware Acquire : Allowing other software programs to gain access to hardware.

(6) Hardware Release : Allowing other software programs to free (unlock) hardware.

(7) Hardware Read : Allowing other software programs to read data from hardware.

(8) Hardware Write : Allowing other software programs to write data to hardware.

(9) Hardware Install : Allowing other software programs to install new hardware.

(10) Hardware Uninstall : Allowing other software programs to remove installed hardware.

1. Device Drivers for Interrupt-Handling

The software that handles interrupts on the master processor and manages interrupt hardware mechanisms (i.e., the interrupt controller) consists of the device drivers for interrupt-handling. These functions implemented in software usually depend on the following criteria

- The types, number, and priority levels of interrupts available (determined by the interrupt hardware mechanisms on-chip and on-board).
- How interrupts are triggered.
- The interrupt policies of components within the system that trigger interrupts, and the services provided by the master CPU processing the interrupts.

Following are some of the **device drivers for handling interrupt**.

Interrupt-Handling Start up: This driver is used for initialization of the interrupt hardware (i.e., interrupt controller, activating interrupts, etc.) upon power-on or reset.

Interrupt-Handling Shutdown:- This driver is used for configuring interrupt hardware (i.e., interrupt controller, deactivating interrupts, etc.) into its power-off state.

Interrupt-Handling Disable:- This driver is used for allowing other software to disable active interrupts on the fly (not allowed for Non-Maskable Interrupts (NMIs), which are interrupts that cannot be disabled).

Interrupt-Handling Enable:- This driver is used for allowing other software to enable inactive interrupts on-the fly.

Interrupt-Handler Servicing:- This driver is used for the interrupt-handling code itself, which is executed after the interruption of the main execution stream (this can range in complexity from a simple non-nested routine to nested and/or re entrant routines).

2. Memory Device Drivers

The software must provide the processors in the system with the ability to access various portions of the memory map. The software involved in managing the memory on the master processor and on the board, as well as managing memory hardware mechanisms, consists of the device drivers for the management of the overall memory subsystem.

Device drivers for the management of 'Memory Subsystem' -'Memory Subsystem' includes all types of memory management components, such as

-Memory Controller ICs/Modules -MMUs,

-Different types of memories in the memory map, such as CPU registers, cache, ROM, DRAM etc.

-The memory management software must provide ability to the master processor of the system to access various portions of the memory map. The memory management software should manage -The master processor's on-chip memories.

-The external memories on the embedded board -Memory hardware mechanisms

Eg:-.Memory Controllers, MMUs.

Following are some of the **device drivers for handling memory**

Memory Subsystem Start up :- This driver is used for initialization of the hardware upon power-on or reset (initialize TLBs for MMU, initialize/configure MMU).

Memory Subsystem Shutdown :- This driver is used for configuring hardware into its power-off state.

Memory Subsystem Disable :- This driver is used for allowing other software to disable hardware on-the-fly (disabling cache).

Memory Subsystem Enable :- This driver is used for allowing other software to enable hardware on-the-fly (enable cache).

Memory Subsystem Write :- This driver is used for storing in memory a byte or set of bytes (i.e., in cache, ROM, and main memory).

Memory Subsystem Read :- This driver is used for retrieving from memory a "copy" of the data in the form of a byte or set of bytes (i.e., in cache, ROM, and main memory).

3. On-board Bus Device Drivers

Every bus is associated with some *protocol* that defines

- How devices gain access to the bus.
- The rules must followed by the devices to communicate over the bus
- The signals associated with the bus lines. Bus protocol is supported by the bus device drivers

Following are some of the **device drivers for handling on board bus**

Bus Startup :- This driver is used for initialization of the bus upon power-on or reset.

Bus Shutdown :- This driver is used for configuring bus into its power-off state.

Bus Disable :- This driver is used for allowing other software to disable bus on-the-fly.

Bus Enable :- This driver is used for allowing other software to enable bus on-the-fly.

Bus Acquire :- This driver is used for allowing other software to gain singular (locking) access to bus.

Bus Release :- This driver is used for allowing other software to free (unlock) bus.

Bus Read :- This driver is used for allowing other software to read data from bus.

Bus Write :- This driver is used for allowing other software to write data to bus.

Bus Install :- This driver is used for allowing other software to install new bus device on-the-fly for expandable buses.

Bus Uninstall :- This driver is used for allowing other software to remove installed bus device on the- fly for expandable buses.

MODULE 4

MODULE IV

Syllabus

Programming concepts of Embedded programming – Features of Embedded C++ and EmbeddedJava (basics only). Software Implementation, Testing, Validation and debugging, system-on- chip. Design Examples: Mobile phones, ATM machine, Set top box

ADVANTAGES OF ASSEMBLY-LANGUAGE PROGRAMMING (ALP)

- Assembly instructions are specific to the processor, memory, ports & internal devices.
- It gives a 'precise control' of the hardware devices in the processor.
- Assembly code makes full use of the specific features of the 'Instruction Set' & the'Addressing Modes' of the processor.
- Assembly codes are 'Compact', 'Processor-Sensitive' & ' Memory-Sensitive'
- Thus, the system needs a smaller memory only for program storage .
- No additional memory is needed due to proper data size selection, conditions etc.
- Assembly program code is also not 'Compiler-Specific'& 'Library Function-Specific'.
- Program codes such as 'Device Driver' codes may need only a few assembly program instructions .
- For eg, Consider a small embedded system such as a timer device in a microwave oven or automatic washing machine or an automatic chocolate-vending machine .
- Assembly codes for the above timer can be made as compact & precise by convenientlywritten using assembly language coding.
- Bottom-up-Design' approach is easily usable for assembly language coding .
- It is an approach of designing program in which coding/ programming is first done for the basic functional modules for a specific set of actions .
- Then use these modules to build a bigger modules ie. Programs for 'Delay', 'Counting', finding 'Time Intervals' & many applications can be written or coded first .Then the finalprogram is designed by integrating these modules.

ADVANTAGES OF HIGH-LEVEL LANGUAGE PROGRAMMING

High-level language coding in C, C++, C#, Visual C++ or Java provides great programming ease & many advantages. Most of the embedded programming is done in the high-level language. High-level language coding makes the program development cycle short duration. It enables the use of the 'Modular-Programming' approach. It also facilitates the program development with 'Top-Down' design approach.

'Program-Development Cycle' in high-level language is of much shorter duration due to the followings:

(i) Use of Routines (called 'functions' in 'C/C++' & 'Methods' in Java).

(ii) Use of standard library functions.

(iii) Use of 'Modular Programming' approach.

(iv) 'Top-Down' design or 'Object-Oriented' design approaches.

Function - A 'Function' defines 'A set of Statements & Commands' or a 'Method of Operation' which runs when that function is called.

Library Functions - 'Library Functions' are standard functions, which are readily available to a programmer & the codes for them are not defined by programmer.

Eg., A 'square root( )' function . The use of the standard library function, square root( ), saves the programmer's time for coding .

New sets of library functions such as Delay( ), Wait ( ) & Sleep( ) exist in an embedded system-specific C or C++ compiler.

Modular Programming - 'Modular Programming' is an approach in which the building blocks of a software program are some other re-usable software components. A 'module' is built by software components & the components are built by a set of functions.

Eg., Making a 'Calculator' software using a 'Java' can be done as modular programmingusing the pre-built 'Java program modules' for calculator 'keys'.

Just as an IC (integrated circuit) which has several circuits integrated into one chip; similarlya module building block that may call several functions & library functions.

A module is tested for a well-defined goal and data inputs & outputs. It should not affect anydata other than the one it operates

Top-Down Design - 'Top-down' design is a programming approach used in high-level languages in which the main( ) program is first designed (or written), then its 'Modules', 'Sub- Modules' & finally the 'functions'.

(2) A High-level language program facilitates declarations of data types.

Data type declarations simplify the programming. For example there are four types of integers: int, unsigned int, short & long.

When dealing with positive only values only, we declare a variable as 'unsigned int' . In arithmetical calculations, we need a signed integer, int (32 bits) . An integer can also be declared as short (16 bits) or long (64 bits). Another data type is char is used to manipulate the text & strings for a character.

(3) High-level program facilitates 'Type Checking'

This makes the program less prone to error. For eg., The 'Type Checking' does not permit subtraction, multiplication & division on the 'Char' data types , but it permit 'plus' operator to used for 'concatenation' when using char data types . It permits the 'plus' operator to be used forarithmetic addition when using int, unsigned int, short & long type of data.

(1) The high-level language program facilitates use of **program-flow-control structures** such asloops & conditional statements (Eg., while, do-while, for, break, continue etc.)

(2) A 'high-level' language program code is **not 'Processor-Specific'**. Hence it is '**Portable**'.

Therefore, when the hardware changes, only the modules for the device drivers & device management, initialization etc. need modification . Operating System (OS) takes care of the above issues.

ADVANTAGES OF EMBEDDED 'C'

Embedded system programmers prefer 'C' for the following special features/reasons:

(i) Assembly codes can be embedded into C program codes using 'in-line assembling'.

(ii) Availability of special modules in 'C' compilers for the embedded systems .

(iii) Availability of library codes that can directly insert into the user program codes In-LineAssembly.

(iv) It is the possible facility of inserting assembly language codes in-between a high-levellanguage code .

(v) So, the complex part of the program can be written in high-level language & the assemblylanguage can be used for the code sections that need direct hardware control.

OBJECT-ORIENTED PROGRAMMING

'C++' is an 'Object-Oriented Programming' (OOP) language. An object-oriented language provides for the following:
(1) Defining of an 'Object' or 'Set of Objects', which are common or similar objects within aprogram & between many programs.
(2) Defining the methods/ functions that manipulate the objects without modifying their definitions.
(3) Creation of 'Multiple Instances' of the defined object or set of objects or new objects.
(4) Inheritance .
(5) Data Encapsulation .
(6) Design of re-usable components.

EMBEDDED PROGRAMMING IN C++ -

'Embedded C++' is a 'C++ version' which makes large program development simpler byproviding OOP features of using an object.

We use objects in a way that minimizes memory needs & run-time overheads in the system. Embedded system programmers use C++ due to the OOP features such as 'Software Re- Usability', 'Extendibility', 'Polymorphism', 'Function Over Riding' & 'Operator Overloading'.

C++ offers portability with the 'C' codes & 'Assembly' codes - C++ also provides overloadingof operators

ADVANTAGES OF 'C++'

Program coding in C++ codes provides the following advantage of OOP as well as the advantageof both 'C' & 'Assembly' language programming.

(1) Class & Objects.

(2) Inheritance.

(3) Method Overloading & Method over-riding.

(4) Operator Overloading.

(5) Polymorphism.

(1) Class & Objects

A 'class' binds all the member functions together for creating 'objects' - A class can be declared as public or private. The data access are restricted when a class is declared private.

The objects will have memory allocation as well as default assignments to its variables that are not declared 'Static'. Let us assume that a 'Software Timer' is an object . It gets the count input from a real-time clock (RTC).It has a 'Terminal Count Value' after which it generates a 'Software Interrupt' . It is initialized to a 'Initial Count' value.

Now consider the C++ codes for a 'class RTCSWT' - A no. of software timer objects can be created as the instances of 'RTCSWT'. Each object of RTCSWT can have different values of 'Present', 'Initial' & 'Terminal' count values -But it has identical methods to manipulate the count.

(2) Inheritance

A class can derive (inherit) from another class also - From the above example, Creating a 'Child Class' of RTCSWT as a 'Parent Class' creates a new application of the RTCSWT.

(3) Method Overloading & Method over-riding

'Methods' (or 'C functions') can have the same name in the inherited class. This is called 'Method Overloading' or 'Function Overloading'. 'Methods/ Functions' can have the same nameas well as the same no. & type of arguments in the inherited class . This is called 'Method Over- riding' or 'Function Over-Riding' . These are the two significant features that are extremely useful in a large program.

(4) Operator Overloading

Operators in C++ can be overloaded like method/function overloading. For example, usually the '++' operator is used for 'post-increment' & 'pre-increment' and the '!' operator is used for a not operation) . But using operator overloading, the'++' operator and '!' operator can be overloaded to perform a set of operations.

(5) Polymorphism

A 'C++' class has object features. It can be extended & 'Child Classes' can be derived from it. A no. of child classes can be derived from a common class . This feature is called 'Polymorphism'.

DISADVANTAGES OF 'C++'

The program codes becomes lengthy when certain features of the standard C++ are usedsuch as:

(1) Multiple Inheritances (Deriving a class from many parents).
(2) Exceptional Handling in C++ .
(3) Virtual base classes of C++.
(4) Template in C++.
(5) Classes for I/O streams (cin & cout).
    (i) The I/O stream library provides for the input & output streams of characters (bytes) .
    (ii) Two I/O stream library functions are 'cin' for 'character in' & 'cout' for 'char out'

Additional Features of Embedded C++

The special features in C++ for Embedded programming environment are :

(1) Micro-controller specific program coding features exits for handling the data at the 'Ports' of a micro-controller. Handling 'Special-Function Registers' of a micro-controller . Handling of interrupt functions (ISR). 'Bit Addressing' of bits in the registers & memory.

(2) Embedded C++ uses global variables for handling interrupts & ISRs.

(3) Special modifiers are used in embedded C++ for various cases.

Compiler for Embedded C++

'Embedded C++' is a version of 'C++' that provides for a selective disabling some of the features of 'C++' so that there is a less run-time delay & less library requirement. A special compiler for an embedded system can facilitate the above mentioned disabling of specific features provided in C++.

The library functions are available & they can be ported in C directly -The I/O stream library functions in an embedded C++ compiler are also re-entrant - Hence, using embedded C++ compilers or the special compilers make the 'C++' a significantly more powerful coding language than 'C' for embedded systems.

Code & Memory Optimization in 'Embedded C++ Program

Declare 'private' as many classes as possible which helps in code optimization. Use 'char', 'int' & 'boolean' in place of the objects as function arguments . Use local variables as much as possible instead of using global variables.

EMBEDDED PROGRAMMING IN JAVA

'Embedded Java' provides an embedded run-time environment. Java programming starts from coding for the classes. A class has members, a class contain many fields, a 'field' is like a variable or structure (struct) in C. A method defines the operations on the fields, similar to function in C. '**Class**' is a named set of codes that has a no. of members such as 'Data Fields' (variables)& 'Methods'.

Each **class** is a logical group with 'Identity', 'State' & 'Behaviour Specifications'. Java has large no. of readily available classes for 'I/Os', 'Networking', 'Security' etc. Object-oriented features & ready availability of classes make a large program development simpler task using Java. Class is used to create objects with instances of these members

'Instance Fields' & 'Instance Methods' of class are the members, whose new instances are also created as when the objects are created from the class. The operations are performed on the objects by passing the messages to objects in object-oriented programming.

Java Program Element and Explanation

Class:- A class is a basic structural unit in a Java program. A class consists of data fields and methods that operate on the fields. A class defines a group of objects with similar attributes and common behaviour and relationships. A class is used to create objects as its instances. It has instance and static fields and methods.

Inheritance :- Java class inherits members when a Java class is extended from a parent class called super class. The inherited instance fields and methods can be over-ridden by redefining them in extended class using same name, arguments and argument-types. Methods can be overloaded by redefining them for different numbers or types of arguments.

Local variable :- A variable within a block of codes is defined inside the curly braces and has limited scope

Instance method :- Blocks of Java codes, which are given a name, a call (invocation) is made by other Java codes that can also pass (transmit) the needed reference to the values, parameters, and so on.

Instance field :- An identifier with a name and using that name a declaration is made in a Java class. It does have a default value and the field is also present in the objects which areinstances of the class

Interface:- Interface has only the abstract methods and the corresponding static data fields and the methods do not, have implementation in the interface. A Java class which is interfaced to an interface implements the abstract methods specified at the interface

Data types :- Java class uses primitive data types: byte (8-bit), short (16-bit), int (32-bit), long (64-bit), float, double, unicode char (16-bit). Java class uses reference data types. A reference can be to the class type in which there are groups of fields and methods to operate on the fields. A reference can be to the array type in which there are groups ofobjects as array elements.

Exception :- Java has built-in exception classes. The occurrences of exceptional conditions are handled when exception is thrown. It is also possible to define exception conditions ina program so that exceptions are thrown from try block codes and caught by catch exception method

ADVANTAGES OF JAVA PROGRAMMING

Java has advantages for embedded programming as follows

(1) Java is completely an OOP language. It starts with classes .Application program consists of classes, objects & interfaces.

(2) There is a huge library of pre-built classes is available in Java platform which makes programdevelopment quick.

(3) Java has in-built support for creating multiple ‘Threads’ . It obviates the need for an operating system (OS) based scheduler for handling threads.

(4) After compiling, a Java program generates ‘Java Byte Codes’ (similar to HEX files).These ‘Java byte codes’ are executed on Java Virtual Machine (JVM) software module installed on a computing machine/ platform. JVM takes the ‘Java byte codes’ as its input & runs on the any platform defined by the processor & Operating System (OS). In embedded systems, the JVM can be stored at the ROM . Therefore ‘Java byte codes’ can run on different types platforms . ‘Platform Independence’ in running the compiled codes permit Java for network applications.

(5) Platform independence of Java gives ‘code portability’ feature with respect to the

processor & OS used ie. A Java program once written can run anywhere (or any processor platform/ any OS).

(6) Java is the language for most 'Web Applications'. Java allows computing machines of different types to communicate on the web.

(7) Java is easier to learn by a C++ programmer.

(8) Java does not permit multiple inheritances. It does not permit operator overloading except for the 'plus' sign which is used for string concatenation as well as arithmetic addition. Java does not permit dual way of object manipulation by value & reference - Unlike C/C++, There are no 'struct', 'enum', 'typedef' etc.

(9) Java has code extensibility.

(10) Java does not permit pointer manipulation instructions . So Java is 'Robust' in the sense that memory leaks & memory-related errors do not occur . A memory leak occurs, for example, whenattempting to write the end of a bounded array.

DISADVANTAGES OF JAVA PROGRAMMING

Java has following disadvantages for embedded programming as follows:

(1) Since Java codes are executed by the JVM, it runs comparatively slowly. This disadvantage can be overcome as follows:

✓ Java byte codes can be converted to native machine codes for fast running using Just-In- Time (JIT) compilation .

✓ A 'Java accelerator co-processor' can be used in the system, for the faster running of Java codes.

(2) Java byte codes that are generated need a larger memory . An embedded Java system may need a minimum of 512 kB ROM & 512 kB RAM because of the need to first install JVM software & run the application.

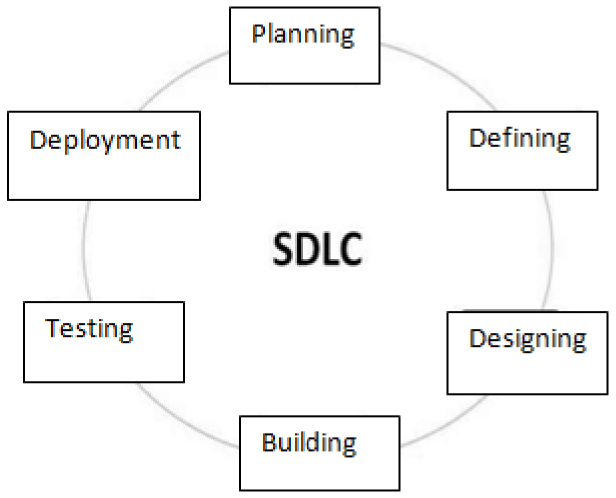
SOFTWARE IMPLEMENTATION

Software Development Life Cycle (SDLC) is a process used in the software industry to design, develop & test high quality software

<u>SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)</u>

It is also called as 'Software Development Process'. SDLC is a framework defining tasks performed at each step in the software development process. 'ISO/ IEC 12207' is an international standard for software life-cycle processes It aims to be the standard that defines all the tasks required for developing & maintaining software .

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace & alter or enhance specific software. The life cycle defines a methodology for improving the quality of software & the overall development process.



(1) Planning & Requirement Analysis **:-**

Requirement analysis is the most important & fundamental stage in SDLC . It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys & domain experts in the industry . This info. is then used to plan the basic project approach & to conduct product feasibility study in the economical, operational & technical areas.

Planning for the quality assurance requirements & identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various 'Technical Approaches' that can be followed to implement the project successfully with minimum risks.

(2) Defining Requirements –

Once the requirement analysis is done the next step is to clearly define & document the product requirements & get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed & developed during the project life cycle.

(3) Designing the Product Architecture –

SRS is the reference for product architects to come out with the best architecture for the product to be developed . Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed & documented in a DDS (Design Document Specification)

This DDS is reviewed by all the important stakeholders & based on various parameters as'Risk Assessment', 'Product Robustness', 'Design Modularity', 'Budget', 'Time Constraints' etc. for selecting the best 'Design Approach' for the product .

A design approach clearly defines all the architectural modules of the product along with its communication & data flow representation with the external & third party modules (if any) - The internal design of all the modules of the proposed architecture should be clearly defined withthe minutest of the details in DDS

(4) Building or Developing the Product

In this stage of SDLC the actual development starts & the product is built - The programming code is generated as per DDS during this stage - If the design is performed in a detailed & organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization & programming tools like compilers, interpreters, debuggers, etc. are used to generate the code

Different high level programming languages such as C, C++, C#, Pascal, Java & PHP are used for coding - The programming language is chosen with respect to the type of software beingdeveloped

(5) Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testingactivities are mostly involved in all the stages of SDLC. However, this stage refers to the testing .only stage of the product where product defects are reported, tracked, fixed & re-tested, until theproduct reaches the quality standards defined in the SRS .

(6) Deployment in the Market & Maintenance –

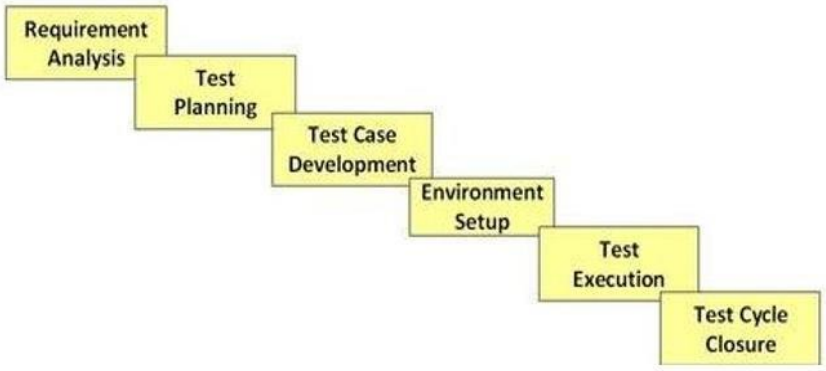
Once the product is tested & ready to be deployed it is released formally in the appropriate market . Sometimes product deployment happens in stages as per the business strategy of that organization . The product may first be released in a limited segment & tested in the real business environment (UAT- User Acceptance Testing).

EMBEDDED SOFTWARE TESTING

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not . In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

STLC : Software Testing Life Cycle

Software Testing Life Cycle (STLC) is defined as a sequence of activities conducted to perform Software Testing . It consists of series of activities carried out methodologically to help certify your software product.



Each of these stages have a definite 'Entry & Exit Criteria', 'Activities' & 'Deliverables' associated with it.

Entry Criteria : Entry Criteria gives the prerequisite items that must be completed before testingcan begin.

Exit Criteria : Exit Criteria defines the items that must be completed before testing can beconcluded.

(1) Requirement Analysis -

During this phase, test team studies the requirements from a testing point of view to identify the testable requirements . The QA (Quality Assurance) team may interact with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) to understand the requirements in detail. Requirements could be either

(i) Functional Requirements (Defining what the software must do).

(ii) Non-Functional Requirements (Defining system performance / Security availability) 'Automation feasibility' for the given testing project is also done in this stage.

Steps taken - Identify types of tests to be performed.
-Gather details about testing priorities & focus.
- Prepare 'Requirement Traceability Matrix' (RTM).
- Identify test environment details where testing is supposed to be carried out .
- Automation feasibility analysis (if required).

(2) Test Planning –

Typically , in this stage, a Senior QA (Quality Assurance) manager will determine effort & cost estimates for the project & would prepare and finalize the 'Test Plan'. .In this phase, Test Strategy is also determined.

Steps taken - Preparation of test plan/ strategy document for various types of testing.
- 'Test Tool' Selection.
- Test effort estimation.
- Resource planning & Determining roles and responsibilities.

- Training Requirement.

(3) Test Case Development –

This phase involves creation, verification & rework of test cases and test scripts . Testdata, is identified/created & is reviewed & then reworked as well.

Steps taken - Create test cases, automation scripts (if applicable).
- Review & baseline test cases and scripts.
- Create test data (If Test Environment is available)

(4) Test Environment Setup –

Test environment decides the software & hardware conditions under which a work product is tested. Test environment set-up is one of the critical aspects of testing process & it can be done in parallel with 'Test Case Development Stage'. Test team may not be involved in this activity if the customer/development team provides the test environment in which case the test team is required to do a readiness check (smoke testing) of the given environment

Steps taken - Understand the required architecture, environment set-up & Prepare hardware and software requirement list for the Test Environment.
- Setup test Environment & test data - Perform smoke test on the build.

(5) Test Execution –

During this phase the testers will carry out the testing based on the test plans & the test cases prepared. Bugs will be reported back to the development team for correction & re-testing will be performed.

Steps taken - Execute tests as per plan.
- Document test results & Log defects for failed cases.
- Map defects to test cases in RTM.
- Retest the Defect fixes.
- Track the defects to closure .

(6) Test Cycle Closure –

Testing team will meet , discuss & analyze testing artifacts to identify strategies that haveto be implemented in future, taking lessons from the current test cycle . The idea is to remove theprocess bottlenecks for future test cycles & share best practices for any similar projects in future. Steps taken - Evaluate cycle completion criteria based on Time, Test coverage, Cost, Software,
Critical Business Objectives , Quality.
- Prepare test metrics based on the above parameters.
- Document the learning out of the project.
- Prepare Test closure report.
- Qualitative & quantitative reporting of quality of the work product to the customer.
- Test result analysis to find out the defect distribution by type & severity.

Types of Testing

1) Black-Box Testing - The technique of testing without having any knowledge of the interiorworkings of the application is called black-box testing.

➢ The tester is oblivious to the system architecture & does not have access to the source code(or program) .

➢ Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs & examining outputs without knowing how & where the inputs are worked upon.

Advantages of Black-Box Testing

1) Well suited & efficient for large code segments
2) Code access is not required
3) Clearly separates user's perspective from the developer's perspective through visiblydefined roles
4) Large no. of moderately skilled testers can test the application with no knowledge ofimplementation, programming language, or operating systems

Disadvantages of Black-Box Testing

1) Limited coverage, since only a selected no. of test scenarios is actually performed
2) Inefficient testing, due to the fact that the tester only has limited knowledge about anapplication
3) Blind coverage, since the tester cannot target specific code segments or error-prone areas
4) The test cases are difficult to design

***2)* White-Box Testing** - White-box testing is the detailed investigation of internal logic &structure of the code . White-box testing *is also called 'Glass Testing' or 'Open-Box Testing'.*

➢ In order to perform white-box testing on an application, a tester needs to know theinternal workings of the code.

➢ The tester needs to have a look inside the source code & Find-out which unit/chunk ofthe code is behaving inappropriately.

Advantages of White-Box Testing

1) As the tester has knowledge of the source code, it becomes very easy to find out whichtype of data can help in testing the application effectively
2) It helps in optimizing the code
3) Extra lines of code can be removed which can bring in hidden defects
4) Due to the tester's knowledge about the code, maximum coverage is attained during testscenario writing

Disadvantages of White-Box Testing

1) Due to the fact that a skilled tester is needed to perform white-box testing, the costs areincreased
2) Sometimes it is impossible to look into every nook and corner to find out hidden errorsthat may create problem, as many paths will go untested
3) It is difficult to maintain white-box testing, as it requires specialized tools like codeanalyzers & debugging tools

3) Grey-Box Testing - Grey-box testing is a technique to test the application with having alimited knowledge of the internal workings of an application

➢ In software testing, the phrase the more you know, the better carries a lot of weightwhile testing an application.

➢ Mastering the domain of a system always gives the tester an edge over someone withlimited domain knowledge.

➢ Unlike black-box testing, where the tester only tests the application's user interface; ingrey-box testing, the tester has access to design documents and the database.

➢ Having this knowledge, a tester can prepare better test data and test scenarios whilemaking a test plan.

Advantages of Grey-Box Testing

1) Offers combined benefits of black-box & white-box testing wherever possible
2) Grey box testers don't rely on the source code; instead they rely on interface definitionand functional specifications
3) Based on the limited information available, a grey-box tester can design excellent testscenarios especially around communication protocols and data type handling
4) The test is done from the point of view of the user & not the designer

Disadvantages of Grey-Box Testing

I) since the access to source code is not available, the ability to go over the code and test coverage is limited
2) The tests can be redundant if the software designer has already run a test case
3) Testing every possible input stream is unrealistic because it would take an unreasonableamount of time; therefore, many program paths will go untested

Comparison of Black-Box testing, Grey-Box Testing & White-Box Testing

Black-Box Testing	**White-Box Testing**	**Grey-Box Testing**
The internal workings of an application need not be known	The tester has limited knowledge of the internal workings of the application	Tester has full knowledge of the internal workings of theapplication
Also known as closed- box testing, data-driven testing, orfunctional testing	Also known as translucenttesting, as the tester has limited knowledge of theinsides of the application	Also known as clear-box testing, structural testing, orcode-based testing
Performed by end-users and also by testers & developers	Performed by end-users and also by testers & developers	Normally done by testers & developers

Testing is based on external expectations; Internal behaviorof the application is unknown	Testing is done on the basis of high-level database diagram &data flow diagram	Internal workings are fully known & the tester can designtest data accordingly
It is exhaustive & the least time-consuming	Partly time-consuming & exhaustive	The most exhaustive & time-consuming type of testing
Not suited for algorithm testing	Not suited for algorithm testing	Suited for algorithm testing
This can only be done by trial& error method	Data domains & internal boundaries can be tested, if known	Data domains & internal boundaries can be better tested

DEBUGGING

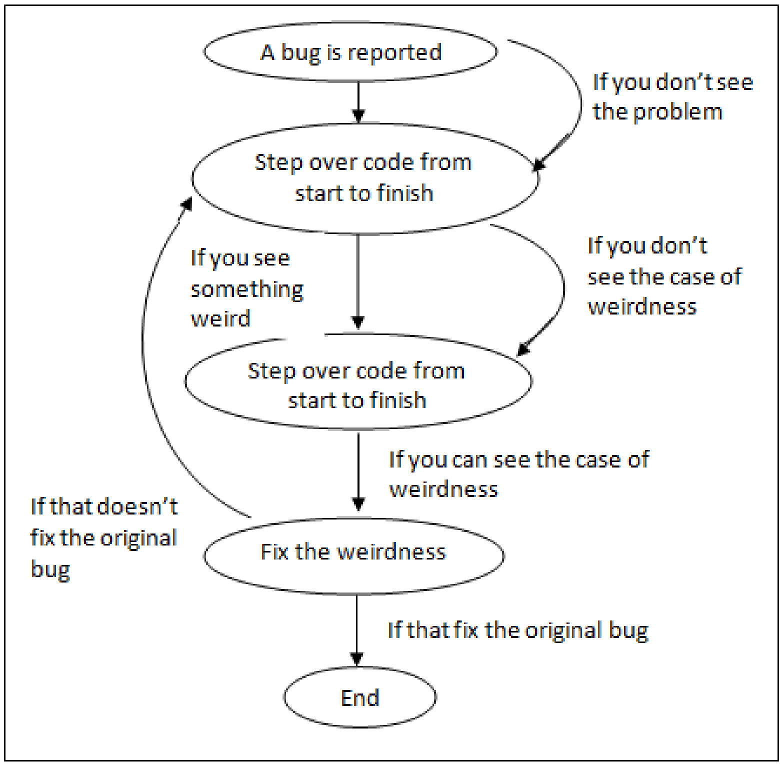
Debugging is a systematic process of spotting & fixing the no. of bugs (or defects), in a piece of software so that the software is behaving as expected. It is harder for 'complex systems' in particular when various sub-systems are tightly coupled as changes in one system or interface may cause bugs to emerge in another.

Debugging is a developer activity. Effective debugging is very important before testing begins to increase the quality of the system.

In general, **three categories** for debugging approaches may be proposed.

(i) **Brute Force** - The brute force category of debugging is probably the most common & efficient method for isolating the cause of a software error.

- Brute force debugging methods are applied when all methods of debugging fail.
- Using a philosophy, memory dumps are taken, run time traces are invoked & the programis loaded with write statement.
- When this is done, one finds a clue by the information which leads to cause of an error.


A bug is reported
If you don't see the problem
Step over code from start to finish
If you see something weird
If you don't see the case of weirdness
Step over code from start to finish
If you can see the case of weirdness
If that doesn't fix the original bug
Fix the weirdness
If that fix the original bug
End

(ii) **Back Tracking Method** - It is a popular approach of debugging which is used effectively in case of small applications.

• The process starts from the site where a particular symptom gets detected, from there on backward tracing is done across the entire source code till we are able to lay our hands on the site being the cause.

• Unfortunately, as the no. of source lines increases, the no. of potential backward paths may become unmanageably large.

(iii) **Cause Elimination 'Brute Force' Approach** - The cause elimination approach to debugging, is manifested by induction (or deduction).

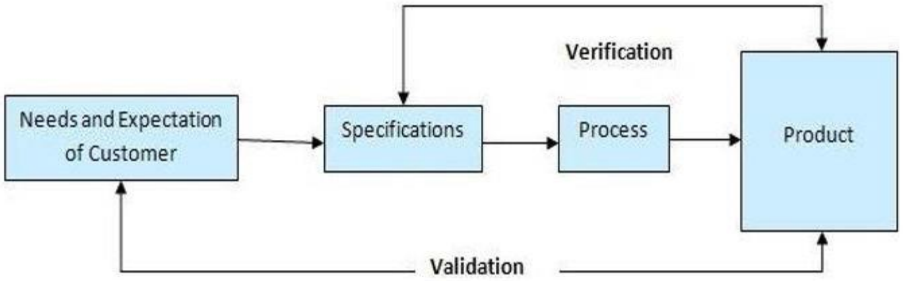
- This approach is also called 'Induction & Deduction
- Data related to the error occurrence are organized to isolate potential causes
- A "cause hypothesis" is devised & the data are used to prove or disprove the hypothesis.
- Alternatively, a list of all possible causes is developed & tests are conducted to eliminateeach.
- • If initial tests indicate that a particular cause hypothesis shows promise, the data arerefined in an attempt to isolate the bug.

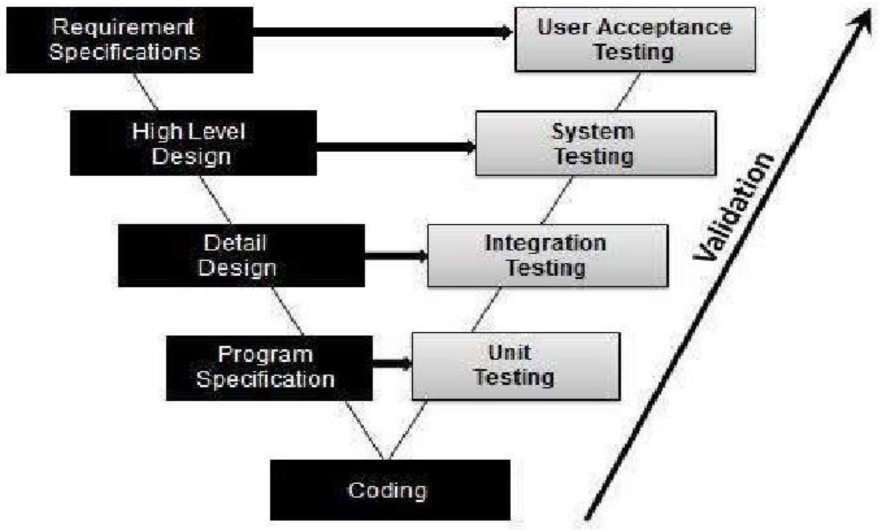
VALIDATION

The process of evaluating software during the development process or at the end of thedevelopment process to determine whether it satisfies specified business requirements.

'Validation Testing' ensures that the product actually meets the client's needs.

It can also be defined as to demonstrate that the product fulfills its intended use whendeployed on appropriate environment





Unit Testing

A level of the software testing process where individual units of software are tested. The purpose is to validate that each unit of the software performs as designed

Integration Testing.

A level of the software testing process where individual units are combined & tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units.

System Testing

A level of the software testing process where a complete, integrated system is tested. Thepurpose of this test is to evaluate the system's compliance with the specified requirements **Acceptance Testing**

A level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business requirements and assess whether it is acceptable for delivery

SYSTEM-ON-CHIP (SoC)

SoC is a complete embedded system on a single chip. Usually includes programmable processors, memory, accelerating function units, input/output interfaces, software, re-usable intellectual property. It is an integrated circuit that integrates all components of a computer or other electronic systems. It may contain digital, analog, mixed signals, radio frequency functions all are embedded in a single substrate. Soc integrates a microcontroller with advanced peripherals like graphics processing unit, Wi-Fi module, or co-processor

There are **two approaches to design** an embedded system such as

I. Using System on Chip (**SoC)**

2. Using chips on a circuit Board (**COB)**

System-on-chip approach is used for extensively used systems, for example mobile phoneand cameras.

In chips on a circuit board approach, separate chips for the microprocessor or ASIP, memory, single-purpose processors and peripherals are interconnected on a board. All chips interface with an appropriate circuit. The chips interconnect on a board and software embeds into memory in a chip.

Types of SoC - In general, there are three distinguishable types of SoCs.

(1) SoCs built around a microcontroller.

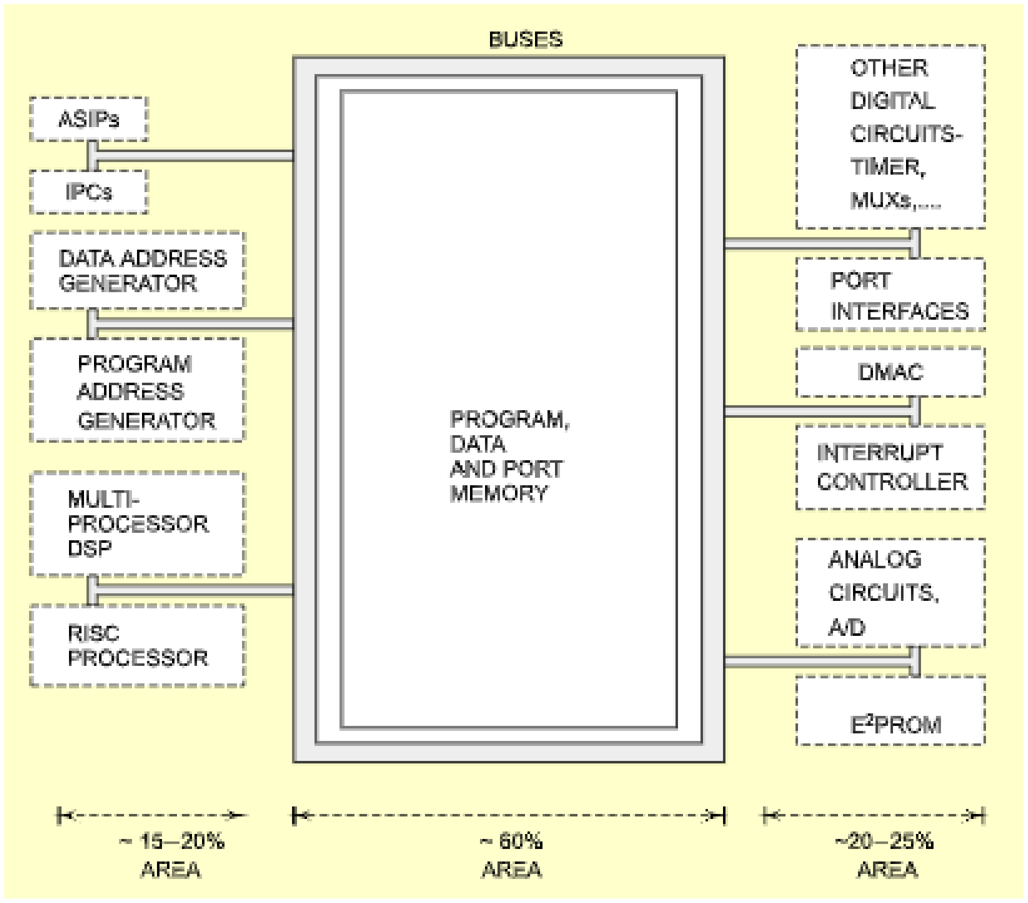
(2) SoCs built around a microprocessor.

(3) Specialized SoCs designed for specific **applications that do not fit into the above twocategories**

A separate category may be Programmable SoC where some of the internal elements are
not predefined and can be programmable in a manner analogous to a field-programmable gatearray *(FPGA)*or a complex programmable logic device *(CPLD).*

A typical SoC consists of a

- Microcontroller, microprocessor or digital signal processor (DSP) core – multiprocessorsocs (mpsoc) having more than one processor core.
- Memory blocks including a selection of ROM, RAM, EEPROM and flash memory.
- Timing sources including oscillators and phase-locked loops.
- Peripherals including counter-timers, real-time timers & power-on reset generators. External interfaces, including industry standards such as USB, firewire, Ethernet, USART, SPI.
- Analog interfaces including ADCs and DACs. Voltage regulators and power management circuits



A bus – either proprietary or industry-standard such as the AMBA bus from ARM Holdings – connects these blocks. DMA controllers route data directly between external interfaces and memory, bypassing the processor core and thereby increasing the data throughput of the SoC

A SoC consists of both the hardware, described above, and the software controlling the microcontroller, microprocessor or DSP cores, peripherals and interfaces.

The **design flow** for a SoC aims to develop this hardware and software in parallel. Most SoCs are developed from pre-qualified hardware blocks for the hardware elements described above, together with the software drivers that control their operation. Have a particular importance are the protocol stacks that drive industry-standard interfaces like USB.

The hardware blocks are put together using CAD tools; the software modules are integrated using a software-development environment.

➢ Once the architecture of the SoC has been defined, any new hardware elements are written inan abstract language termed RTL which defines the circuit behavior

➢ These elements are connected together in the same RTL language to create the full SoCdesign.

Advantage of SOC

- The System has a very high performance, functionalities and very low power dissipation.
- Therefore, SoCs are used in Embedded systems, which are extensively and commonly used, need the small size, high system performance, much functionality, very low power dissipation and have low energy consumption.
- Examples of extensively used systems are mobile Phones. Tablets, computers, Set-top boxes, digital TVs and cameras.

SoC Challenges

o More complex, more functions, higher gate counts, faster , cheaper, smaller, time to
market

o To handle these challenges the system should design at multiple abstraction levels,
integration of heterogeneous technologies & tools, signal integrity and timing, power management, soc test methodology

Soc design: IP Based Design

o Intellectual property cores:

o Parametrized components with standard interfaces facilitating high level synthesis.

Available in three forms

➢ Hard: black –box in optimized layout form and encrypted simulation model. Eg- Microprocessor .

➢ Firm: synthesized netlist which can be simulated and changed if needed.

➢ Soft: Register transfer level – user is responsible for synthesis and layout.

CASE STUDY OF MOBILE-PHONES

Mobile phones are smart and each has many APIs. Examples are phones SMS (Short Message Service), MMS (Multimedia Messaging Service) e-mail, addles book, web browsing, calendar, task-to-do list, WordPad, pocket-Excel, pocket word, note-pad for memos, pocket-PPTs, slide shows, and camera.

Mobile phones with large touch scream use a virtual keypad and small screens use a

T9 Keypad. Here we discuss about case study relates to "SMS create Application" in a mobile phone.

REQUIREMENTS

- A processor, keypad, screen, scratch pad memory, persistence memory andcommunication units are required for SMS create and send application.
- Scratch pad memory addresses are used f or temporary saving of character (bytes) during an application.
- A Persistence memory addresses are used such that as soon as a change is made in the byte. it persists even after the power switch off.

When there is a change, the identical change is reflected in other correlated objects. For example, if a name is edited in a file for the contacts, the same change takes in the file for Address Book for sending the e-mails.

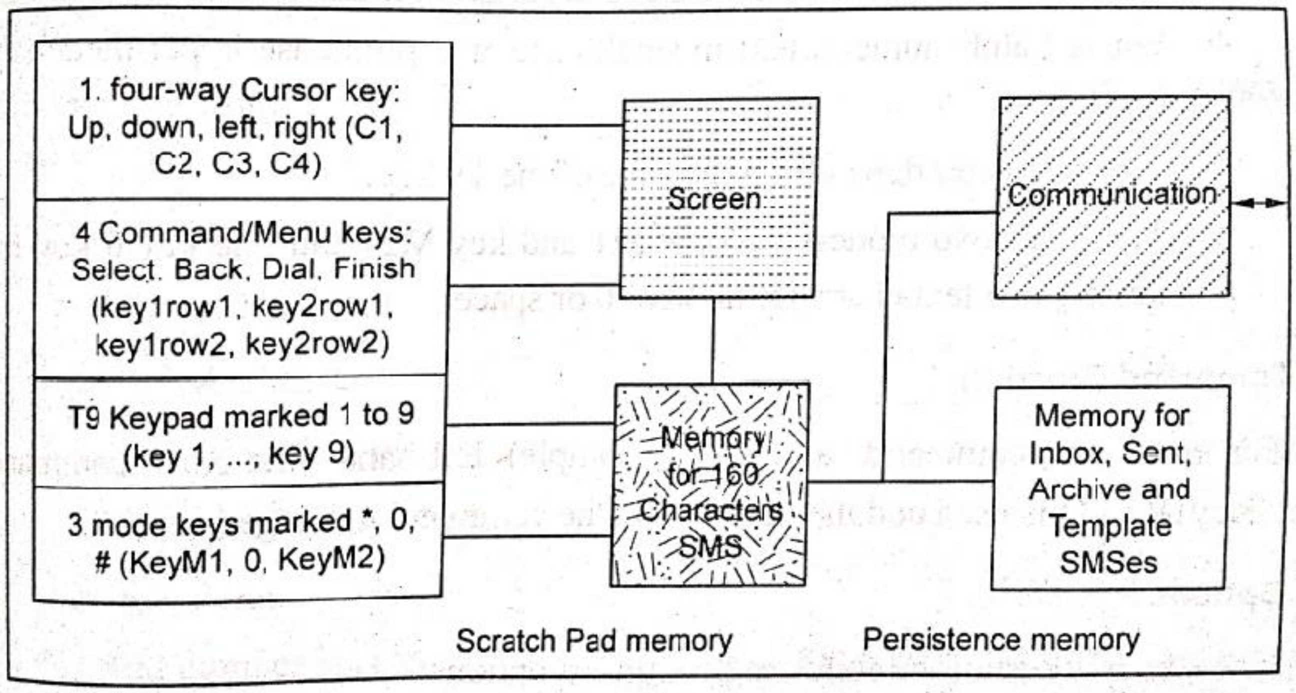

1. four-way Cursor key: Up, down, left, right (C1, C2, C3, C4)
4 Command/Menu keys: Select, Back, Dial, Finish (key1row1, key2row1, key1row2, key2row2)
T9 Keypad marked 1 to 9 (key 1, ..., key 9)
3 mode keys marked *, 0, # (KeyM1, 0, KeyM2)
Screen
Communication
Memory for 160 Characters SMS
Memory for Inbox, Sent, Archive and Template SMSes
Scratch Pad memory
Persistence memory

Fig. 4.10. Keypad, screen, memory and communication units

Cursor key

The screen is used for displaying mends for GUIs. There are four cursor keys

I. Up – CI
2. Down - C2
3. Left - C3
4. Right - C4

When the cursor key is pressed towards left, it moves the cursor left, Right moves thecursor Right, towards up moves the cursor up or towards down to move down.

The same key is used for movements of the cursor. In a computer keyboard, four different cursor keys are used.

Command/Menu Keys

- There are four commend keys (Right-corner second-Row, Left-corner Second-Row, Right-corner first-Row and Left-corner first-Row) denoted by Key2 Row2, Key1 Row2, Key2 Row1 and Key1 Row1
- There are nine T9 keys for nine numbers 1 to 9. well as alphabets a to z (or A to Z).
- Entered alphanumeric text in small case or capital case is per mode-key state
- Text character depends on the state of the T9 key.
- There are two mode-keys (key MI and key M2) and one key 0 key for keying in a textcharacter number 0 or space.

Command Function

For messages command, a key (for example) left-hand first-Row command key (Key1Row1) is used and the user selects the command messages.

Options

The cursor at one of the following command options

I. Text messages

2. Voice message and

3. Mini browser messages.

There are certain application-options are displayed at the displayed command option such as

I. Create message

2. Inbox

3 Sent items

4. Templates

5. My folders

6. Distribution Lifts

7. Delete messages and

8. Message settings

SMS Create Application

For selecting SMS create application option, the create message application option is selected as the application using cursor and click. There are two types of messages can be created such as Text and Numeric page.

Tasks

We can select the task option using the cursor at one of the following task options

I. Add Number

2. Add E-mail

3. Add List

4. Edit Message

5. List Recipient and

6. Send.

Add number is selected as the task option. On Return from the task Add Number, a newtask option Edit Message is selected

On Return from task Edit message, the neat task option send is selected.

If the message is sent to two numbers, then before send, a number is added using the taskAdd Number

MODULE 5

Subject : EC 308 EMBEDDED SYSTEMS
Module 5

Syllabus
Inter Process Communication and Synchronization -Process, tasks and threads –Shared data– Inter process communication - Signals – Semaphore – Message Queues – Mailboxes Pipes – Sockets – Remote Procedure Calls (RPCs).

<u>Process</u>

<u>Process Features</u>

- A process consists of executable program (codes), *state* of which is controlled by OS, the *state* during running of a process represented by process-status (running, blocked, or finished), process structure—its data, objects and resources, and process control block (PCB).
- Runs when it is scheduled to run by the OS (kernel)
- OS gives the control of the CPU on a process‘s request (system call).
- Runs by executing the instructions and the continuous changes of its state takes place as the program counter (PC) changes.
- Process is that executing unit of computation, which is controlled by some process (of the OS) for a scheduling mechanism that lets it execute on the CPU and by some process at OS for a resource management mechanism that lets it use the system- memory and other system resources such as network, file, display or printer.
- Application program can be said to consist of number of processes

Example - Mobile Phone Device embedded software

- Software highly complex.
- Number of functions, ISRs, processes threads, multiple physical and virtual device drivers, and several program objects that must be concurrently processed on a single processor.
- Voice encoding and convoluting process─ the device captures the spoken words through a speaker and generates the digital signals after analog to digital conversion, the digits are encoded and convoluted using a CODEC,
- Modulating process,
- Display process,
- GUIs (graphic user interfaces), and
- Key input process ─ for provisioning of the user interrupts

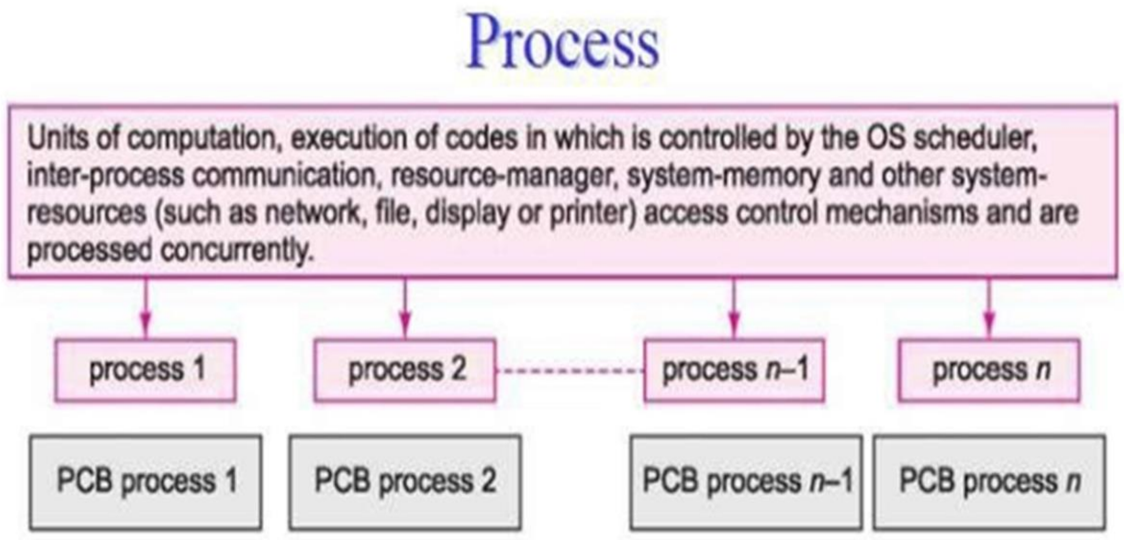
Process Control Block

- A data structure having the information using which the OS controls the Process state.
- Stores in protected memory area of the kernel.

– Consists of the information about the process state

Information about the process state at Process Control Block:

– Process ID,
– Process priority,
– Parent process (if any),
– Child process (if any), and
– Address to the next process PCB which will run,
– Allocated program memory address blocks in physical memory and in secondary (virtual) memory for the process-codes,
– Allocated process-specific data address blocks
– Allocated process-heap (data generated during the program run) addresses,
– Allocated process-stack addresses for the functions called during running of the process,
– Allocated addresses of CPU register-save area as a process context represents by CPU registers, which include the program counter and stack pointer
– Allocated addresses of CPU register-save area as a process context [Register- contents (define process context) include the program counter and stack pointer contents]
– process-state signal mask [when mask is set to 0 (active) the process is inhibited from running and when reset to 1, the process is allowed to run],
– Signals (messages) dispatch table [process IPC functions],
– OS allocated resources‘ descriptors (for example, file descriptors for open files, device descriptors for open (accessible) devices, device-buffer addresses and status, socket-descriptor for open socket), and
– Security restrictions and permissions.


Process
Units of computation, execution of codes in which is controlled by the OS scheduler, inter-process communication, resource-manager, system-memory and other system-resources (such as network, file, display or printer) access control mechanisms and are processed concurrently.
process 1
process 2
process n–1
process n
PCB process 1
PCB process 2
PCB process n–1
PCB process n

Thread

Thread Features

- A thread consists of executable program (codes), *state* of which is controlled by OS,
- The state information– *thread-status* (running, blocked, or finished), *thread structure*— its data, objects and a subset of the process resources, and *thread- stack*.
- Considered a lightweight process and a process level controlled entity.[Light weight means its running does not depend on system resources] .
- Process… heavyweight
  - Process considered as a heavyweight process and a kernel-levelcontrolled entity.
  - Process thus can have codes in secondary memory from which the pages can be swapped into the physical primary memory during running of the process. [Heavy weight means its running may depend on system resources]
  - May have process structure with the virtual memory map, file descriptors, user–ID, etc.
  - Can have multiple threads, which share the process structure thread
  - *A* process or sub-process within a process that has its own program counter, its own stack pointer and stack, its own priority parameter for its scheduling by a thread scheduler
  - Its' variables that load into the processor registers on context switching.
  - Has own signal mask at the kernel. Thread's signal mask
  - When unmasked lets the thread activate and run.
  - When masked, the thread is put into a queue of pending threads.

Multiprocessing OS

- ➢ A multiprocessing OS runs more than one processes.
- ➢ When a process consists of multiple threads, it is called multithreaded process.
- ➢ A thread can be considered as daughter process.
- ➢ A thread defines a minimum unit of a multithreaded process that an OS schedules onto the CPU and allocates other system resources.

Thread parameters

- ➢ Each thread has independent parameters ID, priority, program counter, stack pointer, CPU registers and its present status.

- Thread states─ starting, running, blocked (sleep) and finished

Thread's stack

- When a function in a thread in OS is called, the calling function state is placed on the stack top.
- When there is return the calling function takes the state information from the stack top
- A data structure having the information using which the OS controls the thread state.
- Stores in protected memory area of the kernel.
- Consists of the information about the thread state

Task

Task Features

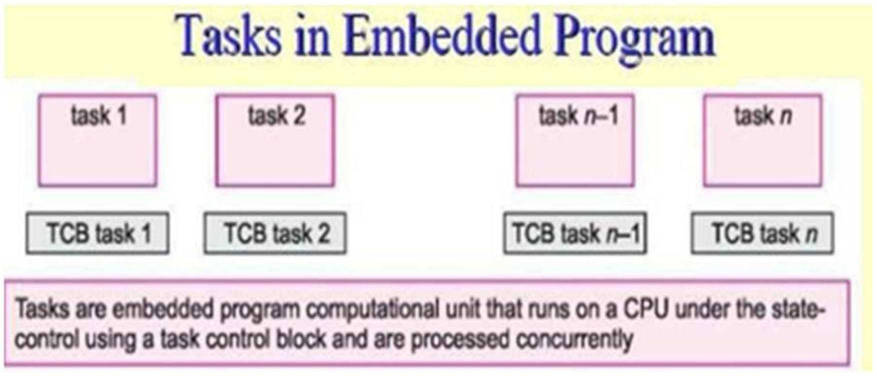
- An application program can also be said to be a program consisting of the tasks and task behaviors in various states that are controlled by OS.
- A task is like a process or thread in an OS.
- Task─ term used for the process in the RTOSes for the embedded systems.
  - For example, VxWorks and µCOS-II are the RTOSes, which use the term task.

Main Features of a 'Task' -

(i) 'Task State'(such as 'Running, 'Blocked' or 'Finished').

(ii)Task Structure(ie. Its Data, Objects & Resources).

(iii)Task Control Block(TCB) .


Tasks in Embedded Program
task 1
task 2
task n–1
task n
TCB task 1
TCB task 2
TCB task n–1
TCB task n
Tasks are embedded program computational unit that runs on a CPU under the state-control using a task control block and are processed concurrently

- A task consists of executable program (codes), ***state*** of which is controlled by OS, the *state* during running of a task represented by information of process status (running, blocked, or finished),process-structure—its data, objects and resources, and task control block (PCB).
- Runs when it is scheduled to run by the OS (kernel), which gives the control of the CPU on a task request (system call) or a message.
- Runs by executing the instructions and the continuous changes of its state takes place as the program counter (PC) changes.
- Task is that executing unit of computation, which is controlled by some process at the

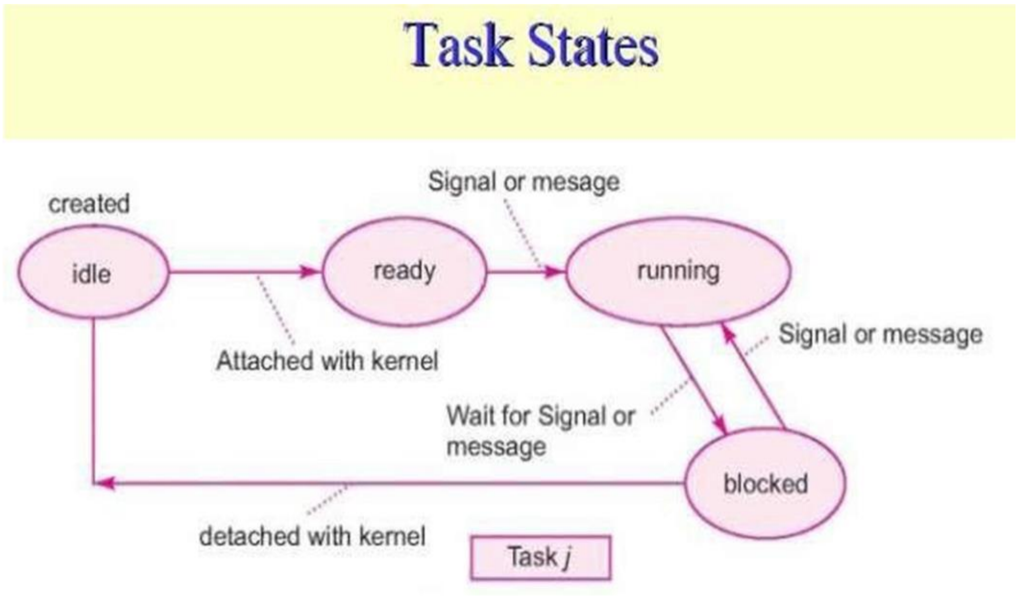
OS scheduling mechanism, which lets it execute on the CPU and by some process at OS for a resource-management mechanism that lets it use the system memory and other system-resources such as network, file, display or printer.

- A task is an independent process.
- No task can call another task. [It is unlike a C (or C++) function, which can call another function.]
- The task─ can send signal (s) or message(s) that can let another task run.
- The OS can only block a running task and let another task gain access of CPU to run the servicing codes

Task States

Following are the five states of a task that may have.

(i) Idle state [Not attached or not registered]
(ii) Ready State [Attached or registered]
(iii) Running state
(iv) Blocked (waiting) state
(v) Delayed for a preset period



Idle (created) state

- The task has been created and memory allotted to its structure however, it is not ready and is not schedulable by kernel.

Ready (Active) State

- The created task is ready and is schedulable by the kernel but not running at present as another higher priority task is scheduled to run and gets the system resources at this instance.

Running state

- Executing the codes and getting the system resources at this instance. It will run till it needs some IPC (input) or wait for an event or till it gets pre- empted by another higher priority task than thisone.

Blocked (waiting) state

- Execution of task codes suspends after saving the needed parameters into its Context. It needs some IPC (input) or it needs to wait for an event or wait for higher priority task to block to enable running after blocking.

Deleted (finished) state

- Deleted Task─ The created task has memory deallotted to its structure. It frees the memory. Task has to be re-created.

Task Switching - 'Task Switching' is done for 'Multi -Tasking' . Each task is allowed to use the CPU for some time, after which it relinquishes the CPU & this allows the next task to execute by using the CPU. In effect, bundles of tasks get executed each time.

Context of a Task - There is a TCB (Task Control Block) associated with each task, which contains all the information of that task. This information is called the 'Context' of the task.

For eg., In a particular task, there are Register Contents, Memory Pointers etc & this is meant by the context of the task.

Context Switching –The context of the new task will be different from that of the stopped/abandoned task. So for a task switching, a 'Context Switching' also needed.

Context Saving – All the info of the old task is to be saved & this is 'Context Saving'. Since context saving is done, there is no difficulty in resuming this task from the point at which it stopped earlier when it gets the service of the CPU again later.

Context Retrieval –When the CPU takes up a new task, the context changes to that of the new task i.e. when an old task which was stopped earlier is taken up again, a 'Context Retrieval' occurs. In a multi-tasking system, context switching, context saving & context retrieval are activities that occur continuously. All these activities are time-consuming activities & all of them cause memory writes (for context saving) & memory reads (for context retrieval) as the TCBs are saved in the memory.

MULTI-TASKING

If there are multiple programs that need execution at the same time, we call it a 'Multi-programming' Environment & the system is a 'Multi – tasking System'. In the case of a single CPU & multiple tasks will be awaiting service from the CPU. One task should not monopolize the use of the CPU, so the CPU must be shared between different tasks. To create a feel of parallelism, each task is getting executed , but not at the same time. This is sort of parallelism is achieved by 'Multi - tasking'.

Process/Task Scheduling

A **scheduler** is a very important unit in an operating system. In general, a scheduler does the sharing of 'resource' between the requesters. The scheduler does not want to give the resource to just one of the requesting parties alone, because all tasks are equally important. In a computer system, when a number of tasks arise, the scheduler has the duty of allocating the services of the CPU to each one of them, based on a set of scheduling conditions. The conditions depend on the types of applications for which the operating system is used.

There are two main strategies for Scheduling Algorithms –

(1) Non-Preemptive
(2)Pre-Emptive

Non-Pre emptive Scheduling Methods

The various non pre emptive scheduling methods are

1. **Co-operative Scheduling -** In this scheduling strategy, each task is allowed to execute up to its finishing state, then only the next task is taken up for the execution. While one task executes , the others must be willing to wait. This gives the name 'co-operative' to the scheduling algorithm. This is a typical case of 'First Come First Serve Scheduling (FCFS) or a First In First Out(FIFO)scheme.

2. **Shortest Job Next (SJN)** –This is also called as the 'Shortest Job First '(SJF) algorithm. Here the tasks are queued in the order of increasing service times & the one task with the shortest service time gets to execute first. But the problem is that, usually the execution times of tasks are not known in advance. Then the service time can be estimated by using their history of each of the tasks.

3. **Priority-Based Scheduling -** Here, tasks have priorities which are represented in terms of priority numbers For eg., the numbers 0 to 255 as representing priorities, with 0 representing the highest priority & higher no indicating lower priorities. Systems such as real-time operating systems use priority-based scheduling methods.

Pre-Emptive Scheduling Methods

Here, when a task is running (using the CPU), it is pushed out from 'Running' State to 'Ready' State & an another task from the ready queue is taken up the usage of the CPU. So, we can say that the task has been pre-empted (or removed or replaced).

There are different types of scheduling methods.

1. **Round-Robin Scheduling -** This method is also called as 'Time Sharing' or 'Time Slice' system. Here, 'Time Slices' are defined. Suppose there are 'n' no. of tasks in a system,

then each of them is allowed to execute for a period equal to the time slice. After this, the next task gets its turn, but it can also use the CPU only for a time equal to the defined time slice. Thus all tasks get their chance at least once, in one round.

2. **Pre-emptive Priority based Scheduling**: - In this method, at any time it should be ensured that it is the highest priority task in the ready queue that should be running. At any time, if a task with a higher priority than the one that is currently running enters the 'Ready Queue', the current task is pre-empted & the new one is taken up for execution.

3. **Pre-emptive SJN / Shortest Remaining Time (SRT) :-** Here also the scheduling is in the order of increasing service times. Thus, tasks with the shortest service times are executed earlier. If a new task enters the ready queue, its service time is compared with the remaining service time of the currently executing task. If the new task has a shorter service time than the remaining running time of the current task, the current task is pre-empted & the new one is serviced until it completes.

SHARED DATA

Assume that several tasks (or ISRs or functions) share a variable. Let us assume that at an instant, the value of that variable operates. During the operations on it, only a part of the operation is completed & another part remains as incomplete. At that moment, let us assume that there is an interrupt occurred or the OS scheduler transfers the control to other task/ process (with higher priority).

Now, assume that there is another task (or function) also shares this same variable mentioned earlier. The value of the variable may differ from the expected value if the earlier stopped operation had not been completed. This is due to the incomplete operation an occurred earlier. "*Use modifier volatile'* function with a declaration of a variable that returns from the interrupt. *'Use re-entrant functions'* with the instructions in that part of a function that needs its complete execution before it can be interrupted. Put a shared variable in a circular queue. Disable the interrupts before a critical section starts executing & enable the interrupts on its completion. Lock the OS scheduling mechanism before a critical section starts executing & lock the scheduler at the end of the critical section.

Inter Process Communication (IPC)

Inter process communication (IPC) means that a process (scheduler or task or ISR) generates some information by setting or resetting a Token or value, or generates an output so that it lets another process take note or to signal to OS for starting a process or use it under the control of an OS.

Inter process communication functions provide answer to these questions. IPCs in a multi-process system are used:

I. To interrupt present process for an interrupt service by another process,

2. To notify occurrence of an event to another process waiting for the event.

3. To set or reset a signal, token, or flag or generate message from the certain sets of computations finishing on one task, and to let the other tasks take note of signal or get the message.

4. To generate information about certain sets of computations finishing on one process, to let the other process wait for finishing those computations and when the computations finish then take note of the information and let others generate information.

5. To generate some information in a process (scheduler, task or ISR) or value or generate a message in output so that it lets another process take note or use it through the kernel.

OSs provides the **software programmer** the following **IPC functions,** which can he used.

I. Signals

2. Semaphores as taken or counting Semaphores for the inter task communication between tasks sharing a common buffer or operations

3, Message Passing, Queues and mailboxes

4. Pipes and socket

5. Remote procedure calls (RPCs) for distributed processes,

Signals

Features:

- One way for inter process communication is to use an OS function signal ( ), The OS itself can issue an interrupt signal similar to the interrupts provided by different external/ internal hardware sources.
- It is provided in Unix, Linux and several RTOSs.
- Unix and Linux OSs use signals profusely and have thirty-one different types of Signals for the various events.
- Unless masked by a signal mask, the signal allows the execution of the **signal handling function** and allows the handler to run just as a hardware interrupt allows the execution of an ISR.
- When a signal is sent from a process, OS interrupts the process execution and calls the function for signal handling. On return from the signal handler, the process continues as before.
- Signal provides the shortest communication.

The following are **the signal related IPC functions**

1. signal ( ) :- The OS function signal ( ) enables the OS to run a particular 'Task' (or a process) as per the integer no. 'n' given in the argument of the signal function called.

   For eg., the instruction 'signal (8)' issued by the OS when an illegal mathematical operation is attempted by a user program.

2. SigHandler ( ) :- This function is used to create a signal handler corresponding to a signal identified by the signal number and define a pointer to signal context. The signal context saves the registers on signal. The 'Signal Handler' runs in a way similar to a 'Highest Priority ISR'

   o Similarly, a 'handler task' runs for a signal provided that the signal is not masked. When the signal is sent from a process, OS interrupts the process execution & calls the function for signal handling. When there is return from the signaled task (or process), the process which sent the signal, runs the return codes similar to that happens while returning from an ISR. On return from the signal handler, the process continues as before

3. Connect ():- This function is used to connect an interrupt vector to a signal number, with signaled handler function and signal handler arguments. The interrupt vector provides the program counter value for the signal handler function address.
4. .Mask ():- This function is used to mask the signal
5. Unmask ():- This function is used to unmask the signal
6. Ignore () :- This function is used to ignore the signal

Signal-Dispatch Table

- ■ The OS maintains a 'Signal-Dispatch Table' which sets or resets a flag corresponding each signal number 'n'
- ■ A flag corresponding to 'n' sets when the 'signal (n)' is issued.
- ■ This flag will get resets when the service to that signal starts using the corresponding 'Signal Handler' task.

Signal-Mask Table

- ■ The OS maintains a 'Signal-Mask Table' which sets or reset a mask bit for each signal number 'n' .
- ■ The mask sets when a signal is ignored in a situation in which issuing of a signal has occurred; but the corresponding 'Signal Handler' does not get started.
- ■ The mask resets when the signal is serviced on issuing a signal function & the signal handler executes.
- ■ 'Issuing of a signal' is equivalent to setting a flag bit at an SFR register in the processor on the occurrence of a hardware interrupt.
- ■ If the 'signal ( )' is not masked by a 'Signal Mask', the signal allows the execution of the 'Signal-Handling Function' & allows the handler to run just as a hardware interrupt allows the execution of an ISR.

Application Example:

- □ An important application of the signals is to handle exception.
- □ An exception is a process that is executed on a specific reported run-time condition.
- □ A signal reports an error (called 'Exception') during the running of a task and then lets the scheduler initiate an error-handling process or ISR, for example, initiate error-login task.

Advantage of using a Signal

- It takes the shortest possible CPU time.

Drawbacks of using a Signal

- Signal is handled only by a very high priority process (service routine). That may disrupt the usual schedule and usual priority inheritance mechanism.
- Signal may cause reentrancy problem [process not returning to state identical to the one before signal handler process executed].

Semaphore

- A semaphore is special kind of shared program variable.
- A semaphore is a **kernel object that one or more tasks can acquire or release** for the purpose of synchronization or mutual exclusion.
- Mutual exclusion is a provision by which only one task at a time can access a shared resource (port, memory block, etc.)
- The value of a semaphore is a non-negative integer.
- If the integer data is only allowed to take the values 0 and 1 then the semaphore is referred to as a **binary semaphore**.
- **Binary semaphore** allows only one process at a time to access the shared resource.
- If the integer data is allowed to take any non-negative value then the semaphore is referred to as a **General Semaphore or Counting Semaphore**.
- **Counting Semaphore** allows N > 1 processes to access the resource simultaneously. Instead of having states empty and full, it uses a counter S. The counter is initialized to N, with S = 0 corresponding to the full state.
- *It is that part of the program (or code) where the 'Shared Memory/ 'Shared Variable/ 'Shared Data' / 'Shared Device' is accessed*
- OS provides the semaphore IPC functions for creating, releasing and taking of the semaphore

Semaphore functions:

The following are the functions, which are generally provided in an OS, for example µCOS-II for the semaphores.

I.OSSemCreate. a semaphore function to create the semaphore in an event control block (ECB). Initialize it with an initial value.

2. OSSemPost:-. a function which sends the semaphore notification to ECB and its value increments on event occurrence (used in ISR, as well as tasks).

3. OSSemPend. a function, which waits the semaphore from an event, and its value decrements on taking note of that event occurrence (used in tasks not in ISRs)

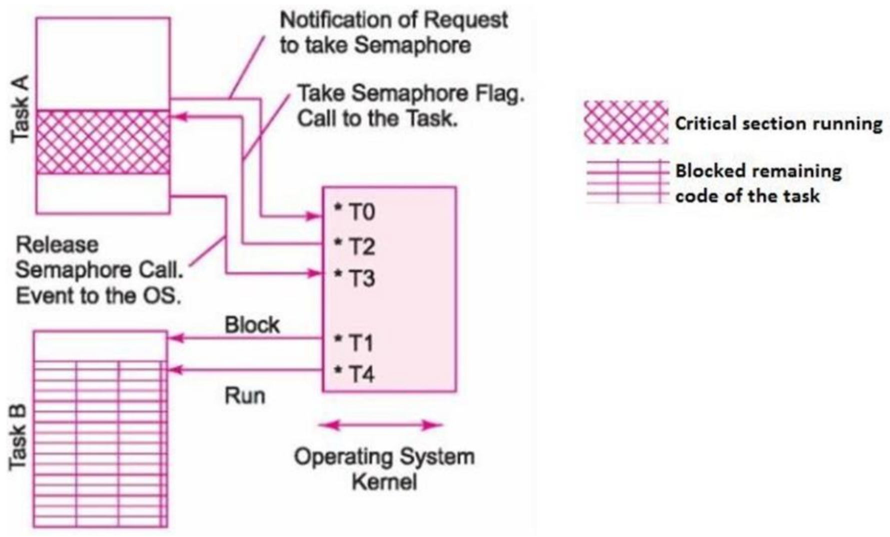
4, OSSemAccept:- a function. which reads and returns the present semaphore value and if it shows occurrence of an event (by non-zero value) then it takes note of that and decrements that value (no wait; used in ISRs as well as tacks).

5. OSSemOuery:- a function, which queries the semaphore for an event occurrence or non-occurrence by reading its value and returns the present semaphore value, and returns pointer to the data structure

6. OSSemData. The semaphore value does not decrease. The OSSemData points to the present value and table of the tasks waiting for the semaphore used in tasks.

Semaphore as a resource key and for critical sections having shared for critical sections having shared resource (s):

- Shared Resources like shared memory buffer are to be used only by one task (process or thread) at an instance.
- OS Functions provide for the use of a semaphore resource key for running of the codes in critical section.


Notification of Request to take Semaphore
Take Semaphore Flag. Call to the Task.
Task A
Release Semaphore Call. Event to the OS.
Block
Run
Task B
* T0
* T2
* T3
* T1
* T4
Operating System Kernel
Critical section running
Blocked remaining code of the task

- Let a task A, when getting access to a resource/critical section notifies to the OS to have taken the semaphore (take notice)That is, an OS function *OSSemPend()* runs to notify. The OS returns the semaphore as taken (accepted) by decrementing the semaphore from 1 to 0.
- Now the task A accesses the resource.
- The task A, after completing the access to a resource/critical section it notifies to the OS to have posted that semaphore (post notice). That is an OS function *OSSemPost()* runs to notify. The OS returns the semaphore as released by incrementing the semaphore from 0 to 1.
- Figure below shows the use of semaphore between A and B. It shows the five sequential actions at five different times.

Mutex Semaphore for use as resource key:

- Mutex means mutually exclusive key.
- Mutex is a binary semaphore usable for protecting use of resource by other task section at an instance
- Let the semaphore sm has an initial value = 1
- When the semaphore is taken by a task the semaphore sm decrements from 1 to 0 and the waiting task codes starts.
- Assume that the sm increments from 0 to 1 for signalling or notifying end of use of the semaphore that section of codes in the task.
- When sm = 0 ─ assumed that it has been taken (or accepted) and other task code section

has not taken it yet and using the resource

- When sm = 1─ assumed that it has been released (or sent or posted) and other task code section can now take the key and use the resource.

P and V semaphores

- An efficient synchronization mechanism
- P and V semaphores represent by integers in place of binary or unsigned integers.
- The semaphore, apart from initialization, is accessed only through two standard atomic operations - P and V.
- P semaphore function signals that the task requires a resource and if not available waits for it.
- V semaphore function signals which the task passes to the OS that the resource is now free for the other users.

Message Queues

- A message queue is an inter process communication mechanism which uses an n byte memory block as queue.
- The message pointers post into a queue by the tasks either at the back as ina queue or at the front as in a stack.
- Every OS provides message queue IPC functions.

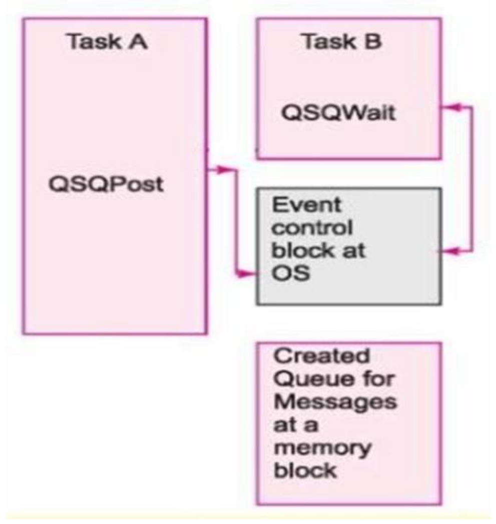
Features:

1. OS provides for inserting and deleting the message-pointers or messages.
2. Each queue for a message need initialization (creation) before using the functions in the scheduler for the message queue
3. There may be a provision for multiple queues for the multiple types or destinations of messages. Each queue may have an ID.
4. Each queue either has a user definable size (upper limit for number of bytes) or a fixed pre-defined size assigned by the scheduler.
5. When an RTOS call is to insert into the queue, the bytes are as per the pointed number of bytes. For example, for an integer or float variable as a pointer, there will be four bytes inserted per call. If the pointer is for an array of 8 integers, then 32 bytes will be inserted into the queue.
6. When a queue becomes full, there may be a need for error handling and user codes for blocking the task(s). There may not be self-blocking.

Queue IPC functions

- Figure (a) shows the memory blocks at OS for inserting deleting and other functions.
- Figure (b) shows the functions for the queue in the OS.

- Figure (c) shows a queue message block with the messages or message pointers.
- Two pointers *QHEAD and *QTAIL are for queue head and tail memory locations


Task A
QSQPost
Task B
QSQWait
Event control block at OS
Created Queue for Messages at a memory block

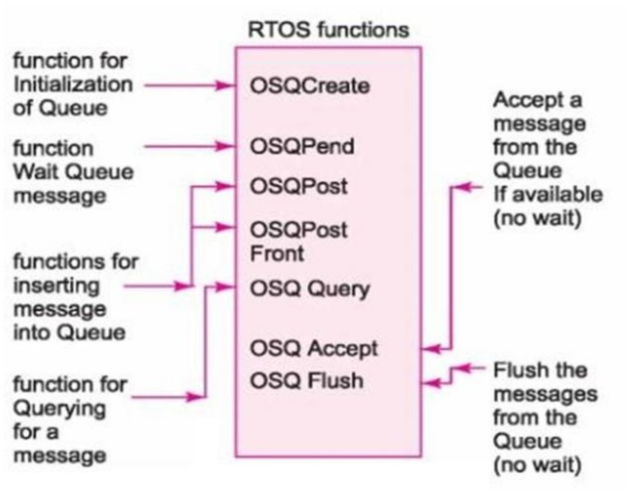

RTOS functions
function for Initialization of Queue
OSQCreate
function Wait Queue message
OSQPend
OSQPost
OSQPost Front
functions for inserting message into Queue
OSQ Query
OSQ Accept
OSQ Flush
function for Querying for a message
Accept a message from the Queue If available (no wait)
Flush the messages from the Queue (no wait)

Figure (a) Queue handling by tasks Figure (b) Queue Functions

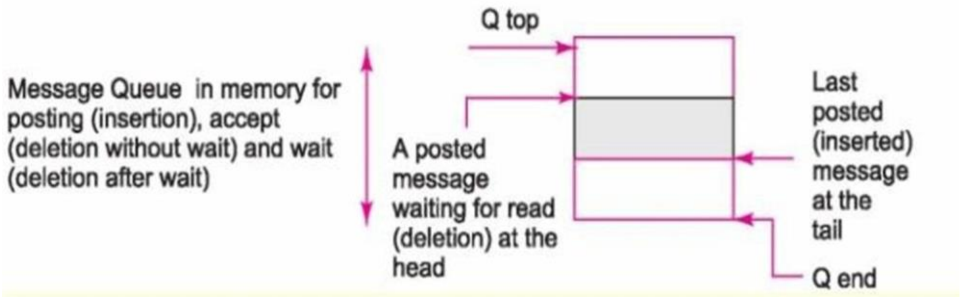

Q top
Message Queue in memory for posting (insertion), accept (deletion without wait) and wait (deletion after wait)
A posted message waiting for read (deletion) at the head
Last posted (inserted) message at the tail
Q end

Figure (c) Queue message block

OSQCreate

- This function is used to create a queue and initialize the queue.

OSQPost

- This function is used to post a message to the message block as per the queue tail pointer, *QTAIL.

OSQPend

- This function is used to wait for a queue message at the queue and reads and deletes that when received.

OSQAccept

- This function is used to delete the present queue head after checking its presence yes or no and after the deletion the queue head pointer increments.

OSQFlush

- This function is used to deletes messages from queue head to tail, after the flush the queue head and tail points to QTop, pointer to start of the queue.

OSQQuery

- This function is used to query the queue message-block when read and but the message is not deleted. The function returns pointer to the message queue
  *QHEAD if there are the messages in the queue or else NULL. It return a pointer to data structure of the queue data structure which has *QHEAD, number of queued messages,

size of the queue and. table of tasks waiting for the messages from the queue.

OSQPostFront

- This function is used to send a message as per the queue front pointer, *QHEAD. Use of this function is made in the following situations. A message is urgent or is of higher priority than all the previously posted message into the queue.

Mailbox

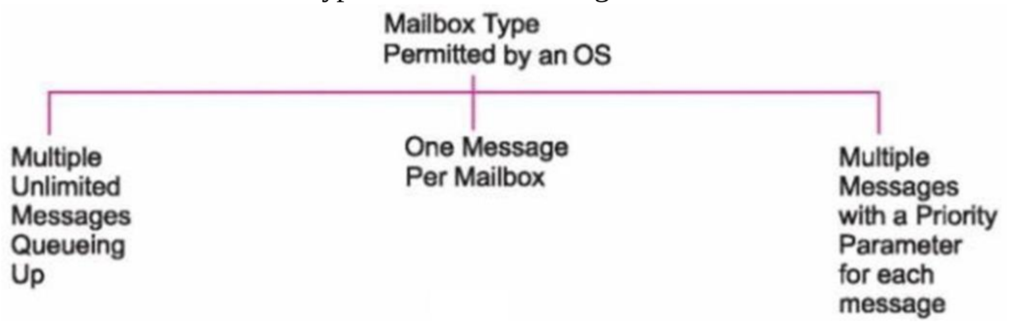
`Mailbox' is an indirect way of sending messages. In an OS scenario, it is like a mailbox maintained by the OS ie. The OS manages the mailbox which is common to the whole system. An OS provides the IPC functions such as 'Create', 'Post', 'Pend', 'Accept' & 'Query' for using the mailbox

Features

- Mailbox (for message) is an IPC through a message-block at an OS that can be used only by a single destined task.
- The source (mail sender) is the task that sends the message pointer to a created mailbox.
- OS provides for inserting and deleting message into the mailbox message-pointer.
- Each mailbox for a message need initialization (creation) before using the functions in the scheduler for the message queue and message pointer pointing to Null.
- There may be a provision for multiple mailboxes for the multiple types or destinations of messages.
- Each mailbox has an ID.
- Each mailbox usually has one message pointer only, which can point to message.

Types:

- There are three types of mailboxes as given below.


Mailbox Type Permitted by an OS
Multiple Unlimited Messages Queueing Up
One Message Per Mailbox
Multiple Messages with a Priority Parameter for each message

- One type is only one message per mailbox is available.
- Another type is mailbox with provision for multiple messages or message pointers.
- Third one is OS can provide multiple mailbox messages with each message having a priority parameter. The read (deletion) can then only be on priority basis in case mailbox has multiple messages.

Mailbox Functions

1. OSMBoxCreate:- creates a box and initializes the mailbox contents with a NULL pointer.
2. OSMBoxPost :- sends (writes) a message to the box.
3. OSMBoxWait (pend):- waits for a mailbox message, which is read when received.
4. OSMBoxAccept:- reads the current message pointer after checking the presence yes or no (no wait). Deletes the mailbox when read.
5. OSMeciRQuery queries the mailbox when mad and not needed later.

Pipes

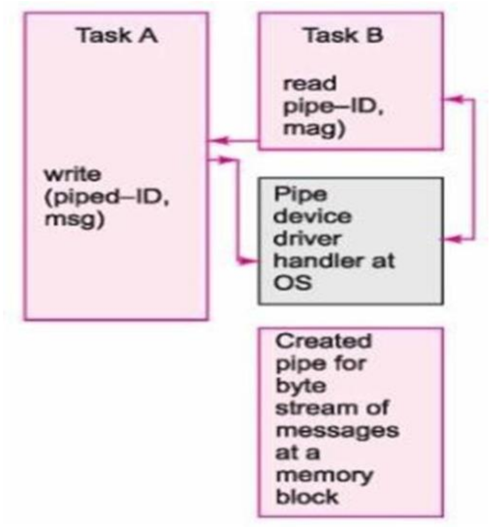
- A pipe is a 'data structure' acting as a 'FIFO Buffer' into which writing from one end & reading from the other end is possible.
- Pipe is a device used for the inter process communication.
- A message-pipe is a device for inserting (writing) and deleting (reading) from that between two given inter-connected tasks or two sets of tasks.
- Writing and reading from a pipe is like using a C command *fwrite* with a file name to write into a named file, and C command *fread* with a file name to read into a named file.
- The total no. of messages limit & max. size per message can also be provided when creating a pipe.
- Pipe is a '**uni-directional**' message transfer mechanism.
- If bi-directional data transfer is needed between two tasks, two pipes are needed

Write and read using Pipe

- One task using the function *fwrite* in a set of tasks can write to a pipe at the back pointer address, *pBACK.
- One task using the function *fread* in a set of tasks can read (delete) from a pipe at the front pointer address, *pFRONT.
- In a pipe there may be no fixed number of bytes per message but there is end pointer.
- A pipe can therefore be limited and have a variable number of bytes per message between the initial and final pointers.
- Pipe is unidirectional. One thread or task inserts into it and other one deletes from it.

Pipe Functions

- Figure (a) shows the write and read the pipe using device drivers.
- Figure (b) shows the functions for the pipe in the OS.
- Figure (c) shows a pipe messages in the message buffer


Task A
write (piped–ID, msg)
Task B
read pipe–ID, mag)
Pipe device driver handler at OS
Created pipe for byte stream of messages at a memory block

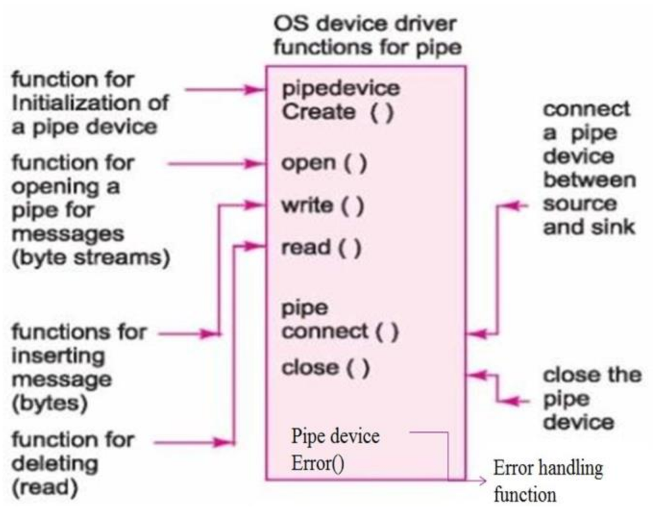

OS device driver functions for pipe
function for Initialization of a pipe device
function for opening a pipe for messages (byte streams)
functions for inserting message (bytes)
function for deleting (read)
pipedevice Create ( )
open ( )
write ( )
read ( )
pipe connect ( )
close ( )
Pipe device Error()
connect a pipe device between source and sink
close the pipe device
Error handling function

Figure (a) Pipe handling by tasks Figure (b) Pipe Functions

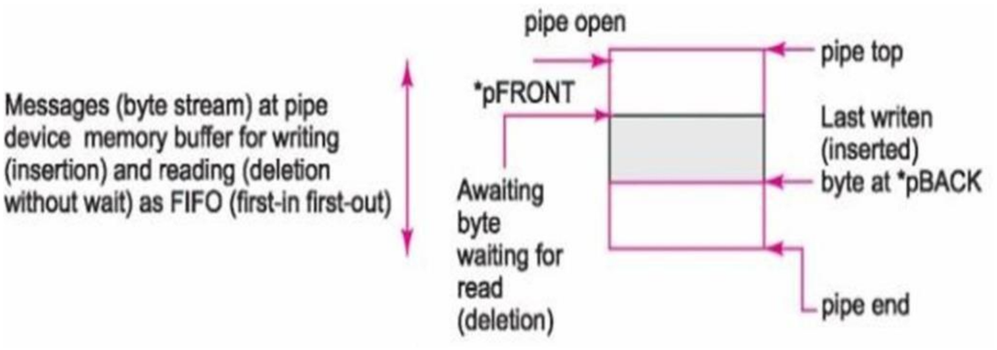

Messages (byte stream) at pipe device memory buffer for writing (insertion) and reading (deletion without wait) as FIFO (first-in first-out)
pipe open
*pFRONT
Awaiting byte waiting for read (deletion)
pipe top
Last writen (inserted) byte at *pBACK
pipe end

Figure (c) Pipe message block

The **OS functions for pipe are** the following:

5 pipeDevCreate:- for creating a device.,which functions as pipe.

6 open ( ):- for opening the device to enable its use from beginning of its allocated buffer. its use is with options and restrictions (or permissions) defined at the time of opening.

7 connect ():-for connecting a thread or task inserting bytes to the thread or task deleting bytes from the pipe.

8 write () function for inserting (writing) from the bottom of the empty memory space in the buffer allotted to it

9 read () function for deleting (reading) from the pipe from the bottom of the unread memory spaces in the buffer filled after writing into the pipe.

10 close ( ) for closing the device to enable its use from beginning of its allocated buffer only after opening it again.

Sockets

- □ Provides a device like mechanism for bi-direction communication.

Need for Sockets for the Inter Process Communication

- – A pipe could be used for inserting the byte steam by a process and deleting the bytes from the stream by another process.
- – However, for example, we need that the card information to be transferred from a process A as byte stream to the host machine process B and the B sends messages as byte stream to the A
- – There is need for bi-directional communication between A and B.
- – We need that the A and B ID or address information when communicating must also be specified either for the destination alone or for the source and destination both. [The messages in a letter are sent along with address specification.]
- – We need to use a protocol for communication
- – A protocol, for example, provides along with the byte stream information of the address or port of the destination or addresses or ports of source and destination both or the protocol may provide for the addresses and ports of source and destination in case of the remote processes.
- – For example, UDP (user datagram protocol) is used as connectionless protocol.
- – UDP header contains source port (optional) and destination port numbers, length of the datagram and checksum.
- – Port means a process or task for specific application.
- – Port number specifies the process.
- – Datagram means a data, which is independent need not in sequence with the previously sent data.
- – Checksum is sum of the bytes to enable the checking of the erroneous data transfer.
- – TCP (transport control protocol) is used as connection oriented protocol.

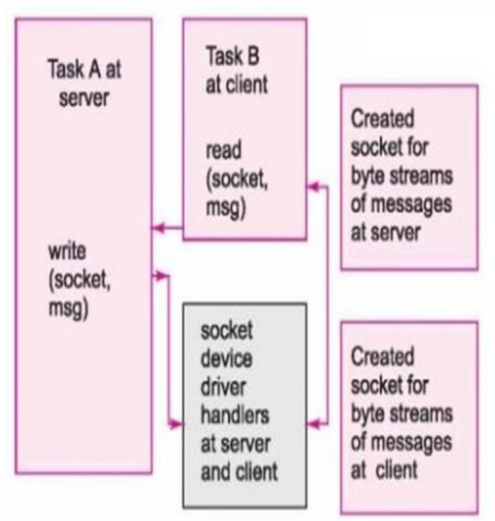
Features:

- – Socket Provides for a bi-directional pipe like device, which also use a protocol between source and destination processes for transferring the bytes.
- – Provides for establishing and closing a connection between source and destination processes using a protocol for transferring the bytes.
- – May provide for listening from multiple sources or multicasting tomultiple destinations.
- – Two tasks at two distinct places or locally interconnect through the sockets.
- – Multiple tasks at multiple distinct places interconnect through the sockets to a socket at a server process.
- – The client and server sockets can run on same CPU or at distant CPUs on the Internet.
- – Sockets can be using a different domain. For example, a socket domain can be TCP (transport control protocol) , another socket domain may be UDP(transport control protocol), the card and host example socket domain is different.
- – Two processes (or sections of a task) at two sets of ports interconnect (perform inter process communication) through a socket at each.
- – A socket stream between two specified sections (ports)─at the specified sets (addresses) sent with a port-specific protocol.

- Each socket ─process address (similar to a network or IP address) and section (similar to a port) specification. The sections (or ports or tasks) and sets of processes (addresses) may be on the same computer or on a network.
- There has to be a specific protocol in which the messages at the socket interconnect between (i, j) and (m, n). [i and m are process addresses, and j and n are port or section specifications]
- A pipe does not have protocol based inter-processor communication, while socket provides that.
- A socket can be a client-server socket. Client Socket and server socket functions are different.
- A socket can be a peer-to-peer socket IPC. At source and destination sockets have similar functions.

Socket Functions

- Figure (a) shows the write by a server task and read by a client taskusing device drivers.
- Figure (b) shows the functions for the socket for both server and client in the OS.
- Figure (c) shows byte stream at socket device.


Task A at server
write (socket, msg)
Task B at client
read (socket, msg)
socket device driver handlers at server and client
Created socket for byte streams of messages at server
Created socket for byte streams of messages at client
Figure (a) Socket handling by tasks

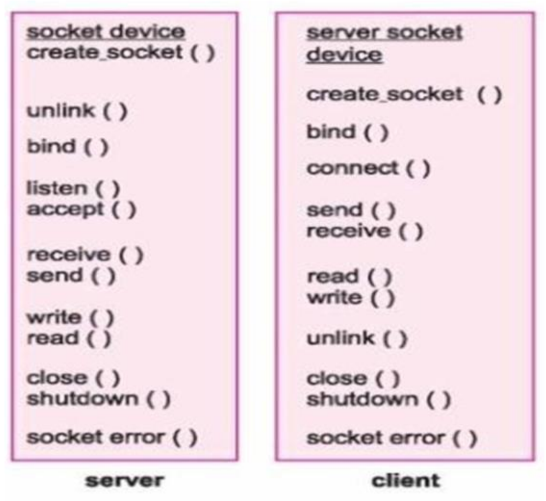
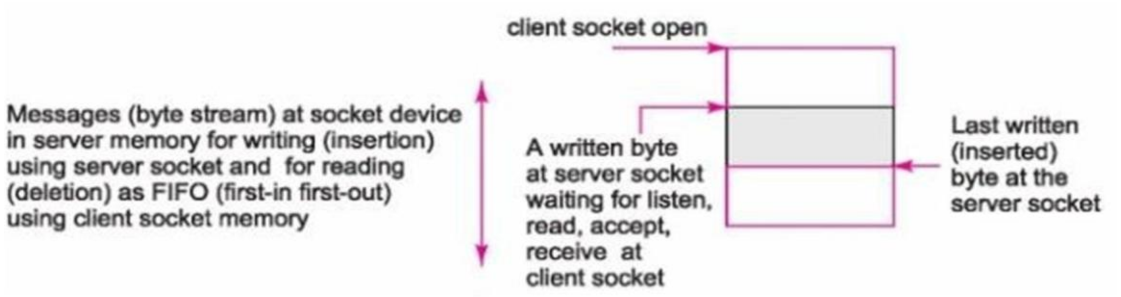


Figure (b) Socket Functions for both server and client.


client socket open
Messages (byte stream) at socket device in server memory for writing (insertion) using server socket and for reading (deletion) as FIFO (first-in first-out) using client socket memory
A written byte at server socket waiting for listen, read, accept, receive at client socket
Last written (inserted) byte at the server socket
Figure (c) Socket message stream

The **OS functions for socket** in Unix are as following.

I. The socket ( ) :- in place of open ( ) in case of pipe gives a socket descriptor sfd. The socket ( ) enables its use from beginning of its allocated buffer at the socket address its use with option and restrictions or permissions defined at the time of opening. A socket can be a stream. SUCK_STREAM or UDP datagram SOCK_DGRAM.

2. The unlink ( ) before the bind ( ).

3. The bind ( ) for binding a thread or task inserting bytes into the socket to the thread or task and deleting bytes from the socket. bind ( ) the socket descriptor to an address in the Unix domain, bind (sfd, (struct sockaddr *) &local, len); where len is string length. sockaddr is a data structure with a record of I6-bit unsigned num and a path for the file and a data structure struct sockaddr_un (unsigned short num; char path(108)

4. The listen (sfd, 16) function for listening 16 queued connections from the client socket.

5. The accept ( ) accepts the client connection and gives a second socket descriptor.

6. The recv O function for deleting (reading) and receiving from the socket from the bottom of unread memory spaces in buffer. The buffer has messages after writing into the socket.

7. The send ( ) function for inserting (writing) and sending from the socket from the bottom of the memory spaces in the buffer filled after writing into the socket.

8. The close ( ) for closing the device to enable its use from beginning of its allocated buffer only after opening it again.

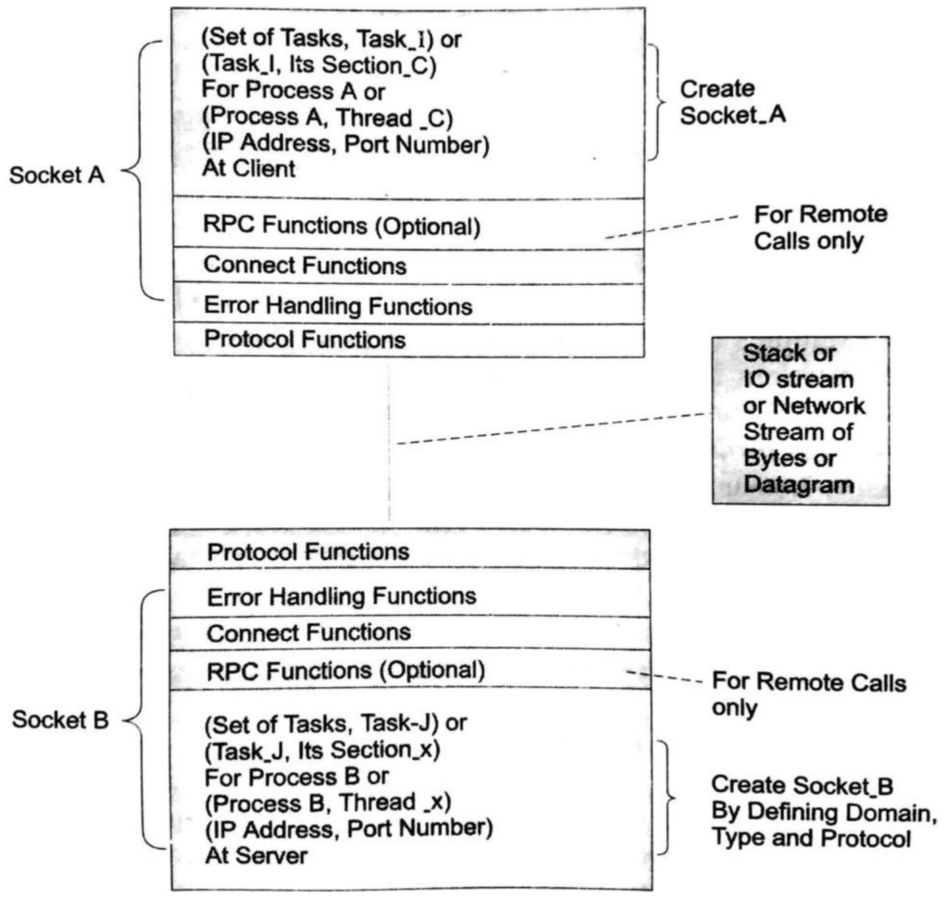
<u>RPC (remote procedure call)</u>

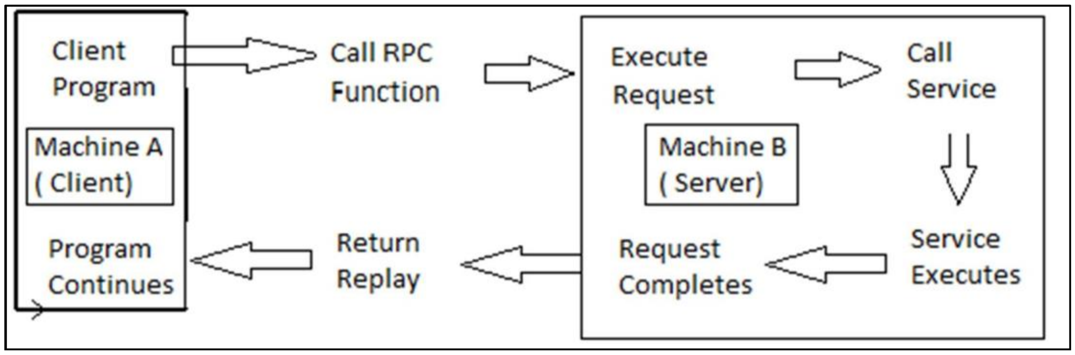
- □ A method used for connecting two remotely placed functions by first using a protocol for connecting the processes. It is used in the cases of distributed tasks.
- □ The RTOS can provide for the use of RPCs. These permits distributed environment for the embedded systems.
- □ The OS IPC function allows a function or method to run at another address space of shared network or other remote computer.
- □ The client makes the call to the function that is local or remote and the server response is either remote or local in the call.
- □ Both systems work in the peer-to-peer communication mode. Each system in peer- to-peer mode can make an RPC.
- □ An RPC permits remote invocation of the processes in the distributed systems.
- □ The RPC provides the inter task communication when a task is at system1 and another task is at system 2.
- □ Figure below shows the initialized virtual sockets between the client set of tasks and a server set of tasks at RTOS.

Terms involved in RPC

- ■ **Client** - A 'process', such as a program or task that requests a service provided by another program.

- **Server** - A 'process', such as a program or task, that responds to requests from a client
- **Endpoint** - The 'name', 'port' or 'group of ports' on a host system that is monitored by a server program for incoming client requests .





MODULE 6

Subject : EC 308 EMBEDDED SYSTEMS
Notes – Module 6

Syllabus

Real time operating systems - Services- Goals – Structures - Kernel - Process Management – Memory Management – Device Management – File System Organization. Micro C/OS-II RTOS - System Level Functions – Task Service Functions – Memory Allocation Related Functions – Semaphore Related Functions. Study of other popular Real Time Operating Systems.

OPERATING SYSTEMS (OS) - INTRODUCTION

- An Operating System is a collection of programs that provides an interface between application programs and the computer system (hardware).
- Its primary function is to provide application programmers with an abstraction of the system resources, such as memory, input-output and processor, which enhances the convenience, efficiency and correctness of their use.
- These programs or functions within the OS provide various kinds of services to the application programs. The application programs, in turn, call these programs to avail of such services. Thus the application programs can view the computer resources as abstract entities.
- In real time systems, most often, such programs cooperate with each other, by exchanging data and synchronizing each other's execution, to achieve the overall functionality and performance of the system.

Types of Operating Systems

1) Stand-Alone Operating System

   It is a complete operating system that works on a desktop or notebook computer. Examples of stand-alone operating systems are:

   DOS, Windows 2000 Professional, Mac OS X

2) Network Operating System

   It is an operating system that provides extensive support for computer networks. A network operating system typically resides on a server. Examples of a network operating system are:

   Windows 2000 Server, Unix, Linux, Solaris

3) Embedded Operating System

   You can find this operating system on handheld computers and small devices, it resides on a ROM chip. Examples of embedded operating systems are:

   •Windows CE, Pocket PC 2002, Palm OS

The above classification is based on the computing hardware environment towards which the OS is targeted. All three types can be either of a real-time or a non-real-time type. For example, VxWorks is an RTOS of the first category, RT-Linux and ARTS are of the second category and Windows CE of the third category.

Real-Time Operating Systems

- □ A Real-Time OS (RTOS) is an OS with special features that make it suitable for building real-time computing applications also referred to as Real-Time Systems (RTS).
- □ An RTS is a (computing) system where correctness of computing depends not only on the correctness of the logical result of the computation, but also on the result delivery time.
- □ An RTS is expected to respond in a timely, predictable way to unpredictable external stimuli.

TYPES OF RTOS

There are two types of Real-Time Operating Systems.

Hard Real-Time Systems - Hard real-time means strict adherence to each task schedule. In hard real-time systems, secondary storage is limited or missing with data stored in ROM. Example: Air traffic control, Nuclear power plant control etc.

Soft Real-Time Systems - Soft Real-Time means that only the precedents and sequence for the task operations are defined. Soft real-time systems have limited utility than hard real-time systems. Example: Telecom, Networks, Web Services, etc.

Multitasking

A multitasking environment allows applications to be constructed as a set of independent tasks, each with a separate thread of execution and its own set of system resources. The inter-task communication facilities allow these tasks to synchronize and coordinate their activity. Multitasking provides the fundamental mechanism for an application to control and react to multiple, discrete real-world events and is therefore essential for many real-time applications.

The following are important requirements that an OS must meet to be considered an RTOS in the contemporary sense.

1. The OS must support priority of tasks and threads
2. A system of priority inheritance must exist. Priority inheritance is a mechanism to ensure that lower priority tasks cannot obstruct the execution of higher priority tasks.
3. The OS must support various types of thread/task synchronization mechanisms.
4. For predictable response:
    a) The time for every system function call to execute should be predictable and independent of the number of objects in the system.

b) Non pre-emptible portions of kernel functions necessary for inter process synchronization and communication are highly optimized, short and deterministic.

c) Non-pre-emptible portions of the interrupt handler routines are kept small and deterministic

d) Interrupt handlers are scheduled and executed at appropriate priority

e) The maximum time during which interrupts are masked by the OS and by device drivers must be known.

f) The maximum time that device drivers use to process an interrupt, and specific IRQ information relating to those device drivers must be known.

g) The interrupt latency (the time from interrupt to task run) must be predictable and compatible with application requirements.

5. For fast response:

a) Run-time overhead is decreased by reducing the unnecessary context switch.

b) Important timings such as context switch time, interrupt latency, semaphore get/release latency must be minimum

REAL TIME - A real time is the time which continuously increments at regular intervals after the start of the system and time for all the activities at difference instances take that time as a reference in the system.

**** REAL TIME OPERATING SYSTEM** - A real time operating system (RTOS) is multitasking operation system for the applications with hard or soft real time constraints.

Real-time constraint means constraint on occurrence of an event and system expected response and latency to the event.

****CHARACTERISTICS OF RTOS**

As like normal operating system, RTOS can have following important characteristics.

1. **Deterministic** - Operations are performed at a fixed, predetermined times or within predetermined time intervals.
2. **Responsiveness** - If interrupt occur, then how long it takes the operating system to service interrupt. It includes amount of time to begin execution of the interrupt.
3. **Response Time** requirements are critical for real-time systems because such systems must meet timing requirements imposed by individuals, devices and data flows external to the system.
4. **Reliability** - Real time systems are performed and controlled all the events in real-time.

So loss of performance may cause major damage and even loss of life. Reliability is the most important characteristics of RTOS.

5. **User Control** – it is important to allow the user control over task priority.

The OS/ RTOS goals are :

- Perfection
- Correctness
- Portability
- Inter-operability
- Providing a common set of interfaces for the system
- Orderly access & control when managing the processes

The RTOS goals are correctness & perfection, to achieve the following

(1)Facilitating easy sharing of resources as per schedule & allocations - Resources mean Processor, Memory, I/Os, Devices, Pipes, Sockets, System Timer, Keyboard, Displays, Printer & other such resources which processes (tasks or threads) request from the OS .No processing task (or thread) uses any resource until it has been allocated by the OS at a given instance .

(2) Facilitating easy implementation of the application software program with the given system hardware - An application programmer can use the OS functions & processes that are provided in the OS.

(3) Optimally scheduling the processes/ tasks on the CPU - OS should provide appropriate ‘Context-Switching & ‘Interrupt-Servicing’ mechanisms.

(4) Providing the Management of the Processes, Tasks, Threads, Memory, IPCs, Devices, I/Os & other OS functions - Management means ‘Creation’, ‘Resource Allocation’, ‘Resource Freeing’, ‘Scheduling (or Synchronizing)’ & ‘Deletion.’

(5) Providing the Management & Easy Interfacing of Files, I/Os, Networking & Network Protocols .

(6) Providing ‘Portability of Application’ on different hardware configurations .

(7) Providing ‘Inter-Operability of the Application’ on different networks .

(8) Providing a ‘common set of interfaces’ that integrates various devices & applications through the standard & open systems .

(9) Easy use of the interfacing functions, GUIs & APIs

(10) Maximizing the system performance to let different processes (tasks or threads) share the resources most efficiently- OS provides protection & security.

STRUCTURE

A system can be assumed to have following layered structure

1. **Application Software : -** Executes as per the applications run on the given system hardware using the interfaces and the system software
2. **Application Programming Interface (API)** :- Provides the interface (for inputs and outputs) between the application software and system software so that it is able to run on the processor using the given system-software
3. **System software:-** This layer gives the system software services other than those provided by the OS service function interface like the device drivers
4. **OS interface** – interface between application software and OS.
5. **OS** – kernel supervisory mode services, file management
6. **Hardware—OS Interface:--** Interfaces to let the functions be executed on the given hardware (processor. memory, ports and devices) of the system
7. **Hardware** :-. Processor(s), memories, buses, interfacing circuits, ports, physical devices, timers, and buses for devices networking.

When using an OS, the processor in the system runs in **two modes**

The two mode are: User and Supervisory Mode Structure.

- **User mode** – user process is permitted to run and use only a subset of functions and instructions in OS. In 'User Mode', the use of hardware resources including memory is not permitted without making a 'System Call'. (A 'System Call' is a call by an OS functions & The OS calls the resources by system call) . The 'user mode function call' is not permitted to read & write into the protected memory allotted to the OS functions, data, stack etc .This protected memory space is also called 'Kernel Space'. So, 'user function call' is not like a 'system call'. Hence execution of 'user functions calls' is slower than the execution of the 'OS function calls' (where the OS function calls run on system call). This is because of the protected access to memory by the functions running in user-space
- **Supervisory mode –** OS runs the privileged functions and instructions in the protected mode and the OS only accesses the hardware resources and the protected area memory. In the supervisory mode, the kernel codes run .So, the 'Kernel-Mode' functions' execute faster than the 'User Mode' functions.

After completing the supervisory functions in the OS, the 'system context' switches back to the user mode.

RTOS' Vs 'Non-Real-Time OS'

An RTOS permits running of the processes, tasks & threads in the 'Supervisory Mode(or kernel mode)

- Therefore, the threads execute fast
- This improves the system performance.

In non-real-time OS, the threads are executed in the user mode

Then the execution slows down due to checks on the code access to the protected kernel space

KERNEL

It's the basic structural unit of any OS in which the memory space of the functions, data and stack are protected from access by any call other than system-call.

It can be defined as a secured unit of an OS that operates in the supervisory mode while the remaining part and application software operates in the user mode.

The kernel has management functions for processes, resources, ISRs, ISTs, files, devices and IO subsystems and network subsystems. The memory or device and file management functions may be outside the kernel in a given OS.

Kernel services in an operating system are -

Process management –

- Creation to deletion – enables creation, activation, running, blocking, resumption, deactivation and deletion and maintenance of process structure at PCB(Process control box)
- Processing resource request :- by process made either by making calls that are known as system calls or by sending messages
- Scheduling – enables scheduling of various process on various terms like priority based scheduling.
- IPC [inter process communication] – processes synchronizing by sending data as messages from one task to another. The OS effectively manages shared memory access by using IPC signals . Exceptions are queues, semaphores, pipes and sockets.
- Memory management allocation and de-allocation.:- It restricts the memory access region for a task. There by dynamic memory allocation
- I/O management. : Character or block I/O management to ensure action such that a parallel port or serial port obtains access to only one task at a time
- Process structure – enables maintenance of process structure at PCB.
- File management – provides management of the creation, deletion, Read () and write () to the files on the secondary memory disk. A file in the embedded system can be in RAM,

where the operations are done in RAM memory in a way identical to file on disk.

- Device management – A physical device management is such that it is accessible to one task or process only at an instant. Device manager components are: (1) device drivers and device ISRs (device interrupt handlers): (ii) resource managers for the devices. Besides physical devices the management of virtual device like pipe or socket is also provided. Virtual devices emulate a hardware device and the virtual device driver send signals similar to the ISR calls by the physical device.

PROCESS MANAGEMENT

At restart of processor in a computer system, an OS is initialized first (boot), and then a process, which can be called initial process, is created (initialization of OS means enabling the use of OS function, which includes to create the processes).

Process management also means 'Task Management' & 'Thread Management' .
Following are the 'Process Management Functions' of the OS kernel.

1) Process Creation to Deletion
2) Process Structure Maintenance
3) Processing the 'Resource Requests'
4) Process Scheduling
5) Inter-Process Communication (IPC)

1. Process Creation –

➢ After reset of the processor in a computer system, an OS is 'initialized' first & Then a process, which can be called as 'Initial Process' is created. (Here, 'initialization of OS' means enabling the use of 'OS Functions', which includes the function to create the processes).

➢ Then the OS is 'started' & that runs an initial process ('Starting the OS' means calling the 'OS Functions', which includes the call to all the created processes before the OS start but after the OS initialization).

➢ The initial process creates subsequent processes

➢ So, the processes can be created hierarchically

➢ OS schedules the processes & provides for 'Context Switching' between the processes & threads.

➢ Creation of a process means specifying the resources for the process such as

- Address Spaces (or Memory Blocks) for the created process

- Stack Memory Area
- Data for the process
- Placing the process initial info. at a Process Control Block (PCB)

➢ The 'Process Manager' allocates a PCB (or TCB- Task Control Block) when it creates the process & later, manages that PCB.

A **PCB or TCB** describes the following.

1) Context: Processor status word, PC, SP and other CPU registers at the instant of the last instruction run executed when the process was left and the processor switched to another process.
2) Process stack pointer.
3) Current state: Is it created, activated or spawned? Is it running? Is it blocked? (spawn means create activate).
4) Addresses that are allocated and that are presently in use.
5) Pointer for the parent process in case there exists a hierarchy of the processes.
6) Pointer to a list of daughter processes (processes lower in the hierarchy).
7) Pointer to a list of resources, which are usable (consumed) only once. For example, input data, memory buffer or pipe, mailbox message, semaphore (there may be producers and consumers of these resources).
8) Pointer to a list of resource type usable more than once: A resource type example is a memory block. Another example is an I/O port. Each resource type will have a count of these types. For example, the number of memory blocks or number of I/O ports
9) Pointer to queue of messages. It is considered as a special case of resources that are usable once. It is because messages from the OS also queue up to be controlled by a process.
10) Pointer to access permissions descriptor for sharing a set of resources globally, and with another process.
11) ID by which identification is made by the process manager.

2. Management of the Created Processes

Process Manager' is a unit of the OS that is the entity responsible for controlling a process execution.

A process manager

- Creates the processes.
- Allocates to each a PCB.

- Manages access to resources .
- Facilitates switching from one process state to another.

- Process manager is a unit of the OS that is responsible for controlling a process execution.
- Process management enables process creation, activation, running, blocking, resumption, deactivation and deletion.
- A process manager facilitates the following.

- Each process of a multiple process (or multitasking or multithreading) system is executed such that a process state can switch from one to another.
- A process does the following sequential execution of the states: 'created', 'ready or activate', 'spawn' (means create and activate), 'running', `blocked' or 'suspended', 'resumed' and 'finished' and 'ready' after 'finish' (when there is an infinite loop in a process) and finally 'deleted'.
- Blocking and resuming can take place several times in a long process. The different OSs make the provisions for possible states between creation and deletion differently.
- The process manager executes a process request for a resource or OS service and then grants that request to let the processes share the resources. For example, an LCD display is a shared resource. The LCD display can be used only by one task or thread at an instance.

3. 'Process Request' for Resource - The process manager executes a process request for a resource or OS service & then grants that request to let the processes share the resources

For eg., an LCD display is a shared resource .The LCD display can be used only by one task or thread at an instance.

The 'Request for a Resource' is the OS Service by a Running Process. A 'running process' make requests by two methods:

(1) Message

(2) System Call

4. Process Scheduling

The process manager

a) Manages the 'Processes' & 'Resources' of the given system

b) Makes it feasible for a process to sequentially (or concurrently execute or To 'block' a process when needing a resource or To 'resume' when the resource becomes available

c) Implements the logical link to the resource manager for resources manages including scheduling of processes on the CPU.

d) Allows specific resources sharing between specified processes only.

e) Allocates the resources as per the resource allocation mechanism of the system.

MEMORY MANAGEMENT

When a process is created, the memory manager allocates the memory addresses (blocks) to it by mapping the process address space. Threads of a process share the memory space of the process.

Memory Management after Initial Allocation

- The memory manager of the OS has to be secure, robust and well protected.
- There must be control such that there are no memory leaks and stack overflows.
- Memory leaks mean attempts to write in the memory block not allocated to a process or data structure.
- Stack overflow means that the stack exceeds the allocated memory block(s) when there is no provision for additional stack space.

The task consists of several fixed size memory blocks. The fixed size memory blocks allocation and de-allocation time takes fixed time (deterministic). Therefore, it leads to a predictable task-performance.

Memory Managing Strategy for a System

I. Fixed blocks allocation - Memory address space is divided into blocks with processes having small address spaces getting a lesser number of blocks and processes with big address spaces getting a larger number of blocks.

II. Dynamic blocks allocation - Memory address space is divided into fixed blocks as above and then later the memory manager later allocates variable size blocs (in units of say 64 or 256 bytes) dynamically allocated from a free (unused) list of memory blocks description table at the different computation phases of a process.

III. Dynamic page allocation - Memory has fixed sized blocks called pages and the memory manager MMU (memory management unit) allocates the pages dynamically with a page descriptor table.

IV. Dynamic data memory allocation - The manager allocates memory dynamically to different data structures like the nodes of a list, queues and stacks.

V. Dynamic address relocation - The manager dynamically allocates the addresses initially bound to the relative addresses. It adds the relative address to address with relocation register. The memory manager now dynamically changes only the contents of relocation register. It also takes into account a limit-defining register so that the relocated addresses are within the limit of available addresses. This is also called run-time dynamic address binding.

VI. Multiprocessor memory allocation - The memory adopts an allocation strategy either shared with tight coupling between two or more processors or shared with loose coupling or multi segmented allocations.

VII. Memory protection to OS functions - Memory protection to the OS functions means that the system call (call to an OS function) and function call in user space are distinct. The OS function code, data and stack are in the protected memory area. It means that when a user function call attempts to write or read in the exclusive memory space allocated to the OS functions, it is blocked and the system generates an error. The memory of kernel functions is distinct and can be addressed only by the systems calls. The memory space is called kernel space.

VIII. Memory protection among the tasks - Memory protection to the tasks means that a task function call cannot attempt to write or read in the exclusive area of memory space allocated to another task. The protection increases the memory requirement for each task and also the execution time of the code of the task.

The memory manager manages the following:

- Use of memory addresses space by a process.
  - Specific mechanisms to share the memory space.
  - Specific mechanisms to restrict sharing of a given memory space and
  - Optimization of the access periods of a memory by using an hierarchy of memories (caches. primary and external secondary magnetic and optical memories).

Remember that the access periods are in the following increasing order: caches, primary and external secondary magnetic and then optical. The memory manager allocates memory to the processes and manages it with appropriate protection. There may be static and dynamic allocations of memory. The manager optimizes the memory needs and memory utilization. An RTOS may disable the support to the dynamic block allocation. MMU support to the dynamic page allocation and dynamic binding as this increases the latency of servicing the tasks and ISRs. An RTOS may or may not support memory protection in order to reduce the latency and memory needs of the processes.

Memory Allocation

- When a process is created, the OS memory manager allocates the memory addresses (blocks) to it by mapping the process-address space.
- Threads of a process share the memory space of the process.
- There may be ‘Static’ & ‘Dynamic’ allocations of memory.

Memory Allocation & De-Allocation

- An OS provides for dynamic memory allocation & de-allocation functions
- Dynamic memory allocation is used for creating memory address space for a ‘Messages Queue’ or a buffer some other purpose during execution of a task.

- Dynamic Memory De-Allocation is used for freeing the memory taken up for the buffer/ queue during execution of the task.
- MMU may support to the ‘Dynamic Page Allocation’
- An RTOS may disable the support to the dynamic block allocation
- This is because dynamic allocation of ‘memory pages’ increases the latency (or Delay) of servicing the tasks & ISRs.

Memory Management after Initial Allocation

There must be control such that there are no ‘Memory Leaks’ & ‘Stack Overflows’(Memory leaks mean attempts to write in the memory block not allocated to a process or data structure) (Stack overflow means that the stack exceeds the allocated memory blocks when there is no provision for additional stack space.

Memory Manager

- The ‘Memory Manager’ allocates memory to the processes & manages it with appropriate protection.
- The manager optimizes the memory needs & memory utilization
- The memory manager of the OS has to be ‘Secure, robust & well protected
- The memory manager manages the following:
  - Use of memory address space by a process.
  - Specific mechanisms to share the memory space,
  - Specific mechanisms to restrict sharing of a given memory space
  - Optimization of the access periods of a memory by using an hierarchy of memories (Caches, Primary & External Secondary Magnetic/ Optical Memories).(The memory access time periods are in the following increasing order: Caches, Primary & External Secondary Magnetic/ Optical Memories).

DEVICE MANAGEMENT

A device manager is the software that manages various device drivers available in the system. An OS device manager provides and executes the modules for managing the devices and their drivers ISRs.

There are number of device drive ISRs for each device in a system, each driver-function of a device (e.g. open, close, read) calls a separate ISR. These calls are made by device manager.

The manager coordinates between application process, driver- and device-controller.

A process sends a request to the driver functions by an interrupt using SWI (Software Interrupt) and the driver provides the actions on calling and executing the ISR.

The device manager polls the requests at the devices and the actions occur as per their priorities. The device manager manages I/O interrupt (requests) queues.

The device manager creates, appropriate kernel interface and API, and that activates the control register-specific actions of the device.

An OS device manager provides and executes the modules for managing the devices and their driver ISRs.

1) It manages the physical as well as virtual devices like the pipes and sockets through a common strategy.

2) Device management has three standard approaches to three types of device drivers:

(i) programmed I/0s by polling the service need from each device.

(ii) interrupt(s) from the device driver ISR and

(iii) DMA operation used by the devices to access the memory. Most common is the use of device driver ISRs.

Set of Command functions for Device Management

<u>Create and Open</u> - create is for creating and open is for creating (if not created earlier) and configuring and initializing the device.

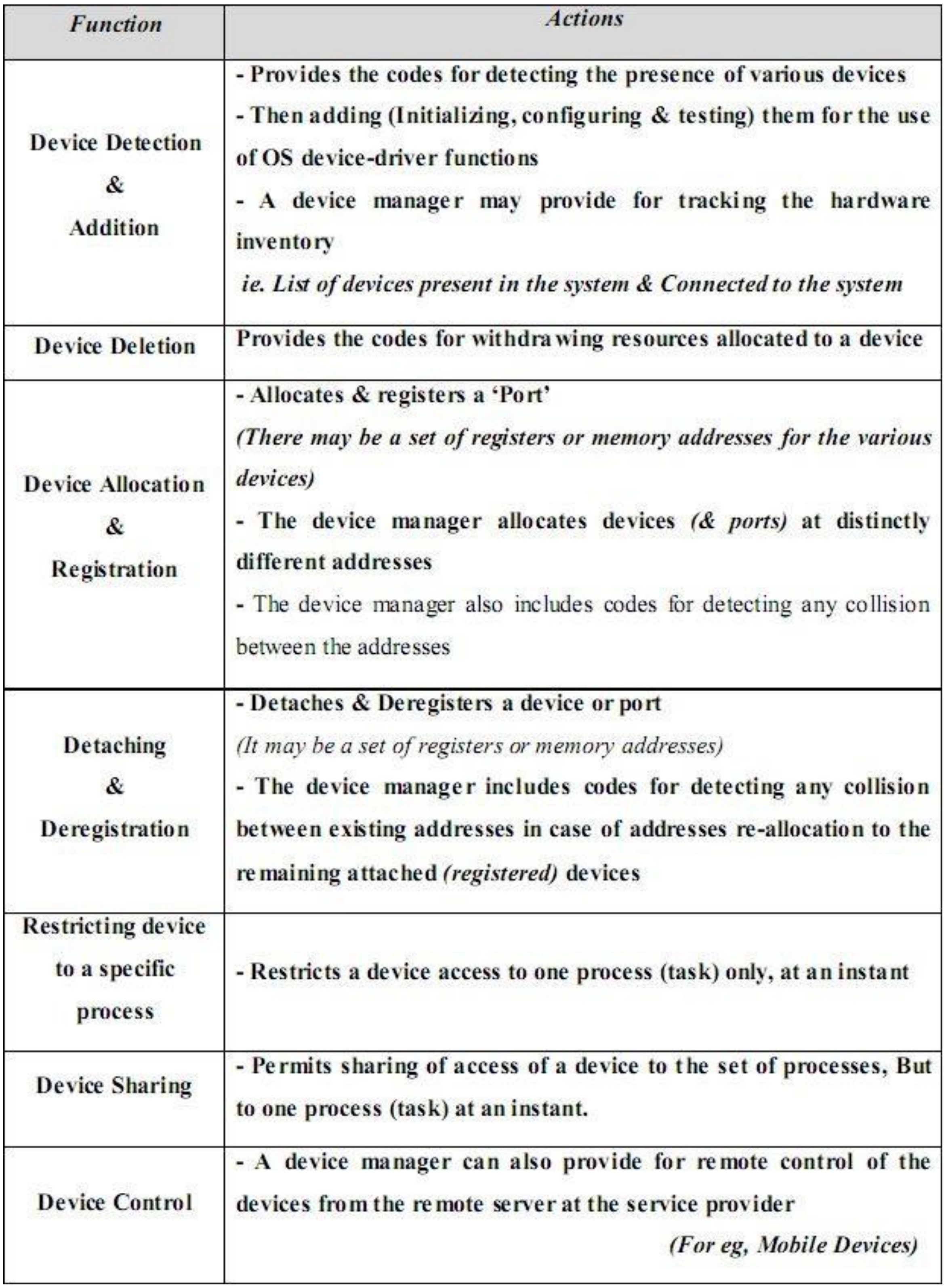
<u>Write</u> - Write into the device buffer or send output from the device and advance the pointer (cursor).

<u>Read</u> - Read from the device buffer or read input from the device and advance the pointer (cursor).

<u>Ioctl</u> - Specified device configured for specific functions and given specific parameters.

<u>Close and delete</u> - close is for de-registering the device from the system and delete is for close (if not closed earlier) and detaching the device.

Function of device manager



Function	*Actions*
Device Detection & Addition	**- Provides the codes for detecting the presence of various devices** **- Then adding (Initializing, configuring & testing) them for the use of OS device-driver functions** **- A device manager may provide for tracking the hardware inventory** ***ie. List of devices present in the system & Connected to the system***
Device Deletion	**Provides the codes for withdrawing resources allocated to a device**
Device Allocation & Registration	**- Allocates & registers a 'Port'** ***(There may be a set of registers or memory addresses for the various devices)*** **- The device manager allocates devices *(& ports)* at distinctly different addresses** - The device manager also includes codes for detecting any collision between the addresses
Detaching & Deregistration	**- Detaches & Deregisters a device or port** *(It may be a set of registers or memory addresses)* **- The device manager includes codes for detecting any collision between existing addresses in case of addresses re-allocation to the remaining attached *(registered)* devices**
Restricting device to a specific process	**- Restricts a device access to one process (task) only, at an instant**
Device Sharing	**- Permits sharing of access of a device to the set of processes, But to one process (task) at an instant.**
Device Control	**- A device manager can also provide for remote control of the devices from the remote server at the service provider** ***(For eg, Mobile Devices)***

Functions of OS Device Manager

Function	*Actions*
Device Access Management	**(i) Sequential Access** **(ii) Random Access** **(iii) Semi-random Access** **(iv) Serial communication may be by UART or USB or CAN** ***- The device manager provides the necessary interface***
Device Buffer Management	**- Device hardware may merely have a single byte buffer or double buffer or 8-byte buffer** ***- A device 'Buffer Manager' uses a memory manager to buffer the I/O data streams from the device that sends the data & manages computations without wait while the buffer receives the data at a slow rate***
Device Queue Management	**- Device manager organizes device IO data streams of device queue, circular queues or blocks of queues**
Device Driver	**- Device manager manages the device drivers** ***- A 'Device Driver' is the software for interface with the device hardware through the buses in the one hand & for the interface with the OS and application programs on other hand*** **- The software commands for read( ) & write( ) enables the read & write functions through the ISRs *(via SWI)*** **- The interface software to OS enables the Creation. Connection, Binding, Opening & Closing of the device**
Device Driver Updating	**- Device manager can also provide for updating the driver software from the internet & uploading the new device functions, which become available at a later date**
Backup & Restoration	**- Device manager can also provide for the backup & restoration for drivers**

FILE SYSTEM ORGANIZATION & IMPLEMENTATION

A ‘File’ is a named entity on a memory card, magnetic/optical disc or system memory.

- A file contains the data, characters & text; It may also have a mix of these .Each OS may have differing abstractions of a file.
- A ‘File’ may be a named entity that is a ‘Structural Record’ on a disk having ‘random access’ in the system.
- A ‘File’ can be a structural record on a RAM (similar to that in a disk) and it may also be called ‘RAM disk’.
- A ‘File’ may be an unstructured record of bits or bytes.
- A ‘File’ is a virtual device such as a ‘pipe’ like device for inter-process communication (IPC).
- A ‘File’ device may be a ‘socket’ like device for inter-process communication (IPC).

File in Embedded Systems

A file in embedded system can be in the RAM(In most cases, embedded systems doesn’t contain disk type memories).

So, the file operations are done in RAM memory in a way identical to with a file on the Disk. A file can also be in flash memory like memory card or pen drive.

File Management Functions

It is necessary to organize the files in a systematic way & to have a set of command functions.

File-management functions provide functions for creation, deletion, read ( ), write ( ) to the files.

UNIX Command	*Action(s)*
open ( )	Function for creating the file
write ( )	Writing into the file
read ( )	Reading from the file
Seek List or Set the file pointer	Setting the pointer for the appropriate place in the file for the next read or write
close ( )	Closing the file

File systems

There are two types of file systems.

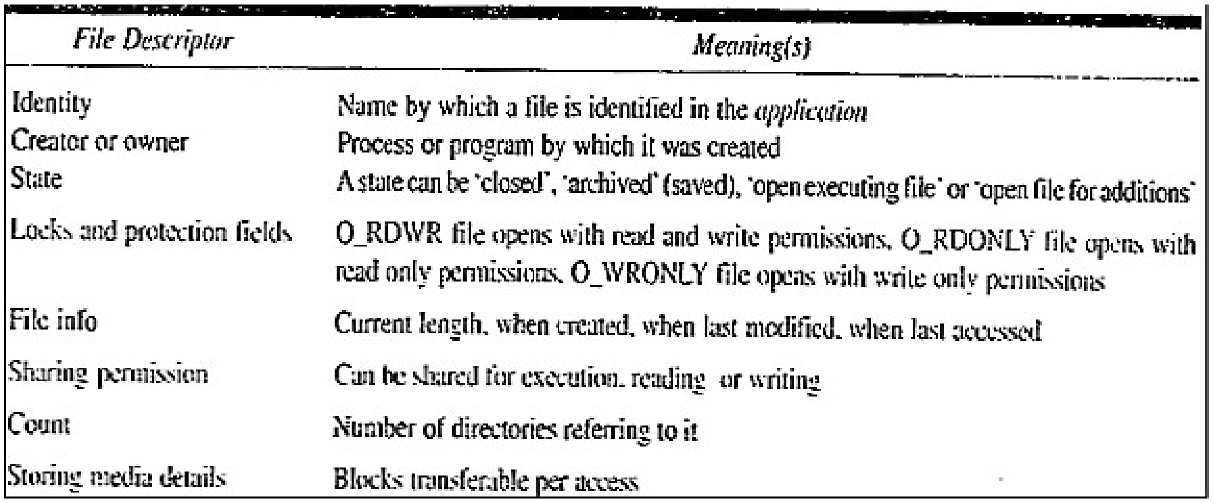
(1) Block File System - Here an OS application generates records to be saved into the memory. These are first structured into a suitable format & then translated into blocks (or block streams).

A 'File Pointer' (record) points to a block from the start to the end of the file .

(2) Byte-Stream File System - Here the application generates record streams.These streams are to be saved into the memory. These are first structured into a suitable format & then translated into byte streams. A file pointer (byte index) points to a byte from the start index = 0 to 'N-l' in a file of N bytes.

File Descriptor

- Just as each process has a processor descriptor (or PCB- process control block), a file system has a 'Data Structure', called 'File Descriptor', fd .
- The structure differs from one 'File Manager' to another.
- File descriptor, fd, for a file is an integer, which returns on opening a file.
- The, fd, points to the data structure of the file.
- The 'fd' is usable till the closing of the file.



File Descriptor	*Meaning(s)*
Identity	Name by which a file is identified in the *application*
Creator or owner	Process or program by which it was created
State	A state can be 'closed', 'archived' (saved), 'open executing file' or 'open file for additions'
Locks and protection fields	O_RDWR file opens with read and write permissions, O_RDONLY file opens with read only permissions, O_WRONLY file opens with write only permissions
File info	Current length, when created, when last modified, when last accessed
Sharing permission	Can be shared for execution, reading or writing
Count	Number of directories referring to it
Storing media details	Blocks transferable per access

MICRO C/OS-II RTOS (MUCOS)

One of the popular RTOS for an embedded system is µC/OS-II. µC/OS-II is a free ware for non commercial use

6.9 µC/OS-II System level Functions

A set of system level functions in MUCOS are as follows.

1. Initiating the operating system before starting the use of the RTOS functions.

Function void OSInit (void) is used to initiate the operating system. Its use is compulsory before calling any OS kernel functions. It returns no parameter.

2. Starting the use of RTOS multitasking functions and running the tasks

Function void OSStart (void) is used to start the initiated operating system and created tasks. Its use is compulsory for the multitasking OS kernel operations. It returns no parameter.

3. Starling the use of RTOS System clock

Function void OSTicklnit (void) is used to initiate the system clock ticks and interrupts at regular intervals as per OS_TICKS_PER_SEC predefined during configuring the MUCOS, Its use is compulsory for the multitasking OS kernel operations when the timer functions are to be used. It returns or passes no parameter.

4, Sending message to RTOS kernel for taking control at the start of an 1SR

Function void OSIntEnter (void) is used at the start of an ISR, It is for sending a message to RTOS kernel for taking control, Its use is compulsory to let the multitasking OS kernel, control the nesting of the ISRs in case of occurrences of multiple interrupts of varying priorities, It returns no parameter.

5. Sending a message to RTOS kernel for quitting the control at the return from an 1SR

Function void OSIntExit (void) is used just before the return from the running ISR. It is for sending a message to RTOS kernel for quitting control from the nesting loop, Its use is compulsory to let the OS kernel quit the ISR from the nested loop of the ISRs. It returns no parameter.

6. Sending a message to RTOS kernel for taking control at the start of a critical section

Macro-function OS_ENTER_CRITICAL is used at the start of an ISR. It is for sending a message to RTOS kernel for disabling the interrupts. Its use is compulsory to let the OS kernel take note of and disable the interrupts of the system. It returns no parameter.

7. Sending a message to RTOS kernel for quitting the control at the return from a critical section

Macro-function OS_EXIT_CRITICAL is used just before the return from the critical section. It is for sending a message to RTOS kernel for quitting control from the section. Its use is compulsory to let the OS kernel quit the section and enable the interrupts to the system. It returns parameter.

8. Locking OS scheduler : OSSchedlock ()f disables preemption by a higher priority task.

   This function, inhibits preemption by higher priority task. but does not inhibit interrupt. If an interupt occur then locking enables return of OS control to that task, which executes this function. The control returns to the task after any ISR completers

9. Unlocking OS schedule. OSScheduleunlock() enables preemption by higher priority' task. enables return of OS control to the high priority cask after the execution of OSScheduledUnlock. In case of any interrupt occurring after executing OSScheduleunlock and after the end of the ISR, the higher-priority task. which is ready will execute on return hum the ISR.

BEYOND THE SYLLABUS

INTERNET OF THINGS

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

IoT (Internet of Things) is an advanced automation and analytics system which exploits networking, sensing, big data, and artificial intelligence technology to deliver complete systems for a product or service. These systems allow greater transparency, control, and performance when applied to any industry or system.

IoT systems have applications across industries through their unique flexibility and ability to be suitable in any environment. They enhance data collection, automation, operations, and much more through smart devices and powerful enabling technology.

This tutorial aims to provide you with a thorough introduction to IoT. It introduces the key concepts of IoT, necessary in using and deploying IoT systems.

Audience

This tutorial targets IT professionals, students, and management professionals who want a solid grasp of essential IoT concepts. After completing this tutorial, you will achieve intermediate expertise in IoT and a high level of comfort with IoT concepts and systems.

Prerequisites

This tutorial assumes general knowledge of networking, sensing, databases, programming, and related technology. It also assumes familiarity with business concepts and marketing.

Copyright & Disclaimer

Table of Contents

1. IoT – Overview

IoT systems allow users to achieve deeper automation, analysis, and integration within a system. They improve the reach of these areas and their accuracy. IoT utilizes existing and emerging technology for sensing, networking, and robotics.

IoT exploits recent advances in software, falling hardware prices, and modern attitudes towards technology. Its new and advanced elements bring major changes in the delivery of products, goods, and services; and the social, economic, and political impact of those changes.

IoT – Key Features

The most important features of IoT include artificial intelligence, connectivity, sensors, active engagement, and small device use. A brief review of these features is given below:

- **AI –** IoT essentially makes virtually anything "smart", meaning it enhances every aspect of life with the power of data collection, artificial intelligence algorithms, and networks. This can mean something as simple as enhancing your refrigerator and cabinets to detect when milk and your favorite cereal run low, and to then place an order with your preferred grocer.
- **Connectivity –** New enabling technologies for networking, and specifically IoT networking, mean networks are no longer exclusively tied to major providers. Networks can exist on a much smaller and cheaper scale while still being practical. IoT creates these small networks between its system devices.
- **Sensors –** IoT loses its distinction without sensors. They act as defining instruments which transform IoT from a standard passive network of devices into an active system capable of real-world integration.
- **Active Engagement –** Much of today's interaction with connected technology happens through passive engagement. IoT introduces a new paradigm for active content, product, or service engagement.
- **Small Devices –** Devices, as predicted, have become smaller, cheaper, and more powerful over time. IoT exploits purpose-built small devices to deliver its precision, scalability, and versatility.

IoT – Advantages

The advantages of IoT span across every area of lifestyle and business. Here is a list of some of the advantages that IoT has to offer:

- **Improved Customer Engagement –** Current analytics suffer from blind-spots and significant flaws in accuracy; and as noted, engagement remains passive. IoT completely transforms this to achieve richer and more effective engagement with audiences.
- **Technology Optimization –** The same technologies and data which improve the customer experience also improve device use, and aid in more potent improvements to technology. IoT unlocks a world of critical functional and field data.

- **Reduced Waste –** IoT makes areas of improvement clear. Current analytics give us superficial insight, but IoT provides real-world information leading to more effective management of resources.
- **Enhanced Data Collection –** Modern data collection suffers from its limitations and its design for passive use. IoT breaks it out of those spaces, and places it exactly where humans really want to go to analyze our world. It allows an accurate picture of everything.

IoT – Disadvantages

Though IoT delivers an impressive set of benefits, it also presents a significant set of challenges. Here is a list of some its major issues:

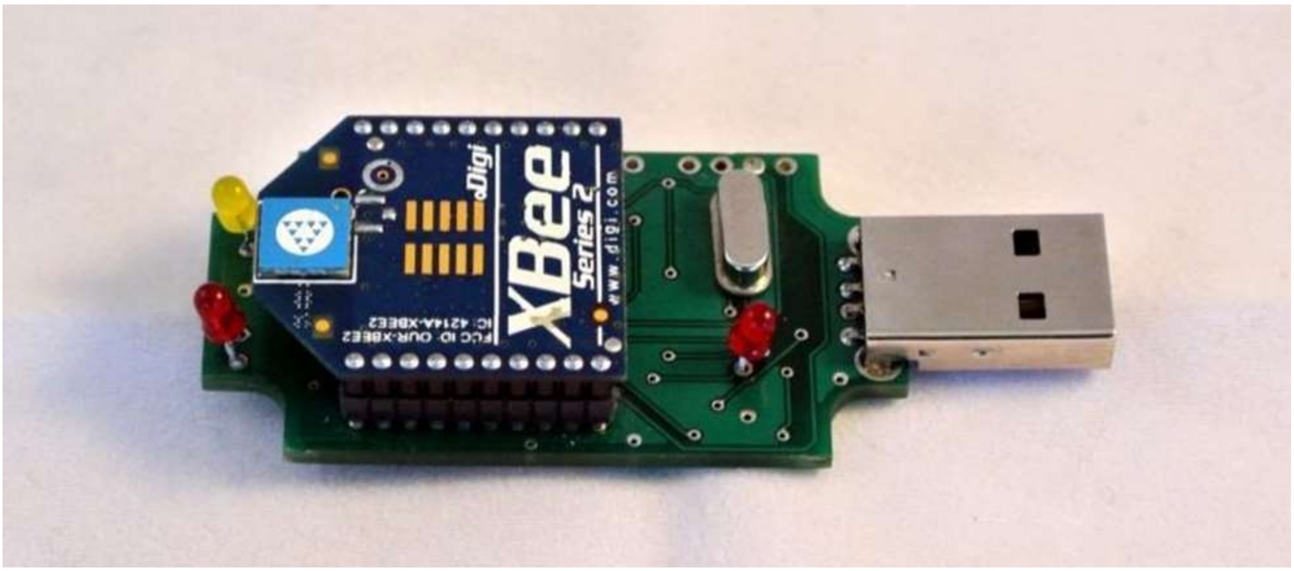
- **Security –** IoT creates an ecosystem of constantly connected devices communicating over networks. The system offers little control despite any security measures. This leaves users exposed to various kinds of attackers.
- **Privacy –** The sophistication of IoT provides substantial personal data in extreme detail without the user's active participation.
- **Complexity –** Some find IoT systems complicated in terms of design, deployment, and maintenance given their use of multiple technologies and a large set of new enabling technologies.
- **Flexibility –** Many are concerned about the flexibility of an IoT system to integrate easily with another. They worry about finding themselves with several conflicting or locked systems.
- **Compliance –** IoT, like any other technology in the realm of business, must comply with regulations. Its complexity makes the issue of compliance seem incredibly challenging when many consider standard software compliance a battle.

2. IoT – Hardware

The hardware utilized in IoT systems includes devices for a remote dashboard, devices for control, servers, a routing or bridge device, and sensors. These devices manage key tasks and functions such as system activation, action specifications, security, communication, and detection to support-specific goals and actions.

IoT – Sensors

The most important hardware in IoT might be its sensors. These devices consist of energy modules, power management modules, RF modules, and sensing modules. RF modules manage communications through their signal processing, WiFi, ZigBee, Bluetooth, radio transceiver, duplexer, and BAW.



The sensing module manages sensing through assorted active and passive measurement devices. Here is a list of some of the measurement devices used in IoT:

Devices	
accelerometers	temperature sensors
magnetometers	proximity sensors
gyroscopes	image sensors
acoustic sensors	light sensors
pressure sensors	gas RFID sensors
humidity sensors	micro flow sensors

Wearable Electronics

Wearable electronic devices are small devices worn on the head, neck, arms, torso, and feet.

Smartwatches not only help us stay connected, but as a part of an IoT system, they allow access needed for improved productivity.

Current smart wearable devices include:

- **Head** – Helmets, glasses
- **Neck** – Jewelry, collars
- **Arm** – Watches, wristbands, rings
- **Torso** – Clothing, backpacks
- **Feet** – Socks, shoes

Smart glasses help us enjoy more of the media and services we value, and when part of an IoT system, they allow a new approach to productivity.

Standard Devices

The desktop, tablet, and cellphone remain integral parts of IoT as the command center and remotes.

- The **desktop** provides the user with the highest level of control over the system and its settings.

- The **tablet** provides access to the key features of the system in a way resembling the desktop, and also acts as a remote.

- The **cellphone** allows some essential settings modification and also provides remote functionality.

Other key connected devices include standard network devices like **routers** and **switches**.

3. IoT – Software

IoT software addresses its key areas of networking and action through platforms, embedded systems, partner systems, and middleware. These individual and master applications are responsible for data collection, device integration, real-time analytics, and application and process extension within the IoT network. They exploit integration with critical business systems (e.g., ordering systems, robotics, scheduling, and more) in the execution of related tasks.

Data Collection

This software manages sensing, measurements, light data filtering, light data security, and aggregation of data. It uses certain protocols to aid sensors in connecting with real-time, machine-to-machine networks. Then it collects data from multiple devices and distributes it in accordance with settings. It also works in reverse by distributing data over devices. The system eventually transmits all collected data to a central server.

Device Integration

Software supporting integration binds (dependent relationships) all system devices to create the body of the IoT system. It ensures the necessary cooperation and stable networking between devices. These applications are the defining software technology of the IoT network because without them, it is not an IoT system. They manage the various applications, protocols, and limitations of each device to allow communication.

Real-Time Analytics

These applications take data or input from various devices and convert it into viable actions or clear patterns for human analysis. They analyze information based on various settings and designs in order to perform automation-related tasks or provide the data required by industry.

Application and Process Extension

These applications extend the reach of existing systems and software to allow a wider, more effective system. They integrate predefined devices for specific purposes such as allowing certain mobile devices or engineering instruments access. It supports improved productivity and more accurate data collection.

4. IoT – Technology and Protocols

IoT primarily exploits standard protocols and networking technologies. However, the major enabling technologies and protocols of IoT are RFID, NFC, low-energy Bluetooth, low-energy wireless, low-energy radio protocols, LTE-A, and WiFi-Direct. These technologies support the specific networking functionality needed in an IoT system in contrast to a standard uniform network of common systems.

NFC and RFID

RFID (radio-frequency identification) and NFC (near-field communication) provide simple, low-energy, and versatile options for identity and access tokens, connection bootstrapping, and payments.

- RFID technology employs 2-way radio transmitter-receivers to identify and track tags associated with objects.

- NFC consists of communication protocols for electronic devices, typically a mobile device and a standard device.

Low-Energy Bluetooth

This technology supports the low-power, long-use need of IoT function while exploiting a standard technology with native support across systems.

Low-Energy Wireless

This technology replaces the most power hungry aspect of an IoT system. Though sensors and other elements can power down over long periods, communication links (i.e., wireless) must remain in listening mode. Low-energy wireless not only reduces consumption, but also extends the life of the device through less use.

Radio Protocols

ZigBee, Z-Wave, and Thread are radio protocols for creating low-rate private area networks. These technologies are low-power, but offer high throughput unlike many similar options. This increases the power of small local device networks without the typical costs.

LTE-A

LTE-A, or LTE Advanced, delivers an important upgrade to LTE technology by increasing not only its coverage, but also reducing its latency and raising its throughput. It gives IoT a tremendous power through expanding its range, with its most significant applications being vehicle, UAV, and similar communication.

WiFi-Direct

WiFi-Direct eliminates the need for an access point. It allows P2P (peer-to-peer) connections with the speed of WiFi, but with lower latency. WiFi-Direct eliminates an element of a network that often bogs it down, and it does not compromise on speed or throughput.

5. IoT – Common Uses

IoT has applications across all industries and markets. It spans user groups from those who want to reduce energy use in their home to large organizations who want to streamline their operations. It proves not just useful, but nearly critical in many industries as technology advances and we move towards the advanced automation imagined in the distant future.

Engineering, Industry, and Infrastructure

Applications of IoT in these areas include improving production, marketing, service delivery, and safety. IoT provides a strong means of monitoring various processes; and real transparency creates greater visibility for improvement opportunities.

The deep level of control afforded by IoT allows rapid and more action on those opportunities, which include events like obvious customer needs, nonconforming product, malfunctions in equipment, problems in the distribution network, and more.

Example

Joan runs a manufacturing facility that makes shields for manufacturing equipment. When regulations change for the composition and function of the shields, the new appropriate requirements are automatically programmed in production robotics, and engineers are alerted about their approval of the changes.

Government and Safety

IoT applied to government and safety allows improved law enforcement, defense, city planning, and economic management. The technology fills in the current gaps, corrects many current flaws, and expands the reach of these efforts. For example, IoT can help city planners have a clearer view of the impact of their design, and governments have a better idea of the local economy.

Example

Joan lives in a small city. She's heard about a recent spike in crime in her area, and worries about coming home late at night.

Local law enforcement has been alerted about the new "hot" zone through system flags, and they've increases their presence. Area monitoring devices have detected suspicious behavior, and law enforcement has investigated these leads to prevent crimes.

Home and Office

In our daily lives, IoT provides a personalized experience from the home to the office to the organizations we frequently do business with. This improves our overall satisfaction, enhances productivity, and improves our health and safety. For example, IoT can help us customize our office space to optimize our work.

Example

Joan works in advertising. She enters her office, and it recognizes her face. It adjusts the lighting and temperature to her preference. It turns on her devices and opens applications to her last working points.

Her office door detected and recognized a colleague visiting her office multiple times before she arrived. Joan's system opens this visitor's messages automatically.

Health and Medicine

IoT pushes us towards our imagined future of medicine which exploits a highly integrated network of sophisticated medical devices. Today, IoT can dramatically enhance medical research, devices, care, and emergency care. The integration of all elements provides more accuracy, more attention to detail, faster reactions to events, and constant improvement while reducing the typical overhead of medical research and organizations.

Example

Joan is a nurse in an emergency room. A call has come in for a man wounded in an altercation. The system recognized the patient and pulls his records. On the scene, paramedic equipment captures critical information automatically sent to the receiving parties at the hospital. The system analyzes the new data and current records to deliver a guiding solution. The status of the patient is updated every second in the system during his transport. The system prompts Joan to approve system actions for medicine distribution and medical equipment preparation.

6. IoT – Media, Marketing, & Advertising

The applications of IoT in media and advertising involve a customized experience in which the system analyzes and responds to the needs and interests of each customer. This includes their general behavior patterns, buying habits, preferences, culture, and other characteristics.

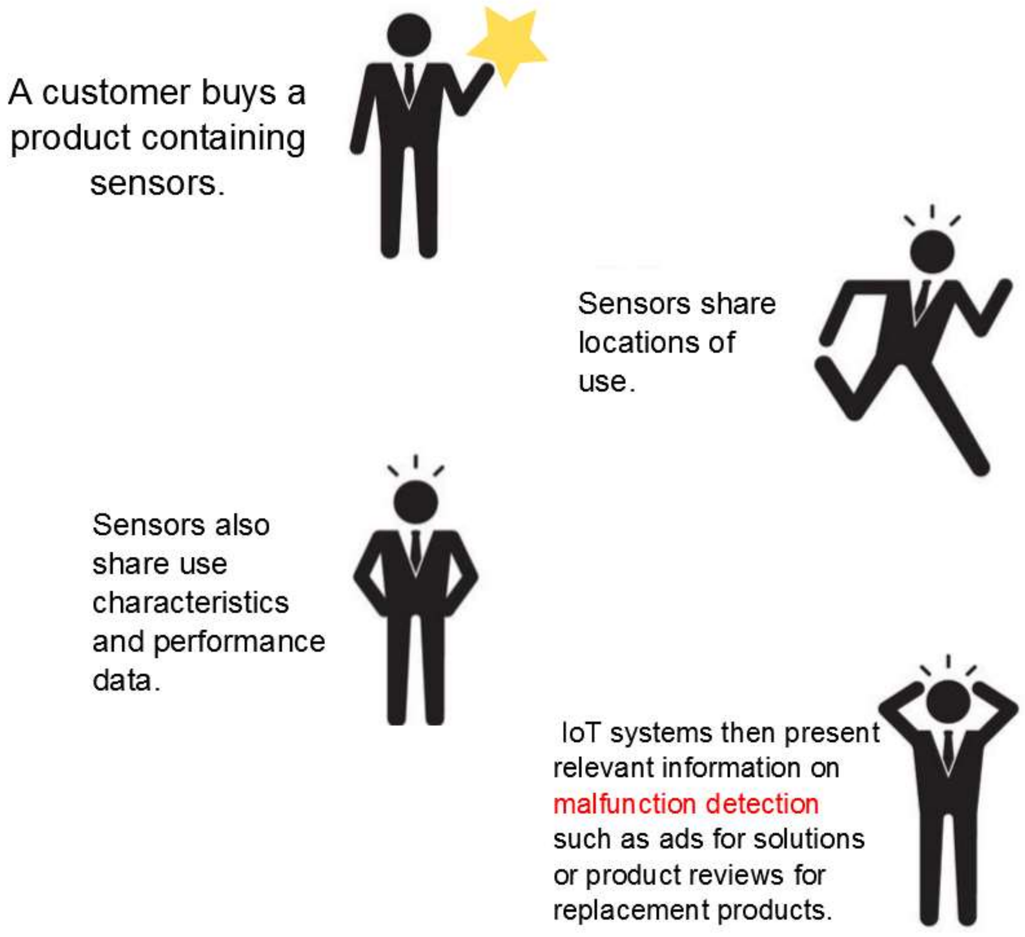
Marketing and Content Delivery

IoT functions in a similar and deeper way to current technology, analytics, and big data. Existing technology collects specific data to produce related metrics and patterns over time, however, that data often lacks depth and accuracy. IoT improves this by observing more behaviors and analyzing them differently.

- This leads to more information and detail, which delivers more reliable metrics and patterns.

- It allows organizations to better analyze and respond to customer needs or preferences.

- It improves business productivity and strategy, and improves the consumer experience by only delivering relevant content and solutions.


A customer buys a product containing sensors.
Sensors share locations of use.
Sensors also share use characteristics and performance data.
IoT systems then present relevant information on malfunction detection such as ads for solutions or product reviews for replacement products.

Improved Advertising

Current advertising suffers from excess and poor targeting. Even with today's analytics, modern advertising fails. IoT promises different and personalized advertising rather than one-size-fits-all strategies. It transforms advertising from noise to a practical part of life because consumers interact with advertising through IoT rather than simply receiving it. This makes advertising more functional and useful to people searching the marketplace for solutions or wondering if those solutions exist.

7. IoT – Environmental Monitoring

The applications of IoT in environmental monitoring are broad: environmental protection, extreme weather monitoring, water safety, endangered species protection, commercial farming, and more. In these applications, sensors detect and measure every type of environmental change.

Air and Water Pollution

Current monitoring technology for air and water safety primarily uses manual labor along with advanced instruments, and lab processing. IoT improves on this technology by reducing the need for human labor, allowing frequent sampling, increasing the range of sampling and monitoring, allowing sophisticated testing on-site, and binding response efforts to detection systems. This allows us to prevent substantial contamination and related disasters.

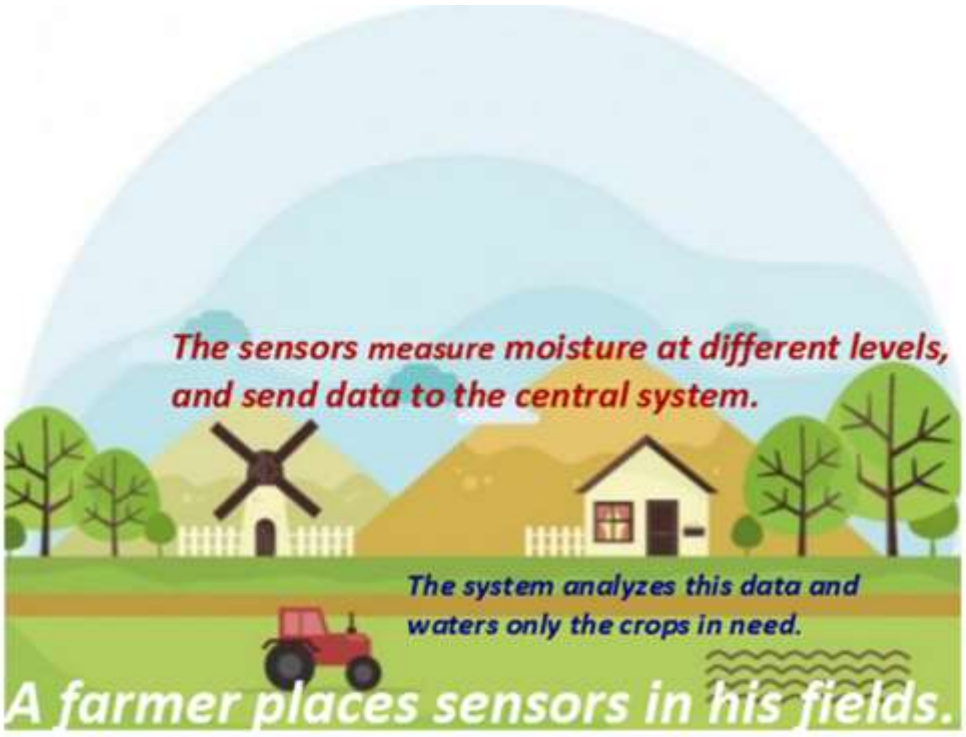
Extreme Weather

Though powerful, advanced systems currently in use allow deep monitoring, they suffer from using broad instruments, such as radar and satellites, rather than more granular solutions. Their instruments for smaller details lack the same accurate targeting of stronger technology.

New IoT advances promise more fine-grained data, better accuracy, and flexibility. Effective forecasting requires high detail and flexibility in range, instrument type, and deployment. This allows early detection and early responses to prevent loss of life and property.

Commercial Farming

Today's sophisticated commercial farms have exploited advanced technology and biotechnology for quite some time, however, IoT introduces more access to deeper automation and analysis.


The sensors measure moisture at different levels, and send data to the central system.
The system analyzes this data and waters only the crops in need.
A farmer places sensors in his fields.

Much of commercial farming, like weather monitoring, suffers from a lack of precision and requires human labor in the area of monitoring. Its automation also remains limited.

IoT allows operations to remove much of the human intervention in system function, farming analysis, and monitoring. Systems detect changes to crops, soil, environment, and more. They optimize standard processes through analysis of large, rich data collections. They also prevent health hazards (e.g., *e. coli*) from happening and allow better control.

8. IoT – Manufacturing Applications

Manufacturing technology currently in use exploits standard technology along with modern distribution and analytics. IoT introduces deeper integration and more powerful analytics. This opens the world of manufacturing in a way never seen before, as organizations become fully-developed for product delivery rather than a global network of suppliers, makers, and distributors loosely tied together.

Intelligent Product Enhancements

Much like IoT in content delivery, IoT in manufacturing allows richer insight in real-time. This dramatically reduces the time and resources devoted to this one area, which traditionally requires heavy market research before, during, and well after the products hit the market.

IoT also reduces the risks associated with launching new or modified products because it provides more reliable and detailed information. The information comes directly from market use and buyers rather than assorted sources of varied credibility.

Dynamic Response to Market Demands

Supplying the market requires maintaining a certain balance impacted by a number of factors such as economy state, sales performance, season, supplier status, manufacturing facility status, distribution status, and more. The expenses associated with supply present unique challenges given today's global partners. The associated potential or real losses can dramatically impact business and future decisions.

IoT manages these areas through ensuring fine details are managed more at the system level rather than through human evaluations and decisions. An IoT system can better assess and control the supply chain (with most products), whether demands are high or low.

Lower Costs, Optimized Resource Use, and Waste Reduction

IoT offers a replacement for traditional labor and tools in a production facility and in the overall chain which cuts many previously unavoidable costs; for example, maintenance checks or tests traditionally requiring human labor can be performed remotely with instruments and sensors of an IoT system.

IoT also enhances operation analytics to optimize resource use and labor, and eliminate various types of waste, e.g., energy and materials. It analyzes the entire process from the source point to its end, not just the process at one point in a particular facility, which allows improvement to have a more substantial impact. It essentially reduces waste throughout the network, and returns those savings throughout.

This XRS relay box connects all truck devices (e.g., diagnostics and driver cell) to the XRS fleet management supporting software, which allows data collection.

Improved Facility Safety

A typical facility suffers from a number of health and safety hazards due to risks posed by processes, equipment, and product handling. IoT aids in better control and visibility. Its monitoring extends throughout the network of devices for not only performance, but for dangerous malfunctions and usage. It aids (or performs) analysis and repair, or correction, of critical flaws.

Product Safety

Even the most sophisticated system cannot avoid malfunctions, nonconforming product, and other hazards finding their way to market. Sometimes these incidents have nothing to do with the manufacturing process, and result from unknown conflicts.

In manufacturing, IoT helps in avoiding recalls and controlling nonconforming or dangerous product distribution. Its high level of visibility, control, and integration can better contain any issues that appear.

9. IoT – Energy Applications

The optimization qualities of IoT in manufacturing also apply to energy consumption. IoT allows a wide variety of energy control and monitoring functions, with applications in devices, commercial and residential energy use, and the energy source. Optimization results from the detailed analysis previously unavailable to most organizations and individuals.

Residential Energy

The rise of technology has driven energy costs up. Consumers search for ways to reduce or control consumption. IoT offers a sophisticated way to analyze and optimize use not only at device level, but throughout the entire system of the home. This can mean simple switching off or dimming of lights, or changing device settings and modifying multiple home settings to optimize energy use.

IoT can also discover problematic consumption from issues like older appliances, damaged appliances, or faulty system components. Traditionally, finding such problems required the use of often multiple professionals.

Commercial Energy

Energy waste can easily and quietly impact business in a major way, given the tremendous energy needs of even small organizations. Smaller organizations wrestle with balancing costs of business while delivering a product with typically smaller margins, and working with limited funding and technology. Larger organizations must monitor a massive, complex ecosystem of energy use that offers few simple, effective solutions for energy use management.

A smart-meter still requires a reader to visit the site. This automated meter reader makes visits unnecessary, and also allows energy companies to bill based on real-time data instead of estimates over time.

IoT simplifies the process of energy monitoring and management while maintaining a low cost and high level of precision. It addresses all points of an organization's consumption across devices. Its depth of analysis and control provides organizations with a strong means of managing their consumption for cost shaving and output optimization. IoT systems discover energy issues in the same way as functional issues in a complex business network, and provide solutions.

Reliability

The analytics and action delivered by IoT also help to ensure system reliability. Beyond consumption, IoT prevents system overloads or throttling. It also detects threats to system performance and stability, which protects against losses such as downtime, damaged equipment, and injuries.

10. IoT – Healthcare Applications

IoT systems applied to healthcare enhance existing technology, and the general practice of medicine. They expand the reach of professionals within a facility and far beyond it. They increase both the accuracy and size of medical data through diverse data collection from large sets of real-world cases. They also improve the precision of medical care delivery through more sophisticated integration of the healthcare system.

Research

Much of current medical research relies on resources lacking critical real-world information. It uses controlled environments, volunteers, and essentially leftovers for medical examination. IoT opens the door to a wealth of valuable information through real-time field data, analysis, and testing.

IoT can deliver relevant data superior to standard analytics through integrated instruments capable of performing viable research. It also integrates into actual practice to provide more key information. This aids in healthcare by providing more reliable and practical data, and better leads; which yields better solutions and discovery of previously unknown issues.

It also allows researchers to avoid risks by gathering data without manufactured scenarios and human testing.

Devices

Current devices are rapidly improving in precision, power, and availability; however, they still offer less of these qualities than an IoT system integrating the right system effectively. IoT unlocks the potential of existing technology, and leads us toward new and better medical device solutions.

IoT closes gaps between equipment and the way we deliver healthcare by creating a logical system rather than a collection of tools. It then reveals patterns and missing elements in healthcare such as obvious necessary improvements or huge flaws.

The ClearProbe portable connected ultrasound device can use any computer anywhere as a supporting machine. The device sends all imaging records to the master system.

Care

Perhaps the greatest improvement IoT brings to healthcare is in the actual practice of medicine because it empowers healthcare professionals to better use their training and knowledge to solve problems. They utilize far better data and equipment, which gives them a window into blind spots and supports more swift, precise actions. Their decision-making is no longer limited by the disconnects of current systems, and bad data.

IoT also improves their professional development because they actually exercise their talent rather than spending too much time on administrative or manual tasks. Their organizational decisions also improve because technology provides a better vantage point.

Medical Information Distribution

One of the challenges of medical care is the distribution of accurate and current information to patients. Healthcare also struggles with guidance given the complexity of following guidance. IoT devices not only improve facilities and professional practice, but also health in the daily lives of individuals.

IoT devices give direct, 24/7 access to the patient in a less intrusive way than other options. They take healthcare out of facilities and into the home, office, or social space. They empower individuals in attending to their own health, and allow providers to deliver better and more granular care to patients. This results in fewer accidents from miscommunication, improved patient satisfaction, and better preventive care.

Emergency Care

The advanced automation and analytics of IoT allows more powerful emergency support services, which typically suffer from their limited resources and disconnect with the base facility. It provides a way to analyze an emergency in a more complete way from miles away. It also gives more providers access to the patient prior to their arrival. IoT gives providers critical information for delivering essential care on arrival. It also raises the level of care available to a patient received by emergency professionals. This reduces the associated losses, and improves emergency healthcare.

11. IoT – Building/Housing Applications

IoT applied to buildings and various structures allows us to automate routine residential and commercial tasks and needs in a way that dramatically improves living and working environments. This, as seen with manufacturing and energy applications, reduces costs, enhances safety, improves individual productivity, and enhances quality of life.

Environment and Conditioning

One of the greatest challenges in the engineering of buildings remains management of environment and conditions due to many factors at work. These factors include building materials, climate, building use, and more. Managing energy costs receives the most attention, but conditioning also impacts the durability and state of the structure.

IoT aids in improving structure design and managing existing structures through more accurate and complete data on buildings. It provides important engineering information such as how well a material performs as insulation in a particular design and environment.

Health and Safety

Buildings, even when constructed with care, can suffer from certain health and safety issues. These issues include poor performing materials, flaws that leave the building vulnerable to extreme weather, poor foundations, and more.

The Boss 220 smart plug allows the user to monitor, control, optimize, and automate all plug-in devices. Users employ their mobile device or desktop to view performance information and control devices from anywhere.

Current solutions lack the sophistication needed to detect minor issues before they become major issues, or emergencies. IoT offers a more reliable and complete solution by observing issues in a fine-grained way to control dangers and aid in preventing them; for example, it can measure changes in a system's state impacting fire safety rather than simply detecting smoke.

Productivity and Quality of Life

Beyond safety or energy concerns, most people desire certain comforts from housing or commercial spaces like specific lighting and temperature. IoT enhances these comforts by allowing faster and easier customizing.

Adjustments also apply to the area of productivity. They personalize spaces to create an optimized environment such as a smart office or kitchen prepared for a specific individual.

12. IoT – Transportation Applications

At every layer of transportation, IoT provides improved communication, control, and data distribution. These applications include personal vehicles, commercial vehicles, trains, UAVs, and other equipment. It extends throughout the entire system of all transportation elements such as traffic control, parking, fuel consumption, and more.

Rails and Mass Transit

Current systems deliver sophisticated integration and performance, however, they employ older technology and approaches to MRT. The improvements brought by IoT deliver more complete control and monitoring. This results in better management of overall performance, maintenance issues, maintenance, and improvements.

Mass transit options beyond standard MRT suffer from a lack of the integration necessary to transform them from an option to a dedicated service. IoT provides an inexpensive and advanced way to optimize performance and bring qualities of MRT to other transportation options like buses. This improves services and service delivery in the areas of scheduling, optimizing transport times, reliability, managing equipment issues, and responding to customer needs.

Road

The primary concerns of traffic are managing congestion, reducing accidents, and parking. IoT allows us to better observe and analyze the flow of traffic through devices at all traffic observation points. It aids in parking by making storage flow transparent when current methods offer little if any data.

This smart road sign receives data and modifications to better inform drivers and prevent congestion or accidents.

Accidents typically result from a number of factors, however, traffic management impacts their frequency. Construction sites, poor rerouting, and a lack of information about traffic status are all issues that lead to incidents. IoT provides solutions in the form of better information sharing with the public, and between various parties directly affecting road traffic.

Automobile

Many in the automotive industry envision a future for cars in which IoT technology makes cars "smart," attractive options equal to MRT. IoT offers few significant improvements to personal vehicles. Most benefits come from better control over related infrastructure and the inherent flaws in automobile transport; however, IoT does improve personal vehicles as personal spaces. IoT brings the same improvements and customization to a vehicle as those in the home.

Commercial Transportation

Transportation benefits extend to business and manufacturing by optimizing the transport arm of organizations. It reduces and eliminates problems related to poor fleet management through better analytics and control such as monitoring idling, fuel consumption, travel conditions, and travel time between points. This results in product transportation operating more like an aligned service and less like a collection of contracted services.

13. IoT – Education Applications

IoT in the classroom combines the benefits of IoT in content delivery, business, and healthcare. It customizes and enhances education by allowing optimization of all content and forms of delivery. It enables educators to give focus to individuals and their method. It also reduces costs and labor of education through automation of common tasks outside of the actual education process.

Education Organizations

Education organizations typically suffer from limited funding, labor issues, and poor attention to actual education. They, unlike other organizations, commonly lack or avoid analytics due to their funding issues and the belief that analytics do not apply to their industry.

IoT not only provides valuable insight, but it also democratizes that information through low-cost, low-power small devices, which still offer high performance. This technology aids in managing costs, improving the quality of education, professional development, and facility management improvement through rich examinations of key areas:

- Student response, performance, and behavior
- Instructor response, performance, and behavior
- Facility monitoring and maintenance
- Data from other facilities

Data informs them about ineffective strategies and actions, whether educational efforts or facility qualities. Removing these roadblocks makes them more effective.

Educators

Information provided by IoT empowers educators to deliver improved education. They have a window into the success of their strategies, their students' perspective, and other aspects of their performance. IoT relieves them of administrative and management duties, so they can focus on their mission. It automates manual and clerical labor, and facilitates supervising through features like system flags or controls to ensure students remain engaged.

A school in Richmond, California, embeds RFID chips in ID cards to track the presence of students. Even if students are not present for check-in, the system will track and log their presence on campus.

IoT provides instructors with easy access to powerful educational tools. Educators can use IoT to perform as a one-on-one instructor providing specific instructional designs for each pupil; for example, using data to determine the most effective supplements for each student, and auto-generating content from lesson materials on-demand for any student.

The application of technology improves the professional development of educators because they truly see what works, and learn to devise better strategies, rather than simply repeating old or ineffective methods.

IoT also enhances the knowledge base used to devise education standards and practices. Education research suffers from accuracy issues and a general lack of data. IoT introduces large high quality, real-world datasets into the foundation of educational design. This comes from IoT's unique ability to collect enormous amounts of varied data anywhere.

Personalized Education

IoT facilitates the customization of education to give every student access to what they need. Each student can control their experience and participate in instructional design, and much of this happens passively. The student simply utilizes the system, and performance data primarily shapes their design. This combined with organizational and educator optimization delivers highly effective education while reducing costs.

14. IoT – Government Applications

IoT supports the development of *smart* nations and *smart* cities. This includes enhancement of infrastructure previously discussed (e.g., healthcare, energy, transportation, etc.), defense, and also the engineering and maintenance of communities.

City Planning and Management

Governing bodies and engineers can use IoT to analyze the often complex aspects of city planning and management. IoT simplifies examining various factors such as population growth, zoning, mapping, water supply, transportation patterns, food supply, social services, and land use. It gathers detailed data in these areas and produces more valuable and accurate information than current analytics given its ability to actually “live” with people in a city.

Smart trashcans in New York tell garbage collectors when they need to be emptied. They optimize trash service by ensuring drivers only make necessary stops, and drivers modify their route to reduce fuel consumption.

In the area of management, IoT supports cities through its implementation in major services and infrastructure such as transportation and healthcare. It also aids in other key areas like water control, waste management, and emergency management. Its real-time and detailed information facilitate more prompt decisions in contrast to the traditional process plagued by information lag, which can be critical in emergency management.

Standard state services are also improved by IoT, which can automate otherwise slow processes and trim unnecessary state expenses; for example, it can automate motor vehicle services for testing, permits, and licensing.

IoT also aids in urban improvement by skipping tests or poor research, and providing functional data for how the city can be optimized. This leads to faster and more meaningful changes.

Creating Jobs

IoT offers thorough economic analysis. It makes previous blind spots visible and supports better economic monitoring and modeling. It analyzes industry and the marketplace to spot opportunities for growth and barriers.

National Defense

National threats prove diverse and complicated. IoT augments armed forces systems and services, and offers the sophistication necessary to manage the landscape of national defense. It supports better protection of borders through inexpensive, high performance devices for rich control and observation.

IoT automates the protection tasks typically spread across several departments and countless individuals. It achieves this while improving accuracy and speed.

15. IoT – Law Enforcement Applications

IoT enhances law enforcement organizations and practice, and improves the justice system. The technology boosts transparency, distributes critical data, and removes human intervention where it proves unnecessary.

Policing

Law enforcement can be challenging. IoT acts as an instrument of law enforcement which reduces manual labor and subjective decisions through better data, information sharing, and advanced automation. IoT systems shave costs by reducing human labor in certain areas such as certain traffic violations.

IoT aids in creating better solutions to problems by using technology in the place of force; for example, light in-person investigations of suspicious activities can be replaced with remote observation, logged footage of violations, and electronic ticketing. It also reduces corruption by removing human control and opinion for some violations.

This dart planted in a truck gate prevents dangerous car chases. A patrol car launches the tracking dart which pierces the vehicle. Then the main system receives all data needed to locate the vehicle.

Court System

Current court systems utilize traditional technology and resources. They generally do not exploit modern analytics or automation outside of minor legal tasks. IoT brings superior analytics, better evidence, and optimized processes to court systems which accelerate processes, eliminate excessive procedures, manage corruption, reduce costs, and improve satisfaction.

In the criminal court system, this can result in a more effective and fair system. In routine court services, it introduces automation similar to that of common government office services; for example, IoT can automate forming an LLC.

IoT combined with new regulations can remove lawyers from many common legal tasks or reduce the need for their involvement. This reduces costs and accelerates many processes which often require months of traversing legal procedures and bureaucracy.

16. IoT – Consumer Applications

Consumers benefit personally and professionally from the optimization and data analysis of IoT. IoT technology behaves like a team of personal assistants, advisors, and security. It enhances the way we live, work, and play.

Home

IoT takes the place of a full staff:

- **Butler –** IoT waits for you to return home, and ensures your home remains fully prepared. It monitors your supplies, family, and the state of your home. It takes actions to resolve any issues that appear.
- **Chef –** An IoT kitchen prepares meals or simply aids you in preparing them.
- **Nanny –** IoT can somewhat act as a guardian by controlling access, providing supplies, and alerting the proper individuals in an emergency.
- **Gardner –** The same IoT systems of a farm easily work for home landscaping.
- **Repairman –** Smart systems perform key maintenance and repairs, and also request them.
- **Security Guard –** IoT watches over you 24/7. It can observe suspicious individuals miles away, and recognize the potential of minor equipment problems to become disasters well before they do.

This smart, connected stove from Whirlpool allows two different heat settings on the same surface, remote monitoring, and remote control.

Work

A smart office or other workspace combines customization of the work environment with smart tools. IoT learns about you, your job, and the way you work to deliver an optimized environment. This results in practical accommodations like adjusting the room temperature, but also more advanced benefits like modifying your schedule and the tools you use to increase your output and reduce your work time. IoT acts as a manager and consultant capable of seeing what you cannot.

Play

IoT learns as much about you personally as it does professionally. This enables the technology to support leisure:

- **Culture and Night Life –** IoT can analyze your real-world activities and response to guide you in finding more of the things and places you enjoy such as recommending restaurants and events based on your preferences and experiences.
- **Vacations –** Planning and saving for vacations proves difficult for some, and many utilize agencies, which can be replaced by IoT.
- **Products and Services –** IoT offers better analysis of the products you like and need than current analytics based on its deeper access. It integrates with key information like your finances to recommend great solutions.

17. IoT – ThingWorx

Thingworx is a platform for the rapid development and deployment of smart, connected devices. Its set of integrated IoT development tools support connectivity, analysis, production, and other aspects of IoT development.

It offers Vuforia for implementing augmented reality development, and Kepware for industrial connectivity. KEPServerEX provides a single point for data distribution, and facilitates interoperability when partnered with a ThingWorx agent.

Components

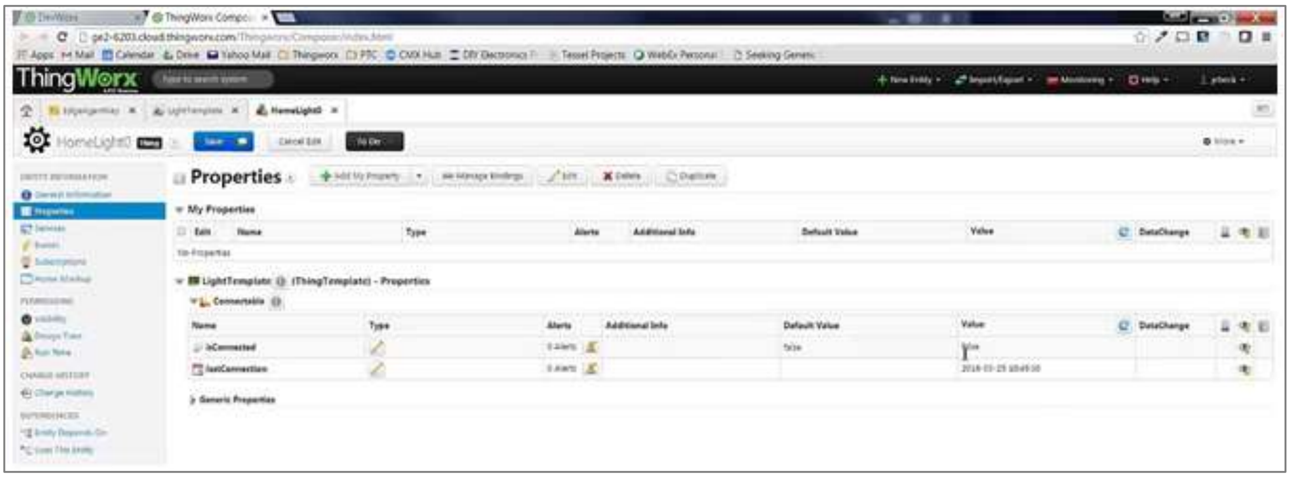
Thingworx offers several key tools for building applications. These tools include the Composer, the Mashup Builder, storage, a search engine, collaboration, and connectivity. The Composer provides a modeling environment for design testing. The Mashup Builder delivers easy dashboard building through common components (or widgets); for example, buttons, lists, wikis, gauges, and etc.

Thingworx uses a search engine known as SQUEAL, meaning Search, Query, and Analysis. Users employ SQUEAL in analyzing and filtering data, and searching records.

Interface

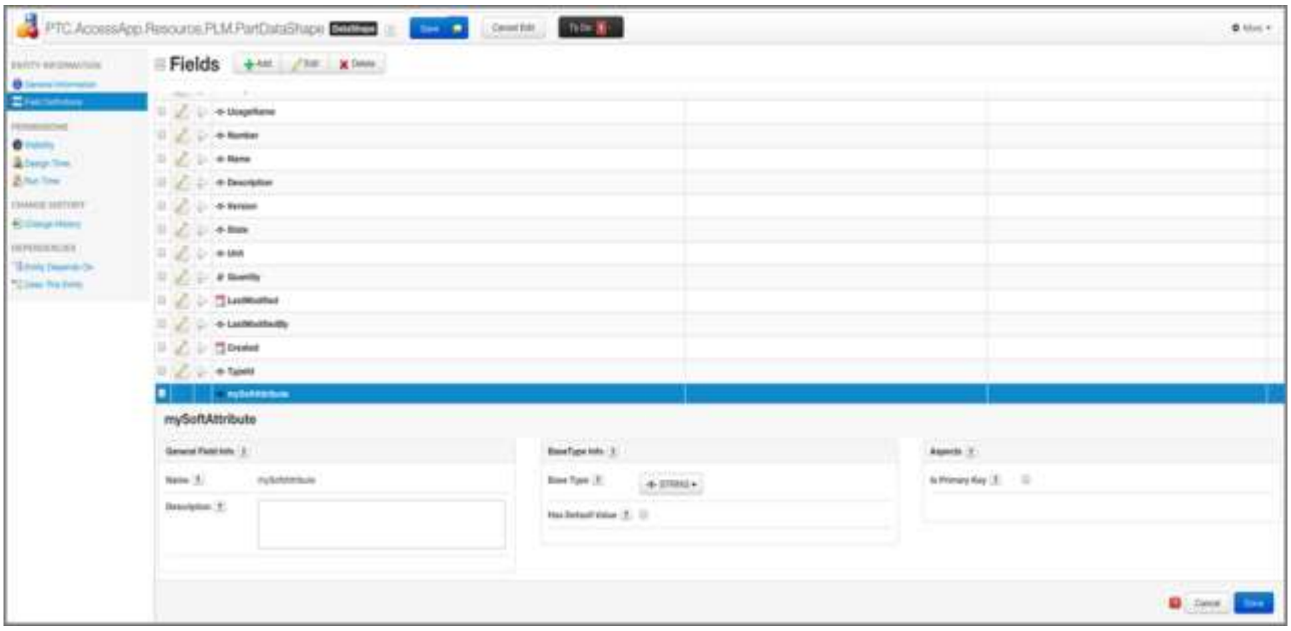
The ThingWorx platform uses certain terms you must familiarize yourself with. In the main screen's top menu, you search for **entities** or create them. "Entity" refers to something created in ThingWorx. You can also import/export files and perform various operations on them.

In the left menu, you find entity groups, which are used to produce models and visualize data; and manage storage, collaboration, security, and the system.



When you select the Modeling category in the menu, you begin the process by creating an entity. The entity can be any physical device or software element, and it produces an **event** on changes to its property values; for example, a sensor detects a temperature change. You can set events to trigger actions through a subscription which makes decisions based on device changes.

Data Shapes consist of one or more fields. They describe the data structure of custom events, infotables, streams, and datatables. Data shapes are considered entities.



Thing Templates and **Thing Shapes** allow developers to avoid repeating device property definitions in large IoT systems. Developers create Thing Templates to allow new devices to inherit properties. They use Thing Shapes to define Templates, properties, or execute services.

Note a Thing only inherits properties, services, events, and other qualities from a single template, however, Things and templates can inherit properties from multiple Thing Shapes.

Development

ThingWorx actually requires very little programming. Users connect devices, establish a data source, establish device behaviors, and build an interface without any coding. It also offers scalability appropriate for both hobbyist projects and industrial applications.

18. IoT – Cisco Virtualized Packet Core

Cisco Virtualized Packet Core (VPC) is a technology providing all core services for 4G, 3G, 2G, WiFi, and small cell networks. It delivers networking functionality as virtualized services to allow greater scalability and faster deployment of new services at a reduced cost. It distributes and manages packet core functions across all resources, whether virtual or physical. Its key features include packet core service consolidation, dynamic scaling, and system agility.

Its technology supports IoT by offering network function virtualization, SDN (software-defined networking), and rapid networked system deployment. This proves critical because its virtualization and SDN support low-power, high flow networking, and the simple deployment of a wide variety of small devices. It eliminates many of the finer details of IoT systems, and conflicts, through consolidating into a single system and single technology for connecting and integrating all elements.

Use Case: Smart Transportation

Rail transportation provides a viable example of the power of VPC. The problems VPC solves relate to safety, mobility, efficiency, and service improvement:

- Rail applications use their own purpose-built networks, and suffer from interoperability issues; for example, trackside personnel cannot always communicate with local police due to different technologies.

- Determining if passengers need extra time to board remains a mostly manual task.
- Data updates, like schedules, remain manual.
- Each piece of equipment, e.g., a surveillance camera, requires its own network and power source.

DELAYS
Station, please expect delays in service on Ditmars Boulevard-bound N and Q, Forest Hills-bound R trains at this time.
There is Planned work for above lines. Please check http://www.mta.info for
GOOD SERVICE
PLANNED WORK
DELAYS
Atlantic Avenue–Pacific Street Station
D
N
R

A smart MRT sign in New York

VPC improves service by introducing direct communication over a standard network, more and automated monitoring, automatic data updates through smart signs, and native IP networks for all devices along with PoE (Power over Ethernet) technology. This results in passengers who feel safer, and enjoy a better quality service.

19. IoT – Salesforce

The Salesforce IoT Cloud is a platform for storing and processing IoT data. It uses the Thunder engine for scalable, real-time event processing. Its collection of application development components, known as Lightning, powers its applications. It gathers data from devices, websites, applications, customers, and partners to trigger actions for real-time responses.

Salesforce, a CRM leader, decided to enter this space due to the need to remain competitive in the coming era. The IoT cloud adds to Salesforce by expanding its reach, and the depth of its analytics.

Salesforce combined with IoT delivers dramatically improved customer service with tighter integration and responses to real-time events; for example, adjustments in wind turbines could trigger automatic rebooking of delayed/canceled connecting flights before airline passengers land.

Electric Imp

The Electric Imp platform is Salesforce's recommended method for quickly connecting devices to the cloud. You develop applications through the Squirrel language; a high level, OO, lightweight scripting language. Applications consist of two modules: the device module, which runs on the device; and the agent module, which runs in the Electric Imp cloud. The platform ensures secure communication between the modules, and you send devices messages with a simple call:

```
agent.send("nameOfmessage", data);
```

Listen for messages on the agent with the following code:

```
device.on("nameOfmessage", function(data) {
  //Data operations
});
```

Beyond these basic tasks, coding for device interaction, monitoring, and response resembles standard web application development, and uses a simple, easy-to-learn syntax.

20. IoT – GE Predix

GE (General Electric) Predix is a software platform for data collection from industrial instruments. It provides a cloud-based PaaS (platform as a service), which enables industrial-grade analytics for operations optimization and performance management. It connects data, individuals, and equipment in a standard way.

Predix was designed to target factories, and give their ecosystems the same simple and productive function as operating systems that transformed mobile phones. It began as a tool for General Electric's internal IoT, specifically created to monitor products sold.

Ge Predix Partnered with Microsoft Azure

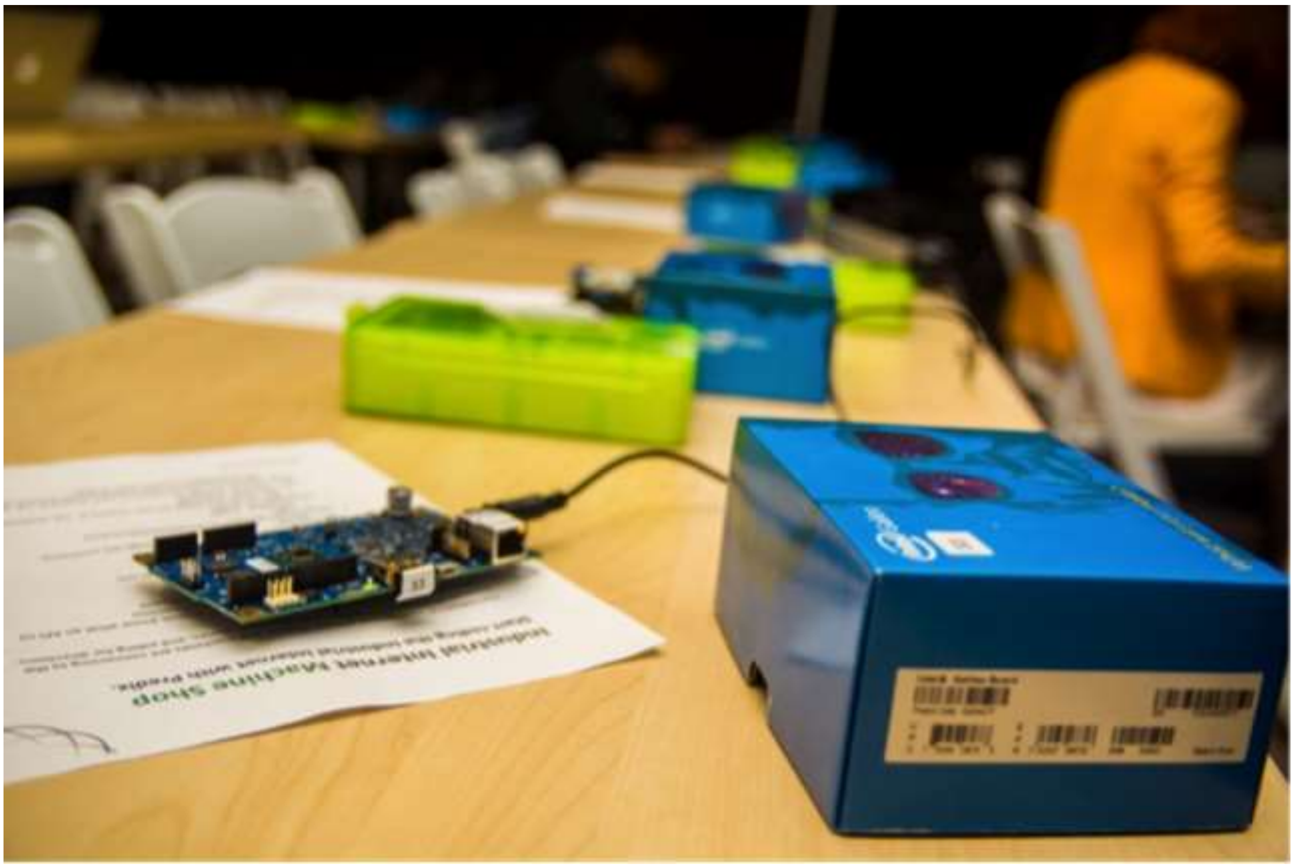
Microsoft's Azure is a cloud computing platform and supporting infrastructure. It provides PaaS and IaaS, and assorted tools for building systems. Predix, recently made available on Azure, exploits a host of extra features like AI, advanced data visualization, and natural language technology. Microsoft plans to eventually integrate Predix with its Azure IoT suite and Cortana Intelligence suite, and also their well-established business applications. Azure will also allow users to build applications using Predix data. Note AWS and Oracle also support Predix.

Developer Kits

GE offers inexpensive developer kits consisting of general components and an Intel Edison processor module. Developers have the options of a dual core board and a Raspberry Pi board. Developers need only provide an IP address, Ethernet connection, power supply, and light programming to set data collection.

The kit automatically establishes the necessary connection, registers with the central Predix system, and begins transmitting environmental data from sensors. Users subscribe to hardware/software output, and GE Digital owns and manages the hardware and software for the user.

This kit replaces the awkward and involved assemblies of simulations and testing environments. In other simulations, developers typically use a large set of software (one for each device), and specific configurations for each connection. They also program the monitoring of each device, which can sometimes take hours. The kit reduces much of the time spent performing these tasks from hours to only minutes.



The Predix developer kit

The kit also includes software components for designing an IoT application that partners with Predix services. GE plans to release other versions of the kit for different applications.

21. IoT – Eclipse IoT

Eclipse IoT is an ecosystem of entities (industry and academia) working together to create a foundation for IoT based exclusively on open source technologies. Their focus remains in the areas of producing open source implementations of IoT standard technology; creating open source frameworks and services for utilization in IoT solutions; and developing tools for IoT developers.

Smarthome Project

SmartHome is one of Eclipse IoT's major services. It aims to create a framework for building smart home solutions, and its focus remains heterogeneous environments, meaning assorted protocols and standards integration.

SmartHome provides uniform device and information access to facilitate interaction between devices. It consists of OSGi bundles capable of deployment in an OSGi runtime, with OSGi services defined as extension points.

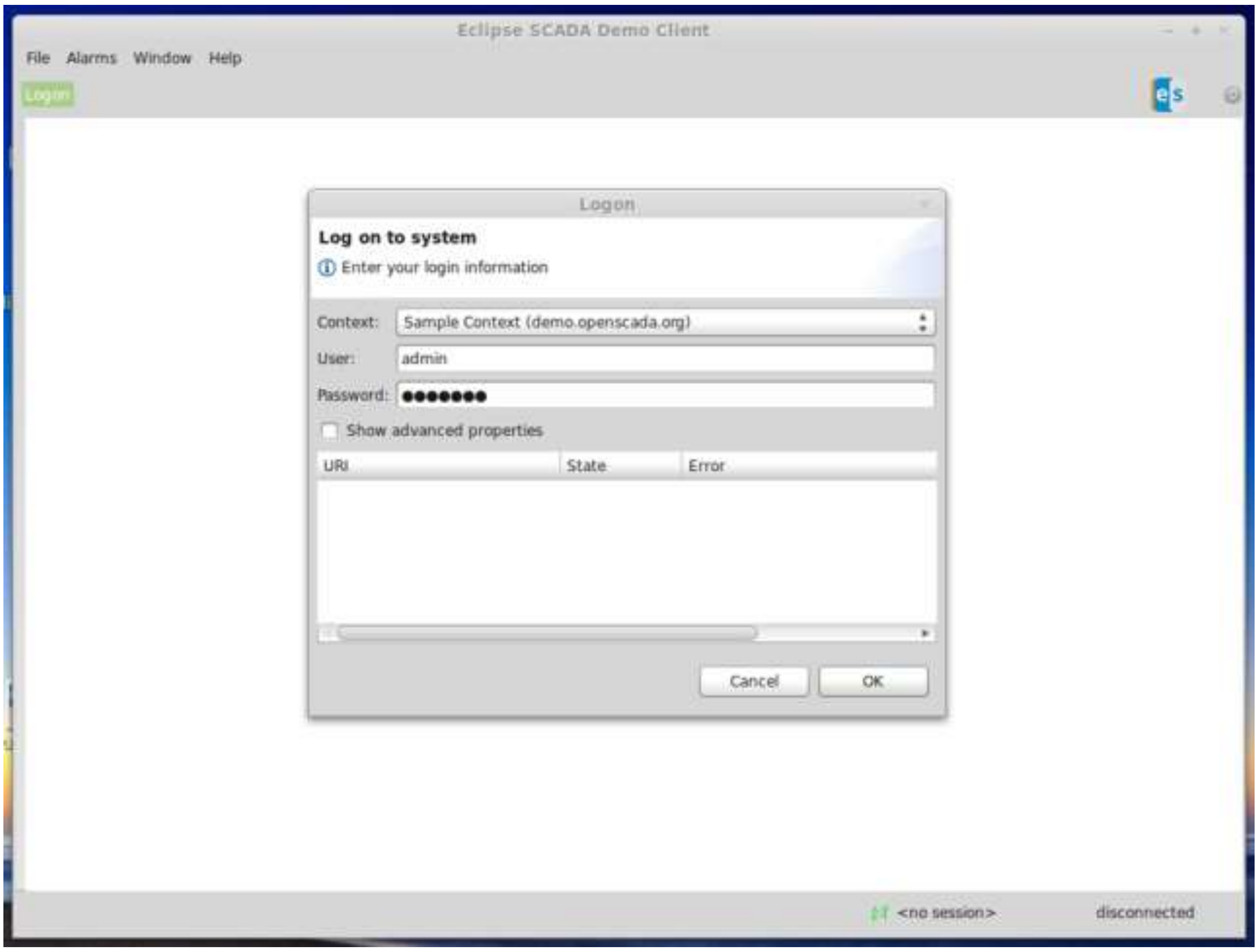
OSGi bundles are Java class groups and other resources, which also include detailed manifest files. The manifest contains information on file contents, services needed to enhance class behavior, and the nature of the aggregate as a component. Review an example of a manifest below:

```
Bundle-Name: Hi Everyone                        // Bundle Name

Bundle-SymbolicName: xyz.xyz.hievery1           // Header specifying an identifier

Bundle-Description: A Hi Everyone bundle        // Functionality description

Bundle-ManifestVersion: 2                       // OSGi specification

Bundle-Version: 1.0.0                           // Version number of bundle
```

```
Bundle-Activator: xyz.xyz.Activator              // Class invoked on bundle activation
Export-Package: xyz.xyz.helloworld;version="1.0.0"   // Java packages available externally
Import-Package: org.osgi.framework;version="1.3.0"   // Java packages needed from
                                                     // external source
```

Eclipse SCADA

Eclipse SCADA, another major Eclipse IoT service, delivers a means of connecting various industrial instruments to a shared communication system. It also post-processes data and sends data visualizations to operators. It uses a SCADA system with a communication service, monitoring system, archive, and data visualization.



Context: Sample Context (demo.openscada.org)
User: admin
Password: ●●●●●●●

It aims to be a complete, state-of-the-art open source SCADA system for developing custom solutions. Its supported technologies and tools include shell applications, JDBC, Modbus TCP and RTU, Simatic S7 PLC, OPC, and SNMP.

22. IoT – Contiki

Contiki is an operating system for IoT that specifically targets small IoT devices with limited memory, power, bandwidth, and processing power. It uses a minimalist design while still packing the common tools of modern operating systems. It provides functionality for management of programs, processes, resources, memory, and communication.

It owes its popularity to being very lightweight (by modern standards), mature, and flexible. Many academics, organization researchers, and professionals consider it a go-to OS. Contiki only requires a few kilobytes to run, and within a space of under 30KB, it fits its entire operating system: a web browser, web server, calculator, shell, telnet client and daemon, email client, vnc viewer, and ftp. It borrows from operating systems and development strategies from decades ago, which easily exploited equally small space.

Contiki Communication

Contiki supports standard protocols and recent enabling protocols for IoT:

- **uIP (for IPv4) –** This TCP/IP implementation supports 8-bit and 16-bit microcontrollers.
- **uIPv6 (for IPv6)** – This is a fully compliant IPv6 extension to uIP.
- **Rime –** This alternative stack provides a solution when IPv4 or IPv6 prove prohibitive. It offers a set of primitives for low-power systems.
- **6LoWPAN –** This stands for IPv6 over low-power wireless personal area networks. It provides compression technology to support the low data rate wireless needed by devices with limited resources.
- **RPL –** This distance vector IPv6 protocol for LLNs (low-power and lossy networks) allows the best possible path to be found in a complex network of devices with varied capability.
- **CoAP –** This protocol supports communication for simple devices, typically devices requiring heavy remote supervision.

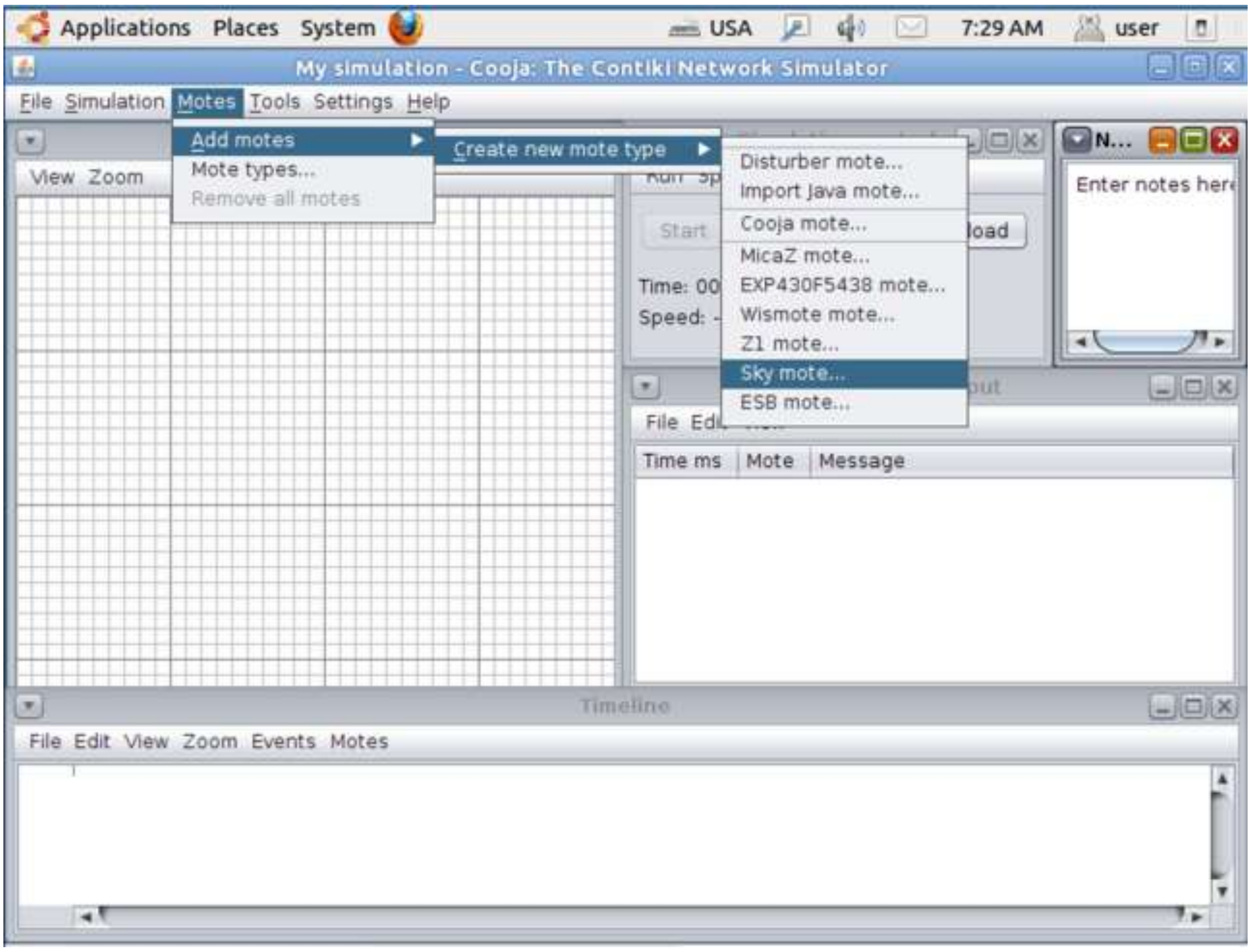
Dynamic Module Loading

Dynamic module loading and linking at run-time supports environments in which application behavior changes after deployment. Contiki's module loader loads, relocates, and links ELF files.

The Cooja Network Simulator

Cooja, the Contiki network simulator, spawns an actual compiled and working Contiki system controlled by Cooja.

Using Cooja proves simple. Simply create a new mote type by selecting the **Motes** menu and **Add Motes > Create New Mote Type**. In the dialog that appears, you choose a name for the mote, select its firmware, and test its compilation.



After creation, add motes by clicking **Create**. A new mote type will appear to which you can attach nodes. The final step requires saving your simulation file for future use.

23. IoT – Security

Every connected device creates opportunities for attackers. These vulnerabilities are broad, even for a single small device. The risks posed include data transfer, device access, malfunctioning devices, and always-on/always-connected devices.

The main challenges in security remain the security limitations associated with producing low-cost devices, and the growing number of devices which creates more opportunities for attacks.

Security Spectrum

The definition of a secured device spans from the most simple measures to sophisticated designs. Security should be thought of as a spectrum of vulnerability which changes over time as threats evolve.

Security must be assessed based on user needs and implementation. Users must recognize the impact of security measures because poorly designed security creates more problems than it solves.

Example: A German report revealed hackers compromised the security system of a steel mill. They disrupted the control systems, which prevented a blast furnace from being shut down properly, resulting in massive damage. Therefore, users must understand the impact of an attack before deciding on appropriate protection.

Challenges

Beyond costs and the ubiquity of devices, other security issues plague IoT:

- **Unpredictable Behavior –** The sheer volume of deployed devices and their long list of enabling technologies means their behavior in the field can be unpredictable. A specific system may be well designed and within administration control, but there are no guarantees about how it will interact with others.
- **Device Similarity –** IoT devices are fairly uniform. They utilize the same connection technology and components. If one system or device suffers from a vulnerability, many more have the same issue.
- **Problematic Deployment –** One of the main goals of IoT remains to place advanced networks and analytics where they previously could not go. Unfortunately, this creates the problem of physically securing the devices in these strange or easily accessed places.
- **Long Device Life and Expired Support –** One of the benefits of IoT devices is longevity, however, that long life also means they may outlive their device support. Compare this to traditional systems which typically have support and upgrades long after many have stopped using them. Orphaned devices and abandonware lack the same security hardening of other systems due to the evolution of technology over time.
- **No Upgrade Support –** Many IoT devices, like many mobile and small devices, are not designed to allow upgrades or any modifications. Others offer inconvenient upgrades, which many owners ignore, or fail to notice.
- **Poor or No Transparency –** Many IoT devices fail to provide transparency with regard to their functionality. Users cannot observe or access their processes, and are left to assume how devices behave. They have no control over unwanted functions or data collection; furthermore, when a manufacturer updates the device, it may bring more unwanted functions.
- **No Alerts –** Another goal of IoT remains to provide its incredible functionality without being obtrusive. This introduces the problem of user awareness. Users do not monitor the devices or know when something goes wrong. Security breaches can persist over long periods without detection.

24. IoT – Identity Protection

IoT devices collect data about their environment, which includes people. These benefits introduce heavy risk. The data itself does not present the danger, however, its depth does. The highly detailed data collection paints a very clear picture of an individual, giving criminals all the information they need to take advantage of someone.

People may also not be aware of the level of privacy; for example, entertainment devices may gather A/V data, or "watch" a consumer, and share intimate information. The demand and price for this data exacerbates the issue considering the number and diversity of parties interested in sensitive data.

Problems specific to IoT technology lead to many of its privacy issues, which primarily stem from the user's inability to establish and control privacy:

Consent

The traditional model for "notice and consent" within connected systems generally enforces existing privacy protections. It allows users to interact with privacy mechanisms, and set preferences typically through accepting an agreement or limiting actions. Many IoT devices have no such accommodations. Users not only have no control, but they are also not afforded any transparency regarding device activities.

The Right to be Left Alone

Users have normal expectations for privacy in certain situations. This comes from the commonly accepted idea of public and private spaces; for example, individuals are not surprised by surveillance cameras in commercial spaces, however, they do not expect them in their personal vehicle. IoT devices challenge these norms people recognize as the "right to be left alone." Even in public spaces, IoT creeps beyond the limits of expected privacy due to its power.

Indistinguishable Data

IoT deploys in a wide variety of ways. Much of IoT implementation remains group targeted rather than personal. Even if users give IoT devices consent for each action, not every system can reasonably process every set of preferences; for example, small devices in a complex assembly cannot honor the requests of tens of thousands of users they encounter for mere seconds.

Granularity

Modern big data poses a substantial threat to privacy, but IoT compounds the issue with its scale and intimacy. It goes not only where passive systems cannot, but it collects data everywhere. This supports creation of highly detailed profiles which facilitate discrimination and expose individuals to physical, financial, and reputation harm.

Comfort

The growth of IoT normalizes it. Users become comfortable with what they perceive as safe technology. IoT also lacks the transparency that warns users in traditional connected systems; consequently, many act without any consideration for the potential consequences.

25. IoT – Liability

The security flaws of IoT and its ability to perform certain tasks open the door to any associated liability. The three main areas of concern are device malfunction, attacks, and data theft. These issues can result in a wide variety of damages.

Device Malfunction

IoT introduces a deeper level of automation which can have control over critical systems, and systems impacting life and property. When these systems fail or malfunction, they can cause substantial damage; for example, if an IoT furnace control system experiences a glitch, it may fail in an unoccupied home and cause frozen pipes and water damage. This forces organizations to create measures against it.

This smart thermostat allows attackers to gain remote access, and breach the rest of the network.

Cyber Attacks

IoT devices expose an entire network and anything directly impacted to the risk of attacks. Though those connections deliver powerful integration and productivity, they also create the perfect opportunity for mayhem like a hacked stove or fire safety sprinkler system. The best measures against this address the most vulnerable points, and provide custom protections such as monitoring and access privileges.

Some of the most effective measures against attacks prove simple:

- **Built-in Security –** Individuals and organizations should seek hardened devices, meaning those with security integrated in the hardware and firmware.
- **Encryption –** This must be implemented by the manufacturer and through user systems.
- **Risk Analysis –** Organizations and individuals must analyze possible threats in designing their systems or choosing them.
- **Authorization –** Devices, whenever possible, must be subject to privilege policies and access methods.

Bitdefender BOX secures all connected devices in the home.

Data Theft

Data, IoT's strength and weakness, proves irresistible to many. These individuals have a number of reasons for their interest: the value of personal data to marketing/advertising, identity theft, framing individuals for crimes, stalking, and a bizarre sense of satisfaction. Measures used to fight attacks are also effective in managing this threat.

26. IoT – Useful Resources

The following resources offer more in-depth information on IoT development and administration. You can refer them to increase your knowledge on IoT further.

Useful IoT Websites

- Internet of Things Council – This European think tank offers the best and latest IoT information. They analyze every aspect of IoT from forecasting to discussing prototype development, and the social implications of IoT.
- LinkedIn Pulse Content – LinkedIn, described as the world's largest professional network, allows over 100 million professionals to network. It opened its publishing platform, Pulse, to the public in 2014, resulting in a wealth of valuable professional and industry information. This information includes rare insight, training media, and more related to IoT systems and technologies.
- Lynda.com IoT Videos – This online learning organization offers thousands of videos on various topics (including IoT) supplied by professionals, organizations, and individuals.
- YouTube IoT Videos – Individuals just like you produce thousands of IoT videos on YT to address particular topics not covered elsewhere, or covered poorly elsewhere. These videos use different languages, styles, and offer unique voices.

Useful IoT Literature

Building Internet of Things with the Arduino (Volume 1) by *Charalampos Doukas*	**Making Things Talk, 2nd Edition** by *Tom Igoe*	**Building Wireless Sensor Networks: with ZigBee, Xbee, Arduino, and Processing** by *Robert Faludi*
Building the Web of Things with Examples in Node.js and Raspberry Pi by *Dominique D. Guinard and Vlad M. Trifa*	**Internet of Things (A Hands-on-Approach)** by *Arshdeep Bahga and Vijay Madisetti*	**The Internet of Things in the Cloud: A Middleware Perspective** by *Honbo Zhou*

If you would like your site or literature listed on this page, please contact us at **contact@tutorialspoint.com**