



NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE (NAAC Accredited)

(Approved by AICTE, Affiliated to APJ Abdul Kalam Technological University, Kerala)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COURSE MATERIALS



CS 404 EMBEDDED SYSTEMS

VISION OF THE INSTITUTION

To mold 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 mold 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 mold them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- ◆ Established in: 2002
- ◆ Course offered : B.Tech in Computer Science and Engineering
M.Tech in Computer Science and Engineering
M.Tech in Cyber Security
- ◆ Approved by AICTE New Delhi and Accredited by NAAC
- ◆ Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.
- 5.

PROGRAMME EDUCATIONAL OBJECTIVES

- PEO1:** Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.
- PEO2:** Graduates will be able to Analyze, design and development of novel Software Packages, Web Services, System Tools and Components as per needs and specifications.
- PEO3:** Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.
- PEO4:** Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, Teamwork and leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

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

PROGRAM SPECIFIC OUTCOMES (PSO)

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

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

Optimization.

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

COURSE OUTCOMES

CO1	Demonstrate the role of individual components involved in a typical embedded system
CO2	Analyze the characteristics of different computing elements and select the most appropriate one for an embedded system
CO3	Model the operation of a given embedded system.
CO4	Substantiate the role of different software modules in the development of an embedded system
CO5	Develop simple tasks to run on an RTOS
CO6	Examine the latest trends prevalent in embedded system design

MAPPING OF COURSE OUTCOMES WITH PROGRAM OUTCOMES

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12
CO1	3	2	1							3		3
CO2	3	2	1	2						3		3
CO3	3	2	1	2						3		3
CO4	3	2	1	2						3		3
CO5	3	3	2							3		3
CO6	3	3	2			2				3		3

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

CO PSO Mapping

CO'S	PSO1	PSO2	PSO3
C412.1	3	3	-
C412.2	2	2	3
C412.3	2	-	-
C412.4	-	-	3
C412.5	3	3	3
C412.6	3	-	3

SYLLABUS

Course code	Course Name	L-T-P -Credits	Year of Introduction
CS404	Embedded Systems	3-0-0-3	2016

Course Objectives:

- To introduce the technologies behind embedded computing systems.
- To introduce and discuss various software components involved in embedded system design and development.
- To expose students to the recent trends in embedded system design.

Syllabus:

Introduction to embedded systems, basic components, its characteristics. Modelling embedded systems, firmware development. Integration and testing of embedded systems, development environment. Characteristics of RTOS, interrupt handling, creating tasks in a typical RTOS. Embedded product development life cycle.

Expected Outcome:

The Student will be able to :

- i. demonstrate the role of individual components involved in a typical embedded system
- ii. analyze the characteristics of different computing elements and select the most appropriate one for an embedded system
- iii. model the operation of a given embedded system
- iv. substantiate the role of different software modules in the development of an embedded system
- v. develop simple tasks to run on an RTOS
- vi. examine the latest trends prevalent in embedded system design

References:

1. J Staunstrup and Wayne Wolf, Hardware / Software Co-Design: Principles and Practice, Prentice Hall.
2. Jean J. Labrose, Micro C/OS II: The Real Time Kernel, 2e, CRC Press, 2002.
3. Raj Kamal, Embedded Systems: Architecture, Programming and Design, Third Edition, McGraw Hill Education (India), 2014.
4. Shibu K.V., Introduction to Embedded Systems, McGraw Hill Education (India), 2009.
5. Steve Heath, Embedded System Design, Second Edition, Elsevier.
6. Wayne Wolf , Computers as Components-Principles of Embedded Computer System Design, Morgan Kaufmann publishers, Third edition, 2012.

Course Plan			
Module	Contents	Hours	End Sem. Exam Marks
I	Fundamentals of Embedded Systems- complex systems and microprocessors- Embedded system design process .Specifications- architecture design of embedded system-design of hardware and software components- structural and behavioural description.	6	15%
II	Hardware Software Co-Design and Program Modelling – Fundamental Issues, Computational Models- Data Flow Graph, Control Data Flow Graph, State Machine,. Sequential Model, Concurrent Model, Object oriented model, UML	9	15%
FIRST INTERNAL EXAMINATION			
III	Design and Development of Embedded Product – Firmware Design and Development – Design Approaches, Firmware Development Languages.	6	15%
IV	Integration and Testing of Embedded Hardware and Firmware- Integration of Hardware and Firmware. Embedded System Development Environment – IDEs, Cross Compilers, Disassemblers, Decompilers, Simulators, Emulators and Debuggers.	6	15%
SECOND INTERNAL EXAMINATION			
V	RTOS based Design – Basic operating system services. Interrupt handling in RTOS environment. Design Principles. Task scheduling models. How to Choose an RTOS. Case Study – MicroC/OS-II.	9	20%
VI	Networks – Distributed Embedded Architectures, Networks for embedded systems, Network based design, Internet enabled systems. Embedded Product Development Life Cycle – Description – Objectives -Phases – Approaches1. Recent Trends in Embedded Computing.	6	20%
END SEMESTER EXAM			

Question Paper Pattern

1. There will be **FOUR** parts in the question paper – A, B, C, D
2. Part A
 - a. Total marks : 40
 - b. **TEN** questions, each have 4 marks, covering all the **SIX** modules (**THREE** questions from modules I & II; **THREE** questions from modules III & IV; **FOUR** questions from modules V & VI). **All** questions have to be answered.
3. Part B
 - a. Total marks : 18
 - b. **THREE** questions, each having 9 marks. One question is from module I; one question is from module II; one question **uniformly** covers modules I & II.
 - c. **Any TWO** questions have to be answered.
 - d. Each question can have **maximum THREE** subparts.
4. Part C
 - a. Total marks : 18
 - b. **THREE** questions, each having 9 marks. One question is from module III; one question is from module IV; one question **uniformly** covers modules III & IV.
 - c. **Any TWO** questions have to be answered.
 - d. Each question can have **maximum THREE** subparts.
5. Part D
 - a. Total marks : 24
 - b. **THREE** questions, each having 12 marks. One question is from module V; one question is from module VI; one question **uniformly** covers modules V & VI.
 - c. **Any TWO** questions have to be answered.
 - d. Each question can have **maximum THREE** subparts.
6. There will be **AT LEAST 50%** analytical/numerical questions in all possible combinations of question choices.

QUESTION BANK

MODULE I

Q:NO:	QUESTIONS	CO	KL	PAGE NO:
1	Define a system and an Embedded system.	CO1	K5	3
2	Write the Classification of Embedded systems.	CO1	K3	4
3	Write a note on Design metrics.	CO1	K2	7
4	Write a note on Complex systems & Microprocessors.	CO1	K3	8
5	Describe Embedded hardware components	CO1	K5	12
6	List software components of an Embedded system.	CO1	K2	17
7	Describe software tools for designing an embedded system	CO1	K5	20

MODULE II

1	Explain the Hardware software Co-design & Program Modeling	CO2	K2	24
2	Define Data flow graph	CO2	K4	31
3	Explain the characteristics of Embedded system	CO2	K2	33
4	Classify different design metrics in design process	CO2	K5	35
5	Explain Embedded system with neat block diagram.	CO2	K5	37
6	Describe the issues in Hardware –Software Co-design	CO2	K3	39
7	Describe sequential model for ACVM.	CO2	K5	42

MODULE III

1	Define Embedded firmware Design & Development.	CO3	K3	48
2	Describe the Embedded OS approach.	CO3	K3	50
3	Describe the Embedded firmware development Languages.	CO3	K2	51
4	Explain High Level Language based Development.	CO3	K3	54
5	Write a note on Mixing Assembly & High level language.	CO3	K5	55
6	Explain Programming in Embedded C.	CO3	K3	56
7	Write note on Compiler Vs Cross-Compiler	CO3	K2	57

MODULE IV

1	What is an OS?	CO4	K2	60
2	Explain Integration of H/w and Firmware.	CO4	K1	63
3	List the Types of files generated on Cross-compilation.	CO4	K2	67
4	Short note on Disassembler /DE compiler	CO4	K3	71
5	Write note on Simulators, Emulators & Debugging	CO4	K1	72
6	Describe Monitor Program Based Firmware Debugging.	CO4	K2	75
7	Explain the Embedded system Development Environment.	CO4	K3	79

MODULE V

1	List the types of Operating System.	CO5	K4	84
2	Explain FIFO task scheduling	CO5	K2	85
3	Write a note on Robin Round scheduling	CO5	K3	89
4	Write a note on Functional Requirements	CO5	K2	91

5	Explain RTOS & Functions	CO5	K3	94
6	Explain Interrupt Handling in RTOS	CO5	K2	93
7	Write RTOS Design Principles.	CO5	K2	98

MODULE VI

1	Define Network Embedded Systems	CO5	K4	104
2	Explain serial bus communication Protocol	CO5	K2	106
3	Describe CAN Bus and USB Bus	CO5	K3	107
4	Define ISA Bus	CO5	K2	109
5	Define Internet Enabled systems	CO5	K3	111
6	Explain TCP	CO5	K2	112
7	List the functions of UDP	CO5	K2	112

MODULE NOTES

- .
- .
- .

MODULE I

Introduction to Embedded Systems:

- > Understanding the Basic Concepts
- > Fundamentals of Embedded S/m
- > The typical Embedded Systems Design process
 - Characteristics - Architecture design
 - Quality Attributes. H/w & S/w components.
 - Structural & behavioural description

Question Bank: (Module I & II).

1. Define a S/m & an Embedded S/m. x
2. List 3 main components of an Embedded S/m. x
3. Write short notes on Computational models.
4. Define Control Data Flow Graph.
5. Demonstrate the role of individual components of involved in a typical embedded S/m.
6. Explain the challenges in Embedded S/m design.
7. Analyse the characteristics of diff. Pgm Models.
8. Classify diff. State Machines with eg;
9. Differentiate b/w Concurrent Model & Object Oriented Model.
10. Identify the features of Seq. Model.
11. ~~B~~ Write short notes on program models.
12. Define Data Flow Graph.
13. Define Microprocessor with basic Inal units.
14. List 3 main constraints on Embedded S/m design.
15. Explain the characteristics of Embedded S/m.
16. Discuss in detail about the classificⁿ of Embedded S/m.
17. Explain design process in Embedded S/m.
18. Classify diff. design metrics in design process.

19. Explain Program Models.
20. Describe the characteristics of ^{Synchronous} Sequential Data Flow Graphs.
21. Differentiate b/w Data Flow graph & Control Data flow graphs.
22. Define Small Scale Embedded Systems.
23. List Diff. Computational Models.
24. Write short notes on Object Oriented Model.
25. Define State Machine.
26. Explain Characteristics of Embedded Systems.
27. Discuss in detail about the classification of Embedded Systems.
28. Explain the challenges in Embedded S/m.
29. Explain & Embedded S/m with neat block diagrams.
30. Discuss in detail about the concepts during the design process of Embedded S/m.
31. Explain the abstraction of Embedded S/m design process.
32. Describe the issues in R/w - s/w Co-Design.
33. Define Concurrent Model with example.
34. Describe Sequential Model for AC VM.
35. Explain Performance Accelerators.

Fundamentals of Embedded Systems:

Systems:

A system is a way of working, organizing or doing one or many tasks acc. to a fixed plan, pgm, or set of rules. It is an arrangement in which all its units assemble and work together acc. to the plan or program.

Embedded System:

An embedded system is a system that has embedded s/w & computer h/w, which makes it a s/m dedicated for an application or specific part of an applⁿ or product or part of a larger system.

• Computer - components → μ processor, Large memory, I/O units, Networking units, O.S.

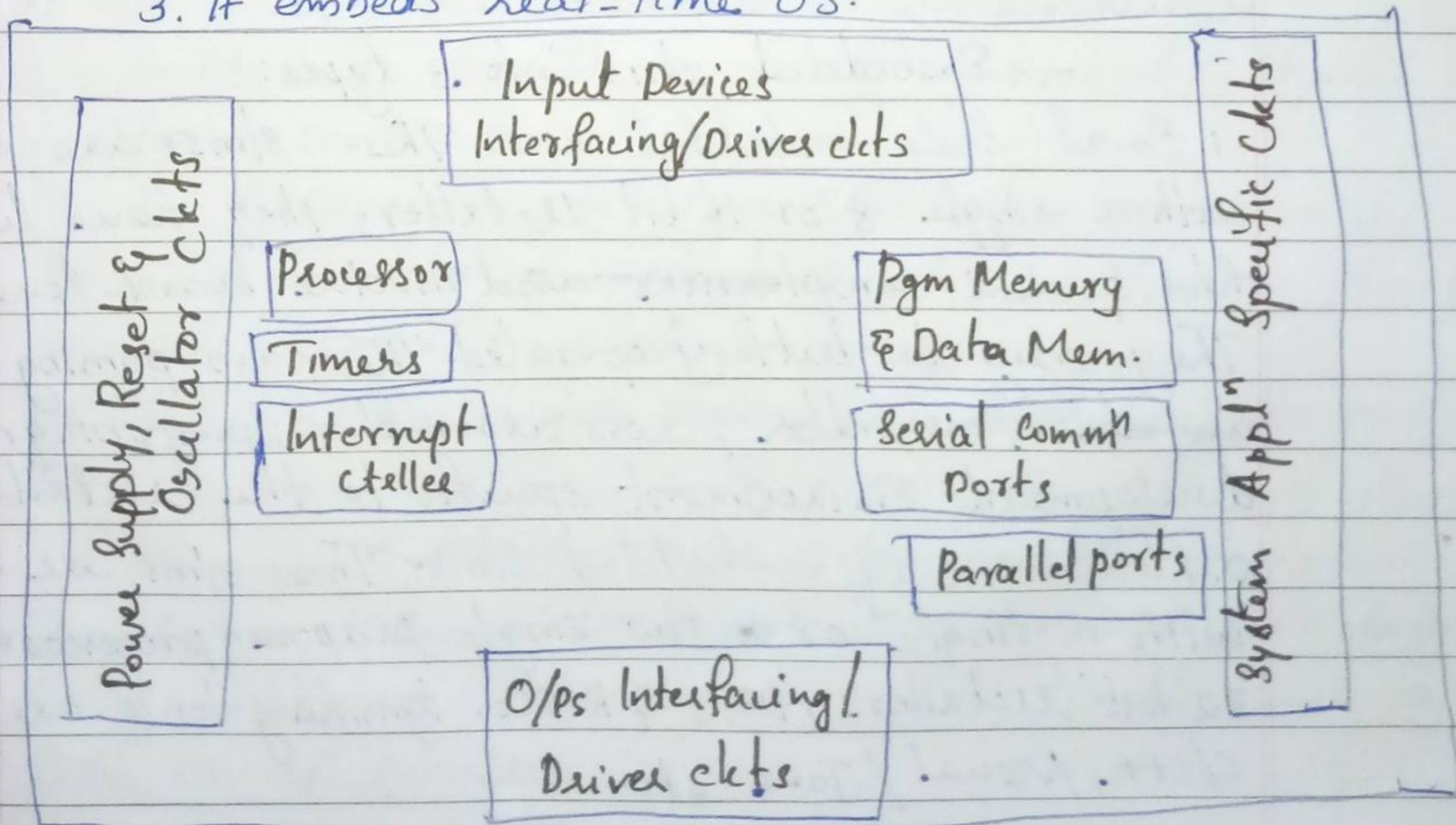
— An embedded s/m is a s/m that has 3 main components embedded into it:

1. It embeds h/w similar to a computer

Eg: s/w embeds in the ROM or flash memory.

2. It embeds main applⁿ s/w.

3. It embeds real-time OS.



Characteristics of Embedded System.

An embedded s/m is characterized by the following:

1. Realtime & multirate operations define the ways in which the s/m works, reacts to events, interrupts, & schedules the s/m's functioning in real time.

For eg: , audio, video, data, n/w stream & events have different rates & time constraints.

2) Complex Algorithms:

3) Complex Graphic user Interfaces

4) Dedicated functions.

Constraints:

An embedded s/m is designed keeping in view 3 constraints:

1. available s/m-memory

2. available processor-speed

3. the need to limit power dissipation

Eg: wait for events, run, stop, wake-up, sleep.

Classification of Embedded Systems:

→ Embedded s/ms into 3 types:

1. **Small scale embedded s/ms:** These s/ms are designed with a single 8 or 16 bit μ controller; they have little h/w & sw complexities and involve board-level design. They may be battery operated. The main pgming tools are an editor, assembler, cross assembler, an integrated development Environment specific to the μ controller.

2. **Medium scale embedded s/ms:** These s/ms are designed with a single or a few single purpose processors 8, 16 or 32 bit μ controllers, DSPs & RISCs. Pgmming tools are available C/C++, Visual/Java etc.

Sophisticated embedded systems: These s/ms have enormous h/w & s/w complexities & may need several IPs, ASIPs, scalable processors or configurable logic arrays. They are constrained by the processing speeds available in their h/w units. - cutting edge applications.

ASIP → Application specific Instrⁿ set processor

Design Process in Embedded Systems:

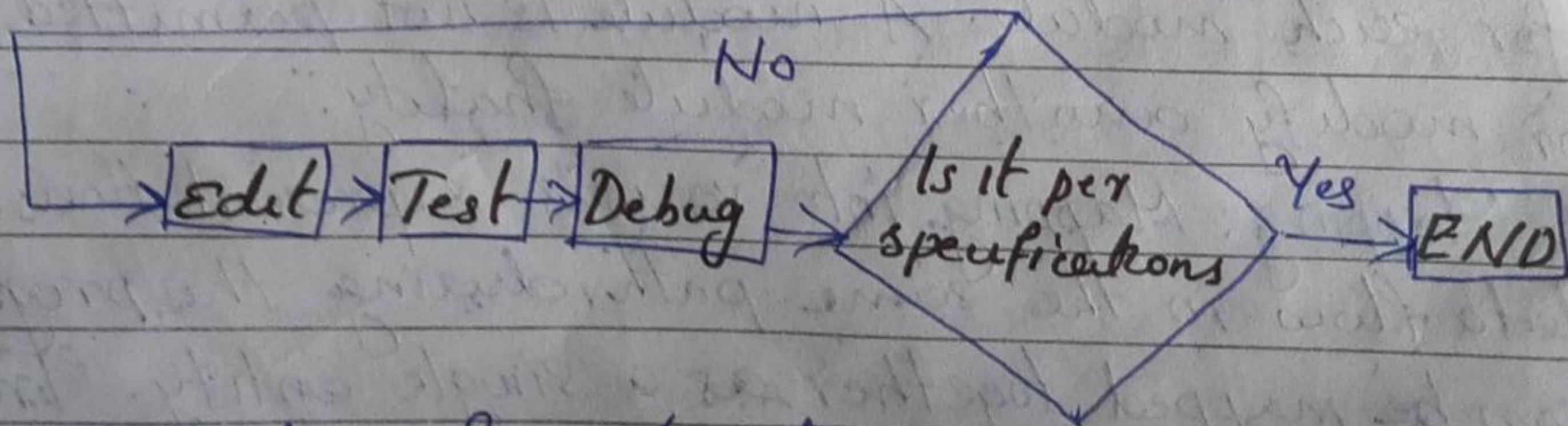
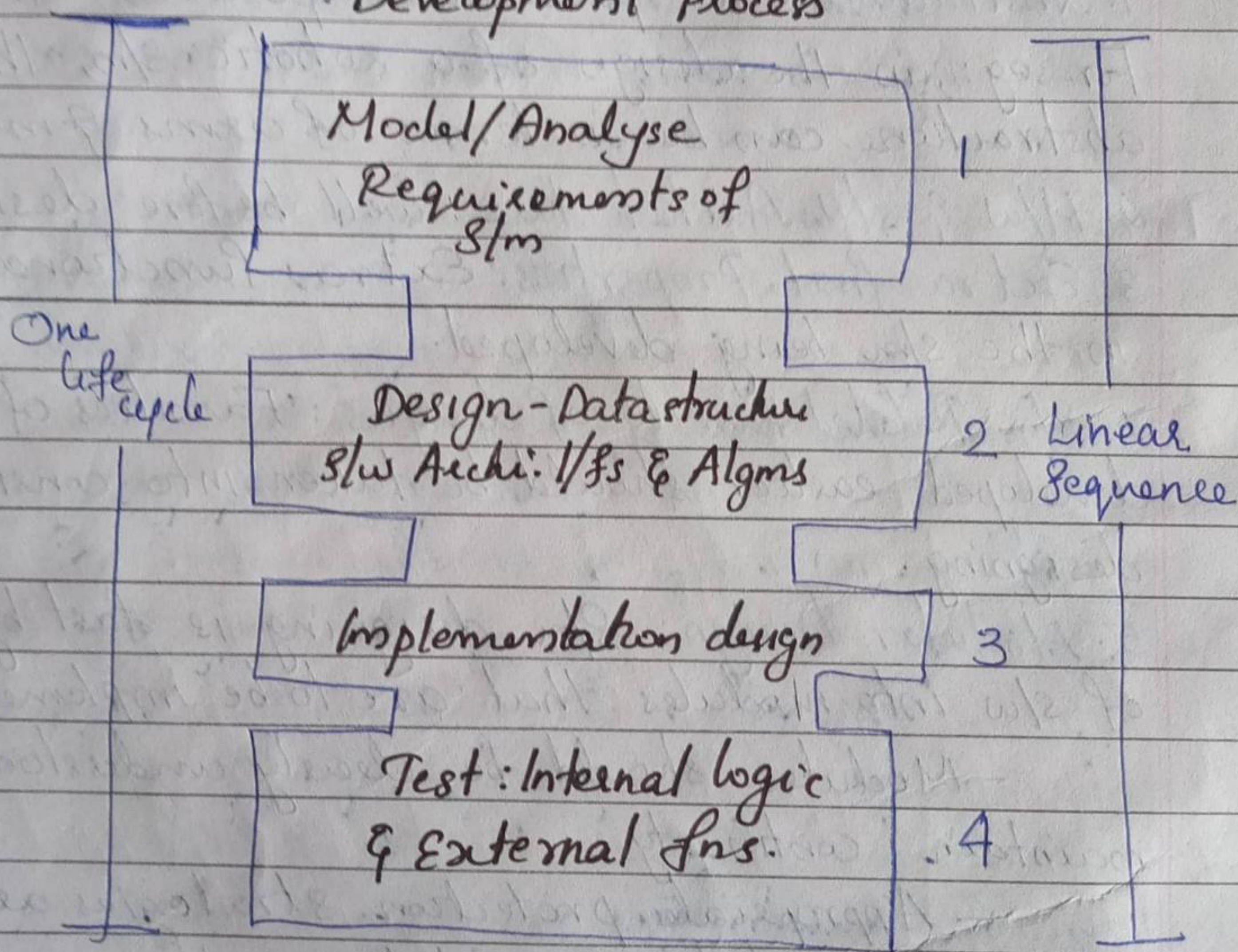
The concepts used during a design process are as follows:

- 1. Abstraction:** Each problem component is first abstracted. For eg:, in the design of a robotic s/m, the pblm of abstraction can be in terms of arms & motors.
- 2. H/w & s/w Architecture:** well before design.
- 3. Extra final Properties:** Extra functionalities required in the s/m being developed.
- 4. S/m Related Family of designs:** Families of related s/ms developed earlier should be taken into consideration during designing.
- 5. Modular Design:** S/m designing is fast by decomposition of s/w into modules that are to be implemented.
 - Modules should be clearly understood & should maintain continuity.
 - Appropriate protection strategies are necessary for each module. A module is not permitted to change or modify another module finality.
- 6. Mapping:** Mapping into various representations. For eg: data flow in the same path during the program flow can be mapped together as a single entity. Transform & transaction mapping design processes are used in designing. For eg; an image is i/p data to a s/m; it can have a diff. no. of pixels & colours.

User Interface Design: important part of design. For eg; in an automatic chocolate vending m/c s/m, the user i/f is an LCD multiline graphics display. It can display a welcome msg as well as specify the coins needed to be inserted into the m/c for each type of chocolate. It may be designed with touch screen use i/f.

Refinements: Each component & module design needs to be refined iteratively till it becomes the most appropriate for implementation by the s/w team.

Development Process



Activities for s/w design during an embedded s/w devpt process.

Design Metrics:

- 1. Power Dissipation:** For many s/ms, battery operated s/ms, such as mobile phone or digital camera the power consumed by the s/m is an imp. feature. The battery needs to be recharged less frequently.
- 2. Performance:** Instructions exⁿ time in the s/m measures the performance. Smaller exⁿ time means higher performance.
- 3. Process Deadlines:** There are no. of processes in the s/m. These have deadlines ^{within} which each of them may be required to finish computations & give results.
- 4. User interfaces:** These include keypad CUIs & VUIs.
- 5. Size:** Size of the s/m is measured in terms of (i) physical space required (ii) RAM in KB & internal flash memory reqmts in MB or GB.
- 6. Engineering Cost:** Initial cost of developing, debugging & testing the h/w & s/w.
- 7. Manufacturing Cost:** Cost of manufacturing each unit.
- 8. Flexibility:** Flexibility in design enables, without any significant engg. cost, development of diff. versions of pdt & advanced versions later on.
- 9. Prototype:** ^{development time} Time taken in days or months for developing the prototype & in-house testing for s/m finalities. It includes engg. time & making the prototype time.
- 10. Time to market:** Time taken in days or months after prototype devpt to put a pdt for users & consumers.
- 11. System & user safety:** S/m safety in terms of accidental fall from hand or table, theft & in terms of user safety when using product.

Complex Systems & Microprocessors:

Microprocessors:

The CPU is a unit that centrally fetches & processes a set of general purpose instructions. The CPU instruction set includes instructions for data transfer op's, ALU op's, stack op's, IO op's & program control, sequencing op's. The general purpose instⁿ set is always specific to a specific CPU. CPU must possess the following basic functional units.

1. A control unit that fetches & controls the sequential processing of a given command or instⁿ.
2. An ALU undertakes arithmetic & logical op's on bytes or words.

A microprocessor is a single VLSI chip that has a CPU & some other units (eg: floating pt. unit, pipelining unit).

Earlier generation μ processor's fetch & execute cycle was guided by a clock fr. of order of $\sim 4\text{MHz}$. Now operate at clock fr. of 4GHz .

High performance processors have pipeline & super scalar architecture, fast ALUs & floating pt. processing units. The important μ processors used in the embedded systems are ARM, 80x86 & SPARC family of μ processors.

A general purpose Processor μ processor can be embedded on a VLSI chip. Diff. streams of μ processors embedded in a complex s/m design.

Stream Processor family Source CISC or RISC or Both features

Stream 1	68 HCxxx	Motorola	CISC
Stream 2	80x86	Intel	CISC
Stream 3	SPARC	Sun	RISC
Stream 4	ARM	ARM	RISC with CISC family

Abstraction of Steps in the Design Process:

A design process is called bottom to top design if it builds by starting from the components. A design process is called top to down design if it starts with abstraction of the process & then after abstraction the details are created. Top to down is the most favored approach.

5 levels of abstraction from top to bottom in the design process.

1. **Requirements**: Definition & analysis of s/m requirement. It is only by a complete clarity of the required purpose, inputs, o/p, timing, design metrics & validation reqmts for finally developed s/ms specific^{ns} that a well designed s/m can be created.

2. **Specifications**: Clear specifications of the required s/m are must. - precise, guide customer expectations from the product. Guide s/m architecture. The designer needs specifications for (i) h/w, eg: peripherals, devices, processor & memory specific^{ns}.

ii) data types & processing specifications

iii) expected s/m behaviour specifications

iv) constraints of design

v) expected life cycle specifications.

Architecture: Data modeling designs of attributes of data structure, data flow graphs, s/w architecture layers & h/w architecture are defined.

s/w architectural layers are

1. 1st layer - architectural design: A design for s/w architecture is developed. - how the elmts - data structures, databases, algms, ctrl fns, state transⁿ fns, process, data & pgm flow are to be organized.
2. Second layer - data design - Design of datastructure & databases.
3. Third layer - Interface design - Interfaces to integrate the components.

Components: Component level design. - design of each component. Each component should be optimised for mem. usage & power dissipation. Components of h/w, processes, i/f's & algms. Hardware components are:

1. Processor & single purpose processor in the s/w.
2. Memory RAM, ROM or internal & external flash or secondary memory.
3. Peripherals & devices internal & external to the s/w.
4. ports & buses in the s/w.
5. Power source or battery in the s/w.

System Integration: Built components are integrated in the s/w. Components work independently. Each component & its i/f s/w is integrated after the design stage. Program implementation is in a lang. & may use an integrated development environment & source code engg. tools.

Challenges in Embedded System Design: Optimizing Design Metrics.

Following are the challenges that arise during the design process.

Amount & type of h/w needed: Optimizing the eqmnt of μprocessors & single purpose processors in the s/m on the basis of performance, power dissipation, cost & other design metrics are the challenges in a s/m design.

Optimizing Power Dissipation & Consumption: Power, consumption during the operational & idle state of s/m should be optimal.

The following methods are used to meet the design challenges.

Clock Rate Reduction: Power dissipation typically reduces $2.5 \mu\text{W}$ per 100kHz of reduced clock rate. So reduction from 8000kHz to 100kHz reduces power dissipation by about $200 \mu\text{W}$, which is nearly similar to when the clock is non final.

The power $25 \mu\text{W}$ is typically the residual dissipation needed to operate the timers & few other units. By operating the clock at a lower frequency or during the power-down mode of the processor, the advantages are:

- i) Power loss due to heat generation reduces.
- ii) Radio frequency interference also reduces due to the reduced power dissipation within the gates.

Voltage Reduction: In portable or hand-held devices such as a cellular phone, compared to 5V operation, a CMOS ckt power reduces by one sixth, in 2.0V opⁿ. The time intervals needed for recharging the battery increases by a factor of six.

Wait, Stop & Cache Disable Instructions: Total power consumption by the s/m while in running, waiting & idle states should be limited. A microcontroller must provide for executing wait & stop instructions for the power down mode. One way to reduce power dissipation is to incorporate into s/w, the wait & stop instructions. Another is to operate the s/m at the lowest voltage levels in the idle state & selecting power-down mode in that state.

Process Deadlines: Meeting the deadline of all processes in the s/m while keeping the memory, power dissipation, processor clock rate & cost at min. is a challenge.

Flexibility & Upgrade ability: in design while keeping the cost minimum & without any engg. cost is a challenge. It provides diff. & advanced versions of a pdt to be introduced in the market.

Reliability: Designing a reliable product by appropriate design, testing & thorough verification is a challenge.

Embedded b/w Components:

Power Source: Most s/ms have a power supply of their own. The Network Interface Card & Graphic Accelerator are eg: of embedded s/m that donot have their own power supply. The supply has a specific range of voltages: $5.0V \pm 0.25V$, $3.3V \pm 0.3V$; $2.0V \pm 0.2V$ & $1.5V \pm 0.2V$.

Low voltage Operations:

1. In portable or hand-held devices such as cellular phone.
2. low voltage s/m processors & i/o ckt's generate lesser heat & can be packed into a smaller space.

Clock Oscillator Circuit and Clocking Units:

→ The clock tells the time for executing an instruction. The clock is the basic unit of a s/m. The clock controls the various clocking requests of the CPU, of the s/m timers & the CPU m/c cycles. The m/c cycles are for fetching codes & data from memory & then decoding & executing them at the processor & for transferring the results to memory.

System Timers & Real-time Clocks:

A timer ckt is configured as the s/m-clock, which ticks & generates s/m interrupts periodically. For eg: 60 times in 1s.

A timer ckt is configured as the real time clk that generates s/m interrupts periodically for the schedulers, real-time pgms & for periodic saving of time & date in the s/m. The RTC or s/m timer is also used to obtain s/w ctelled delays & time outs.

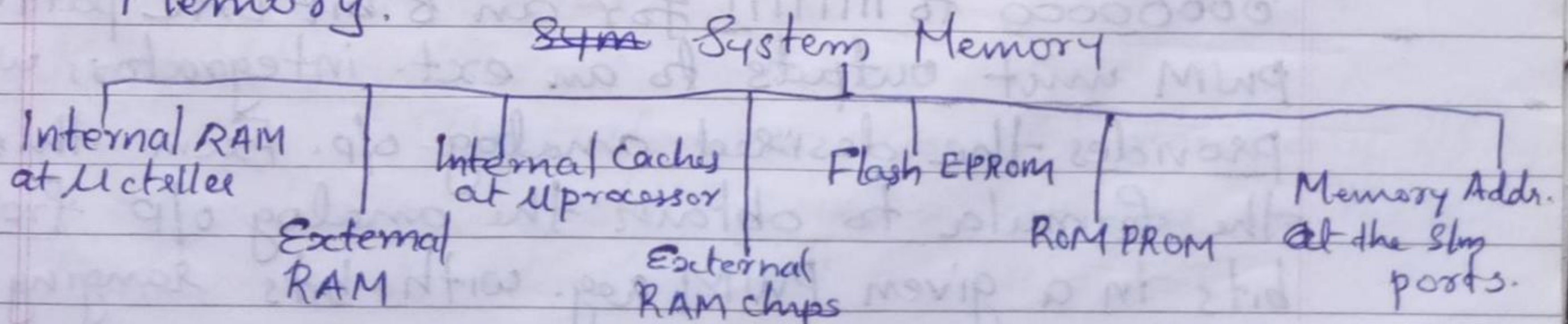
Reset ext circuit, Power-up Reset & Watchdog - Timer Reset:

Reset means that the processor begins the processing of instructions from the starting address. That address is one that is set by default in the processor PC on a power up. A pgm that is reset & runs on a power up can be one of the following: (i) A s/m pgm that executes from the beginning.

(ii) A s/m bootup pgm.

(iii) A s/m initialization.

Memory:



Input, Output & 10 Ports, 10 Buses & 10 Interfaces

→ The μ gets μ ps from touch screen, keys in keypad, or sensors etc.

→ Counter ckt gets μ ps from sensor & transducer ckt.

→ A receiver of μ ps from comm μ ps.

→ Ports receive μ ps from a μ or peripheral.

The μ has O/p ports:

1. LED,
2. Printer
3. Communication μ
4. Alarms, actuators & furnaces.
5. Various motors.

Bus: A μ might have to be connected to a no. of other devices & μ s. A bus consists of a common set of lines to connect multiple devices, μ units & μ s for comm.

DAC using PWM and an ADC

DAC is a ckt that converts digital 8 or 10, 12 bits to analog output. The analog o/p is w.r. to the ref. voltage.

Pulse Width Modulator (PWM): with an integrator ckt is used for the DAC.

→ Pulse width is proportional to the analog o/p needed. PWM μ ps are from 00000000 to 11111111 for an 8 bit DAC pattern. PWM unit outputs to an ext. integrator, which provides the desired analog o/p. From this infⁿ, the formula to obtain the analog o/p from the bits in a given PWM seq. with bits ranging from 00000000 to 11111111 as follows.

Analog o/p $V = K \cdot pw$ where K is constant & pw is the pulse width.

Analog to Digital Converter is a ckt that converts the analog i/p to digital 4, 8, 10, or 12 bits. The analog i/p is applied b/w the positive & -ve pins & is converted w.r. to ref. voltage. When i/p is equal to diff. of ref. +ve & -ve voltages, then all output bits equal 1; when equals -ve ref. voltage, then all o/p bits equal 0.

LCD, LED & Touch Screen Displays.

A s/m requires an i/fing ckt & slw to display the status or msg for a line, for multiline displays, or for flashing displays.

An LCD screen may show up a multi-line display of characters or also show a small graph or icons. A recent innovation in the mobile phone s/m turns the screen blue to indicate an incoming call.

To indicate the ON status of the s/m, there may be an LED that glows. A flashing LED may indicate that a specific task is under completion or running status.

A touchscreen is an i/p as well as o/p device, which can be used to enter a cmd, chosen menu or to give reply. This infⁿ is input by physically touching at a screen posⁿ using a finger or stylus.

Keypad/Key board:

The s/m provides the necessary i/fing & key-debouncing ckt as well as the slw for the s/m to receive i/p from a set of keys.

Interrupt Handler:

A timing device sends a time-out interrupt when a preset time elapses or sends a compare interrupt when the present time equals the preset time. An interrupt-handling mechanism must exist in each s/m to handle interrupts from various processes & for handling mechanisms must exist in each s/m to handle interrupts from simultaneously pending for service.

1. An interrupt may be a h/w s/l that indicates the occurrence of an event.
2. The s/m may prioritize sources & service them accordingly.
3. Certain sources are not maskable & cannot be disabled. Some are assigned the highest priority during processing.
4. The processor's current pgm has to divert to a service routine to complete that task on the occurrence of the interrupt.
5. There is a programmable unit on chip for the interrupt handling mechanism in a u/celler.
6. The OS is expected to ctrl the handling of interrupts & running of routines for the interrupts in a particular application. The s/m give priority to the ISRs over the tasks of an application.

Software Components in an Embedded System

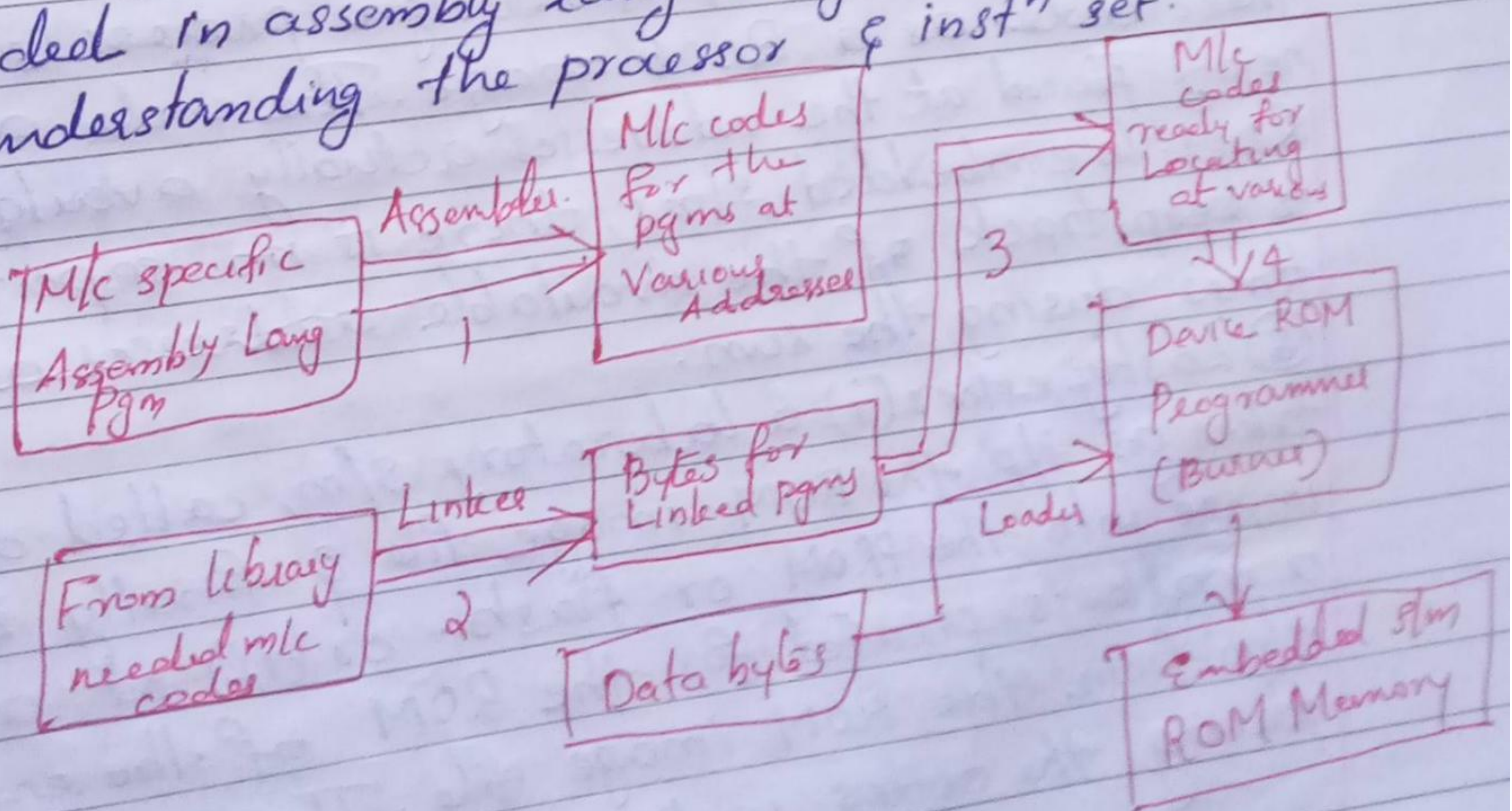
Final Mlc Implementable S/w for a System:

An embedded s/m processor executes s/w that is specific to given applⁿ of that s/m. The instⁿ codes & data in the final phase are placed in the ROM or flash memory for all the tasks that are executed when the s/m runs. The s/w is also called ROM Image.

Coding of S/w in Mlc Codes:

Programmer defines the addresses & the corresponding bytes or bits at each address. For eg: in a transceiver, placing certain mlc code & bits can configure it to transmit at specific Mb per sec. or c/bps, using specific bus & n/wing protocols.

S/w in Processor Specific Assembly Language:
A pgm or a small specific part can be coded in assembly lang using an assembler after understanding the processor & instⁿ set.



Process of converting an assembly lang. pgm into m/c implementable s/w file & then finally obtaining a ROM image file.

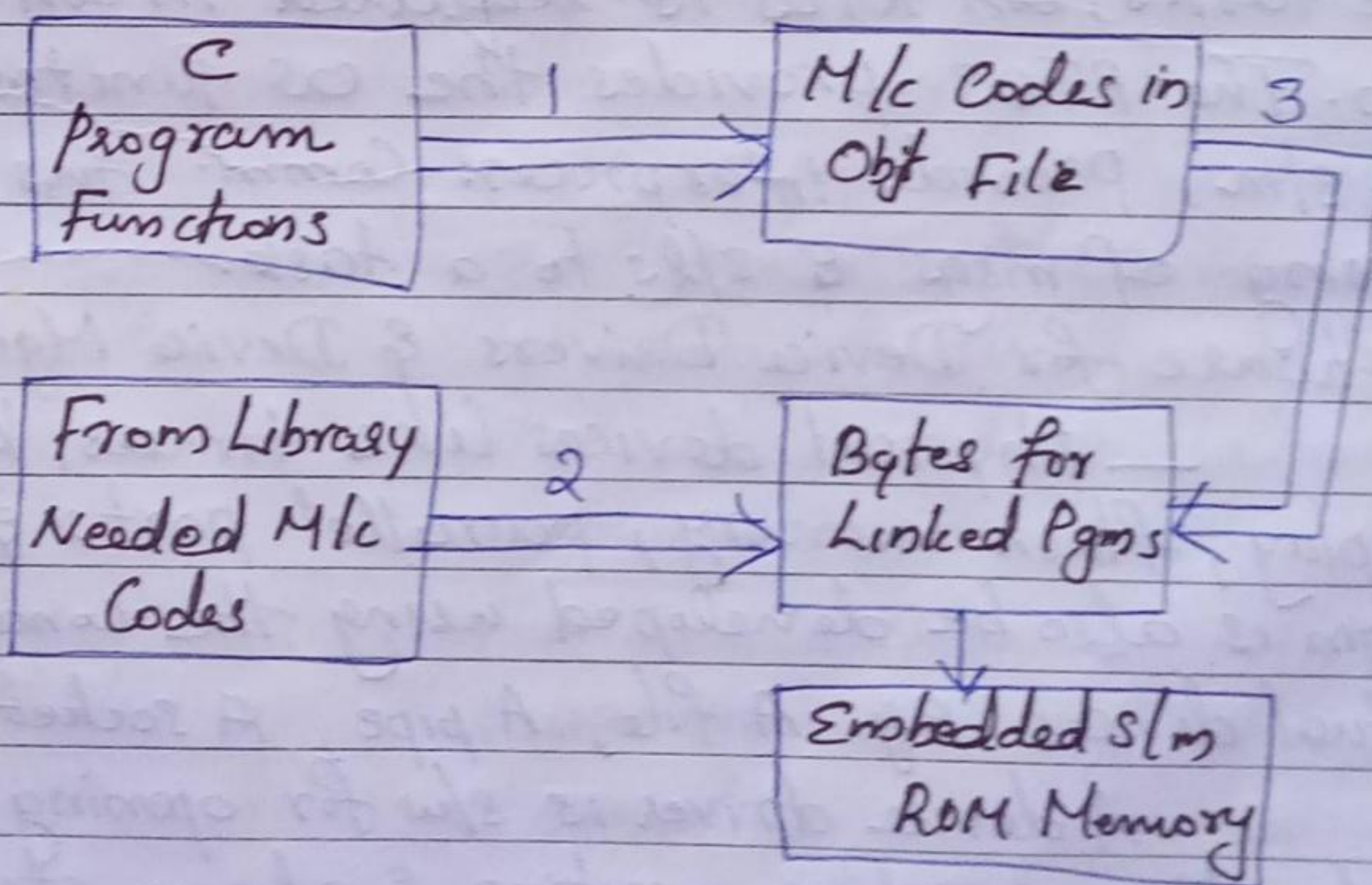
- ① An assembler translates the assembly s/w into the m/c codes using a step called assembling.
- ② In the next step, called linking, a linker links these codes with the other codes required. Linking is necessary b/c of the no. of codes to be linked for the final binary file. The linked file in binary for run on a computer is known as an executable file or simply an '.exe' file. After linking, there has to be reallocation of the sequences of placing the codes before actually placing the codes in memory.
- ③ Next step, the loader pgm performs the task of reallocating the codes after finding the phy. memory addresses available at a given instant. The loader is a part of the O.S & places codes into the memory after reading the '.exe' file.
- ④ The final step of s/m design process is locating these codes as a ROM image. The codes are permanently placed at the addresses actually available in the ROM. In embedded s/ms, there is no separate pgm to keep track of the available addresses at diff. times during the run.
- ⑤ Lastly either (i) a laboratory s/m, called device programmer, takes as i/p the ROM image file & finally burns the image into the PROM or flash. or (ii) at a foundry a mask is created for the ROM of the embedded s/m from the ROM image file. The process of placing the codes in PROM or flash is called burning.

Software in High Level Language:

Preprocessor Commands
Main Function
Interrupt Service Routines
Tasks 1...N
Kernel & Scheduler
standard Library fns including N/w protocol fns for sending stack & Receiving stack

The diff. pgm layers
in the embedded
sl/w in C.

- shows the process of converting a C pgm into the ROM image file. A compiler generates the obj code. It assembles the codes acc. to the processor instⁿ set & other specificⁿs. The C compiler for embedded sl/w, must, as a final step of compilation, use a code - optimizer that optimizes the codes before linking. After compilation, the linker links the obj codes with other needed codes.



process of converting a C pgm into the file for
ROM image.

Programs Models for S/w Designing:

These models that are employed during the design processes of the embedded S/w are as follows:

1. Sequential Program Model
2. Object Oriented Program Model
3. Control & Dataflow Graph
4. Finite state M/c for data paths
5. Multithreaded for concurrent processing.

→ UML.

Software for Concurrent Processing & Scheduling of Multiple Tasks & ISRs using an RTOS.

The multiple tasks are processed most often by the OS not sequentially but concurrently.

Concurrent processing tasks can be interrupted for running the ISRs, & a higher priority task preempts the running of lower priority tasks.

OS S/w have scheduling fns for all processes in the S/m. Since the running of tasks and ISRs may have real time constraints & deadlines for finishing the tasks, an RTOS is required in an embedded S/m. The RTOS provides the OS functions for coding the S/m, provides interprocess commn fns & also the passing of msgs & s/l's to a task.

Software for Device Drivers & Device Mgmt in an O.S.

— physical devices like timers, keyboards, display, flash memory, parallel ports & n/w cards. A pgm is also developed using the concept of virtual devices. eg: A file, A pipe, A socket & RAM disk.

A device driver is S/w for opening, connecting or binding, reading, writing & closing & other actions of the device. It also fns for device open, connect, bind,

listen, read or write or close.

A driver ctels 3 fns: i) Initializing, which is activated by plaving appropriate bits at the ctel reg. or word.
ii) Calling an ISR on interrupt or on setting a status flag in the status reg & running the ISR.
iii) ~~Restarting~~ Reseting the status flag after an interrupt service.

A device driver accesses a parallel or serial port, keyboard, mice, disk, slw, display, file, pipe & socket at speufic addresses.

Software Tools for Designing an Embedded S/m

- Editor : For writing codes or assembly mnemonics
- Interpreter : line by line translation to exe codes
- Compiler : source code to object code.
- Assembler : translating assembly mnemonics to binary opcodes.
- Cross Assembler : For converting objt codes or executable codes for a processor to other codes for another processor. It assembles the assembly codes of the target processor as assembly codes of processor of the PC used in s/m development.
- Simulator : To simulate all fns of an embedded s/m ckt including that or additional memory & peripherals.
- Source code engg. s/w : For source code comprehension, navigation & browsing, editing, debugging, configuring & compiling.

Software Tools Required in Exemplary Cases:

→ Editor, Interpreter, Compiler, Assembler, Loader etc. (Table 1.3 27)

Examples of Embedded s/m

- Washing m/c
- Multitasking toys
- 3. Robotics s/m
- 4. Keyboard controller.

MODULE II

→ Hardware Software Co-Design & Program Modelling

→ Fundamental Issues

→ Computational Models

— Data Flow Graph 6.2 ✓✓

— Control Data Flow Graph 6.2 ✓

— State Machine 6.3 ✓

— Sequential Model ✓✓

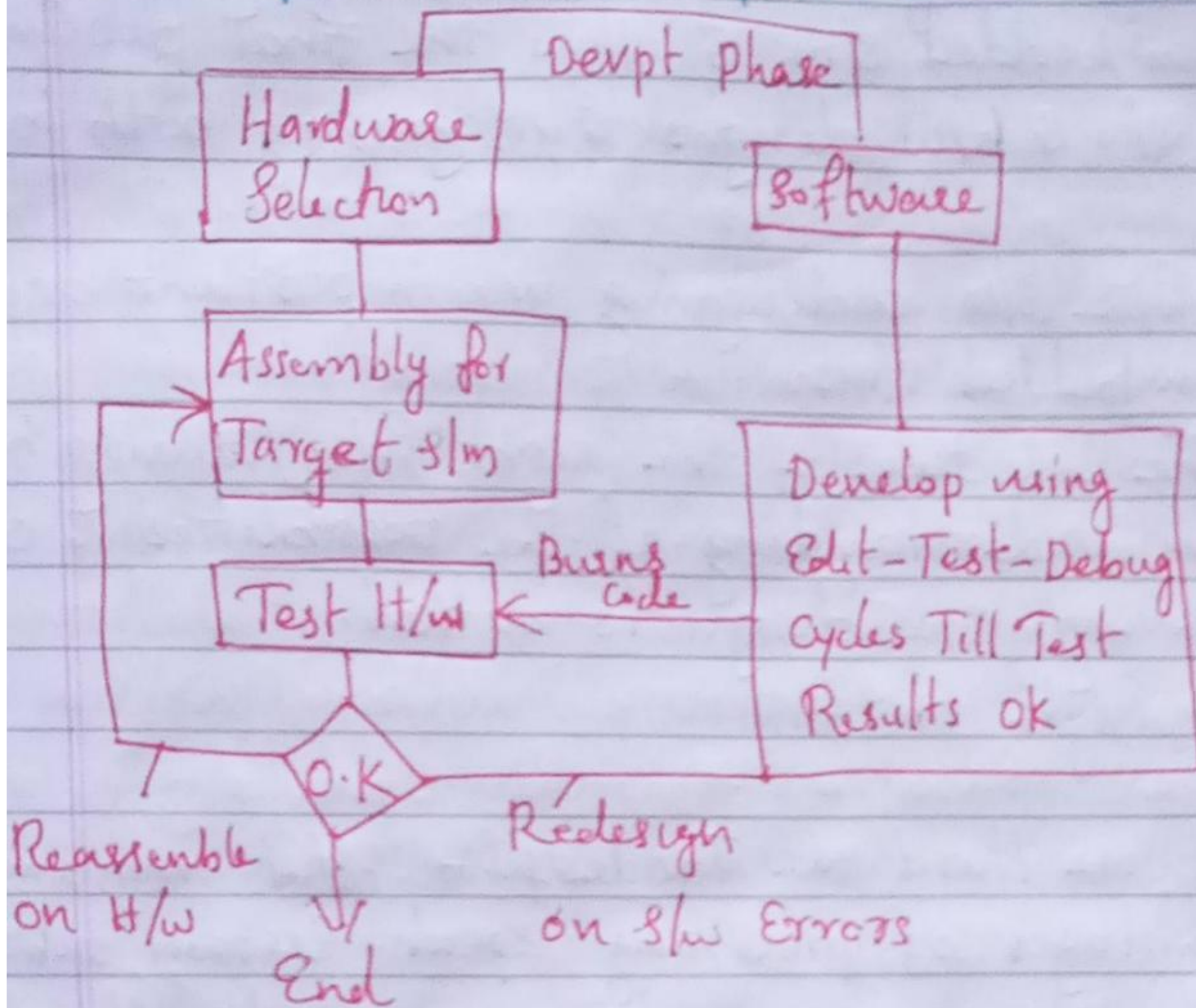
— Concurrent Model - 6.2 eg ✓

— Object oriented Model eg: 6.3 ✓

— UML 6.5

Hardware Software Co-Design & Program Modelling.

Development Process & Hardware - Software.



Edit-test-Debug Cycle

— Main Approaches:

1. An IDE or prototype tool
2. A simulator without any h/w
3. Processor only at the target s/m & uses an in-b/w ICE
4. Target s/m at last stage.

Issues in Hardware - Software Co-Design:

There are 2 approaches for the embedded s/m design:

1. The SDLC ends and the life cycle for process of integrating the s/w into h/w begin at the time when a s/m is designed.

(2) Both cycles concurrently proceed when co-designing a time critical sophisticated s/m.

The final design, when implemented, gives the targetted embedded s/m & thus the final product.

- s/w & h/w designs & integrating both into a s/m &

- h/w s/w codesigning are important aspects of designing embedded s/ms.

The selection of the right h/w during h/w design & an understanding of the possibilities & capabilities of h/w during s/w design is critical.

1. Choosing Right Platform:

- Software Hardware Tradeoff:- H/w implementations provide advantages of processing speed. Certain sub/sms in h/w (cteller, IO memory access ckt, real time clk, s/m clock, pulse width modulation, timer & serial commⁿ) are implemented by s/w.

H/w implementation provides the follo other advtges:

- 1) Reduced memory for the programs.
- 2) Reduced no. of chips but at an increased cost
- 3) Simple coding for the device drivers.
- 4) Internally Embedded codes.

S/w implementation provides the follo. adv:

- 1) Easier to change when new h/w versions become available.
- 2) Programmability for complex operations
- 3) Faster development time
- 4) Modularity & portability
- 5) Use of standard engg.
- 6) Faster speed of opⁿ of complex fns
- 7) Less cost for simple s/ms.

Choosing a right Platform:

A platform consists of a no. of units. Processor, ASIP, Multiple processors, System on Chip, Memory, H/w units of slm, Buses, slw language, RTOS. Code generation tools.

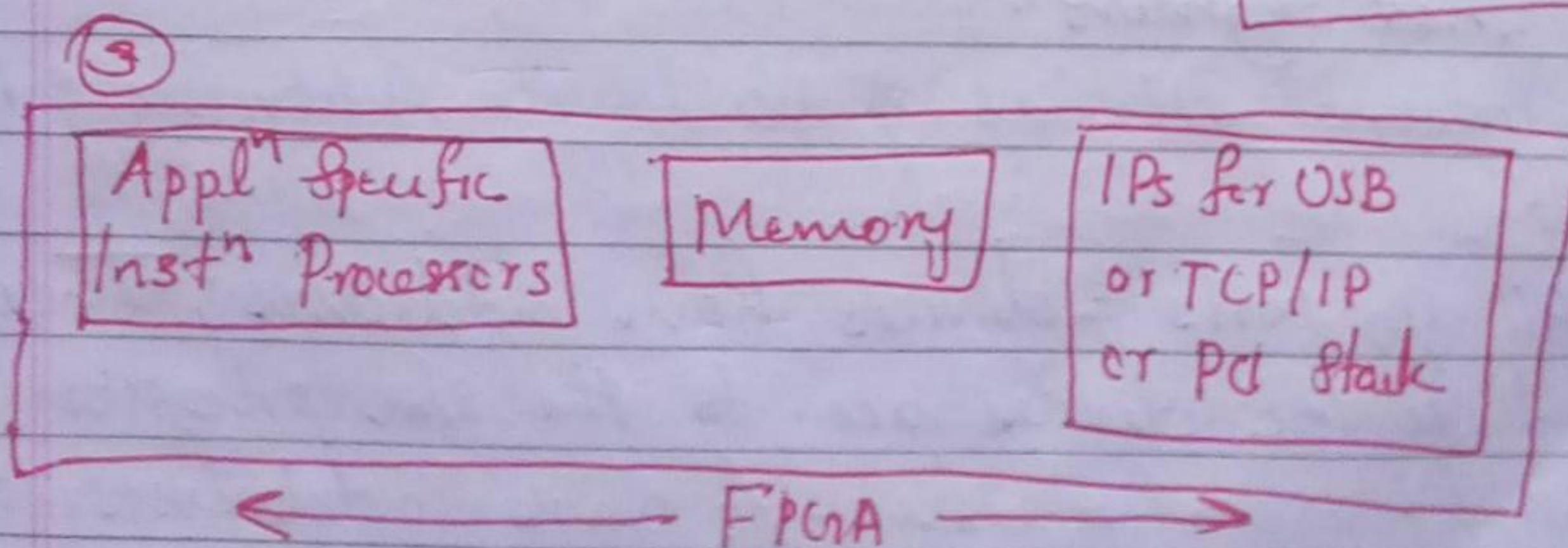
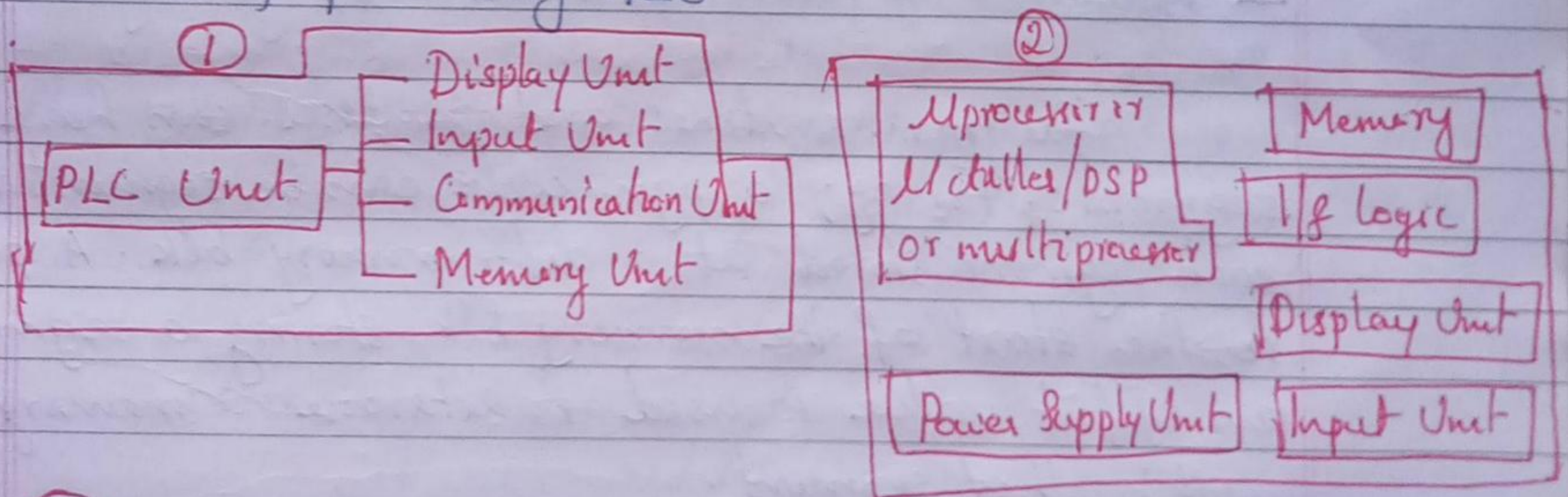
Embedded Systems Processors Choice:

1) Processor-less Systems: PLC can be used in place of processor. PLC can be used for the clothes-in clothes-out type slm. A PLC has very low opⁿ speed. It also has a very low computational ability, very strong i/fing capability with its multiple i/ps & o/ps. Automatic Chocolate-vending m/c can be another application of PLC.

2) Systems with Microprocessor or Microcontroller or DSP:

3) Systems with Single-Purpose Processors in VLSI or FPGAs

The processing of slms by using IPs embedded into VLSI instead of processing ALU.



④ Factors & Needed features taken into Consideration:

Memory & Processor - Sensitive S/W:

Processor Sensitive: I/O instructions are processor sensitive. A processor may be having fixed pt. ALU only. Floating pt. ops when needed are handled differently than in a processor with floating pt. operations. A processor may not provide for exⁿ of Single Instⁿ Multiple Data (SIMD) & VLIW (very Large Instⁿ Word) instructions.

Memory Sensitive: Eg: video processing & real time video processed/sec will depend on memory available as well as the processor performance. If a large memory is available, then higher resolution pictures can be processed.

— Memory address of I/O device registers, buffers, ctrl registers & vector addresses for the interrupt sources or source groups are prefixed in a μ controller.

Allocation of Addresses to Memory, Pgm Segments & Devices:

Program routines and processes can have diff segments. For eg; a pgm code can be segmented & each segment stored at a diff. memory blk. A ptr points to the start of the memory blk storing a segment & an offset value is used to determine a memory addr. within that segment.

Device, Internal Devices & I/O Device Addresses & Device Drivers:

All I/O ports & devices have addresses; These are allocated to the devices acc. to the s/m processor & the s/m h/w configurations. I/O device addresses are considered as part of the memory addresses by certain

processors. A device has an address, allocated to the following:

1. Device data registers
2. Device ctrl register: It saves ctrl bits & may save config bits also
3. Device status registers: It saves flag bits as device status

Porting Issues of OS in an Embedded S/m:

- I/O Instructions: A port instruction data type may be diff. on the diff. platforms.

- Interrupt servicing routines: OS supports these diff. on diff. platforms

- Data types: Appropriate APIs for datatypes.

- Interface-specific data types: Eg: N/w I/f Card supports 32 bit unsigned integers.

- Byte Order, Data Alignment, Linked lists, Mem. page size, Time Intervals.

Performance & Performance Accelerators:

Performance Modelling:

1. S/m performance Index: can be defined as the ability to meet reqd. fns & specifications while using the min amount of resources of memory, power dissipation & devices & min. design efforts & optimum utilization of each resource.

2. Multiprocessor S/m Performance:

is measured by i) an optimized partition of the pgm into the tasks or set of inst's b/w the various processors ii) an optimized scheduling of inst's & data over the available processor times & resources.

3. MIPS, MFLOPS & DMIPS as Performance Indices: One performance design metric is how long a s/m takes to execute the desired s/m fns.

MIPS - Million Instⁿs Per Second

MFLOPs - Million Floating Pt Instⁿs per second

DMIPS - Dheystone million Instⁿs per second.

4) Performance Metrics: Buffer Requirement, I/O Performance & Bandwidth Requirement. \downarrow Accelerates performance

5) Real Time Pgm Performance:

1) ratio of sum of interrupt latencies as a fn of the exⁿ times

2) CPU Load 3) worst case exⁿ w.r. to the mean exⁿ time.

Performance Accelerators:

- Conversion of CDFCs to DFCs.

- Reusing the used arrays & memory, appropriate variable selection, appropriate memory allocation deallocation strategy.

- Using stacks as datastructure when feasible instead of queue & using queue instead of list

- Computing slowest cycle first & examining the possibilities of its speed up.

Computational Models:

→ Polling for events model: There is polling in cyclic loop for the events, state variables, msgs & s/s using the switch-case stmts.

→ Sequential Program Model: sequential programming model in which there are sequential multiple fn calls within a function.

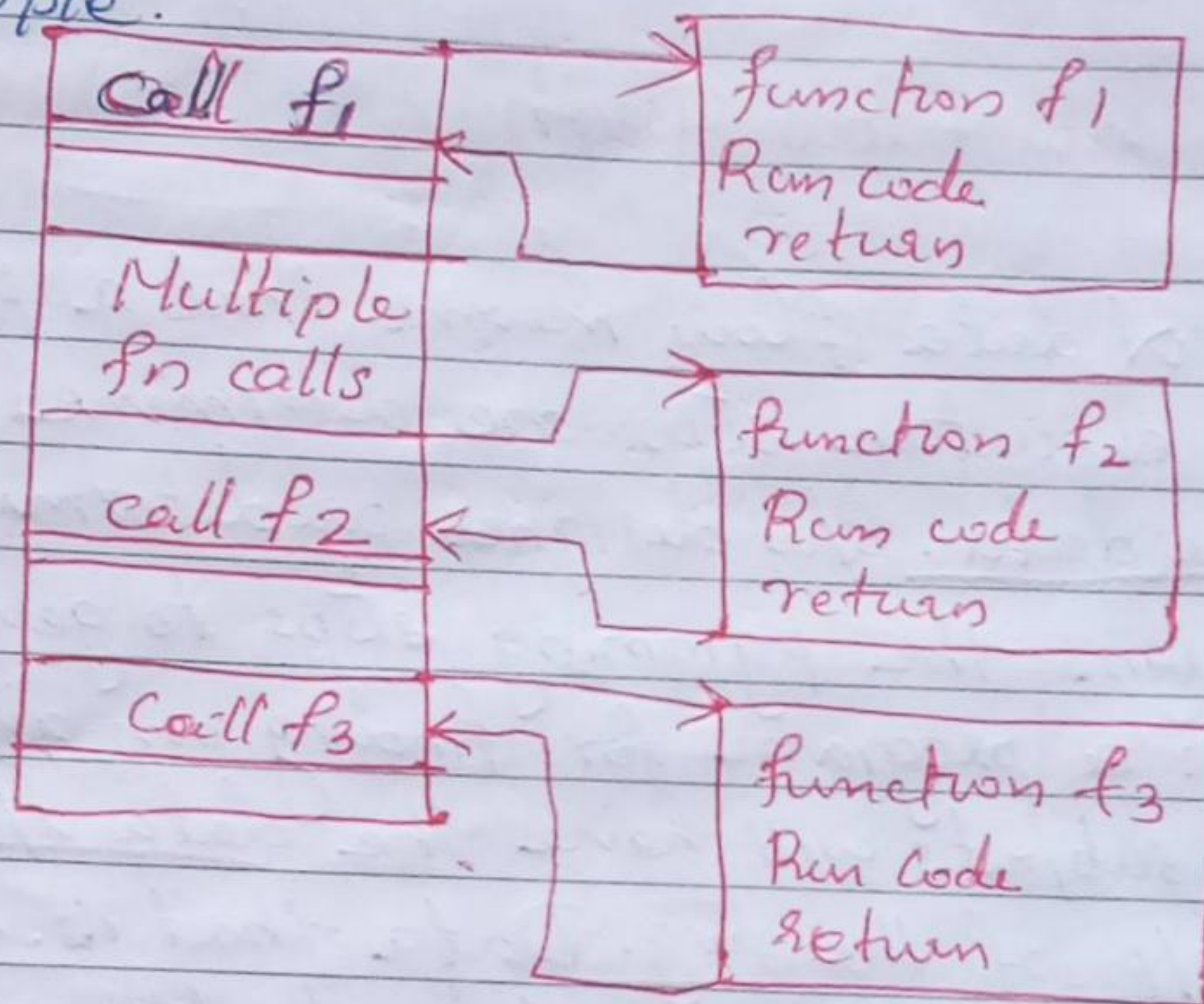
→ Data Flow Model: used for modelling the data paths, and. pgm flows of s/w. A pgm is modeled as handling the ip data streams & creating op data streams

→ State Machine Model: A programming model is that there are diff. states & the model considers a s/m as a m/c., which is producing the states. Program sequentially polled for the screen state & menu choice selected by the user.

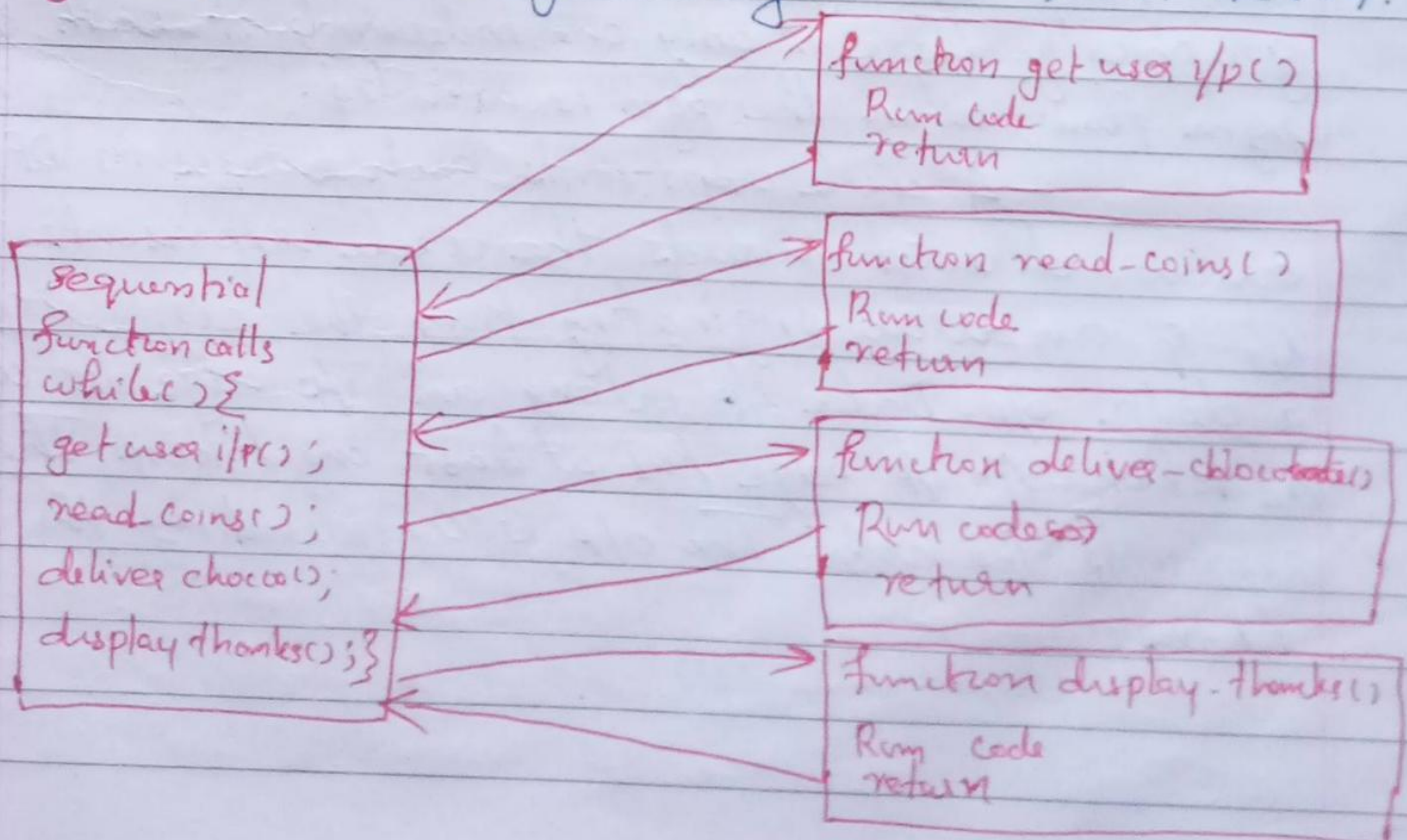
→ Concurrent processes & Interprocess Communication model:

Sequential Programs Model:

Example.



Eg: Sequential Programming model of an ACVM:



- Run function get-user-input() for obtaining i/p for the choice of chocolate from the child.
- Run function read-coins() for reading the coins inserted into the ACVM for the cost of chocolate.
- Run function deliver-chocolate() for delivering chocolate.
- Run function display-thanks() for displaying "Collect the nice chocolate, visit Again".

Data Flow Graph

A data flow means that a program flow & all pgm execution steps are determined specifically only by the data. S/w designer predetermines the data i/p & designs the programming steps to generate the data o/p. For eg: a program for finding an avg of the grades in various subjects will have the data i/p of the grades & data o/p of the avg. Data that is input after the op's in the pgm becomes data that is o/p after a data flow. A diagram called ~~DFG~~ DFG represents this graphically. There is only one independent path for the pgm flow when the pgm is executed.

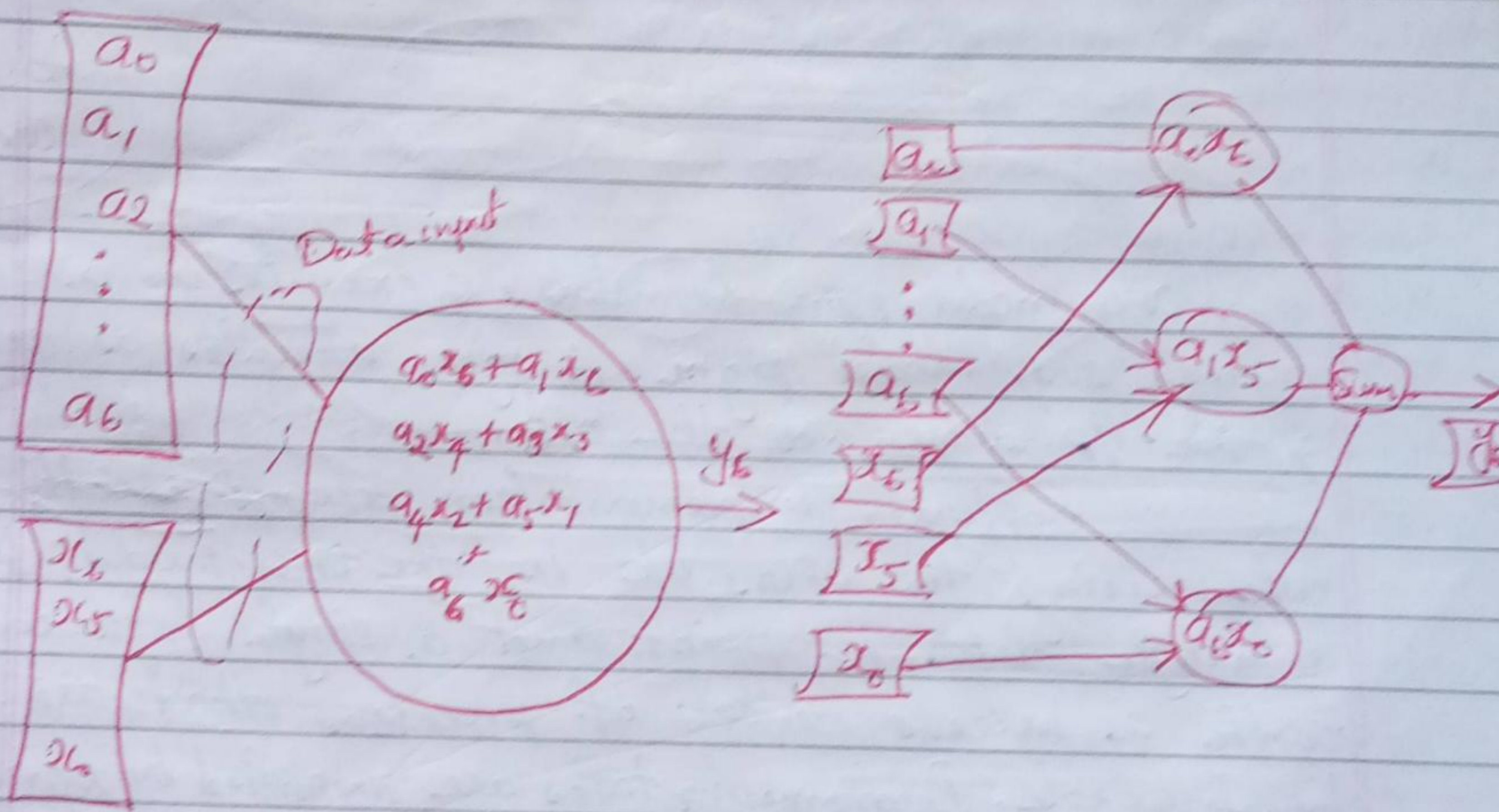
- A circle represents each process in DFG. An arrow directed towards the circle represents the data i/p & an arrow originating from the circle represents a data o/p. Data i/p along an i/p edge is considered as token. An i/p edge has at least one token. The circle represents the node. The o/p is considered by the outgoing tokens;

Eg: An n th filtered output sequence, $y_n = \sum(a_i \cdot x_{n-i})$ where the sum is made for $i=0, 1, 2, \dots, N-1$.

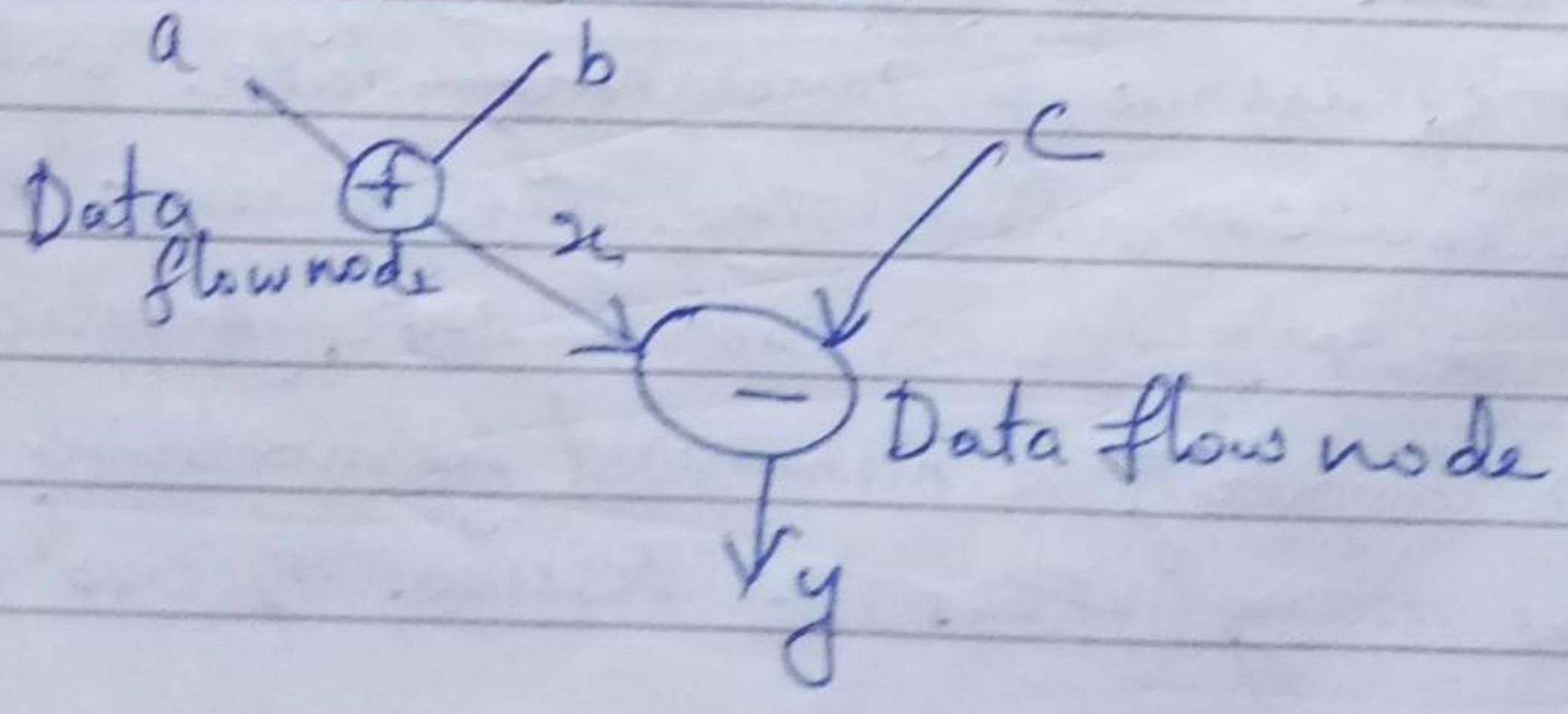
Following are the pts. notable for the process of calculating $y_6 = a_0 \cdot x_6 + a_1 \cdot x_5 + a_2 \cdot x_4 + a_3 \cdot x_3 + a_4 \cdot x_2 + a_5 \cdot x_1 + a_6 \cdot x_0$

1. There is one i/p pt. to the process - represented by the circle for calculating y_6 .
2. There is one o/p pt. for y_6 .
3. There is only one memory address & variable for each coefficient & each filter y_6 . There is only one value of each of the 312 i/ps for x & there is only one value of each of the coeff. a .

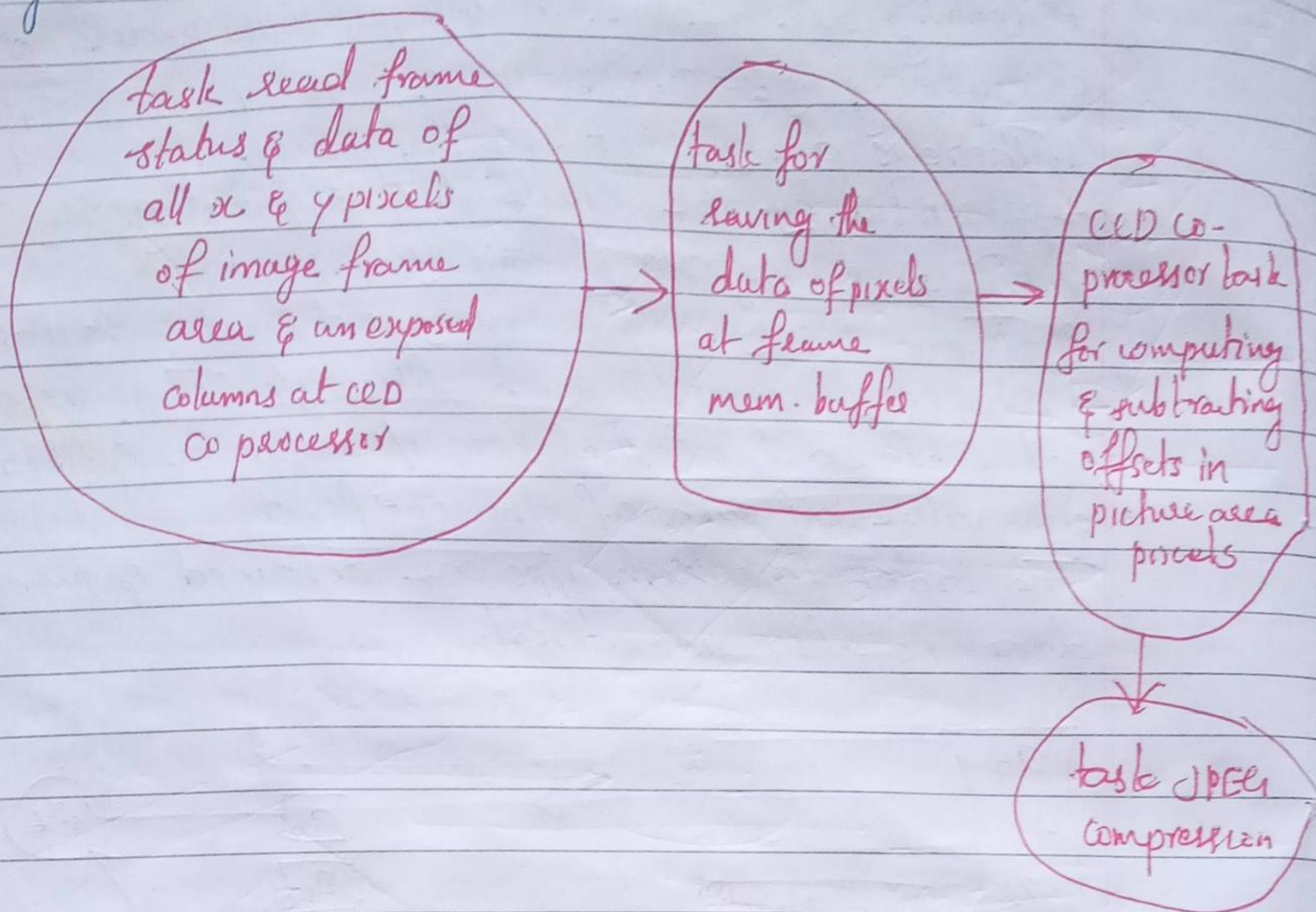
The order in which i/ps are obtained & summation is done is also immaterial.



Eg: $y = x - c$ where $x = a + b$.



DFG model for pgm for saving a picture in a digital camera.



DFG model pgm translates & executes as a single process seq. model pgm. A pgm executes as per i/p & the i/p determines the o/p.

Software implementation becomes greatly simplified when using the DFGs b/c in the DFG model, there is a single data-in point and a single data-out point, with a process or set of processes that are represented by a circle. Programming tasks are simplified by representing the code for each process by a circle, using the data input from incoming arrow & generating data output along outgoing arrows. When the assignment to an i/p is fixed in a DFG, it is called ADFG (Acyclic DFG).

Programming complexity is minimized by modeling a pgm in terms of as many DFGs as possible & use of as many

ADDF ADFUs as possible.

Control DFC Model

A ctrl flow means that specifically only the pgm determines all pgm exⁿ steps & flow of the pgm. The slw designer pgms & predetermines these steps. A process may have the stmts that ctrl the i/ps or o/ps. It may have loops or condition stmts in b/w. Data that is i/p generate the data o/p after a ctrl data flow as per the ctrlg conditions. Output depends on the ctrl stmts for various decisions in a process. A condⁿ is a diagram, which graphically represents the condⁿs & the pgm flow along a condition dependent path.

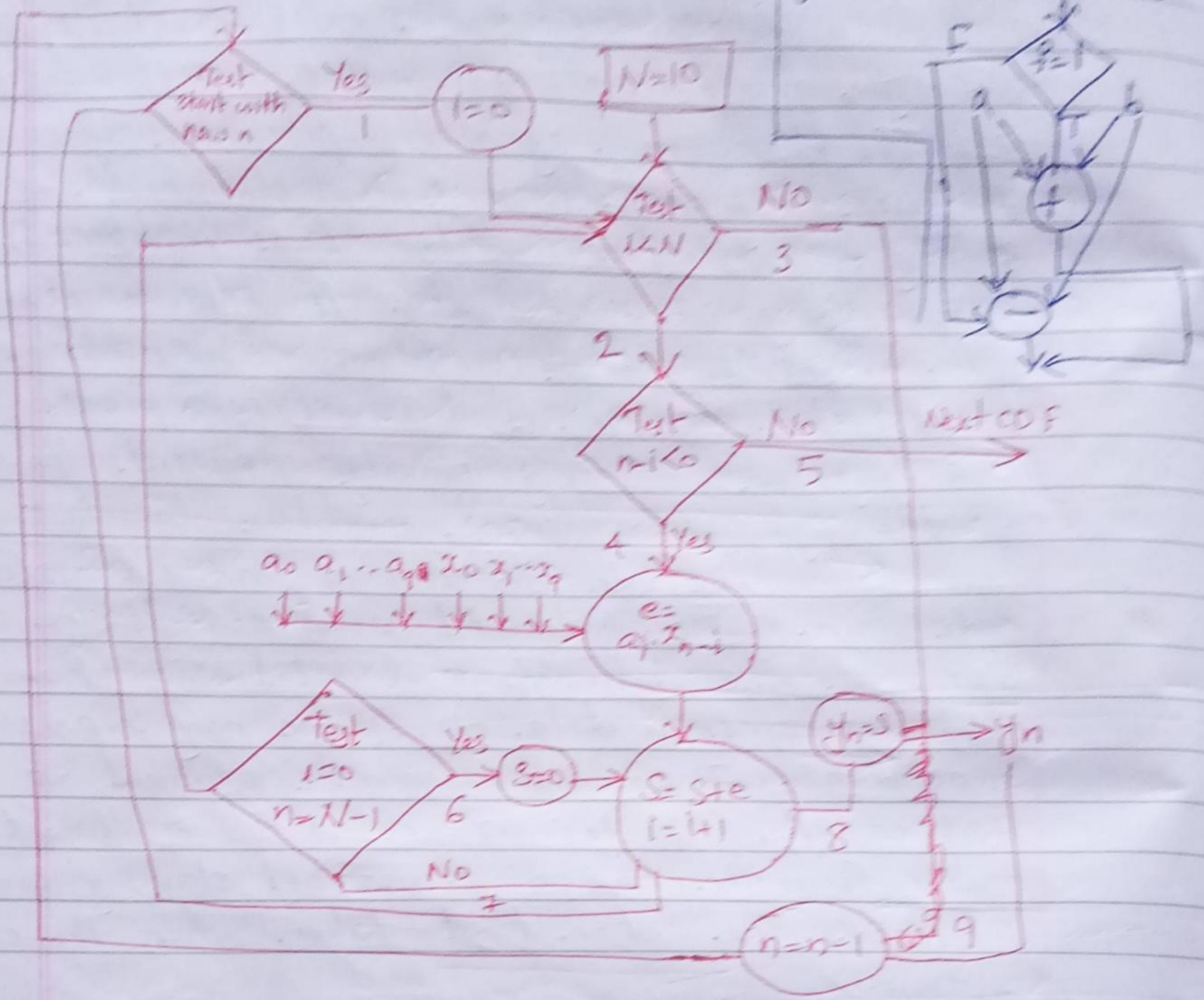
A circle also represents each process (called node) in a CDFU. A directed arrow towards the circle represents the data i/p & directed arrow from the circle represents a data o/p. A box may represent a condition. A condition can be marked at the start of the directed arc or arrow. A directed arrow from the box or a marked starting condition determines the action to be taken when the condⁿ is true.

Example: The ctrlg i/p nodes by the test condⁿ specifying boxes, & the data i/ps to a CDFU for an filter with 10 i/ps & 10 coefficients; $y_n = \sum(a_i \cdot x_{n-i})$. Following are the pts. notable for the process of calculating y_n . There is one i/p pt. to the process represented by the circle for calculating y_n .

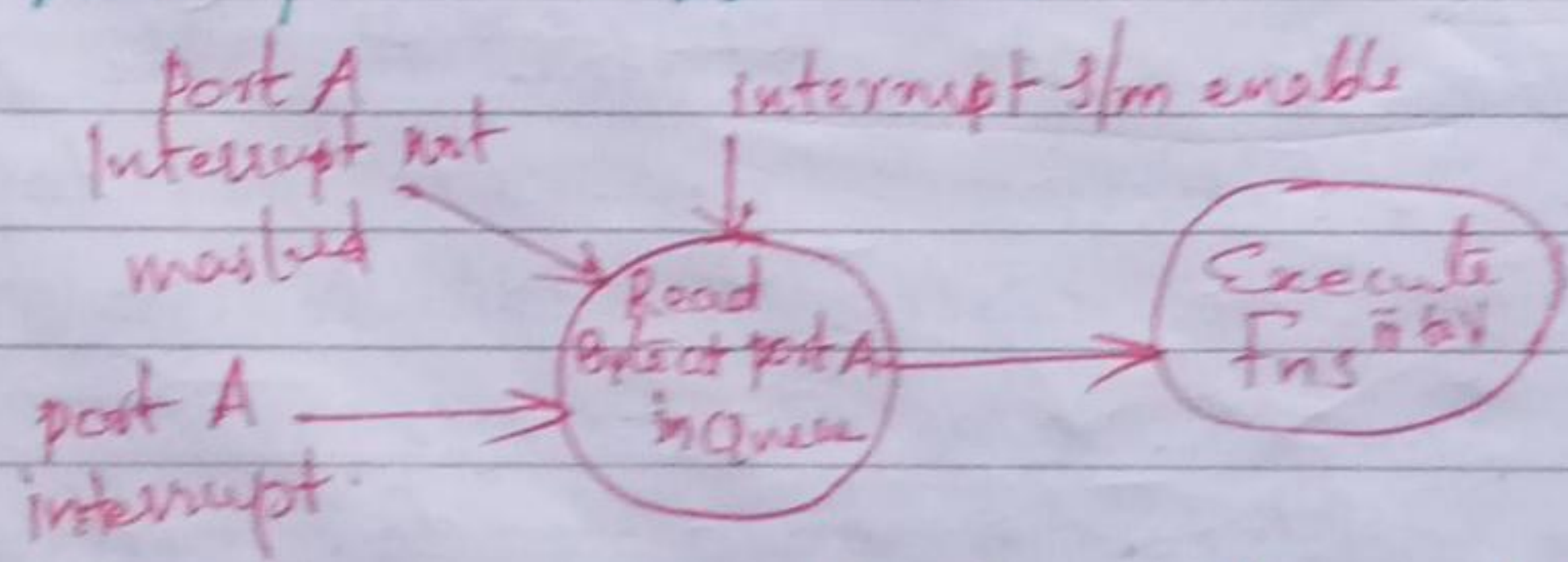
1. There is one o/p pt. for y_n . There is only one memory address & variable for each coefficient & each filter i/p. These are the variables, i, n, s &

which take multiple values during the prog flow.
 Q. The order in which i/p's are obtained & summation is done does matter.

eg: If flag, $x = 0$ then...



Data i/p's & controlling i/p nodes shown by test boxes in a ctrl DFG for a finite impulse Response (FIR) filter with 10 i/p's & 10 coefficients



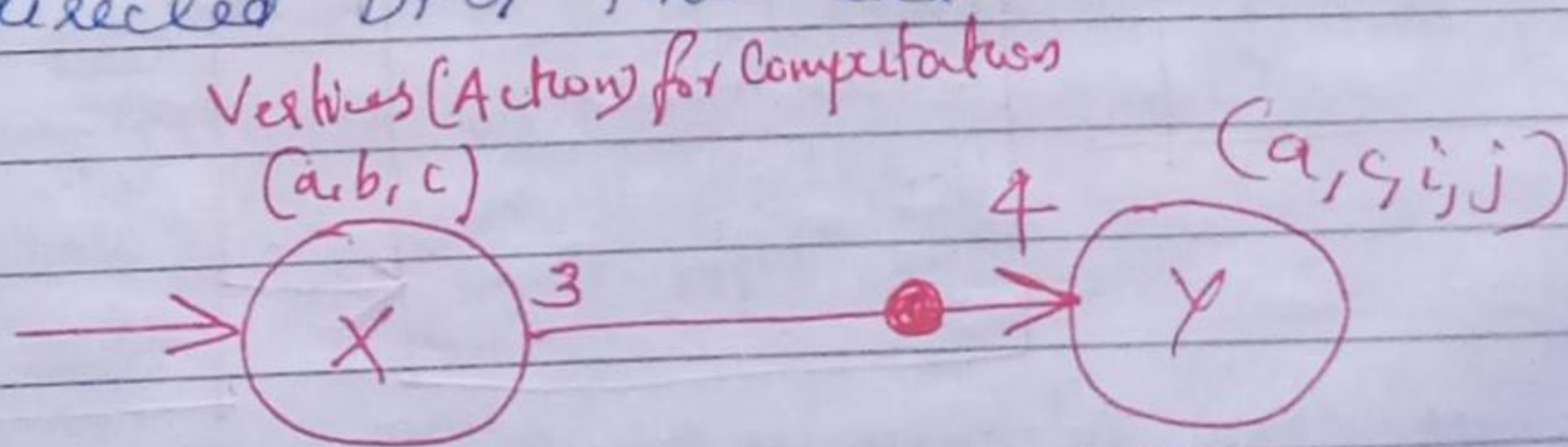
Controlling i/p conditions marked in the A out B pgms.

→ Synchronous Data Flow Graph Model (SDFG)

When there are no. of tokens required for a computation to generate more tokens in a single firing, the data flow is said to be synchronous. The SDFG model is as follows. Let an arc represent a buffer in physical memory. The arc can contain one or more initial tokens with the delays. A token doesn't fire the computations at a vertex till it is received at the vertex.

Vertices (circles) in this graph are called the actors. Actors do the computations. An actor also represents a complete DFG within itself. An edge b/w the vertices represents a queue of output values from one vertex & a queue of i/p values to another vertex. Edges carry values from one actor to another.

Let X & Y be 2 sets of instructions that once started, do not need any further i/ps from any source during the computations. Let X generate the o/p values $a, b, \& c$. Let Y get the i/p values a, c, i & j and let i have a delay. The no. of i/ps to Y need not equal the no. of o/ps from X . Y gets additional i/ps and doesn't need all the o/ps from X . These computations & data are now modelled by a directed DFG that exist from X to Y .



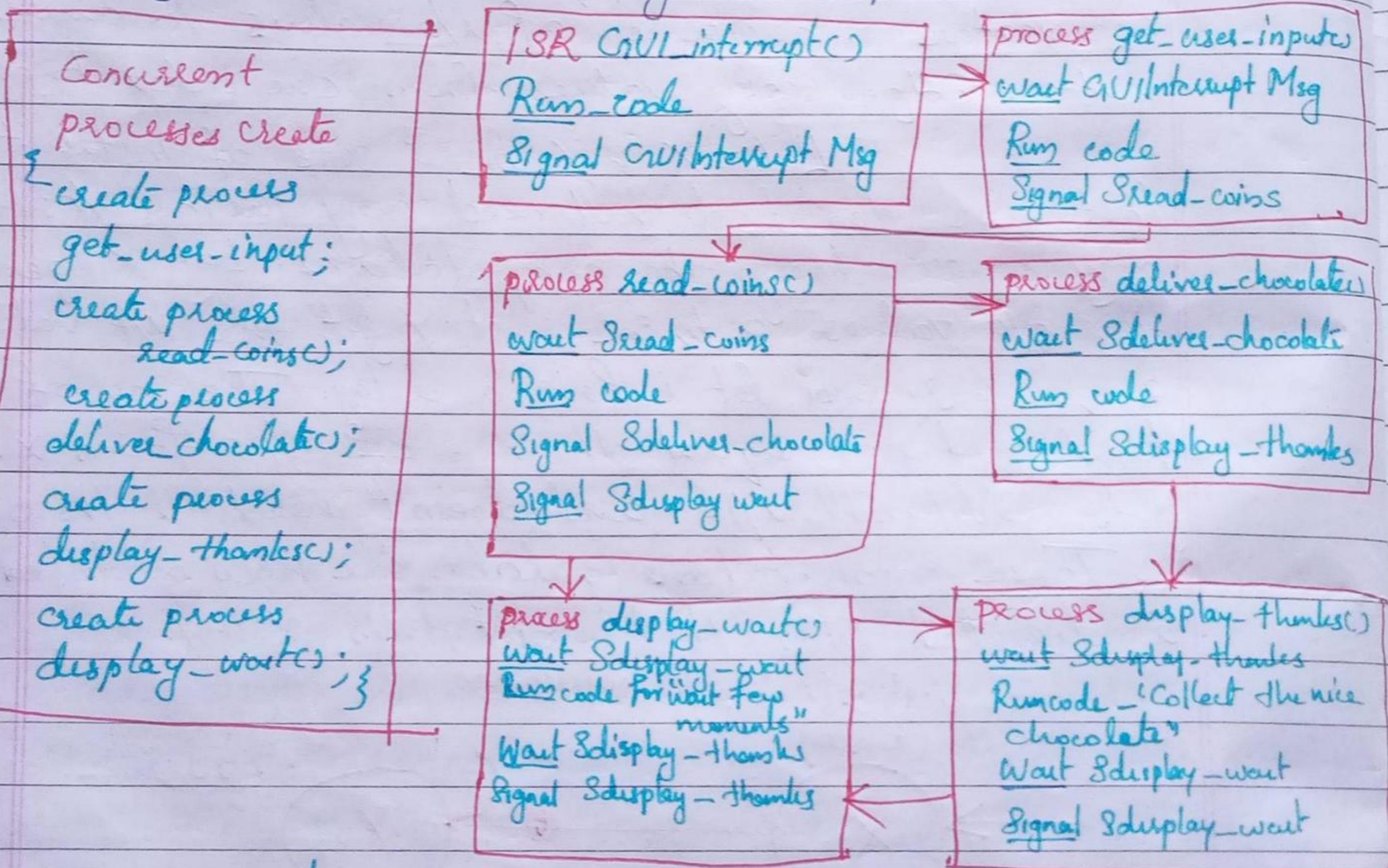
The no. of outputs & i/ps are labelled near the arc origin & arc end. Fig shows actors (vertices, which does the computations on firing) and arcs in a directed graph b/w X & Y . o/ps: $a, b \& c$ & i/ps: $a, c, i \& j$. The 'i' is with a delay (dot).

An SDFC model is like a DFC, but also models the delays as well as the no. of i/ps & o/ps. The edges directed to the circle can be assumed to have a phy. mem. buffer & till the buffer has the data, the computations don't fire.

Concurrent Model

A programming model is that there are several concurrent tasks & each task has the sequential codes in infinite loop. A task sends a msg or s/l for another task. A task, which gets a msg or s/l, runs & the remaining tasks remain in the blocked state.

Eg: Concurrent running of the processes in ACVM.



Arrows show inter-process communications.

The pgm consists of follo. processes, which can run concurrently.

1. Process `get-user-i/p()` for obtaining `i/p` for the choice of chocolate from child & `s/l` to process `read-coins` start.
2. Process `read-coins()` wait for `s/l` `get-user-input()` & start reading on `s/l` from for reading the coins inserted in the ACVM for the cost of chocolate. Post a `s/l` to process `deliver-chocolate` to start & also post a `s/l` to process `display-wait()` to start.
3. Process `deliver-chocolate()` wait for `s/l` from `read-coins()` & starts delivering the chocolate & post a `s/l` to `display-thanks()` to start.
4. Process `display-wait()` waits for `s/l` from `read-coins()` & starts displaying "wait few moments" & then wait for `s/l` for `display-thanks()`.
5. Process `display-thanks()` waits for `s/l` from `deliver-chocolate()` and from `display-wait()` & starts displaying "Collect the nice chocolate, visit Again!"

Object Oriented Programming Model:

Main features:

- a. When there is a need for reusability of the defined objt or set of objts that are common within a program or b/w many applications; when there is a need for abstraction & when by defining objts by inheritance & polymorphs, new objts can be created. There is data encapsulation within an objt.
- b. An objt is characterized by its identity, by its state, & by its behaviour (op^s & methods, fields & attributes)
- c) Defining the logically related groups makes a class. Class defines the state & behaviour. It has internal user level fields for its state & behaviour. It defines the methods of processing the fields.

d) Objects are created from the instances of a class. A class can thus create many objects by copying the group & making it final. Each object is functional. Each object can interact with other objects to process states as per the defined behaviour -

Example:

Classes & objects:

1. Class CUI for graphic-user interaction. It has 2 methods display-menu() & get-user-ipp() & for obtaining I/P for the choice of chocolate from the child. It has method set-choice() to set the choice selected.
2. class Read-coins() for reading coins inserted. It has a method read() to read one, 2 & 5 rupee coins from 3 ports & method sum() for summing the total coins.
3. class Deliver-chocolate. It has methods get-choice() to get choice & deliver() for delivering the chocolate.
4. Class MsgDisplay. It has methods display-wait() & display-thanks() for displaying wait & thanks msgs.

Class CUI is used to create CUI objects as multiple instances of CUI. Class MsgDisplay is used to create msg display objects as multiple instance of wait & thanks msgs. Class MsgDisplay can be interfaced to an interface screen-size(), which has an abstract method screen-size(). Extending class MsgDisplay can specify a new class MsgTime-Display. Extended class MsgTime-Display inherits all attributes & methods of class MsgDisplay.

paste: fig: 6.3

UML Modelling

UML is a unified modeling lang. for any general s/m for which objt oriented analysis & design. are feasible & which can be abstracted by models.

Diagrams: 6 Basic UML elmts:

* class * package, stereotype, object, anonymous objt and state.

Table 6.2, 6.3. Fig: 6.17, 6.18.

A conceptual design modelling can follow the UML approach. It can use the user, object, sequence, state, class & activity diagrams. UML allows the SpecCharts & Statecharts:

SpecCharts is another lang. for specifications & charts. It allows state m/c's to use seq. pgms to model the state actions. StateChart is a lang. for implementing the activity diagram, FSM states & state transitions, concurrency, synchronization, timing & behavioral hierarchy.

State M/c Pgmning Models

A state m/c is a model in which it is assumed that there are states & state transitions fns, which produce the states. A state transition fn. is a fn which changes a state to its next state.

Eg:

- Telephone s/m idle, receiving a ring, dialing, connected & exchanging msgs. There are 5 finite no. of states.
- Consider states of a timer in running state.

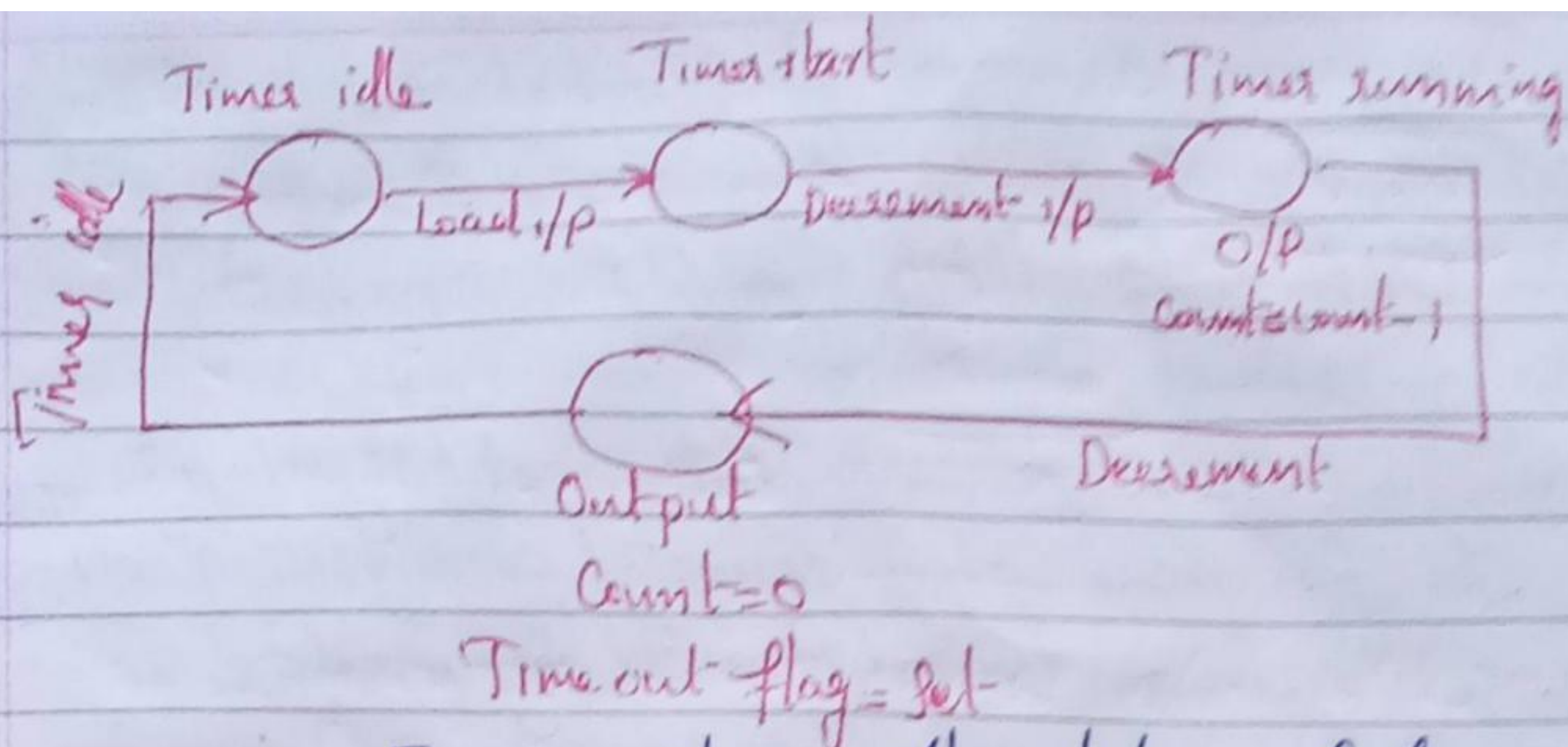


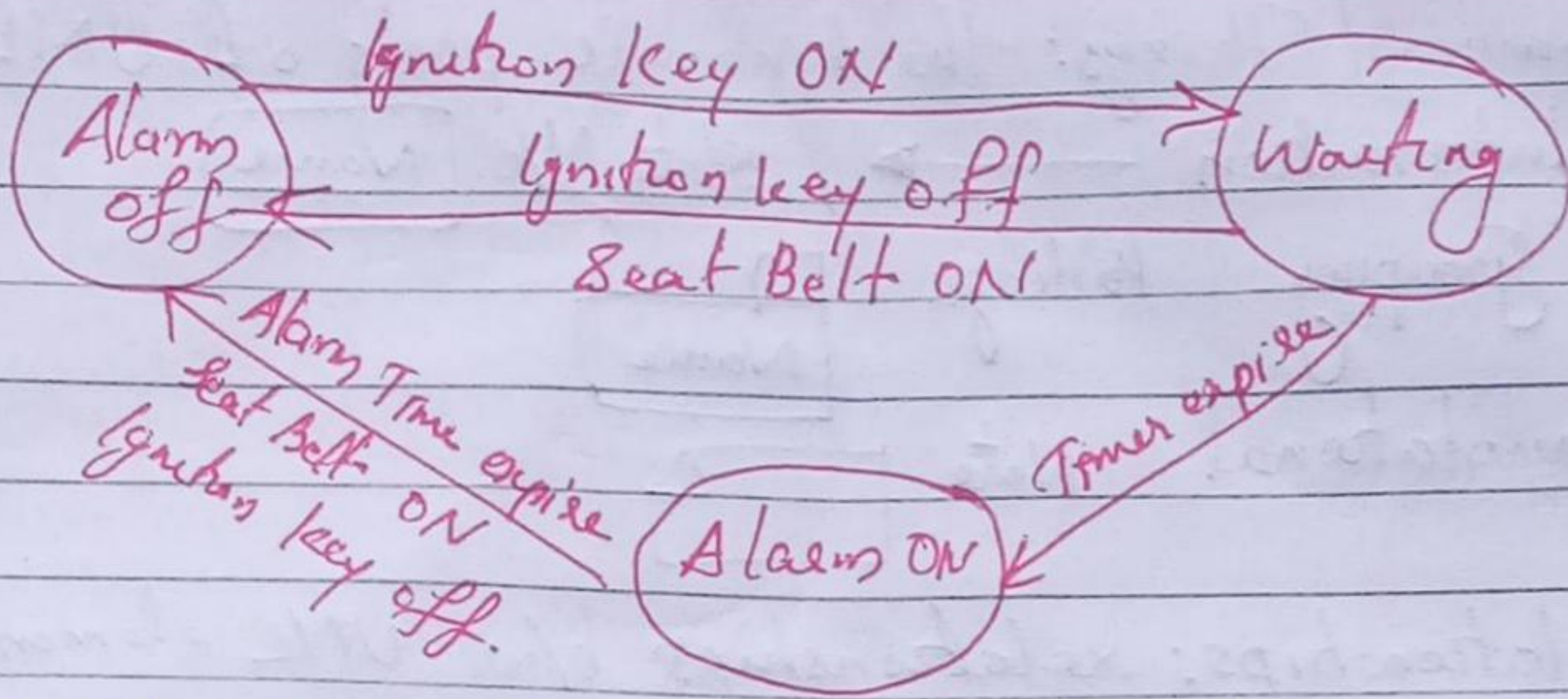
Figure shows the states of timer by circles & state transition by arrows. The count i/p is the clock i/p. The changed count value is the output. The o/p for is the ~~inc~~ increment in the count value. The state transition is the time-out on overflow when a prede

The State M/c model is used for modelling reactive or event driven embedded s/m whose processing behaviours are dependent on state transitions. The state M/c model describes the s/m behaviour with 'states', 'events' & 'Transitions'. State is a repⁿ of a current situation. An event is an i/p to the state. The event acts as stimuli for state transition. Transition is the movement from one state to another. Action is an activity to be performed by the state m/c.

A Finite State M/c (FSM) is one in which the no. of states are finite.

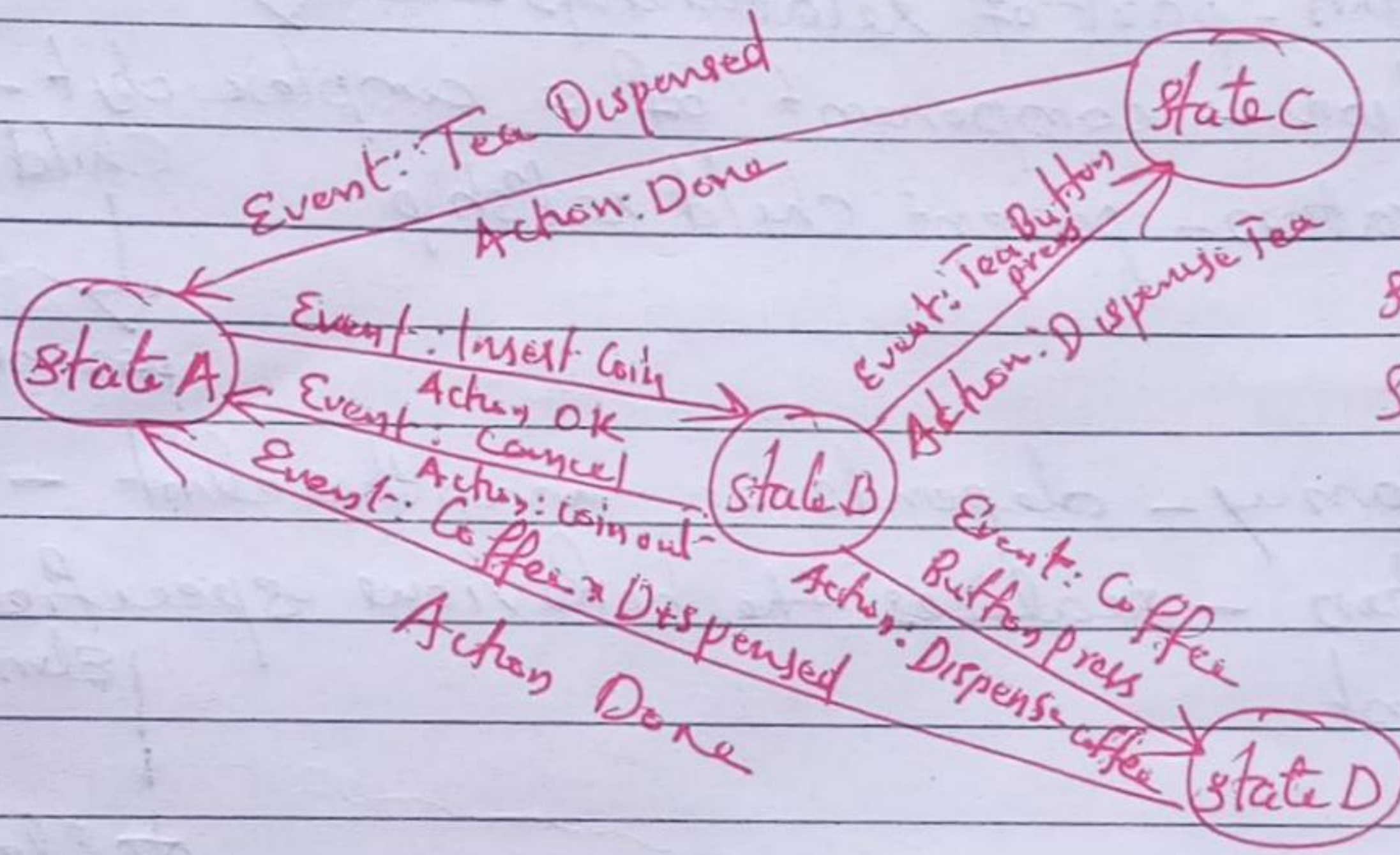
Eg: An Embedded s/m for driver/passenger 'Seat Belt Warning' in an automobile using the FSM model. The s/m reqts:

1. When the vehicle ignition is turned on & the seat belt is not fastened within 10 seconds of ignition ON, the s/m generates an alarm s/l for 5 seconds.
2. The Alarm is turned off when the alarm time (5s) expires or if driver/psngr fastens the belt or if the ignition switch is turned off.



ACVM, Automatic Tea/coffee Vending M/c, Coin Operated Public telephone Unit.

Automatic Tea/coffee Vending M/c



State A: Wait for Coin

State B: Wait for user i/p

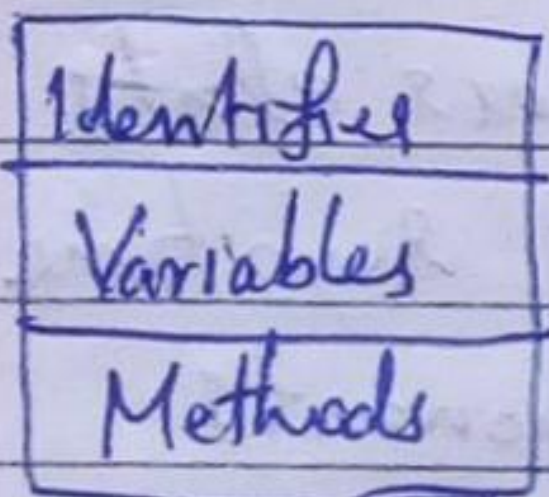
State C: Dispense Tea

State D: Dispense Coffee.

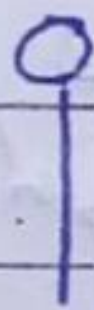
UML Modelling Cont.

→ Things, Relationships & Diagrams.

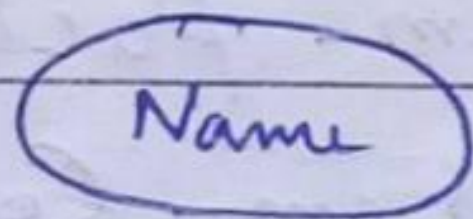
Things → Structural Things: represents static parts of a UML model. Eg: Class, Interface, Usecase, Component, node, Collaboration etc.



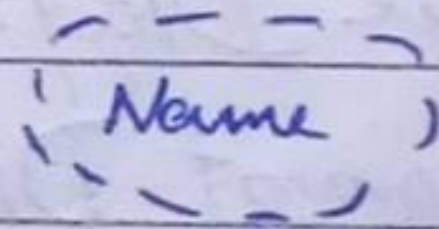
class



Interface

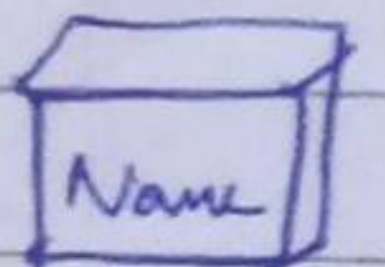
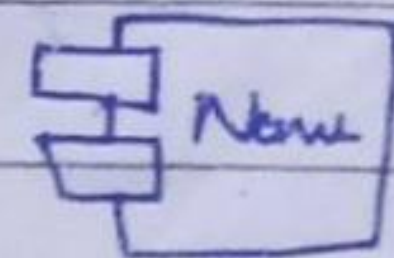


Usecase



Collaboration

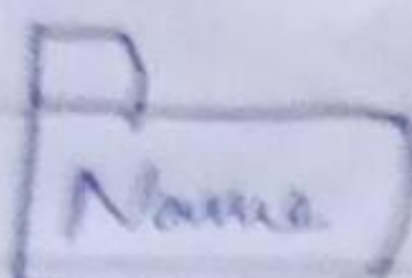
Component




Node

Behavioral Things: the dynamic parts of UML


Eg: Interaction , State M/c 

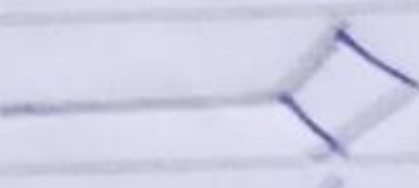
Grouping Package 


Annotational Note 

Relationships: relationships b/w UML elements (objts, classes)

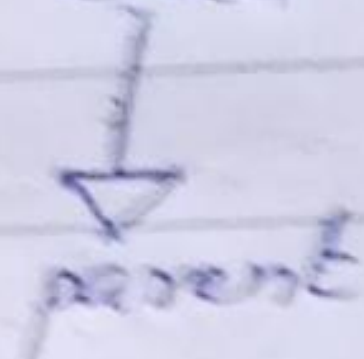
Relationships

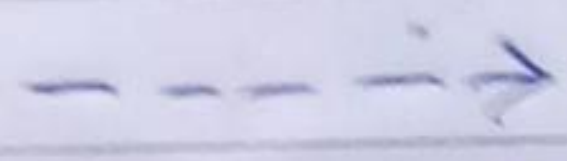
Association: structural rel'ship describing the link b/w objts. 

Aggregation - part of relationships 

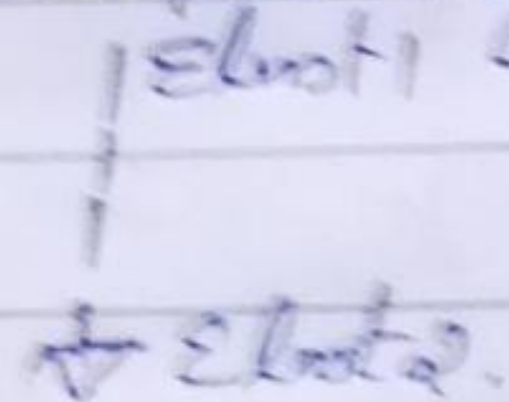
Composition - component of a complex objt 

Generalization - parent child rel'ship



Dependency - depends on another elmt 

Realization - realizes the behaviours specified by the other elmt.



UML Diagrams:

Static Diagrams - represents static/structural aspects.

Eg: Objt Diagrams - set of objts & relationships.

Class Diagrams - classes, their iff's, interactions & rel'ships.

Component Diagrams - Implⁿ view of a sm.

Package Diagrams - packages & their elmts

Deployment Diagram - Configⁿ of runtime processing nodes & components.

Use Case Diagram : s/m finality as seen by users.
- usecases & actors.

Sequence Diagram

~~UAD~~
27/13/19

Name	
Reg. No.	

**NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE
(NAAC Accredited)**

(Approved by AICTE, Affiliated to KTU University, Kerala)

SRIES TEST - 1 (2018-19)

CS 404 Embedded Systems

Semester: VIII	Programme: B.TECH	Max. Mark: 40	
Course Code & Name: CS 404 Embedded Systems		Duration: 90 min	
Knowledge Level (KL)	K1: Remembering	K3: Applying	K5: Creating
Course Outcome (COL)	K2: Understanding	K4: Analysing	K6: Evaluation

PART-A

Answer ALL Questions (4 x 3 = 12 Marks)

1.	Define a system and an Embedded System.	K1/CO1
2.	List three main components of an Embedded System.	K4/CO1
3.	Write short notes on Computational models.	K4/CO2
4.	Define Control Data Flow Graph.	K1/CO2

PART-B

Answer ALL Questions (2 x 14 = 24 Marks)

5 a.	Demonstrate the role of individual components involved in a typical embedded system.	14 Marks	K6/CO1
<i>OR</i>			
6 a.	Explain the challenges in Embedded System design.	14 Marks	K6/CO1
7 a.	Analyze the characteristics of different Program Models.	10 Marks	K4/CO2
b.	Classify different State Machines with example	4 Marks	K5/CO2
<i>OR</i>			
8 a.	Differentiate between Concurrent Model and Object Oriented Model.	7 Marks	K4/CO2
b.	Identify the features of Sequential Model.	7 Marks	K5/CO2

Embedded Firmware Design & Development:

The embedded firmware is responsible for controlling the various peripherals of the embedded h/w & generating response in accordance with the final reqmts mentioned in the reqmts for a particular embedded product. Embedded firmware is stored at a permanent memory (ROM) and they are nonalterable by end users.

Designing embedded firmware requires understanding of the particular embedded product h/w, like various component i/fing, memory map details, I/O port details, configuration & register details of various h/w chips used & some pgmng lang.

Embedded firmware development process starts with the conversion of the firmware reqmts into a pgm model using modelling tools like UML or flow chart based repⁿ.

Embedded Firmware Design Approaches:

Approaches dependent on the complexity of the fns to be performed, the speed of opⁿ reqd etc.

Two basic approaches:

1. Conventional Procedural Based Firmware Design ✓
2. Embedded OS Based design. ✓

1. The Super Loop Based Approach:

- This approach is adopted for applⁿs that are not time critical & where the response time is not important. It is conventional procedural programming where the code is executed task by task. The task listed at the top of the pgm code is executed first & the tasks just below the top are executed after completing the first task.

The firmware exⁿ flow for this will be

1. Configure the common parameters & perform initialisation for various h/w components memory, regs etc.
2. Start the first task & execute it.
3. Execute the 2nd task
4. Execute the next task
- 5 :
- 6 :
- 7 execute the last task
8. Jump back to the first task & follow the same flow

Eg: void main()

```
{ configurations();  
  initializations();  
  while(1)  
  { task 1();  
    task 2();  
    :  
    task n();  
  } }
```

At Almost all tasks in embedded appl^s are non-ending & are repeated infinitely throughout the opⁿ. task 1 to n are performed one after another and when the last task (nth task) is executed, the firmware exⁿ is again redirected to Task 1 & it is repeated forever in the loop. This repetition is achieved by using an infinite loop. This approach is referred to as 'super loop based Approach'.

→ This approach doesn't require an O.S, since there is need for scheduling which task is to be executed & assigning priority to each task. The priorities are fixed & the order in which the tasks to be executed are also fixed.

→ This type of design is deployed in low-cost embedded pdts & pdts where response time is not time critical. For eg: reading/writing data to & from a card using a card reader requires a seq. of ops like checking the presence of card, authenticating the ops, reading/writing etc. It should strictly follow a specified sequence.

* The major drawback of this approach is that any failure in any part of a single task will affect the total s/m. If the pgm hangs up at some point while executing a task, it will remain there forever & st pdt stops functioning.

The remedy: Use of h/w & s/w watch Dog Timers helps in coming out from the loop when an unexpected failure occurs. In turn may cause additional h/w cost & firmware overheads.

* Lack of real timeliness. If the no. of tasks to be executed within an applⁿ increases, the time at which each task is repeated also increases.

2. The Embedded OS Approach

It contains operating s/ms, which can be either a General Purpose OS or a Real Time O.S to host the user written applⁿ s/w. The OSPOS is based applⁿ development where the device contains an OS (Windows/Linux) & user applⁿs can be created on top of it. Eg: Microsoft Windows Xp (PDA, Handheld devices).

Eg: Driver s/w for diff. h/w present on the board to communicate with them.

Real Time Operating S/m based design approach is employed in embedded products demanding Real Time Response.

RTOS respond in timely and predictable manner to events. A Real Time kernel responsible for performing preemptive multitasking, scheduler for scheduling tasks, multiple threads. eg: Symbian, Embedded Linux, Windows CE. eg: for RTOS.

Embedded Firmware Development Languages:

→ Either a target processor/compiler specific lang or target processor/compiler independent language or combination of Assembly & HLL.

Assembly Lang. based Development:

Assembly lang is the human readable notation of m/c lang where as is a processor understandable lang.

M/c lang. is made readable by using specific symbols called 'mnemonics'.

Assembly lang. pgm is the task of writing processor specific m/c code in mnemonic form, converting the mnemonics into actual processor instⁿs & associated data using an assembler.

The general format of an assembly lang. instⁿ is an Opcode followed by Operands. The Opcode tells the processor what to do & the Operands provide the data & infⁿ reqd to perform the action specified by the opcode.

Eg: MOV A, # 30 $\xrightarrow{\text{m/c lang}}$ 01110100 00011110
 MOV A 30

Each line of an assembly lang. pgm is split into 4 fields: LABEL OPCODE OPERAND COMMENTS.

↓
optional

LABEL is commonly used for

- * A mem. location, address of a pgm, sub-routine, code etc.
- * Max. length of label differs b/w assemblers.

Eg: ; subroutine for generating delay
; Delay parameter passed through reg. R₁
; Return value None
; Registers used R₀, R₁

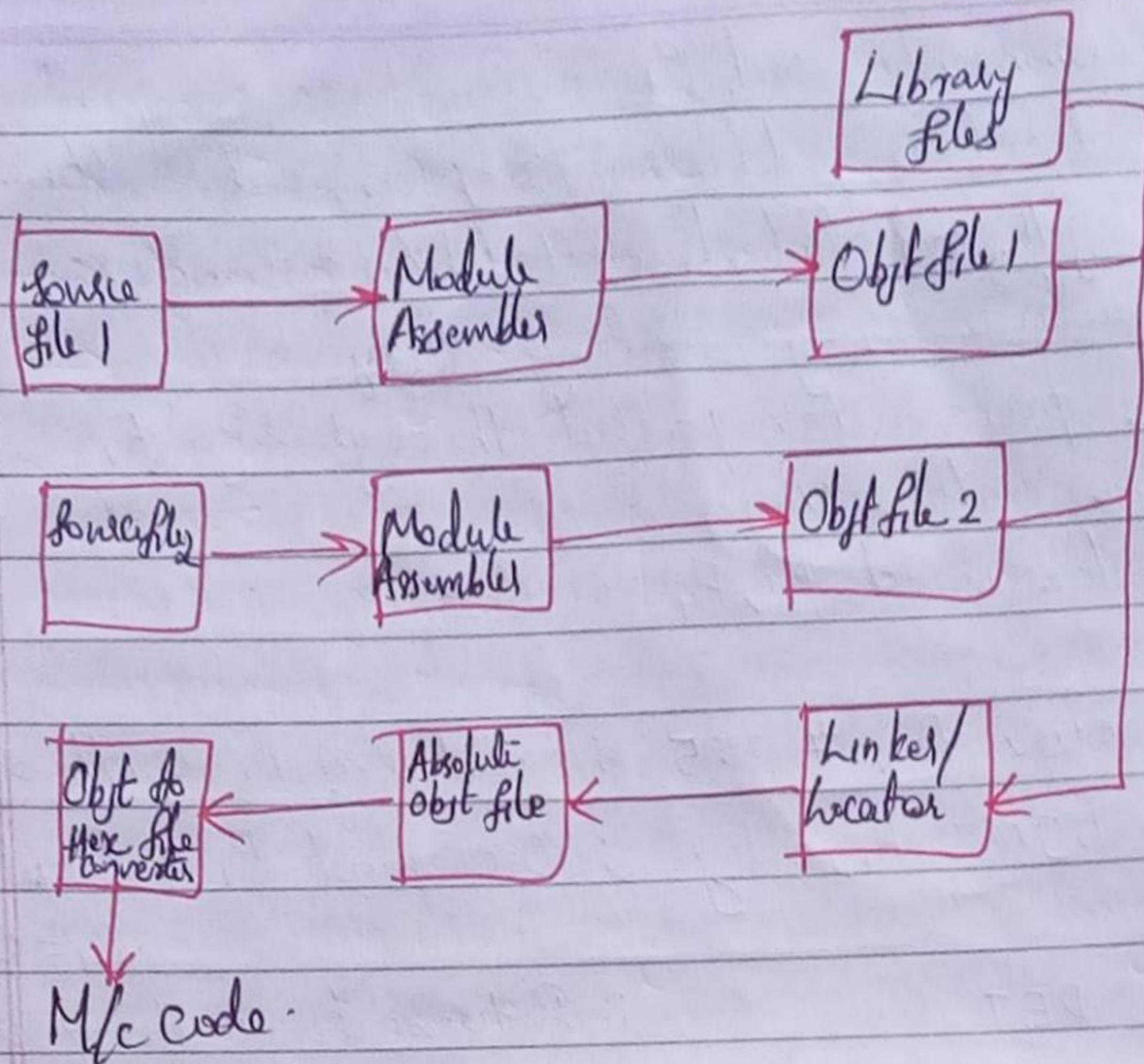
```
DELAY: MOV R0, #255 ; Load reg R0 with 255  
      DJNZ R1, Delay ; Decrement R1 & loop till  
                          R1=0  
      RET ; Return to calling pgm.
```

Conversion of the assembly lang. to m/c lang is carried out by a seq. of op's

1. Source file Objt File File Translation.

→ Translation of assembly code to m/c code is performed by assembler.

Each source module is written in Assembly & is stored as .src file or .asm file. Each file can be assembled separately to examine the syntax errors & incorrect assembly instructions. On successful assembling of each .src / .asm file a corres. objt file is created with extension '.obj'. The objt file doesn't contain the absolute address of where the generated code needs to be placed on the pgm. memory & hence it is called relocatable segment. It can be placed at any code memory location and it is the responsibility of linker/locator to assign absolute address for this module.



2. Library file Creation & Usage:

Libraries are specially formatted, ordered pgm collections of objt modules that may be used by the linker at a later time. When the linker processes a library, only those objt modules in the library that are necessary to create the pgm are used. Library files are generated with extension 'lib'.

Eg: LIB51 from Keil s/w eg: library creator used for A51 Assembler.

3. Linker and locator: responsible for linking the various objt modules in a multi-module prjt & assigning address to each module.

BL 51 from Keil s/w eg: ~~for~~

4. Objt to hex file Converter: Final stage in the conversion of Assembly lang. to m/c lang. Hex file is the repⁿ of m/c code & the hex file is dumped into the code memory of processor.

OH51 from Keil s/w eg:

Advantages of Assembly lang. based Devpt:

- * Efficient Code Memory & Data Memory Usage. (Memory Optimization)
- * High Performance: ✓
- * Low level H/w Access ✓
- * Code Reverse Engg. ✓

Drawbacks:

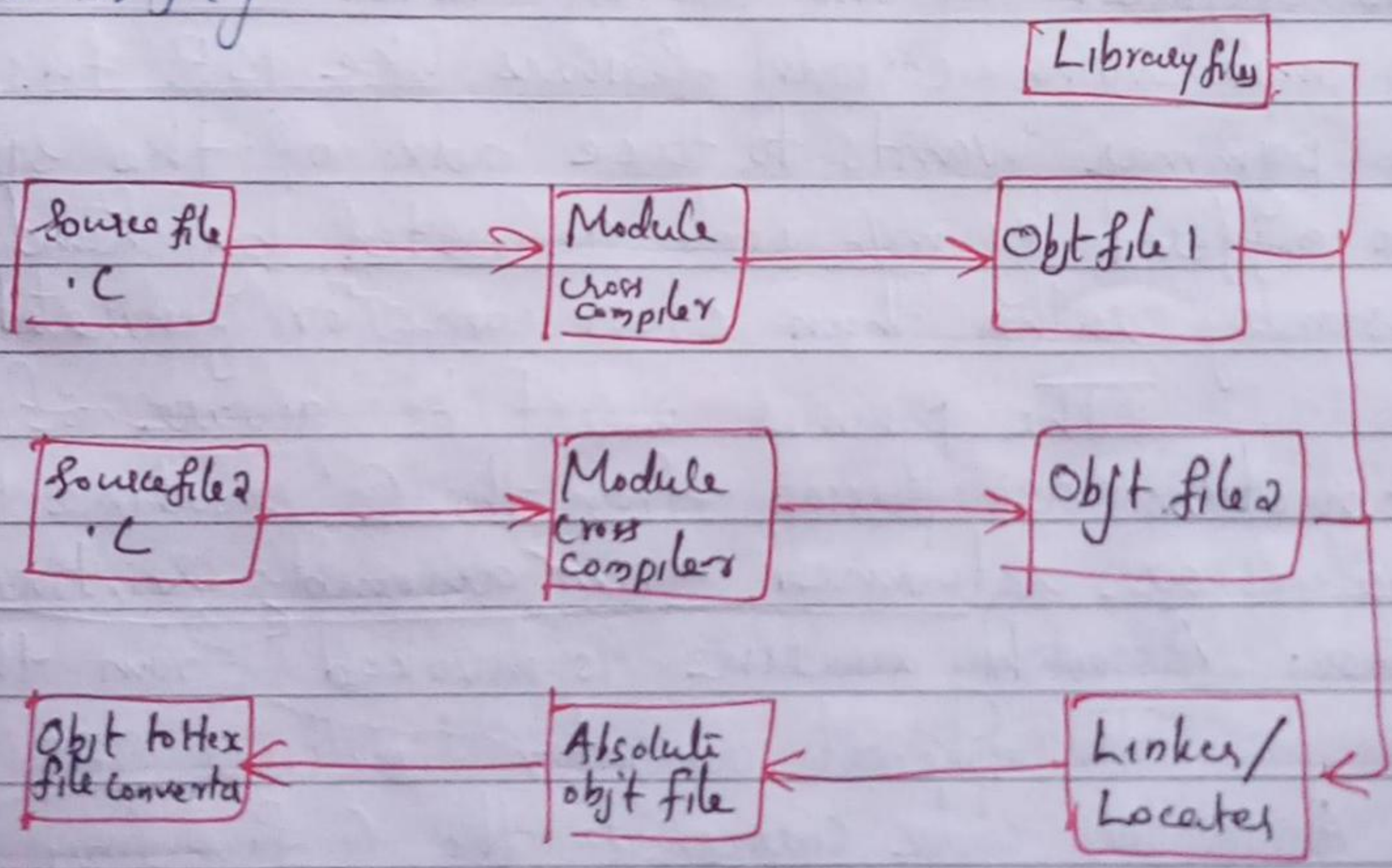
- High development Time ✓
- Developer Dependency ✓
- Non portable.

High Level Lang. Based Development:

Eg: C, C++, or Java.

The most commonly used high level lang. for embedded firmware applⁿ development is C.

The various steps involved in the conversion of a pgm written in high level lang. to codes. binary file/mc lang. is in fig.



Advantages of HLL based Devpt:

- Reduced Devpt Time
- Developer Independence
- Portability.

Limitations

* not so efficient in generating optimised target processor specific instructions.

* High cost

3. Mixing Assembly & High level language:

High level lang & assembly languages are usually mixed in 3 ways:

1. Mixing Assembly with HLL (eg: Assembly lang. with C).

Assembly routines are mixed with C in situations where the entire pgm is written in C & the cross compiler in use do not have a built in support for implementing certain features like ISR fns or if the programmer wants to take adv. of the speed & code offered by m/c code generated by hand written assembly rather than cross compiler generated m/c code.

The programmer must be aware of how parameters are passed from the 'C' routine to Assembly & values are returned from assembly routine to C & how Assembly routine is invoked from the C code.

The special compiler directive SRC generates the Assembly code corres. to the C function & each line of the source code is converted to the Assembly instⁿ.

2. Mixing High level lang. with Assembly (C with Assembly)

—It is useful in follo. scenarios.

1. The source code is already available in Assembly lang & a routine written in a HLL like 'C' needs to be included to the existing code.

2. The entire source code is planned in Assembly code for various reasons like optimised code, optimal performance. But some portions of the code may be very difficult & tedious to code in Assembly.

eg: 16 bit multiplication & division in 8051 Assembly

3. To include built-in library fns written in C lang. provided by cross compiler.

Inline Assembly: for inserting target processor/utelles specific Assembly instⁿs at any location of a source code written in HLL C. This avoids the delay in calling an assembly routine from a 'C' code. Special keywords are used to indicate that the start & end of

Programming in Embedded C.

Whenever the conventional 'C' language & its extensions are used for programming embedded s/ms, it is referred as 'Embedded C' pgming. Almost all embedded s/ms are limited in both storage & working memory resources.

1. 'C' v/s. 'Embedded C'

C → well structured & standardised general purpose programming lang. with extensive bit manipulation support. 'C' offers a combination of the features of HLL and assembly & helps in h/w access pgming as

well as business package developments.

Embedded 'C' can be considered as a subset of conventional C language. Embedded 'C' supports all 'C' instⁿs and incorporates a few target processor specific instructions. The implementation of target processor/compiler specific instⁿs depends upon the processor as well as the supported cross-compiler for the particular Embedded 'C' lang. A s/w pgm. called 'Cross Compiler' is used for the conversion of pgms written in Embedded 'C' language to target processor specific instⁿs.

Compiler vs. Cross-Compiler

→ Compiler is a s/w tool that converts a source code written in HLL on the top of a particular O.S running on a specific target processor architecture. The source code is converted to the target processor specific m/c instⁿs. The devpt. is platform specific. Native Compilers generates m/c code for the same m/c on which it is running.

→ Cross Compilers are the s/w tools used in cross-platform development applⁿs. In cross-platform devpt, the compiler running on a particular target processor/O.S converts the source code to m/c code for a target processor. Keil C51 is an eg: for cross-compiler. The term Compiler is used interchangeably with Cross-Compiler' in embedded firmware applⁿs.

MODULE IV

Date:

Question Bank.

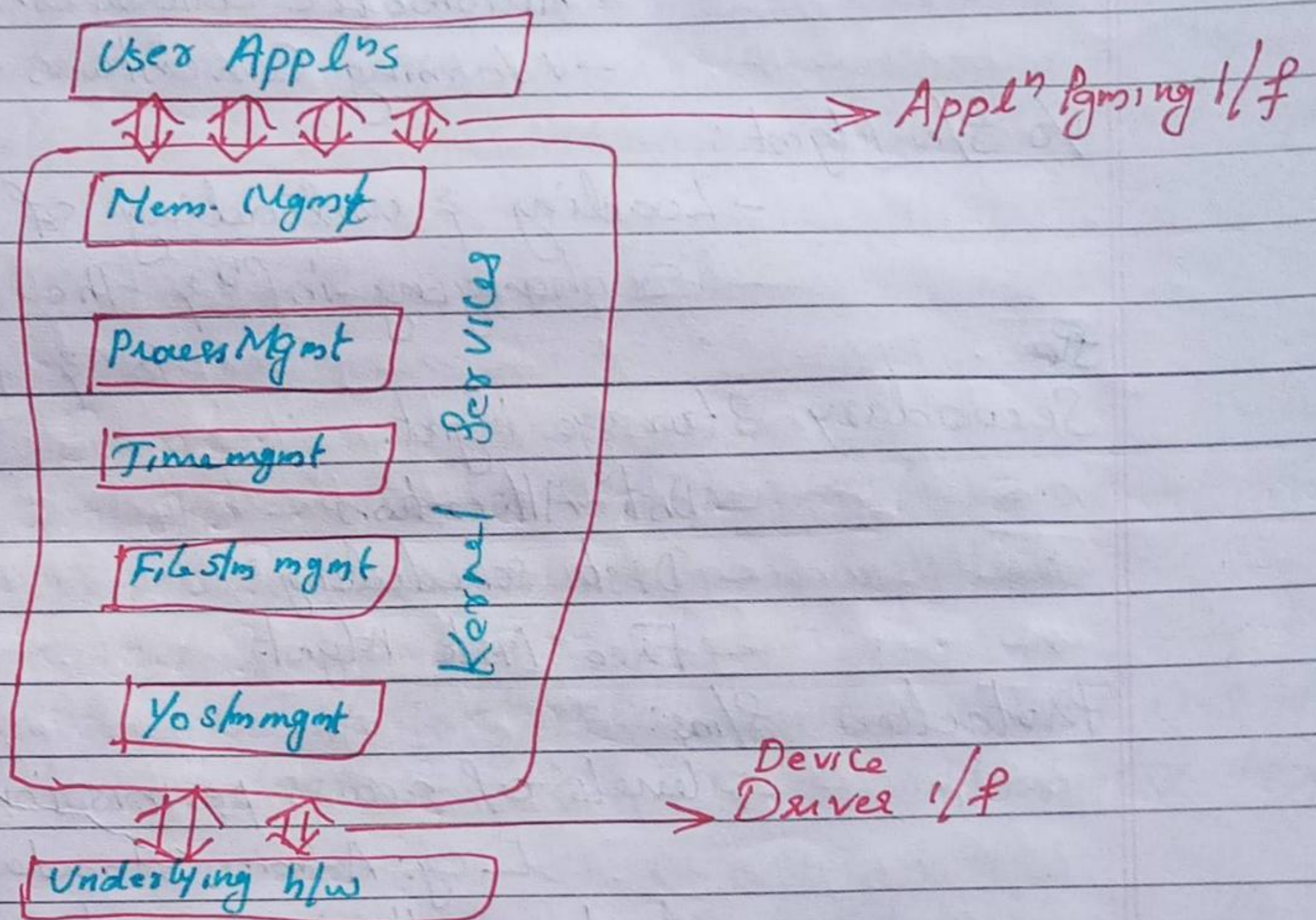
1. What is an O-S? where it is used & what are its primary fns?
2. What is kernel? what are the diff. fns handled by a general purpose kernel?
3. What is kernel space & user space? How is kernel space & user space?
4. Explain the various elmts of an embedded s/m devt environment.
5. Explain the role of IDE for Embedded s/w Devt.
6. What are the diff. files generated during the cross-compilation of an Embedded C file?
7. Explain the various details held by a list file, ^{objfile} generated during cross compiling an Embedded C.
8. Diff. b/w Assembler & disassembler.
9. What is a Monitor Program? Explain its role in Embedded firmware debugging?

10. Explain On Chip Debugging
11. Diff. techniques for Firmware debugging.
12. Diff. b/w simulator & Emulator.
13. Advantages & Limitations of simulator based debugging.
14. What is ROM emulation.
15. Diff. tools for h/w debugging.
16. Explain Boundary Scan based Debugging.

Operating S/m Basics (IV)

O.S is a s/m s/w which makes the user to use the s/m more effectively. It acts as i/f b/w the user applications & the underlying s/m resources through a set of s/m facilities & services.

- make the s/m convenient to use
- Organise & manage the s/m resources efficiently.



- Process Mgmt: - setting up memory space for the process
 - loading process's code into memory.
 - allocating s/m resources.
 - scheduling & managing exⁿ of the process.
 - Managing PCB
 - Interprocess Comm, Synchronization
 - process termination.

Page No: _____
Date: _____

Memory Mgmt: Memory Mgmt Unit of the kernel is responsible for - Keeping track of which part of the memory area is currently used by which process.
- Allocating & deallocating memory space.

File S/m Mgmt: text file, image file word etc
- creation, deletion & alteration of files
- " " " " of directories
- Saving of files in 2^o storage memory
- Automatic allocation
- Naming Conventions

Io S/m Mgmt:
- Loading & unloading of device drivers
- Exchanging info & the s/m specific ctrl s/ls to & from the device.

Secondary Storage Mgmt:
- Disk Allocation
- Disk scheduling
- Free Disk Mgmt

Protection S/m:
- levels of access permissions
- eg: Admin, standard, restricted etc.
- deals with implementing the security policy.

Interrupt Handler: - mechanism for all external/internal interrupts generated by the s/m.

Kernel Space & User Space: The memory space at which the kernel code is located is known as Kernel space.
All user appl's are loaded to a specific area of primary memory & this mem. area is referred as User Space.

Swapping: The act of ~~load~~ loading the code into part of m/m is termed as swapping. - happens b/w m/m & mem.

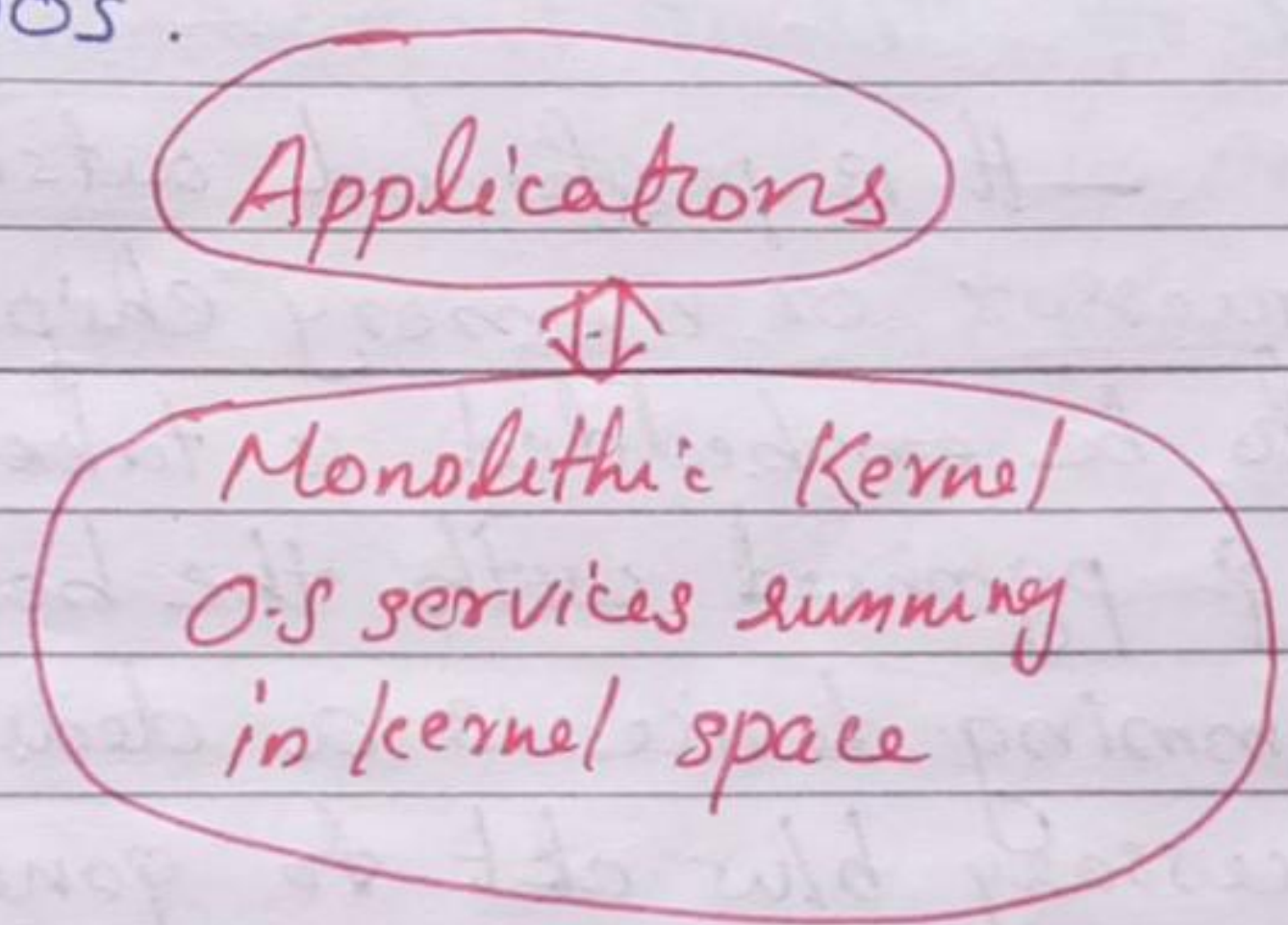
Monolithic Kernel & Micro Kernel: Approaches for building O.S Kernel.

Monolithic Kernel: All kernel services run in the kernel space. All kernel modules run within the same mem. space under a single kernel thread.

- allows effective utilization of the low level features of the s/m.

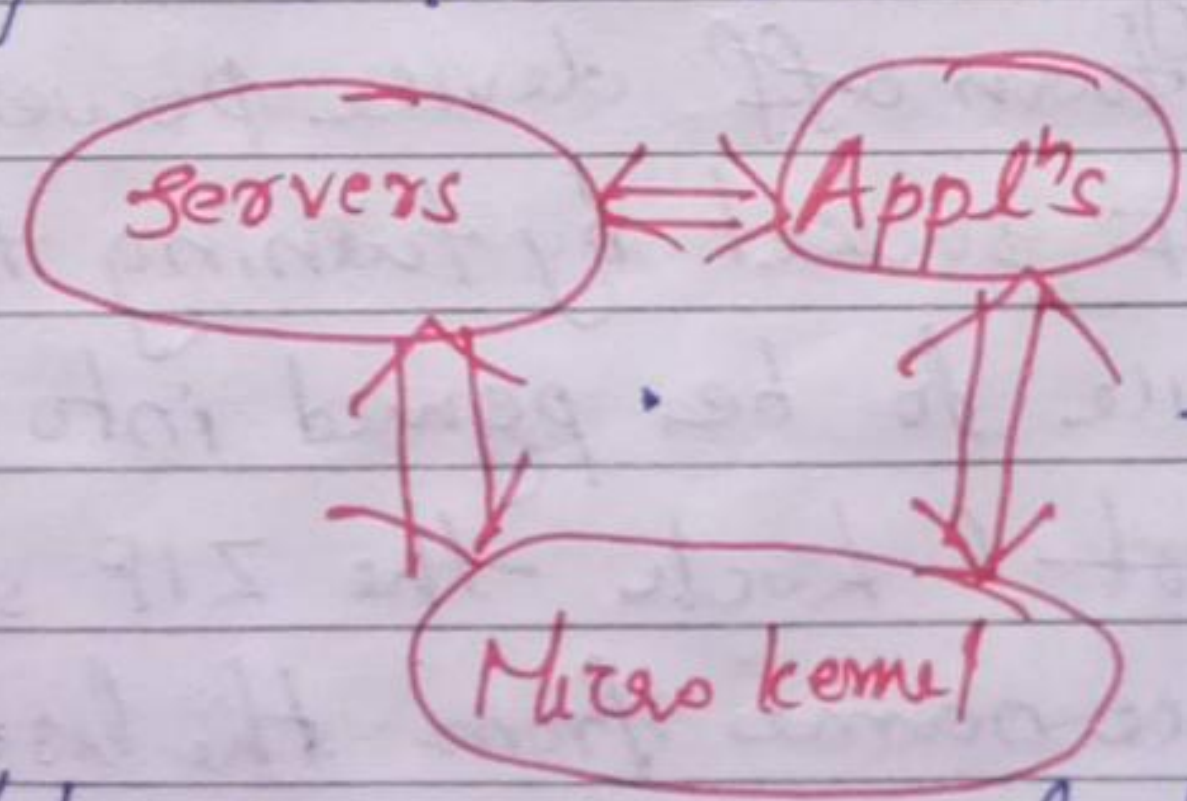
- draw back: error or failure in any of the kernel modules leads to the crashing of entire kernel appl?

Eg: LINUX, MS-DOS.



Micro kernel: incorporates only the essential set of O-S into the kernel. The rest of O-S services are implemented in pgs known as servers which runs in user space.

Mem. mgmt, process mgmt, times mgmt & interrupt handlers are the essential services which forms the part of the kernel. Eg: Mach, QNX, Minix 3



Robustness: If a pblm encountered, in any of the services, the same can be reconfigured & restarted without the need for restarting the entire O-S.

Configurability: Any services which run as 'Server appl' can be changed without the need to restart the whole s/m.

Integration of H/w & Firmware

Integration of h/w & f/w deals with the embedding of firmware into the target h/w board. It is the process of Embedding Intelligence to the product. A variety of techniques are used for embedding the firmware into the target board. The commonly used firmware embedding techniques for a non-OS based embedded s/m are explained below. The non OS based embedded s/ms store the firmware either in the onchip processor/controller memory or offchip memory.

1. Out-of-Circuit Programming:

- It is performed outside the target board.

The processor or memory chip into which the firmware needs to be embedded is taken out of the target board & programmed with the help of a programming device.

The programming device is a dedicated unit which contains the necessary h/w ckt to generate the s/l's.

The sequence of op's for embedding the firmware

1. Connect the programming device to the specified port of PC (USB port). ✓

2. Power up the device (LED is ON)

3. Execute the programming utility on the PC & ensure proper connectivity established b/w PC & programmer.

In case of error, turn off device power & try connecting.

4. Undo the ZIF socket by turning the lock pin.

5. Insert the device to be programmed into the open socket ~~as per the insert~~

6. Lock the ZIF socket

7. Select the device name from the list of supported devices

8. Load the hex file which is to be embedded into the device.

9. Pgm the device by 'Pgm' option of utility pgm.
10. Wait till the completion of programming opⁿ.
11. Ensure the programming is successful by checking the status LED on the programmer (Green Success Red for error).
12. Unblock the ZIF socket & take the device out of pgmer.

Now firmware is successfully embedded into the device. Insert the device into the board, power up the board & test it for eqd finalities.

The major drawback — high development time. Whenever the firmware is changed, the chip should be taken out of the devpt board for re-programming. This tedious & prone to chip damages.

The pgmer facilitates pgming of only one chip at a time & it is not suitable for batch prodⁿ.

— used in low volume pdts & Proof of Concept PoC pdt Development.

In-System Programming (ISP)

→ Pgming is done 'within the s/m'; the firm/w. is embedded into the target device without removing it from the target board. The only pre-requisite is that the target device must have an ISP support. Apart from the target board, PC ISP cable & ISP utility, no other additional h/w is eqd for ISP. Chips supporting ISP generates the necessary pgming s/l's internally, using the chip's supply voltage. In order to perform ISP opⁿs the target device should be powered up in a special ISP mode.

ISP mode allows device to communicate with an external host through a serial i/f such as a PC or terminal.

The device receives cmds & data from the host, erases & repgms code memory acc to the received cmd.

Once ISP ops are completed, the device is reconfigured so that it will operate normally by applying a reset or re-power up.

In s/m Pgmng with SPI Protocol:

Devices with Serial Peripheral I/f contains a built-in SPI i/f & the on-chip EEPROM or flash memory is pgmed through this i/f.

The primary i/o lines involved in SPI

MOSI - Master Out Slave In ✓

MISO - Master In Slave Out ✓

SCK - S/m clk ✓

RST - Reset Target Device ✓

GND - Ground of Target Device ✓

PC acts as the master & target device acts as the slave in ISP. The pgm data is sent to the MOSI pin of target device & the device ACK is originated from the MISO pin of the device. SCK pin acts as the clock for data transfer. A utility pgm can be developed on the PC side to generate the above S/I lines. The target device works under 5V.

3. In Application Programming (IAP)

- is a technique used by the firmware running on the target device for modifying a selected portion of the code memory. It is not first time embedding technique. It modifies the pgm code memory under the ctrl of the embedded application.

Eg: Updating calibration data, look up tables & which are stored in code memory.

A common entry pt. to these API routines is provided for iffing them to the end user's applⁿ.

Functions are performed by setting up specific regs. are reqd by a specific opⁿ & performing a call to the common entry pt.

The Boot ROM resident API instⁿs which perform various fns such as pgmng, erasing & reading the flash memory during ISP mode, are made available to the end user written firmware for IAP. The Boot ROM is shadowed with the user code memory in its address range. This shadowing is controlled by a status bit. When this status bit is set, accesses to the internal code memory in this addr. range will be from the Boot ROM. When cleared, accesses will be from the user's code memory.

4. Use of Factory Programmed Chip:

It is possible to embed the firmware into the target processor/microcontroller memory at the time of chip fabrication itself. Such chips are known as 'Factory programmed chips'. Once the firmware design is over & the firmware achieved operational stability, the firmware files can be sent to the chip fabricator to embed it into the code memory.

- convenient for mass production applications & it greatly reduces the pdt devt time.

5. Firmware Loading for OS based Devices:

The OS based embedded s/ms contain a special piece of code called 'Boot loader' pgm which takes care of the OS & applⁿ firmware embedding & copying of the OS image to the RAM of the s/m for exⁿ.

The Boot loader for such embedded s/ms comes as pre-loaded or it can be loaded to the memory using various inf if supported like JTAG.

Types of Files Generated on Cross-Compilation

Cross compilation is the process of converting a source code written in HLL (Embedded C) to a target processor/controller understandable m/c code. (The conversion of the code is done by s/w running on a processor/controller which is diff. from the target processor.) The s/w performing this opⁿ is referred as Cross-compiler. Cross compilation is the process of cross platform s/w/f/w devpt.

Cross assembling is similar to cross compiling the only diff. is that the code written in target processor/controller specific assembly code is converted into its corres. m/c code. The applⁿ converting Assembly instⁿ to target processor/controller specific m/c code is known as cross assembler.

The various files generated during the cross compilation cross assembling process are:

List file (.lst), Hex file (.hex), preprocessor Output file, Map file, object file (.obj).

1) List File (.LST File)

Listing file is generated during the cross compilation process & it contains an abundance of info about cross compilation process, like cross compiler details, formatted source text ('C' code), assembly code generated from the source file, symbol tables, errors detected during cross compilation process.

An Eg: sample.c.

The list file generated contains the follo. sections.

1) Page Header: A header on each page of the listing file which indicates the compiler version no, source file name, date, time & page no.

C51 compiler V8.02 sample 5/23/2006 11:29:58 page 1.

2) Command line: represents the entire card line that was used for invoking the compiler.

C51 Compiler V8.02, Compilation of module sample
Object Module placed in sample.obj

Compiler invoked by: c:\keil\C51\BIN\C51.exe sample.c
Browse & Debug object extend code
LIST include symbols.

3) Source Code: The source code listing outputs the line no. as well as the source code on that line. Special cross compiler directives can be used to include or exclude the conditional codes in the source code listings.

Assembly listing: contains the assembly code generated by the cross compiler for the 'C' source code. Assembly code generated can be excluded from the list file by using special compiler directives.

(4) Symbol Listing: contains symbolic infoⁿ about various symbols present in the cross compiled source file. Symbol listing contains the sections symbol name, symbol class (structure, typedef), memory space, datatype, offset & size in bytes.

NAME	CLASS	MSPACE	TYPE	OFFSET	SIZE
------	-------	--------	------	--------	------

(5) Module Information: provides the size of initialized & uninitialized memory areas defined by the source file.
Warnings & Errors: records the errors encountered or any statement that may create issues in applⁿ, during cross compilation.

CSI Compilation Complete. 0 warnings(s), 0 error(s)

(6) Preprocessor Output File: contains the preprocessor output for the preprocessor instⁿs used in the source file. It is used for verifying the opⁿ of macros & conditional preprocessor directives. File extension of preprocessor output file is cross compiler dependant.

(7) Object File (.OBJ file): The format/internal repⁿ of the .OBJ file is cross compiler dependant. OMF51 or OMF2 are the 2 object's file formats supported by CSI cross compiler.

The list of some of the details stored in an objt file.

1. Reserved memory for global variables
2. Public symbol (variable & fn) names.
3. External symbol (variable & fn) names
4. Library files which to link
5. Debugging infoⁿ.

① Map File: The objt files created are re-locatable codes; their locⁿ in the code memory is not fixed. It is the responsibility of linker to link all these objt files. Linking & locating re-locatable objt files will also generate a list file called linker list file or map file. Map file contains info about the link/locate process & is composed of no. of sections. The diff. sections listed in a map file are cross compiler dependant.

① Page Header - linker version, date, time & pg no.

② Command Line - entire command line for invoking the linker

③ CPU Details - about target CPU & mem. model (internal data memory, ext. data mem, paged data mem).

eg: Memory model: small.

④ Input Modules: names of all objt modules, & library files & modules that are included in the linking process.

⑤ Memory Map - lists starting address, length, relocation type & name of each segment in the pgm.

⑥ Symbol Table: contains value, type & name for all symbols from the diff. i/p modules.

⑦ Program Size: size of various memory areas as well as constant & code space for the entire applⁿ.

eg: Pgm size: data = 301, xdata = 0, code = 1079

⑧ Warning & Errors

① Hex File (.HEX): - binary executable file created from the source code. The absolute objt file created by linker/locator is converted into processor understandable code. The utility used for converting an objt file to a hex file is known as Objt to Hex file converter. Hex files embed the m/c code in a particular format.

Eg: Intel HEX, Motorola Hex

Disassembler/Decompiler

Disassembler - converts m/c codes into target processor specific Assembly codes/inst's.

This process is known as Disassembling.

Decompiler - for translating m/c codes into corres. H/L inst's. ☺

Disassembler/Decompiler are deployed in reverse engg. powerful tools for analysing the presence of malicious codes in an executable image. - Available as freeware tools readily available or commercial tools. Disassemblers/compilers generate a source code which is somewhat matching to the original source code from which the binary code is generated.

Simulators, Emulators, core to & Debugging:

Simulator is a sw tool used for simulating the various conditions for checking the fnality of the applⁿ firmware. The Integrated Devt Environment (IDE) itself will be providing simulator support & they help in debugging the firm-ware for checking its required fnality. For eg: if the pdt under devt is a handheld device, to test the fnalities of the various menu & user i/fo, a soft form model of the pdt with all vi as given in the end of pdt can be developed in sw.

Emulator is a h/w device which emulates the fnalities of the target device & allows real time debugging of the embedded firmware in a h/w envt.

Simulators

Simulators simulate the target h/w & the firmware exⁿ can be inspected using simulators.

Features:

1. Purely s/w based. ✓
2. Doesn't require a real target s/m. ✓
3. Very primitive. ✓
4. Lack of Real-time behaviour. ✓

Advantages:

1. No need for original target Board: IDE's s/w support simulates the CPU of the target board. Since the real h/w is not required, firmware devt can start well in advance immediately after the device i/f & memory maps are finalised. This saves devt-time.

2. Simulate I/O Peripherals: Simulator provides the option to simulate various I/O peripherals. Using simulator's I/O support you can edit the values for I/O registers & can be used as the I/O value in the firmware exⁿ. Hence it eliminates the need for connecting I/O devices for debugging the firmware.

3. Simulates Abnormal Conditions: It helps the developer to study the behaviour of the firmware under abnormal i/p condⁿs.

Disadvantages:

→ Deviation from Real Behaviour: Simulator based firmware debugging is always carried out in a devt envt where the developer may not be able to debug the firmware under all possible combⁿs of i/p. The simulation debugging result may not be the same when the firmware runs in a pdn envt.

→ Lack of timeliness: - it is not real-time in behaviour. A real appⁿ the I/O condⁿ may be varying. Simulation goes for simulating those condⁿs for known I values.

Emulators & Debuggers:

Debugging in embedded applⁿ is the process of diagnosing the firmware exⁿ, monitoring the target processor's regs. & memory while the firmware is running and checking the s/l's from various buses of the Embedded h/w. Debugging process is classified into

- h/w debugging ✓
- f/w debugging ✓

H/w debugging deals with the monitoring of various bus s/l's & checking the status lines of the target h/w.

F/w debugging deals with examining the firmware exⁿ, exⁿ flow, changes to various CPU regs & status regs. on exⁿ of firmware to ensure that the firmware is running as per the design.

Incremental EEPROM Burning Technique:

The code is separated into diff. final code units. Instead of burning the entire code into the EEPROM chip at once, the code is burned in incremental order, where the code corres. to all finalities are separately coded, cross compiled & burned into the chip one by one. If the 1st finality is found working perfectly on the ~~the~~ target board with the corres. code burned into EEPROMs, go for burning the code corres. to next finality & check whether it is working. Repeat this process till all the finalities are covered. Ensure that before entering into one level up, the previous level

has delivered a correct result. If the code corres. to any functionality is found not giving the expected result, fix it by modifying the code & then only go for adding the next functionality for burning into the EEPROM.

Inline Breakpoint Based Firmware Debugging:

→ Within the firmware where you want to ensure that firmware exⁿ is reaching upto a specified pt., insert an inline debug code immediately after the pt. The debug code is a printf function which prints a string as per the firmware. You can insert debug codes (printf()) cmds at each pt. where you want to ensure the firmware exⁿ is covering that pt. Cross compile the source code with the debug codes embedded within it. Burn the corres. hex file into the EEPROM. You can view the printf() generated data on the HyperTerminal. If the firmware is error free and the exⁿ occurs properly, you will get all the debug msgs on the HyperTerminal.

Eg: 11 First Inline Debug Code

```
printf("Starting Config"...);  
Configurations ...  
- - -
```

```
// inline debug code ensuring exn  
printf("End of Conf");  
printf("Beginning of Firmware exn");  
Code Segment 1 - - -
```

```
printf("End of code segment 1");  
- - -
```

Target Debug: HyperTerminal
Starting Config" - - -
End of Config"
Beginning of Firmware ex ⁿ
End of Code segment 1

Monitor Program Based Firmware Debugging:

Monitor pgm which acts as a supervisor is developed. The monitor pgm

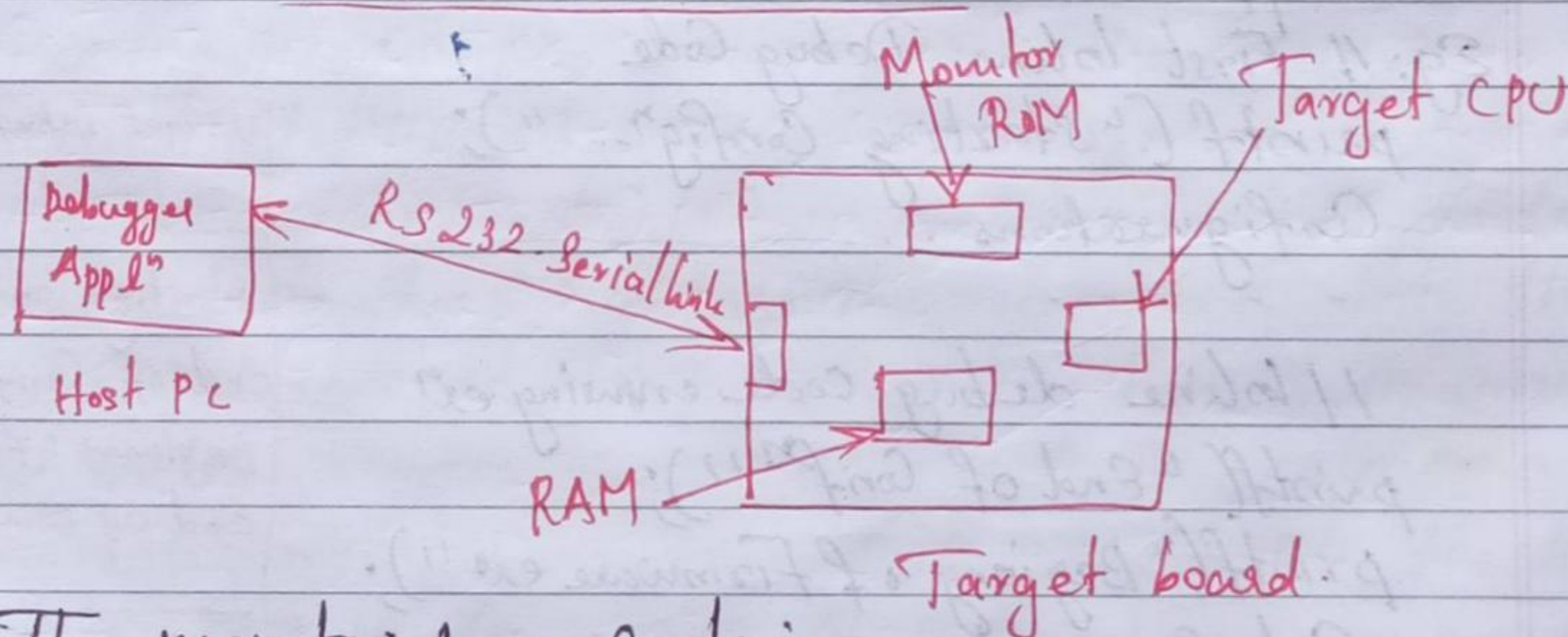
- controls the downloading of user code into the code memory

- inspects & modifies reg/memory locⁿs

- implements debugging fns as per pre-defined cmd set from the debug applⁿ i/f.

All the above fns are done acc. to the cmd received from the serial i/f.

The entire code handling the command reception & corres. action implementation is known as the monitor pgm. The most type of i/f b/w target board & debug applⁿ is RS-232C serial i/f. After successful completion of the monitor pgm, devpt, it is compiled & burned into flash memory or ROM of the target board. The code memory containing the monitor pgm is known as the 'Monitor ROM'.



The monitor pgm contains:

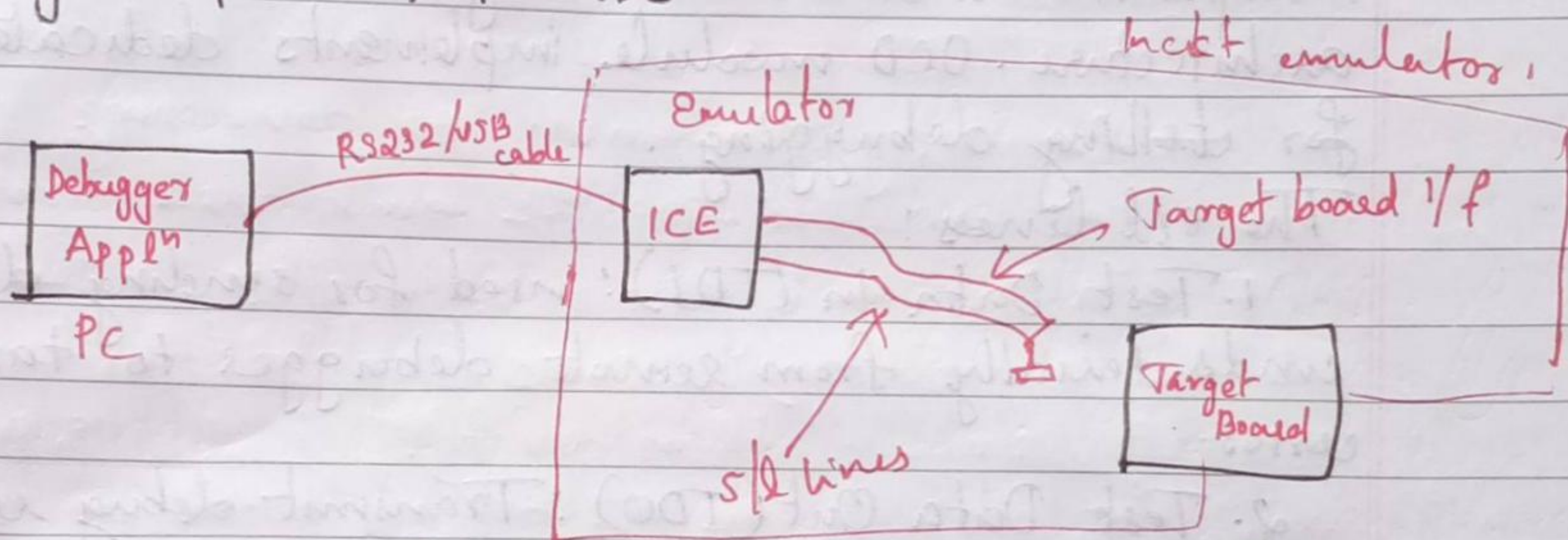
- Cmd i/f to establish commⁿ with the debugging applⁿ.
- Firmware download option to code memory
- Examine & modify processor regs. & working memory.
- Single step pgm exⁿ.
- Set break pts in firmware exⁿ.
- Send debug infⁿ to debug applⁿ running on host m/c.

Drawbacks: shrinks the total available memory to 64k mem. & it needs to accommodate all kinds of mem. reqt.

- Hence the serial port of the target processor becomes dedicated for the monitor applⁿ & it cannot be used for any other device / thing.

In Circuit Emulator (ICE) Based Firmware Debugging

A h/w emulator is called by a debugger applⁿ running on the devpt. PC. The debugger applⁿ may be part of the IDE



The emulator s/m contains the follo. Inal units.

- 1) Emulation Device: replica of the target CPU which receives various s/l's from the target board through a device adaptor connected to the target board & performs the applⁿ exⁿ of f/w under the ctrl of debug cmds from the debug applⁿ. The emulation device can be either a standard chip same as the target processor or a Programmable Logic Device (PLD), configured to fn as the target CPU.
- 2) Emulation Memory: It is a RAM in the Emulator Device. The original EEPROM memory is emulated by the RAM of emulator. This is known as 'ROM Emulation'. It acts as a trace buffer in debugging. Trace buffer is a memory pool holding the instⁿs executed by the processor while debugging.

3. Emulator chkl logic: is the logic ckts used for implementing complex h/w breakpts, trace buffer trigger detection, trace buffer pool chkl.

4. Device Adaptors: act as an iff b/w the target board & emulator. They are normally pin to pin compatible sockets which can be inserted into the target board.

On Chip Firmware Debugging (OCD)

Processors/chklers with OCD support incorporate a dedicated debug module to the existing architecture. OCD module implements dedicated regs. for stalling debugging.

The s/l lines:

1. Test Data In (TDI): used for sending debug cmds serially from remote debugger to target processor.

2. Test Data Out (TDO): Transmit debug response to the remote debugger from target CPU.

3. Test Clock (TCK): synchronises the serial data transfer

4. Test Mode Select (TMS): sets the mode of testing

5. Test Reset (TRST): It is an optional s/l line used for resetting the target CPU.

Target Hardware Debugging

→ various h/w related reasons:

* dry soldering of components

* missing connⁿs in PCB

* misplaced components

* s/l corruption due to noise

The various h/w debugging tools used in Embedded Pdt Despt

1. **Magnifying Glass:** to view minute components inside the watch in an enlarged manner so that they can easily work with them. It's a visual inspection tool. With magnifying glass, the surface of the target board can be examined thoroughly for dry soldering of components, missing components etc.

2. **Multimeter:** for measuring various electrical quantities like voltage, current, resistance, capacitance, transistor, cathode, & anode identification of diodes etc.

3. **Digital CRO:** Cathode Ray Oscilloscope used for waveform capturing & analysis, measurement of s/l strength etc. - analysing interference noise in the power supply line & other s/l lines.

4. **Logic Analyser:** for capturing digital data (1 & 0) from a digital ckt. It contains special connectors & clips which can be attached to the target board for capturing digital data.

5. **Function Generator:** an i/p s/l simulator tool. - producing various periodic waveforms like sine wave, square wave, saw tooth wave etc. with diff. frequencies & amplitude.

The Embedded S/m Development Env't:

Ref Fig 13.1

The devpt. env't. consists of a Devpt Computer or Host which acts as the heart of the devpt. env't., Integrated Devpt. Env't (IDE) Tool for embedded firmware devpt & debugging, Electronic Design Automation (EDA) Tool for Embedded h/w design, An emulator h/w for debugging the target board, Signal sources for simulating the i/ps to the target board, Target h/w debugging tools & target h/w.

Integrated Devpt Env't. (IDE)

— An integrated ^{Devpt} env't. for developing & debugging the target processor specific embedded firmware.

IDE is a s/w package which bundles a 'Text Editor' (Source Code Editor), 'Cross Compiler' (for cross platform devpt & same platform devpt), Linker & Debugger.

Some IDEs may provide i/f to target board emulators, Target processor's Flash memory programmer etc.

incorporate other s/w devpt. utilities like version ctrl Tool, Help file. IDEs can be either command line based or GUI based. GUI based IDEs provide a Visual Devpt Env't with mouse click support.

MPLAB is an IDE tool, Keil uVision 3, Code Warrior

Keil uVision 3 IDE for 8051: 8051 family Mettel based embedded firmware devpt.

→ It contains various menu options, a project window showing files, Reg. view, Books & Functions Tab & an o/p window

Features

- Create New Project, Save, pop up dialogs Bosc
- Target CPU Vendor selection
- Target CPU selection
- Startup file addition to the project.
- Writing the first Embedded 'C' Code.
- Adding files to the Project.
- Output File creation settings.
- List File generation settings
- Firmware Debugging options
- Target h/w debug serial link config's.
- Target Flash Memory pgming Config's
- Conversion of the Embedded C Pgm to 8051 M/c code.
- Linking all objt files.
- Cross Compilation.
- Breakpoint insertion & Debugging.

1) It has a facility for defining a processor family as well as defining its version. It has source code engg. tools which incorporate the editor, compiler for C; .

2) It has the facility of a user-definable assembler to support a new version or type of processor. provides multi-user envt.

3) The design process divides into no. of subparts. Each pgmer is assigned independent but linked tasks.

4) It simulates h/w unit like emulator, peripherals & I/O devices on a host s/m. It supports conditional & unconditional break points.

5) It debugs by single stepping, synchronizing the int. peripherals.

6) It provides Windows on the screen.

7) It verifies the performance of a target s/m.

MODULE V

- Operating S/m Basics
- Types of O.S
- Task Scheduling
- How to choose an RTOS.

Question Bank:

1. What is monolithic & microkernel? Which one is widely used in RTOS?
2. What is the diff. b/w GP Kernel & RT Kernel?
Eg:
3. Explain fns of RT Kernel.
4. What is ~~the~~ task control block. Structure of TCB.
5. Differentiate b/w Hard & Soft Real Time S/ms?
Give Eg:
6. Explain task & process in O.S.
8. What is task scheduling in the O.S?
9. Different queues associated with process scheduling.
10. Diff. types of non preemptive scheduling Algs.
Merits & demerits of each.
11. Explain preemptive scheduling Algs?
12. Explain RR scheduling.
13. Explain starvation in scheduling.
14. 3 processes with process IDs P_1, P_2, P_3 with exⁿ time 12, 10, 2 ms. respectively enters the ready queue together in the order P_2, P_3, P_1 . Process P_4 with estimated exⁿ completion time 4ms enters the ready

queue after 8ms. Calculate waiting time & TAT for each process & Avg also in the FIFO schedule.

15

Explain the diff. final & non final reqmts need to be evaluated in the selection of an RTOS.

Date: _____

> Types of Operating S/m:

→ Depending on the type of kernel & kernel services, purpose & type of computing S/m's, O.S are classified into diff. types.

1. General Purpose Operating System (GPOS):

→ Kernel is more generalised & it contains all kinds of services required for executing generic appl's. Their services can inject random delays into appl' s/w & may cause slow responsiveness of an appl' at unexpected times., windows XP/MS-DOS etc eg:

2. Real Time Operating Systems (RTOS)

'Real Time' implies deterministic timing behaviour. Deterministic timing behaviour in RTOS context means the O.S services consumes only known & expected amounts of time regardless the no. of services. RTOS implements policies & rules concerning time critical allocation of a S/m's resources.

Eg: Windows CE, QNX, VxWorks etc.

The Real Time Kernel:

Real Time Kernel is specialised & it contains only the minimal set of services required for running the user appl's. The basic fns of a Real-Time Kernel are:

- * Task/process Mgmt
- * Task/process Scheduling
- * Task/process Synchronization
- * Error/Exception Handling
- * Memory Mgmt,
- * Interrupt Handling
- * Time Mgmt.

Task/Process Mgmt: Deals with setting up the memory space for the tasks, loading the task's code into mem. space, allocating S/m resources, setting Task Control Block (TCB).

TCB contains: Task ID, Task state, Task type (hard/soft rt), Task priority, Task Context Pointer, Task Memory ptr. (code mem., data mem., stack memory), Task I/O Resource Ptr, Task Pointers

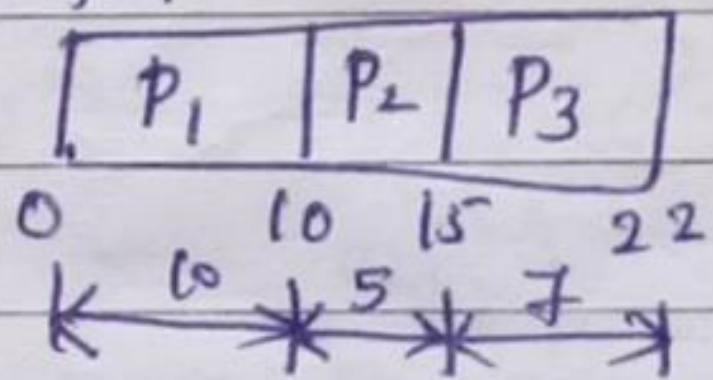
Task Mgmt service utilizes the TCB of a task in the follo. way:

- Creates a TCB for a task on creating a task
- Delete/remove the TCB when task is terminated
- Reads TCB to get state of a task
- Update the TCB with updated parameters on need basis.
- Modify the TCB to change the priority of the task.

Task Scheduling:

FIFO/FCFS Scheduling:

Eg: 3 processes with process IDs P_1, P_2, P_3 with estimated completion time 10, 5, 7 ms respectively enters the ready queue together in the order P_1, P_2, P_3 . Calculate the waiting time & TAT for each process & avg wt and ~~TAT~~ avg TAT.



waiting time for $P_1 = 0$ ms

" " $P_2 = 10$ ms

" " $P_3 = 15$ ms

$$\text{Avg wt} = (0 + 10 + 15) / 3 = 8.33 \text{ ms.}$$

$$\text{Avg wt} = \frac{\text{wt. for all processes}}{\text{No. of processes}}$$

Date: _____

Turn Around Time: Time spent in Ready Queue + Exⁿ Time.

TAT for $P_1 = 10\text{ms}$

$P_2 = 15\text{ms}$ Avg TAT = $(10+15+22)/3 = \underline{\underline{15.66\text{ms}}}$

$P_3 = 22\text{ms}$

Avg: Exⁿ time = (Exⁿ time for all processes) / No. of processes
 $= (10+15+22)/3 = 7.33$

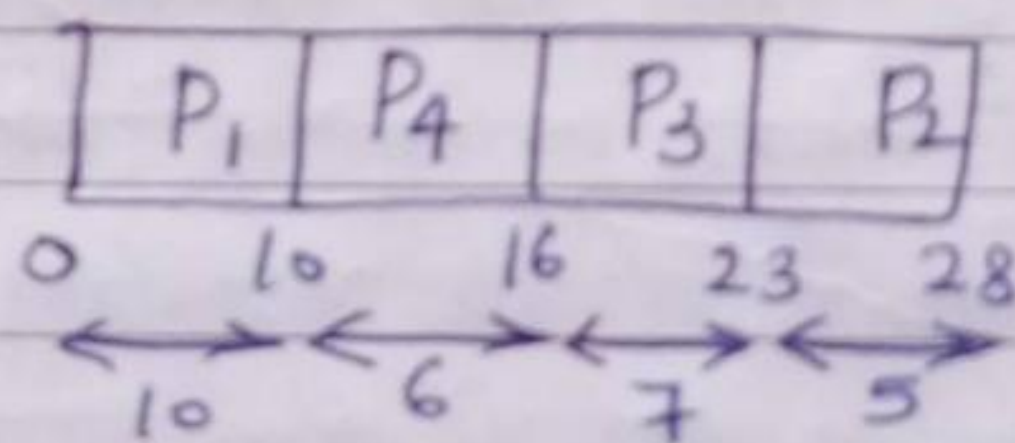
~~Avg Exⁿ~~ Avg TAT = Avg wt. + Avg Exⁿ time

$= 8.33 + 7.33$

$= \underline{\underline{15.66\text{ms}}}$

Last Come Last Serve (LCFS/LIFO) Scheduling:

Eg: 3 processes P_1, P_2, P_3 with estimated completion time 10, 5, 7 ms. enters ready queue in the order P_1, P_2, P_3 . Calculate the waiting time and Turn Around Time for each process & the Avg wt. & TAT. Assume all the processes contain only CPU opⁿ & no I/O opⁿs are involved.



wt. time for $P_1 = 0\text{ms}$

" $P_4 = 5\text{ms}$

" $P_3 = 16\text{ms}$

" $P_2 = 23\text{ms}$

Avg wt = $(0+5+16+23)/4 = 11\text{ms}$.

TAT for $P_1 = 10\text{ms}$

" $P_4 = 11\text{ms}$

" $P_3 = 23\text{ms}$

" $P_2 = 28\text{ms}$

Avg TAT = $(10+11+23+28)/4$

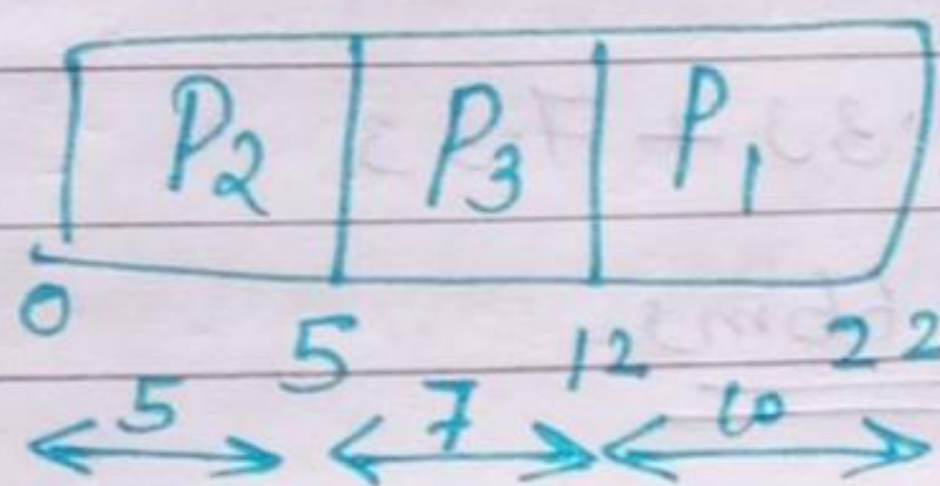
$= \underline{\underline{18\text{ms}}}$

Shortest Job First Scheduling:

In SJF, the process with the shortest estimated run time is scheduled first, followed by next shortest.

Example:

3 processes P_1, P_2, P_3 with est. time 10, 5, 7 ms. respectively enters the ready queue together. Calculate the waiting time & TAT for each processes & Avg. wt. & TAT in SJF Algm.



$$\text{TAT for } P_2 = 5 \text{ ms}$$

$$\text{" } P_3 = 12 \text{ ms}$$

$$\text{" } P_1 = 22 \text{ ms}$$

$$\text{Avg TAT} = \frac{(5 + 12 + 22)}{3} = 13 \text{ ms.}$$

~~wt. time~~ wt. time for $P_2 = 0 \text{ ms}$

wt. time for $P_3 = 5 \text{ ms}$

" $P_1 = 12 \text{ ms}$

$$\text{Avg wt. time} = \frac{(0 + 5 + 12)}{3} = 5.66 \text{ ms}$$

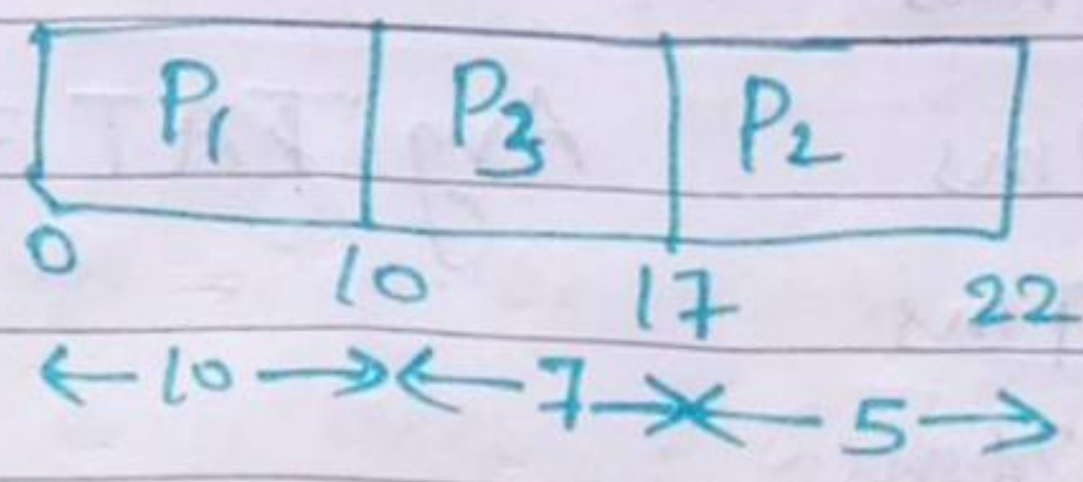
$$\text{Avg Est. time} = \frac{(10 + 5 + 7)}{3} = 7.33$$

$$\text{Avg TAT} = 5.66 + 7.33 = 13 \text{ ms}$$

Comp. Priority Scheduling, ensures that a process with high priority is serviced at the earliest compared to other low priority processes in the Ready Queue.

The priority is a no. ranging from 0 to the max. priority supported by the O.S.

Example: 3 processes P_1, P_2, P_3 with est. time 10, 5, 7 ms & priorities 0, 3, 2 respectively enters the ready queue together. Calculate the wt. & TAT for each process & Avg. wt & TAT in priority based Algm.



TAT for $P_1 = 10\text{ms}$

TAT for $P_3 = 17\text{ms}$

TAT for $P_2 = 22\text{ms}$

Avg TAT = $(10 + 17 + 22) / 3$

$= 16.33\text{ms}$

Wt time for $P_1 = 0\text{ms}$

Wt. time for $P_3 = 10\text{ms}$

Wt. time for $P_2 = 17\text{ms}$

Avg wt. time = $(0 + 10 + 17) / 3 = 9\text{ms}$

Preemptive Scheduling: The scheduler can preempt the currently executing task/process & select another task from the ready queue for exⁿ.

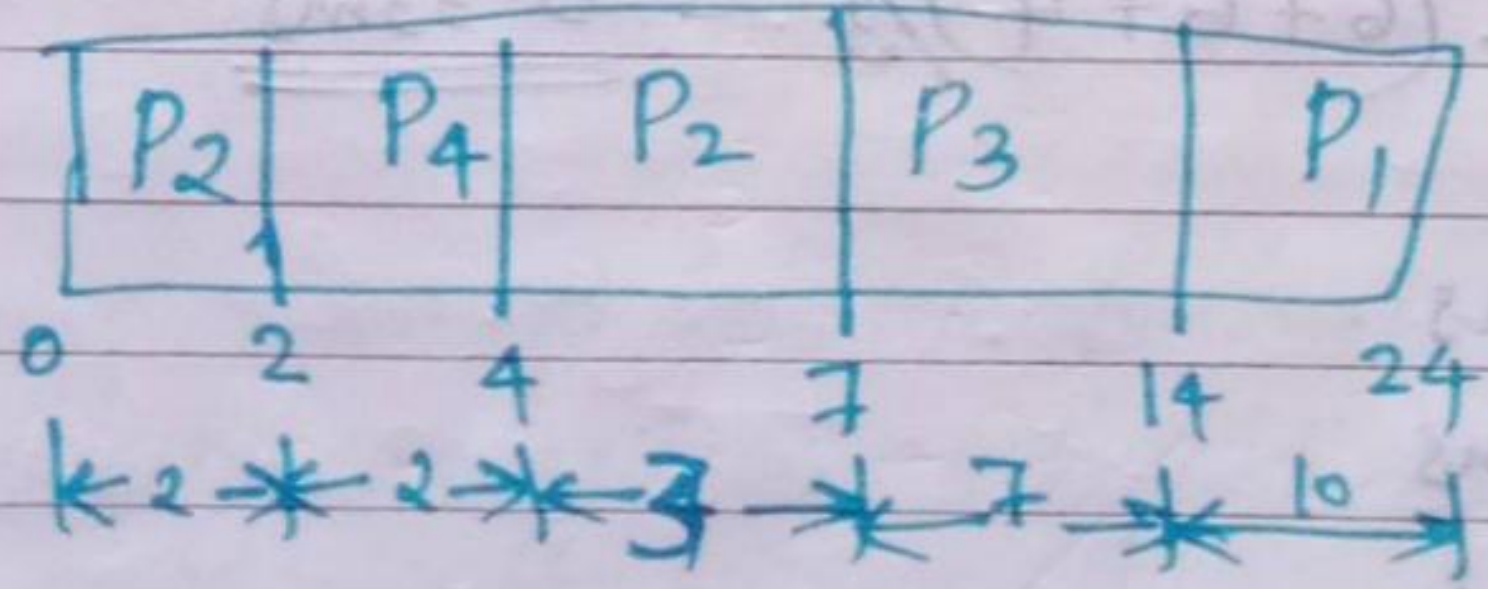
→ time based preemption

→ priority based preemption.

Preemptive SJF Algm (Shortest Remaining Time):

- sorts the Ready queue when a new process enters the 'Ready' queue & checks whether the exⁿ time of the new process is shorter than the remaining of the total estimated time for the currently executing process. If the exⁿ time of the new process is less, the currently executing process is preempted & the new process is scheduled for exⁿ.

Eg: 3 processes with P_1, P_2, P_3 with exⁿ time 10, 5, 7ms enters the ready queue together. A new process P_4 with exⁿ time 2ms enters the ready queue after 2ms.



Wt. time for $P_2 = 0\text{ms} + (2)\text{ms}$

TAT = 2ms.

Wt. time for $P_4 = 0\text{ms}$

Wt. time for $P_3 = 7\text{ms}$

Wt. time for $P_1 = 14\text{ms}$

Avg wt. = $(0 + 2 + 7 + 14) / 4 = 5.75\text{ms}$

$$\text{TAT for } P_2 = 7 \text{ ms}$$

$$\text{TAT for } P_4 = 2 \text{ ms}$$

$$\text{TAT for } P_3 = 14 \text{ ms}$$

$$\text{TAT for } P_1 = 24 \text{ ms}$$

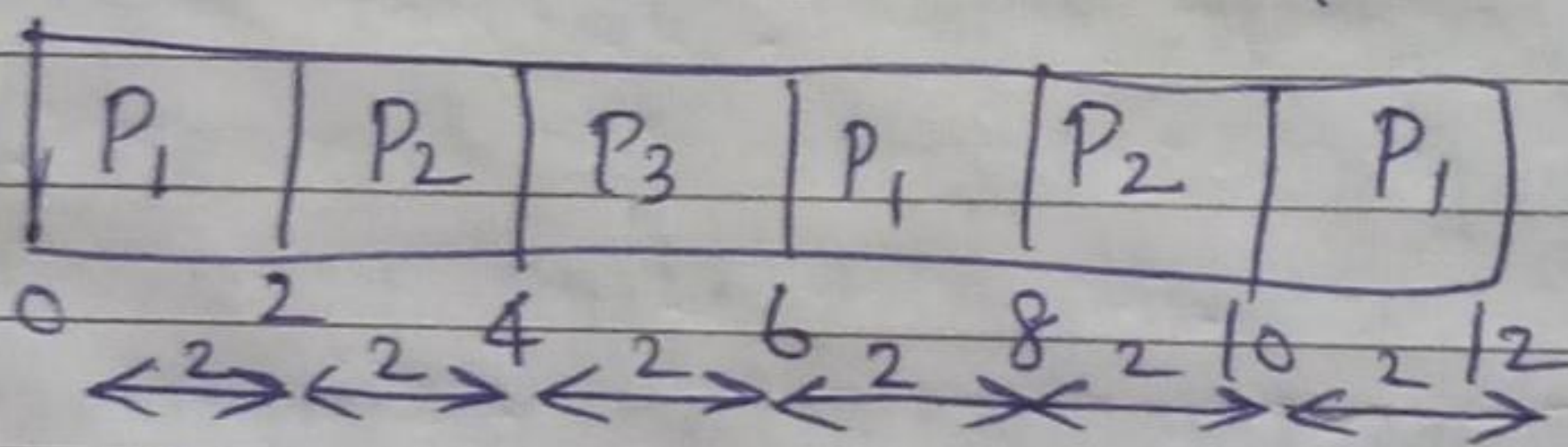
$$\text{Avg TAT} = \frac{(7+2+14+24)}{4}$$

$$= 11.75 \text{ ms}$$

Round Robin Scheduling:

The time slice is provided by the timer tick feature of time mgmt unit of the O-S kernel: R-R scheduling ensures that every process gets a fixed amount of CPU time for exⁿ. If a process terminates before the elapse of the time slice the process releases the CPU voluntarily & next process in the queue is scheduled for exⁿ by the scheduler.

Example: 3 Processes with process IDs P_1, P_2, P_3 with exⁿ time 6, 4, 2ms. respectively, enters the ready queue together in the order P_1, P_2, P_3 . Calculate wt. time & TAT for each process & the avg wt. time & TAT.



$$\text{wt. time for } P_1 = 0 + (6-2) + (10-8) = 6 \text{ ms}$$

$$\text{" } P_2 = 2 + (8-4) = 6 \text{ ms.}$$

$$\text{" } P_3 = 4 \text{ ms}$$

$$\text{Avg wt. time} = \frac{(6+6+4)}{3} = \underline{5.33 \text{ ms}}$$

$$\text{TAT for } P_1 = 12 \text{ ms.}$$

$$\text{" } P_2 = 10 \text{ ms}$$

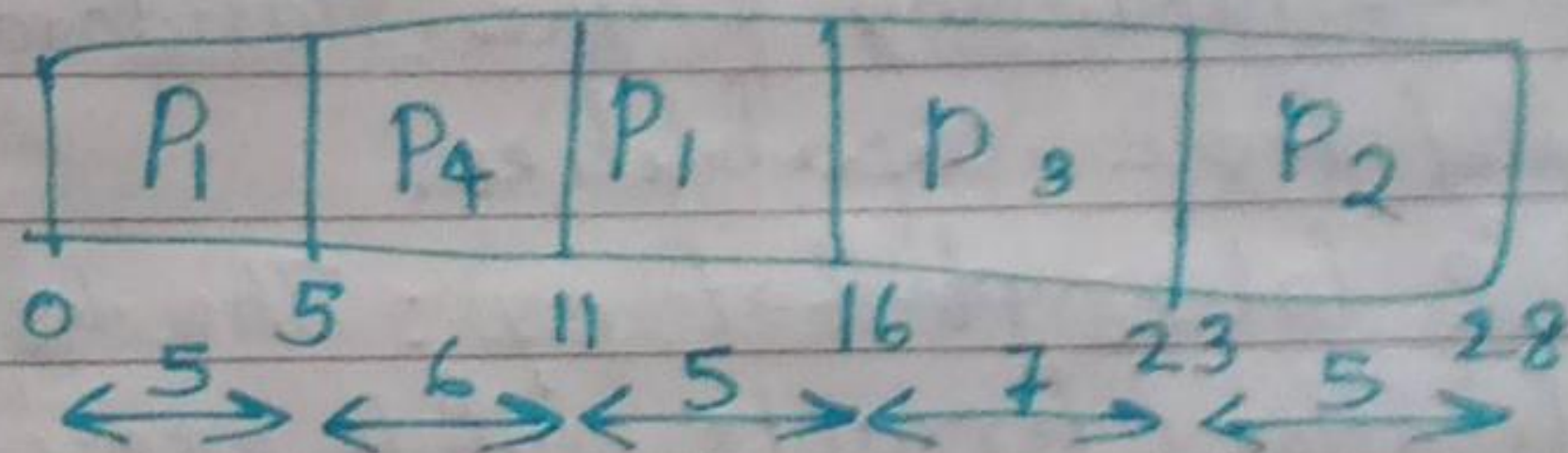
$$\text{" } P_3 = 6 \text{ ms}$$

$$\text{Avg TAT} = \frac{(12+10+6)}{3} = \underline{9.33 \text{ ms}}$$

Priority Based Preemptive Scheduling:

Any high priority process entering the 'Ready' queue is immediately scheduled for exⁿ whereas in the non-preemptive scheduling any high priority process entering the Ready queue is scheduled only after currently executing process completes its exⁿ or only when it voluntarily relinquishes the CPU.

Example: 3 processes P_1, P_2, P_3 with exⁿ time 10, 5, 7 ms & priorities 1, 3, 2 respectively enters the ready queue together. A new process P_4 with exⁿ time 6ms & priority 0 enters the ready queue after 5ms of start of exⁿ of P_1 .



$$\text{wt. time for } P_1 = 0 + (11 - 5) = 0 + 6 = \underline{6 \text{ ms}}$$

$$\text{,, } P_4 = 0 \text{ ms.}$$

$$\text{,, } P_3 = 16 \text{ ms}$$

$$\text{,, } P_2 = 23 \text{ ms}$$

$$\text{Avg wt. time} = \frac{(6 + 0 + 16 + 23)}{4}$$

$$= 11.25 \text{ ms.}$$

$$\text{TAT for } P_1 = 16 \text{ ms}$$

$$\text{TAT for } P_4 = 6 \text{ ms}$$

$$\text{TAT for } P_3 = 23 \text{ ms}$$

$$\text{TAT for } P_2 = 28 \text{ ms}$$

$$\text{Avg TAT} = \frac{(16 + 6 + 23 + 28)}{4}$$

$$= \underline{18.25 \text{ ms}}$$

> How To Choose an RTOS

A lot of factors needs to be analysed before making selection of an RTOS. They can be functional or non-fnal.

Functional Requirements:

Processor Support: It is essential to ensure the processor support by the RTOS.

Memory Rqmts: OS requires ROM memory for holding the O.S files & it is normally stored in a non-volatile memory like flash. Since embedded s/ms are memory constrained, it is essential to evaluate the minimal ROM & RAM reqmts for the O.S under Consideration.

Real Time Capabilities: The task scheduling policies plays an imp. role in the Real-Time behaviour of an O.S. Analyse the real-time capabilities of the O.S under consideration & the standards met by the O.S for real time capabilities.

Kernel & Interrupt Latency: The kernel of the O.S may disable interrupts while executing certain services & it may lead to interrupt latency.

Inter Processes Communication & Task Synchronization: The implementation of Inter process Commⁿ & Synch. is O.S kernel dependant. Certain kernels may provide a bunch of options whereas others provide very limited options.

Modularisation Support: Most of the O.S provide a bunch of features. At times it may not be necessary for an embedded pdt for its functioning. It is useful if the OS supports modularisation where in which the developer can choose the essential modules & re compile the O.S image for functioning.

Support for N/wing & Commⁿ: The OS kernel may provide stack implⁿ & driver support for a bunch of commⁿ i/f's & n/wing.

Development Language Support: Certain O.S include the run time libraries reqd for running applⁿs written in lang. like Java & C#. A JVM customised for O.S is essential for running Java applⁿs.

Non-functional Requirements:

- 1) Custom Developed or off the Shelf: Depending on the OS reqmt, it is possible to go for the complete devpt of an OS suiting the embedded s/m needs or use an off the shelf, readily available O.S, which is either a commercial ptt or an Open Source ptt which is in close match with s/m reqmts.
2. Cost: The total cost for developing or buying the OS & maintaining it in terms of commercial ptt & custom build needs to be evaluated before taking on the selection of OS.
3. Devpt. & Debugging Tools Availability: The availability of devpt & debugging tools is critical decision making factor in the selection of an OS for embedded design.
4. Ease of Use: How easy it is to use a commercial RTOS is another important feature that needs to be considered in the RTOS selection.
5. After Sales: For a Commercial embedded RTOS, after sales in the form of e-mail, on-call services, etc for bug fixes, critical patch updates & support for pain issues etc should be analyzed thoroughly.

Interrupt Handling in RTOS

→ Handling of various types of interrupts.

Interrupts inform the processor that an external device or an associated task requires immediate attention of the CPU. Interrupts

↳ Synchronous - occurs in sync with currently executing task.
↳ Asynchronous

Synchronous Interrupts: occurs in sync with the currently executing task. Eg: slw interrupts.

like Divide by zero, segmentation error etc.

Interrupt handler runs in the same context of the interrupting task.

Asynchronous Interrupts: occurs at any pt. of exⁿ of any task, & are not in sync with the currently executing task. The interrupts generated by external devices connected to the processor/chipset, timer overflow interrupts, serial data transmn interrupts are examples.

For asynch. interrupts, the interrupt handler is usually written as separate task & it runs in a diff. context. A context switch happens while handling the asynchronous interrupts. Priority levels can be assigned to the interrupts & each interrupt can be enabled or disabled individually.

RTOS kernel implements Nested interrupt architecture. It allows the ~~prev~~ preemption of an ISR, servicing an interrupt, by a high priority interrupt.

RTOS & Functions

An RTOS is multitasking O.S for the appl's needing meeting of time deadlines & firing in real time constraints. Real time constraint means constraint on time interval b/w occurrence of an event & s/m expected response to the event.

RTOS Services (Functions)

Basic O.S fns: Process Mgmt, resource mgmt, device mgmt, I/O devices subs/ms & n/w devices & subs/ms mgmt.

Process priority mgmt: user level priorities allocation, called priority allocation. Static priority allocation or real time priority allocation is permitted.

Process Mgmt: preemption: The RTOS kernel preempts a lower priority process when a msg or event for which it is waiting to run higher priority process takes place.

Process priority mgmt: priority inheritance: Priority inheritance enables a shared resource in low priority task.

Process predictability: A predictable timing behaviour of the s/m & a predictable task synchronization with min jitter.

Mem. Mgmt: protection: In RTOS, threads of applⁿ pgms can run in kernel space. The real time performance becomes high. A thread can access the kernel codes, stack & data mem. space, & this could lead to unprotected kernel code.

Mem mgmt. MMU: Either by disabling the use of MMU & virtual mem. or by using mem locks. Memory locking stops the page swapping b/w phy. mem. & disk when MMU is disabled.

Memory Allocation: In RTOS, mem. allocⁿ is fast when there are fixed length mem. block allocations.

RTOS scheduling & Interrupt latency ctrl fns:

Timer fns & time mgmt: provides for timer functions.

There is time allocation & de allocation to attain efficiency in given timing constraints.

Asynch. I/O fns:

IPC synchronization fns: Semaphores, mailboxes, msg queues, pipes, sockets & RPCs.

Spin locks: Spin locks for critical section handling (busy waiting).

Time slicing: of exⁿ of processes which have equal priority.

Hard & soft real time operability:

RTOS Task Scheduling Models:

1. Cooperative Scheduling Model: (FIFO & priority)

→ sequential exⁿ (FIFO)

→ Pr as per precedence (priority)

2. Cyclic & Round Robin Scheduling.

① Cyclic Scheduling: Each of the N tasks in a cyclic scheduler completes in its allotted time frame when the time frame size is based, on the deadline.

A cyclic scheduler is clock driven & is useful for the periodic tasks. It repeats the schedule decided after computations based on the period of occurrences of task instances. Each task has the same priority for exⁿ in the cycle mode.

② Round Robin Time Slicing.

3. Preemptive Scheduling Model:

A disadv. of the cooperative scheduler is that a long exⁿ time of a low priority task makes a high priority task wait at least until it finishes.

A Preemptive: A higher priority task takes control from a lower priority task. A higher priority task switches into running state after blocking the low priority task.

4. Scheduling using earliest deadline first (EDF) Precedence:

Aperiodic task is one in which the period of occurrence is not known because it may not be known when an event can occur. For eg: , an event of receiving a phone call is aperiodic event.

Sporadic task: has periods of bursts when the task events occur.

A deadline is the period in which a task must finish. A task which has a least deadline that is which has little time left for completion, must be scheduled first. This algm of the scheduler is known as EDF algm.

EDF precedence: when a task becomes ready, it will be considered at a scheduling point. The scheduler assigns any priority. It computes the deadline left at a scheduling pt. Scheduling pt is an instance at which scheduler blocks the running task & recomputes the deadlines & runs the EDF algm & finds the task to be run.

An EDF algm can also maintain a priority queue based on the computation when the new task inserts.

5. Rate monotonic Scheduling using 'higher rate of event occurrence First' precedence. (RMS):

RMS computes priorities, p_i from the rate of occurrence of the tasks. Higher priority tasks always get executed.

- it doesn't support aperiodic & sporadic tasks.

6. Fixed (Static) Real-Time Scheduling Model:

Every task is allotted fixed schedules to run. Let there be m tasks & in real time clk interrupts, the scheduler can thus assign each task a fixed schedule. Each task undergoes a ready place to running place transition on the timeouts of the corres. timer.

A scheduler is said to be using a fixed time scheduling method when the schedule is static & deterministic.

An Scheduling Models for Periodic, Sporadic & Aperiodic Tasks.

Performance Metric

1. Ratio of the sum of interrupt latencies w.r. to the sum of exr times.
2. CPU load.
3. Worst case exr time w.r. to the mean exr time.

A preemptive scheduler must take into account 3 types of tasks separately.

1. An aperiodic task needs to be preempted only once.
2. A periodic task needs to be preempted after the fixed periods & it must be executed before its next preemption is needed.
3. A sporadic task needs to be checked for preemption after a minimum time period of its occurrence.

RTOS - Design Principles

1. Design with the ISRs & Tasks: The embedded s/w also sense call generates interrupts. On interrupt, if the interrupt is not masked the interrupt senses the current process context on a stack & executes the ISR context to that interrupt. Interrupts are masked by disable interrupt cmd & unmasked by enable interrupt cmd.

* The ISR can only post the msgs for the RTOS & parameters for the tasks. No ISR instⁿ should block task.

* RTOS provides for nesting of ISRs. This means a running ISR can be interrupted by a higher priority interrupt & higher priority ISR starts executing block the running of low priority ISR.

* A task can wait & take the msgs (IPCs) & post the msgs using the s/w calls.

* A task or ISR should not call another task or ISR. Each ISR has or task has to be under the control of the RTOS.

2. Each ISR Design Consisting of Shorter Code:

As ISRs have higher priorities over the tasks the ISR code should be made short so that task does not wait longer to execute. A design principle that the ISR code should be made optimally short the detailed computations be given to an ISR or task by

3. Design in the form of tasks for the better & Predictable Response time Control: The RTOS provides the control over the response time of diff. tasks. The different tasks are assigned diff. priorities & those tasks which s/m needs to execute with faster response are separated out.
4. Design in the form of tasks for modular design: S/m of multiple tasks makes the design modular. For eg: in a mobile phone device, we consider the user key ip & display as separate tasks.
5. Design in the form of Tasks for Data Encapsulation: S/m of multiple tasks encapsulates the code & data of one task from the other.
6. Design with Taking care of the Time spent in the S/m calls: The ~~exp~~ expected time in general depends on the specific target processor of the embedded s/m & the mem. access times. In order to provide the relative magnitude of the time taken for basic actions at a preemptive scheduler, a new parameter is defined. It defines the time taken for an action by an RTOS scheduler in terms of an assumed scaling parameter S . S emphasizes the relative magnitudes of ex^n times for various actions in a typical RTOS.

7. Design with using Interrupt Service Threads or Tasks:
- for servicing the interrupts, there are 2 levels, fast level ISRs & slow level ISTs, the priorities are first for ISRs, then for ISTs & then the task.
8. Design each task with an infinite loop from start (Idle state) up to finish (Last state): Each task has a while loop which never terminates. The task which gets the s/l runs or takes the IPC for which it is waiting, runs from the pt. where it was blocked or preempted.
9. Limit the no. of tasks & select the appropriate no. of tasks to increase the response time to the tasks, better ctrl over shared & reduced memory syst. for tasks.
10. Use appropriate precedence assignment strategy & use preemption in place of time sharing. The task of higher priority preempts the low priority tasks & ISRs preempt the tasks. The ISRs have higher priorities & over ISTs & tasks.
11. Avoid Task Deletion: Create tasks at start up only & avoid creating & deleting tasks later. The only adv. of deleting is the availability of addⁿ mem. space.
12. Use Idle CPU Time for internal Functions: The CPU may not be running any task. The CPU at that instant may associate the RTOS for the follo. Read the internal queue, Manage the memory, search for free blk of memory. Delete a task, Perform the internal & IPC fns.

3. Design with Memory Allocation and De-Allocation by the task.
If memory allocation & deallocation are done by the task the no. of RTOS functions is ~~is~~ reduced. This reduces interrupt latency periods as exⁿ of these fns takes significant time by RTOS whenever the RTOS preempts a task.

14. Design with taking care of the Shared Resource or Data among the ~~Fast~~ Tasks: The ISR coding should be like a reentrant function or should take care of pbms from the shared resources or data such as buffers or global variables.

15. Design with hierarchical & scalable Limited RTOS fns.
Use an RTOS, which is hierarchical as well as scalable so that has only the needed fns are at the ported section of kernel with the rest left outside.

~~Here~~ Hierarchical RTOS

23/5/19

MODULE VI

> Networked Embedded Systems

Each specific I/O device may be connected to others using specific interfaces. For eg: I/O device connects & interfaced to an LCD controller, keyboard controller or print controller using specific I/Os. Bus communication simplifies the no. of conn's & provides a common protocol for interconnecting diff. or same type of I/O devices.

Any device that is compatible with a s/m's I/O bus can be added to the s/m & a device that is compatible with a particular I/O bus can be integrated into any s/m that uses that type of bus.

This makes s/ms that use I/O buses very flexible; as

The main disadvantage of an I/O bus is that each bus has a fixed bandwidth that must be shared by all the devices, which connect to the bus.

Embedded s/ms connected internally on the same IC or s/ms at very short, short & long distances & can be used using the follo. types of I/O buses, each being acc. to specific distances.

① Using a serial I/O bus allows a computer or controller or embedded s/m to interf. w/ a wide range of I/O devices without having to implement a specific I/O for each I/O device. When the I/O devices in the distributed I/O devices in the distrib. embedded s/ms are used at long distances of 25 cm & above can communicate through a common serial bus. A serial bus has very few lines.

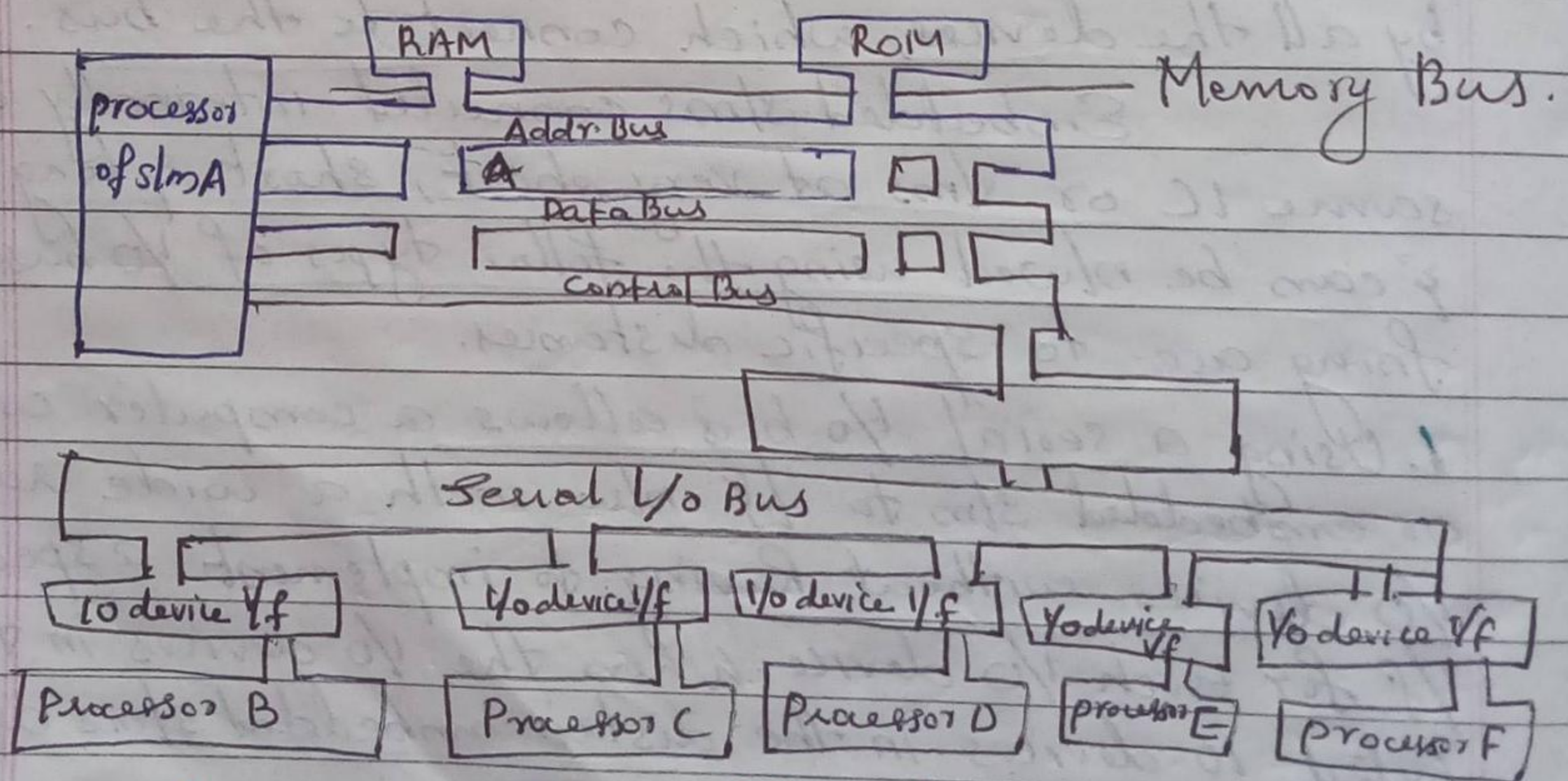
② Using a parallel I/O bus allows a computer or controller or embedded s/m to interf. with a no. of internal s/ms at

very short distances without having to implement a specific I/F for each I/O device. (parallel commⁿ protocols).

③ Using the internet or intranet, a computer, cellular or embedded s/m's I/O device can interface globally & can n/w with other s/ms or computers & a wide range of devices in the distri. s/ms.

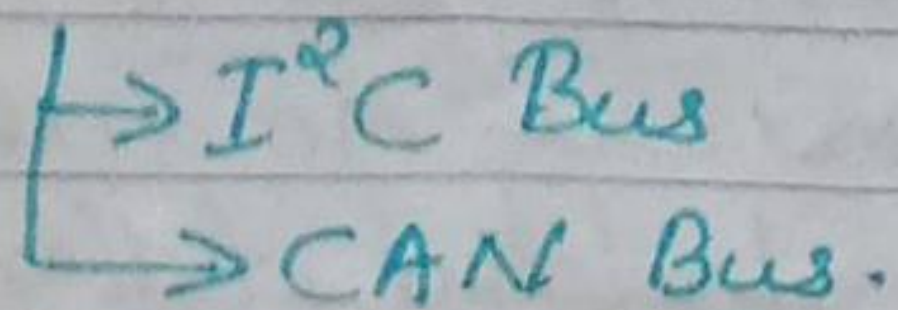
④ Using wireless protocol allows a handheld computer, cellular or embedded s/m I/O device to I/F & n/w with a no. of handheld I/O devices at short distances upto 100m using wireless personal Area N/w.

Embedded s/ms are distributed & n/wed using serial or parallel bus or wireless protocol s/w & appropriate h/w.



A processor of embedded s/m connected to s/m memory bus & n/wed to other s/ms through a serial bus.

Serial Bus Commⁿ Protocols:



I²C Bus:

There are no. of device ckt's in a no. of processes in a plant, one IC each for measuring temperatures & pressures. These ICs mutually n/w through a common synch. serial bus. I²C (Inter IC connect) bus is popular bus for these ckt's.

There are 3 I²C bus standards.

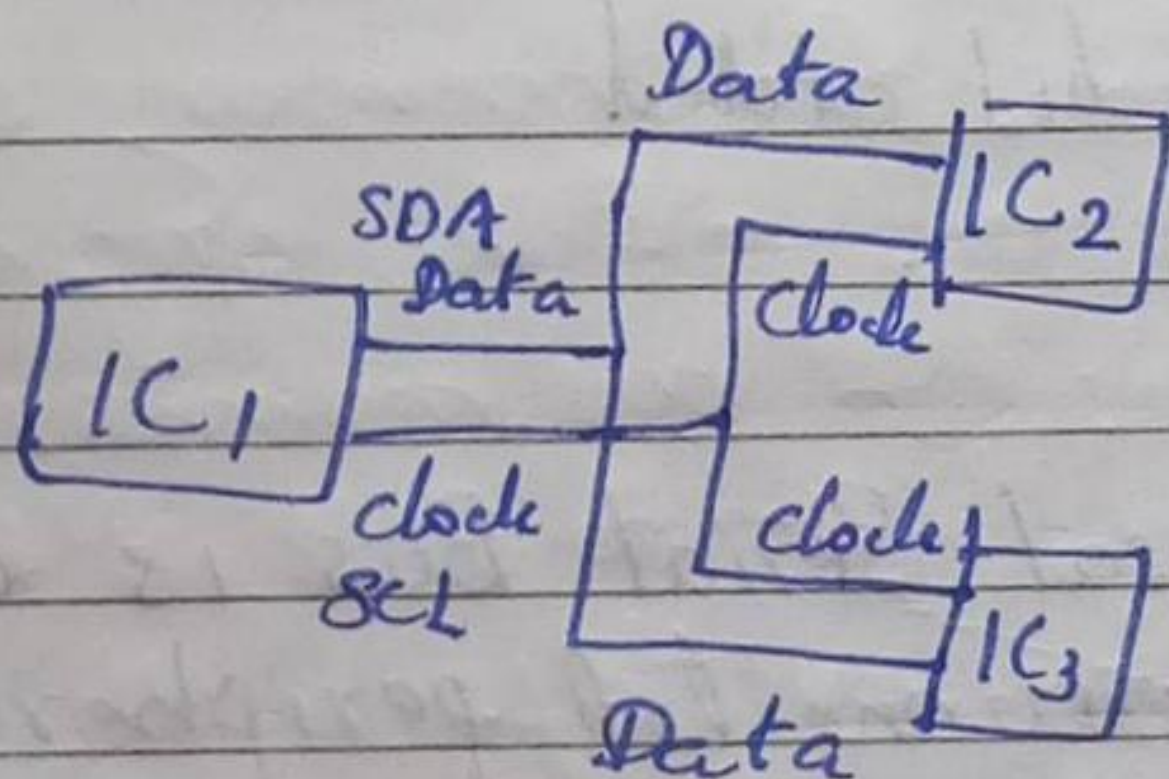
* Industrial 100 kbps I²C

* 100 kbps SM I²C

* 400 kbps I²C.

- developed at Philips Semiconductors.

The I²C bus has 2 lines that carry its s/l's. one line is for clock & one is for bidirectional data.



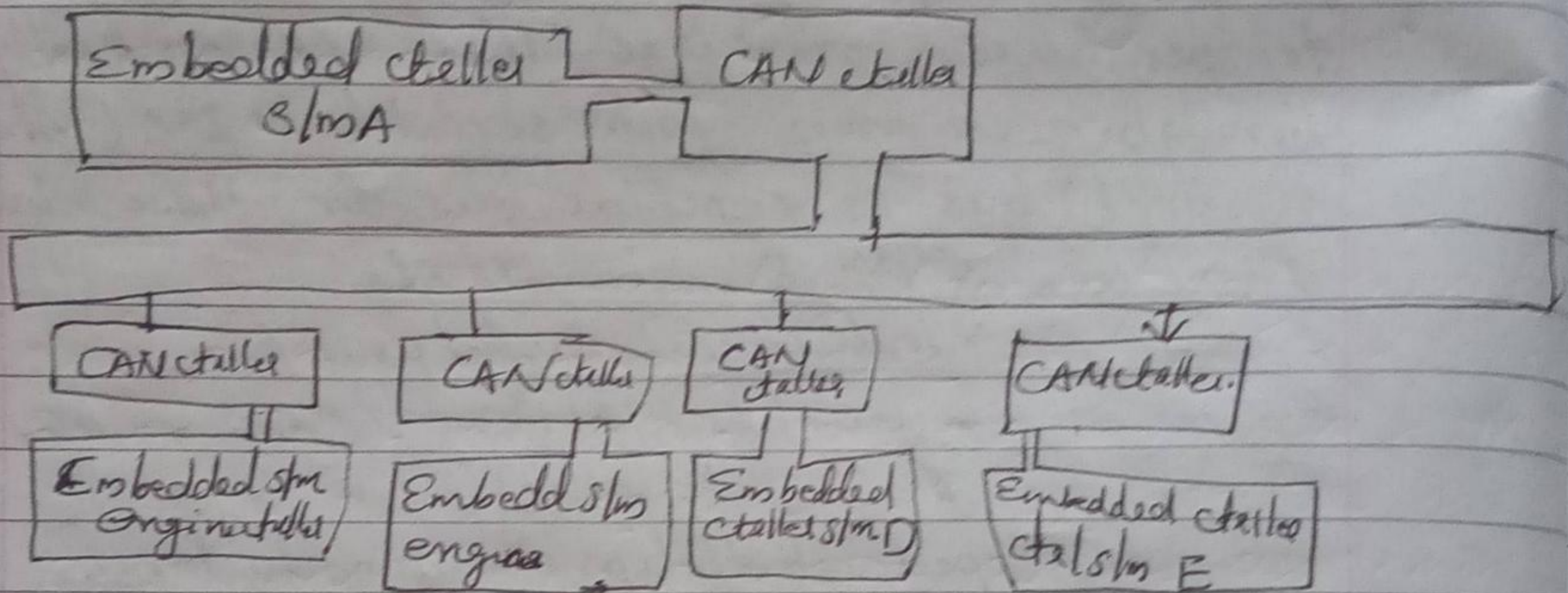
s/l's ~~using~~ during a transfer of a byte when using the I²C

- Start bit
- start addr. bit
- Read/write indicating bit
- Acknowledgment bit
- Data bits
- NACK bit
- stop bit

Format of SDA bits at I²C bus
↓
Serial Data

CAN Bus

No. of devices & controllers are located & distributed in a car. An automobile uses no. of distributed embedded controllers, including those for the brakes, engine, electric power, lamps, A/C, meter display etc. CAN (Controller Area Network) bus is a standard bus in distributed n/w.



* Start bit R to D transition. 3) bit - control field.

* 12 (Recessive) 6 bit coded

0-24

USB Bus.

Universal Serial Bus is a bus b/w host & no. of interconnected peripheral devices. A max. of 127 devices can connect to a host. It provides a fast rate (upto 12Mbps) & as well as a low speed (upto 1.5Mbps) serial transmission & reception b/w host & serial devices. A USB host, which includes controller for fn as bus master can connect flash memory cards, pen like memory devices, digital camera, printer etc. There are 3 standards - USB 1.1 (a low speed 1.5Mbps 3m channel high speed 12Mbps)

- USB 2.0 (high speed 480Mbps 25m channel)
~~Wireless USB~~

- wireless USB (high speed 480Mbps 3m)

→ A USB device can be hot plugged (attached), configured & used, reset, re-configured & used. It can share bandwidth with other devices, detached & re-attached. USB host connects to devices or nodes using USB port driving software & the host controller connected to a root hub.

USB supports 3 types of pipes

1. Stream with no USB defined protocol
2. Default Control providing access.
3. Message for the ctrl fns.

pipes configured for the following:

- a) data bandwidth to be used
- b) transfer service type
- c) buffer sizes.

Fibre Wire: IEEE 1394 Bus Standard

is a high speed 800Mbps serial bus for interconnecting a s/m with multimedia streaming devices & s/ms.

Eg: Digital video cameras, digital cam recorders, DVD, set top boxes etc.

Advanced Serial High Speed Buses:

→ for homehub devices, Cps transceivers serial interfaces.

Parallel Bus Device Protocols: Paralleled communication using ISA, PCI, PCI-X & advanced Buses.

Parallel Bus interconnects I/O devices & peripherals over very short distances & at high speed. ISA, PCI & ARM buses are eg. A parallel bus i/f's the spm memory bus through a bridge or switching ekt.

ISA Bus:

IBM Standard Architecture) connects only to an embedded device that has an 8086 or 80186 or 80286 processor, & in which the processor addressing & IBM PC architecture address limitations & interrupt vector address assignments are taken into account. There is no geographical addressing.

The limitation for memory access by a system using the ISA bus of the original IBM PC was as follows: ISA bus memory accesses can be in two ranges, 640 to 1MB & 15 to 16 MB. The former range also overlaps with the range used by video boards & BIOS.

The I/O port address limitations for devices are as follows: The 8086 & 80286 has 10 mapped I/Os, not memory mapped I/Os.

The following

PCI & PCI/x Buses:

The most used synchronous parallel bus in the computer s/m for i/fing PC based devices is PCI (Peripheral Component Interconnect). PCI provides a superior throughput than EISA. It is almost platform independent, unlike the ISA, which depended on the IBM PC platform, interrupt vectors, IO addresses & memory allocations.

PCI bus has 32 bit data bus extendible to 64 bits. Its protocol specifies the interaction b/w the diff. components of a computer.

A device or host identifies its addr. space by 3 identification no.s. i) IO port, (ii) mem. location iii) config registers of total 256 B with 4 byte unique ID.

ARM Bus:

ARM processor i/fs the memory, ext. DRAM which connect to 32 bit data & 32 bit address lines at high speed using ARM's Memory Bus Architecture.

ARM Bus is of 2 types

→ AMBA-AHB - connects to high speed memory.

→ AMBA-APB connects the ext. peripherals to the s/m mem. bus through a bridge.

Internet Enabled Systems (Page No 170, Raj Kamal)

- Communicating to other s/ms on the Internet.
- use html or MIME type files, TCP or UDP & are addressed by an IP address.
- An IP addr. is of 32 bits or 48 bits in IPv4 or IPv6 respectively. (Ref. fig 3-14 Rajkamal Pg: 171)

Hypertext Transfer Protocol (HTTP)

— An application layer protocol. This layer accepts the data, for eg: in HTML or text format & puts the header words as per the protocol & sends the applⁿ layer header plus data to the transport layer. A port no. specifies the applⁿ in the header.

Eg: NTP, MIME, HTTP, FTP, TELNET, DNS etc.

Features:

- standard protocol for requesting for a URL
- stateless protocol.
- FTP for HTTP.
- Simplicity & flexibility.
- based on OOPS.
- request from a client or response from server consists of 2 parts: A start line, none or several msg.
: Body of the msg
- HTTP provides for entity headers.
- client request server directly or through proxy or a gateway.

Transport Control Protocol (TCP)

— used in transport layer. This layer accepts msgs from the upper layer on transmⁿ by applⁿ or session layer. This layer also accepts a data stream from the n/w layer at receiving end. Before communicating a msg to the next n/w layer, it may add a header. The msg may communicate in parts or segments or fragments.

→ reliable

→ connection oriented

→ Byte oriented

→ Flow ctrl & error ctrl

→ Connⁿ establishment → Data transfer → Disconnect.

User Datagram Protocol (UDP)

— TCP/IP supports at the transport layer.

— Connectionless, stateless

→ supports broadcast networking mode.

→ Header specifies source & destination ports, length, checksum.

→ Stream oriented.

Internet Protocol (IP)

— Communicate using IP.

→ Data → pkts at n/w layer → transmits through a chain of routers on the Internet.

— A packet is a minimum unit of data that transmits on the Internet through routers.

— IPv4 header format (Table 2.11 pg: 174 Rajkumar)

Ethernet

→ about one third of the LANs are Ethernet LANs. & and in each frame, there is a header like in a packet.

→ protocol for local n/w of computers, workstations & devices.

→ Data fragments into the frames.

→ Header - 8 bytes - preamble.

↳ start & used for synch.

- 6 bytes of destⁿ addr.

- 6 bytes of source addr.

- 6 bytes for type field.

- 72-1500 bytes of data length.

Wireless and Mobile System protocols:

→ Infrared Data Association (IrDA)

→ Bluetooth.

→ 802.11

→ Zig Bee

Qub
23/8/19