COURSE FILE

Subject (Name)	:	**Embedded System Design**
Name (of the Faculty Member)	:	**VijayaBhaskerReddy**
Designation	:	**Asst. Prof.**
Regulation /Course Code	:	**R 16/ EC734PE**
Year / Semester	:	**IVth / Ist**
Department	:	**Electronics and Communication Engineering**
Academic Year	:	**2018-19**

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

R-16

EMBEDDED SYSTEM DESIGN

IV B. Tech

2018-19

Course File Prepared by

Vijaya Bhasker Reddy
Assistant Professor

HOD **PRINCIPAL**

COURSE FILE CONTENTS

1. Vision, Mission, PEO's, & PO's, PSOs

Vision

To be recognized as a full-fledged center for learning and research in various fields of Electronics and Communication Engineering through industrial collaboration and provide consultancy for solving the real time problems

Mission

- To inculcate a spirit of research and teach the students about contemporary technologies in Electronics and Communication to meet the growing needs of the industry.

To enhance the practical knowledge of students by implementing projects based on real time problems through industrial collaboration

Programme Educational Objectives (Peo's):

After 3-5 years from the year of graduation, our graduates will,

1. With an exposure to the areas of VLSI, Embedded Systems, Signal / Image Processing, Communications, Wave theory, and Electronic circuits in modern electronics and communications environment.
2. Demonstrate the impact of Electronics and Communications Engineering on the society, ethical, social and professional responsibilities/implications of their work.
3. With strong foundations in mathematical, scientific and engineering fundamentals necessary to formulate, solve and analyze Electronics and communications. Engineering problems.
4. Have strong communication inter-personal skills, multicultural adoptability and to work effectively in multidisciplinary teams.
5. Engage life-long learning to become successful in their professional work.

PEO 1: To inculcate the adaptability skills into the students for design and use of analog and digital circuits and communications and any other allied fields of Electronics.

PEO 2: Ability to understand and analyze engineering issues in a broader perspective with ethical responsibility towards sustainable development.

PEO 3: To develop professional skills in students that prepares them for immediate employment and for lifelong learning in advanced areas of Electronics and communications and related fields.

PEO 4: To equip with skills for solving complex real-world problems related to VLSI, Embedded Systems, Signal/Image processing, Communications, and Wave theory.

PEO 5: Graduates will make valid judgment, synthesize information from a range of sources and communicate them in sound ways in order to find an economically viable solution.

PEO 6: To develop overall personality and character with team spirit, professionalism, integrity, and moral values with the support of humanities, social sciences and physical educational courses.

A graduate of the Electronics and Communication Engineering Program will demonstrate:

PO1: **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2: **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.

PO3: **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.

PO4: **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.

PO5: **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.

PO6: **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.

PO7: **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.

PO8: **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: 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.

PO11: 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.

PO12: 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.

Programme Specific Outcomes (Pso's)

PSO 1: **Problem Solving Skills** – Graduate will be able to apply latest electronics techniques and communications principles for designing of communications systems.

PSO 2: **Professional Skills** – Graduate will be able to develop efficient and effective Communications systems using modern Electronics and Communications engineering techniques.

PSO 3: **Successful Career** – To produce graduates with a solid foundation in Electronics and Communications engineering who will pursue lifelong learning and professional development including post-graduation.

PSO 4: **The Engineer and Society**– Ability to apply the acquired knowledge for the advancement of society and self.

3. COURSE OBJECTIVES

At the end of this course, the student shall be able to:

1. Describe the basics of an embedded system.
2. Develop software and program for an embedded system.
3. Design an embedded system for any type of application.
4. Analyze operating systems concepts, types and choosing RTOS.
5. Design, implement and test an embedded system.

COURSE OUTCOMES

Upon completion of this course, the student will be able to:

CO1. Discuss and design embedded systems.
CO2. Identify the significance of Real Time Operating Systems.
CO3. Analyze the types of memory and interfacing to external hardware.
CO4. Describe embedded firmware design approaches.
CO5. Analyze the issues for development of task communication techniques and device drivers

Topic Outcomes :

Lecture No	TOPIC TO BE COVERED	TOPIC OUTCOME
UNIT –I **Introduction to Embedded Systems**		
L1	Definition of Embedded System, Embedded Systems Vs General Computing Systems	Students are adept at differences between embedded systems and general computing systems.
L2	History of embedded systems and Classification	Students are adept at classification of embedded systems.
L3	Major application areas of embedded systems	Students are able to identify different domains of embedded systems.
L4	Purpose of embedded systems	Students are able to describe the significance of embedded systems,
L5	Characteristics and Quality Attributes of embedded systems	Students are able to analyze the characteristics and parameters to identify the quality of embedded systems.
L6		
L7	Review on Unit –I (Writing the minute note/ Presentation/ Group activity)	Students are able to discuss and demonstrate topics in Unit – I
L8	Activity- Test / Quiz / Assessment (Assignment)	Students are able to evaluate their performance in the Assessment using Grading criteria.
UNIT – II **Typical Embedded System**		
L9	Core of an embedded system	Students are able to analyze the requirements of an embedded system
L10	General purpose and Domain specific processors	Students are able to gain an intuitive understanding of general and domain specific processors.
L11	ASICs and PLDs	Students are able to describe ASICs and PLDs
L12	Commercial Off-The-Shelf Components (COTS)	Students are able to describe COTS
L13	Memory : RAM, ROM	Students are able to describe memory devices
L14	Memory according to the type of interface	Students are able to identify the memory requirement for an interface
L15	Memory shadowing	Students are able to describe memory

		shadowing
L16	Memory selection for embedded systems	Students are able to identify the criterion for selection of memory
L17	Sensors and actuators	Students are able to describe sensors and actuators
L18	Communication interface : On-board	Students are able to describe on-board communication interfaces
L19	Communication interface : External Communication Interfaces	Students are able to describe off-board communication interfaces
L20	Review of Unit – II (Writing the minute note/ Presentation/ Group activity)	Students are able to discuss and demonstrate topics in Unit – II
L21	Activity- Test / Quiz / Assessment (Assignment)	Students are able to evaluate their performance in the Assessment using Grading criteria
UNIT – III **Embedded Firmware**		
L22	Reset Circuit	Students are able to describe reset circuits
L23		
L24	Brown-out protection circuit	Students are able to analyze brown-out protection circuits
L25		
L26	Oscillator unit	Students are able to describe oscillators
L27	Real time clock	Students are able to identify the necessity of real time clock
L28		
L29	Review of Unit – III (Partial) (Writing the minute note/ Presentation/ Group	Students are able to discuss and demonstrate topics in Unit – III (Partial)
L30	Activity- Test / Quiz / Assessment(Assignment)	Students are able to evaluate their performance in the Assessment using Grading criteria
Mid Term 1		
L31	Watchdog timer	Students are able to describe watchdog timers
L32	Embedded firmware design approaches	Students are able to analyze design approaches for embedded firmware
L33		
L34		
L35	Development languages	Students are able to describe languages needed for development of embedded
L36		

L37		software
L38	Review of Unit – III (Writing the minute note/ Presentation/ Group activity)	Students are able to discuss and demonstrate topics in Unit – III
L39	Activity- Test / Quiz / Assessment(Assignment)	Students are able to evaluate their performance in the Assessment using Grading criteria
UNIT – IV **RTOS based Embedded System Design**		
L40	Operating Systems (OS) basics	Students are able to explain the basics of OS
L41	Types of operating systems	Students are able to differentiate the features of different operating systems
L42	Tasks	Students are able to describe tasks of operating system
L43	Processes	Students are able to describe processes of operating system
L44	Threads	Students are able to describe threads of operating system
L45	Multiprocessing and Multitasking	Students are able to explain concepts of multiprocessing and multitasking
L46	Task scheduling	Students are able to analyze scheduling approaches of task management
L47		
L48	Review of Unit – IV (Writing the minute note/ Presentation/ Group activity)	Students are able to discuss and demonstrate topics in Unit – IV
L49	Activity- Test / Quiz / Assessment(Assignment)	Students are able to evaluate their performance in the Assessment using Grading criteria
UNIT – V **Task Communication**		
L50	Shared memory	Students are able to describe concepts of shared memory
L51	Message passing	Students are able to describe message passing
L52	Remote Procedure Call (RPC) AndSockets	Students are able to describe RPCs and Sockets
L53	Task synchronization	Students are able to interpret the considerations for synchronization of tasks
L54	Task communication/ synchronization issues	Students are able to analyze the issues, problems and solutions for task communication/ synchronization

L55	Task synchronization techniques	Students are able to describe different task synchronization techniques
L56		
L57	Device drivers	Students are able to describe the software approaches for design of device drivers
L58		
L59	How to choose RTOS	Students are able to identify the criteria for selection of RTOS
L60		
L61	Review of Unit – V (Writing the minute note/ Presentation/ Group activity)	Students are able to discuss and demonstrate topics in Unit – V
L62	Activity- Test / Quiz / Assessment(Assignment)	Students are able to evaluate their performance in the Assessment using Grading criteria
L63	Review of all units	Students are able to discuss and demonstrate topics in total course
L64	Embedded system design process with examples like automatic chocolate vending machine, smart card etc.	Learn how chocolate vending machines and smart cards are working and
L65	Embedded C programming with some examples.	Write prototypes for various real time task operations
L66	Real time projects in Automotive domain and IOT.	Design their own embedded system devices using IOT
L67		
Mid Term – II		

4. COURSE PRE–REQUISITES

- Microprocessors and Microcontrollers
- C-Programming
- Computer Organization.

5. COURSE INFORMATION SHEET

5. a). COURSE DESCRIPTION:

PROGRAMME: B. Tech.(Embedded system Design)	DEGREE: BTECH
COURSE: Embedded system Design	YEAR: IV SEM: I CREDITS: 4
COURSE CODE: ES REGULATION: R15	COURSE TYPE: CORE
COURSE AREA/DOMAIN: Design	CONTACT HOURS: 4 (L)) hours/Week.
CORRESPONDING LAB COURSE CODE ( ):	LAB COURSE NAME:

5. b). SYLLABUS:

Unit	Details	Hours
I	Introduction to Embedded Systems: Definition of Embedded System, Embedded Systems Vs General Computing Systems, History of Embedded Systems, Classification, Major Application Areas, Purpose of Embedded Systems, Characteristics and Quality Attributes of Embedded Systems.	8
II	Typical Embedded System: Core of the Embedded System: General Purpose and Domain Specific Processors, ASICs, PLDs, Commercial Off- The-Shelf Components (COTS), Memory: ROM, RAM, Memory according to the type of Interface, Memory Shadowing, Memory selection for Embedded Systems, Sensors and Actuators, Communication Interface: Onboard and External Communication Interfaces.	13
III	Embedded Firmware: Reset Circuit, Brown-out Protection Circuit, Oscillator Unit, Real Time Clock, Watchdog Timer, Embedded Firmware Design Approaches and Development Languages.	18
IV	RTOS Based Embedded System Design: Operating System Basics, Types of Operating Systems, Tasks, Process and Threads, Multiprocessing and Multitasking, Task Scheduling.	10
V	Task Communication: Shared Memory, Message Passing, Remote Procedure Call and Sockets, Task Synchronization: Task Communication/ Synchronization Issues, Task Synchronization Techniques, Device Drivers,How to Choose an RTOS.	14
	Contact classes for syllabus coverage	**63**
	Lectures beyond syllabus	**02**
	Tutorial classes	**00**
	Classes for gaps& Add-on classes	**02**
	Total No. of classes	**67**

5.c). GAPS IN THE SYLLABUS - TO MEET INDUSTRY/PROFESSION REQUIREMENTS:

S.NO.	DESCRIPTION	PROPOSED ACTIONS	NO. OF LECTURES
1	Embedded system design process with examples like automatic chocolate vending machine, smart card etc.	CHALK & TALK	1
2	Embedded C programming with some examples.	CHALK & TALK	1

5. d). TOPICS BEYOND SYLLABUS/ ADVANCED TOPICS:

S.NO.	DESCRIPTION	PROPOSED ACTIONS	NO. OF LECTURES
1	Real time projects in Automotive domain and IOT.	PPT	2

5. e). WEB SOURCE REFERENCES:

Sl. No.	Name of book/ website
a.	nptel.ac.in/courses/embedded systems
b.	http://www.embedded.com
c.	An embedded software premier/http://books.google.co.in

5. f). DELIVERY / INSTRUCTIONAL METHODOLOGIES:

☑ CHALK & TALK	☑ STUD. ASSIGNMENT	☑ WEB RESOURCES
☐LCD/SMART BOARDS	☑ STUD. SEMINARS	☐ ADD-ON COURSES

5.g). ASSESSMENT METHODOLOGIES - DIRECT

☑ ASSIGNMENTS	☑ STUD. SEMINARS	☑ TESTS/MODEL EXAMS	☑ UNIV. EXAMINATION

☐ STUD. LAB PRACTICES	☐ STUD. VIVA	☐ MINI/MAJOR PROJECTS	☐ CERTIFICATIONS
☐ ADD-ON COURSES	☐ OTHERS		

5.h). ASSESSMENT METHODOLOGIES - INDIRECT

☑ ASSESSMENT OF COURSE OUTCOMES (BY FEEDBACK, ONCE)	☑ STUDENT FEEDBACK ON FACULTY (TWICE)
☐ASSESSMENT OF MINI/MAJOR PROJECTS BY EXT. EXPERTS	☐ OTHERS

5.i). TEXT / REFERENCE BOOKS:

T/R	BOOK TITLE/AUTHORS/PUBLICATION
Text Book	Introduction to Embedded Systems-Shibu.K.V, Mc Graw Hill.
Reference Book	Embedded Systems by Rajkamal,TMH.
Reference Book	Embedded System Design – Frank Vahid, Tony Givargis, John Willey.
Reference Book	Embedded Systems – Lyla, Pearson,2013.
Reference Book	An Embedded Software Primer – David E,Simon, Pearson Education.

6. MICRO LESSON PLAN

Topic wise Coverage [Micro Lesson Plan]			
Lecture No	**Topic To Be Covered**	**Scheduled Date**	**Actual Date**
UNIT –I Introduction to Embedded Systems			
L1	Definition of Embedded System, Embedded Systems Vs General Computing Systems	2018-07-09	2018-07-09
L2	History of embedded systems and Classification	2018-07-11	2018-07-11
L3	Major application areas of embedded systems	2018-07-12	2018-07-12
L4	Purpose of embedded systems	2018-07-13	2018-07-13
L5	Characteristics and Quality Attributes of embedded systems	2018-07-13	2018-07-13
L6		2018-07-16	2018-07-17
L7	Review on Unit –I (Writing the minute note/ Presentation/ Group activity)	2018-07-17	2018-07-17
L8	Activity- Test / Quiz / Assessment (Assignment)	2018-07-18	2018-07-18
UNIT – II Typical Embedded System			
L9	Core of an embedded system	2018-07-20	2018-07-20
L10	General purpose and Domain specific processors	2018-07-23	2018-07-23
L11	ASICs and PLDs	2018-07-24	2018-07-24
L12	Commercial Off-The-Shelf Components (COTS)	2018-07-25	2018-07-25
L13	Memory : RAM, ROM	2018-07-26	2018-07-26
L14	Memory according to the type of interface	2018-07-27	2018-07-27
L15	Memory shadowing	2018-07-28	2018-07-30

L16	Memory selection for embedded systems	2018-07-31	2018-07-31
L17	Sensors and actuators	2018-08-01	2018-08-01
L18	Communication interface : On-board	2018-08-02	2018-08-02
L19	Communication interface : External Communication Interfaces	2018-08-03	2018-08-03
L20	Review of Unit – II (Writing the minute note/ Presentation/ Group activity)	2018-08-20	2018-08-20
L21	Activity- Test / Quiz / Assessment (Assignment)	2018-08-23	2018-08-23
UNIT – III Embedded Firmware			
L22	Reset Circuit	2018-08-27	2018-08-27
L23		2018-08-28	2018-08-28
L24	Brown-out protection circuit	2018-08-29	2018-08-29
L25		2018-08-30	2018-08-30
L26	Oscillator unit	2018-08-31	2018-08-31
L27	Real time clock	2018-08-31	2018-08-31
L28		2018-09-01	2018-09-01
L29	Review of Unit – III (Partial) (Writing the minute note/ Presentation/ Group	2018-09-04	2018-09-04
L30	Activity- Test / Quiz / Assessment(Assignment)	2018-09-05	2018-09-05
Mid Term 1			
L31	Watchdog timer	2018-09-07	2018-09-07
L32	Embedded firmware design approaches	2018-09-10	2018-09-10
L33		2018-09-12	2018-09-12
L34		2018-09-14	2018-09-14

L35	Development languages	2018-09-15	2018-09-15
L36		2018-09-17	2018-09-17
L37		2018-09-18	2018-09-18
L38	Review of Unit – III (Writing the minute note/ Presentation/ Group activity)	2018-09-18	2018-09-18
L39	Activity- Test / Quiz / Assessment(Assignment)	2018-09-20	2018-09-20
UNIT – IV RTOS based Embedded System Design			
L40	Operating Systems (OS) basics	2018-09-22	2018-09-22
L41	Types of operating systems	2018-09-24	2018-09-24
L42	Tasks	2018-09-24	2018-09-24
L43	Processes	2018-09-24	2018-09-24
L44	Threads	2018-09-25	2018-09-25
L45	Multiprocessing and Multitasking	2018-09-25	2018-09-25
L46	Task scheduling	2018-09-28	2018-09-28
L47		2018-09-28	2018-09-28
L48	Review of Unit – IV (Writing the minute note/ Presentation/ Group activity)	2018-10-03	2018-10-03
L49	Activity- Test / Quiz / Assessment(Assignment)	2018-10-04	2018-10-04
UNIT – V Task Communication			
L50	Shared memory	2018-10-04	2018-10-04
L51	Message passing	2018-10-05	2018-10-05
L52	Remote Procedure Call (RPC) and Sockets	2018-10-12	2018-10-12
L53	Task synchronization	2018-10-26	2018-10-26

L54	Task communication/ synchronization issues	2018-10-26	2018-10-26
L55	Task synchronization techniques	2018-10-27	2018-10-27
L56		2018-10-27	2018-10-27
L57	Device drivers	2018-10-29	2018-10-29
L58		2018-10-30	2018-10-30
L59	How to choose RTOS	2018-10-30	2018-10-30
L60		2018-11-01	2018-11-01
L61	Review of Unit – V (Writing the minute note/ Presentation/ Group activity)	2018-11-05	2018-11-05
L62	Activity- Test / Quiz / Assessment(Assignment)	2018-11-05	2018-11-05
L63	Review of all units	2018-11-06	2018-11-06
L64	Embedded system design process with examples like automatic chocolate vending machine, smart card etc.	2018-11-09	2018-11-09
L65	Embedded C programming with some examples.	2018-09-18	2018-09-18
L66	Real time projects in Automotive domain and IOT.	2018-09-26	2018-09-26
L67			
Mid Term – II			

6. TEACHING REFERENCE PLAN

Subject	Embedded system design					
Text Books (to be purchased by the Students)						
Book 1	Introduction to Embedded Systems by Shibu K.V					
Reference Books						
Book 2	Embedded Systems by Raj Kamal					
Book 3	An Embedded Software Primer by David E. Simon					
Unit	**Topic**	**Chapters Nos**				**No of classes**
		Book 1	**Book 2**	***Book 3***	***Book 4***	
I	Introduction to Embedded Systems And purpose of embedded system	1	1	1		3
	Embedded Systems Vs General Computing Systems	1	1			3
	Characteristics and Quality Attributes of Embedded Systems	3	1			2
II	Core of the Embedded System	2	1	1		4
	Memory: ROM, RAM	2	2	3		3
	Sensors and actuators	2				2
	Communication interfaces	2	3			4
III	Reset circuit, oscillator circuit	2	3			6
	Embedded Firmware Design Approaches	9	5	9		5
	Embedded Firmware Development Languages	9	5	9		7
IV	Operating System Basics	10	8			3
	Tasks	10	7	6		3
	Task Scheduling	10	7			4
V	Task communication	10	7			5
	Task synchronization	10	7	7		5
	Device drivers	10	7		2	4
Contact classes for syllabus coverage						**63**
Tutorial classes						**00**
Lectures beyond syllabus						**02**
Classes for gaps& Add-on classes						**02**
Total No. of classes						**67**

11. MID exam Descriptive Question Papers

K. G. Reddy College of Engineering &Technology
(Approved by AICTE, Affiliated to JNTUH)
Chilkur (Vil), Moinabad (Mdl), RR District

Name of the Exam: I Mid Examinations **September– 2019**
Year-Sem & Branch: IV-I & ECE **Duration: 60 Min**
Subject: Embedded System Design **Date & Session:**

Answer **ANY TWO** of the following Questions 2**X5=10**

Q.NO	QUESTION	Bloom's level	Course outcome
1	a) Define Embedded System. Give few examples. b) Explain in detail the classification of embedded systems.	L1,L2	CO1
2	a) Differentiate between general purpose processors and domain specific processors. b) Discuss the role of sensors in embedded system design.	L2,L4	CO1,CO3
3	a) Briefly explain Brown-out protection circuit. b) Explain the function of Watchdog timer in an embedded system	L2	CO1,CO3
4	Discuss about given terms in an Embedded System a) Portability b) Power concerns c) Reactive and Real-time d) Reliability	L2	CO1

K. G. Reddy College of Engineering &Technology (Approved by AICTE, Affiliated to JNTUH) Chilkur (Vil), Moinabad (Mdl), RR District			College Code QM	Regulation R15
Name of the Exam:	**II Mid Examinations NOV-2019**		**Marks:**	**10**
Year-Sem & Branch:	**IV-I ECE**	**Duration:**	**60 Min**	
Subject:	**ESD**	**Date & Session**		

Answer **ANY TWO** of the following Questions 2**X5=10**

Q.NO	QUESTION	Bloom's level	Course outcome
1	Classify and list various TASK scheduling algorithms. Discuss Non-preemptive SJF algorithm with an example	Understand	CO2
2	Distinguish embedded Firmware development approaches and explain.	Analyze	CO4
3	Explain how to choose an RTOS. Discuss about Device Drivers	Understand	CO5
4	Discuss about Dead lock and Live lock TASK Synchronization Issues with examples	Understand	CO5

12. MID exam Objective Question papers

SUBJECT CODE: EC734PE SET NO. 1

KG REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Chilkur (Vill) Moinabad (Mdl) R R Dist

B.TECH IV Year I SEM I MID Term Examinations, SEP-2019 EMBEDDED SYSTEM DESIGN

OBJECTIVE EXAM

NAME_____ HALL TICKET NO | | | **Q** | **M** | | **A** | | | | |

Answer all the questions. All questions carry equal marks. Time: 20min. 10 marks. I choose correct alternative:

1.	What kind of visual panel is used for seven segmented display?			[ ]
A.LCD	B.LED	C.BINARY OUTPUT	D.ANALOG OUTPUT	
2.	Which of the following is one time programmable memory			[ ]
A.SRAM	B.PROM	C.FLASH	D. NVRAM	
3.	Which of the following is not an embedded system			[ ]
A. Smartphone	B. Digital Camera	C. MP3 Player	D. Desktop Computer	
4.	It retains its content when power is removed. What type of memory is this?			[ ]
A. Volatile memory	B. Nonvolatile memory	C.RAM	D.SRAM	
5.	Electronic toy is an example of			[ ]
A. Small scale embedded system	B. Medium scale embedded system	C. Large scale embedded system	D. None	
6.	A Digital CRO is example of an Embedded system for _____			[ ]
A. Communication	B. Control	C.Monitoring	D. All of these	
7.	Which of these are an intended purpose of an Embedded system			[ ]
A. Data Collection	B. Monitoring	C. Data processing	D. All of these	
8.	Which of the processor architecture supports easier instruction pipelining			[ ]
A. Von Neumen	B. Harvard	C. Both	D. None	

9.	_____is the place holder for arranging different hardware components required to build embedded product			[ ]
A.PAB	B. DAB	C.PCB	D. None	
10.	The accuracy of crystal oscillator is normally expressed in terms of _____			[ ]
A.prm	B.ppm	C.pam	D.pwm	

ount…..2

SUBJECT CODE: EC734PE SET NO. 1

II Fill in the Blanks:

1.	Embedded System is a combination of_____and _____
2.	Response is a measure of _____
3.	The first recognized modern embedded system is _____
4.	COTS stands for_____
5.	ASIC stands for _____
6.	Embedded communication interface is classified as_____and _____
7.	_____Number of such devices can be connected to a USB host
8.	RTC acronym_____
9.	Examples of on board communication Interface are_____and _____
10.	Memory cells present in 1kb of RAM is _____

Key Paper SET-1 SEM-I, MID-I

Embedded System Design

MCQ's

1. A
2. B
3. D
4. A
5. A
6. C
7. D
8. B
9. C
10. B

Fill in the Blanks

1. Hardware, Software.
2. Quickness of the System.
3. Apollo Guidance Computer(AGC)
4. Commercial of the Shelf Components.
5. Application Specific Integrated Circuit.
6. Onboard, External.
7. 127
8. Real Time clock.
9. SPI, 1-Wire, 2-Wire, I^2C etc...
10. 8192(1024*8)

SUBJECT CODE: EC734PE SET NO. 1

KG REDDY COLLEGE OF ENGINEERING & TECHNOLOGY

Chilkur (Vill) Moinabad (Mdl) R R Dist

B.TECH IV Year I SEM II MID Term Examinations, NOV-2019

EMBEDDED SYSTEM DESIGN

OBJECTIVE EXAM

NAME_____**HALL TICKET NO**

		Q	M		A				

Answer all the questions. All questions carry equal marks. Time: 20min. 10 marks.

I choose correct alternative:

1.	Which of the following is(are) example(s) of RTOS			[ ]
A. Windows XP	B. Windows 2000	C. Windows CE	D. All of these	
2.	The memory area which holds the program corresponding to the core OS is			[ ]
A. User space	B. kernel space	C. Shared memory	D. None of these	
3.	Multiple threads of the same process share			[ ]
A. Data memory	B. Code memory	C. Stack memory	D. Only a & b	
4.	A process switch from Running to Ready state due to_____scheduling act			[ ]
A. Co-operative	B. Pre-emptive	C. Non Pre-emptive	D. None of these	
5.	The implementation pipe is_____dependent			[ ]
A. Processor	B. OS	C. IPC	D. All of these	
6.	For a good scheduling algorithm the response time should be			[ ]
A. Minimum	B. Maximum	C. Average	D. Varying	
7.	_____is a piece of software that acts as an bridge between the operating system and the hardware			[ ]
A. Compiler	B. Linker	C. Device driver	D. FIFO	
8.	A common data sharing problem where two processes concurrently access as shared buffer with fixed size is			[ ]
A. LIFO	B. producer-consumer problem	C. Dead lock	D. Racing	

9.	Which of the following technique follow the *sleep & wakeup* mechanism			[ ]
A. Semaphore		B. Critical section	C. Mutex	D. Spin lock
10.	A good scheduling algorithm has			[ ]
A. High cpu utilization		B. medium cpu utilization	C. low cpu utilization	D. All of these

Count…..2

SUBJECT CODE: EC734PE SET NO. 1

II Fill in the Blanks:

1.	A process does not get the CPU or system resources required to continue its execution for a long time is called_____
2.	A_____is primitive that can execute code
3.	A process contains at least_____thread(s)
4.	Various_____requirements need to be evaluated before the selection of an RTOS for an embedded design
5.	Symbols used in machine language for representing the machine language is called as_____
6.	Operating systems with a generalized kernel are known as _____
7.	The consumer tries to read data from an empty buffer is called as _____
8.	Super loop model is also called as _____
9.	The core of the operating system is called_____
10.	The binary semaphore implementation for exclusive resources access under certain OS kernel is called_____

Key Paper SET-1 SEM-I, MID-II

Embedded System Design

MCQ's

1. C

2. B

3. D

4. B

5. B

6. A

7. C

8. D

9. C

10. A

Fill in the Blanks

1. Starvation
2. Thread
3. One
4. Functional and Non functional
5. Opcodes
6. General Purpose Operating System (GPOS)
7. Buffer under Run
8. Conventional method
9. Kernel
10. Mutex

13. ASSIGNMENT TOPICS WITH MATERIALS

UNIT-I

1. Embedded system and Its Applications

An **embedded system** is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints. It is *embedded* as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today. Ninety-eight percent of all microprocessors are manufactured as components of embedded systems

Examples of properties of typical embedded computers when compared with general-purpose counterparts are low power consumption, small size, rugged operating ranges, and low per-unit cost. This comes at the price of limited processing resources, which make them significantly more difficult to program and to interact with. However, by building intelligence mechanisms on top of the hardware, taking advantage of possible existing sensors and the existence of a network of embedded units, one can both optimally manage available resources at the unit and network levels as well as provide augmented functions, well beyond those available. For example, intelligent techniques can be designed to manage power consumption of embedded systems.

Modern embedded systems are often based on microcontrollers (i.e. CPUs with integrated memory or peripheral interfaces), but ordinary microprocessors (using external chips for memory and peripheral interface circuits) are also common, especially in more-complex systems. In either case, the processor(s) used may be types ranging from general purpose to those specialized in certain class of computations, or even custom designed for the application at hand. A common standard class of dedicated processors is the digital signal processor (DSP).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, and largely complex systems like hybrid vehicles, MRI, and avionics. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

Embedded systems are commonly found in consumer, cooking, industrial, automotive, medical, commercial and military applications.

Telecommunications systems employ numerous embedded systems from telephone switches for the network to cell phones at the end user. Computer networking uses dedicated routers and network bridges to route data.

Consumer electronics include MP3 players, mobile phones, video game consoles, digital cameras, GPS receivers, and printers. Household appliances, such as microwave ovens, washing machines and dishwashers, include embedded systems to provide flexibility, efficiency and features. Advanced HVAC systems use networked thermostats to more accurately and

efficiently control temperature that can change by time of day and season. Home automation uses wired- and wireless-networking that can be used to control lights, climate, security, audio/visual, surveillance, etc., all of which use embedded devices for sensing and controlling.

Transportation systems from flight to automobiles increasingly use embedded systems. New airplanes contain advanced avionics such as inertial guidance systems and GPS receivers that also have considerable safety requirements. Various electric motors — brushless DC motors, induction motors and DC motors — use electric/electronic motor controllers. Automobiles, electric vehicles, and hybrid vehicles increasingly use embedded systems to maximize efficiency and reduce pollution. Other automotive safety systems include anti-lock braking system (ABS), Electronic Stability Control (ESC/ESP), traction control (TCS) and automatic four-wheel drive.

Medical equipment uses embedded systems for vital signs monitoring, electronic stethoscopes for amplifying sounds, and various medical imaging (PET, SPECT, CT, and MRI) for non-invasive internal inspections. Embedded systems within medical equipment are often powered by industrial computers.[9]

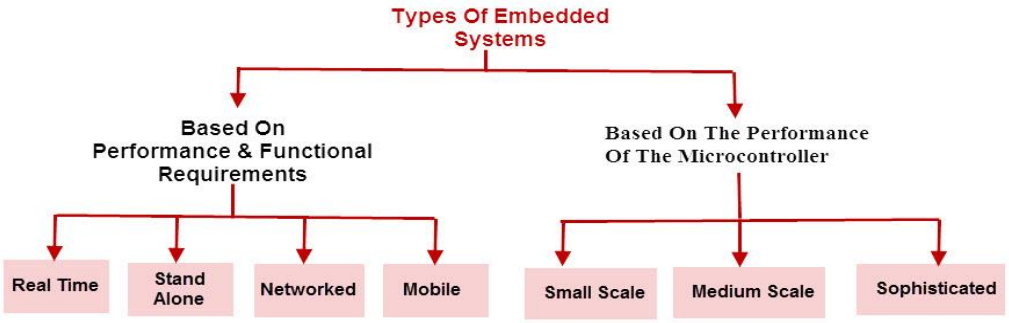
Embedded systems are used in transportation, fire safety, safety and security, medical applications and life critical systems, as these systems can be isolated from hacking and thus, be more reliable, unless connected to wired or wireless networks via on-chip 3G cellular or other methods for IoT monitoring and control purposes.[*citation needed*] For fire safety, the systems can be designed to have greater ability to handle higher temperatures and continue to operate. In dealing with security, the embedded systems can be self-sufficient and be able to deal with cut electrical and communication systems.

A new class of miniature wireless devices called motes are networked wireless sensors. Wireless sensor networking, WSN, makes use of miniaturization made possible by advanced IC design to couple full wireless subsystems to sophisticated sensors, enabling people and companies to measure a myriad of things in the physical world and act on this information through IT monitoring and control systems. These motes are completely self-contained, and will typically run off a battery source for years before the batteries need to be changed or charged.

2. Classifications of embedded systems.

Types of Embedded Systems

Embedded systems can be classified into different types based on performance, functional requirements and performance of the microcontroller.



Types of Embedded systems

Embedded systems are classified into four categories based on their performance and functional requirements:

- Stand alone embedded systems
- Real time embedded systems
- Networked embedded systems
- Mobile embedded systems

Embedded Systems are classified into three types based on the performance of the microcontroller such as

- Small scale embedded systems
- Medium scale embedded systems
- Sophisticated embedded systems

Stand Alone Embedded Systems

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

Real Time Embedded Systems

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

Networked Embedded Systems

These types of embedded systems are related to a network to access the resources. The connected network can be LAN, WAN or the internet. The connection can be any wired or wireless. This type of embedded system is the fastest growing area in embedded system applications. The embedded web server is a type of system wherein all embedded devices are

connected to a web server and accessed and controlled by a web browser.Example for the LAN networked embedded system is a home security system wherein all sensors are connected and run on the protocol TCP/IP

Mobile Embedded Systems

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

Small Scale Embedded Systems

These types of embedded systems are designed with a single 8 or 16-bit microcontroller, that may even be activated by a battery. For developing embedded software for small scale embedded systems, the main programming tools are an editor, assembler, cross assembler and integrated development environment (IDE).

Medium Scale Embedded Systems

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

Sophisticated Embedded Systems

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

3. Characteristics of embedded systems.

Quality Attributes Of Embedded System

These are the attributes that together form the deciding factor about the quality of an embedded system.

There are two types of quality attributes are:-

Operational Quality Attributes.

These are attributes related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.

Non-Operational Quality Attributes.

These are attributes **not** related to operation or functioning of an embedded system. The way an embedded system operates affects its overall quality.

These are the attributes that are associated with the embedded system before it can be put in operation.

Operational Attributes

Response

- Response is a measure of quickness of the system.
- It gives you an idea about how fast your system is tracking the input variables.
- Most of the embedded system demand fast response which should be real-time.

Throughput

Throughput deals with the efficiency of system.

- It can be defined as rate of production or process of a defined process over a stated period of time.
- In case of card reader like the ones used in buses, throughput means how much transaction the reader can perform in a minute or hour or day.

Reliability

- Reliability is a measure of how much percentage you rely upon the proper functioning of the system .
- Mean Time between failures and Mean Time To Repair are terms used in defining system reliability.
- Mean Time between failures can be defined as the average time the system is functioning before a failure occurs.
- Mean time to repair can be defined as the average time the system has spent in repairs.

Maintainability

Maintainability deals with support and maintenance to the end user or a client in case of technical issues and product failures or on the basis of a routine system checkup.

It can be classified into two types:-

Scheduled or Periodic Maintenance

This is the maintenance that is required regularly after a periodic time interval.

Example: Periodic Cleaning of Air Conditioners, Refilling of printer cartridges.

Maintenance to unexpected failure

This involves the maintenance due to a sudden breakdown in the functioning of the system.

Example:

Air conditioner not powering on Printer not taking paper in spite of a full paper stack

Security

- Confidentiality, Integrity and Availability are three corner stones of information security.Confidentiality deals with protection data from unauthorized disclosure.Integrity gives protection from unauthorized modification.Availability gives protection from unauthorized userCertain Embedded systems have to make sure they conform to the security measures.

Ex. An Electronic Safety Deposit Locker can be used only with a pin number like a password.

Safety

Safety deals with the possible damage that can happen to the operating person and environment due to the breakdown of an embedded system or due to the emission of hazardous materials from the embedded products.

A safety analysis is a must in product engineering to evaluate the anticipated damage and determine the best course of action to bring down the consequence of damages to an acceptable level.

Non Operational Attributes

Testability and Debug-ability

- It deals with how easily one can test his/her design, application and by which mean he/she can test it.
- In hardware testing the peripherals and total hardware function in designed manner
- Firmware testing is functioning in expected way
- Debug-ability is means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system

Evolvability

- For embedded system, the qualitative attribute "Evolvability" refer to ease with which the embedded product can be modified to take advantage of new firmware or hardware technology.

Portability

- Portability is measured of "system Independence".
- An embedded product can be called portable if it is capable of performing its operation as it is intended to do in various

  environments irrespective of different processor and or controller and embedded operating systems.

Time to prototype and market

- Time to Market is the time elapsed between the conceptualization of a product and time at which the product is ready for selling or use
- Product prototyping help in reducing time to market.
- Prototyping is an informal kind of rapid product development in which important feature of the under consider are develop.
- In order to shorten the time to prototype, make use of all possible option like use of reuse, off the self component etc.

Per unit and total cost

Cost is an important factor which needs to be carefully monitored. Proper market study and cost benefit analysis should be carried out before taking decision on the per unit cost of the embedded product.

When the product is introduced in the market, for the initial period the sales and revenue will be low. There won't be much competition when the product sales and revenue increase. During the maturing phase, the growth will be steady and revenue reaches highest point and at retirement time there will be a drop in sales volume.

4. Quality attributes to be considered in an embedded system design

Characteristics Of Embedded System

Following are some of the characteristics of an embedded system that make it different from a general purpose computer:

Application and Domain specific

An embedded system is designed for a specific purpose only. It will not do any other task.

Ex. A washing machine can only wash, it cannot cook

Certain embedded systems are specific to a domain: ex. A hearing aid is an application that belongs to the domain of signal processing.

Reactive and Real time

Certain Embedded systems are designed to react to the events that occur in the nearby environment. These events also occur real-time.

Ex. An air conditioner adjusts its mechanical parts as soon as it gets a signal from its sensors to increase or decrease the temperature when the user operates it using a remote control. An embedded system uses Sensors to take inputs and has actuators to bring out the required functionality.

Operation in harsh environment

Certain embedded systems are designed to operate in harsh environments like very high temperature of the deserts or very low temperature of the mountains or extreme rains. These

embedded systems have to be capable of sustaining the environmental conditions it is designed to operate in.

Distributed

Certain embedded systems are part of a larger system and thus form components of a distributed system. These components are independent of each other but have to work together for the larger system to function properly.

Ex. A car has many embedded systems controlled to its dash board. Each one is an independent embedded system yet the entire car can be said to function properly only if all the systems work together.

Small size and weight

An embedded system that is compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.

Ex. Currently available cell phones. The cell phones that have the maximum features are popular but also their size and weight is an important characteristic.

For convenience users prefer mobile phones than phablets. (phone + tablet pc)

Power concerns

- It is desirable that the power utilization and heat dissipation of any embedded system be low.
- If more heat is dissipated then additional units like heat sinks or cooling fans need to be added to the circuit.
- If more power is required then a battery of higher power or more batteries need to be accommodated in the embedded system

UNIT-II

1. Components used as the core of embedded systems

Core of Embedded Systems

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

General purpose and Domain Specific Processors

· Microprocessors

· Microcontrollers

· Digital Signal Processors

Application Specific Integrated Circuits. (ASIC)

Programmable logic devices(PLD's)

Commercial off-the-shelf components (COTs)

MICROPROCESSORS

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

o Intel – *Intel 4004* – November 1971(4-bit). o Intel – *Intel 4040.*

o Intel – *Intel 8008* – April 1972.

o Intel – *Intel 8080* – April 1974(8-bit). o Motorola – *Motorola 6800.*

o Intel – *Intel 8085* – 1976.

MICROCONTROLLERS.

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

Digital Signal Processors

DSP are powerful special purpose 8/16/32 bit microprocessor designed to meet the computational demands and power constraints of today's embedded audio, video and communication applications.

DSP are 2 to 3 times faster than general purpose microprocessors in signal processing applications. This is because of the architectural difference between DSP and general purpose microprocessors.

Application Specific Integrated Circuits. (ASIC)

ASICs is a microchip design to perform a specific and unique applications. Because of using single chip for integrates several functions there by reduces the system development cost. Most of the ASICs are proprietary (which having some trade name) products, it is referred as Application Specific Standard Products(ASSP).

As a single chip ASIC consumes a very small area in the total system. Thereby helps in the design of smaller system with high capabilities or functionalities. The developers of such chips may not be interested in revealing the internal detail of it

Programmable logic devices(PLD's)

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

PLDs having following two major types.

1) CPLD(Complex Programmable Logic Device):

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

2) FPGAs(Field Programmable Gate Arrays):

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

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

Commercial off-the-shelf components(COTs)

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

2. Different types of memories used in embedded systems design

Types of Memory Modules

Different types of memory modules for any system depend on the nature of application of that system. The memory performance and capability requirements are small for low cost systems. Selection of a memory module is the most critical requirement in a designing a microcontroller based project.

The following general types of memory module can be used in an embedded system.

- Volatile Memory
- Non-Volatile Memory

Volatile Memory Module – RAM

Volatile memory devices are types of storage devices which hold their content till power is applied to them.When power is switched off, these memories lose their content.

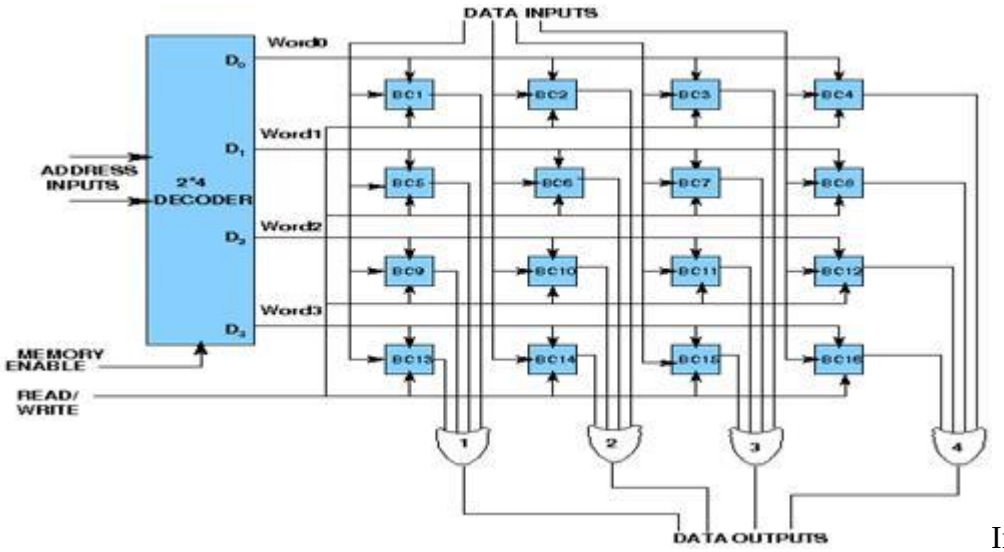
The RAM memory chip, referred to as a main memory, is a storage location that allows information to be stored and accessed quickly from random location with memory module. The memory cell which can be accessed for information transfer to or from any desired random location is called a Random Access Memory.

A RAM memory is designed with a collection of storage cells. Each cell contains either BJT or MOSFET based on type of memory module. For example, 4*4 RAM memory can store 4 bit of information.

Every instruction of a row and column in this matrix is a memory cell. Each block labeled BC, represents the binary cells with its 3 inputs and 1 output. Each block consists of 12 binary cells.

Internal Data Storage Circuit for RAM Memory

To each memory block, each word output from the decoder is the select input. The decoder is enabled with the memory enable input. When the memory enable pin is at logic low level, all outputs of the decoder are at logic low level and memory doesn't select any word. When the enable pin is at logic high level, the parallel output corresponding to the serial input is given as select input to each memory block.



Internal Data storage circuit for RAM memory chip

Once the word has been selected, the read and write pin for each block, determines the operation. If the read/write pin is at logic low level, the input is written onto the memory block. If the read/write pin is at logic high level, the output is read from each block.

Non- Volatile Memory-ROM Memory

Non-volatile memories are permanent storage types of memory chips which can get back stored information even when the power is switched off. An example of non-volatile memory device is Read Only Memory (ROM).

The ROM stands for Read Only Memory. ROM can only be used to read from, but cannot be written upon. These memory devices are non-volatile.

Non volatile memory-ROM Memory

The information is stored permanently in such memories during manufacture. The ROM can store instructions which are required to start computer when power is given to the computer. This operation is referred to as bootstrap.

A ROM memory cell is designed with a single transistor. The ROM memory is not only used in the computers but also in other electronic devices like controllers, micro ovens, washing machines etc.

A ROM family is designed with collection of storage cells. Each memory cell contains either bipolar or MOSFET transistor based on types of memory.

3. Sensors and Actuators

Sensors & Actuators

Sensor

A Sensor is used for taking Input. It is a transducer that converts energy from one form to another for any measurement or control purpose

Ex. A Temperature sensor

Actuator

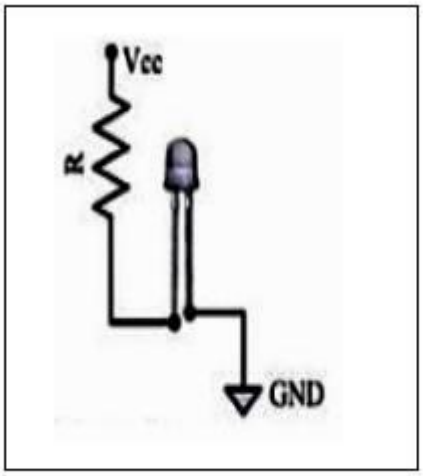
Actuator is used for output. It is a transducer that may be either mechanical or electrical which converts signals to corresponding physical actions.

Ex. LED (Light Emitting Diode)

LED is a p-n junction diode and contains a **CATHODE** and **ANODE**

For functioning the anode is connected to +ve end of power supply and cathode is connected to –ve end of power supply.

The maximum current flowing through the LED is limited by connecting a RESISTOR in series between the power supply and LED as shown in the figure below



There are two ways to interface an LED to a microprocessor/microcontroller:

The Anode of LED is connected to the port pin and cathode to Ground : In this approach the port pin sources the current to the LED when it is at logic high(ie. 1).

The Cathode of LED is connected to the port pin and Anode to Vcc : In this approach the port pin sources the current to the LED when it is at logic high (ie. 1). Here the port pin sinks the current and the LED is turned ON when the port pin is at Logic low (ie. 0)

4. Communication interface.

Communication Interfaces:

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

On board Communication Interfaces

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

Common examples of onboard interfaces are:

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

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

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

It comprises of two buses :

Serial clock –SCL

Serial Data – SDA

SCL generates synchronization clock pulses SDA transmits data serially across devices I2C is a shared bus system to which many devices can be connected. Devices connected by I2C can act as either master or slave. The master device is responsible for controlling communication by initiating/ terminating data transfer. Devices acting as slave wait for commands from the master and respond to those commands.

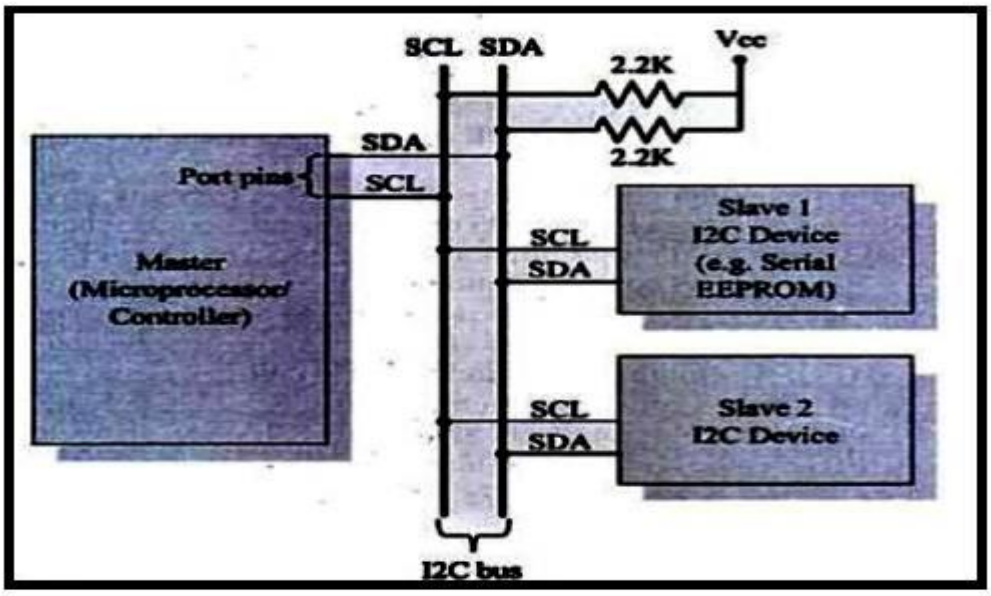


Figure: I2C Bus Interfacing

External or Peripheral Communication Interfaces

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

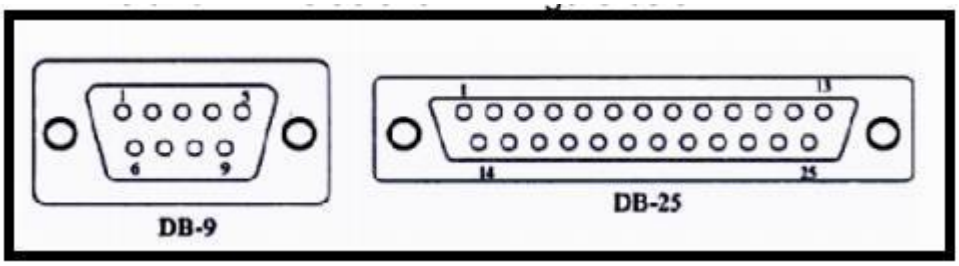
- RS-232 C & RS-485
- Universal Serial Bus (USB)
- IEEE 1394 (Firewire)
- Infrared (IrDA)
- Bluetooth
- Wi-Fi
- Zig Bee
- General Packet Radio Service (GPRS)

Example: RS-232 C & RS-485

It is wired, asynchronous, serial, full duplex communication RS 232 interface was developed by EIA (Electronic Industries Associates) In early 1960s RS 232 is the extension to UART for external communications RS-232 logic levels use:

+3 to +25 volts to signify a "Space" (Logic 0) and
-3 to -25 volts to signify a "Mark" (logic 1).

RS 232 supports two different types of connectors :
DB 9 and DB 25 as shown in figure below



RS 232 interface is a point to point communication interface and the devices involved are called as Data Terminating Equipment (DTE) And Data
Communications Terminating Equipment (DCE)

Embedded devices contain UART for serial transmission and generate signal levels as per TTL/CMOS logic. A level translator IC (like Max 232) is used for converting the signal lines from UART to RS 232 signal lines for communication.

The vice versa is performed on the receiving side. Converter chips contain converters for both transmitters and receivers, RS 232 is used only for point to point connections. It is susceptible to noise and hence is limited to short distances only RS 422 is another serial interface from EIA. It supports multipoint connections with 1 transmitter and 10 receivers. It supports data rates up to 100Kbps and distance up to 400 ft RS 485 is enhanced version of RS 422 and supports up to 32 transmitters and 32 receivers.

UNIT-III

1. Embedded firmware design approaches

Embedded firmware is the flash memory chip that stores specialized software running in a chip in an embedded device to control its functions.

Firmware in embedded systems fills the same purpose as a ROM but can be updated more easily for better adaptability to conditions or interconnecting with additional equipment. Hardware makers use embedded firmware to control the functions of various hardware devices and systems much like a computer's operating system controls the function of software applications. Embedded firmware exists in everything from appliances so simple you might not imagine they had computer control, like toasters, to complex tracking systems in missiles. The toaster would likely never need updating but the tracking system sometimes does. As the complexity of a device increases, it often makes sense to use firmware in case of design errors that an update might correct.

Embedded firmware is used to control the limited, set functions of hardware devices and systems of greater complexity but still gives more appliance-like usage instead of a series of terminal commands. Embedded firmware functions are activated by external controls or external actions of the hardware. Embedded firmware and ROM-based embedded softwareoften have communication links to other devices for functionality or to address the need for the

device to be adjusted, calibrated or diagnosed or to output log files. It is also through these connections that someone might attempt embedded device hacking.

Embedded software varies in complexity as much the devices it is used to control. Although *embedded software* and *embedded firmware* are sometimes used synonymously, they are not exactly the same thing. For example, embedded software may run on ROM chips. Also, embedded software is often the only computer code running on a piece of hardware while firmware can also refer to the chip that houses a computer's EFI or BIOS, which hands over control to an OS that in turn launches and controls programs.

Hardware and firmware engineering design teams often run into problems and conflicts when trying to work together. They come from different development environments, have different tool sets and use different terminology. Often they are in different locations within the same company or work for different companies. The two teams have to work together, but often have conflicting differences in procedures and methods. Since their resulting hardware and firmware work have to integrate successfully to build a product, it is imperative that the hardware/firmware interface – including people, technical disciplines, tools and technology – be designed properly

This article provides seven principles hardware/firmware codesign that if followed will ensure that such collaborations are a success. They are:

1. Collaborate on the Design;
2. Set and Adhere to Standards;
3. Balance the Load;
4. Design for Compatibility;
5. Anticipate the Impacts;
6. Design for Contingencies; and
7. Plan Ahead.

1. Watchdog timers

For those embedded systems that can't be constantly watched by a human, watchdog timers may be the solution.

Most embedded systems need to be self-reliant. It's not usually possible to wait for someone to reboot them if the software hangs. Some embedded designs, such as space probes, are simply not accessible to human operators. If their software ever hangs, such systems are permanently disabled. In other cases, the speed with which a human operator might reset the system would be too slow to meet the uptime requirements of the product.

A watchdog timer is a piece of hardware that can be used to automatically detect software anomalies and reset the processor if any occur. Generally speaking, a watchdog timer is based on a counter that counts down from some initial value to zero. The embedded software selects the counter's initial value and periodically restarts it. If the counter ever reaches zero before the software restarts it, the software is presumed to be malfunctioning and the processor's reset signal is asserted. The processor (and the embedded software it's running) will be restarted as if a human operator had cycled the power.

Figure 1 shows a typical arrangement. As shown, the watchdog timer is a chip external to the processor. However, it could also be included within the same chip as the CPU. This is done in many microcontrollers. In either case, the output from the watchdog timer is tied directly to the processor's reset signal.

Figure 1: A typical watchdog setup

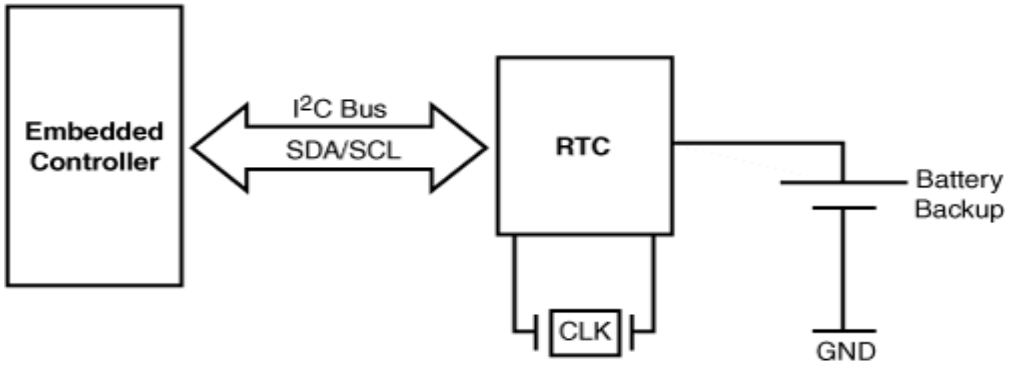
2. Real-time clock

A **real-time clock** (**RTC**) is a computer clock (most often in the form of an integrated circuit) that keeps track of the current time. Although the term often refers to the devices in personal computers, servers and embedded systems, RTCs are present in almost any electronic device which needs to keep accurate time. A common RTC used in single-board computers is the DS1307.

The term *real-time clock* is used to avoid confusion with ordinary hardware clocks which are only signals that govern digital electronics, and do not count time in human units. RTC should not be confused with real-time computing, which shares its three-letter acronym but does not directly relate to time of day.

Although keeping time can be done without an RTC, using one has benefits:

- Low power consumption[2] (important when running from alternate power)
- Frees the main system for time-critical tasks
- Sometimes more accurate than other methods



A GPS receiver can shorten its startup time by comparing the current time, according to its RTC, with the time at which it last had a valid signal.[3] If it has been less than a few hours, then the previous ephemeris is still usable.

The RTC was introduced to PC compatibles by the IBM PC/AT in 1984, which used a Motorola MC146818 RTC. Later, Dallas Semiconductor made compatible RTCs, which were often used in older personal computers, and are easily found on motherboards because of their distinctive black battery cap and silkscreened logo. In newer systems, the RTC is integrated into the south bridge chip.

Some microcontrollers have a real-time clock built in, generally only the ones with many other features and peripherals.

3. **Brownout Protection**

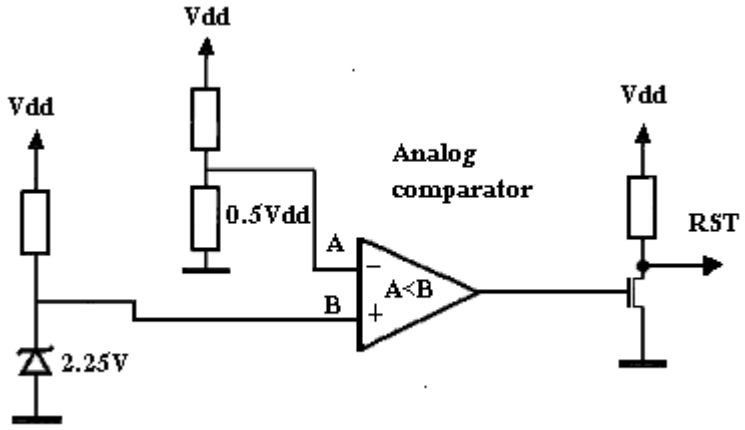
Well almost all microcontrollers have Brownout protection inbuilt in them but when connecting a controller to an industry sensor and controlling devices(which are extremely costly) its better we know what is a brownout and how is it detected in a microcontroller cause many devices in low to medium scale industry may not be as immune to brownout as our controller.

The brown out can cause one of the three things for a dc supply system. These things in turn can damage the connected embedded systems.

- An unregulated direct current supply will produce a lower output voltage for electronic circuits. The output ripple voltage will decrease in line with the usually reduced load current.
- A linear direct current regulated supply will maintain the output voltage unless the brownout is severe and the input voltage drops below the drop out voltage for the regulator, at which point the output voltage will fall and high levels of ripple from the rectifier/reservoir capacitor will appear on the output.
- A switched-mode power supply which has a regulated output will be affected. As the input voltage falls, the current draw will increase to maintain the same output voltage and current, until such a point that the power supply malfunctions.

Brown out detector is nothing more than a comparator. It basically compares supply voltage with a fixed triggered voltage level during operation. So in a microcontroller the BOD comtinuously compares supply voltage so as to trigger a reset of controller if supply voltage fluctuates below a triggered voltage level. When triggered it immediately cuts of the supply to peripheries and memories while freezing the status of controller and doing a reset. All types of memories like SRAM and EEPROM can get corrupted due to effect of a brown-out except flash program memories. Why?? cause most of the controllers cannot write to their own Flash.

A simple Brown out Circuit can be seen below



UNIT-IV

1. User Level threads Vs Kernel Level thread

Kernel-Level Threads

To make concurrency cheaper, the execution aspect of process is separated out into threads. As such, the OS now manages threads and processes. All thread operations are implemented in the kernel and the OS schedules all threads in the system. OS managed threads are called kernel-level threads or light weight processes.

NT: Threads,Solaris: Lightweight processes (LWP).

In this method, the kernel knows about and manages the threads. No runtime system is needed in this case. Instead of thread table in each process, the kernel has a thread table that keeps track of all threads in the system. In addition, the kernel also maintains the traditional process table to keep track of processes. Operating Systems kernel provides system call to create and manage threads.

User-Level Threads

Kernel-Level threads make concurrency much cheaper than process because, much less state to allocate and initialize. However, for fine-grained concurrency, kernel-level threads still suffer from too much overhead. Thread operations still require system calls. Ideally, we require thread operations to be as fast as a procedure call. Kernel-Level threads have to be general to support the needs of all programmers, languages, runtimes, etc. For such fine grained concurrency we need still "cheaper" threads.

To make threads cheap and fast, they need to be implemented at user level. User-Level threads are managed entirely by the run-time system (user-level library).The kernel knows nothing about user-level threads and manages them as if they were single-threaded processes. User-Level threads are small and fast, each thread is represented by a PC, register, stack, and small thread control block. Creating a new thread, switching between threads, and synchronizing threads are done via procedure call

Operating System User Level threads Vs Kernel Level thread

USER LEVEL THREAD	**KERNEL LEVEL THREAD**
User thread is implemented by users.	Kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris.

2. Process Scheduling in Operating Systems

In an operating system (OS), a process scheduler performs the important activity of scheduling a process between the ready queue and waiting queue and allocating them to the CPU. The OS assigns priority to each process and maintains these queues. The scheduler selects the process from the queue and loads it into memory for execution. There are two types of process scheduling: **preemptive** scheduling and **non-preemptive** scheduling.

Preemptive Scheduling

The scheduling in which a running process can be interrupted if a high priority process enters the queue and is allocated to the CPU is called preemptive scheduling. In this case, the current process switches from the running queue to ready queue and the high priority process utilizes the CPU cycle.

Let's take an example where four processes P0, P1, P2, and P3 have different arrival times in the queue. Based on their priority or burst time, these processes are interrupted and allocated the CPU.

Process	Arrival Time	Burst Time (milli-sec)
P0	3	2
P1	2	4
P2	0	6
P3	1	4

P2	P3	P0	P1	P2	
0	1	5	7	11	16

A process P2 arrives at time 0 and allocated the CPU.

Process P3 arrives at time 1, before P2 finishes execution. The time remaining for P2 is 5 milliseconds which is larger than a time required for P3 (4 milliseconds). So, CPU is allocated to process P3. Process P1 arrives at time 2. P3 continues to execute because the remaining time for P3 (3 milliseconds) is less than the time required by processes P1 (4 milliseconds) and P2 (5 milliseconds). Process P0 arrives at time 3. Now P3 continues to run because the remaining time for P3 (2 milliseconds) is equal to the time required by P0 (2 milliseconds). After P3 finishes, the CPU is allocated to P0 as it has smaller burst time than other processes. Later, the CPU is allocated to P1 and then to P2. Non-preemptive Scheduling

The scheduling in which a running process cannot be interrupted by any other process is called non-preemptive scheduling. Any other process which enters the queue has to wait until the current process finishes its CPU cycle.

Let's take the same example as above and apply non-preemptive scheduling to it.

Process	Arrival Time	Burst Time (milli-sec)
P0	3	2
P1	2	4
P2	0	6
P3	1	4

P2	P3	P1	P0	
0	6	10	14	16

Process P2 arrives at time 0 and is allocated the CPU until it finishes execution. While P2 is executing, processes P0, P1, P3 arrive into the ready queue. But, all other processes have to wait until process P2 finishes its execution. After P2 finishes its CPU cycle, based on the arrival time, process P3 is allocated the CPU. After P3 finishes, P1 executes and then P0.

Differences:

The main differences between preemptive and non-preemptive scheduling are:

1. Preemptive scheduling is flexible as it allows any high priority process to access the CPU. On the other hand, a non-preemptive scheduling is rigid as the current process continues to access the CPU even if other processes enter the queue.
2. The running process is interrupted in a preemptive scheduling whereas a running process cannot be interrupted in non-preemptive scheduling.
3. In preemptive scheduling, there is an overhead of switching the process from running queue to waiting queue. There is no overhead of process switching in non-preemptive scheduling.
4. The process with low-priority has to starve for CPU resources in preemptive scheduling. In non-preemptive scheduling, the process with the smallest burst time has to starve if the CPU is allocated to a process with large burst time.

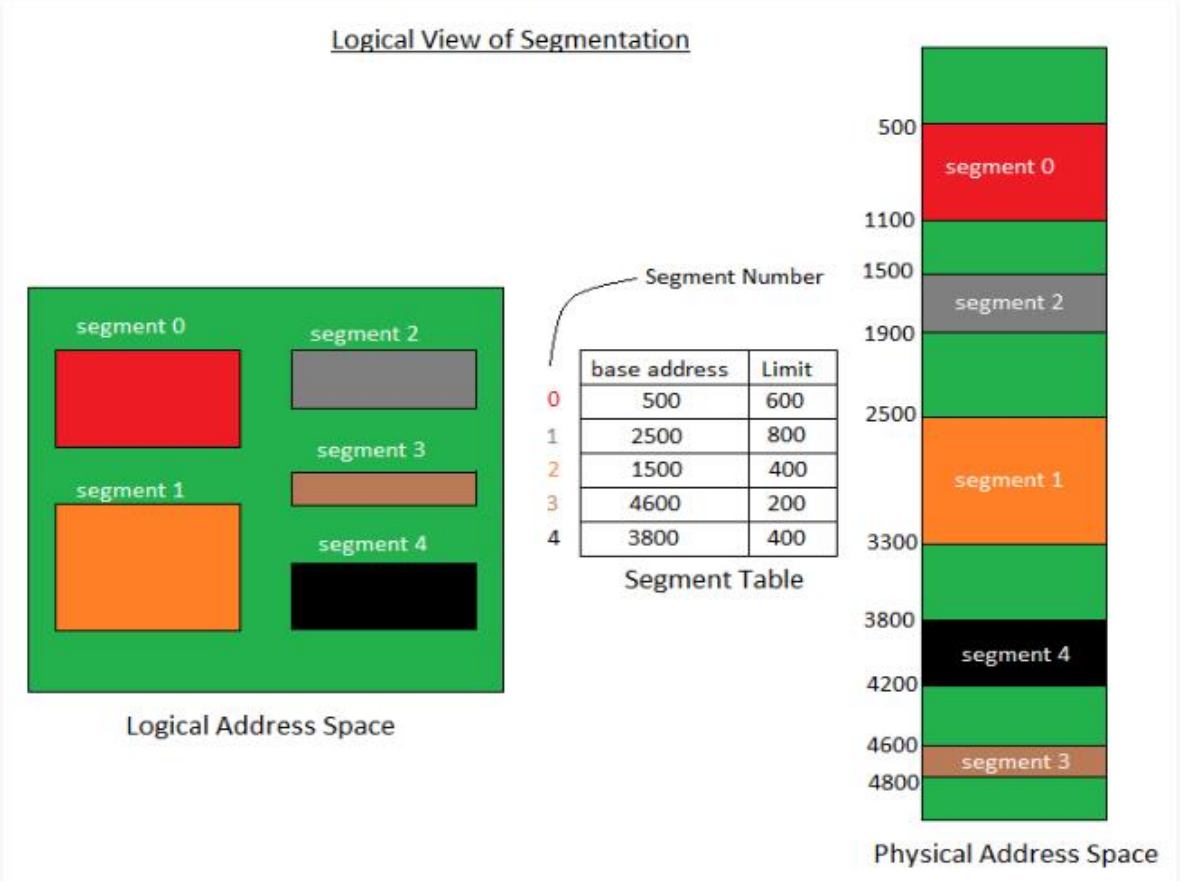
3. Memory management and Segmentation.

Operating System | Segmentation

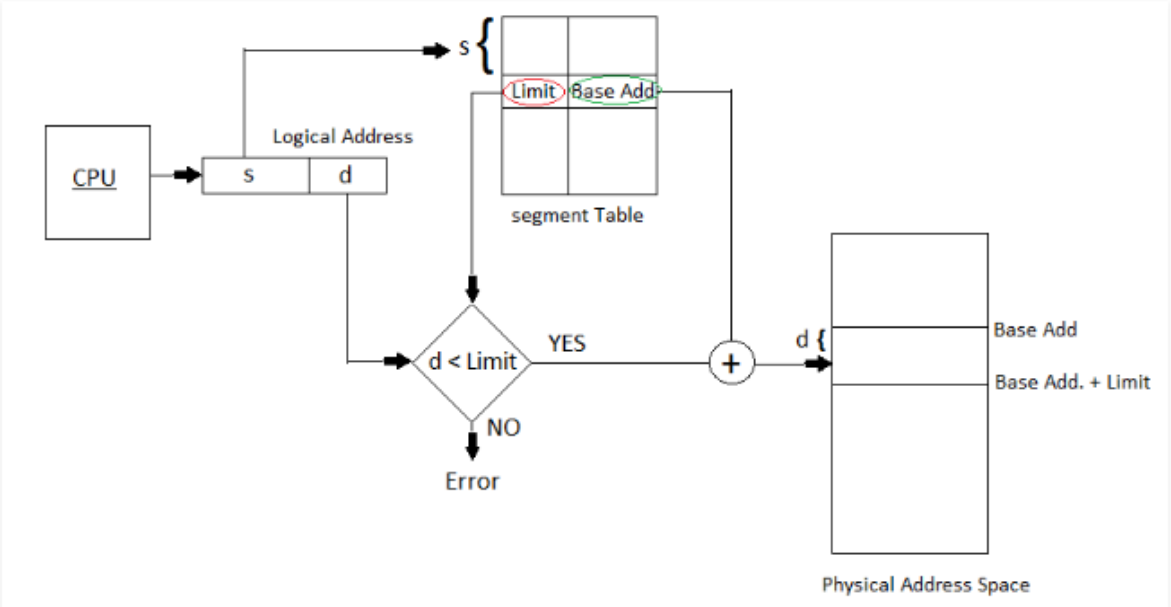
A Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes. Each chunk is called a Segment. A table stores the information about all such segments and is called Segment Table.

Segment Table – It maps two dimensional Logical address into one dimensional Physical address. It's each table entry has:

- Base Address: It contains the starting physical address where the segments reside in memory.
- Limit: It specifies the length of the segment.



Translation of Two dimensional Logical Address to one dimensional Physical Address.



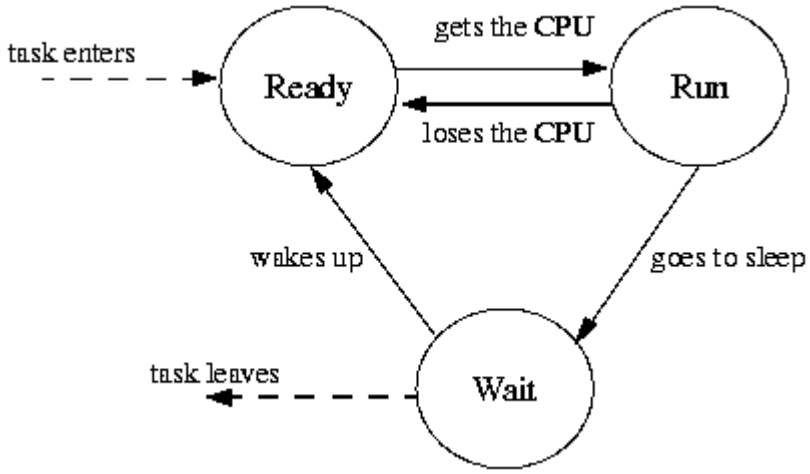
4. Multitasking

Because pre-emptive multitasking was selected for BOAR, this chapter only describes pre-emptive multitasking. However, most of the things apply to co-operative systems also, a function call just replaces the timer interrupt.

The core of a pre-emptive operating system consists of a task switching routine, an interrupt handler, and routines to manage the other system resources. The interrupt handler is the key part in a pre-emptive multitasking system, because it decides when it is time to switch tasks. When a task has used its time-slice, voluntarily given up the processor, or it is pre-empted by another task with a higher priority, the switching routine is called. The scheduler [5] then decides which task to run next, and the switching routine loads that task and starts to run it.

There are three states a task can be in: currently running, ready to be run or waiting for a signal. A task is said to "have the CPU" when it is running. When tasks are switched, the task that was running "loses the CPU," while the task that was ready "gets the CPU" ("acquires the CPU") and begins to run. Other transitions between the states are called "goes to sleep" and "wakes up." In BOAR a newly-created task enters in the ready state and a task that has ended its execution leaves the system through the waiting state. Figure 4 shows these task states and transitions between the states.

Figure 4. Task states, transitions and terms



There are different techniques for the selection of the task to run next. Naturally, complex real-time scheduling algorithms take much more processing time than simple priority-based algorithms.

To make things simpler and faster, BOAR uses a simple priority queue with round-robin algorithm between tasks at the same priority. By default time slices of 100 ms are used, but this may be tuned in 10 ms steps, if needed. If the time slices are too short, some performance is lost because there are more context switches than would be actually needed. If too long time slices are used, response times will suffer. The selected default, 100 ms is a compromise between efficiency and response times.

Round-robin means that all tasks with the same priority get the CPU in turn. Each task that is ready (i.e. not waiting and not currently running) is kept in a task ready list and has a

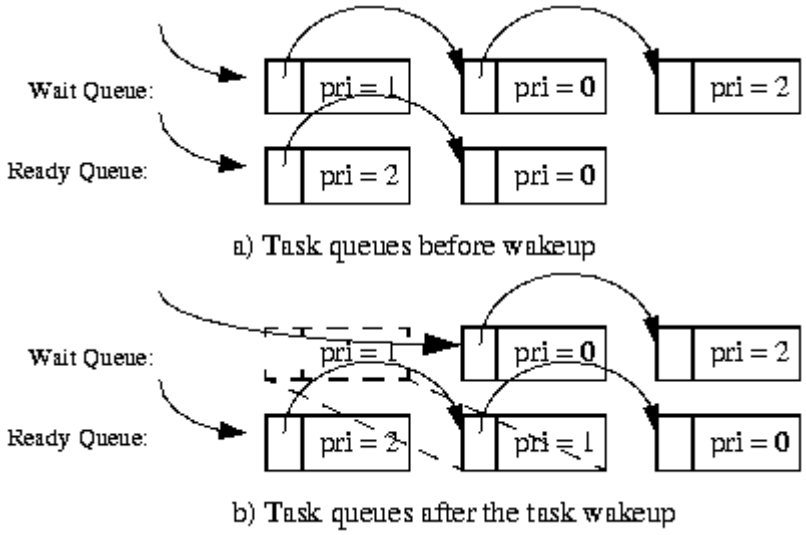
priority attached to it. The elements in the list are sorted according to their priorities (hence the term "priority queue"). Higher priority tasks are put into the head of the list, lower priority tasks are at the tail.

When a new task is being added, it is enqueued to the list, and will be located before all lower priority tasks, but behind all tasks with a higher or the same priority. When the scheduler in the operating system needs a new task to run, it simply takes the first task from the head of the list. This implements priorities and also the round-robin scheduling between all tasks with the same priority.

Because the task ready list is prioritized, no task with a low priority will get the CPU if there is a ready task with a higher priority. This may cause problems in an overloaded system. Lower priority tasks may never get the CPU if one high priority task is at all times running. This is called "starving" and it can be reduced by allowing dynamic priorities. Tasks that use up their time slices get priority demotions and tasks with lower priorities may have a chance to run. Unfortunately, dynamic priorities make the response times unpredictable, because a high priority task does not necessarily get the CPU when it needs to.

When a task goes to sleep it is put into a task wait list. This list does not need to be sorted according to anything. Tasks may be added into the head or to the tail of the list. It is only important what happens when a task wakes up. When a task wakes up, it is removed from the task wait list and enquired to the task ready list. This ensures that the task will be executed as soon as possible.

Figure 5. Priorities, tasks lists



The example in Figure 5 shows what happens to the task lists when a task wakes up. The initial state of the task lists are shown in a). Then a task with priority 1 wakes up. In b) the task has been enquired into proper place in the task ready list.

If the scheduler does not find a ready task to be run (the task ready list is empty), the processor will be stopped until the next interrupt happens. There is a special instruction in the processor for this. When the processor is stopped, power consumption is reduced and the system memory bus is freed. The bus can then be used by another bus master, in this case by the

ethernet controller. Because of this, stopping the processor when there is no real work to be done not only decreases power consumption, but also increases system performance.
An interrupt may and will eventually cause a task wakeup. After the interrupt is serviced, the scheduler tries again to locate a ready task. If there is a ready task, it gets the CPU. If the ready list is still empty, the processor is stopped again.

Because the timer interrupt needs to be serviced whether the running task is switched or not, and there may be more state information to be saved and restored in each task switch, a pre-emptive system has a slightly bigger overhead than a co-operative system. On the other hand, the timer interrupt server can also update a system clock and service timer requests. This removes the interrupt service overhead for the task switching part, because the interrupt service routine would have to be called anyway.

UNIT-V

1. Semaphores

Semaphores are one of the oldest mechanisms introduced by multitasking systems, being used both for managing common resources and synchronisation.

Managing common resources, in its simple form, prevents several threads to concurrently use a shared resource by blocking access of all other threads until the thread that acquired the resource releases it.

Synchronisation is generally required to efficiently implement blocking I/O; when a thread requires some data that is not yet available (for example by performing a read()), it is not efficient to poll until the data becomes available, but it is much better to suspend the thread and arrange for the data producer (usually an ISR) to resume the thread when the data is available.

A semaphore is a synchronisation mechanism offered by most multitasking systems. In its simplest form, a semaphore is similar to the real-world traffic light, which blocks access to a segment of road in certain conditions.

The semaphore concept was introduced in 1965 by the Dutch computer scientist Edsger Dijkstraand historically it is said to be inspired by the railway semaphore (the binary semaphore, which controls access to a single resource by bracing the critical section with the P(S) and V(S) primitives).

The concept was later extended by another Dutchman, Carel S. Scholten, to control access to an arbitrary number of resources. In his proposal the semaphore can have an initial value (or count) greater than one (thus the counting semaphore).

Semaphores were originally used to control access to shared resources. However, depending on the application, better mechanisms exist now to manage shared resources, like locks, mutexes, etc. Semaphores are best used to synchronize a thread with an ISR or with another thread (unilateral rendezvous).

Semaphore types

There are two types of semaphores: binary and counting.

Binary Semaphore

Since we mentioned the analogy with the railway system, let's imagine we have a small train station, with a single platform. The first arriving train enters the station without any restrictions, and stops at the platform. To prevent a second train from entering the station and bumping onto the first, a semaphore is installed at a certain distance before the station. The railway semaphore has a red hand which can be either lowered or raised. Modern semaphores are electric, and also have lights (red and green). After the first train enters the station, the hand

is lowered and the light turns red. If a second train arrives, it reads this as "stop" and waits. When the first train leaves the station, the semaphore arm swings up, the light turns green, and the second train can enter the station.

Like a railway semaphore which has two states, a binary semaphore has only two values, 0 or 1. If the value is 0, the resource associated with the semaphore is not available, and whoever needs it, must wait, like the train that stops at a red semaphore. When the resource becomes available, the semaphore is "posted", allowing the next thread waiting for the semaphore to resume.

Depending on the semaphore usage, it can start either with 0 (when used for synchronisation) or 1 (when used to protect a single shared resource).

What a binary semaphore has in addition to a railway semaphore, is a signalling method (think of this mechanism as a loud horn used to wakeup the sleeping train driver, waiting for the semaphore).

Counting semaphores

To continue the analogy with the railway system, what if we have a larger train station, with multiple platforms, where many trains can be present at the same time? Well, the solution is similar, but the semaphore logic should keep track of the number of trains in the station, and turn red when all tracks are busy. When one train leaves the station, the semaphore can be turned green, and, if there is a train waiting, it'll be allowed to enter the station.

A counting semaphore has a counter with a limit, representing the maximum number of available resources.

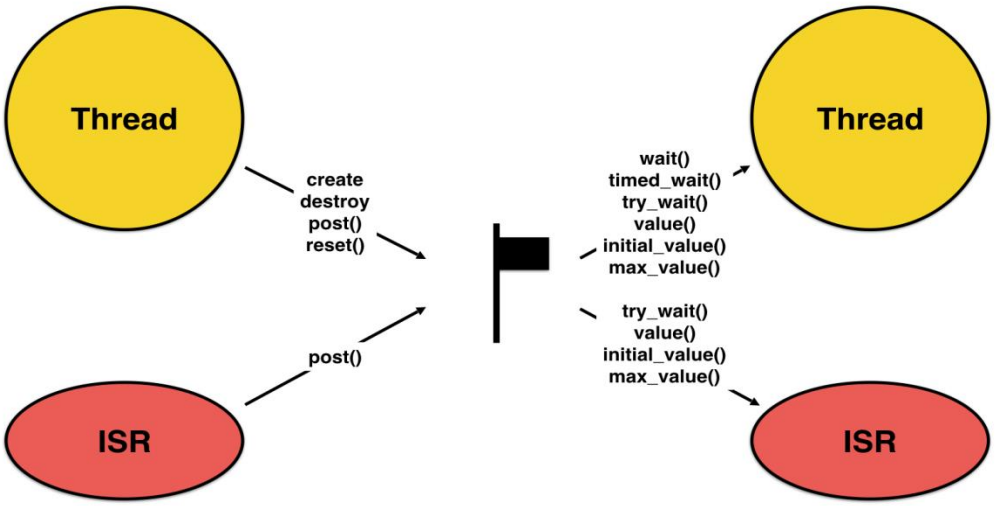
Depending on the semaphore usage, it usually starts either at 0 (when used for synchronisation) or at the limit (when used to protect a multiple shared resources).

Assuming it starts at zero, with no resources available, the semaphore is "posted" each time a new resource becomes available, which increments the counter. When the maximum is reached, further "posts" will fail and the counter will remain unchanged.

On the other side, when resources need to be consumed, as long as the counter is positive, the requesting thread will be allowed access to the resource, and, at each request, the counter will be decremented.

When the counter reaches 0, no more resources are available, and the requesting thread is suspended, until the semaphore will be posted.

A counting semaphore is used when elements of a resource can be used by more than one thread at the same time. For example, a counting semaphore can be used in the management of a buffer pool.



Semaphore services

2. Synchronization Issues.

We have our basic mechanism for starting concurrent executions (separate creation) and for requesting operations from these executions (the usual feature call mechanism). Any concurrent computation, object-oriented or not, must also provide ways to synchronize concurrent executions, that is to say to define timing dependencies between them.

If you are familiar with concurrency issues, you may have been surprised by the announcement that a single language mechanism, *separate* declarations, is enough to add full concurrency support to our sequential object-oriented framework. Surely we need specific synchronization mechanisms too? Actually no. The basic O-O constructs suffice to cover a wide range of synchronization needs, provided we adapt the definition of their semantics when they are applied to separate elements. It is a testament to the power of the object-oriented method that it adapts so simply and gracefully to concurrent computation.

Synchronization vs. communication

To understand how we should support synchronization in object-oriented concurrency, it is useful to begin with a review of non-O-O solutions. Processes (the concurrent units in most of these solutions) need mechanisms of two kinds:

- *Synchronization* mechanisms enforce timing constraints. A typical constraint might state that a certain operation of a process, such as accessing a database item, may only occur after a certain operation of another process, such as initializing the item.
- *Communication* mechanisms allow processes to exchange information, which in the object-oriented case will be in the form of objects or object structures.

A simple classification of approaches to concurrency rests on the observation that some of them focus on the synchronization mechanism and then use ordinary non-concurrent techniques such as argument passing for communication, whereas others treat communication as the fundamental issue and deduce synchronization from it. We may talk about *synchronization-based* and *communication-based* mechanisms.

Synchronization-based mechanisms

The best known and most elementary synchronization-based mechanism is the semaphore, which can be viewed as a locking tool for controlling shared resources. A semaphore is an object on which two operations are available: *reserve* and *free* (traditionally called *P* and *V*, but more suggestive names are preferable). At any time the semaphore is either reserved by a certain client or free. If it is free and a client executes *reserve*, the semaphore becomes reserved by that client. If the client that has reserved the semaphore executes *free*, the semaphore becomes free. If the semaphore is reserved by a client and another executes *reserve*, the new client will wait until the semaphore is free again. The following table summarizes this specification:Events represented by shaded entries are not supposed to occur; they can be treated either as errors or as having no effect.

The policy for deciding which client gets through when two or more are waiting for a semaphore that gets freed may be part of the semaphore's specification, or may be left unspecified. (Usually clients expect a *fairness* property guaranteeing that if everyone gaining access to the semaphore ultimately frees it no one will wait forever.)

This description covers *binary* semaphores. The *integer* variant lets at most *n* clients through at any given time, for some *n*, rather than at most one.

Semaphores are widely considered to be too low-level for building large, reliable concurrent systems. But they can be used for small examples, and provide a starting point for discussing more advanced techniques.

A more abstract approach is based on critical regions. A critical region is a sequence of instructions that may be executed by at most one client at a time. To ensure exclusive access to a certain object *a* you may write something like *hold* a then ... Operations involving fields of *a* ...*end* where the critical region is delimited by *then* ... *end*. Only one client can execute the critical region at any given time; others executing a *hold* will wait. Most applications need a more general variant, the conditional critical region, in which execution of the critical region is subject to a certain Boolean condition. Consider for example a buffer shared by a producer, which can only write into the buffer if it is not full, and a consumer, which can only read from it if it is not full; they may use the two respective schemes *hold* buffer when not buffer. full then "Write into buffer, making it not empty" end *hold* buffer when not buffer. empty then "Read from buffer, making it not full" end Such interplay between input and output conditions cries for introducing assertions and giving them a role in synchronization, an idea to be exploited later in this chapter.

Another well-known synchronization-based mechanism, combining the notion of critical region with the modular structure of some modern programming languages, is the monitor. A monitor is a program module, not unlike the packages of Modula or Ada. The basic

synchronization mechanism is simple: mutual exclusion at the routine level. At most one client may execute a routine of the monitor at any given time.

Also interesting is the notion of path expression. A path expression specifies the possible sequencing of a set of processes. For example the expression *init* ; (*reader** | *writer*)+ ; *finish* prescribes the following behaviour: first an *init* process; then a state in which at any time either one *writer* process or any number of *reader* processes may be active; then a *finish* process. The asterisk * means any number of concurrent instances; the semicolon ; indicates sequencing; | means "either-or"; + means any number of successive repetitions. An argument often cited in favour of path expressions is that they specify synchronization constraints outside of the processes, avoiding interference with the description of their algorithmic tasks.

Communication-based mechanisms

Starting with Hoare's "Communicating Sequential Processes" (CSP) in the late seventies, most non-O-O concurrency work has focused on communication-based approaches. The rationale is easy to understand. If you have solved the synchronization problem, you must still find a way to make concurrent units communicate. But if you devise a good communication mechanism you might very well have solved synchronization too: because two units cannot communicate unless the sender is ready to send and the receiver ready to receive, communication implies synchronization; pure synchronization may be viewed as the extreme case of communicating an empty message. If your communication mechanism is general enough, it will provide *all* the synchronization you need.

CSP is based on this "I communicate, therefore I synchronize" view. The starting point is a generalization of a fundamental concept of computing, input and output: a process receives information *v* from a certain "channel" *c* through the construct *c* ? *v*; it sends information to a channel through the construct *c* ! *v*. Channel input and output are only two among the possible examples of *events*.

For more flexibility CSP introduces the notion of non-deterministic wait, represented by the symbol ▮, enabling a process to wait on several possible events and execute the action associated with the first that occurs. Assume for example a system enabling a bank's customers to make inquiries and transfers on their accounts, and the bank manager to check what is going on:

```
(customer_input ? balance_enquiry ->
  (customer_input ? password ->
    (password_valid -> customer_output ! balance)
      ▮
(password_invalid -> (customer_output ! rejection)))
  ▮
customer_input ? transfer_request -> ...
  ▮
manager ? control_operation ->  ...)
```

In the initial state the system stands ready to accept one of three possible input events, two on the *customer input* channel and one on the *manager* channel. The first event that occurs

will trigger the behaviour described on the right of the corresponding arrow; that behaviour is specified in the same way. Once the event's processing is complete, the system returns to its initial state, listening to possible input events.

CSP was a major influence on the concurrency mechanism of Ada, whose "tasks" are processes able to wait on several possible "entries" through an "accept" instruction. The Occam language, a direct implementation of CSP, is the primary programming tool for the *transporter*, a family of microprocessors designed specifically by Inmost (now SGS-Thomson) for the construction of highly concurrent architectures.

3. Priority Inversion

Most commercial real-time operating systems (RTOSes) employ a priority-based preemptive scheduler. These systems assign each task a unique priority level. The scheduler ensures that of those tasks that are ready to run, the one with the highest priority is always the task that is actually running. To meet this goal, the scheduler may preempt a lower-priority task in mid-execution.

Because tasks share resources, events outside the scheduler's control can prevent the highest priority ready task from running when it should. If this happens, a critical deadline could be missed, causing the system to fail. Priority inversion is the term for a scenario in which the highest-priority ready task fails to run when it should.

Resource sharing

Tasks need to share resources to communicate and process data. This aspect of multi-threaded programming is not specific to real-time or embedded systems.

Any time two tasks share a resource, such as a memory buffer, in a system that employs a priority-based scheduler, one of them will usually have a higher priority. The higher-priority task expects to be run as soon as it is ready. However, if the lower-priority task is using their shared resource when the higher-priority task becomes ready to run, the higher-priority task must wait for the lower-priority task to finish with it. We say that the higher-priority task is "pending" on the resource. If the higher-priority task has a critical deadline that it must meet, the worst-case "lockout time" for all of its shared resources must be calculated and taken into account in the design. If the cumulative lockout times are too long, the resource-sharing scheme must be redesigned.

Since worst-case delays resulting from the sharing of resources can be calculated at design time, the only way they can affect the performance of the system is if no one properly accounts for them.

Priority inversion defined

The real trouble arises at run-time, when a medium-priority task preempts a lower-priority task using a shared resource on which the higher-priority task is pending. If the higher-priority task is otherwise ready to run, but a medium-priority task is currently running instead, a priority inversion is said to occur.

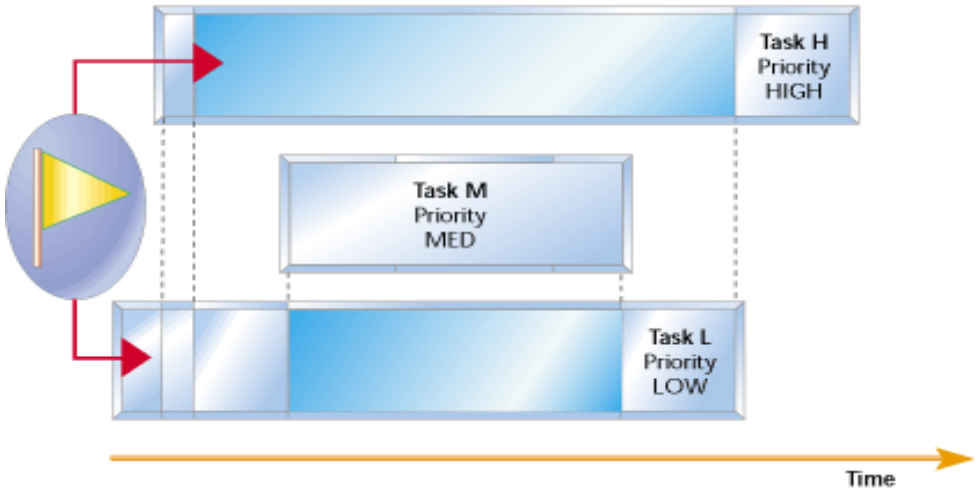


Figure. Priority inversion timeline

This dangerous sequence of events is illustrated in Figure 1. Low-priority Task L and high-priority Task H share a resource. Shortly after Task L takes the resource, Task H becomes ready to run. However, Task H must wait for Task L to finish with the resource, so it ends. Before Task L finishes with the resource, Task M becomes ready to run, preempting Task L. While Task M (and perhaps additional intermediate-priority tasks) runs, Task H, the highest-priority task in the system, remains in a pending state.

Many priority inversions are innocuous or, at most, briefly delay a task that should run right away. But from time to time a system-critical priority inversion takes place. Such an event occurred on the Mars Pathfinder mission in July 1997. The Pathfinder mission is best known for the little rover that took high-resolution color pictures of the Martian surface and relayed them back to Earth.

The problem was not in the landing software, but in the mission software run on the Martian surface. In the spacecraft, various devices communicated over a MIL-STD-1553 data bus. Activity on this bus was managed by a pair of high-priority tasks. One of the bus manager tasks communicated through a pipe with a low-priority meteorological science task.

On Earth, the software mostly ran without incident. On Mars, however, a problem developed that was serious enough to trigger a series of software resets during the mission. The sequence of events leading to each reset began when the low-priority science task was preempted by a couple of medium-priority tasks while it held a mutex related to the pipe. While the low-priority task was preempted, the high-priority bus distribution manager tried to send more data to it over the same pipe. Because the mutex was still held by the science task, the bus distribution manager was made to wait. Shortly thereafter, the other bus scheduler became active. It noticed that the distribution manager hadn't completed its work for that bus cycle and forced a system reset.

This problem was not caused by a mistake in the operating system, such as an incorrectly implemented mutex, or in the application. Instead, the software exhibited behaviour that is a known "feature" of mutexes and intertask communication. In fact, the RTOS used on Pathfinder provided an optional priority-inversion workaround; the scientists at JPL simply

hadn't been aware of that option. Fortunately, they were able to recreate the problem on Earth, remotely enable the workaround, and complete the mission successfully.

4. Inter Process Communication

A process can be of two type:

- Independent process.
- Co-operating process.

An independent process is not affected by the execution of other processes while a co-operating process can be affected by other executing processes. Though one can think that those processes, which are running independently, will execute very efficiently but in practical, there are many situations when co-operative nature can be utilised for increasing computational speed, convenience and modularity. Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

The Figure 1 below shows a basic structure of communication between processes via shared memory method and via message passing.

An operating system can implement both method of communication. First, we will discuss the shared memory method of communication and then message passing. Communication between processes using shared memory requires processes to share some variable and it completely depends on how programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process.

i) Shared Memory Method

ProducerConsumerProblem

There are two processes: Producer and Consumer. Producer produces some item and Consumer consumes that item. The two processes shares a common space or memory location known as buffer where the item produced by Producer is stored and from where the

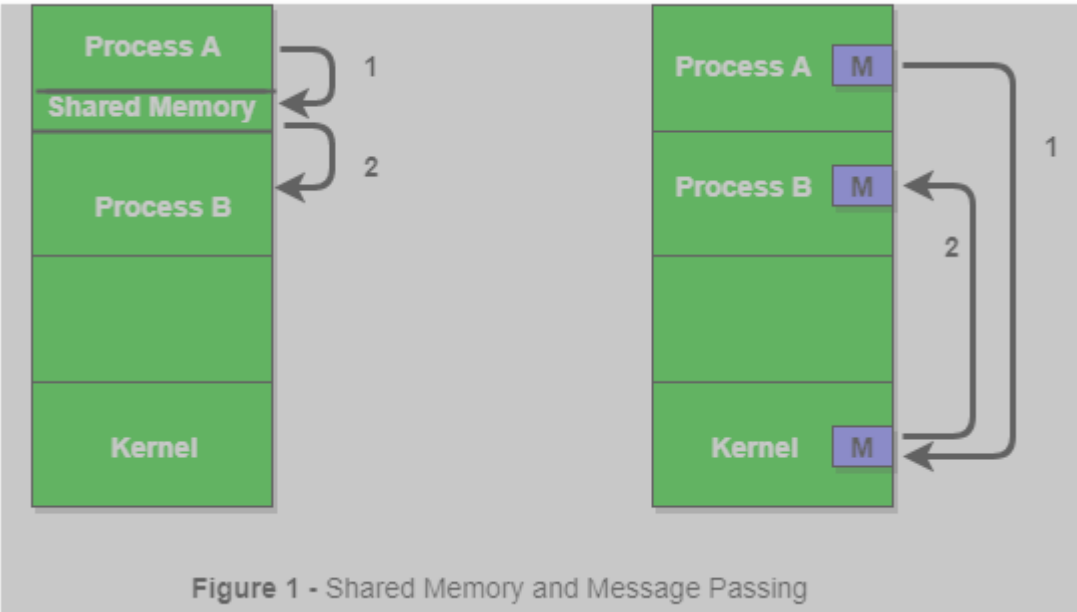


Figure 1 - Shared Memory and Message Passing

Consumer consumes the item if needed. There are two version of this problem: first one is known as unbounded buffer problem in which Producer can keep on producing items and there is no limit on size of buffer, the second one is known as bounded buffer problem in which producer can produce up to a certain amount of item and after that it starts waiting for consumer to consume it. We will discuss the bounded buffer problem. First, the Producer and the Consumer will share some common memory, then producer will start producing items. If the total produced item is equal to the size of buffer, producer will wait to get it consumed by the Consumer.

Similarly, the consumer first checks for the availability of the item and if no item is available, Consumer will wait for producer to produce it. If there are items available, consumer will consume it. The pseudo code are given below:

15. Unit wise-Question bank

UNIT-I

Two marks questions with answers

1. Define embedded system

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.

2. Why we use embedded systems?

Embedded systems avoid lots of electronic components and they have rich built in

functionality. They reduces the cost and maintenance cost and the probability of failure of embedded system is less so embedded system are in very much use now a days.

3. What are advantages of embedded system

Advantages of embedded operating system

Small size and faster to load
More specific to one task
Easy to manage
Low cost
Spend less resources

These operating system is dedicated to one device so performance is good and use less resources like memory and micro-processors

4. What are the examples of embedded operating system?

Windows Xp was using in the ATM machines is one example. Switches, routers systems are other example in which some embedded system are used which keep them working. There are many other examples like operating system used in aircrafts, cars etc.

5. What are the various purposes of embedded systems?

- ✓ Data collection
- ✓ Data communication
- ✓ Data processing

Three marks questions with answers

1. Why we use embedded systems?

Embedded systems avoid lots of electronic components and they have rich built in functionality. They reduces the cost and maintenance cost and the probability of failure of embedded system is less so embedded system are in very much use now a days.

2.What are advantages of embedded system

Advantages of embedded operating system

Small size and faster to load
More specific to one task
Easy to manage
Low cost
Spend less resources

These operating system is dedicated to one device so performance is good and use less resources like memory and micro-processors

3. What are the examples of embedded operating system?

Windows Xp was using in the ATM machines is one example. Switches, routers systems are other example in which some embedded system are used which keep them working. There are many other examples like operating system used in aircrafts, cars etc.

4. Distinguish between General purpose computing systems and Embedded systems

General purpose computing systems	Embedded systems
Contains a general purpose OS	May or may not contain OS for functioning
Applications are alterable by the user.	Applications are non alterable by the user.
Response Time is not critical	Response time may or may Not be Critical
Power Consumption is more	Power Consumption is more
It is combination of generic hardware and a general purpose OS for executing a variety of applications.	It is combination of special purpose hardware and embedded OS for executing specific set of applications
Performance is key deciding factor	Application specific requirements are key deciding factors
Needs not to be deterministic in execution behavior	Executions behavior is deterministic

5. Classify embedded system based On complexity & performance

a).Small scale:

Simple in application need Performance not time critical. Built around low performance & low cost 8 or 16 bit µp/µc.

Example: An electronic toy

b). Medium scale:

Slightly complex in hardware & firmware requirement. Built around medium performance & low cost 16 or 32 bit µp/µc. Usually contain operating system.

Example: Industrial machines.

c).Large scale:

Highly complex hardware & firmware. Built around 32 or 64 bit RISC µp/µc or PLDs or Multi core Processors. Response is time critical.

Examples: Mission critical applications.

Five marks questions with answers

1. Explain any two operational quality attributes in embedded system design.

Throughput:

•It deals with the efficiency of the system.

•In general it can be defined as the rate of production or operation of a defined process over a

given period of time.
•The rate can be expressed in terms of unit of the products, batches produced or any other meaningful measurements.
•Ex: For a card reader, how many transactions does a Reader can perform in a minute or in an hour or in a day?

Reliability:

•It is a measure of how many percent you can rely upon the proper functioning of the system or what is the percentage of the system failure.
•MTBF (Mean Time between Failures), MTTR (Mean Time to Repair) are the terms used in defining system reliability.
•MTBF gives the frequency of failure in hour/week/months.
•MTTR gives how long the system is allowed to be out of order following a failure.

2.Explain any two non-operational quality attributes in embedded system design.

Testability and Debug ability:

•Testing deals with how easily one can test his/her design applications and by which means it can be tested.
•For an embedded product, testability is applicable both the embedded hardware and software.
•Embedded hardware testing ensures that peripheral and total hardware functions in the desired manner whereas software testing ensures that the software is functioning in the expected way.
•Debug ability is a means of debugging the product as such for figuring out the probable sources that create unexpected behavior in the total system.
•Debugging ability has two aspects of embedded system development, namely hardware level debugging & software level debugging.

Portability:

•Portability is a measure of system independence.
•An embedded product is said to be portable if product is capable of functioning (as such in various environment, embedded operating system and target processor)
•A standard embedded product should always be flexible and portable.
•In ES, "porting" means the migration of the embedded firmware written for one target processor (ex: Intel x86) to a different target processor (Hitachi SH3 processor)

3. Define Embedded system and history of embedded system?

Definition:

An embedded system is a combination of 3things:

a. Hardware
b. Software
c. MechanicalComponents

And it is supposed to do one specific task only.

History of embedded system:

- The first recognized embedded system is the Apollo Guidance Computer (AGC) developed by MIT lab.
- AGC was designed on 4K words of ROM & 256 words of RAM.
- The clock frequency of first microchip used in AGCwas 1.024 MHz
- The computing unit of AGC consists of 11 instructions and 16 bit word logic.
- It used 5000ICs.
- The UI of AGC is known DSKY(display/keyboard) which resembles a calculator type keypad with array of numerals.
- The first mass-produced embedded system was guidance computer for the Minuteman-I missile in1961.

4. List some applications of embedded systems

The application areas and the products in the embedded domain are countless.

1. Consumer Electronics: Camcorders, Cameras.
2. Household appliances: Washing machine, Refrigerator.
3. Automotive industry: Anti-lock breaking system(ABS), engine control.
4. Home automation & security systems: Air conditioners, sprinklers, fire alarms.
5. Telecom: Cellular phones, telephone switches.
6. Computer peripherals: Printers, scanners.
7. Computer networking systems: Network routers and switches.
8. Healthcare: EEG, ECG machines.
9. Banking & Retail: Automatic teller machines, point of sales.
10. Card Readers: Barcode, smart card readers.

5. Explain the characteristics of an embedded system?

Following are some of the characteristics of an embedded system that make it different from a general purpose computer:

Application and Domain specific

- An embedded system is designed for a specific purpose only. It will not do any other task.
- Ex. A washing machine can only wash, it cannot cook

- Certain embedded systems are specific to a domain: ex. A hearing aid is an application that belongs to the domain of signal processing.

Reactive and Realtime

- Certain Embedded systems are designed to react to the events that occur in the nearby environment. These events also occur real-time.
- Ex. An air conditioner adjusts its mechanical parts as soon as it gets a signal from its sensors to increase or decrease the temperature when the user operates it using a remote control.
- An embedded system uses Sensors to take inputs and has actuators to bring out the required functionality.

Operation in harsh environment

- Certain embedded systems are designed to operate in harsh environments like very high temperature of the deserts or very low temperature of the mountains or extreme rains.
- These embedded systems have to be capable of sustaining the environmental conditions it is designed to operate in.

Distributed

- Certain embedded systems are part of a larger system and thus form components of a distributed system.
- These components are independent of each other but have to work together for the larger system to function properly.
- Ex. A car has many embedded systems controlled to its dash board. Each one is an independent embedded system yet the entire car can be said to function properly only if all the systems work together.

Small size and weight

- An embedded system that is compact in size and has light weight will be desirable or more popular than one that is bulky and heavy.
- Ex. Currently available cell phones. The cell phones that have the maximum features are popular but also their size and weight is an important characteristic.
- For convenience users prefer mobile phones than phablets. (phone + tablet pc)

Power concerns

- It is desirable that the power utilization and heat dissipation of any embedded system below.

- If more heat is dissipated then additional units like heat sinks or cooling fans need to be added to the circuit.

If more power is required then a battery of higher power or more batteries need to be accommodated in the embedded system.

Objective question with answers

1. What are the essential tight constraint/s related to the design metrics of an embedded system?
a. Ability to fit on a single chip
b. Low power consumption
c. Fast data processing for real-time operations
d. All of the above.

2. Which protocol standard of serial communication specify the bi-directional and half-duplex form of data transmission by allowing various numbers of drivers and receivers in bus configuration?
a. RS232
b. RS2485
c. RS422
d. RS423

3. Which one of the following offers CPUs as integrated memory or peripheral interfaces?
a) Microcontroller
b) Microprocessor
c) Embedded system
d) Memory system

4. How is the protection and security for an embedded system made?
a) OTP
b) IPR
c) Memory disk security
d) Security chips

5. It retains its content when power is removed. What type of memory is this?
a) Volatile memory
b) Nonvolatile memory
c) RAM
d) SRAM

6. The initial routine is often referred to as
a) Initial program
b) Bootstrap program
c) Final program
d) Initial embedded program

7. How an embedded system communicate with the outside world?
a) Peripherals
b) Memory
c) Input
d) Output

8. What kind of visual panel is used for seven segmented display?
a) LED
b) LCD
c) Binary output
d) Analogue output

9. Which of the following allows the reuse of the software and the hardware components?
a) Platform based design
b) Memory design
c) Peripheral design
d) Input design

10. SJMP stands for

a) Small jump b) Serial jump

c) Short jump d) None

Q.No	1	2	3	4	5	6	7	8	9	10
Key	D	B	A	B	B	B	A	B	A	C

Fill in the blanks question with answers

1. Embedded systems are ----------
2. The first recognized modern embedded system is ----------
3. A digital multimeter is an example of an embedded system for --- ---------
4. ASIC stands for ----------
5. Response is a measure of ----------
6. A stepper motor is an ---------- device
7. Which was the first mass produced embedded system ----------
8. Second generations embedded system example ----------
9. ECG stands for ----------
10. MTBF stands for ----------

1. Special purpose
2. AGC
3. Monitoring
4. application specific integrated circuit
5. quickness of system

6. Electro mechanical
7. Minuteman
8. Scada
9. Electrocardiograms
10. Mean time between failures

UNIT-II

Two marks questions with answers

1. What are the languages used in embedded system?

Assembly language and C are basically used for embedded system. Java and ADA are also preferred.

2. How does combination of functions reduce memory requirement in embedded system?

By using functions the amount of code that has to be dealt with is reduced thus redundancy is eliminated for everything common in function.

3. What is digital signal controller ?

DSC is 16 bit RISC machine that combines control advantages of **micro-controller** and digital signal processing to produce tightly coupled single chip-single instruction stream solution for embedded system design.

4. What are the components of embedded system?

Microcontroller, microprocessor, DSC, DSP, busses, system clock, Read only Memory(ROM), RAM, Real time clock these are the components of embedded system.

5. Why we use embedded systems?

Embedded systems avoid lots of electronic components and they have rich built in functionality. They reduces the cost and maintenance cost and the probability of failure of embedded system is less so embedded system are in very much use now a days.

Three marks questions with answers

1. What is an ASIC?

An Application-Specific Integrated Circuit (ASIC) /ˈeɪsɪk/, is an integrated circuit (IC) customized for a particular use, rather than intended for general-purpose use. For example, a chip designed to run in a digital voice recorder or a high-efficiency Bitcoin miner is an ASIC. Application-specific standard products (ASSPs) are intermediate between ASICs and industry standard integrated circuits like the 7400 series or the 4000 series.

2. What does *Commercial Off-The-Shelf (COTS)*mean?

Commercial off-the-shelf (COTS) is a term that references non-developmental items (NDI) sold in the commercial marketplace and used or obtained through government contracts. The set of rules for COTS is defined by the Federal Acquisition Regulation (FAR).

A COTS product is usually a computer hardware or software product tailored for specific uses and made available to the general public. Such products are designed to be readily available and user friendly. A typical example of a COTS product is Microsoft Office or antivirus software. A COTS product is generally any product available off-the-shelf and not requiring custom development before installation.

3. What are the different types of memory in a computer?

There are two main **kinds** of semiconductor **memory**, volatile and non-volatile. Examples of non-volatile **memory** are flash **memory** (used as secondary **memory**) and ROM, PROM, EPROM and EEPROM **memory** (used for storing firmware such as BIOS).

4. Discuss about DRAM and SRAM

Dynamic RAM (DRAM) and static RAM (SRAM) chips are the "working storage" in every computer. DRAM is the main memory in a computer and SRAM is used for high-speed caches and buffers. Both types are "byte addressable," which means that data can be read and written one byte at a time. Their major drawback is that DRAM and SRAM require power to hold their content. See dynamic RAM, static RAM and byte addressable.

5. Give a brief note on sensors.

A sensor is a device, module, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor. A sensor is always used with other electronics, whether as simple as a light or as complex as a computer.

Sensors are used in everyday objects such as touch-sensitive elevator buttons (tactile sensor) and lamps which dim or brighten by touching the base, besides innumerable applications of which most people are never aware. With advances in micro machinery and easy-to-use microcontroller platforms, the uses of sensors have expanded beyond the traditional fields of temperature, pressure or flow measurement,[1] for example into MARG sensors. Moreover, analog sensors such as potentiometers and force-sensing resistors are still widely used. Applications include manufacturing and machinery, airplanes and aerospace, cars, medicine, robotics and many other aspects of our day-to-day life.

Five marks questions with answers

1. Write short note on:

i. **DSP**
ii. **PLD**
iii. **ASIC**
iv. **COTS**

(i) Digital Signal Processors:

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

DSP includes following key units:

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

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

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

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

Examples:

1. Audio video signal processing, telecommunication and multimedia applications.
2. SOP(Sum of Products) calculation, convolution, FFT(Fast Fourier Transform), DFT (Discrete Fourier Transform), etc are some of the operation performed by DSP.

(ii) Application Specific Integrated Circuits(ASIC)

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

(iii) Programmable logic devices (PLD's)

- A PLD is an electronic component. It used to build digital circuits which are reconfigurable .A logic gate has a fixed function but a PLD does not have a defined

function at the time of manufacture.

- PLDs offer customers a wide range of logic capacity, features, speed, voltage characteristics.
- PLDs can be reconfigured to perform any number of functions at anytime.
- A variety of tools are available for the designers of PLDs which are inexpensive and help to develop, simulate and test the designs.

PLDs are following two major types.

- **CPLD (Complex Programmable Logic Device):**
  CPLDs offer much smaller amount of logic up to1000 gates.

- FPGAs (Field Programmable Gate Arrays):
  It offers highest amount of performance as well as highest logic density, the most features.

Advantages of PLDs:-

1) PLDs offer customer much more flexibility during the designcycle.
2) PLDs do not require long lead times for prototypes or production parts because PLDs are already on a distributor's shelf and ready for shipment.
3) PLDs can be reprogrammed even after a piece of equipment is shipped to a customer

(iv) Commercial off-the-shelf components (COTs)

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

Advantages of COTS:

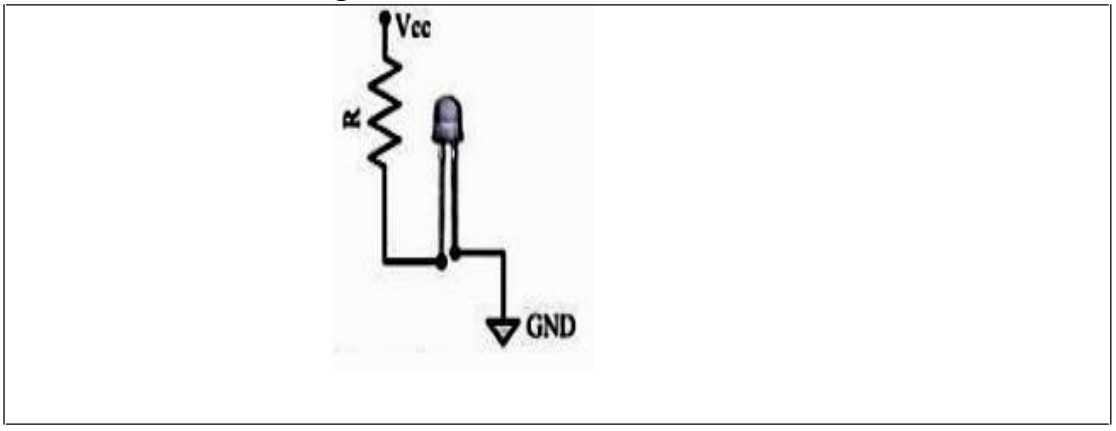
1) Ready to use
2) Easy to integrate
3) Reduces development time

Disadvantages of COTS:

4) No operational or manufacturing standard (all proprietary)
5) Vendor or manufacturer may discontinue production of a particular COTS product

2. DEFINE SENSOR AND ACTUATOR?

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



There are two ways to interface an LED to a microprocessor/microcontroller:

The Anode of LED is connected to the port pin and cathode to Ground:

In this approach the port pin sources the current to the LED when it is at logic high (i.e.1).

The Cathode of LED is connected to the port pin and Anode to V_{CC}:

In this approach the port pin sources the current to the LED when it is at logic

high(i.e.1).Here the port pin sinks the current and the LED is turned ON when the port pin is at Logic low (i.e.0)

3. Explain about Onboard communication interface?

Onboard Communication Interfaces

- These are used for internal communication of the embedded system i.e: communication between different components present on the system.
- Common examples of onboard interfaces are:
  - Inter Integrated Circuit(I2C)
  - Serial Peripheral Interface(SPI)
  - Universal Asynchronous Receiver Transmitter(UART)
  - 1-WireInterface
  - Parallel Interface

Inter Integrated Circuit(I2C)

- It is synchronous
- Bi-directional, half duplex , two wire serial interface bus
- Developed by Phillips semiconductors in1980
- It comprises of two buses:
  1. Serial clock–SCL
  2. Serial Data –SDA
- SCL generates synchronization clock pulses
- I2C is a shared bus system to which many devices can be connected
- Devices connected by I2C can act as either masteror slave
- The master device is responsible for controlling communication by initiating/ terminating data transfer.
- Devices acting as slave wait for commands from the master and respond to those commands.

4. Discuss about External or Peripheral Communication Interfaces?

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

Common examples of external interface sare:

- Universal Serial Bus(USB)
- IEEE 1394(Fire wire)
- Infrared(IrDA)
- Bluetooth
- Wi-Fi

- ZigBee
- General Packet Radio Service(GPRS)
- RS-232 C &RS-485

RS-232 C &RS-485

- It is wired, asynchronous, serial, full duplex communication
- RS 232 interface was developed by EIA(Electronic Industries Associates) In early1960s
- RS -232 is the extension to UART for external communications

RS-232 logic levels use:

- +3 to +25 volts to signify a "Space" (Logic 0)and
- -3 to -25 volts to signify a "Mark" (logic1).

RS 232 supports two different types of connectors:

DB 9 and DB 25 as shown in figurebelow

- RS 232 interface is a point to point communication interface and the devices involved are called as Data Terminating Equipment (DTE) And Data Communications Terminating Equipment(DCE)
- Embedded devices contain UART for serial transmission and generate signal levels as per TTL/CMOSlogic.
- A level translator IC (like Max 232) is used for converting the signal lines from UART to RS 232 signal lines for communication.
- The vice versa is performed on the receivingside.
- Converter chips contain converters for both transmitters andreceivers
- RS 232 is used only for point to pointconnections
- It is susceptible to noise and hence is limited to short distancesonly
- RS 422 is another serial interface fromEIA.
- It supports multipoint connections with 1 transmitter and 10receivers.
- It supports data rates up to 100Kbps and distance up to 400 ft
- RS 485 is enhanced version of RS 422 and supports up to 32 transmitters and 32receivers

5. **Explain about types of memory**

Introduction

Many types of memory devices are available for use in modern computer systems. As an embedded software engineer, you must be aware of the differences between them and understand how to use each type effectively

Types of RAM

The RAM family includes two important memory devices:

- Static RAM (SRAM) and
- Dynamic RAM (DRAM).

The primary difference between them is the lifetime of the data they store. SRAM retains its contents as long as electrical power is applied to the chip. If the power is turned off or lost temporarily, its contents will be lost forever. DRAM, on the other hand, has an extremely short data lifetime-typically about four milliseconds. This is true even when power is applied constantly.

In short, SRAM has all the properties of the memory you think of when you hear the word RAM. Compared to that, DRAM seems kind of useless. By itself, it is. However, a simple piece of hardware called a DRAM controller can be used to make DRAM behave more like SRAM. The job of the DRAM controller is to periodically refresh the data stored in the DRAM. By refreshing the data before it expires, the contents of memory can be kept alive for as long as they are needed. So DRAM is as useful as SRAM after all. When deciding which type of RAM to use, a system designer must consider access time and cost. SRAM devices offer extremely fast access times (approximately four times faster than DRAM) but are much more expensive to produce

Types of ROM

Memories in the ROM family are distinguished by the methods used to write new data to them (usually called programming), and the number of times they can be rewritten.

This classification reflects the evolution of ROM devices from hardwired to programmable to erasable-and-programmable. A common feature of all these devices is their ability to retain data and programs forever, even during a power failure.

The very first ROMs were hardwired devices that contained a preprogrammed set of data or instructions. The contents of the ROM had to be specified before chip production, so the actual data could be used to arrange the transistors inside the chip. Hardwired memories are still used, though they are now called "masked ROMs" to distinguish them from other types of ROM.

The primary advantage of a masked ROM is its low production cost. Unfortunately, the cost is low only when large quantities of the same ROM are required. One step up from the masked ROM is the PROM (programmable ROM), which is purchased in an unprogrammed state. If you were to look at the contents of an unprogrammed PROM, you would see that the data is made up entirely of 1's. The process of writing your data to the PROM involves a special piece of equipment called a device programmer. The device programmer writes data to the device one word at a time by applying an electrical charge to the input pins of the chip. Once a PROM has been programmed in this way, its contents can never be changed. If the code or data stored in the PROM must be changed, the current device must be discarded. As a result, PROMs are also known as one-time programmable (OTP) devices.

An EPROM (erasable-and-programmable ROM) is programmed in exactly the same manner as a PROM. However, EPROMs can be erased and reprogrammed repeatedly. To erase an EPROM, you simply expose the device to a strong source of ultraviolet light. (A window in

the top of the device allows the light to reach the silicon.) By doing this, you essentially reset the entire chip to its initial-unprogrammed-state. Though more expensive than PROMs, their ability to be reprogrammed makes EPROMs an essential part of the software development and testing process .As memory technology has matured in recent years, the line between RAM and ROM has blurred. Now, several types of memory combine features of both. These devices do not belong to either group and can be collectively referred to as hybrid memory devices.

Hybrid memories can be read and written as desired, like RAM, but maintain their contents without electrical power, just like ROM.

Two of the hybrid devices, EEPROM and flash, are descendants of ROM devices. These are typically used to store code. The third hybrid, NVRAM, is a modified version of SRAM. NVRAM usually holds persistent data. EEPROMs are electrically-erasable-and-programmable. Internally, they are similar to EPROMs, but the erase operation is accomplished electrically, rather than by exposure to ultraviolet light. Any byte within an EEPROM may be erased and rewritten. Once written, the new data will remain in the device forever-or at least until it is electrically erased. The primary tradeoff for this improved functionality is higher cost, though write cycles are also significantly longer than writes to a RAM. So you wouldn't want to use an EEPROM for your main system memory

.Flash memory combines the best features of the memory devices described thus far. Flash memory devices are high density, low cost, nonvolatile, fast (to read, but not to write), and electrically reprogrammable. These advantages are overwhelming and, as a direct result, the use of flash memory has increased dramatically in embedded systems. From a software viewpoint, flash and EEPROM technologies are very similar.

The major difference is that flash devices can only be erased one sector at a time, not byte-by-byte. Typical sector sizes are in the range 256 bytes to 16KB. Despite this disadvantage, flash is much more popular than EEPROM and is rapidly displacing many of the ROM devices as well. The third member of the hybrid memory class is NVRAM (non-volatile RAM). Nonvolatility is also a characteristic of the ROM and hybrid memories discussed previously. However, an NVRAM is physically very different from those devices. An NVRAM is usually just an SRAM with a battery backup. When the power is turned on, the NVRAM operates just like any other SRAM. When the power is turned off, the NVRAM draws just enough power from the battery to retain its data.

NVRAM is fairly common in embedded systems. However, it is expensive-even more expensive than SRAM, because of the battery-so its applications are typically limited to the storage of a few hundred bytes of system-critical information that can't be stored in any better way.

Objective question with answers

1. Which of the following is not an embedded system [ ]
   A) Smartphone B) Digital Camera
   C) MP3 Player D) Desktop Computer
2. Embedded systems are [ ]

A) General purpose B) Special purpose
C) Desktop Computer D) None

3. The first recognized embedded system is
   A) AGC B) Apple computer [ ]
   C)Calculator D) none
4. Which of the following is an intended purpose of embedded system [ ]
   A) Data communication B) Data Monitoring
   C) Data collection D) All of these
5. Application specific user interface example [ ]
   A) Mobile phone B) Multimeter
   C) ECG D) None
6. Electronic toy is an example of [ ]
   A) Small scale embedded system B) Medium scale embedded system
   C) Large scale embedded system D) None
7. Example of first generation embedded system [ ]
   A) Z80 and 8085 B) 8086
   C) Both D) None
8. Example of third generation embedded system [ ]
   A) Z80 and 8085 B) 8086
   C) SCADA D) None
9. Real time embedded systems are classified into how many types [ ]
   A) 1 B) 2
   C) 6 D) None
10. The user interface of UGC is called as [ ]
    A) DSKY B) Mouse
    C) Both D) None

Q.No	1	2	3	4	5	6	7	8	9	10
Key	D	B	A	D	A	A	A	C	B	A

Fill in the blanks question with answers

1. Embedded systems are ----------
2. The first recognized modern embedded system is ----------
3. A digital multimeter is an example of an embedded system for --- ---------
4. ASIC stands for ----------
5. Response is a measure of ----------
6. A stepper motor is an ---------- device
7. Which was the first mass produced embedded system ----------

8. Second generations embedded system example ----------
9. ECG stands for ----------
10. MTBF stands for ----------

S.No	Answer
1.	Special purpose
2.	AGC
3.	monitoring
4.	application specific integrated circuit
5.	quickness of system
6.	electro mechanical
7.	Minuteman
8.	Scada
9.	electrocardiograms
10.	Mean time between failures

UNIT-III

1. What is cross compiler

It converts the application to a target processor specific code which is different from the processor architecture on which the compiler is running

2. Define link editor

In computing, a linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file.

3. In how many ways mixing of assembly and high level language can be done

Three ways

- Mixing assembly with c
- Mixing c with assembly
- Invoking assembly instructions inline from the high level code

4. Define PCB

A printed circuit board (PCB) mechanically supports and electrically connects electronic components using conductive tracks, pads and other features etched from copper sheets laminated onto a non-conductive substrate.

5. Discuss real time clock

A real-time clock (RTC) is a computer clock (most often in the form of an integrated circuit) that keeps track of the current time.

Three marks questions with answers

6. Define watch dog timer

The watchdog timer is a timing device that resets the system after a predefined timeout. This time is usually configured and the watchdog timer is activated within the first few clock cycles after power-up. It has a number of applications. In many embedded systems reset by a watchdog timer is very essential because it helps in rescuing the system if a fault develops and the program gets stuck. On restart, the system can function normally. Most microcontrollers have on-chip watchdog timers

7. What is assembly language

An assembly (or assembler) language, often abbreviated ASM, is a low-level programming language for a computer, or other programmable device, in which there is a very strong (but often not one-to-one) correspondence between the language and the architecture's machine code instructions.

8. What is the purpose of reset circuit

It ensures that the device is not operating at a voltage level where the device is not guaranteed to operate during system power on,the reset circuit brings the internal registers of the processor to a known state and starts the firmware execution from the reset vector

4. What is the significance of watchdog timer in ES?

It is a timing device which is set to predefined time interval and some task is to be performed at that time. It is used to reset original state when an inappropriate event take place.It is usually operated by counter device.

5. Discuss about importance of RTC

RTCs are essential; if the battery fails, it must be replaced to ensure continued operation. A dead battery can be diagnosed with an error message at startup or if the user finds that the clock or the setup configuration has become corrupted, flaky or odd.ion.

Benefits of RTCs include:

- RTC ICs have proved to be more precise than other methods — like programming the timer of the controller.
- It frees the main system from time-critical tasks.
- It has low power consumption and improved frequency stability.

Five marks questions with answers

1. **Discuss about super loop based approach**

The 'Super Loop' Architecture for Embedded C programming
an Embedded C Application, there are set of statements which need to be executed forever. Because there are no operating system to return to or an embedded device is running until the power supply is removed.
So, to run set of statements, we need a loop that must not be finished, such kind of loops are known as 'Super Loop' or 'Infinite Loop'.
There is only one difference between 'Super Loop' and 'Infinite Loop': There may only one 'Super Loop' but the 'Infinite Loop' may be infinite (i.e. there is no limit of infinite loops in a program).
Consider the given example (Syntax)

```
intmain()
{
        /*PINs configuration, interrupts, timers initialization etc*/
        initialize();

        /*Super/Infinite Loop*/
        while(1)
        {
                /*application's tasks*/
                ...;
                anything();
                ...;
        }

        /*program's execution will never reach here*/
        return0;
}
```

In this example, we used while (1) as 'Super Loop', here while is a looping statement and 1 is a non zero value that will also true and programming will run forever.

Note: Since, program's execution will not reach to end of the program, hence return 0 will never be executed, we can also use void main() instead of int main() and then there is no need to use return 0.

initialize()

Here, initialize() is not a standard library function, we just wrote this function as an example. That mean at this section before 'Super Loop' you can place initialization related codes (like initializations of interrupts, times, pins configuration, memory and other attached devices).

anything()

Here, anything() is also not a standard library function, at this section you can actual code that you want to execute again and again to keep running the device.

Benefits of 'Super Loop'

These are some of the benefits of using **super loop in an Embedded Application**.

- 'Super Loop' runs give statements within the scope forever.
- It is very simple to use, edit, debug and understand.
- Less or no dependencies on the hardware.

2. Explain about embedded based OS approach

Embedded operating systems are usually used for hardware that have very little computing power, little RAM/ROM and a slow CPU, so they tend to be very specific in their applications and scope. They are usually made using assembly language in order to really take advantage of the limited computing resources, since it is the closest to machine language and is able to squeeze every drop of computing power available. This means that the OS is optimized for whatever hardware it was developed for and will not be compatible with other hardware systems with different configurations.

In most embedded OSs, the applications are built in to the OS or part of the OS, so they are loaded immediately when the OS starts. The most common examples of devices with an embedded OS would be cell phones before Android and iOS popularized the mobile operating system, which may still be considered as embedded but are also arguably desktop-like in the way they handle tasks and apps and their access to vast amounts of computing power. Embedded OSs can also be found in cars, large laser printers, some home appliances, and even military systems.

Notable embedded OSs currently in use by consumers include:

- Symbian - Used in cell phones, mainly ones made by Nokia
- Embedded Linux - Of which Android is a subset, used in many other devices like printers
- BlackBerry OS - For BlackBerry phones
- iOS - Subset of Mac OS X, used in Apple's mobile devices
- Palm OS
- Windows Mobile

3. Discuss about Assembly Language Programming

Assembly codes sensitive to the processor, memory, ports and devices hardware Gives a precise control of the processor internal devices Enables full use of processor specific features in its instruction set and its addressing modes
Machine codes are compact, processor and memory sensitive System needs a smaller memory. Memory needed does not depend on the programmer data type selection and rule declarations Not the compiler specific and library functions specific
Device driver codes may need only a few assembly instructions. Bottom-up-design approach

Assembly languages were developed to provide mnemonics or symbols for the machine level code instructions. Assembly language programs consist of mnemonics, thus they should be translated into machine code. A program that is responsible for this conversion is known as assembler. Assembly language is often termed as a low-level language because it directly works with the internal structure of the CPU. To program in assembly language, a programmer must know all the registers of the CPU
Different programming languages such as C, C++, Java and various other languages are called high-level languages because they do not deal with the internal details of a CPU. In contrast, an assembler is used to translate an assembly language program into machine code (sometimes also called object code or opcode). Similarly, a compiler translates a high-level language into machine code. For example, to write a program in C language, one must use a C compiler to translate the program into machine language.

4. Analyze about Structure of Assembly Language

An assembly language program is a series of statements, which are either assembly language instructions such as ADD and MOV, or statements called **directives**.
An **instruction** tells the CPU what to do, while a **directive** (also called **pseudo-instructions**) gives instruction to the assembler. For example, ADD and MOV instructions are commands which the CPU runs, while ORG and END are assembler directives. The assembler places the opcode to the memory location 0 when the ORG directive is used, while END indicates to the end of the source code. A program language instruction consists of the following four fields

```
[ label: ]  mnemonics  [ operands ]  [;comment ]
```

A square bracket ([ ]) indicates that the field is optional.

- The **label field** allows the program to refer to a line of code by name. The label fields cannot exceed a certain number of characters.

- The **mnemonics** and **operands fields** together perform the real work of the program and accomplish the tasks. Statements like ADD A , C & MOV C, #68 where ADD and MOV are the mnemonics, which produce opcodes ; "A, C" and "C, #68" are operands. These two fields could contain directives. Directives do not generate machine code and

are used only by the assembler, whereas instructions are translated into machine code for the CPU to execute.

5. Explain about brown out protection circuit

Well almost all microcontrollers have Brownout protection inbuilt in them but when connecting a controller to an industry sensor and controlling devices (which are extremely costly) its better we know what is a brownout and how is it detected in a microcontroller cause many devices in low to medium scale industry may not be as immune to brownout as our controller.

So what's a brown out??

Brownout as an intentional or unintentional drop in voltage in an electrical power supply system. Intentional brownouts are used for load reduction in an emergency. The reduction lasts for minutes or hours, as opposed to short-term voltage sag or dip.

The brown out can cause one of the three things for a dc supply system. These things in turn can damage the connected embedded systems.An unregulated direct current supply will produce a lower output voltage for electronic circuits. The output ripple voltage will decrease in line with the usually reduced load current.

A linear direct current regulated supply will maintain the output voltage unless the brownout is severe and the input voltage drops below the drop out voltage for the regulator, at which point the output voltage will fall and high levels of ripple from the rectifier/reservoir capacitor will appear on the output.

A switched-mode power supply which has a regulated output will be affected. As the input voltage falls, the current draw will increase to maintain the same output voltage and current, until such a point that the power supply malfunctions.

Brownouts can cause unexpected behavior in systems with digital control circuits. Reduced voltages can bring control signals below the threshold at which logic circuits can reliably detect which state is being represented. As the voltage returns to normal levels the logic can latch at an incorrect state; even can't happen states become possible. The seriousness of this effect and whether steps need to be taken by the designer to prevent it depends on the nature of the equipment being controlled; for instance a brownout may cause a motor to begin running backwards.

6. Analyze the reset circuit

Reset means that the processor starts the processing of instructions from a starting address. That address is one that is set by default in the processor program counter (or instruction pointer and code segment registers in x86 processors) on a power-up. From that address in memory, the fetching of program-instructions starts following the reset of the processor. [In certain processors, for example, 68HC11 and HC12, there are two

start-up addresses. One is as per power-up reset vector and other is as per reset vector after the Reset instruction or after a time-out (for example from a watchdog timer)]. The reset circuit activates for a fixed period (a few clock cycles) and then deactivates. The processor circuit keeps the reset pin active and then deactivates to let the program proceed from a default beginning address. The reset pin or the internal reset signal, if connected to the other units (for example, I/O interface or Serial Interface) in the system, is activated again by the processor; it becomes an outgoing pin to enforce reset state in other sister units of the system. On deactivation of the reset that succeeds the processor activation, a program executes from start-up address. Reset can be activated by one of the following:

- An external reset circuit that activates on the power-up, on switching-on reset of the system or on detection of a low voltage (for example < 4.5V when what is required is 5V on the system supply rails). This circuit output connects to a pin called the reset pin of the processor. This circuit may be a simple RC circuit, an external IC circuit or a custom-built IC. The examples of the ICs are MAX 6314 and Motorola MC 34064.

- By (a) software instruction or (b) time-out by a programmed timer known as watchdog timer (or on an internal signal called COP in 68HC11 and 68HC12 families) or (c) a clock monitor detecting a slowdown below certain threshold frequencies due to a fault.

Objective question with answers

1. RTC acronym [ ]
   A)real tick time B) real clock time
   C)both D) none
2. LJMP means [ ]
   A)Less jump B) little jump
   C)large jump D) none
3. _____from keil software is an example of library creator [ ]
   A)lib51 B) lib71
   C)a51 D) none
4. _____from keil software is an example of linker creator [ ]
   A)al51 B) bl51
   C)a51 D) none
5. _____from keil software is an example for object to hex file converter [ ]

A)oh51 B) bl51
C)a51 D) none

6. _____ is simple and straight forward without any os related overhead [ ]
A)super loop B)embedded os based
C)a51 D) none
7. _____ are examples of RTOS employed in embedded product development [ ]
A)linux B)vxworks
C)both D) none
8. The accuracy of crystal oscillator is normally expressed in terms of ____ [ ]
A)pam B)prm
C)ppm D) none
9. _____ is the place holder for arranging different hardware components required to build embedded product [ ]
A)PCB B)PAB
C)both D) none
10. _____ are the manufacturer of RTC chips [ ]
A)maxim B)atmel
C)both D)intel

Q.No	1	2	3	4	5	6	7	8	9	10
Key	B	C	A	B	A	A	C	C	A	A

Fill in the blanks question with answers

1. ASCII stands for _____
2. _____ is used for monitoring the firmware execution
3. _____is a system component keeping track of time
4. Symbols used in machine language for representing the machine language is called as _____
5. _____ is an embedded firmware model which executes tasks in infinite loop at a predefined order
6. In high level language the firmware is developed using _____
7. _____ is tool for native development environment
8. Specially formatted , ordered program collection of object modules is called as _____
9. PCB acronym _____
10. Super loop model is also called as _____

Q.No	Answer
1	American standard code for information interchange
2	Watch dog timer

3	Real time clock
4	Mnemonics
5	Super loop model
6	C
7	Compiler
8	Library
9	Printed circuit board
10	Conventional based model

UNIT-IV

Two marks questions with answers

1. **What is process life cycle**

   Process Control Block (PCB, also called Task Controlling Block,[1] Entry of the Process Table,[2] Task Struct, or Switchframe) is a data structure in the operating systemkernel containing the information needed to manage a particular process. The PCB is "the manifestation of a process in an operating system

2. **What is user space and kernel space**

   User space is that set of memory locations in which user processes (i.e., everything other than the kernel) run.
   Kernel space can be accessed by user processes only through the use of system calls.

3. **Define light weight processes**

   A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes. What isd virtual memory

4. **Discuss preemptive scheduling**

   Preemption as used with respect to operating systems means the ability of the operating system to preempt (that is, stop or pause) a currently scheduled task in favour of a higher priority task.

Three marks questions with answers

1. **Define round robin scheduling algorithm**

Round robin scheduling (RRS) is a job-scheduling algorithm that is considered to be very fair, as it uses time slices that are assigned to each process in the queue or line. Each process is then allowed to use the CPU for a given amount of time, and if it does not finish within the allotted time, it is preempted and then moved at the back of the line so that the next process in line is able to use the CPU for the same amount of time

2. **Explain context switching**

A context switch (also sometimes referred to as a process switch or a task switch) is the switching of the CPU (central processing unit) from one process or thread to another. Context switches can occur only in kernel mode. Kernel mode is a privileged mode of the CPU in which only the kernel runs and which provides access to all memory locations and all other system resources. Other programs, including applications, initially operate in user mode, but they can run portions of the kernel code via system calls.

3. **What is process life cycle**

Process Control Block (PCB, also called Task Controlling Block,[1] Entry of the Process Table,[2] Task Struct, or Switch frame) is a data structure in the operating systemkernel containing the information needed to manage a particular process. The PCB is "the manifestation of a process in an operating system

4. **What is virtual memory**

Virtual memory is a memory management capability of an OS that uses hardware and software to allow a computer to compensate for physical memory shortages by temporarily transferring data from random access memory (RAM) to disk storage. Virtual address space is increased using active memory in RAM and inactive memory in hard disk drives (HDDs) to form contiguous addresses that hold both the application and its data.

5. **What is the difference between mutexes and semaphores?**

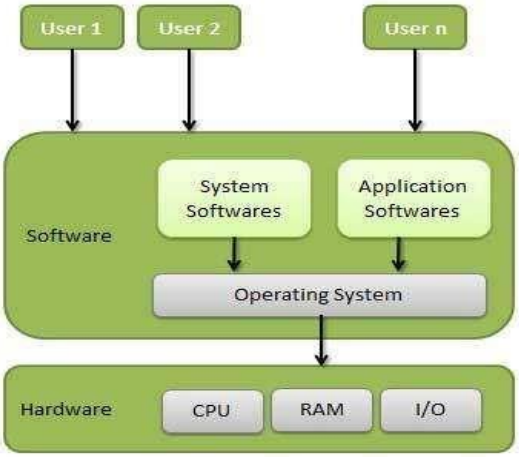
Semaphores are the synchronization tool to overcome critical section problem. Mutex is also a tool that is used to provide deadlock free mutual exclusion. It protects access to every critical data item, if the data is locked and is in use,it either waits for the thread to finish or awakened to release the lock from its inactive state.

Five marks questions with answers

1. Define Operating System and what are the important functions of OS

An Operating System (OS) is an interface between a computer user and computer hardware.

An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.



Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Some important functions of OS are

- MemoryManagement
- ProcessorManagement
- DeviceManagement
- FileManagement
- Security
- Control over systemperformance
- Jobaccounting
- Error detectingaids
- Coordination between other software andusers

2. What is kernel and what are the types of kernel

The kernel is a part of a software. It is like a bridge between the shell and hardware. It is responsible for running programs and providing secure access to the machine's hardware. The kernel is used for scheduling, i.e., it maintains a time table for all processes

Types of Kernels

- ✓ Monolithic kernel
- ✓ Micro kernel
- ✓ Exo kernel
- ✓ Hybrid kernel

Monolitnic kernel

Monolithic kernel is a single large processes running entirely in a single address space. It is a single static binary file.

Micro kernel

Micro kernels, the kernel is broken down into separate processes, known as servers. Some of the servers run in kernel space and some run in user-space.

Exo kernel

Exo kernel is an operating system developed at the Massachusetts Institute of Technology that seeks to provide application-level management of hardware resources. The exo kernel architecture is designed to separate resource protection from management to facilitate application-specific customization.

Hybrid kernel

A hybrid kernel is an operating system kernel architecture that attempts to combine aspects and benefits of microkernel and monolithic kernel architectures used in computeroperating systems.

3. **Discuss Real-Time OperatingSystem**

A real-time system is defined as a data processing system in which the time interval requiredtoprocessandrespondtoinputsissosmallthatitcontrolstheenvironment.The time taken by the system to respond to an input and display of required updated informationistermedastheresponsetime.Sointhismethod,theresponsetimeisvery less as compared to onlineprocessing.

Real-time systems are used when there are rigid time requirements on the operation ofa processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems,etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

Soft real-time systems

Softreal-timesystemsarelessrestrictive.Acriticalreal-timetaskgetspriorityoverother tasksandretainstheprioritytillitcompletes.Softreal-timesystemshavelimitedutility thanhardreal-timesystems.Forexample,multimedia,virtualreality,AdvancedScientific Projects like undersea exploration and planetary rovers,etc.

4. Discuss about memory management and device management

Memory management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management:.

- In multiprogramming, the OS decides which process will get memory when and howmuch.
- Allocates the memory when a process requests it to doso.
- De-allocates the memory when a process no longer needs it or has been terminated.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management:

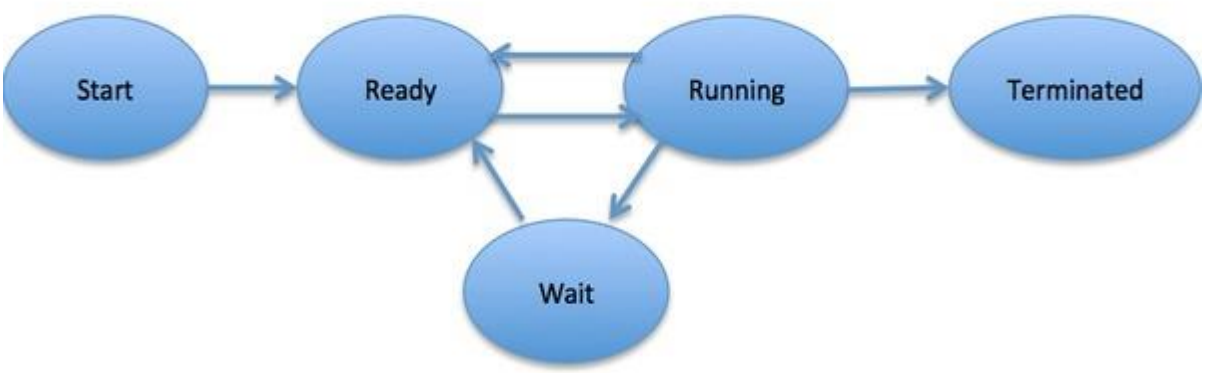
- Keeps tracks of all devices. The program responsible for this task is known asthe **I/O controller.**
- Decides which process gets the device when and for how muchtime.
- Allocates the device in the most efficientway

5. Define process and process state transitions

Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion

Process transitions



Start

This is the initial state when a process is first started/created.

Ready

The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process

Running

Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

Waiting

Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.

Terminated or Exit

Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.

6. Define multitasking and what are the types of it

The ability to execute more than one *task* at the same time, a task being a program. The terms *multitasking* and *multiprocessing* are often used interchangeably, although multiprocessing implies that more than one CPU is involved.
In multitasking, only one CPU is involved, but it switches from one program to another so quickly that it gives the appearance of executing all of the programs at the same time

.
There are two basic types of multitasking:

- *preemptive*
- *cooperative.*

In preemptive multitasking, the operating system parcels out CPU *time slices* to each program.
In cooperative multitasking, each program can control the CPU for as long as it needs it. If a program is not using the CPU, however, it can allow another program to use it temporarily.

OS/2, Windows 95, Windows NT, the Amiga operating system and UNIXuse preemptive multitasking,

whereas Microsoft Windows 3.x and the MultiFinder (for Macintosh computers) use cooperative multitasking.

Objective question with answers

1. Which of the following is an example hard real time system [ ]
   B) Air bag control systems B) Digital Camera
   C)ATM D)none
2. A Thread does not have its own _____ memories [ ]
   C) Data memory B)heap memory
   C) both D) None
3. The example of monolithic kernel is [ ]
   A)Linux B) Apple computer
   C)Unix D) none
4. A good scheduling algorithm has [ ]
   D) High cpu utilization B) medium cpu utilization
   C) low cpu utilization D) All of these
5. TCB stands for [ ]
   a. Task control block B) task change block
   C) the control block D) None
6. A process contains at least ------ thread [ ]
   a. 1 B) 2
   C) 3 D) None
7. The core of the operating system is called [ ]
   a. Kernel B) rtos
   C) both D) None
8. TAT stands for [ ]
   a. Turn around time B) turn round time
   C) through around time D) None
9. SRT means [ ]

a. Shortest remaining time
B) short run time
C) Both
D) None

10. posix.4a deals with [ ]
A) Thread extensions
B) process extension
C) real time extension
D) None

Q.No	1	2	3	4	5	6	7	8	9	10
Key	A	C	A	A	A	A	A	A	A	A

Fill in the blanks question with answers

1. Virtual memory is _____
2. First In First Out is _____scheduling algorithm
3. Example of soft real time system_____
4. A _____is a program or part of it in execution
5. A _____ is preemptive that can execute code
6. LIFO stands for_____
7. Example for RTOS _____
8. Operating systems with a generalized kernel are known as _____
9. RTOS stands for _____
10. The ability of a system to execute multiple processes simultaneously is known as_____

Q.No	Answer
1.	An extremely large main memory
2.	Non preemptive
3.	ATM
4.	Process
5.	Thread
6.	Last in first out
7.	QNX
8.	General purpose operating system
9.	Real time operating system
10.	Multiprocessing

UNIT-V

Two marks questions with answers

1. What is starvation

Starvation: Starvation is a resource management problem where a process does not get the resources it needs for a long time because the resources are being allocated to other processes

2. What is aging

Aging: Aging is a technique to avoid starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the request is priority as time passes and must ensure that a request will eventually be the highest priority request (after it has waited long enough)

3. What are the four conditions for deadlock

- Mutual exclusion
- Hold and wait
- No resource preemption
- Circular wait

4. Define mutex

In computer programming, a mutex (mutual exclusion object) is a program object that is created so that multiple program thread can take turns sharing the same resource, such as access to a file. Typically, when a program is started, it creates a mutex for a given resource at the beginning by requesting it from the system and the system returns a unique name or ID for it.

Three marks questions with answers

1. What is aging

Aging: Aging is a technique to avoid starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the request is priority as time passes and must ensure that a request will eventually be the highest priority request (after it has waited long enough)

2. What is priority ceiling

In real-time computing, the priority ceiling protocol is a synchronization protocol for shared resources to avoid unbounded priority inversion and mutual deadlock due to wrong nesting of critical sections. In this protocol each resource is assigned a priority ceiling, which is a priority equal to the highest priority of any task which may lock the resource. The protocol works by temporarily raising the priorities of tasks in certain situations, thus it requires a scheduler that supports dynamic priority scheduling.

3. What is live lock

A condition that occurs when two or more processes continually change their state in response to changes in the other processes. The result is that none of the processes will complete. An analogy is when two people meet in a hallway and each tries to step around the other but they end up swaying from side to side getting in each other's way as they try to get out of the way

4. Discuss critical section

Critical section is a piece of code that accesses a shared resource (data structure or device) that must not be concurrently accessed by more than one thread of execution. A critical section will usually terminate in fixed time, and a thread, task or process will have to wait a fixed time to enter it (aka bounded waiting). Some synchronization mechanism is required at the entry and exit of the critical section to ensure exclusive use, for example a semaphore.

5. Explain concept of pipe

A pipe is a technique for passing information from one program process to another. Unlike other forms of inter process communication (IPC), a pipe is one-way communication only. Basically, a pipe passes a parameter such as the output of one process to another process which accepts it as input. The system temporarily holds the piped information until it is read by the receiving process.

Five marks questions with answers

1. What is deadlock and explain about Coffman conditions?

Deadlock is when two or more tasks never make progress because each is waiting for some resource held by another process

Example: too little memory

Two processes each demand 1.5 Gb of memory on a machine with only 2 Gb (and no virtual memory). The operating system kindly gives each 1 Gb exactly. Neither process can make progress until the other gives up some of its memory.

Necessary conditions

There are four conditions which must hold for deadlock to occur as classically defined. These are known as *Coffman's conditions* from a 1971 survey paper of which Coffman was the first author in alphabetical order (Coffman, E.G., M.J. Elphick, and A. Shoshani, System Deadlocks, ACM Computing Surveys, 3(2):67–78, 1971; coffman_deadlocks.pdf).

- ✓ **Mutual exclusion**. There must be some resource that can't be shared between processes.
- ✓ **Hold and wait**. Some process must be holding one resource while waiting for another.
- ✓ **No preemption**. Only the process holding a resource can release it.
- ✓ **Circular wait**. There must be some cycle of waiting processes P_1, P_2, ...,P_n such that each process P_i is waiting for a resource held by the next process in the cycle. (Note that this implies hold-and-wait.)
- ✓

2. What is semaphore and explain about its types

Semaphores

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of

interacting processes. This integer variable is called **semaphore**. So it is basically a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P() and V() respectively.

The classical definition of wait and signal are :

- Wait: decrement the value of its argument S as soon as it would become non-negative.
- Signal: increment the value of its argument, S as an individual operation.
-

Properties of Semaphores

- Simple
- Works with many processes
- Can have many different critical sections with different semaphores
- Each critical section has unique access semaphores
- Can permit multiple processes into the critical section at once, if desirable

Types of Semaphores

Semaphores are mainly of two types:

Binary Semaphore

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called *Mutex*. A binary semaphore is initialized to 1 and only takes the value 0 and 1 during execution of a program.

Counting Semaphores

These are used to implement bounded concurrency.

Limitations of Semaphores

- Priority Inversion is a big limitation of semaphores.
- Their use is not enforced, but is by convention only.
- With improper use, a process may block indefinitely. Such a situation is called Deadlock.

3. Discuss a)message passing b)shared memory

Message Passing

Message passing is a form of communication used in interprocess communication. Communication is made by the sending of messages to recipients. Each process should be able to name the other processes. The producer typically uses send() system call to send messages, and the consumer uses receive() system call to receive messages. These system calls can be either synchronous or asynchronous, and could either be between processes running on a single machine, or could be done over the network to coordinate machines in a distributed system. This allows the producer to transfer data to the consumer as it is created

Shared Memory

Shared Memory is an OS provided abstraction which allows a memory region to be simultaneously accessed by multiple programs with an intent to provide communication among them. One process will create an area in RAM which other processes can access (this is typically done using system calls mmap, shmgetetc).

Normally the OS prevents processes from accessing the memory of another process, but the Shared Memory features in the OS can allow data to be shared. Since both processes can access the shared memory area like regular working memory, this is a very fast way of communication (as opposed to other mechanisms of IPC). On the other hand, it is less powerful, as for example the communicating processes must be running on the same machine (whereas other IPC methods can use a computer network), and care must be taken to avoid issues if processes sharing memory are running simultaneously and may try to edit the shared buffer at the same time

4. Explain about device driver

The goal of designing a device driver is to hide the hardware completely. Attempts to hide the hardware completely are difficult.

For example all Flash memory devices share the concept of sectors. An erase operation can be performed only on an entire sector. Once erased individual bites or words can be rewritten. Device drivers for embedded systems are quite different from the workstation counter parts. In modern computers workstation device drivers are most often concerned with satisfying the requirement of the operating system. There are three benefits of good device driver:

i. Modularization, it makes the structure of the overall software is easier to understand.

ii. There exists only one module that interacts directly with the peripheral's registers making communication easier.

iii. Software changes that result from hardware changes are localized to the device driver.

Components of a Device Driver A device driver can be implemented (as components) in the following steps:

- A data structure that overlays the memory-mapped control and status registers of the device: This basic step involves creating a C style structure that is actually a map of the registers present in the device. These registers can be found out by referring to the data sheet for the device. A table is created which maps the control register to their relative offsets.
- A set of variables to track the current state of the hardware and device driver: It involves listing out the required variables needed to keep track of the state of the hardware and device driver
- . Initialize the hardware: Once the variables to be used are known the next step in device driver programming is to initialize the hardware. Next functions can be written to control the device.
- A set of routines that provide an API for users of the device driver This involves writing different functions that will implement the various tasks listed to be performed by the device.
- Interrupt service routines Once the required functions and routines are coded the thing remaining to be done is to identify and write routines for servicing the interrupts

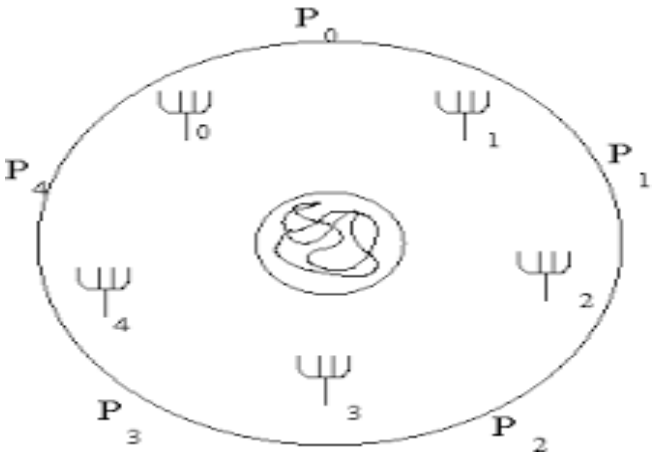
5. What is the meaning of the term busy waiting? Explain why spinlocks are not appropriate for single-processor systems yet are often used in multiprocessor systems

Busy waiting means that a process is waiting for a condition to be satisfied in a tight loop without relinquishing the processor. Alternatively, a process could wait by relinquishing the processor, and block on a condition and wait to be awakened at some appropriate time in the future. Busy waiting can be avoided but incurs the overhead associated with putting a process to sleep and having to wake it up when the appropriate program state is reached

Spinlocks are not appropriate for single-processor systems because the condition that would break a process out of the spinlock can be obtained only by executing a different process. If the process is not relinquishing the processor, other processes do not get the opportunity to set the program condition required for the first process to make progress. In a multiprocessor system, other processes execute on other processors and thereby modify the program state in order to release the first process from the spinlock

6.Discuss about dining Philosophers Problem

Dining Philosophers Problem The Dining Philosophers Problem is an illustrative example of a common computing problem in concurrency. The dining philosophers problem describes a group of philosophers sitting at a table doing one of two things - eating or thinking. While eating, they are not thinking, and while thinking, they are not eating. The philosophers sit at a circular table each with a bowl of spaghetti. A chopstick is placed in between each philosopher, thus each philosopher has one chopstick to his or her left and one chopstick to his or her right. As spaghetti is difficult to serve and eat with a single chopstick, it is assumed that a philosopher must eat with two chopsticks. The philosopher can only use the chopstick on his or her immediate left or right. The philosophers never speak to each other, which creates a dangerous possibility of deadlock. Deadlock could occur if every philosopher holds a left chopstick and waits perpetually for a right chopstick (or vice versa). Originally used as a means of illustrating the problem of deadlock, this system reaches deadlock when there is a 'cycle of unwarranted requests'. In this case philosopher P1 waits for the chopstick grabbed by philosopher P2 who is waiting for the chopstick of philosopher P3 and so forth, making a circular chain. The lack of available chopsticks is an analogy to the locking of shared resources in real computer programming. Locking a resource is a common technique to ensure the resource is accessed by only one program or chunk of code at a time. The challenge occurs when there are multiple resources which must be acquired individually. When several programs are involved in locking multiple resources, deadlock can occur. For example, one program needs two files to process. When two such programs lock one file each, both programs wait for the other one to unlock the other file, which will never happen



Objective question with answers

1. IPC stands for [ ]
   A)inter process communication B) intra process control
   C)inter process control D)none

2. The implementation pipe is _____dependent [ ]
   A)OS B) IPC
   C)both D)none

3. _____specifies the structure of pipes [ ]
   A)dwflags B) dwsize
   C)dwword D)none
4. Pipe follow _____ architecture [ ]
   A)master slave B) client client
   C)client server D)none
5. RMI acronym [ ]
   A)remote method invocation B) remote invocation
   C)both D)none
6. _____ is a piece of software that acts as an bridge between the operating system and the hardware [ ]
   A)device driver B) racing
   C)FIFO D)LIFO
7. A common data sharing problem where two processes concurrently access as shared buffer with fixed size is [ ]
   A)LIFO B) readers writerS problem
   C)dining philosopher's problem D)producer-consumer problem

8. IPC mechanism is _____ dependent [ ]
   A)user space B) OS kernel
   C)monolithic kernel D)none
9. Examples of sleep and wake up based mutual exclusion [ ]

A)semaphores B) mutex
C)events D)all of these

10. Examples of busy waiting based mutual exclusion [ ]
A)test and set B) flags
C)both D)all of these

Q.No	1	2	3	4	5	6	7	8	9	10
Key	A	A	A	C	C	A	D	B	D	C

Fill in the blanks question with answers

1. Multiple processes compete each other to access and manipulate shared data concurrently is _____
2. A process does not get the CPU or system resources required to continue its execution for a long time is called_____
3. A system resource for implementing mutual exclusion in shared resources access or for restricting the access to the shared resource is_____
4. The binary semaphore implementation for exclusive resources access under certain OS kernel is called_____
5. A queue for holding messages for exchanging between processes of a multitasking system is called_____
6. various_____ requirements need to be evaluated before the selection of an RTOS for an embedded design
7. The ability of a system to execute multiple processes simultaneously is known as_____
8. In priority ceiling a priority is associated with _____shared resource
9. The consumer tries to read data from an empty buffer is called as _____
10. The producer tries to put data to a full buffer is called as _____

Q.No	Answer
1	Racing
2	Starvation
3	Semaphore
4	Mutex
5	Message queue
6	Functional and non functional requirements
7	Multiprocessing
8	each
9	Buffer under run
10	Buffer over run

17. CO-PO Mapping:

CO\PO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	M		L	H								
CO2	L	H					M					
CO3	M	H		L								
CO4			H							M		
CO5	H	M						L				

H : HIGH

M : MEDIUM

L : LOW