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

DEPARTMENT OF MECHATRONICS ENGINEERING

COURSE MATERIALS

MR 405 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: 2013
- Course offered: B.Tech Mechatronics Engineering
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of Dr. A P J Abdul Kalam Technological University.

DEPARTMENT VISION

To develop professionally ethical and socially responsible Mechatronics engineers to serve the humanity through quality professional education.

DEPARTMENT MISSION

1) The department is committed to impart the right blend of knowledge and quality education to create professionally ethical and socially responsible graduates.

2) The department is committed to impart the awareness to meet the current challenges in technology.

3) Establish state-of-the-art laboratories to promote practical knowledge of mechatronics to meet the needs of the society

PROGRAMME EDUCATIONAL OBJECTIVES

I. Graduates shall have the ability to work in multidisciplinary environment with good professional and commitment.

II. Graduates shall have the ability to solve the complex engineering problems by applying electrical, mechanical, electronics and computer knowledge and engage in lifelong learning in their profession.

III. Graduates shall have the ability to lead and contribute in a team with entrepreneur skills, professional, social and ethical responsibilities.

IV. Graduates shall have ability to acquire scientific and engineering fundamentals necessary for higher studies and research.

PROGRAM OUTCOME (PO'S)

Engineering Graduates will be able to:

PO 1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO 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.

PO 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.

PO 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.

PO 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.

PO 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.

PO 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.

PO 8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO 9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO 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.

PO 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.

PO 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 OUTCOME (PSO'S)

PSO 1: Design and develop Mechatronics systems to solve the complex engineering problem by integrating electronics, mechanical and control systems.

PSO 2: Apply the engineering knowledge to conduct investigations of complex engineering problem related to instrumentation, control, automation, robotics and provide solutions.

COURSE OUTCOME

After the completion of the course the student will be able to

CO 1	Acquire knowledge to design a embedded system
CO 2	Describe about the hardware and software components of embedded system
CO 3	Acquire knowledge on custom single purpose processor design and optimization
CO 4	Interpret about the general purpose processors
CO 5	Understand the concepts of common memory devices.
CO 6	Explain about various software development tools and RTOS

CO VS PO'S AND PSO'S MAPPING

CO	PO1	PO 2	PO3	PO 4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PS0 1	PSO 2
CO 1	3	-	2	2	-	-	-	-	-	-	-	3	3	2
CO 2	3	-	2	2	-	-	-	-	-	-	-	3	3	2
CO 3	3	-	2	2	-	-	-	-	-	-	-	3	3	2
CO 4	3	-	2	2	-	-	-	-	-	-	-	3	3	2
CO 5	3	-	2	2	-	-	-	-	-	-	-	3	3	2
CO 6	3	-	2	2	-	-	-	-	-	-	-	3	3	2

Note: H-Highly correlated=3, M-Medium correlated=2, L-Less correlated=1

SYLLABUS

Course code	Course Name	L-T-P - Credits	Year of Introduction
MR405	Embedded Systems	3-0-0-3	2016

Prerequisite : NIL

Course Objectives

- To make students familiar with the architecture, hardware and software elements, programming models, tools for embedded system design and implementation of embedded system.
- To give students knowledge on the hardware and real time operating systems used for the embedded systems design.
- To expose students to the concepts of embedded system principles, software development tools and RTOS

Syllabus

Embedded system, Functional building block of embedded system- Characteristics- Challenges in embedded system design- Classification-SOC- Custom Single-purpose processors- Application specific instruction set processors- General-purpose processors- Standard single-purpose processors-Common memory device- Types of I/O devices - Serial devices - Parallel port devices - Sophisticated features- Development tools-S/W Architectures.

Expected outcome.

The student will be familiar with

- the concepts of embedded systems
- the basic concepts of real time Operating system design.
- the design techniques to develop software for embedded systems
- the general purpose operating systems and the real time operating systems

Text Book:

1 Rajkamal, "*Embedded Systems – Architecture, Programming and Design*", Tata McGraw-Hill Publishing Company Ltd., New Delhi, 2010.

2. Frank Vahid and Tony Givargis, *Embedded System Design: A Unified Hardware/Software Introduction*, Wiley, 2002.

3. David E.Simon, "*An embedded software primer*", Pearson Education Asia 2001.

References:

1. Wayne Wolf, "*Computers as Components: Principles of Embedded Computer Systems Design*", The Morgan Kaufmann Series in Computer Architecture and Design, Elsevier Publications, 2008.

2. Dainel W. Lewis, *Fundamentals of embedded software where C and assembly meet*, PHI 2002.

Course Plan			
Module	**Contents**	**Hours**	**Sem. Exam Marks**
I	Embedded system- Functional building block of embedded system- Characteristics of embedded system applications- Challenges in embedded system design- Embedded system design processes	7	15%
II	Classification - Processors in the system - Other h/w units. Software components - Typical applications - Embedded systems on a chip (SoC) and use of VLSI circuits.	7	15%
FIRST INTERNAL EXAMINATION			
III	Custom Single-purpose processors : Hardware-Combinational Logic- Transistors and logic gates- Basic combinational and Sequential logic design- Custom single purpose processor design and optimization. Application specific instruction set processors- Microcontrollers- Digital signal processors	7	15%
IV	General-purpose processors: Software: Basic architecture- Datapath- Control unit- Memory- Instruction execution- Pipelining- Superscalar and VLIW architectures- Instruction set- Program and data memory space- Registers- I/O- Interrupts- Operating Systems- Standard single-purpose processors: Peripherals-some examples such as Timers-counters- Analog-digital converters.	7	15%

	SECOND INTERNAL EXAMINATION		
V	Common memory devices - Memory selection - Memory map - Internal devices & I/O devices map - Direct memory access -. Types of I/O devices - Serial devices - Parallel port devices - Sophisticated features - Timer and Counting devices - Advanced serial bus & I/O - High speed Buses - Common types - Advanced Buses.	7	20%
VI	**Development tools**: Host and Target machines – linker / locators – debugging techniques. **S/W Architectures:** Round robin-round robin with interrupt – function queue scheduling- RTOS.	7	20%
	END SEMESTER EXAM		

QUESTION PAPER PATTERN

Maximum Marks : 100 Exam Duration:3 hours

PART A: FIVE MARK QUESTIONS

8 compulsory questions –1 question each from first four modules and 2 questions each from last two modules (8 x 5= 40 marks)

PART B: 10 MARK QUESTIONS

5 questions uniformly covering the first four modules. Each question can have maximum of three sub questions, if needed. Student has to answer any 3 questions

(3 x10 = 30 marks)

PART C: 15 MARK QUESTIONS

4 questions uniformly covering the last two modules. Each question can have maximum of four sub questions, if needed. Student has to answer any two questions

(2 x15 = 30 marks)

QUESTION BANK

MODULE I				
Q:NO:	**QUESTIONS**	**CO**	**KL**	**PAGE NO:**
1	Define embedded system	CO1	K1	14
2	Explain challenges and applications of embedded system in detail	CO1	K2	15
3	Explain design process	CO1	K2	16
4	Distinguish between requirements and specifications	CO1	K4	24
5	Explain characteristics of embedded system in detail and also mention its application	CO1	K2	15
6	Explain system with an example	CO1	K2	13
7	Explain various levels of abstraction of embedded system	CO1	K2	16
MODULE II				
1	Explain watching dog timer	CO2	K2	41
2	Discuss about various forms of system memories used in the embedded processor	CO2	K2	42
3	Discuss about components of embedded system hardware	CO2	K2	46
4	Discuss embedded system on a chip	CO2	K2	51
5	Compare microprocessor and microcontroller	CO2	K2	32

6	Describe power source , clock oscillator and clocking units	CO2	K2	40
7	Discuss a) Embedded processor b) DSP processor	CO2	K2	35
8	Discuss a) media processor b) ASSP C) Multiprocessor using GPPS	CO2	K2	38
MODULE III				
1	What is a processor? Explain the benefits of using custom single purpose processor	CO3	K2	56
2	Design a counter that counts 0,1,2,3,4,5,6 using JK FF	CO3	K6	70
3	Write a short note on steps involved in design of a combinational circuit using basic logic gates	CO3	K2	57
4	Explain CMOS implementation of some basic logic gates	CO3	K2	58
5	Explain microprocessor	CO3	K2	32
6	Explain digital signal processors	CO3	K2	35
7	Explain multiplexer, decoder, adder, comparator, ALU	CO3	K2	64
MODULE IV				
1	Define operating systems	CO4	K2	98
2	Compare Harvard and Princeton architecture	CO4	K2	84
3	Discuss about instruction execution	CO4	K2	87

4	Discuss about general purpose processor basic architecture	CO4	K2	83
5	Discuss on a) registers b)input/output c)interrupts d)program and data memory space	CO4	K2	97
6	Discuss timers and counters	CO4	K2	101
MODULE V				
1	Explain Common memory devices	CO5	K2	114
2	Write a short note on Memory selection	CO5	K2	126
3	Explain Memory map	CO5	K2	138
4	Explain Internal devices & I/O devices map	CO5	K2	140
5	Describe Direct memory access	CO5	K2	153
6	Explain Types of I/O devices	CO5	K2	155
MODULE VI				
1	Explain Real Time Operating System	CO6	K2	163
2	Explain round robin with interrupt	CO6	K2	190
3	Write a short note on host and target machine	CO6	K2	171
4	Explain different debugging methods	CO6	K2	180
5	Write a short note on linker and locator	CO6	K2	175
6	Write a short note on function queue scheduling	CO6	K2	199

APPENDIX 1		
CONTENT BEYOND THE SYLLABUS		
S:NO	TOPIC	PAGE NO:
1.	Programming concept in high level language	216

MODULE - I

SYSTEM :

→ A system is an arrangement in which all its Unit assemble work together according to a set of rules

→ It can also be defined as an arrangement in which all its Units assemble and work together according to the plan or progra

Examples : • Time display system
• Automatic clothes washing system

Watch

→ It is a time display system

→ Parts : Hardware, Needles, Battery, dial, chassis and strap.

→ These parts are organized to show the real time in every second and continuously Update the time in every second.

→ The system program Updates the display usin three needles after each second. It follows a set of rules.

Rules :

1. All needles moves clockwise only.
2. A thin needle rotates every second.
3. A long needles rotates every minute
4. A short needle rotates every hour.
5. All needles return to the orginal position after 12 hours.

EMBEDDED SYSTEM :

→ Embedded means something that is attac another thing.

→ An embedded system can be thought of as a Computer hardware system having software embedded in it.

→ An embedded system can be an Independent system or it can be a part of large system. An embedde System is a microcontroller or microprocessor base System which is designed to perform a specific task.

→ An embedded System has three components

- it has hardware
- it has application Software
- it has Real Time Operating System (RTOS), that supervises the application software & provide mechanism 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 set the rules during the execution of application program.

→ A small scale embedded system may not have RTOS.

→ So we can define an embedded system as a Microcontroller based, Software driven and reliable real time Control System.

NEETHU.M
Assistant Professor

Characteristics of Embedded System :

(1) Single function.
(2) Tightly Constrained
(3) Reactive and real time
(4) Microcontroller or microprocessor based
(5) Memory
(6) Connects
(7) Hardware & Software Systems.

NEETHU.M
Assistant Professor

Single functions :

→ It performs specialized Operations and does the sam job repeatedly.

Tightly Constrained

→ The circuit size should be small enough to fit on a single chip and must perform fast enough to process data in a real time and consume minimum power to extend battery life.

Reactive and Real time :

It should Continuously react to the changes in the system environment and must compute certain results in realtime without any delay.

Microcontroller or Microprocessor based :

→ Microprocessor are multi-tasking in nature, wherea Microcontroller are single task in nature.

→ RAM, ROM, I/O ports, timers can be added externally and can vary in number in microprocessor.

Characteristics of Embedded System :

(1) Single function.
(2) Tightly Constrained
(3) Reactive and real time
(4) Microcontroller or microprocessor based
(5) Memory
(6) Connects
(7) Hardware & Software Systems.

NEETHU.M
Assistant Professor

Single functions :

→ It performs specialized Operations and does the sam job repeatedly.

Tightly Constrained

→ The circuit size should be small enough to fit on a single chip and must perform fast enough to process data in a real time and consume minimum power to extend battery life.

Reactive and Real time :

It should Continuously react to the changes in the system environment and must compute certain results in realtime without any delay.

Microcontroller or Microprocessor based :

→ Microprocessor are multi-tasking in nature, wherea Microcontroller are single task in nature.

→ RAM, ROM, I/O ports, timers can be added externally and can vary in number in microprocessor

→ RAM, ROM, I/O ports, timers are added externally but these components are to be embedded together as a chip and is fixed in number in Micro Controller.

→ In microprocessor, the designers can decide the number of memory and I/O ports needed whereas there is fixed number of memory and I/O ports which suits best for the specific task in micro Controller.

→ External Support of external memory or I/O makes microprocessor based system heavier and costlier whereas they are light weighted and cheap in microcontroller.

→ Microprocessor requires more space and consume more energy whereas microcontroller requires less space and less power.

Hardware & Software Systems :

→ Softwares used in embedded system is generally for providing flexibility and extra features.

→ Hardware is used for performance and Security.

Memory :

→ It must have a memory, as its Software usually embeds in ROM. It does not need any Secondary memories in the Computer.

CHALLENGES IN EMBEDDED SYSTEM DESIGN

→ The challenges that are encounted during the design process are not Computer related, rather they are mechanical or electrical.

→ of these, the most challenging areas are.

(a) Hardware. (b) deadlines. (c) power Consumption (d) Upgradeability (e) Reliability.

(a) Hardware : (How much hardware do we need?)

→ The choice of the hardware plays a major role in meeting manufacturing cost constraints and Performance deadlines.

→ The choice shouldn't be too expensive or too cheap rather it should be exact to meet the deadlines.

NEETHU.M
Assistant Professor

(b) Deadlines : (How do we meet deadlines)

→ Meeting deadlines is another challenge in designing an embedded Computing system.

→ One method of meeting deadlines is the "brute force method". In this method the speed of the h/w is increased so that the program runs faster of course, that makes the system more expensive

(c) Power Consumption : (How do we minimize Power Consumption)

→ In battery-powered applications, power consumption is extremely important, even in non-battery applications, excess power consumption can increase heat dissipation. One way to make it consume less

power is to run slowly, but slowing down leads to missing deadlines. Therefore careful design must be done to meet performance goals.

(d) <u>Design for Upgradability</u> : (How do we design for Upgradability?)

→ Designing a machine that can match with Upgrades in software is another challege.

→ The same hardware may be used with different versions of the software.

(e) <u>Reliability</u> : (Does it really works?)

→ Reliability plays a vital role, if products are safety-critical products. if these products lack reliability then the consequences can be very dangerous.

→ To ensure reliability, the designer must maintain an equilibrium between cost and the time.

Exp

EMBEDDED SYSTEM DESIGN PROCESSES :

The major steps in the design process are summarized in the top down view as follows.

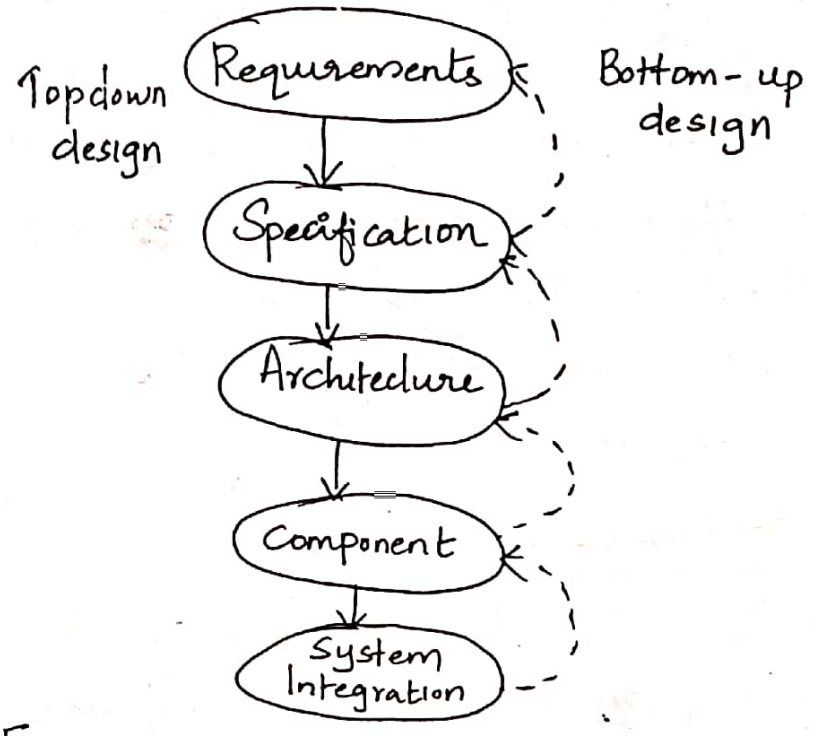


Fig : Major levels of abstraction in the design Process.

Requirements :

→ The requirement phase of the design process capture "what to design".

→ Informal description gathered from customers is known as "requirement".

→ Requirement can be of 2 types ie,

(a) Functional requirements

(b) Non-functional requirements.

→ Some of the non-functional requirement are

① Power Consumption :

→ In the requiremet stage, power can be specfiec in terms of battery life.

→ However, the allowable ~~voltage~~ wattage can't b

defined by the customer.

(2) Physical size and weight :

→ Depending on the application, the physical size and weight of the final system can vary a lot.

→ if the application involves a handheld device then there are constraint on both the size and weight of the device. However, if it is an Industrial control system then there is no Constraint on the size and weight.

(3) Performance

→ The cost and usability of the system are effected by it speed. The performance metrics can be Combination of soft metrics and hard metrics

(4) Cost :

→ The System purchase price or the target Cost is very important. Complete Cost or Simply cost includes the following two Components

a) Manufacturing cost

b) NRE (Non-Recurring Engineering) Cost.

- Manufacturing cost is the cost of the Component
- NRE costs are the cost of hiring personnel and other design related costs.

Requirements Validation :

→ Requirements Validation requires phychological skills, as it deals with Understanding what Custome needs.

→ In the System requirement, the User interface part can be done by creating "Mock-up".

→ In order to stimulate functionality in a restricted area the mock-up make use of scanned data.

→ This mock up can be executed either by PC or work-station.

Requirement form / Requirement chart

→ The requirement form / chart acts like a checklist when the project is in initial stages. A sample form is given below.

Name : ____
Purpose : ____ (what the System Supposed to do
Inputs : ____
Outputs : ____
Functions : ____ (functionality of the System).
Performance : ____
Manufacturing Cost : ____
Power : ____
Physical size & wt : ____

Fig : Sample Requirements form.

Name :- Name not only describe the purpose of the machine, but also helpful while commiting about the project.

Purpose : This should be a brief one or two line description of what the system is supposed to do.

Inputs and Outputs : This field requires information about type of I/O devices, data characteristics and type of data.

Functions : A detailed description about the functionality of the machine is described in this field.

Performance : Inorder to assure proper functionality performance requirements should be identified before hand and they must be measured carefully.

Physical size and weight : Inorder to take architectural decisions, the approximate physical size & weight of the system is important.

Power : The rough idea about power consumption can be very helpful. Decision about whether the system is battery based or non-battery is important here.

SPECIFICATIONS

→ Specifications serves as the contract between the customer and the architect.

→ The specifications must be carefully written so that it accurately reflects the customers requirements.

→ The specification should be understanble enough so that someone can verify that it meets system requirement and overall expectations of the customer.

→ The specification of a GPS system may include the following.

- Data received from the GPS satellite constellatic
- Map data
- User interface
- Operations that must be performed to satisfy customer requirements.

The differences between requirements and specifications are :

Requirements	Specifications
• An informal description gathered from customer	• A contract between custome and architecture.
• This is the first step in design process	• This is the second step in design process.
• Requirements form is used to give a formal listing of requirements	• UML is used to give clear & proper specifications.
• The basic needs to design a system are given by system requirements	• The description of project is given by system specifications.
• Less challenging task	• More complex and challenging task.
• Requirements need not be perfect	• Specifications should be perfect enough in order to develop correct applicatio
• Good psycological skills are required to produce good system requirements	• Good engineering skills are required to produce good specifications.

ARCHITECTURE DESIGN :

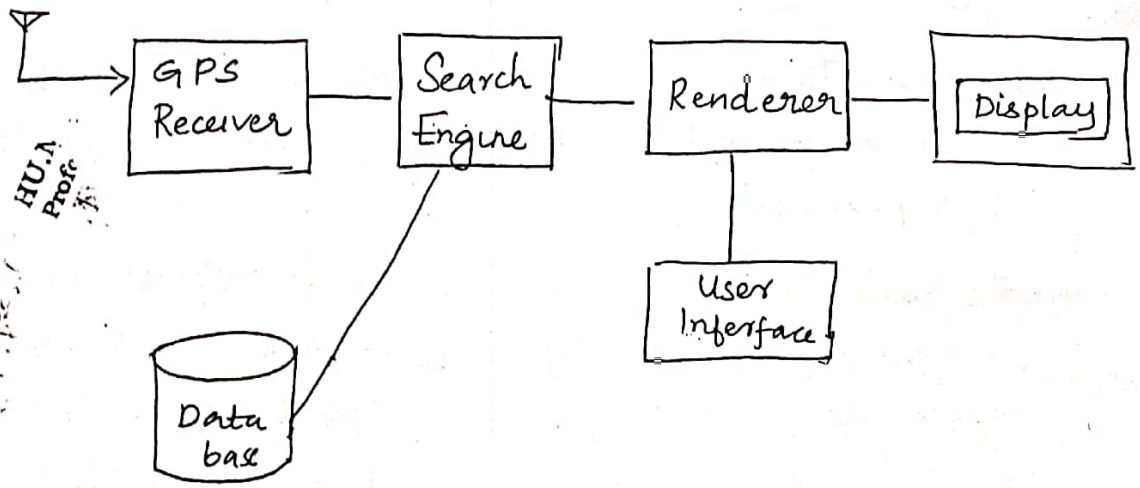
→ The plan used to design the Components of a System is called as an architecture.

→ It deals with how a system will perform the Specific operations. Most of the designers have a perception that creation of the architecture is the first step of the design process.

→ The architecture is the plan for the overall Structure of the System that will be used later to design the Components that make up the architecture

Example :

→ Let us Consider the architecture of GPS moving map in the form of block diagram which shows major operations are



Blockdiagram of moving map

→ The before block diagram does not give clear picture about what operations are to be performed by the Software and Hardware. Inorder to have a clear picture, we have to Separate hardware Operations from Software Operations.

Therefore 2 blocks are drawn here :

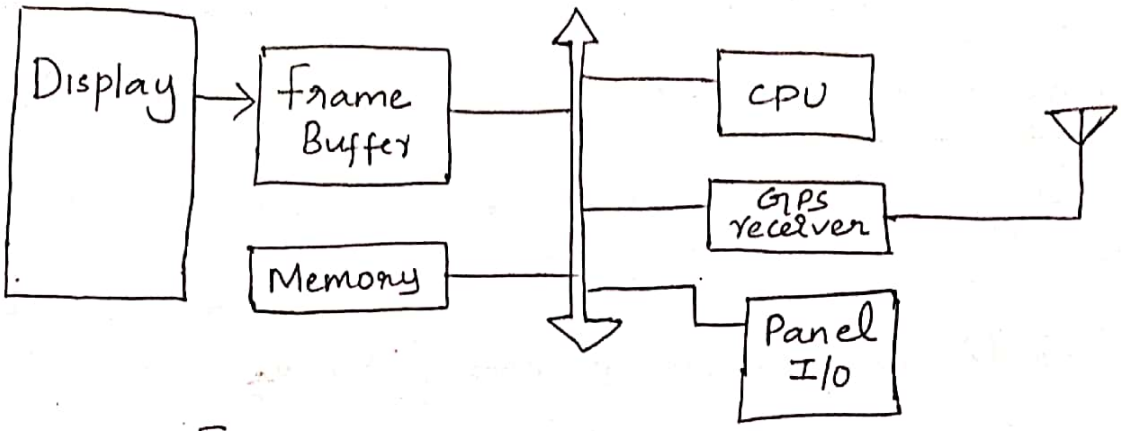


Figure : Hardware Block.

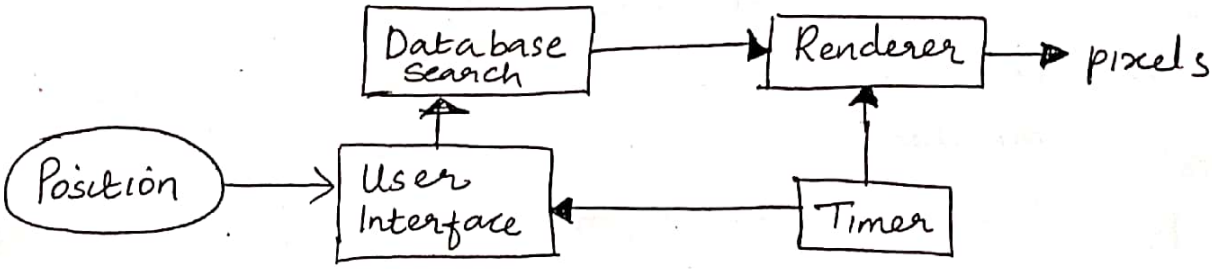


Figure : Software block.

→ The refinement of hardware and software architecture should begin only after the initial architecture is designed.

→ The hardware block diagram depict that memory and I/O devices sourronds the CPU. frame buffer holds the pixels to be displayed and memory for program data which will be used by the CPU.

→ Bus is used to connect all these components.

→ The software block diagram is same as initial architecture but an additional timer is added. This timer is meant for controlling the operations.

→ finally both these block diagram should

- Satisfy user requirements (functional / non functional)
- Include all the required functions.

- should meet speed, cost, power requirements.
- Meet all the specifications.

DESIGNING H/W & S/W COMPONENT :

→ The architectural description tells us about what components we need.

→ The component design effort builds those component in conformance to the architecture & specification.

→ The components will be in general include both hardware - FPGA's, boards and so on and software module.

→ Some of the components will be ready made.

→ The CPU, for example will be standard component in all cases as well as memory.

SYSTEM INTEGRATION :

→ it deals with the integration or assembly of the components created.

→ Debugging is a challenge

→ However, good planning, phase level development good test running at each phase can assist in finding bugs.

→ if bugs are identified and fixed at specified time intervals, then the task of debugging is not big challenge and System Integration become easy.

NEETHU.M
Assistant Professor

APPLICATION OF EMBEDDED SYSTEM :

1. House appliances : Washing machine
2. Automotive Industry : Antilock breaking system (ABS), engine control
3. Home automation & Security systems : Air Conditior, fire alarms.
4. Telecom : Cellular phones.
5. Computer pheripharals : printers, Scanners.
6. Healthcare : EEG, ECG machines
7. Card readers : Barcode, Smart card readers.
8. Embedded System for detecting rash driving on highways :

· Here the speed checker device that identifies rash driving on highways and alarms the traffic authorities if the speed checker finds any vehicle Violating the set speed limits on highways

9. Embedded system for street light Control :

· To detect the movement of vehicles on highways and to Switch on street lights ahead of it, and then to switch off the street lights as the vehicle go past the street lights to Conserve energy.

(PIC microcontroller is programmed by using embedded c)

NEETHU.M
Assistant Professor

(10) Application of embedded System for Home automation System.

→ Here home automation system with the andro application based remote Control.

→ Remote operation is performed by android OS based Smart-phone, upon a graphical user Interface based touch screen Operation.

Topic : Functional building block of embedded S/m

(Refer IInd module : Embedded hardware Units)

(Page : 6 to 8)

Both topics are same.

NEETHU.M
Assistant Professor

MODULE-II

PROCESSORS :

→ A processor is a basic functional Unit in Computer system and using that unit only all the Computations take place.

→ A processor is the heart of the embedded System. So an embedded System designer must have clear knowledge of microprocessors and micro controllers.

→ A processor has 2 essential Units :

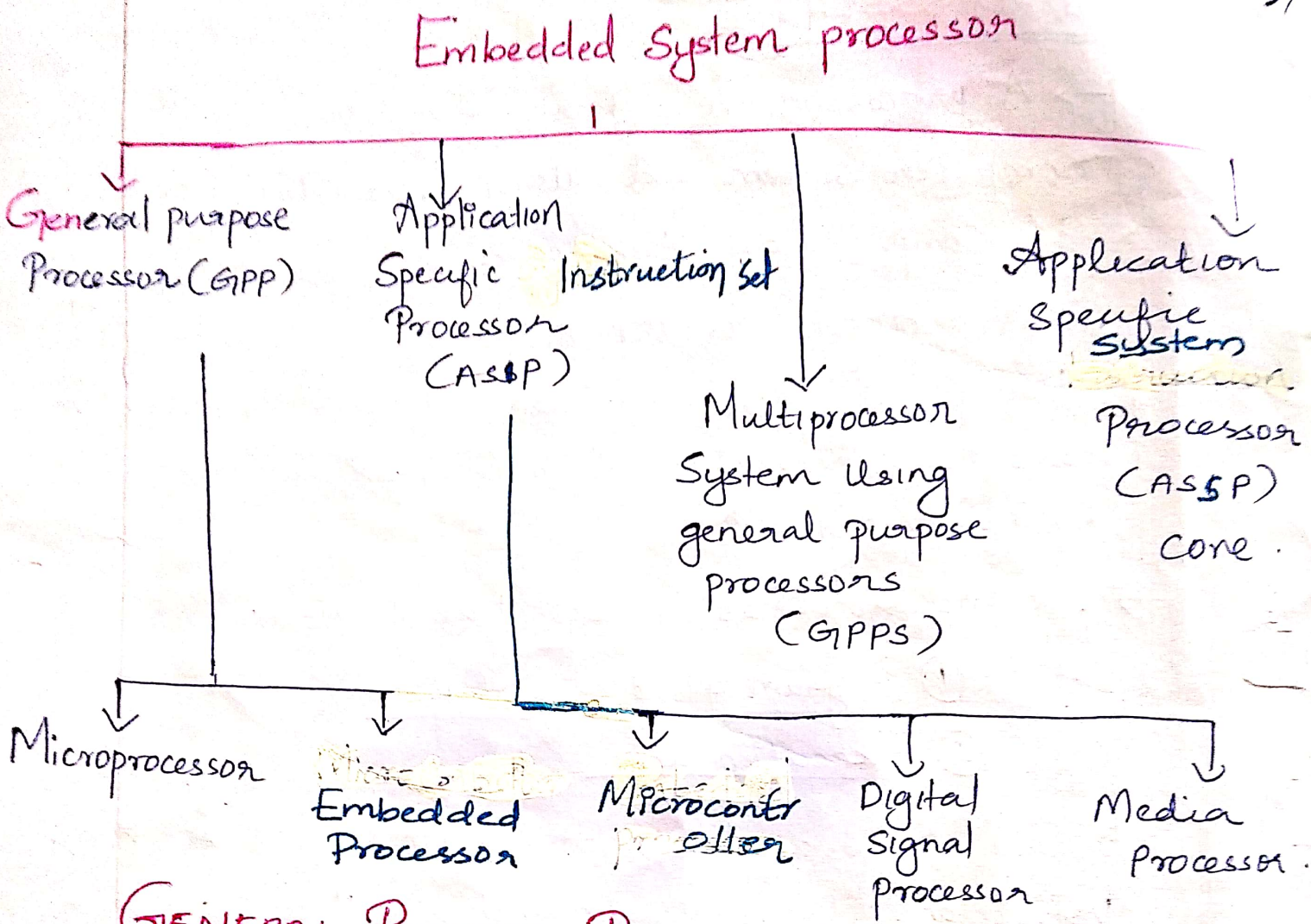
1. Program flow Control Unit (CU)
2. Execution Unit (EU)

→ The CU Includes a fetch Unit for fetching Instructions from the memory. The EU has Circuits that emplement the instruction pertaining to data transfer operations and data conversion from one form to another form.

→ The EU includes Arithmetic and logical Unit (ALU) and also the circuits that execute instructions from a program Control task. It can also execute Instructions for a call or branch to another program and for a Call to a function.

→ A processor is mostly in the form of an IC chip or in Core form in an ASIC. Core means a part of the functional ~~Unit~~ circuit on the VLSI chip.

VARIOUS PROCESSOR :



GENERAL PURPOSE PROCESSOR (GPP)

→ It is a programmable device used in a Variety of applications.

→ Also known as "microprocessor"

→ features : ① program memory
② General datapaths with large register file and general ALU

→ User benefits : ① low time to market.
② High flexibility.

(1) MICRO PROCESSOR :

→ A microprocessor is a single chip semi conductor device also which is a 'Computer on chip', but not a complete computer.

→ its CPU contains an ALU, a program counter, a stack pointer, some working register, a clock timing circuit and interrupt circuit on a single chip.

→ To make complete micro computer one must add memory usually ROM and RAM, memory decoder, an oscillator and no of serial and parallel ports.

→ It has the following instruction set :

1. Instruction for data transfer operations
2. Instruction for ALU operations.
3. Stack operations instruction set.
4. Input and Output (I/O) Operation Instruction set.
5. Program control instruction set.
6. Sequencing and supervising operations Instruction set.
7. General purpose instruction set.

→ A microprocessor is a single VLSI chips that has a CPU and also have some other unit that result in faster processing of instructions.

→ Intel 8085 :- 8 bit processor. It is used in older generation.

→ Intel 8086 OR 8086 } → 16 bit processor

→ Intel 80x86 processors are the 32 bit of 8086.

→ The 'X' means extended 8086 for 32 bits

Eg: 32 bit processors in 80x86 series are Intel 80386 and 80486.

→ The IBM PCs are use 80x86 series of processors and embedded systems incorporated inside the PC for specific tasks use these microprocessors.

→ An example of the new generation 32 and 64 bit microprocessor is the classic pentium series of processors from intel. These have superscalar architecture and also possess powerful ALU's and floating point processing unit

Important Microprosser Used in the embedded Systems:

Stream	Microprocessor family	Source	CISC or RISC or both features.
Stream 1	68 HC xxx	Motorola	CISC
Stream 2	a) 80x86 b) i860	Intel Intel	CISC CISC with RISC
Stream 3	SPARC	Sun	RISC
Stream 4	a) Power PC 601, 604 b) MPC 620	IBM Motorola	RISC

→

→ An RISC processor provides speedy processing of the instructions, each in a single clock cycle.

(a) EMBEDDED PROCESSOR

→ A special microprocessors & microcontrollers often called, embedded processors.

→ An embedded processor is used when fast process fast context switching and atomic ALU operations are needed.

Examples

- ARM 7
- INTEL 1960, AMD 29050

→ For complex real time system normal microprocessors and microcontrollers are not suitable. So special kind of processor known as Embedded processor are required.

→ When a microcontroller or microprocessor is specially designed for complex system it has the following capabilities, then the term embedded processor is prefered instead of microcontroller or microprocessor.

1) Fast context switching and thus lower latencies of the tasks in complex real time applications.

2) Atomic ALU operations & thus no shared data problem

3) RISC core for fast, more precise & intensive calculations by the embedded software

NEETHU.M
Assistant Professor
ECE Dept., NCERC

APPLICATION SPECIFIC INSTRUCTION SET PROCESSOR

(1) DIGITAL SIGNAL PROCESSOR (DSP)

→ DSP is an essential Unit of an embedded System for large number of applications needing processing of signals.

→ Eg: Applications are image processing, multimedia, audio, video, HDTV, DSP modem and telecommunication Processing Systems.

→ The DSP as a GPP is a single chip VLSI Unit. It possesses the computational capabilities of a microprocessor and also has a multiply and accumulate Units.

→ A DSP provides fast, discrete time, signal processing Instructions. It has very large Instruction word (VLIW) processing capabilities. It processes the following thing in fast manner.

1) Single Instruction multiple data (SIMD)
2) Discrete Cosine transformations (DCT)
3) Inverse discrete Cosine transformations (IDCT)

Important DSP used in the embedded Systems.

Stream	DSP family	Source
Stream 1	TMS320C$_{xx}$, OMAP	Texas
stream 2	Tiger SHARC	Analog device
Stream 3	5600xx	Motorola
Stream 4	PNX 1300, 1500^2	Philips.

(2) MICROCONTROLLER :

→ A microcontroller is a functional computer system on a chip.

→ It contains a processor, memory and programmable Input/Output peripherals.

→ Microcontrollers include an Integrated CPU.

→ A microcontroller is used when a small or part of the embedded software has to be located in internal memory and when the on-chip functional units like Interrupt handler, port, timer, ADC are needed.

→ Microcontrollers are particularly suited for use in embedded systems for real time control applications with on-chip program memory and devices.

Important Microcontrollers used in the Embedded Systems :

Stream	Microcontroller family	Source	CISC or RISC or both features
Stream 1	68 HC11xx, HC12xx, HC16xx	Motorola	CISC
Stream 2	8051, 80251	Intel	CISC.
Stream 3	80X86	Intel	CISC
Stream 4	PIC 16F84 or 16C76 16F876 and PIC 18	Microchip	CISC
Stream 5	Enhancements of ARM9, ARM 7	ARM, Texas etc	CISC with RISC core

NEETHU.M
Assistant Professor
ECE Dept., NCERC

Small scale Embedded System Microcontroller

- 68HC05
- 68HC08
- PIC 16F8X
- 8051

Medium Scale Embedded System Microcontroller

- 8051
- 80251
- 80x86
- 68HC11XX
- 68HC12XX
- 80196

Large scale Embedded System

- Intel 80960CA
- ARM7
- POWER PC MPC 604

NEETHU.M
Assistant Professor
Dept., NCERC

Commonly used Microcontroller in small, medium & large scale embedded Systems.

MICROPROCESSOR Vs MICROCONTROLLER

MICROPROCESSOR	MICROCONTROLLER
The functional blocks are ALU, registers, timing & Control Units	It Includes functional blocks of microprocessors and in addition it has timer, parallel I/O, RAM, EPROM, ADC & DAC.
Bit handling instruction is less, one or two type only	Many type of bit handling Instructions.
Rapid movements of code & data between external memory & Microprocessor.	Rapid movements of code & data within microcontroller.
It is used for designing general Purpose digital Computers s/m	They are used for designing application specific dedicated systems.

MEDIA PROCESSOR :

→ Media processor is a new innovative processor with high system performance for real time video performance, audio processing and data streaming. it also known as video processor.

→ Media processor facilitates a seamless fusion of mobile telephony with broad band Internet. it facilitates voice based web accesses, Speech recognition, text to speech Conversation, VOIP (Voice over Internet protocol) and voice based version of mobile net standard XML.

→ A media processor should provide for the following Processing functions :

1. VLIW, fixed as well as floating point arithmetic
2. Discrete Cosine transformation (DCT) processing Unit
3. Quantiser Unit (Quantisation means analog signals from camera or microphone interface signals being converted to digital quantized output after encoding).
4) Processing of library functions for graphics, 2D text, MPEG & Motion JPG.
5) Image color and hue correction, image rotation, image scaling, shadow enhancement, detecting image edges & sharpening the image.
6) Video encoding which preprocesses for noise reduction and then controls the rate of transmission after estimating the motion picture rates, compresses, Synchronizes audio & finally bet streams are sent to a streaming network.

NEETHU.M
Assistant Professor
ECE Dept., NCERC

7). Video decoding, which receives the bit stream, decompresses and separates audio and video eliminates noise by pre processing.

8) Noise reduction and echo cancellation.

APPLICATION SPECIFIC SYSTEM PROCESSOR (AS

→ ASSP is dedicated to specific tasks and provide a faster solution.

→ An ASSP is used as an additional processing unit running the application in place of using embedded Software.

Examples : IIM7100, W3100A.

MULTIPROCESSOR SYSTEM USING GPPS :

→ Multiprocessors are used when a single processor does not meet the needs of difficult task.

→ The operations of all the processors are synchronized to obtain an optimum performance.

EMBEDDED HARDWARE UNITS & DEVICES IN A SYSTEM

(1) 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

- The propagation delay in the gates is inversely proportional to operational voltage; therefore, **the 5 V system is used in most high performance systems.**
- **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.

(2) **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.**

(3) **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.

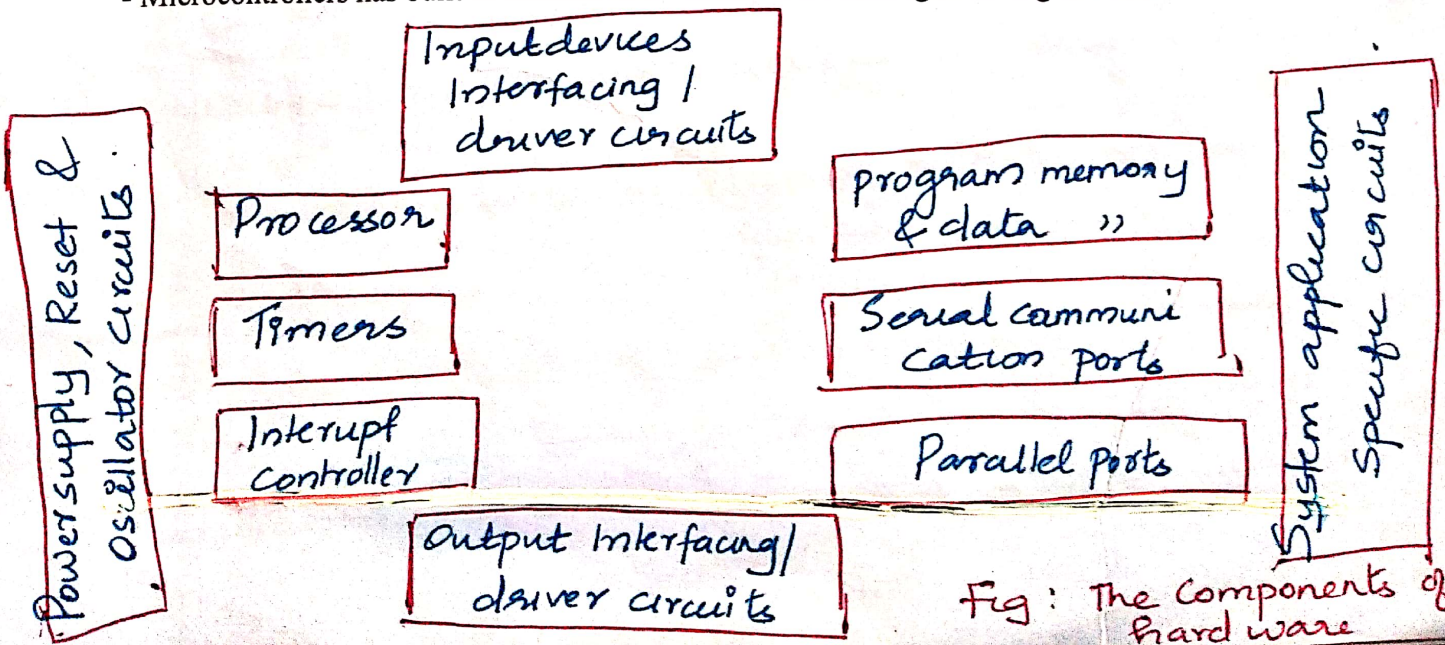


Fig : The Components of ES hardware

(4) Reset Circuit, Power-up Reset & Watchdog-Timer Reset

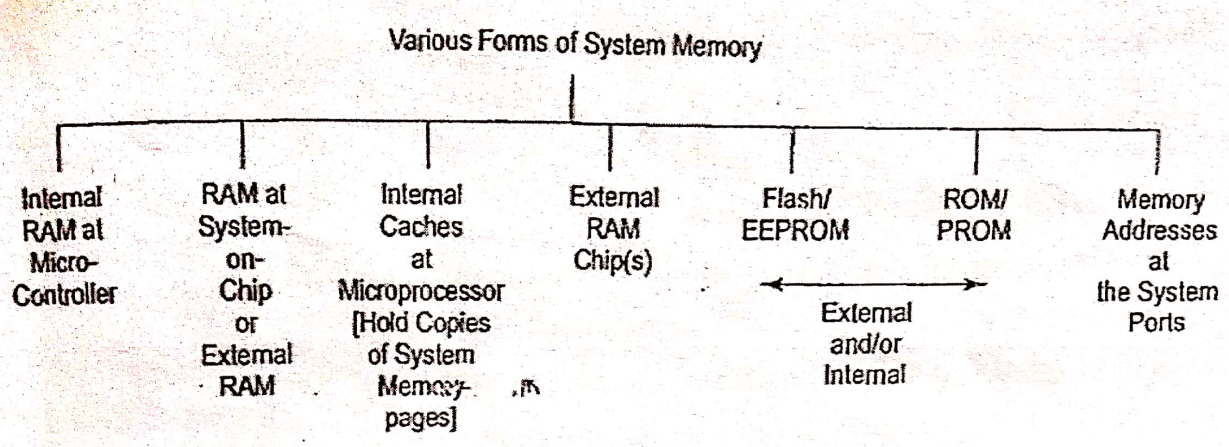
- Reset can be activated by an **external reset circuit that activates on power-up** *(switching-on)* **the system.**
- The reset circuit is a simple **circuit** ***(such as an RC circuit)*** **whose output connects to the reset pin of the processor.**
- To reset a processor, the **reset circuit should 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 time-out 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.

(5) Memory

- In a system, there are **various types of memory** and the figure shows a chart for various forms of memory that are present in systems.



1. **Internal RAM** *(of size in bytes)* **as internal registers in a microcontroller,** temporary data storage (intermediate results/variables) & **stack memory area.**
2. **Internal ROM/** PROM *(of size in kB)* for **storing application programs** in the case of **microcontrollers.**
3. **External RAM** for the **temporary data storage and stack** in the case of **microprocessors.**
4. **Internal flash** as 'non-volatile' memory **to save the results after processing.**
5. **Memory stick** ***(or memory card)*** as **large storage** ***(such as video, images, songs)*** in digital camera & mobile phones.
6. **External ROM/ PROM** for **embedding software programs** in almost all embedded systems **except in microcontroller based embedded systems**
7. **Buffers memory RAM** at ports.
8. **Caches or cache memory** *(for storing copies of frequently used instructions & data)*

(6) Input, Output and 1O Ports, IO Buses and IO interfaces

- **The system gets inputs from physical devices through the input ports.**
- **Examples of inputs are**

  (1) **A system gets inputs from the touch screen, keys in a keyboard, sensor circuits etc**

  (2) A network card receives the input signals from a communication device such as a modem

  (3) Ports receives inputs from a peripherals like USART.

- **A processor identifies each input port by its memory addresses called port addresses.**
- **Just as a memory location, each input port is also identified by its address.**
- The system gets the inputs by the **read operations at the port addresses.**

- **The system has output ports through which it sends output bytes to the real world.**

  (1) Output may be sent to an **liquid crystal display** *(LCD)* **or touch screen display panel or light emitting diode** *(LED)*

  (2) A system may send the output to a **printer.**

  (3) Output may be sent to a **communication device such as a modem.**

  (4) Some systems send the outputs to **alarms, actuators etc.**

  (5) A robot sent outputs for its various motors.

- **Each output port is identified by its memory addresses** *(called port addresses)*
- **The system sends the output by a write operation to the port address.**
- **There are also general-purpose ports for both the input & output operations***(IO ports)*
  - **For eg., a touch screen** sends output as well as gets input when a user touches displayed key on the screen.
- **Each IO port is also identified by an address to which the read and write operations both take place.**
- **Ports can also be serial or parallel communication**
- In serial communication a one-bit data line is used and bits are sent serially in successive time slots.
- Universal Asynchronous Receiver & Transmitter *(UART)* **is a popular serial communication.**
- In parallel communication, several data lines are used and bits are sent in parallel.
- **A system connects to external devices and systems through parallel or serial I/O ports.**

Bus

- A system has to be connected to a number of other external devices or systems.
- **A bus consists of a common set of lines to connect multiple devices,** hardware units and other similar systems for communication.
- **A bus may be a serial or parallel bus that transfers data bit/bits.**
- **A "protocol" must be there to specify how signals communicate on the bus.**
- The protocol specifies **(i)** how bus is shared when several devices need to communicate through the bus (bus arbitration);

  (ii) Ways of polling a bus from each device at an instance;

  (iii) Ways of daisy chaining the devices so that bus is granted to a device according to the device-priority.

- **A system networks to the other devices & systems through an I/O bus using different types of serial & parallel bus protocols such as 12C, CAN, USB, ISA, EISA & PCI bus.**

(7) DAC and ADC

- For controlling & **signal processing applications**, a system provides necessary **interfacing circuits for** the Digital to Analog Conversion ***(DAC)*** unit and Analog to Digital Conversion **(ADC)** unit.
- A DAC operation is done with the help of a combination of a PWM unit in the microcontroller and external integrator chip.
- ADC operations are required for **data acquisition, image processing, voice processing, video processing,** instrumentation and automatic control systems.

(8) LCD, LED & Touchscreen Displays

- The system may need the **necessary interfacing circuit and software to output** to the LCD display, the LED interfacing ports or for the touchscreen I/O.
- A system has to display status messages in **single line or in multiline.**
- **An LCD screen can show multiline display of characters, small icons etc.**
- To indicate **the ON status of the system**, there may be an **LED** that glows.
- A **flashing LED** can indicate that a **specific task is under completion or is running or in wait** status.
- **A touchscreen is an input as well as an output device**, which can be used to enter a command, a chosen menu or to give a reply.
- The information is input by physically touching at a screen position using a finger.
- **The „touch-screen" displays the choices or commands, menus and icons.**

(9) Keypad/ Keyboard

NEETHU.M
Assistant Professor
ECE Dept., NCERC

- The keypad or keyboard is an important device **for getting user inputs.**
- For inputs, a keypad or **keyboard may interface to a system.**
- **Necessary interfacing circuit & software must be provided to receive inputs directly from the keys or through a keypad controller IC.**
- **A keypad has upto a max. of 32 keys** while **a keyboard may have 104 keys or more.**
- The keypad or **keyboard may interface serially or parallely** to the processor **through ports and a keypad controller IC.**

(10) Pulse Dialer, Modem & Transceiver

- **In communication systems, a pulse dialer, modem or transceiver is used.**
- For user connectivity through telephone line network, **system provides the necessary interfacing circuit and software for dialing of the modem and transceiver.**
- **A transceiver is a circuit that can transmit as well as receive byte streams.**

(11) Interrupt Handler

- A system may possess a no. of devices connected as interrupts and **the processor has to control and handle** the requirements of each device by **running an appropriate** ISR ***(interrupt service routine).***

- **An interrupts-handling mechanism must exist in each system to the handle interrupts from various sources** like external physical devices, software instructions etc. by executing the ISRs and for handling multiple interrupts simultaneously.

- Important points regarding the interrupts and their handling by the program are as follows.

1) There can be a no. of interrupt sources in a processor.

- An **interrupt may be a hardware signal** that indicates the occurrence of an event.
- An interrupt may also occur **through timers, serial communication etc.**
- The **interrupt may arise due to an illegal op-code fetch, a division by zero result** or an overflow during an ALU operation.
- A **software interrupt** may arise in an **exceptional condition that may have developed while running a program.**

2) The system may prioritize interrupt sources and service them accordingly.

3) Certain interrupt sources are not maskable and cannot be disabled.

- ***Some interrupt are assigned the highest priority during processing***

4) The processor's current program has to divert to a interrupt service routine to complete that task on the occurrence of the interrupt.

5) There is a on-chip unit for the interrupt handling mechanism in a microcontroller.

6) The system always gives priority to the ISRs over the tasks of an application.

-The operating system is expected to control the handling of interrupts and running of routines for the interrupts in a particular application.

EMBEDDED SOFTWARE COMPONENTS

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.

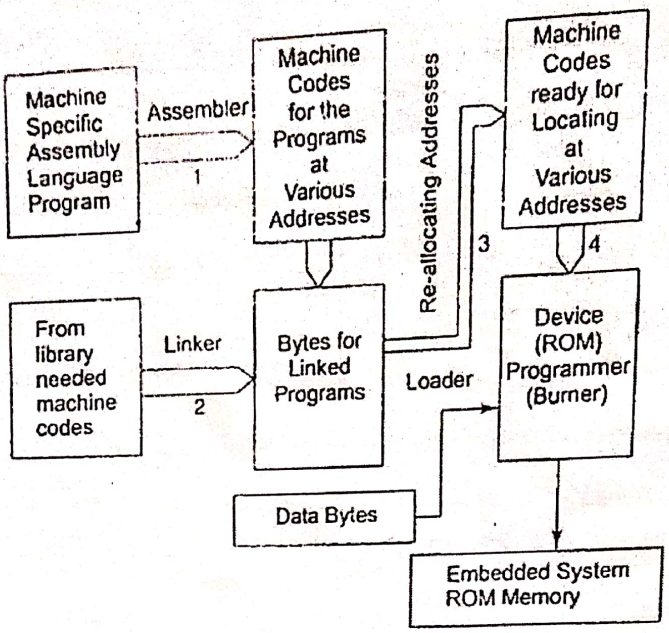
Machine-code based coding

- In machine-code based coding, the **programmer defines the machine code bytes corresponding 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**

Coding in Assembly Language

- **Small programs can be coded in assembly language after understanding the processor & its instruction set.**
- These codes are also **called low-level codes**
- **"Assembler" is the software used** for converting the codes written in assembly language *(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
- Figure shows the process of **converting an assembly language program into machine implementable software file** and then finally obtaining a **ROM image file**



The process of converting an assembly language program into the machine codes and finally obtaining the ROM image

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

(1) 1st step is called **"Assembling" in which an assembler software translates the assembly software into the machine codes**

(2) Next step is called **linking; a linker links these codes (if necessary) with the other codes taken from the library**

- 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)*

(3) 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"*

(4) 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**

(5) **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**

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 wholw embedded system when coding in high level language**
- Figure shows the process of **converting a C program into the ROM image file**

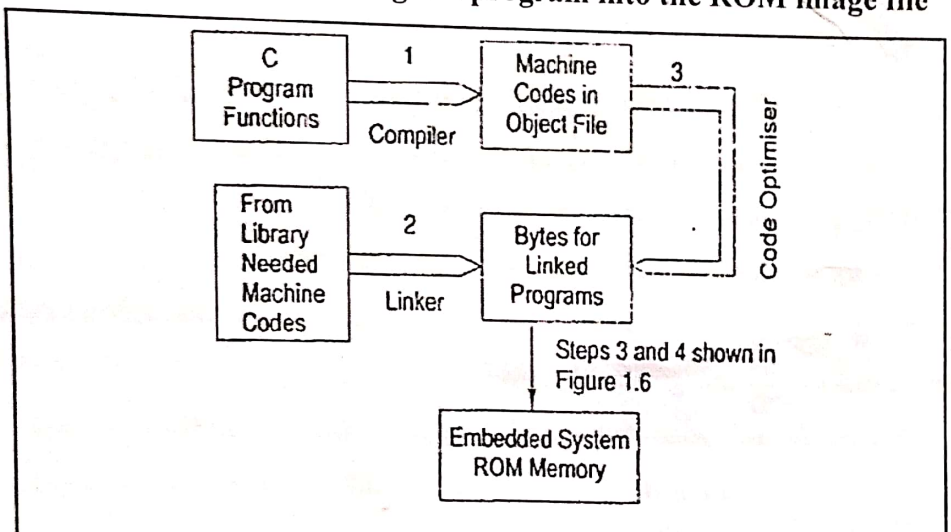


Fig. 1.8 The process of converting a C program into the file for ROM image

- First, **the compiler software generates the object codes** ***(file with .OBJ extension)***
- **The compiler assembles 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 in previous section of assembling process ***(Locating, Loading & Burning)***

SOFTWARE TOOLS IN ES design :

→ There are n number of tools used in designing an embedded system.

→ it is used in assembly language program & high level language programe

① Editor

Editor is used for writing c codes or assembly mnemonics using the keyboard of the PC for entering the program.

→ it allows the entry, addition, deletion, insert, appending previously written lines or files, merging record and files at specific positions.

② Interpreter

→ Converts the line by line translation to the machine executable codes.

NEETHU.M
Assistant Professor
ECE Dept., NCERC

③ Compiler

→ it converts the complete set of codes into machine executable codes. it creates object file.

④ Assembler

It translating the assembling mnemonics into binary op codes. It also creates a list file that can be printed. The list file has address, source code & hexadecimal object codes.

(5) Cross Assembler :

→ It converts object codes or executable Code a processor to other codes for another processor & vice versa.

(6) Simulator :

→ To simulate all functions of an ES circuit Includi additional memory and peripherals.

→ It is independent of a particular target system.

→ It also simulates the processes that will execute when the codes execute on the targeted particular processor.

(7) stethoscope

→ used for dynamically tracking the changes in any program variable.

- It tracks the changes in any parameter. It demonstrate the sequences of multiple processes that execute.
- It also records the entire time history.

(8) Trace scope

→ used to help in tracking the changes in the modules & task with time on the x-axis.

→ A list of actions also produces the desired time scales & the expected times for different tasks.

EMBEDDED SYSTEMS ON A CHIP (SoC) & USE OF VLSI CIRCUITS

A System On chip

NEETHU.M
Assistant Professor
NCERC

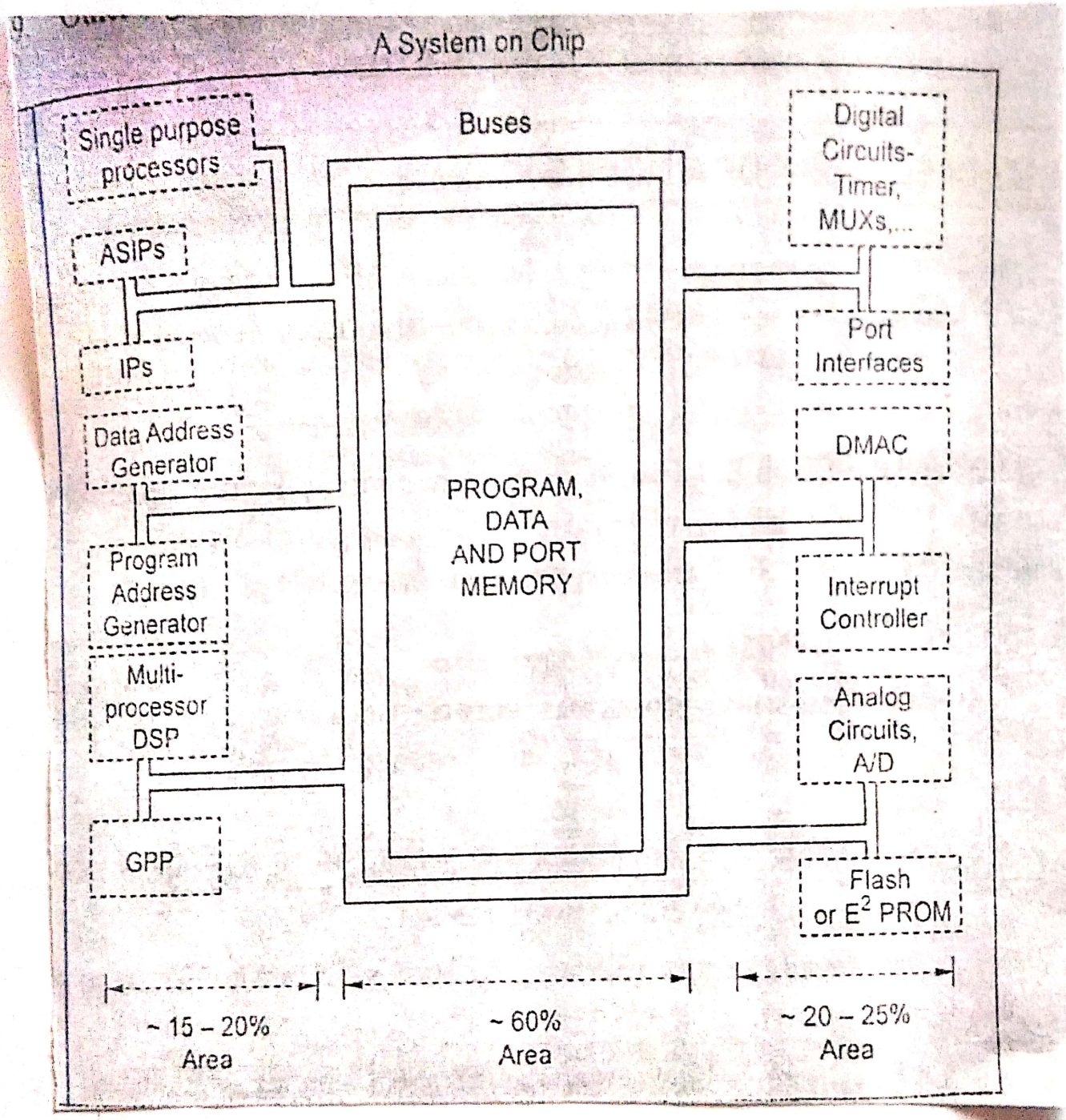


Fig: SoC designed with System Integration of Software & Communication processor, dual core Processor Including graphics processors, ASIP, IP's, program, data & Port memory, and Peripheral Interfaces on a common bus.

→ Embedded System are being designed on a Single Silicon chip called System on chip (SoC)

→ SoC is a system on a VLSI chip that has all the necessary analog as well as digital circuits, Processors and software.

→ A SoC may be embedded with the following Components.

1. Embedded processor GPP or ASIP Core
2. Single purpose processing Cores or multiple processors.
3. A network bus protocol Core.
4. An encryption function Unit.
5. Discrete Cosine transforms for Signal processing applications
6. Memories
7. Multiple standard source solutions, called IP (Intellectual property) Cores.
8. Programmable logic device & FPGA (field programmable gate Array) Cores.
9. Other logic & analog Units.

→ Application of such an embedded SoC is the mobile phone.

→ Single purpose processors ASIPs and IPs on an SoC are Configured to process encoding and deciphering, dialing, modulating, demodulating, Interfacing the key pad and multiple line LCD

LCD matrix displays or touch screen, storing data input and recalling data from memory.

→ Fig. shows an SoC that integrates Internal ASICs, Internal processors (ASIPs), Shared memories and peripheral Interfaces on a Common bus.

→ Besides a processor, memories and digital circuits with embedded Software for specific applications, the SoC may possess analog circuits as well.

Application Specific IC (ASIC)

→ ASIC's are designed using the VLSI design tools with the processor GPP or ASIP and analog Circuits embedded into the design.

→ The designing is done using electronic Design Automation (EDA) tool.

NEETHU.M
Assistant Professor
ECE Dept., NCERC

IP core

→ On a VLSI chip, there may be integration of high level Components.

→ These components possess gate level Sophistication in circuits above that of the Counter, register, multiplier, floating point operation Unit and ALU.

→ A standard source solution for synthesizing a higher level component by Configuring an

FPGA Core or a Core of VLSI circuit may be available as an Intellectual propery called IP.

NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(ACCREDITED BY NAAC)

PAMPADY, THIRUVILWAMALA, THRISSUR (DT), 680588

MR 405 EMBEDDED SYSTEM

MODULE III

COURSE MATERIALS

SIGNATURE OF HOD

CUSTOM SINGLE PURPOSE PROCESSORS :

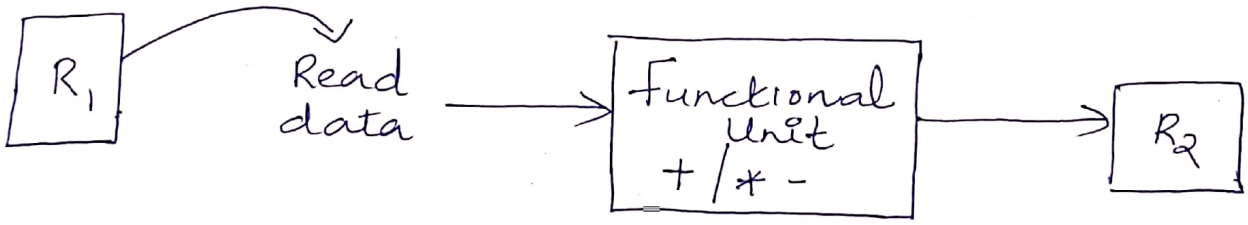
→ Processor Consist of 2 parts

- Controller
- Datapath .

→ Controller : Controls the overall transfer of data through the data path .
(ie store & manipulate data)

→ Data path : Can read data from a particular register and feed the data through functional Unit to carry out a particular operation and stores back to the particular register .

NEETHU.M
Assistant Professor
..., NCERC



→ Single purpose processor is a digital system Intended to solve a Specific Computation task .

→ Custom single purpose processor is to execute a specific task within our embedded system .

Eg : Digital Camera .

Benefits of using custom single purpose processor :

① performance may be fast , due to fewer clock cycles resulting from a customized data path .

② Due to shorter clock cycles resulting simp functional units

③ Less multiplexers

NEETHU.M
Assistant Professor
Dept., NCERC

④ Simpler control logic.

⑤ Size may be small due to a simpler datapath and no program memory.

⑥ The processor may be faster and smaller than a standard one implementing the same functionality, since we can optimize the implementation for our particular task.

BASIC TECHNIQUES FOR DESIGNING CUSTOM PROCESSORS :

For this

- start with a review of combinational and sequential design.
- Describe a method for converting programs to custom single purpose processors.

COMBINATIONAL LOGIC DESIGN :

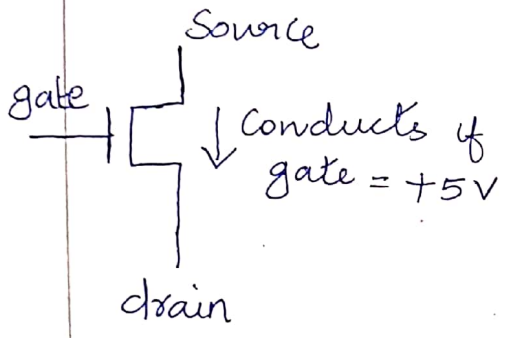
→ A transistor is the basic electrical component of digital systems.

→ Combinations of transistors form more abstract component called logic gates, which designers primarily use when building digital systems.

→ A transistor acts as a simple ON/off switch.

→ CMOS (Complementary Metal oxide Semi Conductor), Combination of P-MOS & N-MOS.

CMOS Implementation of some basic logic gates :-



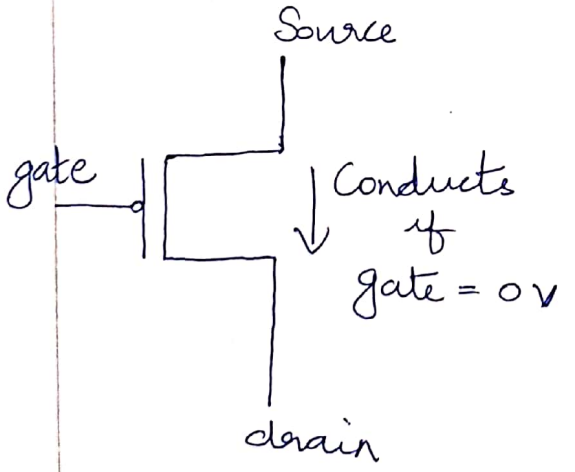
Fig(a) : n-mos transistor

→ Gate controls whether or not current flows from source to drain

→ When a high voltage (typically +5V), which is refered to as logic 1 is applied to the gate, the transistor conducts so current flows.

→ When a low voltage (logic 0) is applied to gate, transistor does not conduct.

We can also build a transistor with the oposite functionality :



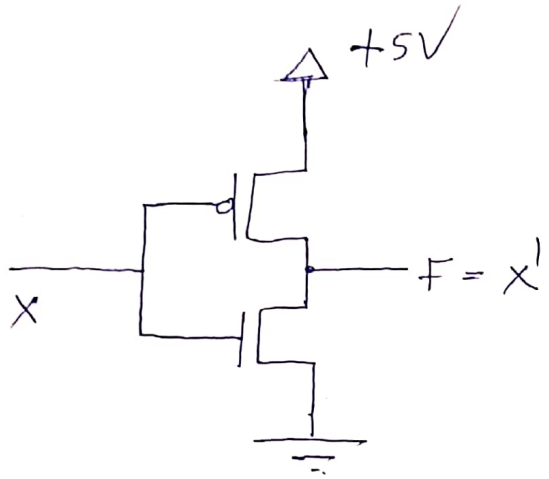
Fig(b) : P-mos transistor

→ When logic 0 is applied to gate, the transistor conducts

→ When logic 1 is applied to the gate, the transistor does not conduct.

NEETHU.M
Assistant Professor
Dept., NCERC

Given these 2 basic transistor, we can build a circuit whose output inverts the gate input as shown in fig(c): INVERTER

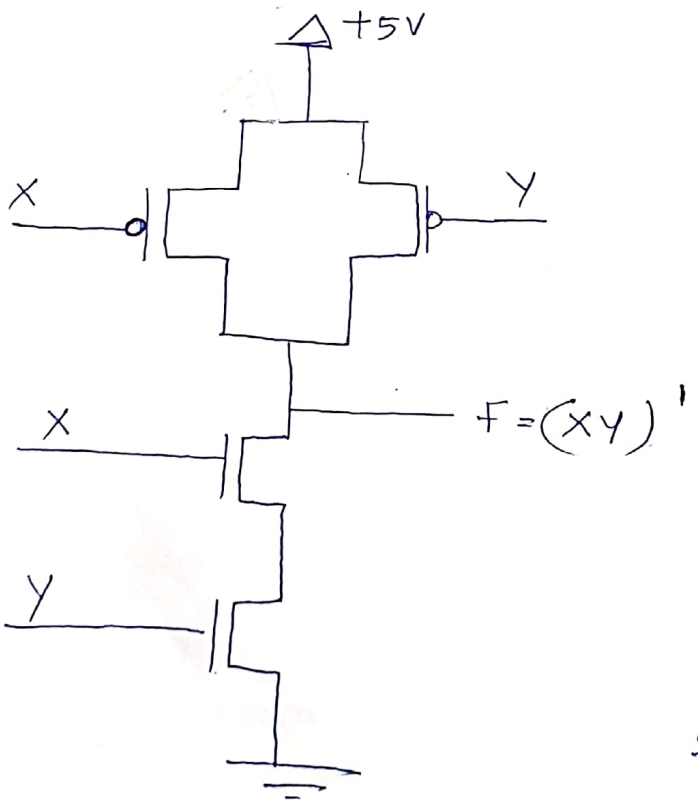


X	F
0	1
1	0

Fig(c): Inverter.

- When x = 0, top trans conducts, so logic 1 appears at the output F (bottom ttr does no

We can easily build a circuit whose output logic 1 when atleast one of its input is logic 0.



X	Y	F
0	0	1
0	1	1
1	0	1
1	1	0

→ When atleast one of the inputs X & Y is logic 0, then atleast one of the top transistor conducts (and the bottom does not)

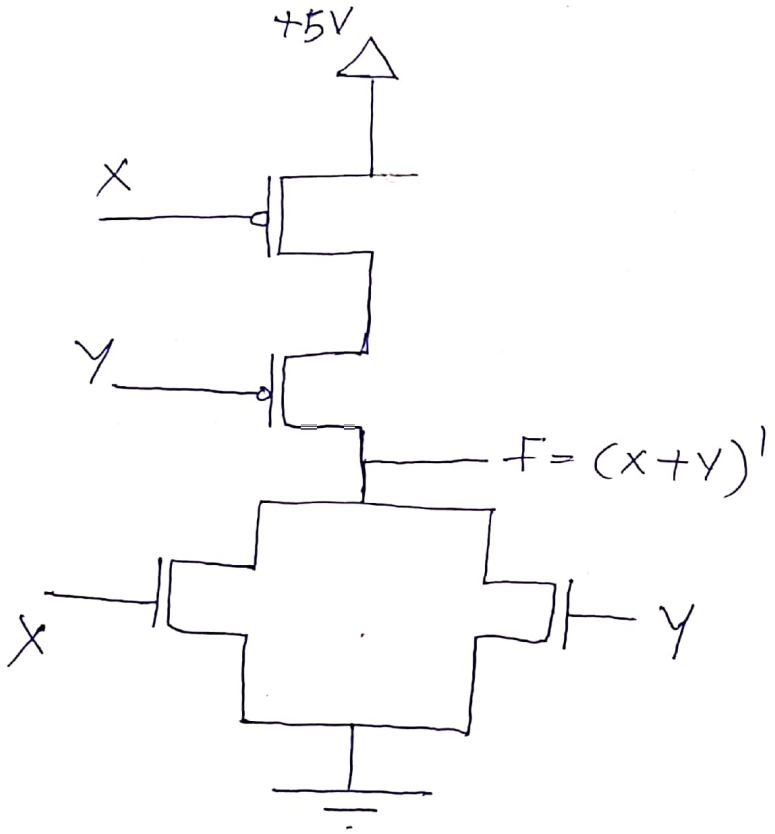
Fig(D): NAND gate

NEETHU.M
Assistant Professor
NCERC

So logic 1 appears at F.

→ If both Inputs are logic 1, then neither of the top transistors conducts, but both of the bottom ones do, so logic 0 appears at F.

We can easily build a circuit whose output is logic 1, when both of its inputs are logic 0.

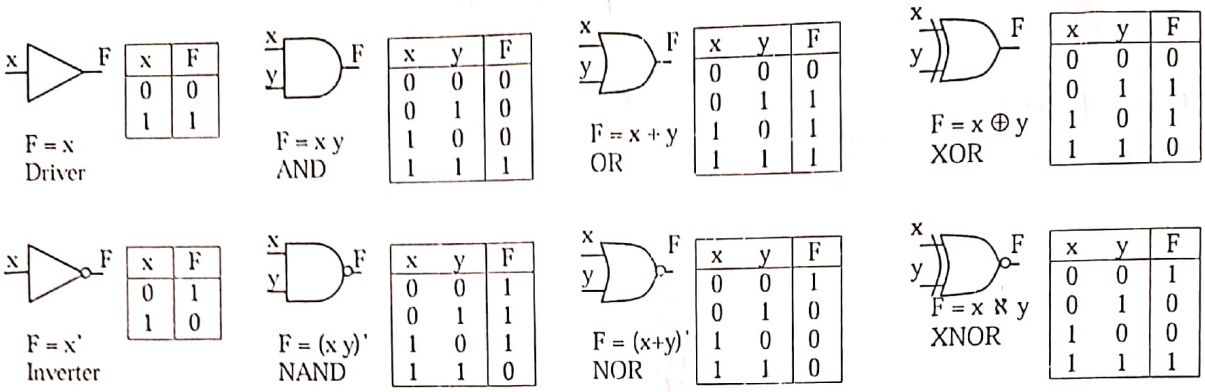


X	Y	F
0	0	1
0	1	0
1	0	0
1	1	0

fig (E) : NOR gate

→ The 3 circuits shown implement 3 basic logic gates : an inverter, a NAND gate and a NOR gate.

Figure 4.2: Basic logic gates



x	F
0	0
1	1

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

x	F
0	1
1	0

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

NEETHU.M
Assistant Professor
, NCERC

→ Digital system designers usually work with logic gates, not transistors. Each gate is represented symbolically, with a boolean equation, and with a truth table.

→ A <u>combinational circuit</u> is a digital circuit whose output is purely a function of its current inputs; such a circuit has no memory of past inputs.

<u>Simple technique to design a combinational circuit using basic logic gates</u>:

<u>Step 1</u>: problem description, which describes the outputs in terms of inputs.

Step 2: Translate the description to a truth table with all possible combinations of input values on the left and desired output

~~terms of the inputs~~.

Output values on the rugh.

Step 3 : For each output column, we can derive an output equation, with one term per row.

Step 4 : Minimize the output equations by algebrically manipulating the eqn's. (use Karnaugh maps).

Step 5 : Draw the circuit diagram.

NEETHU.M
Assistant Professor
., NCERC

Figure 4.3: Combinational logic design.

(a) Problem description

y is 1 if a is equal to 1, or b and c is equal to 1. z is 1 if b or c is equal to 1, but not both. If a=b=c=1 y=z=1

(b) Truth table

Inputs			Outputs	
a	b	c	y	z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

(c) Output equations

y = a'bc + ab'c' + ab'c + abc' + abc

z = a'b'c + a'bc' + ab'c + abc' + abc

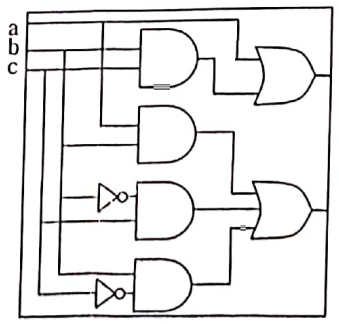
(d) Minimized output equations

y: a \ bc	00	01	11	10
0	0	0	1	0
1	1	1	1	1

y = a + bc

z: a \ bc	00	01	11	10
0	0	1	0	1
1	0	1	1	1

z = ab + b'c + bc'



→ Although we can design all combination circuits in the above manner, large circuits would be very complex to design.

→ A circuit with 16 inputs would have 2^{16} or 64K rows in the truth table.

→ One way to reduce the complexity is to use components that are more abstract than logic gates.

→ Several such components are

① Multiplexer / selector

→ Allows only one of its data input Im to pass through to the output O.

→ Multiplexer acts much like a rail road switch, allowing only one of multiple input tracks to connect to a single output track.

Eg: 8:1 MUX
4:1 MUX.

→ If there are m data inputs, then there are $\log_2(m)$ select lines S and we call this an m by 1 multiplexer (m data inputs and one data output).

→ The binary value of S determines which data input passes through:

00 ; 00 means I0 may pass
01 ; 01 " I1 " "

(2) Decoder

→ Decoder converts its binary input I into a One-hot Output O.

→ "One-hot" means that exactly one of the Output lines can be 1 at a given time.

→ Thus, if there are n-Outputs, then there must be $\log_2(n)$ inputs.

→ Eg: 3x8 decoder

3 inputs & 8 Outputs.

NEETHU.M
Assistant Professor
, NCERC

(3) Adder:

→ An adder adds two n-bit binary inputs A & B generating an n-bit Output sum along with an Output carry.

Eg: 4 bit adder

A = 1010 & B = 1001, then sum S = 0011 and carry = 1.

(4) Comparator

→ A Comparator Compares two n-bit binary inputs A & B generating Outputs that Indicate whether A is less than, equal to or greater than B.

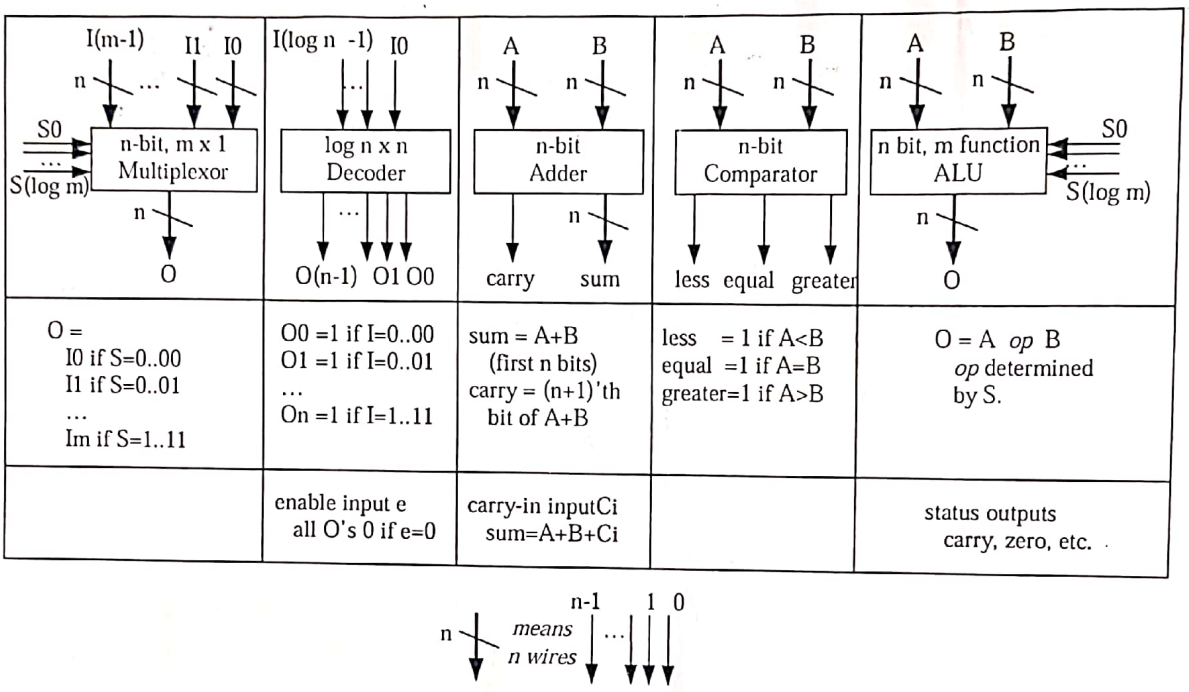
(5) ALU (Arithmetic logic Unit)

→ Can perform a Variety of arithmetic &

logic functions on its n-bit Input A and

→ Common functions Includes addition, Subtraction, etc.

Figure 4.4: Combinational components.



SEQUENTIAL LOGIC DESIGN:

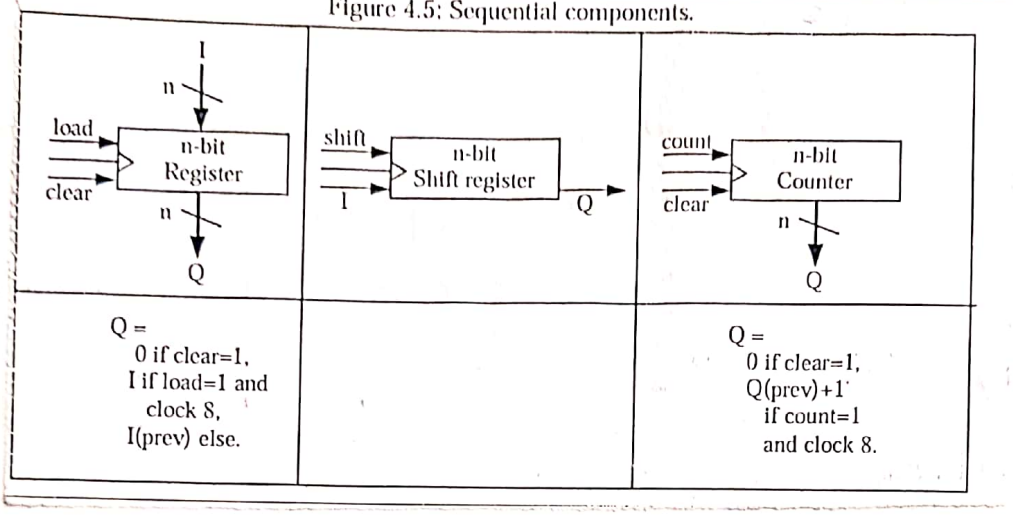
→ A sequential circuit is a digital circuit whose Outputs are function of the current as well as previous inputs values.

→ It possesses memory.

→ The Sequential components are shown in figure below Fig (4.5)

NEETHU.M
Assistant Professor
, NCERC

Figure 4.5: Sequential components.



→ One of the basic sequential circuits is the flip flop.

→ A flipflop stores a single bit.

The Sequential Components are

① <u>Register</u> :

→ A register stores n-bits from its n-bit data I, with those stored bits appearing at its output O.

→ A register usually has atleast two control inputs, clock and load.

→ For a rising edge triggered register, the Input I are only stored when load is 1 and clock is rising from 0 to 1.

→ clear : resets all bits to 0, regardless of the Value of I

→ Because all n bits of the register can be stored in Parallel, we often refer to this type of register as a Parallel load register

③ Shift register

→ Stores n bits, but these bits cannot be stored in parallel. Instead they must be shifted into the register serially, meaning one bit per clock edge.

→ A shift register has a one-bit data input I and atleast two control inputs clock and shift.

→ When clock is rising and shift is 1, the value of I is stored in the nth bit, while the nth bit is stored in the (n-1) th bit and likewise, until the second bit is stored in the first bit.

→ The first bit is typically shifted out, meaning it appears over an output Q.

④ Counter

→ A counter is a register that can also increment (add binary 1) to its stored binary value.

→ A counter has a clear input, which resets all stored bits to 0, and a count input, which enables incrementing on the clock edge.

→ A counter often also has a parallel load data input and associated control signal.

→ A common counter feature is both up and down counting (incrementing & decrementing) required an

additional Control Input to Indicate the count direction.

→ Control inputs can be Synchronous or asynchronous

→ Synchronous : Inputs value only has an effect during a clock edge.

→ Asynchronous : Inputs Value affects the circuit Independent of the clock.

→ clear Control lines are asynchronous.

Steps for designing sequential logic :

Step 1 : Problem description

Step 2 : Translate » to a state diagram.

Step 3 : Each state represents the current "mode" of the circuit, Serving as the circuits memory of past input values.

Step 4 : The desired Output Values are listed next to each other state.

Step 5 : The Input Conditions that cause a transition from one state to another are shown next to each arc.

NEETHU.M
Assistant Professor
, NCERC

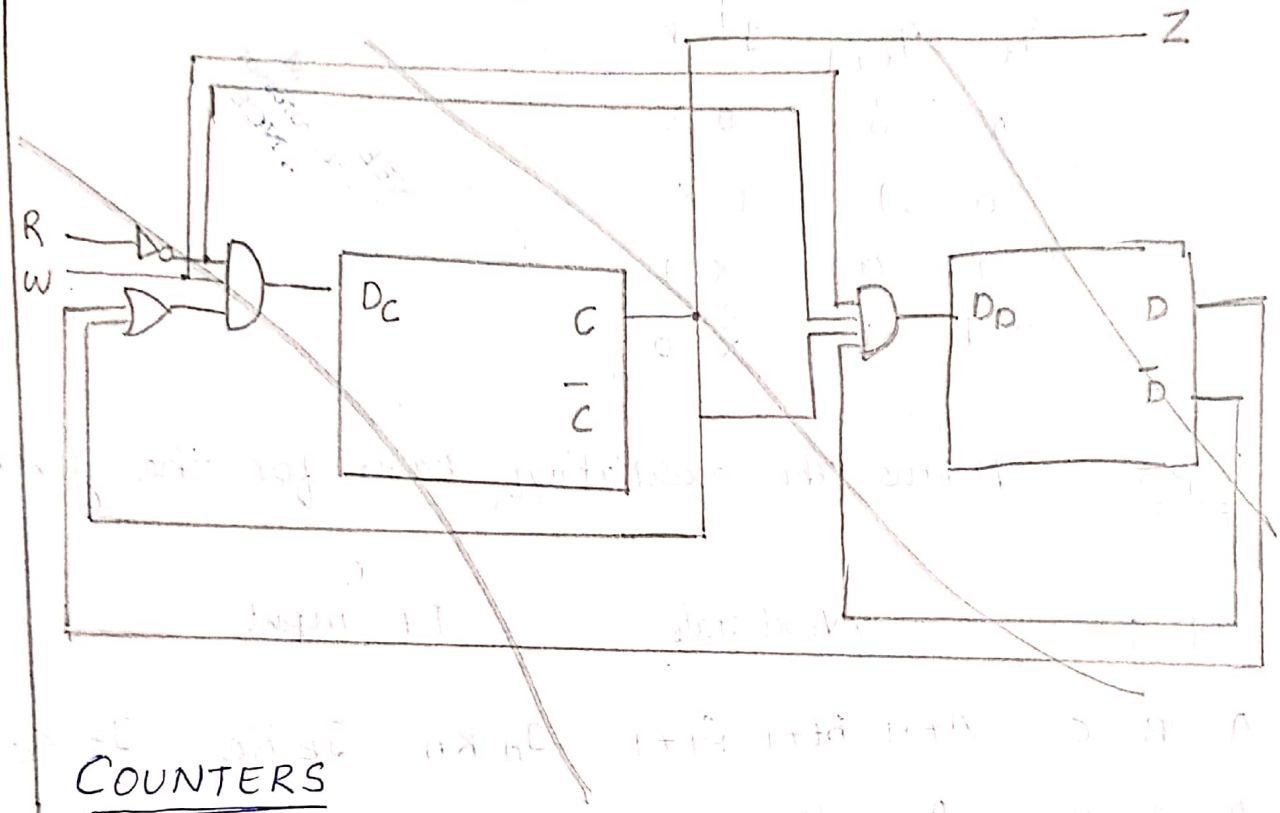
→ Application specific Instruction set processors

* Microcontrollers
* Digital signal processors

(Refer 2nd module notes)

[Very important topics]

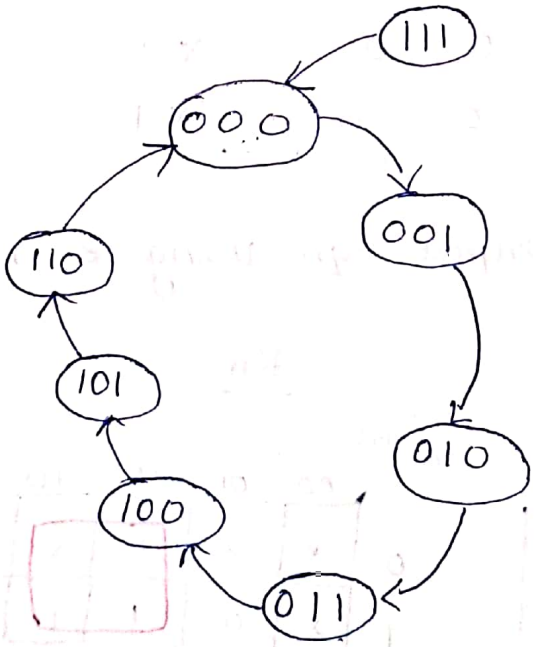
Step5 : logic diagram.



COUNTERS

1. Design a counter that counts 0 1 2 3 4 5 6 using JK flipflop.

Ans. Step 1 : State diagram.



Step2 : Excitation table for JK ff

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Step3 : Draw the excitation table for the given

PS			Nextstate			FF input					
A	B	C	A_{t+1}	B_{t+1}	C_{t+1}	J_A	K_A	J_B	K_B	J_C	K_C
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X
1	1	1	0	0	0	X	1	X	1	X	1

Step5 : flipflop input eqn using k-map.

J_A

A \ BC	00	01	11	10
0	0	0	1	0
1	X	X	X	X

$J_A = BC$.

K_A

A \ BC	00	01	11	10
0	X	X	X	X
1	0	0	1	1

$K_A = B$

J_B

A \ BC	00	01	11	10
0	0	1	X	X
1	0	1	X	X

$J_B = C$

K_B

A \ BC	00	01	11	10
0	X	X	1	0
1	X	X	1	1

$K_B = A + C$

J_C

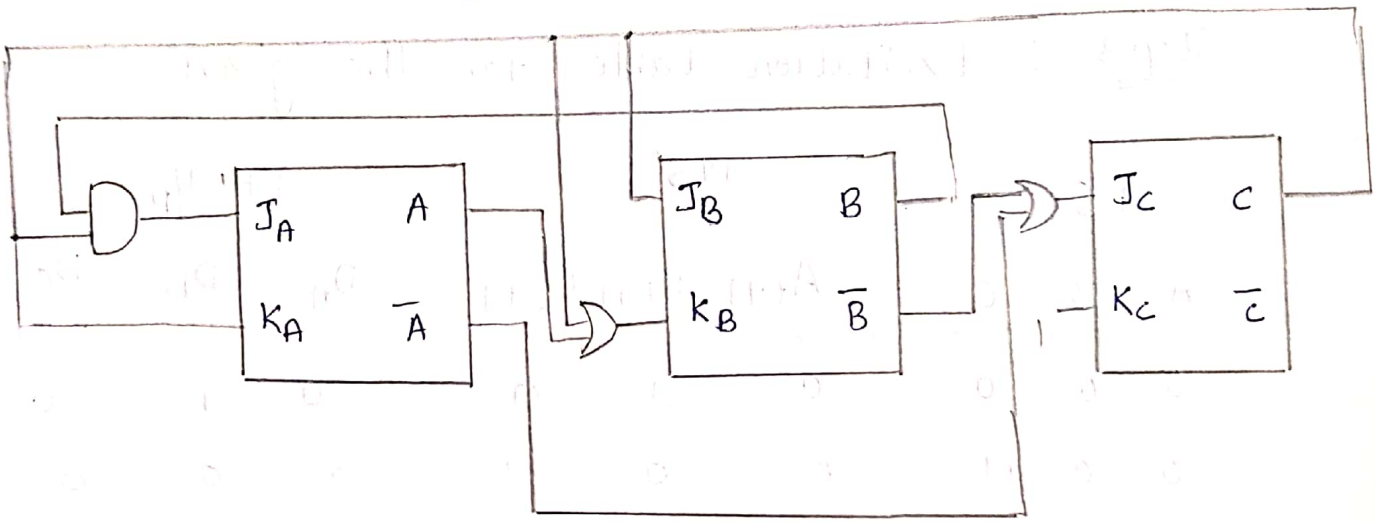
A \ BC	00	01	11	10
0	1	X	X	1
1	1	X	X	0

$J_C = \overline{A} + \overline{B}$

A \ BC	00	01	11	10
0	X	1	1	X
1	X	1	1	X

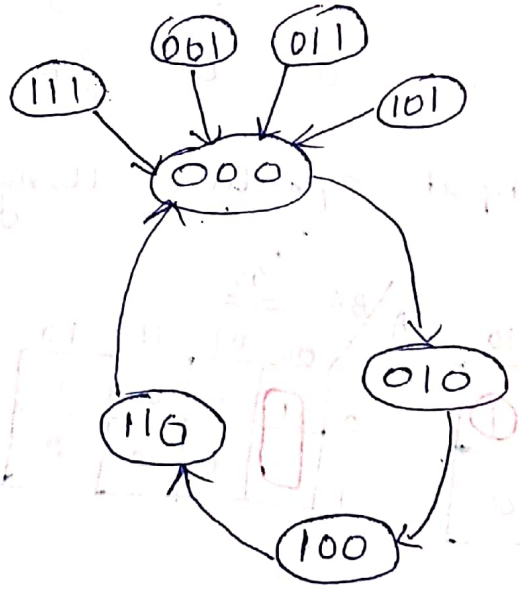
$K_C = 1$.

Step 6 : logic diagram



2. Design a counter that counts 0 2 4 6 using D-FF

Ans. **Step 1** : state diagram



0 000
1 001
2 010
3 011
4 100
5 101
6 110
7 111

Step 2: Excitation table for D-flipflop.

(1)

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Step 3: Excitation table for the given

PS			NS			FF I/P		
A	B	C	A_{t+1}	B_{t+1}	C_{t+1}	D_A	D_B	D_C
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	0	0
0	1	1	0	0	0	0	0	0
1	0	0	1	1	0	1	1	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

Step 4: ff input equation using Kmap.

D_A

A \ BC	00	01	11	10
0	0	0	0	1
1	1	0	0	0

D_B

A \ BC	00	01	11	10
0	1	0	0	0
1	1	0	0	0

D_C

A \ BC	00	01	11	10
0	0	0	0	0
1	0	0	0	0

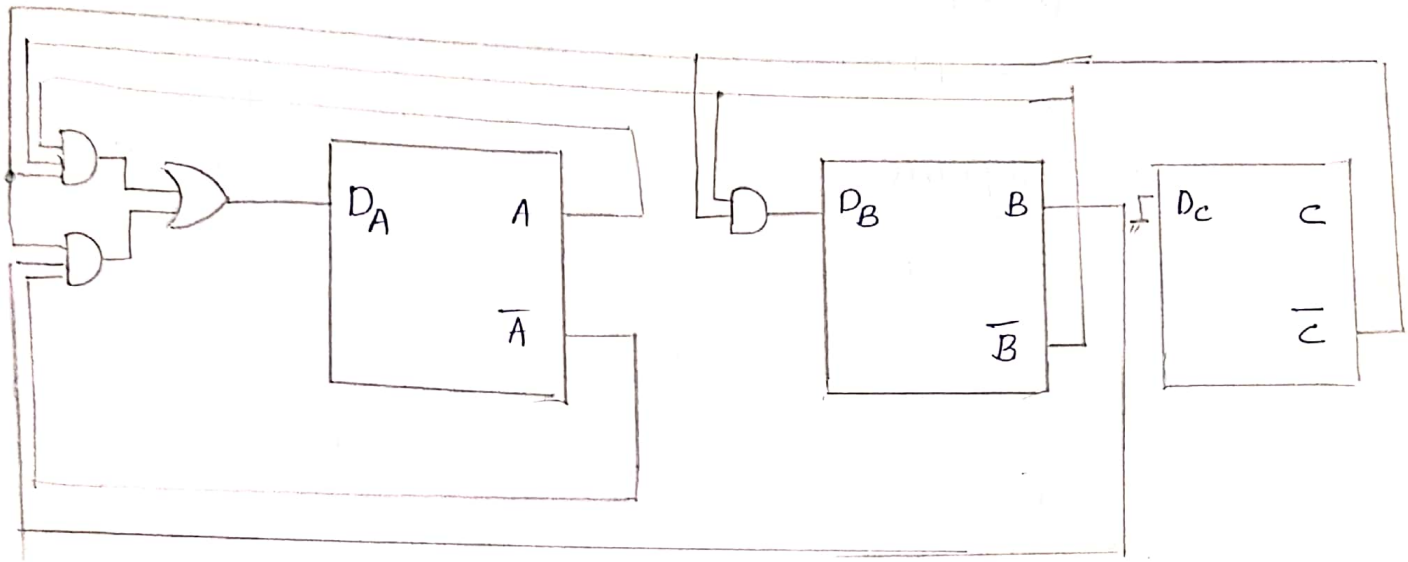
$D_A = A\bar{B}\bar{C} + \bar{A}B\bar{C}$

$= \bar{C}[A \oplus B]$

$D_B = \bar{B}\bar{C}$

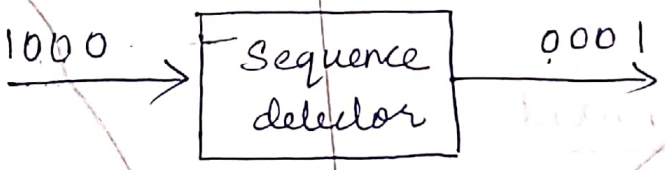
$D_3 = 0$

Step 5 : Logic diagram



SEQUENCE DETECTOR.

It is a single input circuit that will accept a sequence of bit & it will generate o/p only when the desired sequence is accepted.



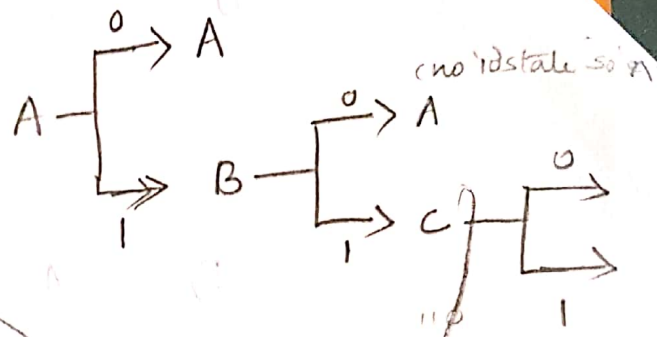
1. Design a finite state machine that detects the sequence 111 in an input with stream ie FSM should o/p a 1 when the sequence is detected and zero otherwise, using D-flipflop.

Ans. We are designing a 3 bit sequence detector, so we need 3 states.

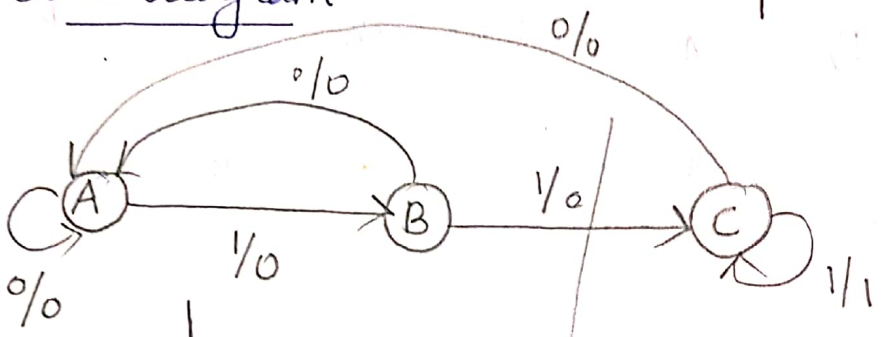
(13) Let A – X (initial state

B – 1

C – 11



State diagram



State table

PS	NS X=0	NS X=1	O/P X=0	O/P X=1
A	A	B	0	0
B	A	C	0	0
C	A	C	0	1

State Assignment

A → 00

B → 01

C → 10

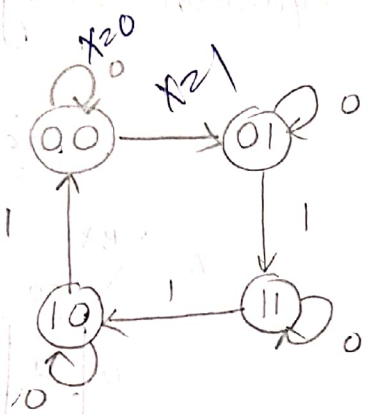
Excitation Table for D ff

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Same as Q (1). Implement using T-flip flop. (10)

(11)

Ans. **Step 1** : Draw the state diagram.



Step 2 : Reduce the state diagram (optional)

Step 3 : choose the type & no. of flip flops.

(2-T flip flops.

Write the excitation table.

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Step 4 : Draw the excitation table for the state machine

PS		Next state				ff input			
		X=0		X=1		X=0		X=1	
A	B	A_{t+1}	B_{t+1}	A_{t+1}	B_{t+1}	T_A	T_B	T_A	T_B
0	0	0	0	0	1	0	0	0	1
0	1	0	1	1	1	0	0	1	0
1	0	1	0	0	0	0	0	1	0
1	1	1	1	1	0	0	0	0	1

Step 5 :

flipflop input equation using K-map.

for T_A

A \ BX	00	01	11	10
0	0	0	1	0
1	0	1	0	0

$$T_A = \bar{A}BX + A\bar{B}X$$
$$= X(A \oplus B)$$

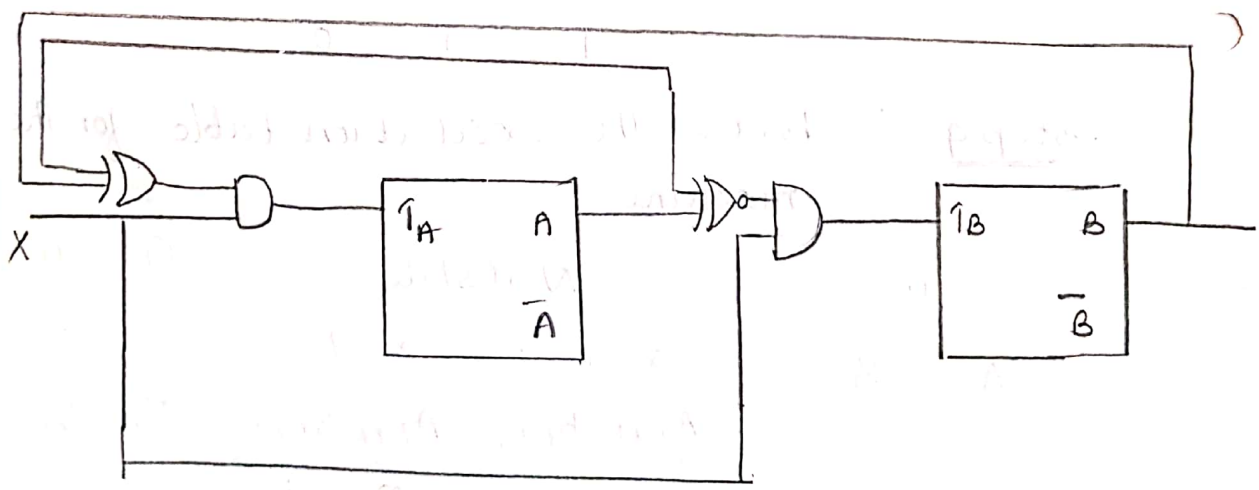
for T_B

A \ BX	00	01	11	10
0	0	1	0	0
1	0	0	1	0

$$T_B = \bar{A}\bar{B}X + ABX$$
$$= X(A \odot B)$$

Step 6

Draw the logic circuit.



NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(ACCREDITED BY NAAC)

PAMPADY, THIRUVILWAMALA, THRISSUR (DT), 680588

MR 405 EMBEDDED SYSTEM

MODULE IV

COURSE MATERIALS

SIGNATURE OF HOD

MODULE - IV

GENERAL PURPOSE PROCESSORS : SOFTWARE :

→ A general purpose processor is a programmable digital system intended to solve computation tasks in a variety of applications.

→ An embedded system designer choosing to use a general purpose processor to implement part of a system's functionality may achieve several benefits.

1. Unit cost of the processor may be very low.
2. Improve performance, size and power.
3. Flexibility will be high, since the designer can perform software rewrites in a straight forward manner.

NEETHU.M
Assistant Professor
, NCERC

BASIC ARCHITECTURE :

→ A general purpose processor, sometimes called a CPU (Central processing unit) or a microprocessor consist of a datapath and a controller, tightly linked with a memory.

Datapath :

→ The datapath consists of the circuitry for transforming data and for storing temporary data.

→ The datapath contains an arithmetic logic unit

(ALU) capable of transforming data through operations such as addition, subtraction, logical AND, logical OR, inverting and shifting.

→ The ALU also generates status signals, often stored in a status register (not shown), indicating particular data conditions.

→ Such conditions include indicating whether data is zero, or whether an addition of two items generates a carry.

→ The datapath also contains registers capable of storing temporary data.

→ Temporary data may include data brought in from memory but not yet sent through the ALU, data coming from the ALU that will be needed for later ALU operations or will be sent back to memory, and data that must be moved from one memory location to another.

→ The internal data bus is the bus over which data travels within the datapath, while the external data bus is the bus over which data is brought to & from the data memory.

→ An N-bit processor may have N-bit wide registers, an N-bit wide ALU, an N-bit wide internal bus over which data moves among datapath components & an N-bit wide external bus over which

NEETHU.M
Assistant Professor
NCERC

data is brought in and out of the datapath.

CONTROLLER :

→ The Controller Consists of circuitry for retrieving program instructions, and for moving data to, from and through the datapath according to those instructions.

→ The Controller contains a program counter (PC) that holds the address in memory of the next program instruction to fetch.

→ The controller also contains an instruction register (IR) to hold the fetched instruction.

→ Based on this instruction, the controller's control logic generates the appropriate signals to control the flow of data in the datapath.

→ Such flows may include inputting two particular register into the ALU, storing ALU results into a particular register, or moving data between memory and a register.

→ Finally, the next state logic determines the next value of the PC.

→ The PC's bit width represents the processors address size.

→ The address size is independent of the data

NEETHU.M
Assistant Professor
NCERC

word size; the address size is often larger.

→ The address size determines the number of directly accessible memory locations, refered to as the address space or memory space.

→ If the address size is M, then the address space is 2^M.

→ Thus, a processor with a 16 bit PC can directly address $2^{16} = 65,536$ memory locations.

→ We would typically refer to this address space as 64K, although if 1K = 1000, this number would represent 64,000 not the actual 65,536. Thus 1K = 1024.

→ For each instruction, the Controller typically sequences through several stages, such as fetching the instruction from memory, decoding it, fetching operands, executing the instruction in a datapath, and storing result.

→ Each stage may consist of one or more clock cycles.

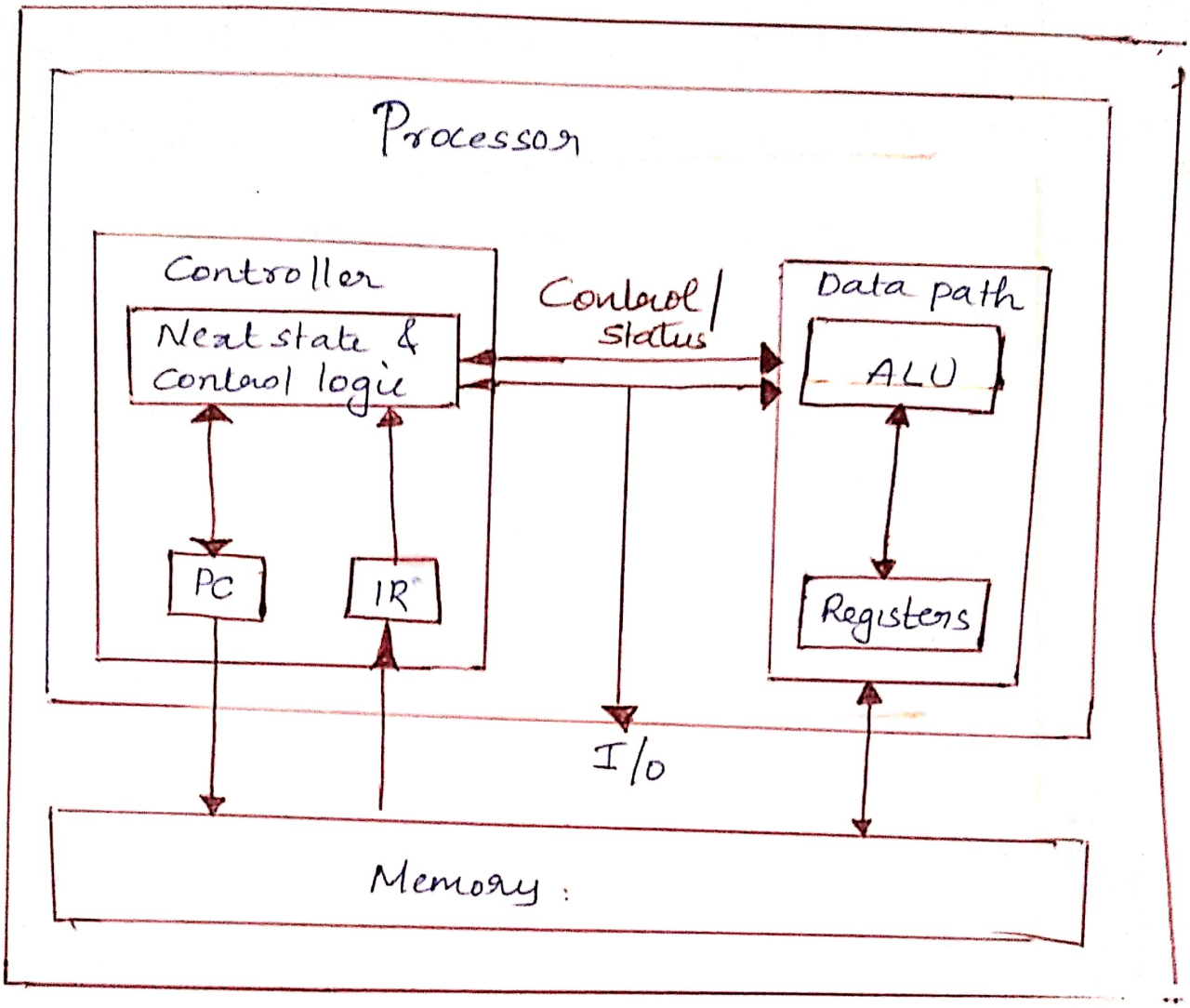
→ A clock cycle is usually the longest time required for data to travel from one register to another.

→ The path through the datapath or controller that results in this longest time is called

NEETHU.M
Assistant Professor
NCERC

critical path.

NEETHU.M
Assistant Professor
NCERC



General purpose processor basic architecture.

MEMORY :

→ Registers : short term storage requirement.

→ Memory : Serves the processors medium & long term Information storage requirements.

⇒ Stored Information can be classified as either Program or data.

→ Program Information consists of sequence of Instructions that cause the processor to carry

Out the desired System functionality.

→ Data Information represents the value being input, Output and transformed by the program.

→ **Princeton architecture**:

→ Data & program words share the same memory space.

→ The princeton architecture may result in a Simpler hardware Connection to memory, since only one Connection is necessary.

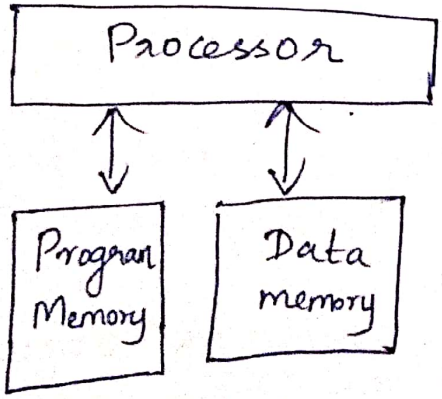
→ **Harvard architecture**

→ The program memory space is distinct from the data memory space.

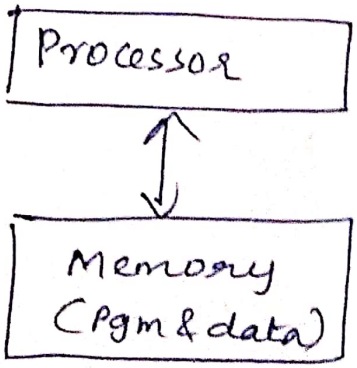
→ A harvard architecture, while requiring two Connections, can perform Instruction ~~set~~ and data fetches simultaneously, so may result in improved performance.

Eg: Intel 8051.

Two memory architectures (a) Harvard (b) princeton



(a)



(b)

NEETHU.M
Assistant Professor
, NCERC

→ Memory may be read only memory (ROM) & writable memory (RAM).

→ ROM is usually much more compact than RAM.

→ An embedded System often uses ROM for program memory, since unlike in desktop systems an embedded System's program does not change.

→ Constant data may be stored in ROM, but other data ~~are~~ required RAM.

→ Memory may be on-chip or off chip.

→ On-chip memory resides on the same IC as the processor, while off-chip memory resides on a separate IC.

→ The processor usually access on-chip memory must faster than off chip memory, perphaps in just one cycle, but finite IC capacity of course emplies only a limited amount of on-chip memory.

→ To reduce the time needed to access (read or write) memory, a local copy of a portion of memory may be kept in a small but especially fast memory called cache.

→ Cache memory often resides on-chip, and often uses fast but expensive static RAM technology rather than slower but cheaper dynamic RAM.

NEETHU.M
Assistant Professor
NCERC

→ Cache memory is based on the principle that if at a particular time a processor accesses a particular memory location, then the processor will likely access that location and immediate neighbors of the location in the near future.

→ Thus, when we first access a location in memory, we copy that location and some number of its neighbors (called a block) into cache, and then access the copy of the location in cache.

→ When we access another location, we first check a cache table to see if a copy of the location resides in cache.

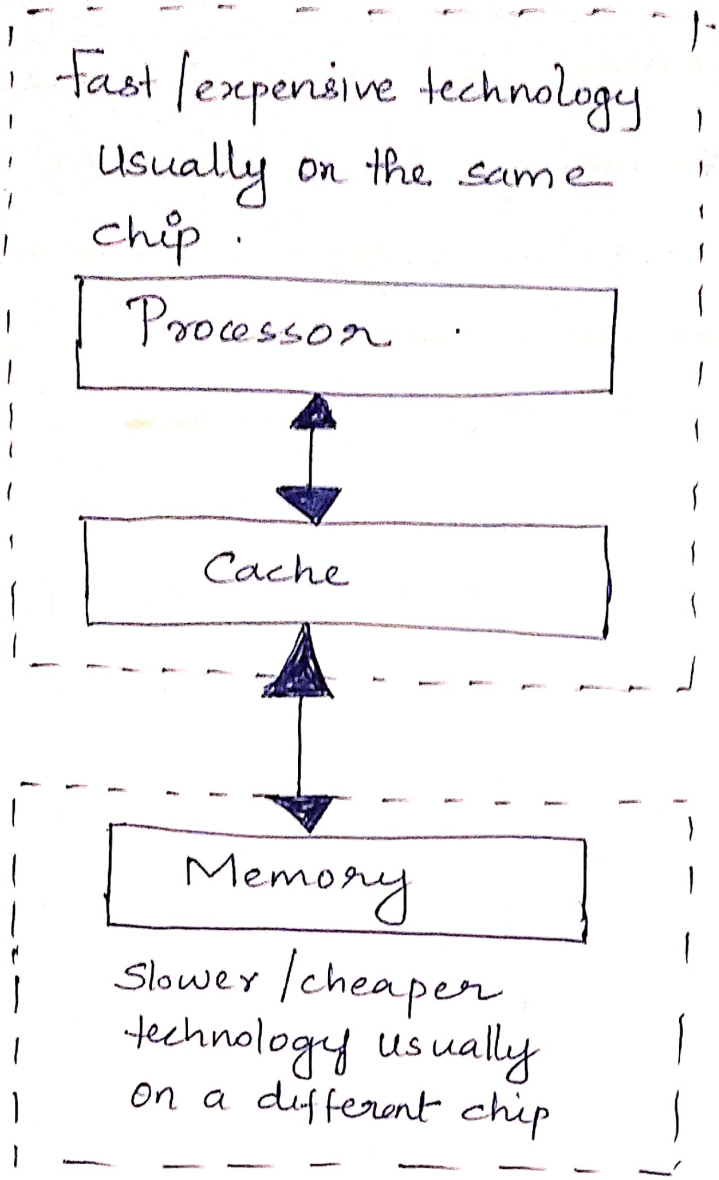
→ If the copy does reside in cache, we have a cache hit, and we can read/write that location very quickly.

→ If the copy does not reside in cache, we have a cache miss, so we must copy the locations block into cache, which takes a lot of time.

→ Thus, for a cache to be effective in improving performance, the ratio of hits to misses must be very high, requiring intelligent caching schemes.

→ Caches are used for both program memory (often called Instruction cache, or I-cache) as well as data memory (often called D-cache)

NEETHU.M
Assistant Professor
NCERC



Cache memory.

INSTRUCTION EXECUTION :

① Fetch Instruction :

The task of reading the next instruction from memory into the Instruction register.

② Decode Instruction :

The task of determining what operation the instruction in the instruction register represents (Eg: add, move etc).

③ Fetch operands :

The task of moving the Instruction's operand data into appropriate registers.

④ Execution operation :

The task of feeding the appropriate registers through the ALU and back into an appropriate registers.

⑤ Store results :

The task of writing a register into memory.

If each stage takes ~~place~~ one clock cycle, then we can see that a single instruction may take several cycles to complete.

NEETHU.M
Assistant Professor
NCERC

PIPELINING

→ pipelining is a common way to Increase the Instruction throughput of a microprocessor.

→ Pipelining is a technique where multiple Instructions are overlapped during execution.

→ pipeline is divided into stages and these stages are connected with one another to form a pipe like Structure.

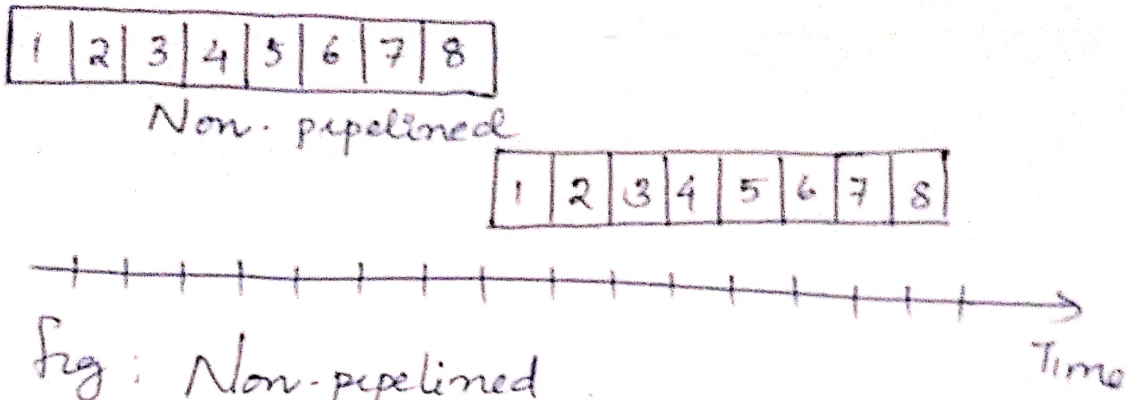


fig: Non-pipelined

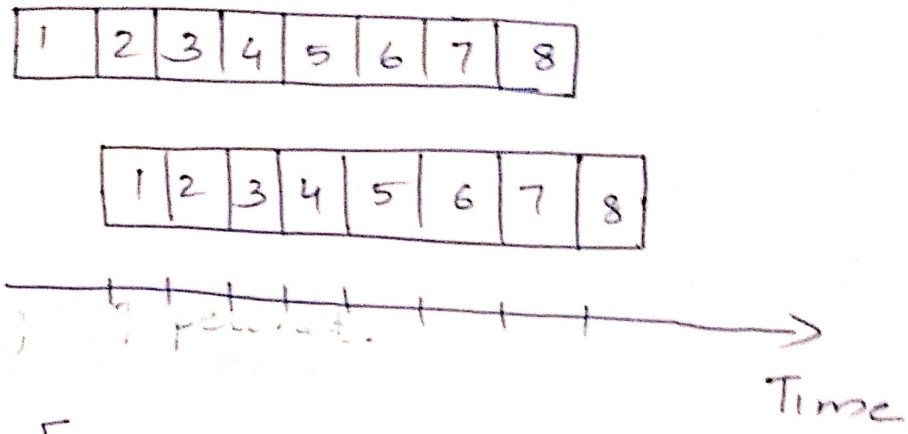
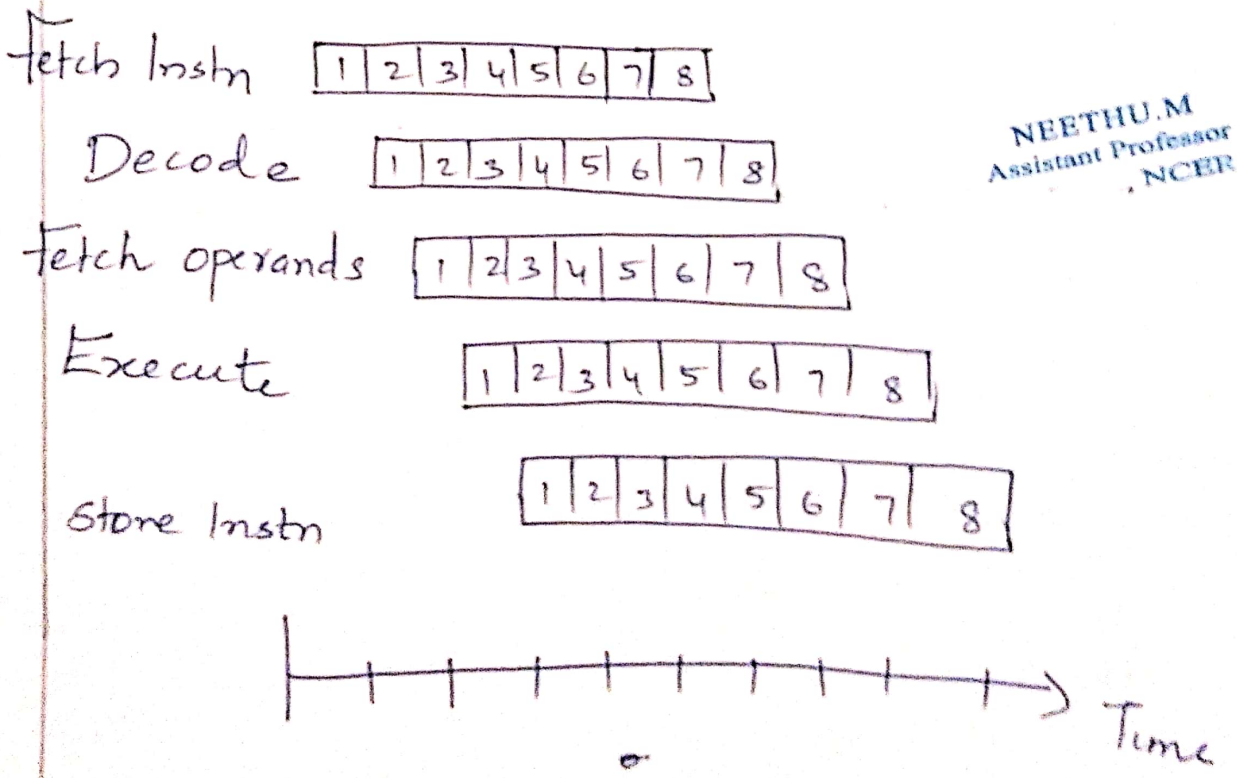


fig: pipelined.



Pipelined

INSTRUCTION SET :

→ An Instruction typically has two parts
- An opcode field
- Operand field.

→ opcode specifies the operation to take place during the Instruction.

→ We can classify Instructions into 3 categories.

① **Data transfer Instructions** :

- Move data between memory and registers,
- between Input/Output channels and registers
- " registers themselves.

② **Arithmetic/logical Instructions**

- Configure the ALU to carry out a particular functions, channel data from registers through the ALU and channel data from the ALU back to a particular register.

③ **Branch Instruction**

It determine the address of the next program Instruction.

Branch Instruction

- Unconditional Jumps
- Conditional Jumps / Procedure call & return Instn

NEETHU.M
Assistant Professor
NCERC

→ Unconditional Jumps always determine the address of the next Instruction, while Conditional Jumps do so only if some Condition evaluates to true, such as particular register Containing zero.

→ A call Instruction, in addition to Indicating the address of next Instruction, Saves the address of the Current Instruction so that a Subsequent return Instruction can jump back to the Instruction Immediately following the most recent envoked Call Instruction.

→ An operand field specifies the location of the actual data that take part in an operation.

→ Source operands serve as input to the operation, while a destination operand stores the Output.

→ The no. of operands per instruction varies among Processors. Even for a given processor, the no. of Operands per instruction may vary depending on the Instruction type.

→ In Immediate addressing, the operand field Contains the data itself.

→ In register addressing the operand field Contains the address of a datapath register on which the data resides.

→ In register Indirect addressing, the operand field Contains the address of a register, which

in turn contains the address of a memory location in which the data resides.

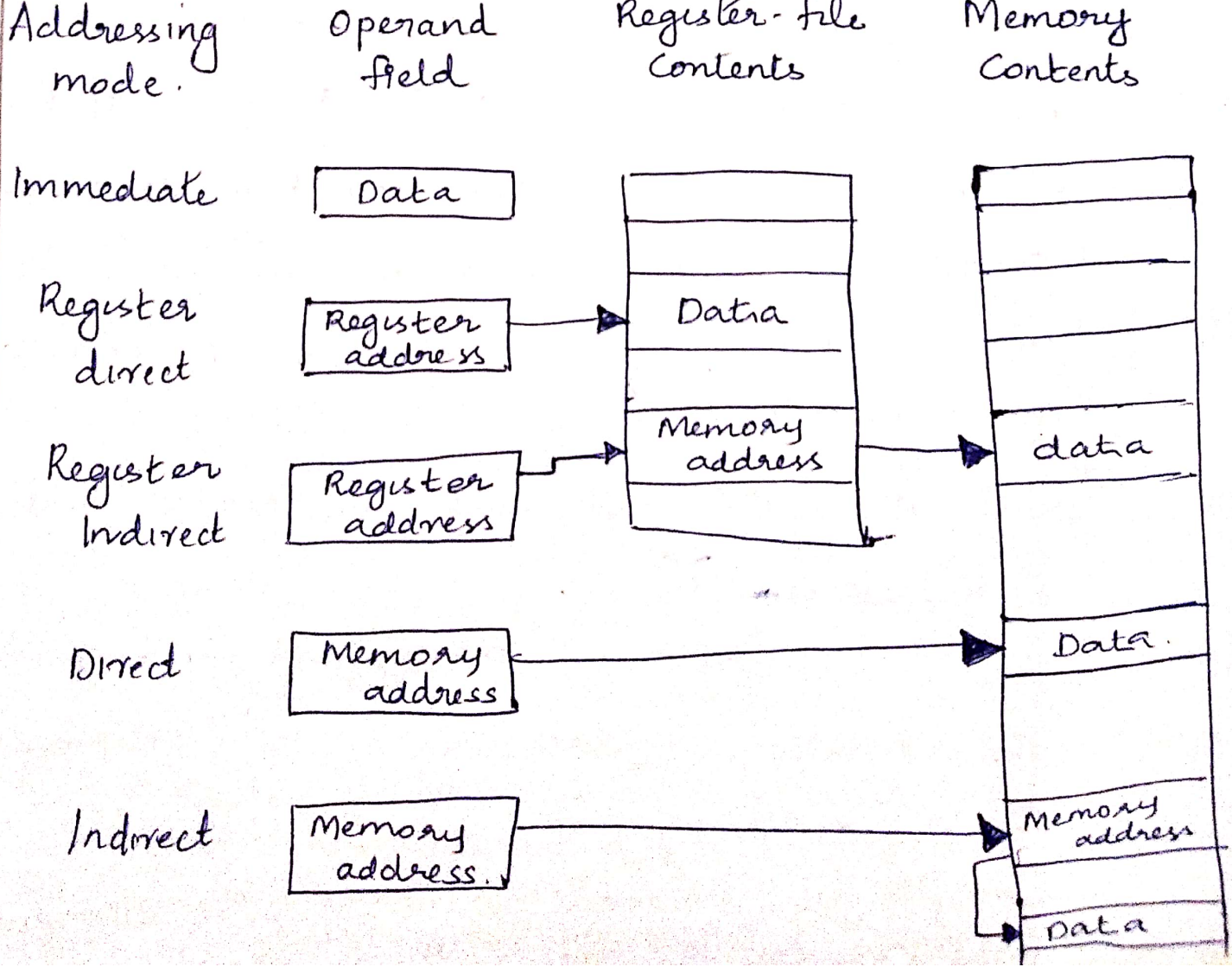
→ In direct addressing, the operand field contains the address of a memory location in which the data resides.

→ In Indirect addressing, the operand field contains the address of a memory location, which in turn contains the address of a memory location in which the data resides.

→ ~~Direct addressing implements regular variables~~

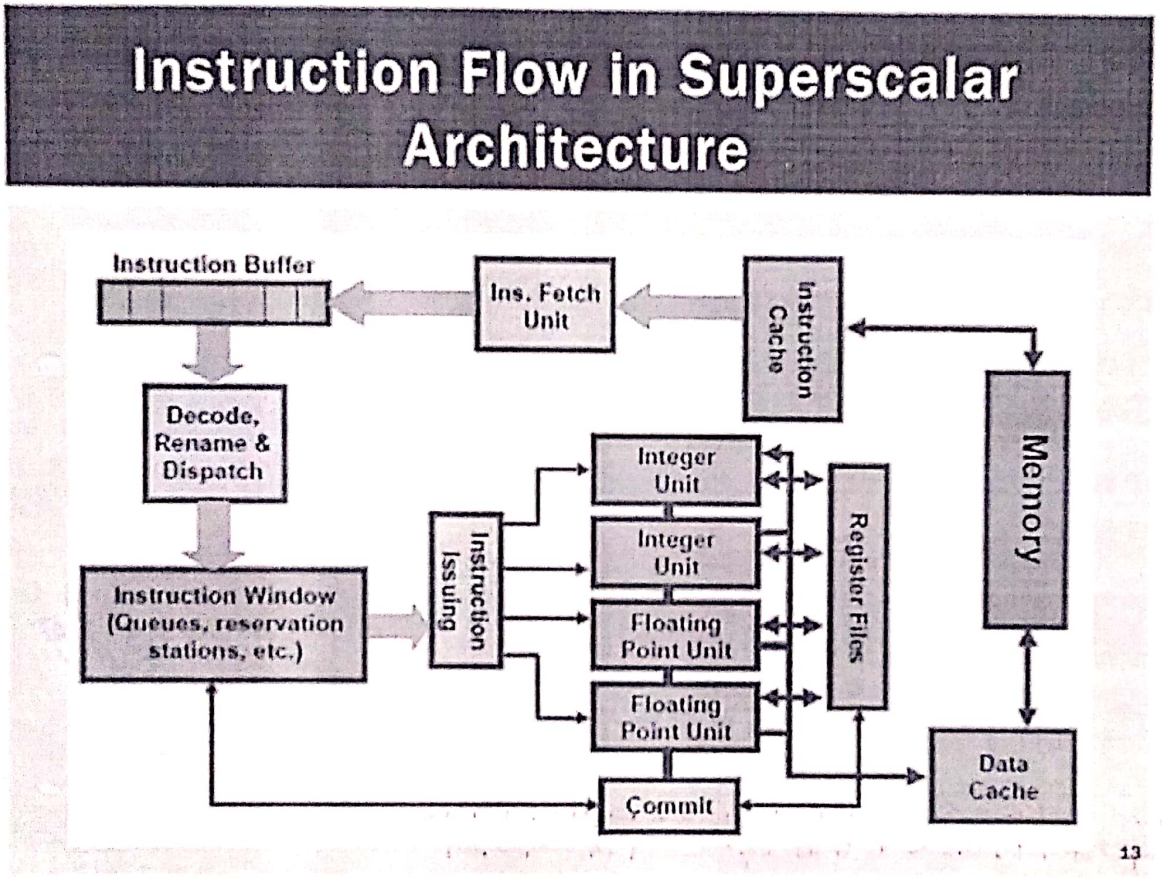
NEETHU.M
Assistant Professor
NCERC

Addressing modes



SUPERSCALAR ARCHITECTURE

Superscalar architecture is a method of parallel computing used in many processors. In a **superscalar** computer, the central processing unit (CPU) manages multiple instruction pipelines to execute several instructions concurrently during a clock cycle.



A superscalarprocessor is a CPU that implements a form of parallelism called instruction-level parallelism within a single processor. In contrast to a scalar processor that can execute at most one single instruction per clock cycle, a superscalar processor can execute more than one instruction during a clock cycle by simultaneously dispatching multiple instructions to different execution units on the processor. It therefore allows for more throughput (the number of instructions that can be executed in a unit of time) than would otherwise be possible at a given clock rate. Each execution unit is not a separate processor (or a core if the processor is a multi-core processor), but an execution resource within a single CPU such as an arithmetic logic unit.

A **superscalar** processor is a microprocessor design for exploiting multiple instructions in one clock cycle, thus establishing an instruction-level parallelism in processors. A superscalar is a super-pipelined model where only the independent instructions are executed sequentially, without any waiting state. The superscalar architecture was first used in RISC processors. This architecture can also be called a 'Second Generation RISC'.

The processor or compiler in a superscalar architecture determines if an instruction is dependent on the output of other sequential instructions, or whether it can be executed independently. The data dependency between instructions is verified dynamically by the CPU hardware at run time. The scheduling of instructions in a superscalar architecture is done dynamically, at run time, by the processor. The superscalar architectures have mechanisms for fetching multiple instructions, determining dependencies between instructions and executing instructions in order.

Pipelining in Superscalar

To completely make use of a superscalar processor having *n* processing units or pipelines, *n* instructions can be executed in parallel. This condition cannot be possible in all the clock cycles, and in such cases, a few pipelines could stall in a waiting state. In the case of superscalar processors, single operation latency requires just one clock cycle. The term latency denotes the time delay from the time of input to the production of desired output. 'Operational latency' in a superscalar system denotes the delay caused by the slowest operation among all the parallel operations running in the execution units of a superscalar processor.

Limitations of Superscalar Architecture

- Limited instruction-level parallelism.
- Dependency checking is an overhead.
- The dependency checking cost increases with an increase in the number of instructions executed in parallel.
- Pipeline stalls are common when an executing instruction is dependent on the result of an unprocessed instruction

VLIW ARCHITECTURE

Very long instruction word (VLIW) describes a computer processing architecture in which a language compiler or pre-processor breaks program instruction down into basic operations that can be performed by the processor in parallel (that is, at the same time)

- **Very long instruction word** or **VLIW** refers to a processor architecture designed to take advantage of instruction level parallelism
  - Instruction of a VLIW processor consists of multiple independent operations grouped together.
  - There are Multiple Independent Functional Units in VLIW processor architecture.
  - Each operation in the instruction is aligned to a functional unit.
  - All functional units share the use of a common large register file.
- This type of processor architecture is intended to allow higher performance without the inherent complexity of some other approaches

Advantages of VLIW

- Dependencies are determined by compiler and used to schedule according to function unit latencies
- Function units are assigned by compiler and correspond to the position within the instruction packet.
- Reduces hardware complexity.
  - Tasks such as decoding, data dependency detection, instruction issues etc. becoming simple.
  - Ensures potentially higher Clock Rate.
  - Ensures Low power consumption

VLIW vs. Superscalar Architecture

- **Instruction scheduling**
  - *Superscalar:*
    - Done dynamically at run-time by the hardware.
    - Data dependency is checked and resolved in hardware.
    - Need a look ahead hardware window for instruction fetch.
  - *VLIW:*
    - Done statically at compile time by compiler.
    - Data dependency is checked by compiler.
    - In case of un-filled opcodes in a VLIW, memory space and instruction bandwidth are wasted.

Program & data memory space :

→ The embedded Systems programmer must be aware of the size of the available memory for program and for data.

→ For example, a particular processor may have a 64k program space and a 64K data space.

→ The programmer must not exceed these limits.

→ In addition, the programmer will probably want to be aware of on-chip program & data memory capacity, taking care to fit the necessary program & data in on-chip memory if possible.

Registers :

→ The assembly language programmer must know how many registers are available for general purpose data storage

→ For example, a base register may exist, which permits the programmer to use a data transfer instruction where the processor adds an operand field to the base register to obtain an actual memory address.

→ other special function registers must be known by both the assembly language & the structured language programmer. Such registers may be used for configuring built-in timers, counters

NEETHU.M
Assistant Professor
, NCERC

and Serial Communication devices, or for writing & reading external pins.

Input / Output :

-> The programmer should be aware of the processor's input and output (I/O) facilities, with which the processor communicates with other devices.

-> One common I/O facility is parallel I/O, in which the programmer can read or write a port (a collection of external pins) by reading or writing a special function register.

-> Another common I/O facility is a system bus, consisting of address and data ports that are automatically activated by certain addresses or types of instructions.

Interrupts :

-> An Interrupt causes the processor to suspend execution of main program & Insted jump to an Interrupt Service Routine (ISR) that fulfills a special / short term processing need.

-> The processor stores the current PC and sets it to the address of the ISR. After the ISR completes

NEETHU.M
Assistant Professor
NCERC

OPERATING SYSTEMS :

An operating system is a layer of software that provides low-level services to the *application layer*, a set of one or more programs executing on the CPU consuming and producing input and output data. The task of managing the application layer involves the loading and executing of programs, sharing and allocating system resources to these programs, and protecting these allocated resources from corruption by non-owner programs. One of the most important resource of a system is the central processing unit (CPU), which is typically shared among a number of executing programs. The operating system, thus, is responsible for deciding what program is to run next on the CPU and for how long. This is called process/task scheduling and is determined by the operating system's preemption policy. Another very important resource is memory, including disk storage, which is also shared among the applications running on the CPU.

In addition to implementing an environment for management of high-level application programs, the operating system provides the software required for servicing various hardware-interrupts, and provides device drivers for driving the peripheral devices present in the system. Typically, on startup, an operating system initializes all peripheral devices, such as disk controllers, timers and input/output devices and installs

Figure 2.9: System call invocation.

```
DB file_name "out.txt"      -- store file name

    MOV R0, 1324            -- system call "open" id
    MOV R1, file_name       -- address of file-name
    INT 34                  -- cause a system call
    JZ  R0, L1              -- if zero -> error

    . . . read the file
    JMP L2                  -- bypass error cond.
L1:
    . . . handle the error

L2:
```

hardware interrupt (interrupts generated by the hardware) service routines (ISR) to handle various signals generated by these devices[2]. Then, it installs *software interrupts* (interrupts generated by the software) to process *system calls* (calls made by high-level applications to request operating system services) as described next.

A system call is a mechanism for an application to invoke the operating system. This is analogous to a procedure or function call, as in high-level programming languages. When a program requires some service from the operating system, it generates a predefined software interrupt that is serviced by the operating system. Parameters specific to the requested services are typically passed from (to) the application program to (from) the operating system through CPU registers. Figure 2.9 illustrates how the file "open" system call may be invoked, in assembly, by a program. Languages like C and Pascal provide wrapper functions around the system-calls to provide a high-level mechanism for performing system calls.

In summary, the operating system abstracts away the details of the underlying hardware and provides the application layer an interface to the hardware through the system call mechanism.

2.4.7 Development environment

Several software and hardware tools commonly support the programming of general-purpose processors. First, we must distinguish between two processors we deal with when developing an embedded system. One processor is the *development processor*, on which we write and debug our program. This processor is part of our desktop computer. The other processor is the *target processor*, to which we will send our program and which will form part of our embedded system's implementation. For example, we may develop our system on a Pentium processor, but use a Motorola 68HC11 as our target processor. Of course, sometimes the two processors happen to be the same, but this is mostly a coincidence.

Assemblers translate assembly instructions to binary machine instructions. In addition to just replacing opcode and operand mnemonics by binary equivalents, an assembler may also translate symbolic labels into actual addresses. For example, a programmer may add a symbolic label *END* to an instruction *A*, and may reference *END* in a branch instruction. The assembler determines the actual binary address of *A*, and replaces references to *END* by this address. The mapping of assembly instructions to machine instructions is one-to-one. A *linker* allows a programmer to create a program in separately-assembled files; it combines the machine instructions of each into a single program, perhaps incorporating instructions from standard library routines.

Compilers translate structured programs into machine (or assembly) programs. Structured programming languages possess high-level constructs that greatly simplify programming, such as loop constructs, so each high-level construct may translate to several or tens of machine instructions. Compiler technology has advanced tremendously over the past decades, applying numerous program optimizations, often yielding very size and performance efficient code. A *cross-compiler* executes on one processor (our development processor), but generates code for a different processor (our target processor). Cross-compilers are extremely common in embedded system development.

Debuggers help programmers evaluate and correct their programs. They run on the development processor and support stepwise program execution, executing one instruction and then stopping, proceeding to the next instruction when instructed by the user. They permit execution up to user-specified breakpoints, which are instructions that when encountered cause the program to stop executing. Whenever the program stops, the user can examine values of various memory and register locations. A *source-level* debugger enables step-by-step execution in the source program language, whether assembly language or a structured language. A good debugging capability is crucial, as today's programs can be quite complex and hard to write correctly.

Device programmers download a binary machine program from the development processor's memory into the target processor's memory.

Emulators support debugging of the program while it executes on the target processor. An emulator typically consists of a debugger coupled with a board connected to the desktop processor via a cable. The board consists of the target processor plus some support circuitry (often another processor). The board may have another cable with a device having the same pin configuration as the target processor, allowing one to plug this device into a real embedded system. Such an *in-circuit emulator* enables one to control and monitor the program's execution in the actual embedded system circuit. In-circuit emulators are available for nearly any processor intended for embedded use, though they can be quite expensive if they are to run at real speeds.

The availability of low-cost or high-quality development environments for a processor often heavily influences the choice of a processor.

Standard Single purpose processors :

→ A single-purpose processor is a digital system intended to solve a specific computation task.

→ The processor may be a standard one, intended for use in a wide variety of applications in which the same task must be performed.

→ An embedded system designer choosing to use a standard single purpose, processor to implement part of a system's functionality may achieve several benefits.

① Performance may be fast

② Size may be small (A single purpose processor does not require a program memory). Also, since it does not need to support a large instruction set, it may have simpler datapaths & controller.

③ Low unit cost

→ We often refer to standard single purpose processors as peripherals, because they usually exist on the periphery of the CPU.

Some of the examples are

① Timers & Counters

→ A timer is a device that generates a signal pulse at specified time intervals.

→ A time interval is a "real time" measure of time, such as 3 milliseconds. These devices are

NEETHU.M
Assistant Professor
NCERC

extremely useful in systems in which a particular action, such as sampling an input signal or generating an output signal, must be performed every x time units.

→ Internally, a simple timer may consist of a register, counter and an extremely simple controller.

→ The register holds a count value representing the no. of clock cycles that equals the desired real time value. This no. can be computed using the simple formula

$$\text{No. of clock cycles} = \frac{\text{Desired real time value}}{\text{clock cycle}}.$$

→ for example: To obtain a duration of 3 milliseconds from a clock cycle of 10 nanoseconds (100 MHz), we must count $(3\times10^{6}\,s / 10\times10^{-9}\,s/cycle)$ = 300 cycles.

→ The counter is initially loaded with the count value & then counts down on every clock cycle until 0 is reached, at which point an output signal is generated, the count value is reloaded and the process repeats itself.

→ To use a timer, we must configure it (write to its registers) and respond to its output signal. When we use a timer signal by assigning "to an

interrupt, so we include the desired action in an interrupt service routine. A counter is nearly identical to a timer, except that instead of counting clock cycles (pulses on the clock signal), a counter counts pulses on some other input signal.

2 A/D converter

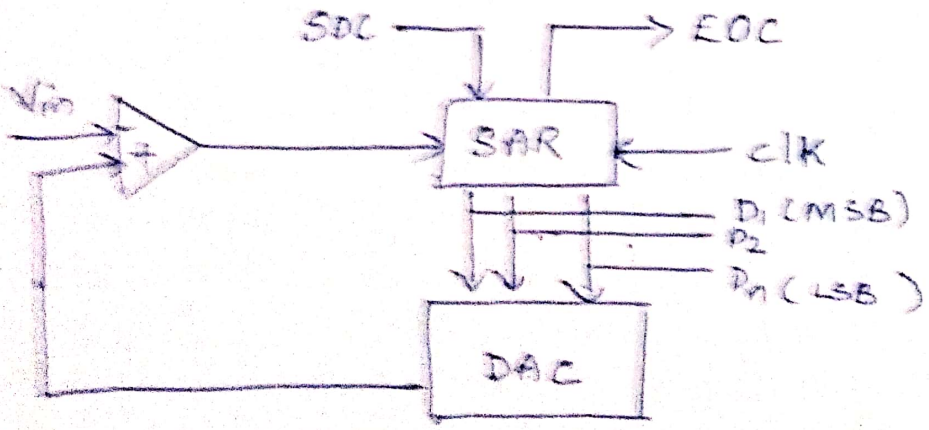
→ An analog to digital converter converts an analog signal to a digital signal and digital to analog does the opposite.

→ Both conversions are necessary because, while embedded systems deal with digital values, an embedded system surroundings typically involve many analog signals.

Successive approximation type ADC

→ Most widely used type ADC

→ It has much shorter conversion time than the other types



Functional diagram of successive approximation type ADC

→ This type of converter uses a SAR (successive approximation register) where contents is approximated bit by bit depending on the output of comparator. ie if the output of comparator is 0, then set the next bit without changing the current bit. if the output of the comparator is 1, then reset the current bit & set the next bit. This process will continue until all the successive bits of SAR is approximated

When $V_{in} > V_d$ ⇒ Set the next bit

$V_{in} < V_d$ ⇒ Already set bit is going to reset & set the next bit

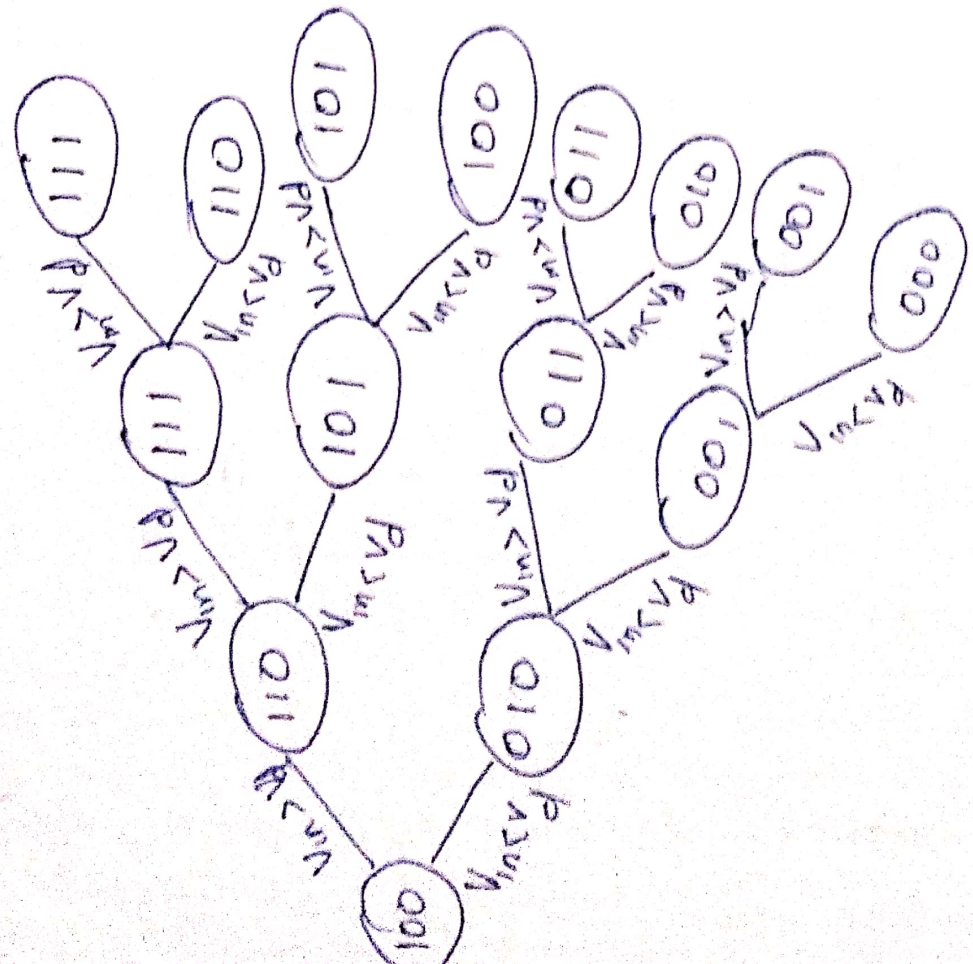


Fig: A 3 bit ADC

MR 405- EMBEDDED SYSTEMS

Module 5

- Common memory devices – Memory selection – Memory map – Internal devices & I/O devices map – Direct memory access -.Types of I/O devices – Serial devices – Parallel port devices – Sophisticated features – Timer and Counting devices – Advanced serial bus & I/O – High speed Buses – Common types – Advanced Buses.

- ROM
  - Masked ROM
  - EPROM,E2PROM,OTP ROM
  - FLASH
- RAM
  - SRAM
  - DRAM

- RAM
  - EDO RAM – Extended Data Output RAM
  - SD RAM -
  - RD RAM

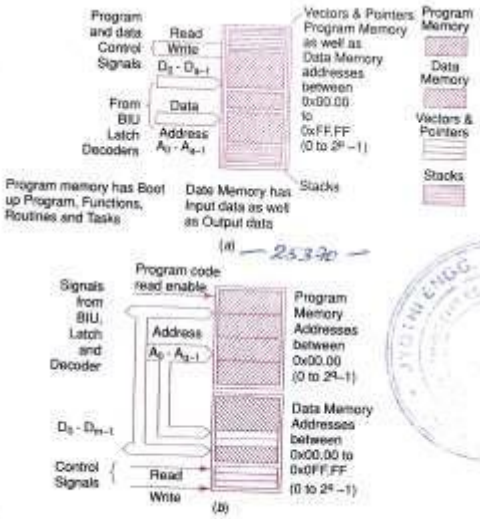
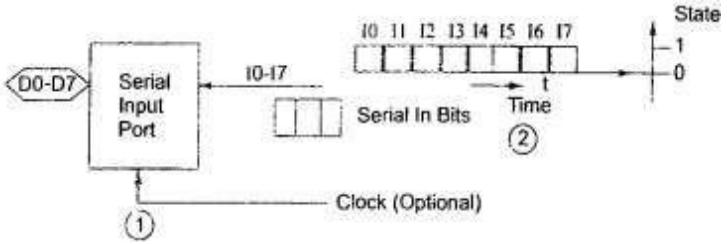


Fig. 2.23 (a) Processor and memory organization in Princeton architecture
(b) Processor and memory organization in Harvard architecture

- Synchronous Serial Input
- Synchronous Serial Output
- Asynchronous Serial UART Input
- Asynchronous Serial UART Output
- Parallel Port One bit Input
- Parallel Port One bit Output
- Parallel Port Input
- Parallel Port Output


D0-D7
Serial
Input
Port
I0-I7
I0 I1 I2 I3 I4 I5 I6 I7
State
1
0
t
Time
Serial In Bits
2
Clock (Optional)
1

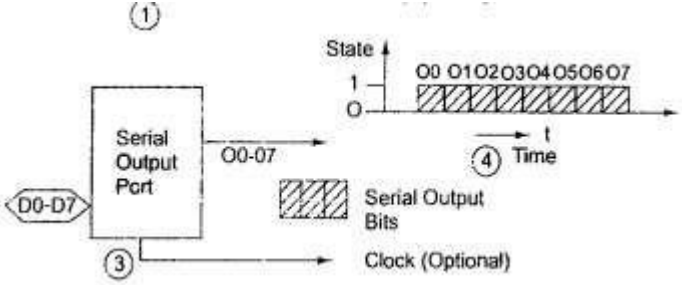
- Synchronization means separation by a constant interval or phase difference.
- If clock period equals T ,then each byte at the port is received at input in period 8 T.
- The bytes are received at constant rates. Each byte at the input port separates by 8 T and data transfer rate for the serial line bits is 1/T bps [1 bps = 1-bit per second].
- The sender. along with the serial bits, also sends the clock pulses SCLK (serial clock) to the receiver port pin.

- The serial data input and clock pulse-input are on same input line when the clock pulses either encode or modulate serial data input bits suitably.
- The receiver detects clock pulses and receives data bits after decoding or demodulating.
- When a separate SCLK input is sent, the receiver detects at the middle, positive or negative edge of the clock pulses that indicate whether data-input is I or 0 and saves the bits in 8-bit shift register.

MOSI/ MISO

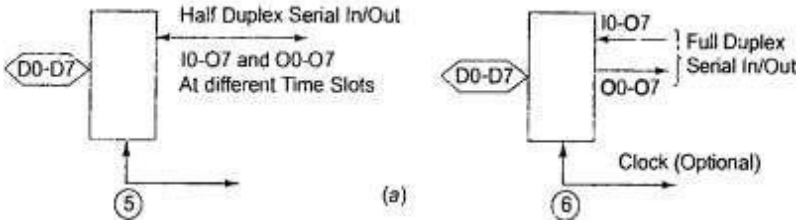
- Synchronous serial input is also called master output slave input (MOS1) when the SCLK is sent from the sender to the receiver and slave is forced to synchronize sent inputs from the master as per the master clock inputs.
- Synchronous serial input is also called master input slave output (MISO) when the SCLK is sent to the sender (slave) from the receiver (master) and the stave is forced to synchronize sending the inputs to master as per the master clock's outputs.

- Synchronous serial input is used for interprocessor transfers, audio inputs and streaming data inputs.


1
State
1
O
O0 O1O2O3O4O5O6O7
t
Time
4
Serial
Output
Port
O0-07
D0-D7
Serial Output
Bits
3
Clock (Optional)

- Each byte is in synchronization with a clock.
- The bytes are sent at constant rates
- If the clock period equals T, then the data transfer rate is 1/T bps.
- The sender sends either the clock pulses at SCLK pin or the serial data output and clock pulse-input through same output line when the clock pulses either suitably modulate or encode the serial output bits.

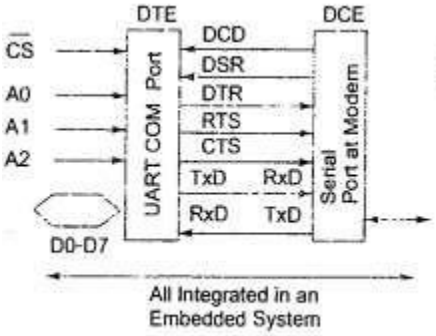
- Each bit in each byte synchronizes with the clock input and output.
- The bytes are sent or received at constant rates .
- The I/O are on same IO line when the clock pulses suitably modulate or encode the serial input and output, respectively.
- If the clock period equals T, then the data transfer rate is 1/ T bps.
- Synchronous serial input/outputs are also called master input slave output (MISO) and muster output slave input (MOSI), respectively.


Half Duplex Serial In/Out
D0-D7
I0-O7 and O0-O7
At different Time Slots
5
(a)
I0-O7
Full Duplex
Serial In/Out
D0-D7
O0-O7
Clock (Optional)
6

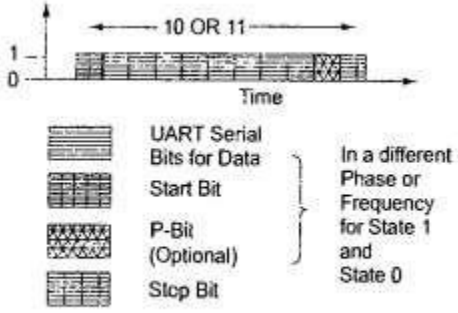
- They are used for interprocessor transfers and streaming data.
- The bits are read from or written on magnetic media such as a hard disk or on optical media such as a CD by using devices with serial synchronous IO ports.

Asynchronous Serial Input

- Each RxD bit is received in each byte at fixed intervals but each received byte is not in synchronization.
- The bytes can separate by variable intervals or phase differences.
- When a sender shifts after every clock period T, then a byte at the port is received at input in period 10T or 11T.
- The time of 2 T is due to use of additional bits at the start and end of each byte. An addition time of IT is taken when a P-bit is sent before the stop hit.


DTE
DCE
CS
A0
A1
A2
D0-D7
UART COM Port
Serial Port at Modem
DCD
DSR
DTR
RTS
CTS
TxD
RxD
RxD
TxD
All Integrated in an Embedded System

(b)


10 OR 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

(c)

- The bit transfer rate (for the serial line bits) is (I/ T) baud per second but different bytes may be received at varying intervals.
- The word 'Baud' is taken from a German word for raindrop.
- Bytes pour from the sender like raindrops at irregular intervals.
- The sender does not send the clock pulses along with the bits.

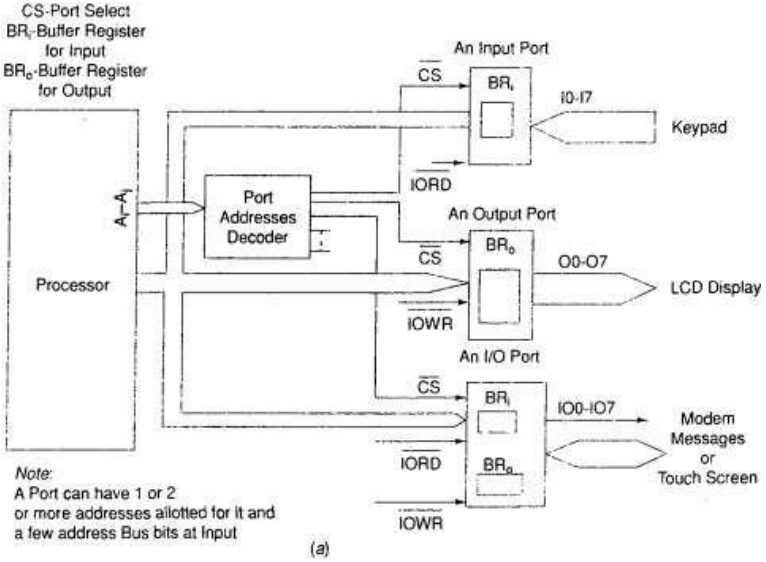
- The receiver detects n bits at the intervals of T from the middle of the first indicating bit.
- n = 0,1 …0 or I I, finds out whether the data-input is 1 or 0 and saves the bits in an 8-bit shift register.
- The processing element at the port (peripheral) saves the byte at a port register, from where the microprocessor reads the byte.
- Asynchronous serial input is also called UART input if the serial input is according to the UART protocol .
- Asynchronous serial input is used for keypad and modem inputs.

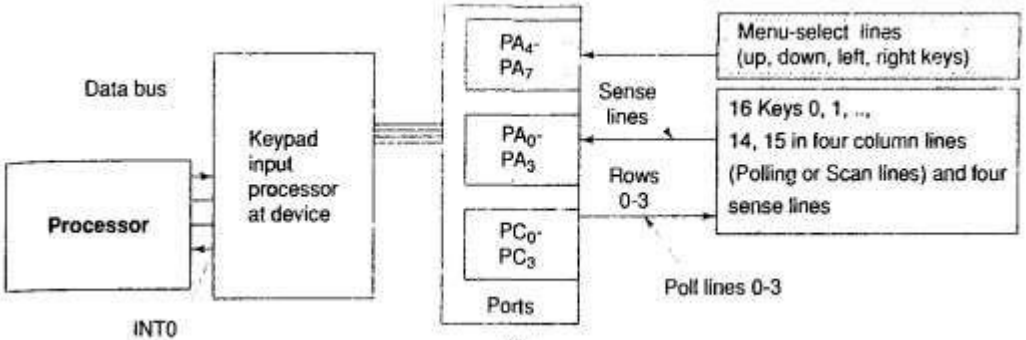
- Only difference in Output Pin TXD

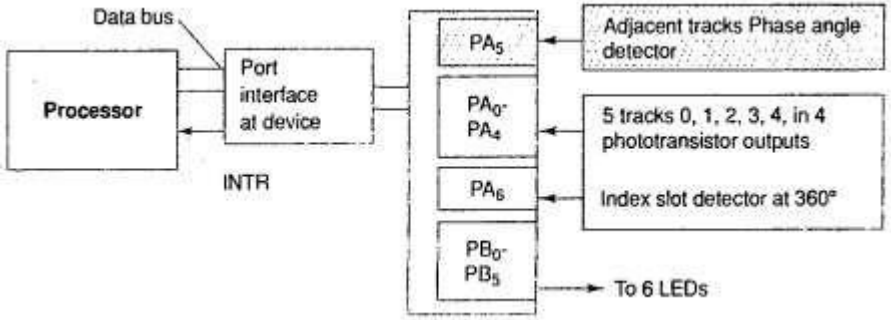
- A parallel port can have one or multi bit input or output and can be bi-directional IO
  - One hit input, output and I0
  - Eight or more hit input, output and I0

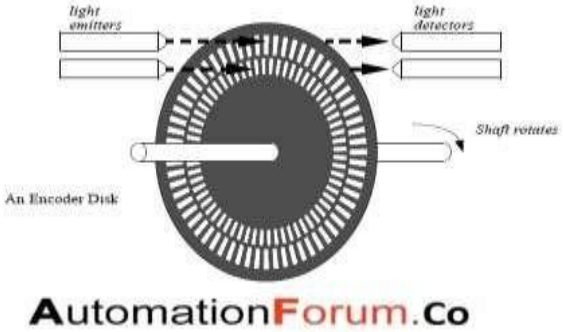
- Half duplex means that at any point communication can only be one way (input or output) on a bi-directional line.
- An example of half-duplex mode is Walki Talki communication.
- Full duplex means that the communication can he both ways simultaneously.
- An example of the full duplex asynchronous mode of communication is communication between the modem and computer through the 14) and RxD lines

- Synchronous Communication
- Asynchronous Communication
  - RS 232/RS485

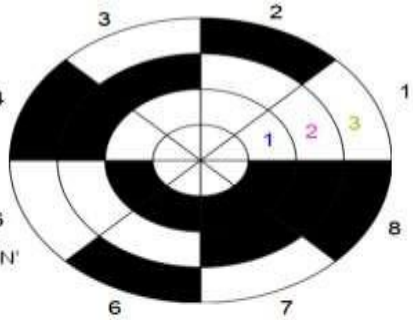

CS-Port Select
BRi-Buffer Register for Input
BRo-Buffer Register for Output
Processor
A7–A0
Port Addresses Decoder
An Input Port
CS
BRi
I0-I7
Keypad
IORD
An Output Port
CS
BRo
O0-O7
LCD Display
IOWR
An I/O Port
CS
BRi
IO0-IO7
Modem Messages or Touch Screen
IORD
BRo
IOWR
Note:
A Port can have 1 or 2 or more addresses allotted for it and a few address Bus bits at Input
(a)


Data bus
Processor
INT0
Keypad input processor at device
PA4-PA7
PA0-PA3
PC0-PC3
Ports
Sense lines
Rows 0-3
Menu-select lines (up, down, left, right keys)
16 Keys 0, 1, ..., 14, 15 in four column lines (Polling or Scan lines) and four sense lines
Poll lines 0-3
(a)


Data bus
Processor
Port interface at device
INTR
PA5
PA0-PA4
PA6
PB0-PB5
Adjacent tracks Phase angle detector
5 tracks 0, 1, 2, 3, 4, in 4 phototransistor outputs
Index slot detector at 360°
To 6 LEDs


light emitters
light detectors
Shaft rotates
An Encoder Disk
AutomationForum.Co

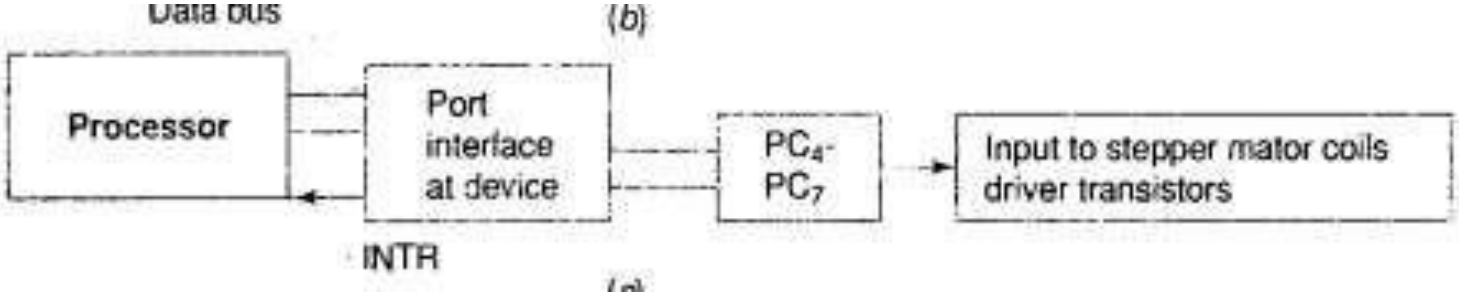
Binary Coding

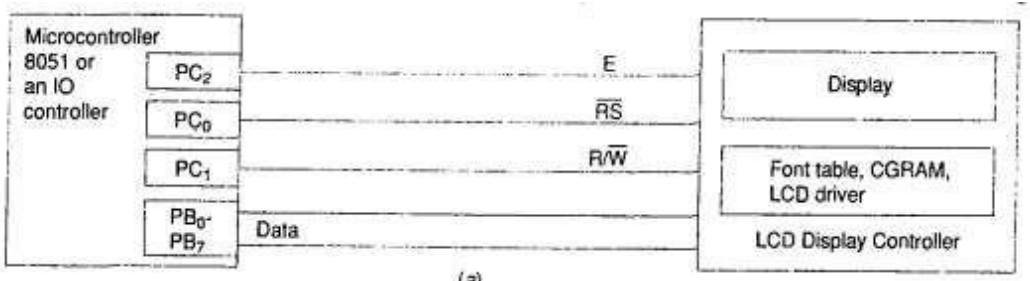


Black sectors are 'ON'

Sector	Contact 1	Contact 2	Contact 3	Angle
1	off	off	off	0° to 45°
2	off	off	on	45° to 90°
3	off	on	off	90° to 135°
4	off	on	on	135° to 180°
5	on	off	off	180° to 225°
6	on	off	on	225° to 270°
7	on	on	off	270° to 315°
8	on	on	on	315° to 360

2 bits change

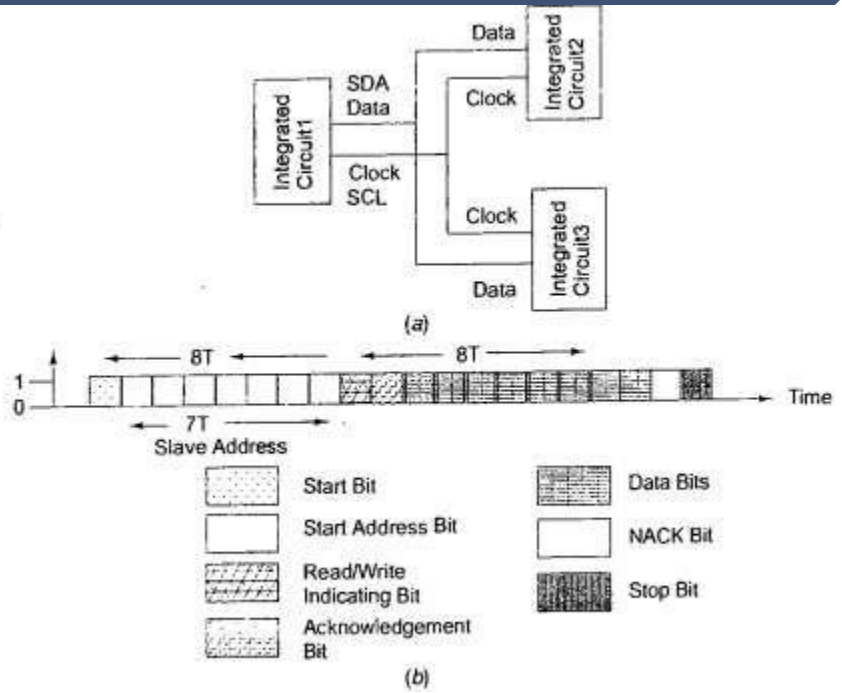

Data bus
(b)
Processor
Port
interface
at device
PC4-
PC7
Input to stepper mator coils
driver transistors
INTR


Microcontroller
8051 or
an IO
controller
PC2
PC0
PC1
PB0-
PB7
E
RS
R/W
Data
Display
Font table, CGRAM,
LCD driver
LCD Display Controller

(a)

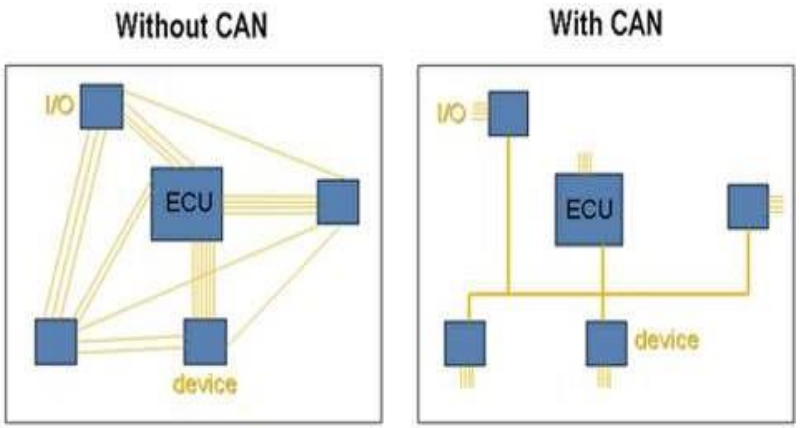
Table 3.9

Field and its length	*Explanation*
First field of 1-bit	It is start bit similar to the one in a UART.
Second field of 7 bits	It is called the address field. It defines the slave address being sent the data frame (of many bytes) by the master.
Third field of 1 control bit	It defines whether a read or write cycle is in progress.
Fourth field of 1 control bit	Next bit defines whether the present data is an acknowledgement (from the slave)
Fifth field of 8 bits	It is used for IC device data bits.
Sixth field of 1-bit	It is a negative acknowledgement bit (NACK) from the master. If active, then acknowledgement after a transfer is not needed from the slave, else acknowledgement is expected from the slave.
Seventh field of 1-bit	It is a stop bit like in a UART.


Integrated Circuit1
SDA
Data
Clock
SCL
Data
Clock
Integrated Circuit2
Clock
Data
Integrated Circuit3
(a)
8T
8T
1
0
Time
7T
Slave Address
Start Bit
Start Address Bit
Read/Write Indicating Bit
Acknowledgement Bit
Data Bits
NACK Bit
Stop Bit
(b)

CAN

- Serial communication
- Multi-Master Protocol
- Compact
  - Twisted Pair Bus line
- 1 Megabit per second

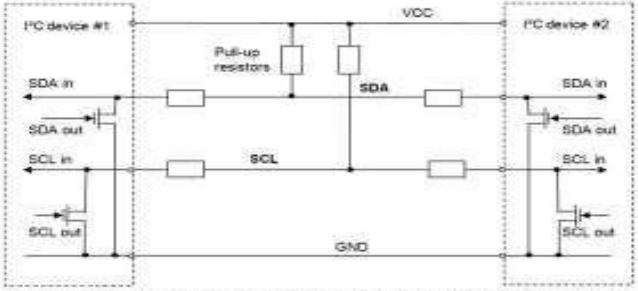


- Controller Area Networks are used in many different fields, the bulk of which are
  - Auto-motive industry
  - Factory Automation
  - Machine Control
  - Medical Equipment and devices
  - And more….

Field and its length	*Function*
First field of 12 bits	This is arbitration field, which contains the packet's 11-bit destination address and RTR bit (Packet means a set of bits sent on the bus). RTR stands for 'Remote Transmission Request'. The receiving addressed device is at destination address specified in 11-bit subfield and RTR is defined on the basis of whether the data byte being sent is a data for the device or a request to the device. 11-bit address identifies the device to which data is being sent or the request being made. When an 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.
Second field of 6 bits	It is control field. The first bit is identifier extension. The second bit is always 1. The last 4 bits are code for data length.
Third field of 0 to 64 bits	Its length depends on the data length code in control field.
Fourth field (third if data field has no bit present) is of 16 bits	It is CRC (Cyclic Redundancy Check) field with 15-bit CRC plus 1-bit delimiter bit. The receiver node uses it to detect errors, if any, during the transmission.
Fifth field of 2 bits	First bit is 'ACK slot'. The sender sends it as 1 and the receiver, which would send back 0 in this slot when it detects error in reception. The sender, after sensing 0 in the ACK slot, retransmits the data frame. The second bit is the '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.
Sixth field of 7 bits	This is the end-of-the-frame specification and has seven 0s.

I²C

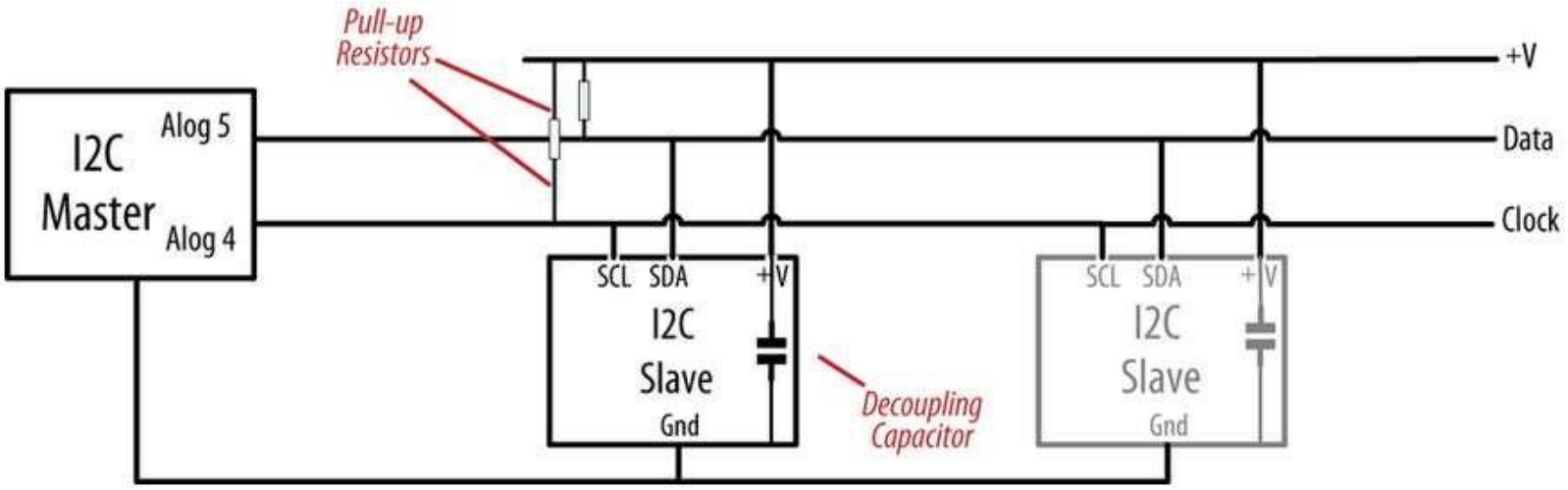
- Developed in 1982 by Philips (now NXP)
  - To connect CPU to peripherals in televisions
- Two signal lines, both are '*open drain*', thus pull-up resistors are needed
  - SDA (serial data)
  - SCL (serial clock)
- Devices are either masters or slaves
  - Master is the device that always drives SCL and initiates a transfer
- Each device has an address
- Data is sent in 8 bit bytes



http://www.byteparadigm.com/kb/article/AA-00255/22/Introduction-to-SPI-and-IC-protocols.html

Use 1.2k - 2.2k pull up resistors for Vcc = 3.3 V or 1.8 - 3.3k resistors for Vcc = 5 V

Analog pins 4 (SCL) and 5 (SDA) must be pulled up, and a common ground is needed



- Master initiates transfer with a START bit (SDA from high to low while SCL is high
- Slave address (7 or 10 bits, 7 is most common)
- Transfer type (1 bit: 0 to write, 1 to read)
  - All ICs compare address to their address
    - If address matches, device sends an ACKNOWLEDGE signal
    - If address does not match, device waits until bus is released by STOP condition

- Once master receives ACKNOWLEDGE,
  it then sends (writes) or receives (reads) data
  - Receiver sends back ACKNOWLEDGE for each byte received
- Master concludes the transfer with STOP bit

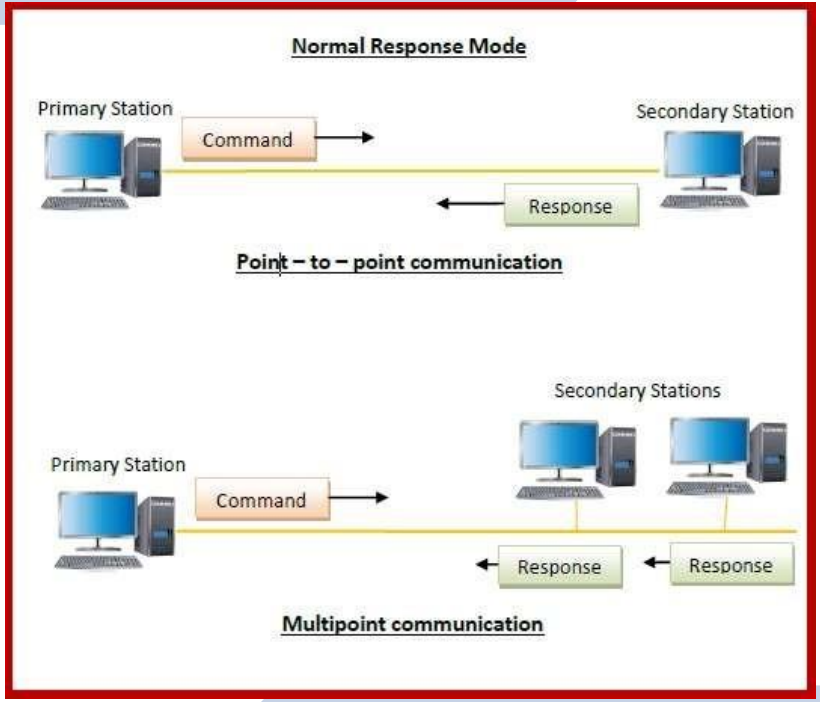
High Level Data Link Control Protocol

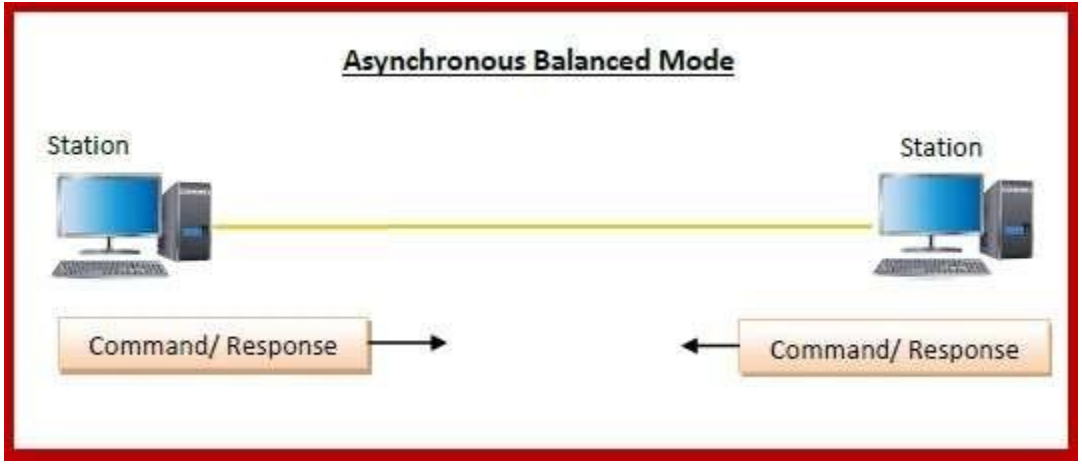
- Bit Oriented Approach – Streams are rep by bits
- Simply views the frames as a collection of bits
- The synchronous Data Link protocol developed by IBM is an example for bit oriented protocol
- SDLC are standardized by ISO as HDLC
- Main protocol for Data link layer

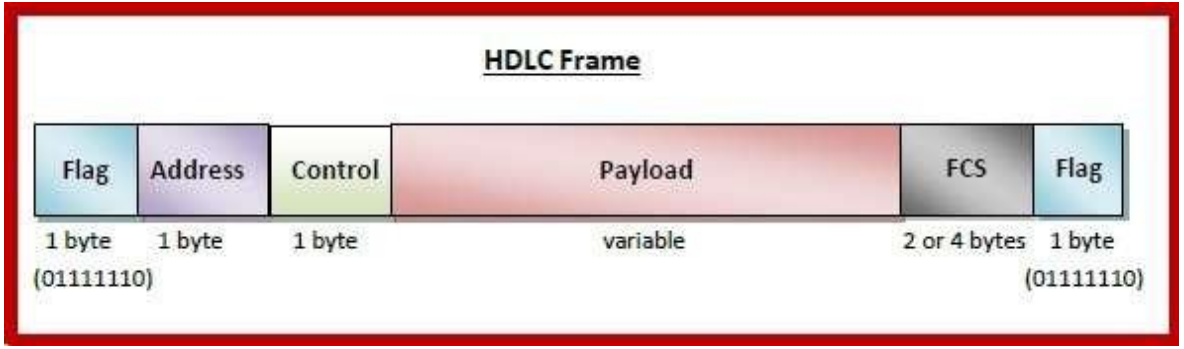
- High-level Data Link Control (HDLC) is a group of communication protocols of the data link layer for transmitting data between network points or nodes.
- Since it is a data link protocol, data is organized into frames.
- A frame is transmitted via the network to the destination that verifies its successful arrival.
- It is a bit - oriented protocol that is applicable for both point - to - point and multipoint communications.

Transfer Modes

- HDLC supports two types of transfer modes, normal response mode and asynchronous balanced mode.
- **Normal Response Mode (NRM)** – Here, two types of stations are there, a primary station that send commands and secondary station that can respond to received commands. It is used for both point - to - point and multipoint communications.
- **Asynchronous Balanced Mode (ABM)** – Here, the configuration is balanced, i.e. each station can both send commands and respond to commands. It is used for only point - to - point communications.


Normal Response Mode
Primary Station
Command
Secondary Station
Response
Point – to – point communication
Secondary Stations
Primary Station
Command
Response
Response
Multipoint communication


Asynchronous Balanced Mode
Station
Station
Command/ Response
Command/ Response

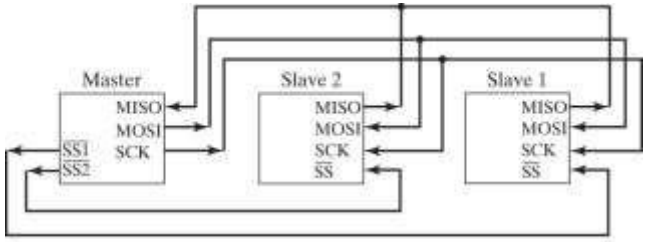

HDLC Frame
Flag
Address
Control
Payload
FCS
Flag
1 byte
(01111110)
1 byte
1 byte
variable
2 or 4 bytes
1 byte
(01111110)

- **Payload** − This carries the data from the network layer. Its length may vary from one network to another.
- **FCS** − It is a 2 byte or 4 bytes frame check sequence for error detection. The standard code used is CRC (cyclic redundancy code)

- **Flag** – It is an 8-bit sequence that marks the beginning and the end of the frame. The bit pattern of the flag is 01111110.
- **Address** – It contains the address of the receiver. If the frame is sent by the primary station, it contains the address(es) of the secondary station(s). If it is sent by the secondary station, it contains the address of the primary station. The address field may be from 1 byte to several bytes.
- **Control** – It is 1 or 2 bytes containing flow and error control information.

Synchronous Peripheral Communication

- Synchronous – Full Duplex Serial device
- Pins – Slave select – MOSI- MISO- SCLK
- Separate registers for control – status – transmit & receive data
- At least one ‘master’ and ‘slave’ needed

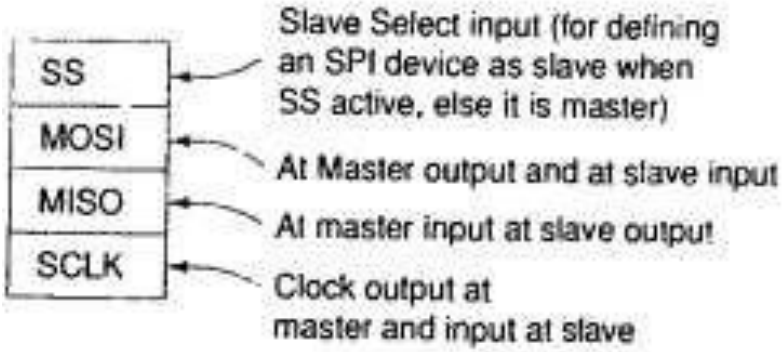


Signals

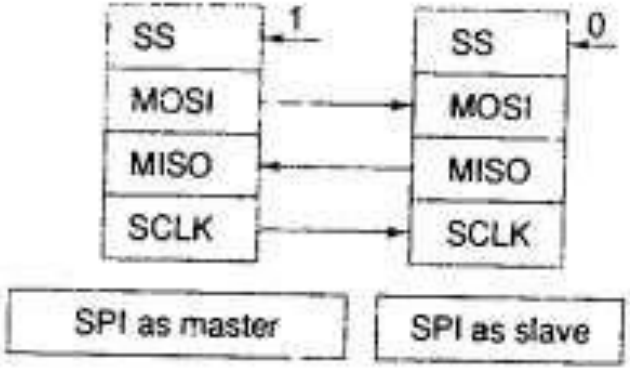
Device	MISO	MOSI	SCK	$\overline{SSx}$
Master	Input	Output	Output	Output
Slave	Output	Input	Input	Input

- At least one ‘master’ and ‘slave’ needed
- Master controls:
  - Unidirectional data line, MOSI
    - Master Out, Slave In (data from the master to the slave)
  - Shared clock line, SCK (synchronizes the data transfer)
  - Slave select line(s), SS*
    - Which slave to be addressed
- ceived by the slave, the slave clocks out a bit that is received by the master

- Slave controls:
  - Unidirectional data line, MISO
    - Master In, Slave Out (data from slave to the master)
    - Shared by slaves
    - Non-selected slaves, “tri-state” their MISO outputs
- SPI is a ‘data exchange’ protocol
  - As a bit is clocked out of the master and received by the slave, the slave clocks out a bit that is received by the master


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

(a)


SS
MOSI
MISO
SCLK
1
SS
MOSI
MISO
SCLK
0
SPI as master
SPI as slave

USB

- **Universal Serial Bus** (**USB**) provides a serial bus standard for connecting devices, usually to a computer, but it also is in use on other devices such as set-top boxes, game consoles and PDAs.
- Four wires (+5V, Return, data twisted pair)
- Up to 5 m (16.4 ft) Longer connections use hubs or active extensions

- **Asynchronous:** This is transmission at any time, with arbitrary delay between transmission of any two successive data items.
- **Synchronous:** This is continuous transmission with no gaps between transmission of successive data items.
- **Isochronous:** This is transmission at regular intervals with a fixed gap between the transmission of successive data items.

Features

- Fast
- Bi-directional
- Isochronous
- low-cost
- dynamically attachable serial interface

- USB 1.0 specification introduced in 1994
- USB 2.0 specification finalized in 2001
- Became popular due to cost/benefit advantage
  - Eg. IEEE 1394 – high bandwidth, high cost
- Three generations of USB
- USB 1.0
- USB 2.0
- USB 3.0

Module 6 - Syllabus

- Development tools: Host and Target machines – linker / locators – debugging techniques.
- S/W Architectures: Round robin-round robin with interrupt – function queue scheduling- RTOS.

Host and Target machines

- During the development process, a host system is used before locating and burning the codes in the target board.
- The target board hardware and software is later copied to get the final embedded system, which will function exactly as the one tested and debugged and finalized during the development process.

Using a Host System

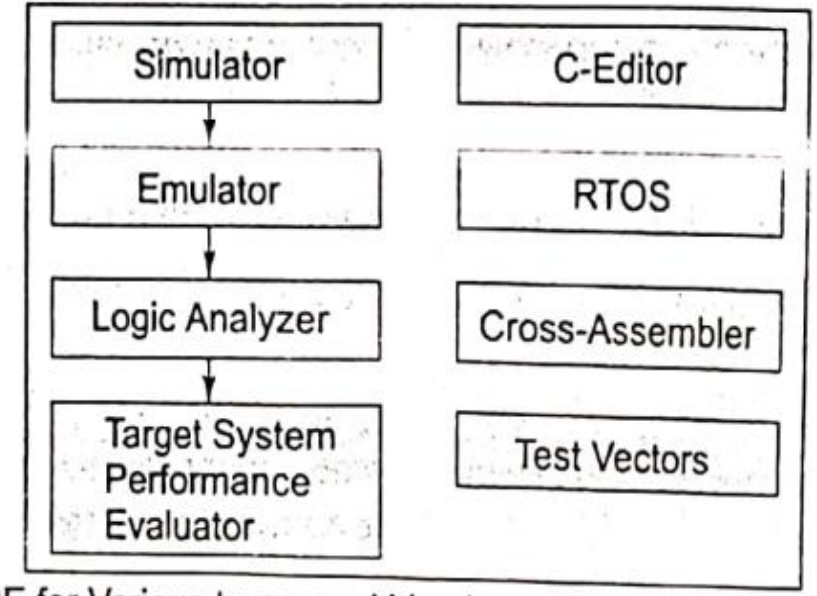
- Host system is a PC or workstation or laptop.
- It has the following hardwares.
  - High-performance processor with caches
  - Large RAM memory
  - ROM BIOS (read only memory basic input-output system)
  - Very large memory on disk
  - Keyboard , Mouse , Display monitor
  - Network connection

- In a full-fledged computer. It has software tools and must include the following:
  - Programs development kit for a high-level language program or IDE
  - Host processor compiler and cross- compiler
  - Cross-Assembler

Program Development Tool Kit

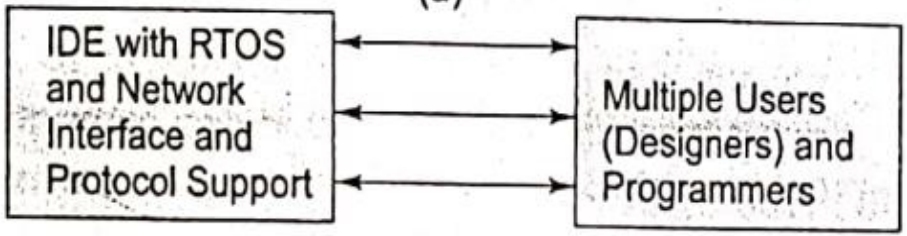
- Program development tool kit or IDE has an editor.
- Editor is used for writing C codes or assembly or C++ or Java or Visual C++ using (a)the keyboard of the host system (PC) for entering IDE with the program.
- Using GUIs, it allows the entry,naddition, deletion, insert, appending previous written lines or files, merging record and files at :the specific positions.

- A high-level language is machine-independent.
- It will have an expression like X = X + 23, or X = 2*Y+V*Z+ 19 and so on.
- When we use a high-level language C, a tool is needed for obtaining the machine codes for a target system.
- The programmer writes the mnemonics or C program, using the editor.
- The mice and keyboard combinations of the host system (PC) or host system are for entering the program codes. Each language needs a compiler.



IDE for Various types and Versions of Microcontroller with Upgradability of IDE for future Versions.

(a)



(b)

- 1. An interpreter does expression-by-expression (line-by-line) translation to the machine-executable codes.
- 2. A compiler - convert high level to machine
- 3. An assembly language program has the mnemonics that are machine-dependent.
- 4. A dissembler translates the object codes into the mnemonics form of assembly language. It helps in understanding the previously made object codes.

- 5. An assembler is a program that translates the assembly mnemonics into the binary opcodes and instructions, that is, into an executable file, called object file.
- A loader is a program that helps in this task by reallocating addresses before loading the opcode and operands in the computer memory.

Target System

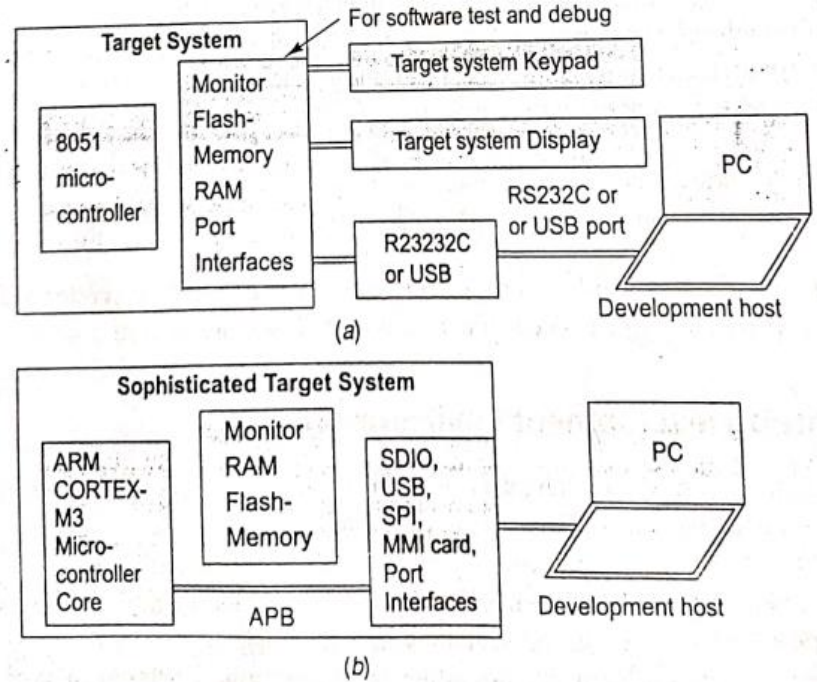
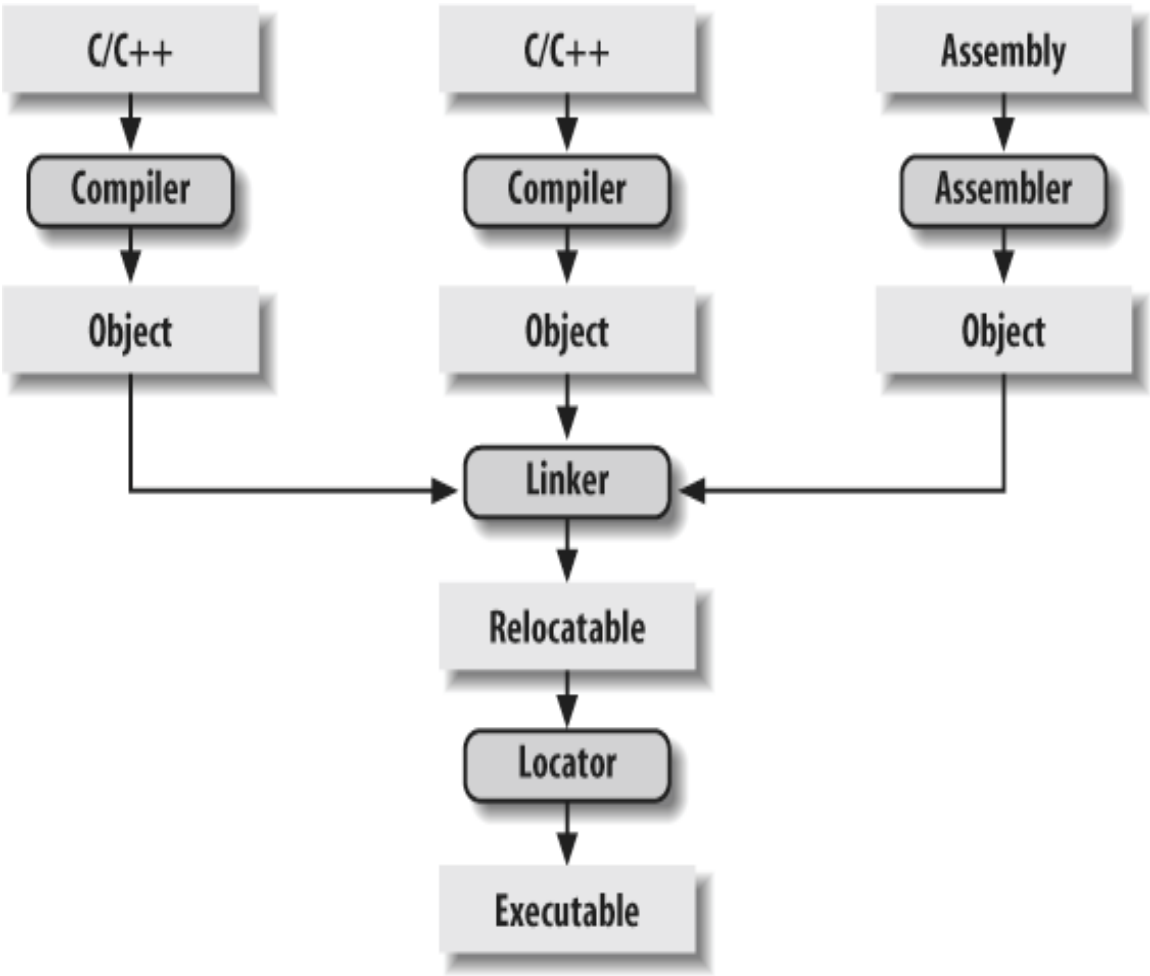


Fig. 13.3 (*a*) Simple target system (*b*) Sophisticated target system

- A target system has a processor, ROM memory for ROM image of the embedded software, RAM for stack, temporary variables and memory buffers, peripherals and interfaces.
- A target system may possess the RS232 as well as 10/100-base Ethernet connectivity or USB port for software test and debug.
- A target system differs from a final system.

Steps

- The codes of application software have to be written.
- These have to be embedded in flash.
- These have to be repeatedly written or modified and tested using diagnostic, simulation and debugging tools and embedded till a final testing in an edit-test-debug cycle shows it working according to specifications.
- The programmer later on simply copies it into the final system or product.


C/C++
C/C++
Assembly
Compiler
Compiler
Assembler
Object
Object
Object
Linker
Relocatable
Locator
Executable

Linking and Locating Software

- A linker links the compiled codes of application software.
- Linking is necessary because there are number of codes to be linked for the final binary file.
- Example – delay function
  - There are the standard codes to program a delay task for which there is a reference in the assembly language program.
  - The codes for the delay must link with the assembled codes.
  - The delay code is sequential from a certain beginning address.

Linking and Locating Software

- Example – delay function
  - The assembly software code is also sequential from a certain beginning address.
  - Both the codes are present at the distinct and the available addresses in the system.
  - A linker links these

- The linked file in binary for Trun on a computer is commonly known as executable file or simply ‘exe’ file.
- After linking, there has to be reallocation of the sequences of placing the codes before the actual placement of the codes in the memory.
- A program is loaded in a computer RAM.
- The **loader** program performs the task of reallocating the codes after finding the physical memory addresses available at a given instant.
- The loader finds the appropriate start address.

Locator

- When the code embeds into ROM or flash, a system design process locates these codes as a ROM image.
- The codes are permanently placed at the actually available addresses in flash-ROM.
- In embedded systems, there is no separate program to keep track of the available addresses at different times during the run as in a computer, In embedded systems, therefore next step after linking is the use of a locator for the program-codes and data in place-of the loader.

Types of Executable file

- Motorola S file
- Intel Hex file

Development & Debugging

- How platforms used during the design
- Programming and testing of target using host
- How hosts and other techniques can be used for debugging an embedded systems.

Development Envirnment

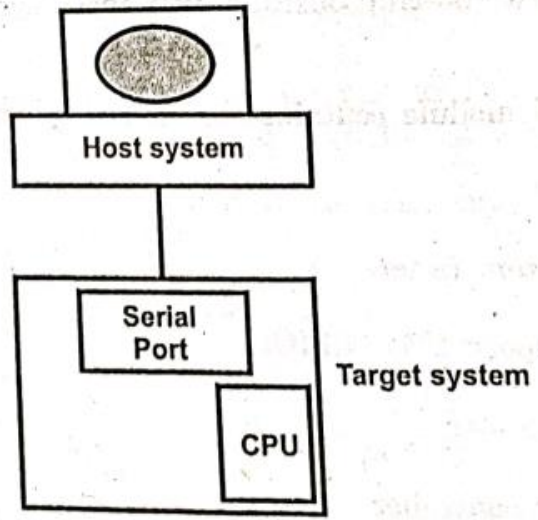


Fig. 2.40: Connecting a host and target system

Debugging

- A software debugging can be done by compiling and executing the code on a PC
- The serial port on evaluation board is most important debugging tools.
- Another very important debugging tool is the break point.
- The simplest form of break point is for the user to specify an address at-which the programs execution is to break
- When the PC reaches that address, control is returned to monitor program and execution can be continued.

LEDs as debugging devices:

- LEDs can be entertaining a simple flashing, it can be used to show error conditions, when the code enters certain routines or to show idle time activity.

In Circuit Emulator

- When software tools are insufficient to debug the system, a specialized hardware tool aids is known as microprocessor in-circuit emulator (ICE).
- ICE can help debug software in a working embedded system.
- It surrounds specialized microprocessor with additional logic that allows the users to specify break points and examine & modify the CPU state.
- The CPU provides as much debugging functionality as a debugger within a monitor program, but it does not take up any memory.
- ICE is specific to a microcontroller and expensive.

Logic Anlayser

- The logic analyzers records the values on the signals into an internal memory and then displays the results on a display once the memory is full of run is aborted.
- It captures thousands or even millions of samples of data on channels than is possible with a conventional oscilloscope.
- Logic analyzer is an array of inexpensive oscilloscopes.
- The analyzer can sample many different signals simultaneously (tens to hundreds) but can display only 0, 1, or changing values for each.

- All these logic analysis channels can be connected to the system to record the activity on many signals simultaneously.
- A logic analyzer has two modes to acquire data (i) state (ii) timing modes. These two modes represent different ways of sampling the values.

- The system's data signals are sampled at a latch within the logic analyzer.
- The latch is controlled by either the system clock or sampling clock depends whether the analyzer is being used in state or timing mode.
- After the sampling is complete, an embedded microprocessor takes over control the display of the data captured in the sample memory.

Architecture of Logic Analyser

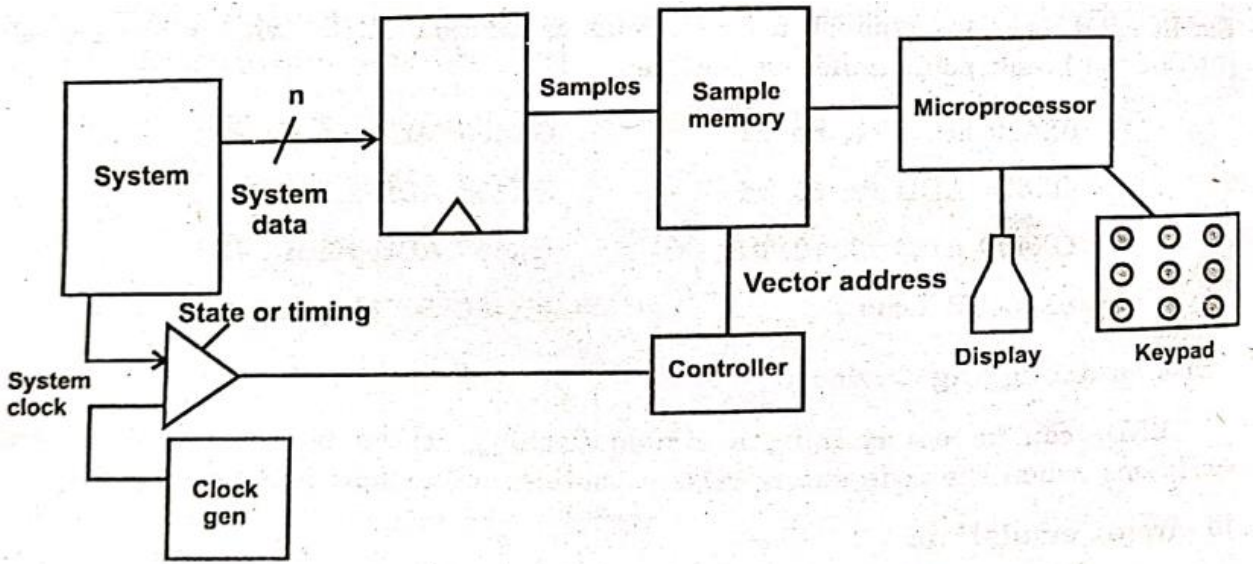


Fig. 2.41: Architecture of a logic analyzer

Embedded Software Architecture

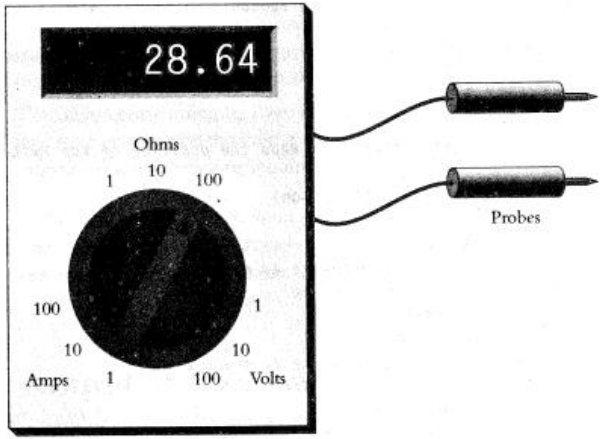
- Round Robin
- Round Robin with Interrupt
- Function queue scheduling
- RTOS

Round Robin

- Round robin is the simplest imaginable architecture.
- There is no interrupts
- The main loop simple checks each of the I/O devices in turn and service any that need service.
- Simple Architecture- no interrupts and no shared data .
- Process are dispatched in FIFO manner , but a given limited amount of time – Quantum
- Time Quantum / Time Slice – A small unit of Time (10 -100 ms)

Figure 5.1 Round-Robin Architecture

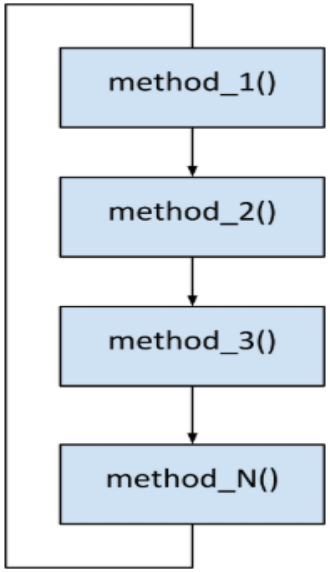
```
void main (void)
{
   while (TRUE)
   {
      if (!! I/O Device A needs service)
      {
         !! Take care of I/O Device A
         !! Handle data to or from I/O Device A
      }
      if (!! I/O Device B needs service)
      {
         !! Take care of I/O Device B
         !! Handle data to or from I/O Device B
      }
      etc.
      etc.
      if (!! I/O Device Z needs service)
      {
         !! Take care of I/O Device Z
         !! Handle data to or from I/O Device Z
      }
   }
}
```




```
void vDigitalMultiMeterMain (void)
{
   enum {OHMS_1, OHMS_10, .... VOLTS_100} eSwitchPosition;

   while (TRUE)
   {
      eSwitchPosition = !! Read the position of the switch;

      switch (eSwitchPosition)
      {
         case OHMS_1:
            !! Read hardware to measure ohms
            !! Format result
            break;
         case OHMS_10:
            !! Read hardware to measure ohms
            !! Format result
            break;
         :
         :
         case VOLTS_100:
            !! Read hardware to measure volts
            !! Format result
            break;
      }
      !! Write result to display
   }
}
```



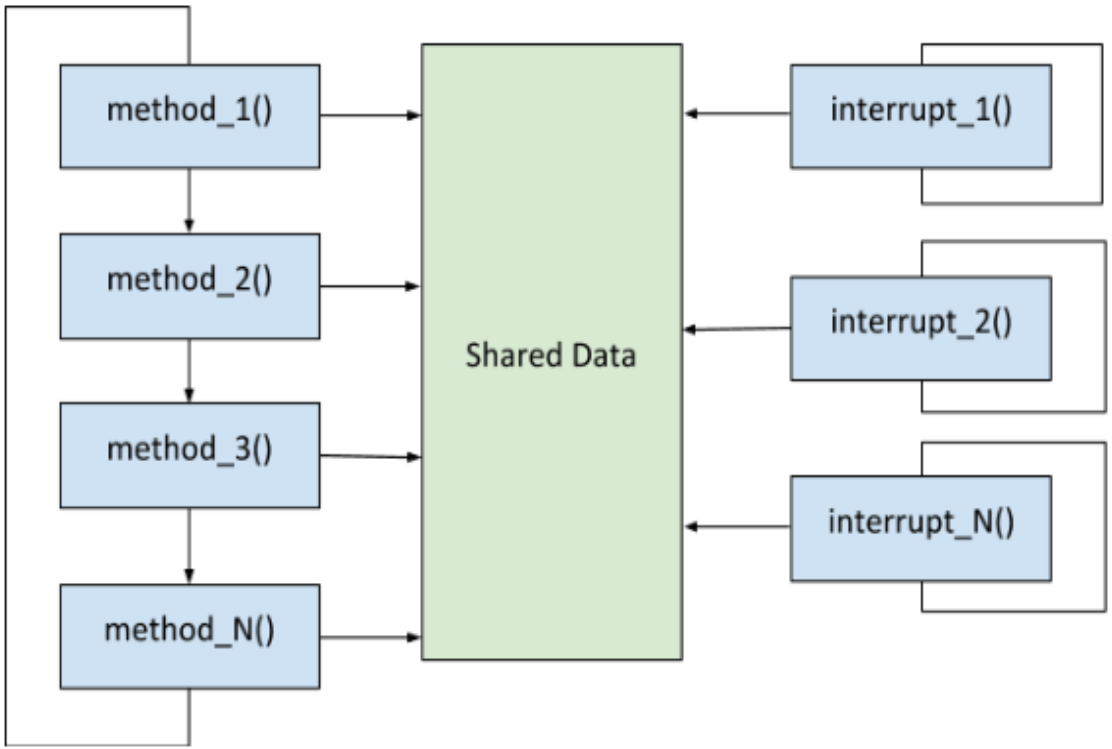
Advantage

- Simplest of all architecture
- No Interrupts
- No Shared Data
- No latency Concern
- No Tight response requirements

Drawbacks

- A sensor connected to the Arduino that urgently needs service must wait its turn.
- Fragile. Only as strong as the weakest link. If a sensor breaks or something else breaks, everything breaks.
- Response time has low stability in the event of changes to the code

Round Robin with Interrupts

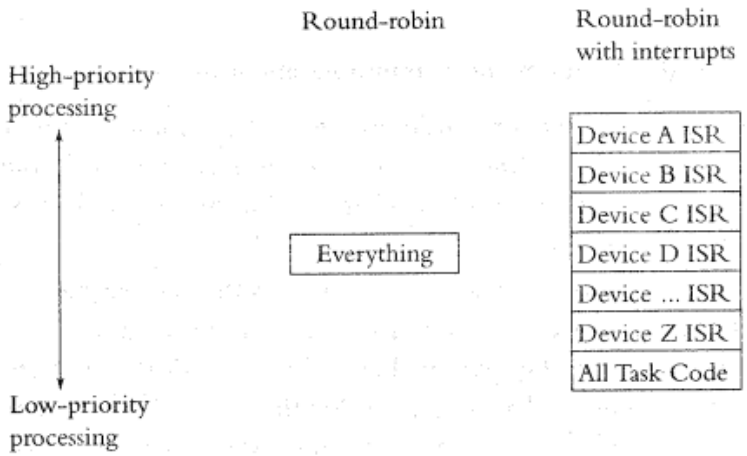


- This Round Robin with Interrupts architecture is similar to the Round Robin architecture, except it has interrupts.
- When an interrupt is triggered, the main program is put on hold and control shifts to the interrupt service routine.
- Code that is inside the interrupt service routines has a higher priority than the task code.

Drawbacks

- Shared data
- All interrupts could fire off concurrently

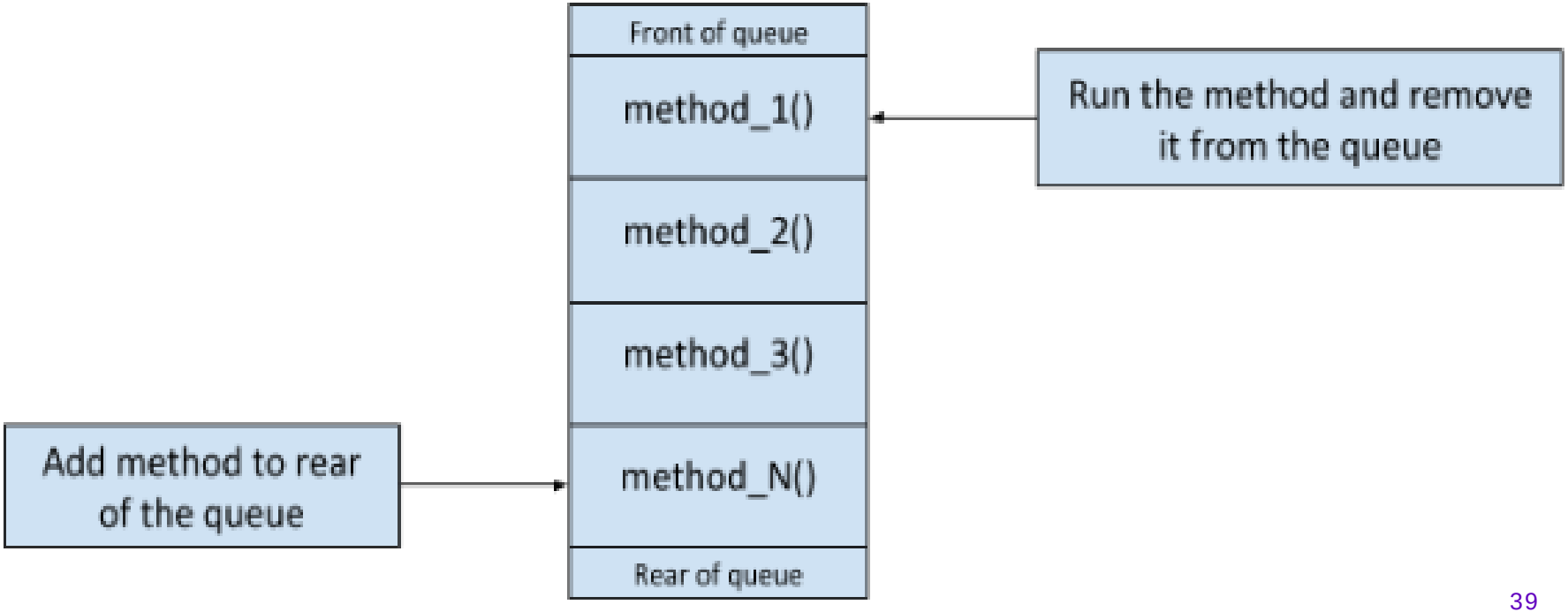
Figure 5.5 Priority Levels for Round-Robin Architectures



Advantages

- Greater control over the priority levels
- Flexible
- Fast response time to I/O signals
- Great for managing sensors that need to be read at prespecified time intervals

Function Queue Scheduling



Advantage

- In the Function Queue Scheduling architecture, interrupt routines add function pointers to a queue of function pointers.
- The main program calls the function pointers one at a time based on their priority in the queue.

Drawbacks

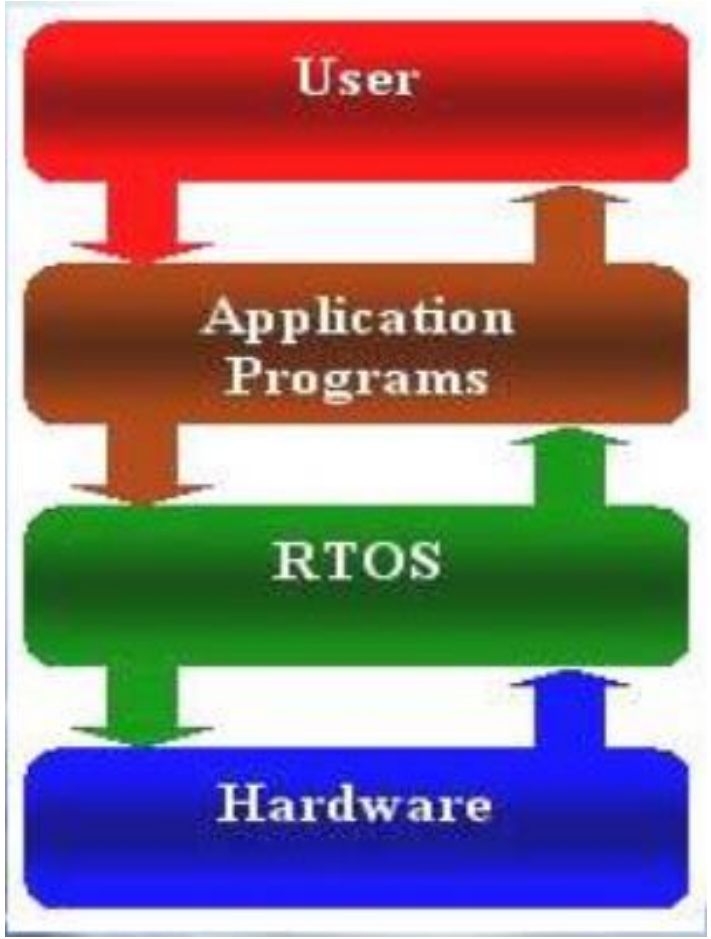
- Shared data
- Low priority tasks might never execute

- Great control over priority
- Reduces the worst-case response for the high-priority task code
- Response time has good stability in the event of changes to the code

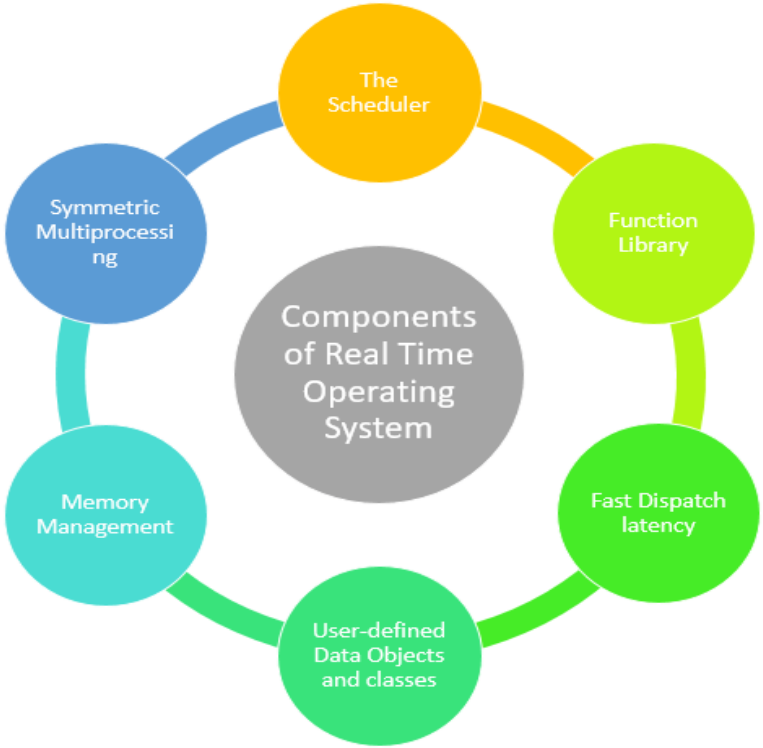
RTOs

- A real time operating system commonly known as RTOS , is a software component that are rapidly switches between tasks giving the impression that multiple programs are being executed at the same time on a single processing core.
- The difference between an OS (Operating System) such as Windows or Unix and an RTOS (Real Time Operating System) found in embedded systems, is the response time to external events.
- Normal OS – soft realtime ; RTOS – Hard Realtime

- RTOS used priority to execute the process enters in the system. Low priority tasks preempted to serve higher priority process.
- Example – traffic light


User
Application Programs
RTOS
Hardware

Components of RTOS



- **The Scheduler**: This component of RTOS tells that in which order, the tasks can be executed which is generally based on the priority.
- **Symmetric Multiprocessing (SMP)**: It is a number of multiple different tasks that can be handled by the RTOS so that parallel processing can be done.
- **Function Library**: It is an important element of RTOS that acts as an interface that helps you to connect kernel and application code. This application allows you to send the requests to the Kernel using a function library so that the application can give the desired results.

- **Memory Management**: this element is needed in the system to allocate memory to every program, which is the most important element of the RTOS.
- **Fast dispatch latency**: It is an interval between the termination of the task that can be identified by the OS and the actual time taken by the thread, which is in the ready queue, that has started processing.
- **User-defined data objects and classes**: RTOS system makes use of programming languages like C or C++, which should be organized according to their operation.

Types of RTOS

- Hard Real Time
- Firm Real Time
- Soft Real Time

Hard Real Time

- In Hard RTOS, the deadline is handled very strictly which means that given task must start executing on specified scheduled time, and must be completed within the assigned time duration.
- Example: Medical critical care system, Aircraft systems, etc.

Firm Real Time

- These type of RTOS also need to follow the deadlines. However, missing a deadline may not have big impact but could cause undesired affects, like a huge reduction in quality of a product.
- Example: Various types of Multimedia applications.

Soft Real Time

- Soft Real time RTOS, accepts some delays by the Operating system. In this type of RTOS, there is a deadline assigned for a specific job, but a delay for a small amount of time is acceptable. So, deadlines are handled softly by this type of RTOS.
- Example: Online Transaction system and Livestock price quotation System.

Characteristics of RTOS

- Deterministic
- Responsive
- Reliability
- User Control

Reference

- https://automaticaddison.com/round-robin-vs-function-queue-scheduling-embedded-software-architecture/
- https://www.highintegritysystems.com/rtos/what-is-an-rtos/#:~:text=A%20Real%20Time%20Operating%20System,on%20a%20single%20processing%20core.
- https://www.guru99.com/real-time-operating-system.html

APPENDIX I

CONTENT BEYOND THE SYLLABUS

Programming concept in high level language

High-level language (HLL) is a programming language such as C, FORTRAN, or Pascal that e High level language is the next development in the evolution of computer languages. Examples of some high-level languages are given below

- PROLOG (for "PROgramming LOGic")
- FORTRAN (for 'FORrmula TRANslation')
- LISP (for "LISt Processing")
- Pascal (named after the French scientist Blaise Pascal).

High-level languages are like English-like language, with less words also known as keywords and fewer ambiguities. Each high level language will have its own syntax and keywords. The meaning of the word syntax is grammar.

Now let us discuss about the disadvantages of high-level languages

- A high level language program can't get executed directly. It requires some translator to get it translated to machine language. There are two types of translators for high level language programs. They are interpreter and compiler. In case of interpreter, prior execution, each and every line will get translated and then executed. In case of compiler, the whole program will get translated as a whole and will create an executable file. And after that, as when required, the executable code will get executed. These translator programs, specially compilers, are huge one and so are quite expensive.

- The machine language code generated by the compiler might not be as compact as written straightaway in low-level language. Thus a program written in high-level language usually takes longer time to execute.

Now we shall discuss about the advantages of high-level languages

- High-level language programs are easy to get developed. While coding if we do some errors then we can easily locate those errors and if we miss then during compilation those errors would get detected by the compiler. And the programmer will initiate respective corrections to do needful accordingly.

- By a glance through the program it is easy to visualize the function of the program.

- The programmer may not remain aware about the architecture of the hardware. So people with our hardware knowledge can also do high level language programming.

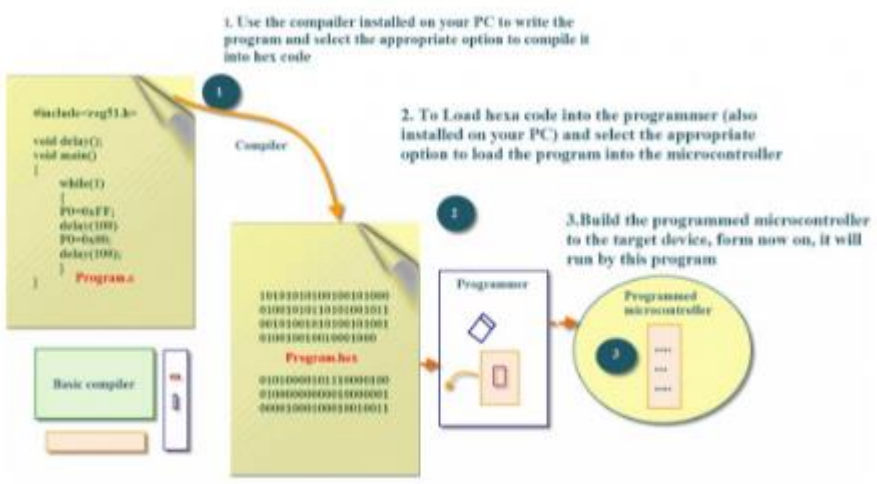
- The same high level language program works on any other computer, provided the respective compiler is available for the target new architecture. So high-level languages are portable.

- Productivity against high level language programming is enormously increased.

To conclude, high-level languages are almost always used nowadays except where very high-speed execution is required.

KEIL C programming for timers, interrupts & serial communication.

Embedded C is the most popular programming language in the software field for developing electronic gadgets. Each processor is associated with embedded software. Embedded C Programming plays a major role in performing specific functions by the processor. In our day-to-day life, we frequently use many electronic devices such as washing machines, mobile phones, digital camera and so on will work based on microcontrollers that are programmed by embedded C.

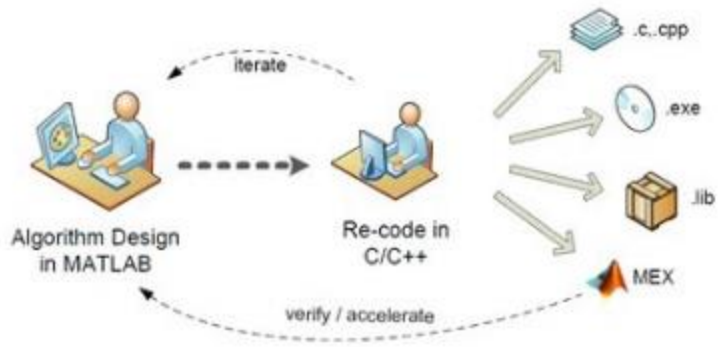


Embedded System Programming

The C code written is more reliable, portable, and scalable; and in fact, much easier to understand. The first and foremost tool is the embedded software that decides the operation of an embedded system. Embedded C programming language is most frequently used for programming the microcontrollers.

Embedded C Programming Tutorial (8051)

For writing the program the embedded designers must have sufficient knowledge on the hardware of particular processors or controllers as the embedded C programming is a full hardware related programming technique.



Programming Tutorial

Earlier, many embedded applications were developed by using assembly level programming. However, they did not provide portability to overcome this problem with the advent of various high-level languages like C, COBOL, and Pascal. However, it was the C language that got extensive acceptance for embedded systems application development, and it continues to do so.