CBCS SCHEME

USN | | | | | | | | | | |

18CS44

Fourth Semester B.E. Degree Examination, July/August 2022
Microcontroller and Embedded Systems

Time: 3 hrs. Max. Marks: 100

Note: *Answer any FIVE full questions, choosing ONE full question from each module.*

Module-1

1 a. Compare Microprocessors and Microcontrollers. **(06 Marks)**
b. Discuss the ARM design Philosophy. **(06 Marks)**
c. With a neat diagram, explain the four main hardware components of an ARM based embedded device. **(08 Marks)**

OR

2 a. Explain the ARM Core data flow model with a neat diagram. **(08 Marks)**
b. Draw the basic layout of a generic program status register and briefly explain the various fields. **(06 Marks)**
c. What is Pipelining? Illustrate it with a simple example. **(06 Marks)**

Module-2

3 a. Explain the different Data Processing Instructions in ARM. **(10 Marks)**
b. Briefly explain the different Load – Store Instruction categories used with ARM. **(10 Marks)**

OR

4 a. Write a program for forward and backward branch by considering an example. **(06 Marks)**
b. Explain Co – Processor Instructions of ARM processor. **(06 Marks)**
c. Write a note on Profiling and Cycle Counting. **(08 Marks)**

Module-3

5 a. What is an Embedded System? Differentiate between general purpose computing system and embedded system. **(06 Marks)**
b. List any four purposes of Embedded system with examples. **(08 Marks)**
c. Write short notes on : i) Real Time Clock ii) Watch Dog Timer. **(06 Marks)**

OR

6 a. Briefly describe the classification of Embedded system. **(08 Marks)**
b. Explain the following :
i) I 2 C Bus ii) S P I Bus iii) Reset Circuit iv) 1 – Wire Interface. **(12 Marks)**

Module-4

7 a. What are the Operational and Non – Operational Quality Attributes of an Embedded system? **(10 Marks)**
b. Explain the different communication buses used in Automotive applications. **(06 Marks)**
c. Design an FSM model for Tea / Coffee vending machine. **(04 Marks)**

OR

Important Note : 1. On completing your answers, compulsorily draw diagonal cross lines on the remaining blank pages.
2. Any revealing of identification, appeal to evaluator and /or equations written eg, 42+8 = 50, will be treated as malpractice.

8 a. Explain the Fundamental issues in Hardware Software Co - design. (06 Marks)
b. Explain the Assembly language based Embedded firmware development with a diagram. (06 Marks)
c. With a neat block diagram, how source file to object file translation takes place in High level language based firmware development. (08 Marks)

Module-5

9 a. With a neat diagram, explain Operating System Architecture. (08 Marks)
b. Explain Multithreading. (06 Marks)
c. Explain the concept of Binary Semaphore. (06 Marks)

OR

10 a. Explain the role of Integrated Development Environment (IDE) for Embedded Software development. (08 Marks)
b. Write a note on Message passing. (08 Marks)
c. Explain the concept of deadlock with a neat diagram. (04 Marks)

* * * * *

USN	

CMRIT — CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 4 – March 2022

Sub:	Microcontroller and Embedded System					Sub Code:	18CS44	Branch:	ISE
Date:	26/09/2022	Duration:	180 min's	Max Marks:	100	Sem/Sec:	III / A, B and C		OBE

Answer any FIVE FULL Questions — MARKS | CO | RBT

1 a. Compare Microprocessor and Microcontroller — 6 | CO1 | L2

Solution:

Microprocessor	Microcontroller
Microprocessor is the heart of Computer system.	Micro Controller is the heart of an embedded system.
It is only a processor, so memory and I/O components need to be connected externally	Micro Controller has a processor along with internal memory and I/O components.
Memory and I/O has to be connected externally, so the circuit becomes large.	Memory and I/O are already present, and the internal circuit is small.
You can't use it in compact systems	You can use it in compact systems.
Cost of the entire system is high	Cost of the entire system is low
Due to external components, the total power consumption is high. Therefore, it is not ideal for the devices running on stored power like batteries.	As external components are low, total power consumption is less. So it can be used with devices running on stored power like batteries.
Most of the microprocessors do not have power saving features.	Most of the microcontrollers offer power-saving mode.
mainly used in personal computers.	It is used mainly in a washing machine, MP3 players, and embedded systems.
Microprocessor has a smaller number of registers, so more operations are memory-based.	Microcontroller has more register. Hence the programs are easier to write.

b. Discuss the ARM design philosophy. — 6 | CO1 | L2

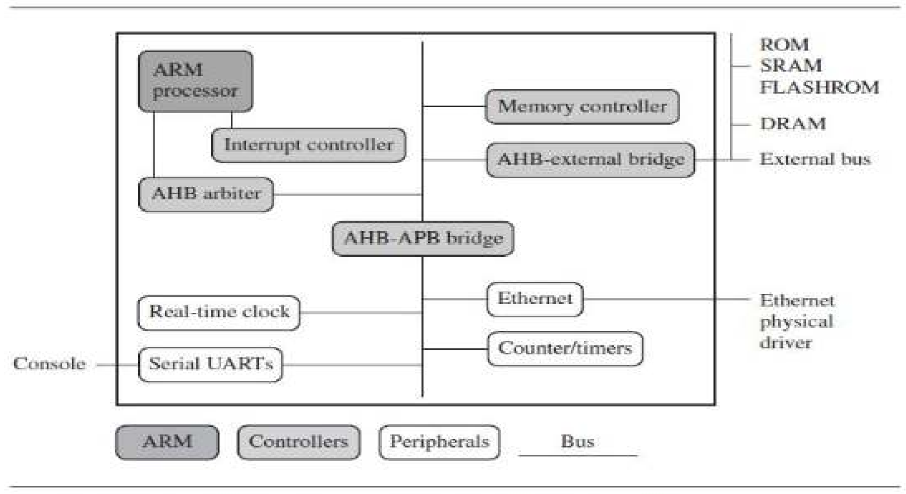
Solution:

Physical features of ARM processor design:

1. Power: The ARM processor has been specifically designed to be small to reduce power Consumption and extend battery operation essential for applications such as mobile phones and personal digital assistants (PDAs).
2. High Code Density: since embedded systems have limited memory due to cost and/or physical size restrictions.
3. Price sensitive: use slow and low-cost memory devices.
4. Reduce the area of the die: For a single-chip solution, the smaller the area used by the embedded processor, the more available space for specialized peripherals. This in turn reduces the cost.
5. Debug technology within the processor: So the software engineers can view what is happening while the processor is executing code. With greater visibility. Software engineers can resolve issues faster. Hence reduces development costs.

c. With a neat diagram, explain the four main hardware components of an ARM based embedded device. — 8 | CO1 | L2

Solution:



An example of an ARM-based embedded device, a microcontroller.

The device into four main hardware components:

– ARM processor
– Controllers
– Peripherals
– Bus

1. ARM Processor :

An ARM processor comprises a core (the execution engine that processes instructions and manipulates data) It is surrounded with components that interface it with a bus. These components can include memory management and caches.

2. Controllers:

Coordinates important functional blocks of the system.

2 Controllers:

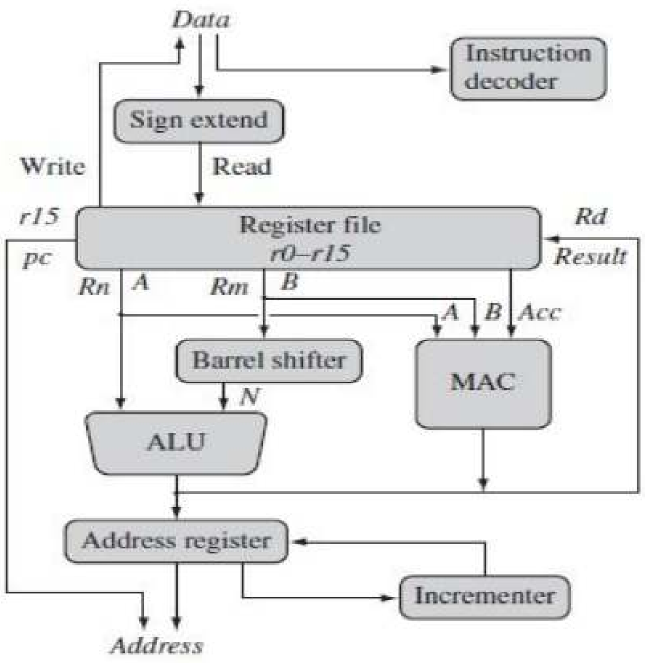
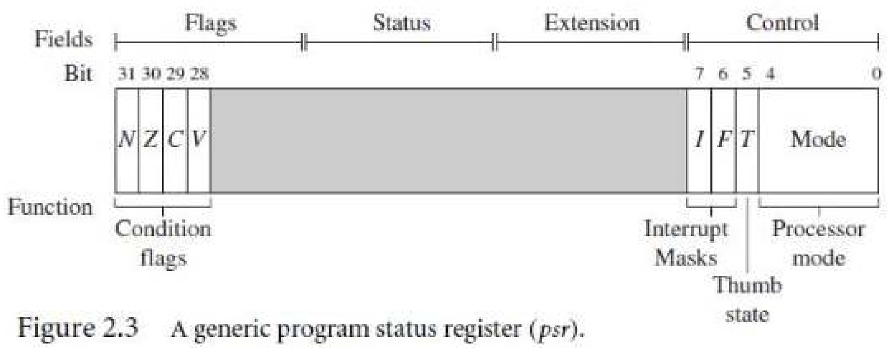
• Interrupt controllers
• memory controllers

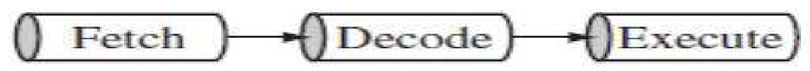
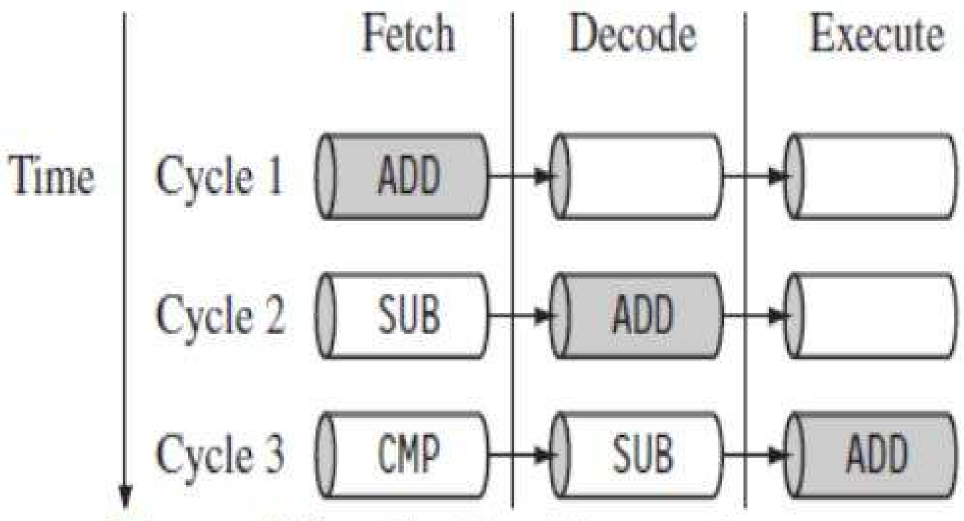
3. Peripherals:

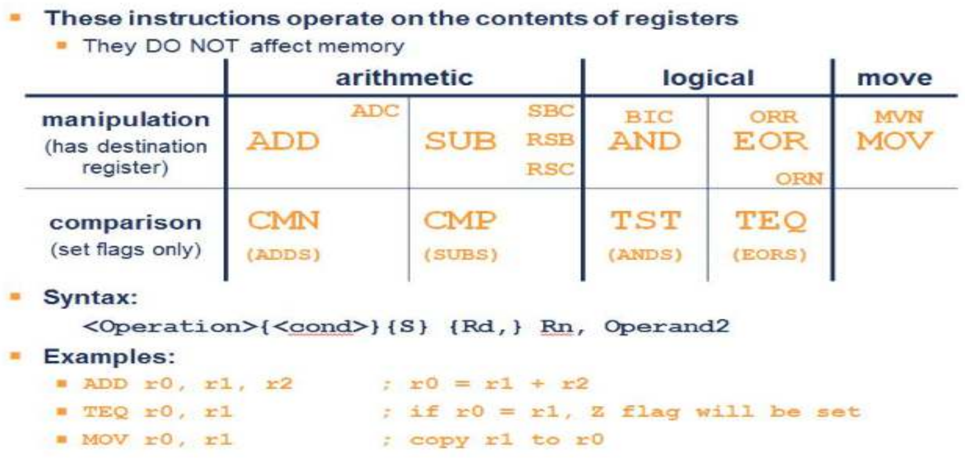
Required for input-output operation external to the chip

4. A bus :

Used to communicate between different parts of the device

2	a. Explain the ARM core data flow model with a neat diagram Solution:  Data Instruction decoder Sign extend Write Read r15 pc Register file r0–r15 Rd Result Rn A Rm B A B Acc Barrel shifter N MAC ALU Address register Incrementer Address Von-Neumann implementation – data items and instructions share the same bus. Instruction decoder translates instructions before they are executed. Load instruction: copy data from memory to register. Store instruction: copy data from register to memory . There are no data processing instructions that directly manipulate data in memory. Data items are placed in the register file – a storage bank made up of 32-bit registers. ARM instructions typically have two source registers Rn and Rm and one destination register Rd. ALU and MAC (Multiply-accumulate) unit takes the register values Rn and Rm from A and B buses and computes a result. Load and store instructions use the ALU to generate an address to be held in the address register and broadcast on the Address bus. The register Rm can be alternatively pre-processed in the barrel shifter before it enters the ALU. For load and store instructions the incrementer updates the address register before the core reads or writes the next register value from or to the next sequential memory location.	8	CO1	L2
	b. Draw the basic layout of a generic program status register and briefly explain the various fields. **Solution:** • The *cpsr* :Dedicated 32-bit register and resides in the register file  Fields Flags Status Extension Control Bit 31 30 29 28 7 6 5 4 0 N Z C V I F T Mode Function Condition flags Interrupt Masks Thumb state Processor mode Figure 2.3 A generic program status register (*psr*).	6	CO1	L2

	cpsr : Divided into four fields, each 8 bits wide: flags, status, extension, and control. The control field contains the processor mode, state, and interrupt mask bits. The flags field contains the condition flags. • Condition Codes: N, Z, C, V • Interruption mask: I(IRQ), F(FIQ) • Thumb Enable Bit • Mode(5-bit)			
	c. What is Pipelining? Illustrate it with a simple example. **Solution:** A pipeline is the mechanism a RISC processor uses to execute instructions. Using a pipeline speeds up execution by fetching the next instruction while other instructions are being decoded and executed. Figure 2.7 ARM7 Three-stage pipeline.   Fetch Decode Execute Time Cycle 1 Cycle 2 Cycle 3 ADD SUB ADD CMP SUB ADD Figure 2.8 Pipelined instruction sequence.	6	CO1	L2
3	**a** Explain the different Data Processing Instructions in ARM Solution:	10	CO2	L2



- **These instructions operate on the contents of registers**
  - They DO NOT affect memory

	arithmetic		logical		move
manipulation (has destination register)	ADD ADC	SUB SBC RSB RSC	AND BIC	EOR ORR ORN	MOV MVN
comparison (set flags only)	CMN (ADDS)	CMP (SUBS)	TST (ANDS)	TEQ (EORS)	

- **Syntax:**
  `<Operation>{<cond>}{S} {Rd,} Rn, Operand2`
- **Examples:**
  - `ADD r0, r1, r2      ; r0 = r1 + r2`
  - `TEQ r0, r1          ; if r0 = r1, Z flag will be set`
  - `MOV r0, r1          ; copy r1 to r0`

Example: The following simple subtract instruction subtracts a value stored in register r2 from a value stored in register r1. The result is stored in register r0.
PRE r0 = 0x00000000
r1 = 0x00000002
r2 = 0x00000001
SUB r0, r1, r2
POST r0 = 0x00000001

```
Syntax: <instruction>{<cond>}{S} Rd, N
```

MOV	Move a 32-bit value into a register	*Rd = N*
MVN	move the NOT of the 32-bit value into a register	*Rd = ~N*

b. Briefly explain the different Load-Store Instruction categories used with ARM	10	CO2	L2

Solution:
LOAD-STORE INSTRUCTIONS:
Load-store instructions transfer data between memory and processor registers. There are three types of load-store instructions: single-register transfer, multiple-register transfer, and swap.
Single-Register Transfer:
- These instructions are used for moving a single data item in and out of a register.
- The data types supported are signed and unsigned words (32-bit), half-words (16-bit), and bytes.

Here are the various load-store single-register transfer instructions.

LDR	load word into a register	*Rd <- mem32[address]*
STR	save byte or word from a register	*Rd -> mem32[address]*
LDRB	load byte into a register	*Rd <- mem8[address]*
STRB	save byte from a register	*Rd -> mem8[address]*

LDRH	load halfword into a register	*Rd <- mem16[address]*
STRH	save halfword into a register	*Rd -> mem16[address]*
LDRSB	load signed byte into a register	*Rd <- SignExtend (mem8[address])*
LDRSH	load signed halfword into a register	*Rd <- SignExtend (mem16[address])*

LDR and STR instructions can load and store data on a boundary alignment that is the same as the data type size being loaded or stored.

- For example, LDR can only load 32-bit words on a memory address that is a multiple of four bytes—0, 4, 8, and so on.

```
Syntax: <LDR|STR>{<cond>}{B} Rd,addressing1
        LDR{<cond>}SB|H|SH Rd, addressing2
        STR{<cond>}H Rd, addressing2
```

4

a. Write a program for forward and backward branch by considering an example — 06 — CO2 — L3

Solution:

Example: This example shows a forward and backward branch. Because these loops are address specific, we do not include the pre- and post-conditions. The forward branch skips three instructions. The backward branch creates an infinite loop.

```
        B       forward
        ADD r1, r2, #4
        ADD r0, r6, #2
        ADD r3, r7, #4
forward
        SUB r1, r2, #4
 -------------------------------------------------
backward
        ADD r1, r2, #4
        SUB r1, r2, #4
        ADD   r4, r6, r7
        B       backward
```

b. Explain Co-Processor Instructions of ARM Processor — 06 — CO2 — L2

Solution:

```
Syntax: CDP{<cond>} cp, opcode1, Cd, Cn {, opcode2}
        <MRC|MCR>{<cond>} cp, opcode1, Rd, Cn, Cm {, opcode2}
        <LDC|STC>{<cond>} cp, Cd, addressing
```

CDP	coprocessor data processing—perform an operation in a coprocessor
MRC MCR	coprocessor register transfer—move data to/from coprocessor registers
LDC STC	coprocessor memory transfer—load and store blocks of memory to/from a coprocessor

- ✓ In the syntax of the coprocessor instructions,
  - o The *cp* field represents the coprocessor number between *p0* and *p15*
  - o The *opcode* fields describe the operation to take place on the coprocessor.
  - o The *Cn*, *Cm*, and *Cd* fields describe registers within the coprocessor.
- ✓ The coprocessor operations and registers depend on the specific coprocessor you are using.
- ✓ *Coprocessor 15 (CP15)* is reserved for system control purposes, such as memory management, write buffer control, cache control, and identification registers.

Example: This example shows a *CP15* register being copied into a general-purpose register.

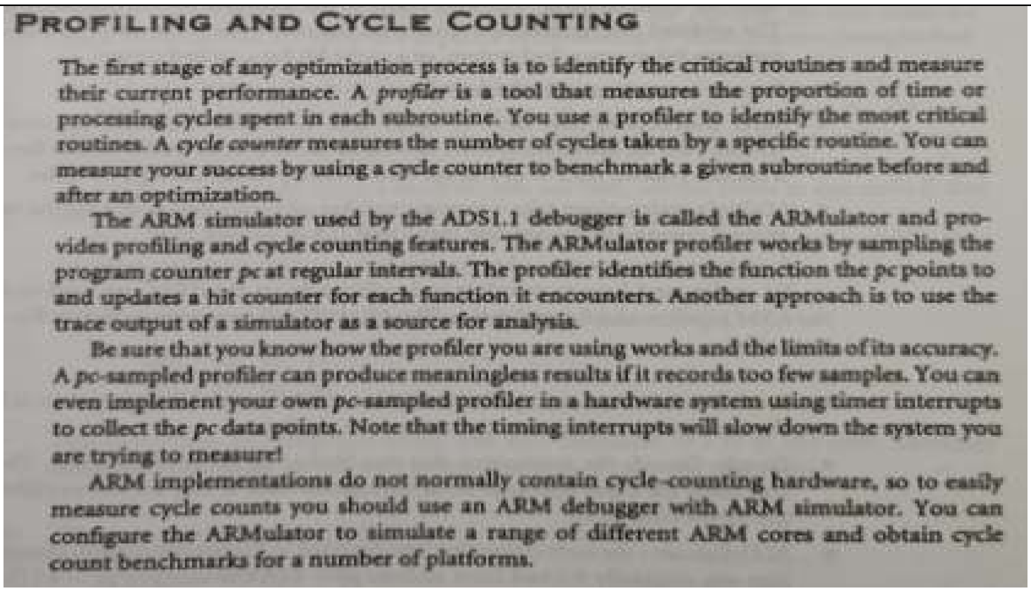
; transferring the contents of CP15 register c0 to register r10

MRC p15, 0, r10, c0, c0, 0

Here *CP15 register-0* contains the processor identification number. This register is copied into the general-purpose register *r10*.

c. Write a note on Profiling and Cycle Counting. — 08 — CO2 — L2

Solution:



PROFILING AND CYCLE COUNTING

The first stage of any optimization process is to identify the critical routines and measure their current performance. A *profiler* is a tool that measures the proportion of time or processing cycles spent in each subroutine. You use a profiler to identify the most critical routines. A *cycle counter* measures the number of cycles taken by a specific routine. You can measure your success by using a cycle counter to benchmark a given subroutine before and after an optimization.

The ARM simulator used by the ADS1.1 debugger is called the ARMulator and provides profiling and cycle counting features. The ARMulator profiler works by sampling the program counter *pc* at regular intervals. The profiler identifies the function the *pc* points to and updates a hit counter for each function it encounters. Another approach is to use the trace output of a simulator as a source for analysis.

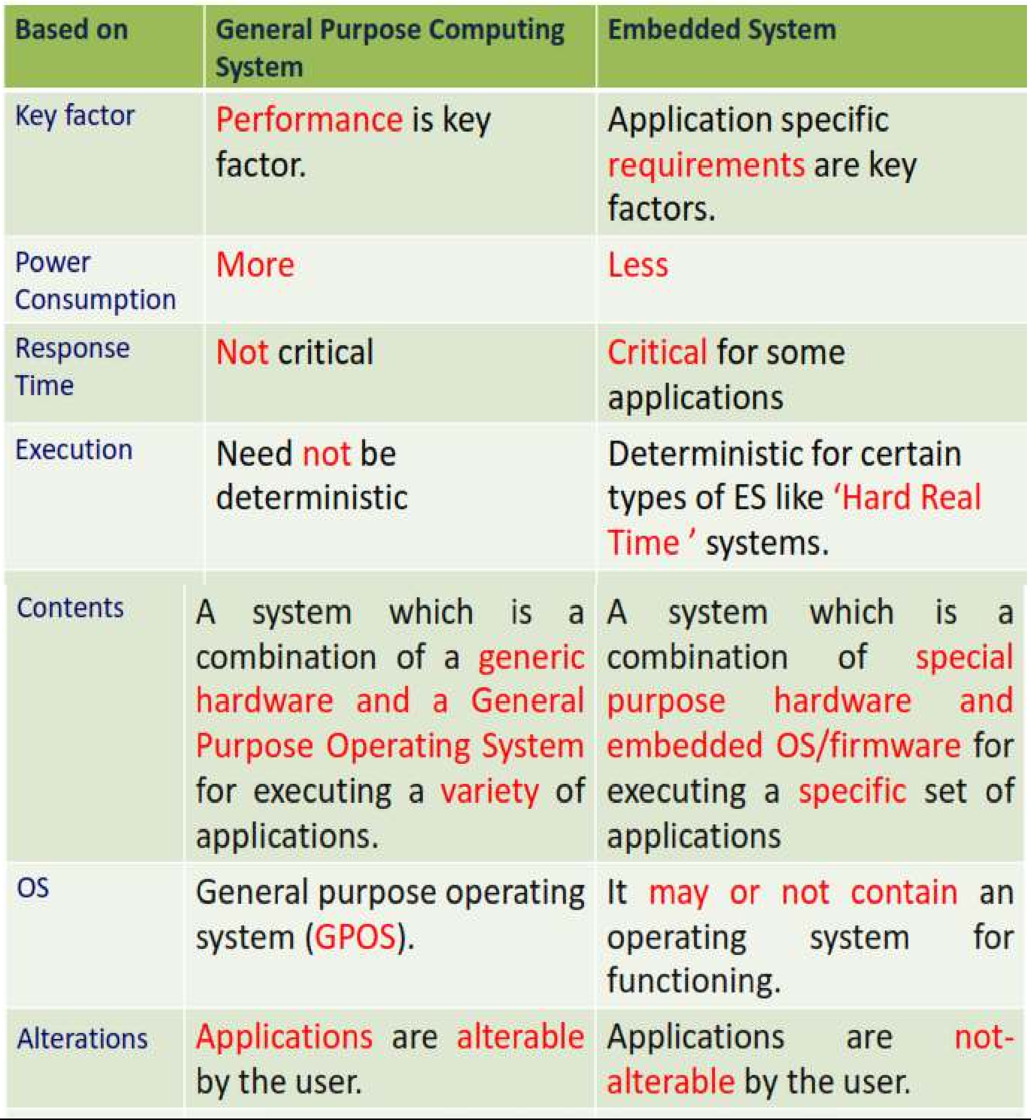
Be sure that you know how the profiler you are using works and the limits of its accuracy. A *pc*-sampled profiler can produce meaningless results if it records too few samples. You can even implement your own *pc*-sampled profiler in a hardware system using timer interrupts to collect the *pc* data points. Note that the timing interrupts will slow down the system you are trying to measure!

ARM implementations do not normally contain cycle-counting hardware, so to easily measure cycle counts you should use an ARM debugger with ARM simulator. You can configure the ARMulator to simulate a range of different ARM cores and obtain cycle count benchmarks for a number of platforms.

5 a. What is an embedded system? Different between 06 CO4 L2

Solution:

An ES is an electronic /electro-mechanical system designed to perform a specific function and a combination of both hardware and firmware (software) Every ES is Unique and hardware as well as the firmware is highly specialized to the application domain.



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

	b. List any four purposes of Embedded system with examples.	08	CO4	L2

Solution:

Each embedded system is designed to serve the purpose of any one or a combination of the following tasks:

1. Data collection/Storage/Representation
2. Data Communication
3. Data (signal) processing
4. Monitoring

1. Data collection/Storage/Representation:

ES is designed for one of the purpose:

– Data collection from external world: Data collection is done mainly for:
– Storage
– Analysis
– Manipulation
– Transmission

Examples :

– Videos, images, text, electrical signals etc.

2. Data communication:

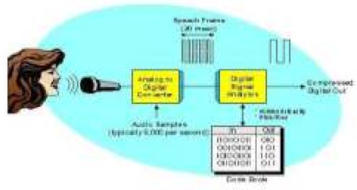
• Communication means :
– The way data is transmitted / received
– It means communication can be from:
– Complex satellite communication systems -> simple home networking systems.

• The transmission medium can be:
– Wired
– Wireless

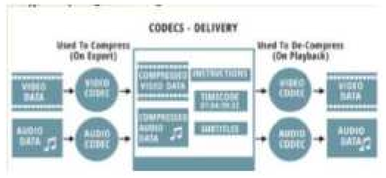
Data can be transmitted either by analog /digital

3. Data (Signal) Processing:

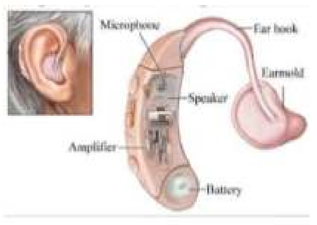
- Data of any type collected can be used for :
  - Data processing:- like
    - Speech coding


Speech Frame
Analog to Digital Converter
Digital Signal Analysis
Compressed Digital Out
Audio Samples (typically 8,000 per second)
Code Book

    - Audio video coding



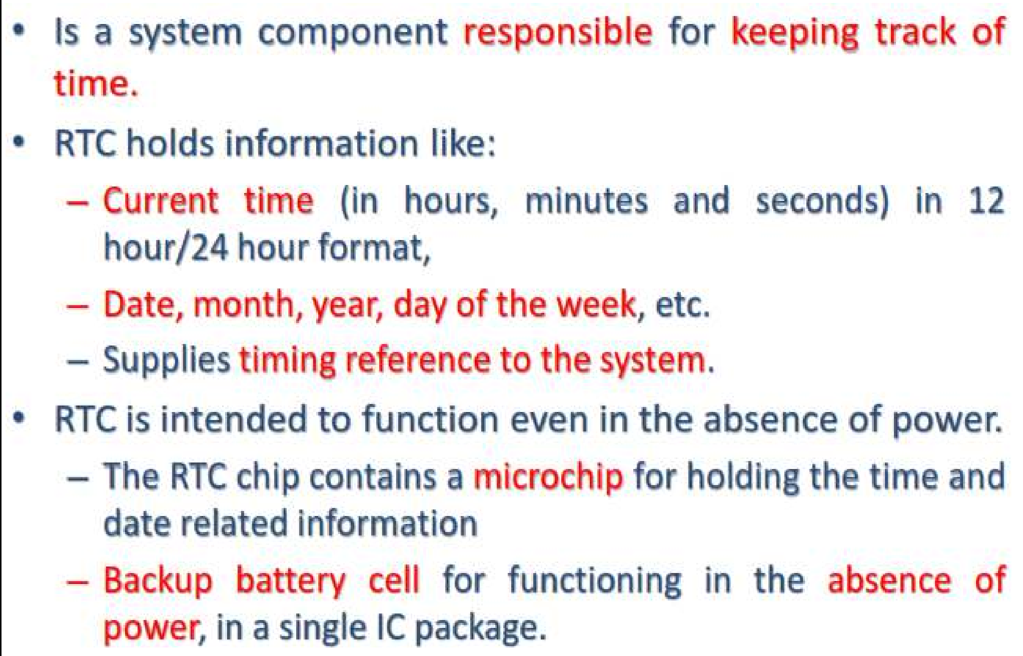
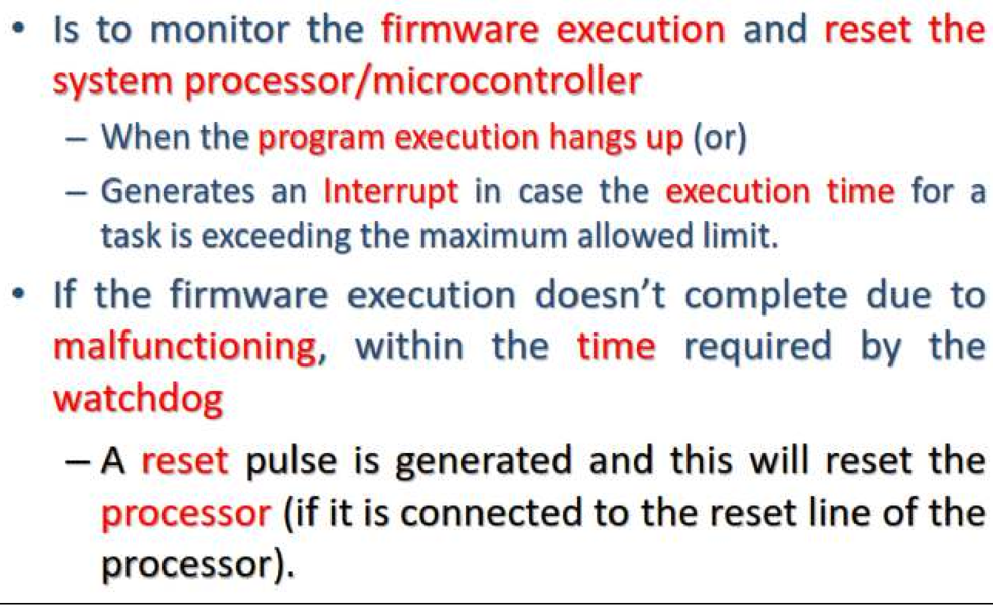
Best Example:
Digital hearing aid(improves the hearing capacity by data processing)



4. Monitoring:

• Almost all embedded products coming under the medical domain are with monitoring functions only.

• Electro cardiogram machine (ECG) is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.

	• Other examples with monitoring function are digital CRO, digital multimeters, and logic analyzers. c. Write short notes on i) Real Time Clock ii) Watch dog timer **Solution:** **Real Time Clock**  • Is a system component responsible for keeping track of time. • RTC holds information like: – Current time (in hours, minutes and seconds) in 12 hour/24 hour format, – Date, month, year, day of the week, etc. – Supplies timing reference to the system. • RTC is intended to function even in the absence of power. – The RTC chip contains a microchip for holding the time and date related information – Backup battery cell for functioning in the absence of power, in a single IC package. **Watch Dog Timer:**  • Is to monitor the firmware execution and reset the system processor/microcontroller – When the program execution hangs up (or) – Generates an Interrupt in case the execution time for a task is exceeding the maximum allowed limit. • If the firmware execution doesn't complete due to malfunctioning, within the time required by the watchdog – A reset pulse is generated and this will reset the processor (if it is connected to the reset line of the processor).	06	CO3	
6	a. Briefly describe the classification of Embedded system **Solution:**	8	CO4	L2

CLASSIFICATION OF EMBEDDED SYSTEMS

It is possible to have a multitude of classifications for embedded systems, based on different criteria. Some of the criteria used in the classification of embedded systems are as follows:

(1) Based on generation
(2) Complexity and performance requirements
(3) Based on deterministic behaviour
(4) Based on triggering.

The classification based on deterministic system behaviour is applicable for 'Real Time' systems. The application/task execution behaviour for an embedded system can be either deterministic or non- deterministic. Based on the execution behaviour, Real Time embedded systems are classified into *Hard* and *Soft*. We will discuss about hard and soft real time systems in a later chapter. Embedded Systems which are 'Reactive' in nature (Like process control systems in industrial control applications) can be classified based on the trigger. Reactive systems can be either *event triggered* or *time triggered.*

1.4.1 Classification Based on Generation

This classification is based on the order in which the embedded processing systems evolved from the first version to where they are today. As per this criterion, embedded systems can be classified into the following:

1.4.1.1 First Generation The early embedded systems were built around 8bit microprocessors like 8085 and Z80, and 4bit microcontrollers. Simple in hardware circuits with firmware developed in Assembly code. Digital telephone keypads, stepper motor control units etc. are examples of this.

1.4.1.2 Second Generation These are embedded systems built around 16bit microprocessors and 8 or 16 bit microcontrollers, following the first generation embedded systems. The instruction set for the second generation processors/controllers were much more complex and powerful than the first generation processors/controllers. Some of the second generation embedded systems contained embedded operating systems for their operation. Data Acquisition Systems, SCADA systems, etc. are examples of second generation embedded systems.

1.4.1.3 Third Generation With advances in processor technology, embedded system developers started making use of powerful 32bit processors and 16bit microcontrollers for their design. A new concept of application and domain specific processors/controllers like Digital Signal Processors (DSP) and Application Specific Integrated Circuits (ASICs) came into the picture. The instruction set of processors became more complex and powerful and the concept of instruction pipelining also evolved. The processor market was flooded with different types of processors from different vendors. Processors like Intel Pentium, Motorola 68K, etc. gained attention in high performance embedded requirements. Dedicated embedded real time and general purpose operating systems entered into the embedded market. Embedded systems spread its ground to areas like robotics, media, industrial process control, networking, etc.

1.4.1.4 Fourth Generation The advent of System on Chips (SoC), reconfigurable processors and multicore processors are bringing high performance, tight integration and miniaturisation into the embedded device market. The SoC technique implements a total system on a chip by integrating different functionalities with a processor core on an integrated circuit. We will discuss about SoCs in a later chapter. The fourth generation embedded systems are making use of high performance real time embedded operating systems for their functioning. Smart phone devices, mobile internet devices (MIDs), etc. are examples of fourth generation embedded systems.

1.4.1.5 What Next? The processor and embedded market is highly dynamic and demanding. So 'what will be the next smart move in the next embedded generation?' Let's wait and see.

1.4.2 Classification Based on Complexity and Performance

This classification is based on the complexity and system performance requirements. According to this classification, embedded systems can be grouped into the following:

1.4.2.1 Small-Scale Embedded Systems Embedded systems which are simple in application needs and where the performance requirements are not time critical fall under this category. An electronic toy is a typical example of a small-scale embedded system. Small-scale embedded systems are usually built around low performance and low cost 8 or 16 bit microprocessors/microcontrollers. A small-scale embedded system may or may not contain an operating system for its functioning.

1.4.2.2 Medium-Scale Embedded Systems Embedded systems which are slightly complex in hardware and firmware (software) requirements fall under this category. Medium-scale embedded systems are usually built around medium performance, low cost 16 or 32 bit microprocessors/microcontrollers or digital signal processors. They usually contain an embedded operating system (either general purpose or real time operating system) for functioning.

1.4.2.3 Large-Scale Embedded Systems/Complex Systems Embedded systems which involve highly complex hardware and firmware requirements fall under this category. They are employed in mission critical applications demanding high performance. Such systems are commonly built around high performance 32 or 64 bit RISC processors/controllers or Reconfigurable System on Chip (RSoC) or multi-core processors and programmable logic devices. They may contain multiple processors/controllers and co-units/hardware accelerators for offloading the processing requirements from the main processor of the system. Decoding/ encoding of media, cryptographic function implementation, etc. are examples for processing requirements which can be implemented using a co-processor/hardware accelerator. Complex embedded systems usually contain a high performance Real Time Operating System (RTOS) for task scheduling, prioritisation, and management.

	b. Explain the following: i) I2C Bus ii) SPI Bus iii) Reset Circuit iv) 1-Wire Interface **Solution:**	12	CO4	L2

Inter Integrated Circuit (I2C) Bus The Inter Integrated Circuit Bus (I2C–Pronounced 'I square C') is a synchronous bi-directional half duplex (one-directional communication at a given point of time) two wire serial interface bus. The concept of I2C bus was developed by 'Philips semiconductors' in the early 1980s. The original intention of I2C was to provide an easy way of connection between a microprocessor/microcontroller system and the peripheral chips in television sets. The I2C bus comprise of two bus lines, namely; Serial Clock–SCL and Serial Data–SDA. SCL line is responsible for generating synchronisation clock pulses and SDA is responsible for transmitting the serial data across devices. I2C bus is a shared bus system to which many number of I2C devices can be connected. Devices connected to the I2C bus can act as either 'Master' device or 'Slave' device. The 'Master' device is responsible for controlling the communication by initiating/terminating data transfer, sending data and generating necessary synchronisation clock pulses. 'Slave' devices wait for the commands from the master and respond upon receiving the commands. 'Master' and 'Slave' devices can act as either transmitter or receiver. Regardless whether a master is acting as transmitter or receiver, the synchronisation clock signal is generated by the 'Master' device only. I2C supports multi masters on the same bus. The following bus interface diagram shown in Fig. 2.26 illustrates the connection of master and slave devices on the I2C bus.

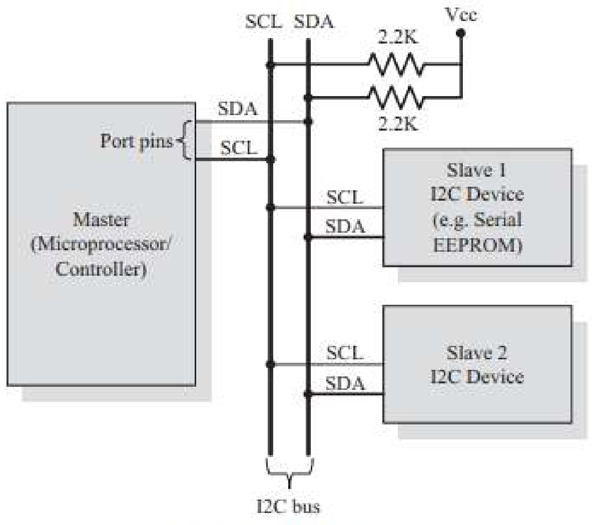

SCL SDA
Vcc
2.2K
2.2K
SDA
Port pins
SCL
Master (Microprocessor/ Controller)
SCL
SDA
Slave 1 I2C Device (e.g. Serial EEPROM)
SCL
SDA
Slave 2 I2C Device
I2C bus

Fig. 2.26 I2C Bus Interfacing

The I2C bus interface is built around an input buffer and an open drain or collector transistor. When the bus is in the idle state, the open drain/collector transistor will be in the floating state and the output lines (SDA and SCL) switch to the 'High Impedance' state. For proper operation of the bus, both the bus lines should be pulled to the supply voltage (+5V for TTL family and +3.3V for CMOS family devices) using pull-up resistors. The typical value of resistors used in pull-up is 2.2K. With pull-up resistors, the output lines of the bus in the idle state will be 'HIGH'.

The address of a I2C device is assigned by hardwiring the address lines of the device to the desired logic level. The address to various I2C devices in an embedded device is assigned and hardwired at the time of designing the embedded hardware. The sequence of operations for communicating with an I2C slave device is listed below:

1. The master device pulls the clock line (SCL) of the bus to 'HIGH'
2. The master device pulls the data line (SDA) 'LOW', when the SCL line is at logic 'HIGH' (This is the 'Start' condition for data transfer)
3. The master device sends the address (7 bit or 10 bit wide) of the 'slave' device to which it wants to communicate, over the SDA line. Clock pulses are generated at the SCL line for synchronising the bit reception by the slave device. The MSB of the data is always transmitted first. The data in the bus is valid during the 'HIGH' period of the clock signal
4. The master device sends the Read or Write bit (Bit value = 1 Read operation; Bit value = 0 Write operation) according to the requirement
5. The master device waits for the acknowledgement bit from the slave device whose address is sent on the bus along with the Read/Write operation command. Slave devices connected to the bus compares the address received with the address assigned to them

6. The slave device with the address requested by the master device responds by sending an acknowledge bit (Bit value = 1) over the SDA line
7. Upon receiving the acknowledge bit, the Master device sends the 8bit data to the slave device over SDA line, if the requested operation is 'Write to device'. If the requested operation is 'Read from device', the slave device sends data to the master over the SDA line
8. The master device waits for the acknowledgement bit from the device upon byte transfer complete for a write operation and sends an acknowledge bit to the Slave device for a read operation
9. The master device terminates the transfer by pulling the SDA line 'HIGH' when the clock line SCL is at logic 'HIGH' (Indicating the 'STOP' condition)

The first generation I2C devices were designed to support data rates only up to 100kbps. Over time there have been several additions to the specification so that there are now five operating speed categories; Namely, Standard mode (Sm - Data rate up to 100kbit/sec), Fast mode (Fm - Data rate up to 400kbit/sec), Fast mode Plus (Fm+ - Data rate up to 1Mbit/sec), and High-speed mode (Hs-mode - Data rate up to 3.4Mbit/sec) and an Ultra Fast-mode (UFm), with a bit rate up to 5 Mbit/s for unidirectional I2C bus.

2.4.1.2 Serial Peripheral Interface (SPI) Bus The Serial Peripheral Interface Bus (SPI) is a synchronous bi-directional full duplex four-wire serial interface bus. The concept of SPI was introduced by Motorola. SPI is a single master multi-slave system. It is possible to have a system where more than one SPI device can be master, provided the condition only one master device is active at any given point of time, is satisfied. SPI requires four signal lines for communication. They are:

Master Out Slave In (MOSI):	Signal line carrying the data from master to slave device. It is also known as Slave Input/Slave Data In (SI/SDI)
Master In Slave Out (MISO):	Signal line carrying the data from slave to master device. It is also known as Slave Output (SO/SDO)
Serial Clock (SCLK):	Signal line carrying the clock signals
Slave Select (SS):	Signal line for slave device select. It is an active low signal

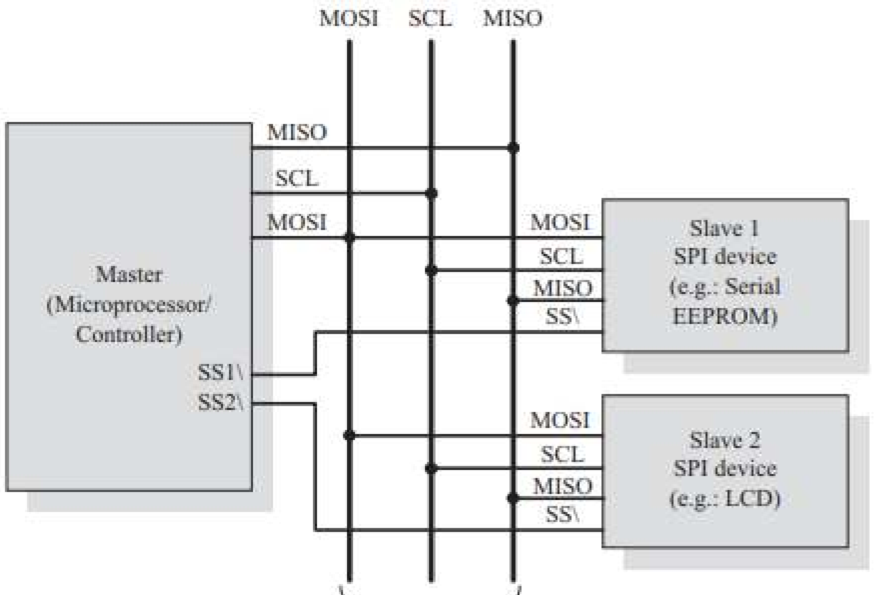
The bus interface diagram shown in Fig. 2.27 illustrates the connection of master and slave devices on the SPI bus.

The master device is responsible for generating the clock signal. It selects the required slave device by asserting the corresponding slave device's slave select signal 'LOW'. The data out line (MISO) of all the slave devices when not selected floats at high impedance state.

The serial data transmission through SPI bus is fully configurable. SPI devices contain a certain set of registers for holding these configurations. The serial peripheral control register holds the various configuration parameters like master/slave selection for the device, baudrate selection for communication, clock signal control, etc. The status register holds the status of various conditions for transmission and reception.

SPI works on the principle of 'Shift Register'. The master and slave devices contain a special shift register for the data to transmit or receive. The size of the shift register is device dependent. Normally it is a multiple of 8. During transmission from the master to slave, the data in the master's shift register is shifted out to the MOSI pin and it enters the shift register of the slave device through the MOSI pin of the slave device. At the same time the shifted out data bit from the slave device's shift register enters the shift register of the master device through MISO pin. In summary, the shift registers of 'master' and 'slave' devices form a circular buffer. For some devices, the decision on whether the LS/MS bit of data needs to be sent out first is configurable through configuration register (e.g. LSBF bit of the SPI control register for Motorola's 68HC12 controller).

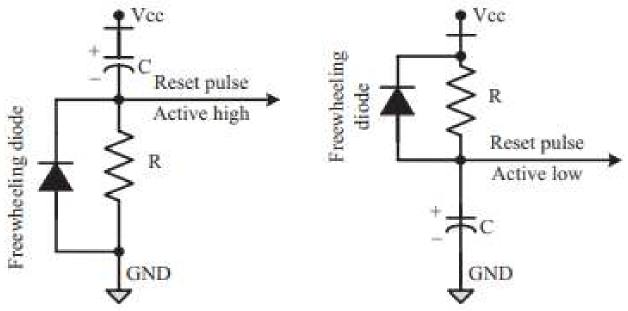
When compared to I2C, SPI bus is most suitable for applications requiring transfer of data in 'streams'.


MOSI
SCL
MISO
MISO
SCL
MOSI
Master (Microprocessor/ Controller)
SS1\
SS2\
MOSI
SCL
MISO
SS\
Slave 1 SPI device (e.g.: Serial EEPROM)
MOSI
SCL
MISO
SS\
Slave 2 SPI device (e.g.: LCD)

SPI bus

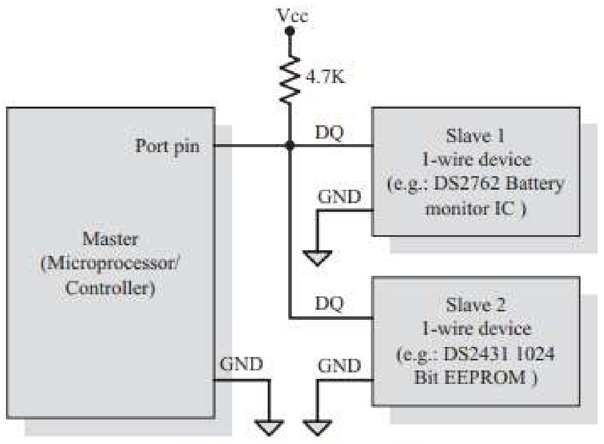
SPI bus interfacing

Reset Circuit



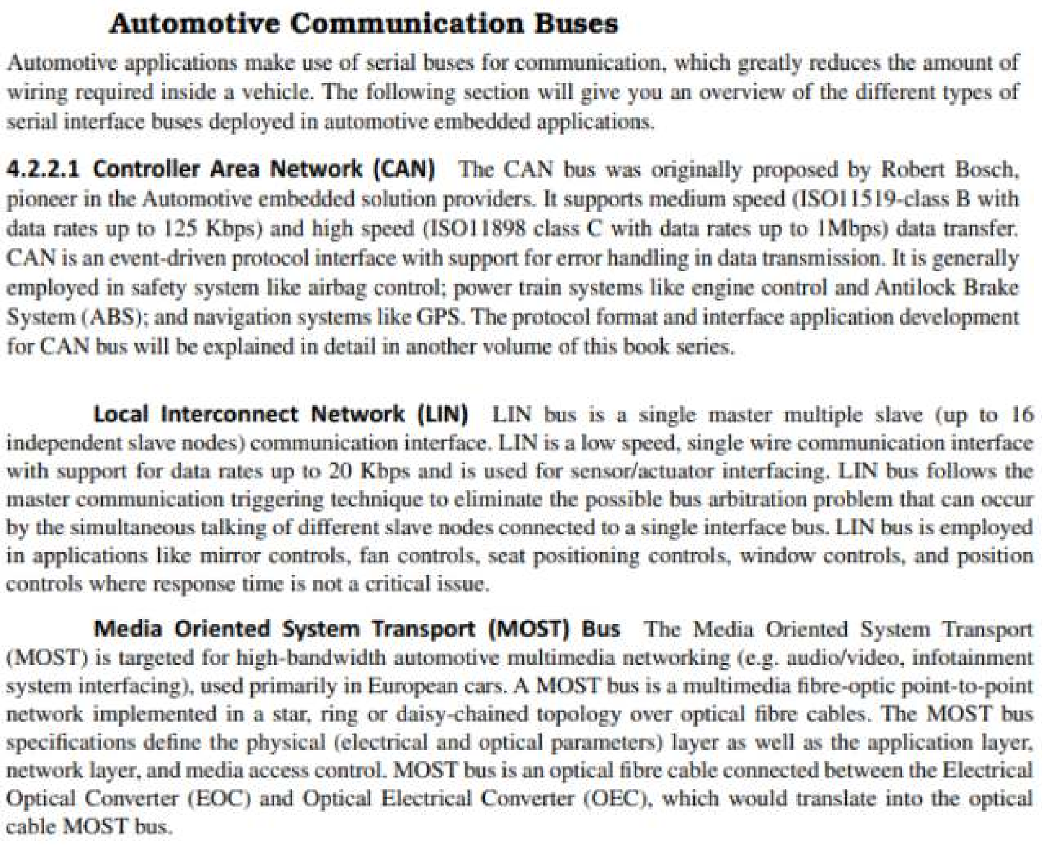
RC based reset circuit

The reset circuit is essential to ensure that the device is not operating at a voltage level where the device is not guaranteed to operate, during system power ON. The reset signal brings the internal registers and the different hardware systems of the processor/ controller to a known state and starts the firmware execution from the reset vector (Normally from vector address 0x0000 for conventional processors/controllers. The reset vector can be relocated to an address for processors/controllers supporting bootloader). The reset signal can be either active high (The processor undergoes reset when the reset pin of the processor is at logic high) or active low (The processor undergoes reset when the reset pin of the processor is at logic low). Since the processor operation is synchronised to a clock signal, the reset pulse should be wide enough to give time for the clock oscillator to stabilise before the internal reset state starts. The reset signal to the processor can be applied at power ON through an external passive reset circuit comprising a Capacitor and Resistor or through a standard Reset IC like MAX810 from Maxim Dallas (www.maxim-ic.com). Select the reset IC based on the type of reset signal and logic level (CMOS/TTL) supported by the processor/controller in use. Some microprocessors/controllers contain built-in internal reset circuitry and they don't require external reset circuitry. Figure 2.35 illustrates a resistor capacitor based passive reset circuit for active high and low configurations. The reset pulse width can be adjusted by changing the resistance value *R* and capacitance value *C*.

	2.4.1.4 1-Wire Interface 1-wire interface is an asynchronous half-duplex communication protocol developed by Maxim Dallas Semiconductor (http://www.maxim-ic.com). It is also known as **Dallas 1-Wire® protocol**. It makes use of only a single signal line (wire) called DQ for communication and follows the master-slave communication model. One of the key feature of 1-wire bus is that it allows power to be sent along the signal wire as well. The 1-Wire slave devices incorporate internal capacitor (typically of the order of 800 pF) to power the device from the signal line. The 1-wire interface supports a single master and one or more slave devices on the bus. The bus interface diagram shown in Fig. 2.29 illustrates the connection of master and slave devices on the 1-wire bus.  Vcc 4.7K Port pin DQ Slave 1 1-wire device (e.g.: DS2762 Battery monitor IC) GND Master (Microprocessor/ Controller) DQ Slave 2 1-wire device (e.g.: DS2431 1024 Bit EEPROM) GND GND **Fig. 2.29 1-Wire Interface bus**			
7	a. What are the operational and non operational quality attributes of a embedded system **Solution:** **Operational Quality Attributes** The operational quality attributes represent the relevant quality attributes related to the Embedded System when it is in the operational mode or 'online' mode. The important quality attributes coming under this category are listed below: (1) Response (2) Throughput (3) Reliability (4) Maintainability (5) Security (6) Safety **Non-Operational Quality Attributes** The quality attributes that needs to be addressed for the product 'not' on the basis of operational aspects are grouped under this category. The important quality attributes coming under this category are listed below. (1) Testability & Debug-ability (2) Evolvability (3) Portability (4) Time to prototype and market (5) Per unit and total cost.	10	CO4	L2

b. Explain the different communication buses used in automotive applications. 6 CO4 L2

Solution:



Automotive Communication Buses

Automotive applications make use of serial buses for communication, which greatly reduces the amount of wiring required inside a vehicle. The following section will give you an overview of the different types of serial interface buses deployed in automotive embedded applications.

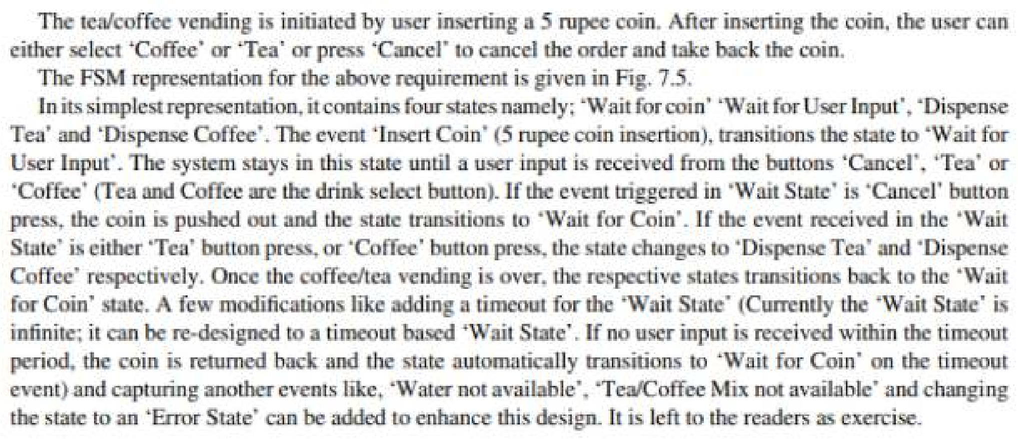
4.2.2.1 Controller Area Network (CAN) The CAN bus was originally proposed by Robert Bosch, pioneer in the Automotive embedded solution providers. It supports medium speed (ISO11519-class B with data rates up to 125 Kbps) and high speed (ISO11898 class C with data rates up to 1Mbps) data transfer. CAN is an event-driven protocol interface with support for error handling in data transmission. It is generally employed in safety system like airbag control; power train systems like engine control and Antilock Brake System (ABS); and navigation systems like GPS. The protocol format and interface application development for CAN bus will be explained in detail in another volume of this book series.

Local Interconnect Network (LIN) LIN bus is a single master multiple slave (up to 16 independent slave nodes) communication interface. LIN is a low speed, single wire communication interface with support for data rates up to 20 Kbps and is used for sensor/actuator interfacing. LIN bus follows the master communication triggering technique to eliminate the possible bus arbitration problem that can occur by the simultaneous talking of different slave nodes connected to a single interface bus. LIN bus is employed in applications like mirror controls, fan controls, seat positioning controls, window controls, and position controls where response time is not a critical issue.

Media Oriented System Transport (MOST) Bus The Media Oriented System Transport (MOST) is targeted for high-bandwidth automotive multimedia networking (e.g. audio/video, infotainment system interfacing), used primarily in European cars. A MOST bus is a multimedia fibre-optic point-to-point network implemented in a star, ring or daisy-chained topology over optical fibre cables. The MOST bus specifications define the physical (electrical and optical parameters) layer as well as the application layer, network layer, and media access control. MOST bus is an optical fibre cable connected between the Electrical Optical Converter (EOC) and Optical Electrical Converter (OEC), which would translate into the optical cable MOST bus.

c. Design an FSM model for Tea/coffee vending machine 4 CO4 L2

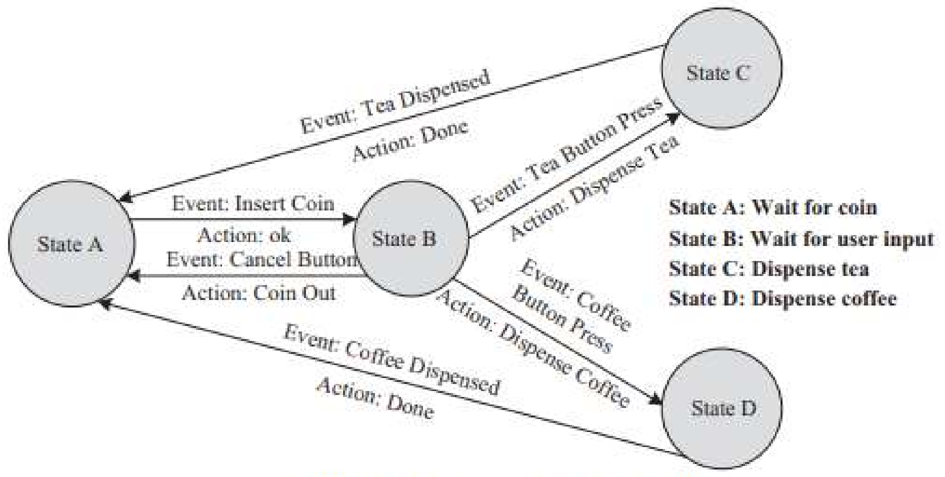
Solution:



The tea/coffee vending is initiated by user inserting a 5 rupee coin. After inserting the coin, the user can either select 'Coffee' or 'Tea' or press 'Cancel' to cancel the order and take back the coin.

The FSM representation for the above requirement is given in Fig. 7.5.

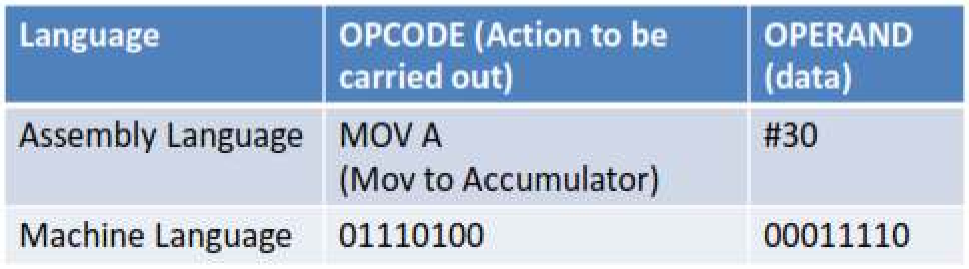
In its simplest representation, it contains four states namely; 'Wait for coin' 'Wait for User Input', 'Dispense Tea' and 'Dispense Coffee'. The event 'Insert Coin' (5 rupee coin insertion), transitions the state to 'Wait for User Input'. The system stays in this state until a user input is received from the buttons 'Cancel', 'Tea' or 'Coffee' (Tea and Coffee are the drink select button). If the event triggered in 'Wait State' is 'Cancel' button press, the coin is pushed out and the state transitions to 'Wait for Coin'. If the event received in the 'Wait State' is either 'Tea' button press, or 'Coffee' button press, the state changes to 'Dispense Tea' and 'Dispense Coffee' respectively. Once the coffee/tea vending is over, the respective states transitions back to the 'Wait for Coin' state. A few modifications like adding a timeout for the 'Wait State' (Currently the 'Wait State' is infinite; it can be re-designed to a timeout based 'Wait State'. If no user input is received within the timeout period, the coin is returned back and the state automatically transitions to 'Wait for Coin' on the timeout event) and capturing another events like, 'Water not available', 'Tea/Coffee Mix not available' and changing the state to an 'Error State' can be added to enhance this design. It is left to the readers as exercise.



FSM Model for Automatic Tea\Coffee Vending Machine

8	a. Explain the fundamental issues in hardware and software co design. Fundamental Issues of H/w, S/w Co-Design: Few are listed below: 1. Selecting the model 2. Selecting the Architecture 3. Selecting the Language 4. Partitioning system requirements into hardware and software 1. Selecting the model: • Models are used for capturing and describing the system characteristics • A model is a formal system consisting of objects and composition rules • It is hard to make a decision on which model should be followed in a particular system design. • Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design. Have objectives vary with each phase. 2. Selecting the Architecture: • A model only captures the system characteristics – Does not provide information on 'how the system can be manufactured • The architecture specifies: – how a system is going to implement in terms of the number and types of different components and the interconnection among them. 3. Selecting the Language: • A programming Language captures: – 'Computational Model' and maps it into architecture • A computational model can be captured: – Multiple programming languages like C, C++, C#, Java etc. for software implementations – Languages like VHDL, System C, Verilog etc. for hardware implementations • The only pre-requisite in selecting a programming language for capturing a model is that the language should capture the model easily. 4. Partitioning system requirements into hardware and software: • Deals with the implementation aspect of a System level Requirement • It may be possible to implement the system requirements in either hardware or software (firmware) • Various hardware software trade-offs like performance, re- usability, effort etc. are used for making a decision on the hardware-software partitioning	6	CO5	L2
	b. Explain the assembly language based embedded firmware development with a diagram • 'Assembly Language' is the human readable notation with Mnemonics • 'Machine language' is a processor understandable language with 1s and 0s • Assembly language and machine languages are: – Processor/controller dependent – I.e. Program written for one processor will not work with others • Assembly language programming:	6	CO5	L2

– Is the process of writing processor specific code in mnemonic form
– Converting the mnemonics into actual processor instructions machine language
– By using an assembler
• The general format of an assembly language instruction :
– Opcode followed by Operands
– In 8051 Processor:
• MOV A, #30



Language	OPCODE (Action to be carried out)	OPERAND (data)
Assembly Language	MOV A (Mov to Accumulator)	#30
Machine Language	01110100	00011110

• An opcode can have no-operand / single operand / dual operand / more

	c. With a neat diagram, how source file to object file translation takes place in high level language based firware development.	6	CO5	L2

• High Level Language : Like C,C++
• A software utility called 'cross-compiler' :
– Converts the high level language to target processor specific machine code
• The cross-compilation of each module generates a corresponding object file.
• The software program called linker/locater is responsible for assigning absolute address to object files during the linking process
• The Absolute object file created from the object files corresponding to different source code modules contain information about the address where each instruction needs to be placed in code memory
• A software utility called 'Object to Hex file converter' translates the absolute object file to corresponding hex file (binary file)

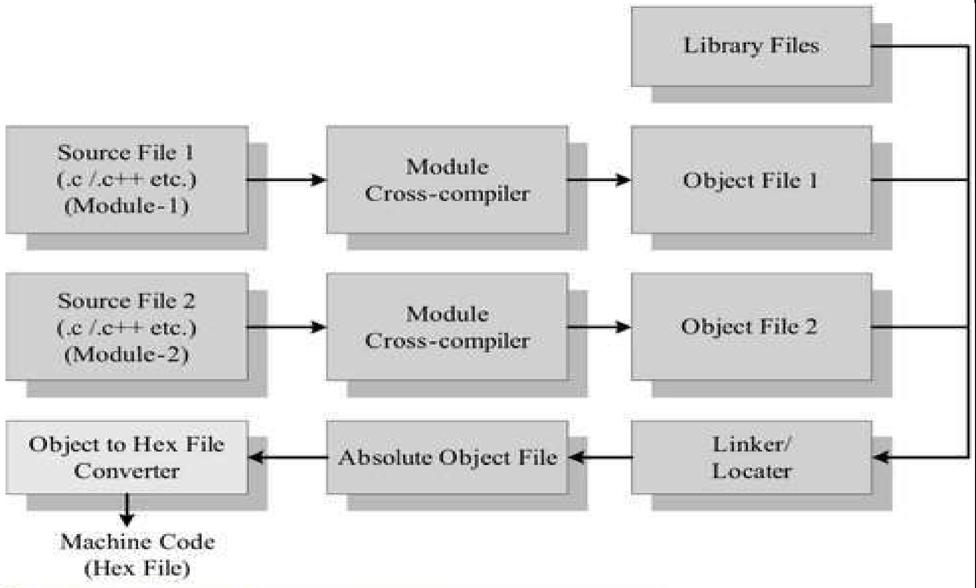
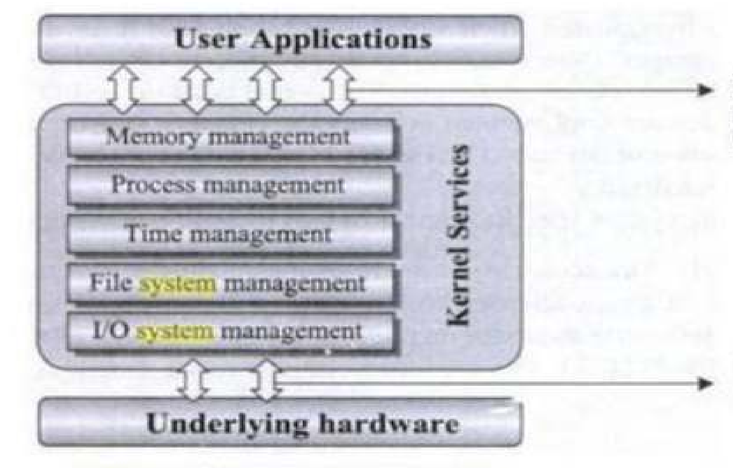


Fig. 9.2 High level language to machine language conversion process

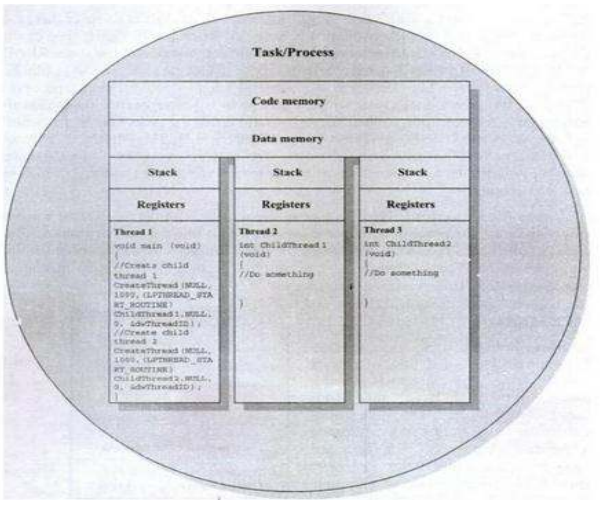
9	a. With a neat diagram, explain operating system architecture	8	CO6	L2

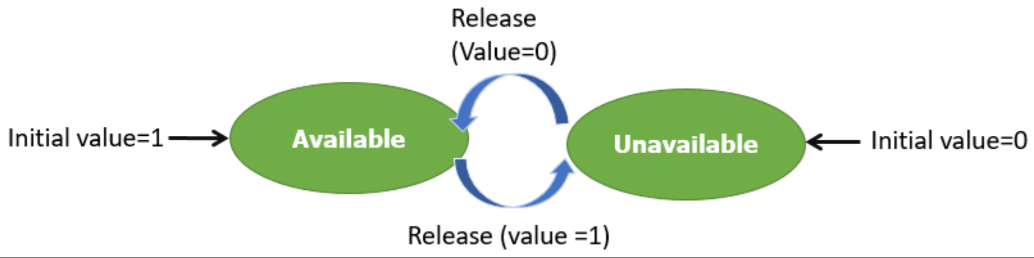
Solutions:



The Operating System Architecture

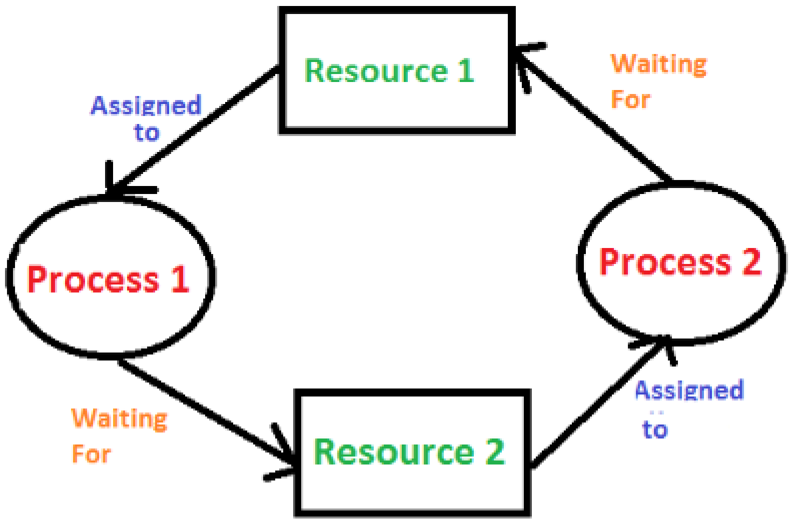
The kernel is the core of the operating system
– It is responsible for managing the system resources and the communication among the hardware and other system services
– Kernel acts as the abstraction layer between system resources and user applications
– Kernel contains a set of system libraries and services.
• For a general purpose OS, the kernel contains different services like
– Process Management
– Primary Memory Management
– File System management
– I/O System (Device) Management
– Secondary Storage Management
– Protection
– Time management
– Interrupt Handling
The program code corresponding to the kernel applications/services are kept:
– In a contiguous area □ i.e. (OS dependent) □ In primary (working) memory (and)
– Is protected from the un-authorized access by user programs/applications
• The memory space at which the kernel code is located is known as ‘Kernel Space’
• All user applications are loaded to a specific area of primary memory and this memory area is referred as ‘User Space’
• The partitioning of memory into kernel and user space is purely Operating System dependent.

	b. Explain Multithreading
	Solutions:
	Multithreading advantages:
	• Better memory utilization.
	• Since the process is split into different threads, when one thread enters a wait state, the CPU can be utilized by other.
	• Hence Efficient CPU utilization. The CPU is engaged all time.
	•Speeds up the execution of the process.
	
	 Task/Process
	Code memory
	Data memory
	Stack
	Stack
	Stack
	Registers
	Registers
	Registers
	Thread 1
	Thread 2
	Thread 3
	`void main (void)`
	`{`
	`//Create child thread 1`
	`CreateThread(NULL, 1000, (LPTHREAD_START_ROUTINE) ChildThread1,NULL, 0, &dwThreadID);`
	`//Create child thread 2`
	`CreateThread(NULL, 1000, (LPTHREAD_START_ROUTINE) ChildThread2,NULL, 0, &dwThreadID);`
	`}`
	`int ChildThread1 (void)`
	`{`
	`//Do something`
	`}`
	`int ChildThread2 (void)`
	`{`
	`//Do something`
	`}`
		06	CO6	L2

	c. Explain the concept of Binary Semaphore Solution: The value of a semaphore variable in binary semaphores is either 0 or 1. The value of the semaphore variable is initially set to 1, but if a process requests a resource, the wait() method is invoked, and the value of this semaphore is changed from 1 to 0. When the process has finished using the resource, the signal() method is invoked, and the value of this semaphore variable is raised to 1. If the value of this semaphore variable is 0 at a given point in time, and another process wants to access the same resource, it must wait for the prior process to release the resource. Process synchronization can be performed in this manner. The binary semaphores are quite similar to counting semaphores, but their value is restricted to 0 and 1. In this type of semaphore, the wait operation works only if semaphore = 1, and the signal operation succeeds when semaphore= 0. It is easy to implement than counting semaphores.  Release (Value=0) Initial value=1 Available Unavailable Initial value=0 Release (value =1)	6	CO6	L2

10	a. Explain the role of Integrated Development Environment (IDE) for embedded software development. **Solutions:** • The Integrated Development Environment (IDE) itself will be providing simulator support and they help in debugging the firmware for checking its required functionality. • In certain scenarios, simulator refers to a soft model (GUI model) of the embedded product. For example, if the product under development is a handheld device, to test the functionalities of the various menu and user interfaces, a soft form model of the product with all UI as given in the end product can be developed in software. • Soft phone is an example for such a simulator. • Emulator is hardware device which emulates the functionalities of the target device and allows real time debugging of the embedded firmware in a hardware environment. In embedded system development context, Integrated Development Environment (IDE) stands for an integrated environment for developing and debugging the target processor specific embedded firmware. IDE is a software package which bundles – □ a “Text Editor (Source Code Editor)”, □ “Cross-complier (for cross platform development and complier for same platform development)”, □ “Linker”, and □ a “Debugger”. Some IDEs may provide – □ interface to target board emulators, □ target processor‟s/ controller‟s Flash memory programmer, etc. IDE may be command line based or GUI based. NOTE: The Keil µVision IDE & An Overview of IDEs – lest as an exercise/ self study topic.	8	CO6	L2

	b. Write a note on Message passing **Solutions:** Message Passing is a synchronous / asynchronous information exchange mechanism for Inter Process/ thread Communication • Through shared memory lot of data can be shared: – Whereas only limited amount of information / data is passed through message passing • Message passing is relatively fast and free from the synchronization overheads compared to shared memory. • Three ways it can be done: – Message Queue – Mailbox – Signalling Process which wants to talk to another process posts the message to a First-In-First-Out (FIFO) queue called 'Message queue', – Which stores the messages temporarily in a system defined memory object, to pass it to the desired process Messages are sent and received through: – Send: Name of the process to which the message is to be sent. – Receive: Name of the process from which the message is to be received. The messages are exchanged through a message queue The implementation of the message queue, send and receive methods are OS kernel dependent. Mailbox is a special implementation of message queue • Usually used for one way communication • Only a single message is exchanged through mailbox whereas 'message queue' can be used for exchanging multiple messages • One task/process creates the mailbox and other tasks/process can subscribe to this mailbox for getting message notification • The implementation of the mailbox is OS kernel dependent	8	CO6	L2
	c. Explain the concept of deadlock with a neat diagram **Solution:** **Deadlock** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process. Consider an example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other. A similar situation occurs in operating systems when there are two or more processes that hold some resources and wait for resources held by other(s). For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1 **Deadlock can arise if** the **following four conditions hold simultaneously (NecessaryConditions)** **Mutual Exclusion:** Two or more resources are non-shareable (Only one processcanuseatatime) **Hold and Wait:** A process is holding at least one resource.	4	CO6	L2

	No Preemption: A resource cannot be taken from a process unless the process releases sources. **Circular Wait:** A set of processes are waiting for each other in circular form.  Resource 1 Assigned to Waiting For Process 1 Process 2 Waiting For Resource 2 Assigned to			