Q.1 Attempt the following (any THREE) [15]

Q.1(a) Explain the various possible purposes of using an embedded system. [5]

Ans.: (i) Data Collection/Storage/Representation
- Embedded system designed for the purpose of data collection performs acquisition of data from the external world.
- Data collection is usually done for storage, analysis manipulation and transmission.
- Data can be analog or digital.
- Embedded systems with analog data capturing techniques collect data directly in the form of analog signal whereas embedded systems with digital data collection mechanism converts the analog signal to the digital signal using analog to digital converters.

(ii) Data communication
- Embedded data communication systems are deployed in applications from complex satellite communication to simple home networking systems.
- The transmission of data is achieved either by a wire-line medium or by a wire-less medium.
- Data can either be transmitted by analog means or by digital means.
- Wireless modules-Bluetooth, Wi-Fi.
- Wire-line modules-USB, TCP/IP.

(iii) Data signal processing
- Embedded systems with signal processing functionalities are employed in applications demanding signal processing like speech coding, audio video codec, transmission applications etc.
- A digital hearing aid is a typical example of an embedded system employing data processing.
- Digital hearing aid improves the hearing capacity of hearing impaired person

(iv) Monitoring
- All embedded products coming under the medical domain are with monitoring functions.
- Electrocardiogram machine is intended to do the monitoring of the heartbeat of a patient but it cannot impose control over the heartbeat.

(v) Control
- A system with control functionality contains both sensors and actuators.
- Sensors are connected to the input port for capturing the changes in environmental variable and the actuators connected to the output port are controlled according to the changes in the input variable.
- Air conditioner system used to control the room temperature to a specified limit is a typical example for CONTROL purpose.

(vi) Application specific user interface
- Buttons, switches, keypad, lights, bells, display units etc are application specific user interfaces.
- Mobile phone is an example of application specific user interface.
- In mobile phone the user interface is provided through the keypad, system speaker, vibration alert etc.
Q.1(b) Compare micro-processors and micro-controllers.  
Ans.:  

<table>
<thead>
<tr>
<th>Micro-processor</th>
<th>Micro-controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>A silicon chip representing CPU</td>
<td>A highly integrated chip that contains CPU, memory, timers and I/O ports.</td>
</tr>
<tr>
<td>It is dependent unit.</td>
<td>It is a self contained unit.</td>
</tr>
<tr>
<td>Requires external chips as timers, memory and interrupts.</td>
<td>Doesn’t require any external chips.</td>
</tr>
<tr>
<td>Mostly general purpose</td>
<td>Mostly application oriented or domain specific.</td>
</tr>
<tr>
<td>For I/O needs external peripheral interfaces.</td>
<td>Built in I/O ports that can be operated as 8, 16 or 32 bits, or as individual pins.</td>
</tr>
<tr>
<td>Targeted for market where performance is important</td>
<td>Targeted for embedded market where performance is not critical.</td>
</tr>
<tr>
<td>Limited power saving features</td>
<td>Lots of power saving features.</td>
</tr>
</tbody>
</table>

Q.1(c) Compare RISC and CISC controllers.  
Ans.:  

<table>
<thead>
<tr>
<th>RISC</th>
<th>CISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meaning</td>
<td>Reduced Instruction Set Computing</td>
</tr>
<tr>
<td>Number of instructions</td>
<td>Lesser</td>
</tr>
<tr>
<td>Instruction pipelining</td>
<td>Exist</td>
</tr>
<tr>
<td>Number of registers</td>
<td>Large</td>
</tr>
<tr>
<td>Length of instruction</td>
<td>Fixed</td>
</tr>
<tr>
<td>Instructions for one task</td>
<td>Needs many</td>
</tr>
<tr>
<td>Architecture used</td>
<td>Harvard</td>
</tr>
<tr>
<td>Silicon usage</td>
<td>Less</td>
</tr>
<tr>
<td>Operations performed on</td>
<td>Registers only. (only load and store works with memory)</td>
</tr>
</tbody>
</table>

Q.1(d) Write a note on Application specific ICs. (IC technology)  
Ans.:  
- Processors vary in their customization for the problem at hand  
- Depending on that it can be general purpose processor, application specific processor or special purpose processor.  
- The manner in which a digital (gate-level) implementation is mapped onto an IC  
  - IC: Integrated circuit, or "chip"  
  - IC technologies differ in their customization to a design  
  - IC’s consist of numerous layers  
- Three types of IC technologies  
  - Full-custom/VLSI  
  - Semi-custom ASIC (gate array and standard cell)  
  - PLD (Programmable Logic Device)  
- All layers are optimized for an embedded system’s particular digital implementation  
  - Placing transistors  
  - Sizing transistors  
  - Routing wires
• Benefits
  - Excellent performance, small size, low power
• Drawbacks
  - High NRE cost, long time-to-market
• Lower layers are fully or partially built
  - Designers are left with routing of wires and maybe placing some blocks
• Benefits
  - Good performance, good size, less NRE cost than a full-custom implementation
• Drawbacks
  - Still require weeks to months to develop

Q.1(e) What is embedded firmware? [5]
Ans.: • In electronic systems and computing, firmware is a type of computer program that provides the low-level control for the device’s specific hardware.
• Firmware can either provide a standardized operating environment for the device’s more complex software (allowing more hardware-independence)
• Or, for less complex devices, act as the device’s complete operating system, performing all control, monitoring and data manipulation functions.
• Typical examples of devices containing firmware are embedded systems, consumer appliances, computers, computer peripherals, and others.
• Almost all electronic devices beyond the simplest contain some firmware.
• Firmware is held in non-volatile memory devices such as ROM, EPROM, or flash memory.
• Hanging the firmware of a device may rarely or never be done during its lifetime; some firmware memory devices are permanently installed and cannot be changed after manufacture.
• Common reasons for updating firmware include fixing bugs or adding features to the device.
• This may require ROM integrated circuits to be physically replaced, or flash memory to be reprogrammed through a special procedure.
• Firmware such as the program of an embedded system may be the only program that will run on the system and provide all of its functions.
• Flashing involves the overwriting of existing firmware or data, contained in EEPROM or flash memory modules present in an electronic device, with new data.
• There are following ways for developing the embedded firmware.
  1. Write program in high level languages like c/c++ using IDE which contains an editor, compiler, linker, debugger, simulator etc.
  2. Write program in assembly language using instructions supported by your application’s target controller.
• Program written in higher language is converted into hex file using cross compiler.

Q.1(f) Explain the characteristics of an embedded system. [5]
Ans.: Following are some of the characteristics of an embedded system that make it different from a general purpose computer:
1. Application and Domain specific
   • An embedded system is designed for a specific purpose only. It will not do any other task.
   • Eg. A washing machine can only wash, it cannot cook
2. 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.
• Eg. 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.

3. 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.

4. 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 dashboard. Each one is an independent embedded system yet the entire car can be said to function properly only if all the systems work together.

5. 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.

6. 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.

Q.2 Attempt the following (any THREE) [15]
Q.2(a) Explain domain specific embedded system. [5]
Ans.:

![Diagram of a car with various embedded systems highlighted]
Automotive Embedded System (AES)
The Automotive industry is one of the major application domains of embedded systems.

Automotive embedded systems are the one where electronics take control over the mechanical system. Ex. Simple viper control.

The number of embedded controllers in a normal vehicle varies somewhere between 20 to 40 and can easily be between 75 to 100 for more sophisticated vehicles.

One of the first and very popular use of embedded system in automotive industry was microprocessor based fuel injection.

Some of the other uses of embedded controllers in a vehicle are listed below:
1. Air Conditioner
2. Engine Control
3. Fan Control
4. Headlamp Control
5. Automatic break system control
6. Wiper control
7. Air bag control
8. Power Windows

AES are normally built around microcontrollers or DSPs or a hybrid of the two and are generally known as Electronic Control Units (ECUs).

Q.2(b) What is Memory mapped? [5]

Ans.:
A Memory Map is the processor’s “address book.” It shows what these devices look like to the processor. The memory map contains one entry for each of the memories and peripherals that are accessible from the processor’s memory space.

All processors store their programs and data in memory.

These chips are located in the processor’s memory space, and the processor communicates with them by way of two sets of electrical wires called the address bus and the data bus. To read or write a particular location in memory, the processor first writes the desired address onto the address bus. The data is then transferred over the data bus.

A memory map is a table that shows the name and address range of each memory device and peripheral that is located in the memory space.

Organize the table such that the lowest address is at the bottom and the highest address is at the top. Each time a new device is added, add it to the memory map, place it in its approximate location in memory and label the starting and ending addresses, in hexadecimal. After inserting all of the devices into the memory map, be sure to label any unused memory regions as such.

The block diagram of the Printer sharing device shown above contains three devices attached to the address and data buses. These devices are the RAM and ROM and a Serial Controller.

Let us assume that the RAM is located at the bottom of memory and extends upward for the first 128 KB of the memory space.

The ROM is located at the top of memory and extends downward for 256 KB. But considering the ROM contains two ROMs-an EPROM and a Flash memory device-each of size 128 KB.
The third device, the Serial Controller, is a memory-mapped peripheral whose registers are accessible between the addresses say 70000h and 72000h.

The diagram below shows the memory map for the printer sharing device.

For every embedded system, a header file should be created that describes these important features and provides an abstract interface to the hardware. It allows the programmer to refer to the various devices on the board by name, rather than by address.

The part of the header file below describes the memory map:

```
#define RAM_BASE (void *) 0x00000000
#define SC_BASE (void *) 0x70000000
#define SC_INTACK (void *) 0x70001000
#define FLASH_BASE (void *) 0xC0000000
#define EPROM_BASE (void *) 0xE0000000
```

Q.2(c) What is device driver? 
Ans.: 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 counterparts. 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:
- Modularization, it makes the structure of the overall software is easier to understand.
- There exists only one module that interacts directly with the peripheral's registers making communication easier.
- 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.
An example is shown below for a timer counter data structure.

```c
struct TimerCounter
{
    unsigned short count; // Current Count, offset 0x00
    unsigned short maxCountA;// Maximum Count, offset 0x02
    unsigned short _reserved; // Unused Space, offset 0x04
    unsigned short control; // Control Bits, offset 0x06
};
```

To make the bits within the control register easier to read and write individually, we define the following bitmasks:

- `#define TIMER_ENABLE 0xC000 // Enable the timer.`
- `#define TIMER_DISABLE 0x4000 // Disable the timer.`
- `#define TIMER_INTERRUPT 0x2000 // Enable timer interrupts.`
- `#define TIMER_MAXCOUNT 0x0020 // Timer complete?`
- `#define TIMER_PERIODIC 0x0001 // Periodic timer?`

**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.

**Q.2(d) Short note on Memory Testing.**

**Ans.**

The purpose of a memory test is to confirm that each storage location in a memory device is working.

Memory Testing is performed when prototype hardware is ready and the designer needs to verify that address and data lines are correctly wired and memory chips are working properly.

Basic idea implement in testing can be understood by this simple task:

Write some set of Data values to each Address in Memory and Read it back to verify.

Ex. If number '50' is stored at a particular Address it is expected to be there unless rewritten or erased.

If all values are verified by reading back then Memory device passes the test.

Only through careful selection of data values can make sure passing result to be meaningful.

Difficulties involved in memory testing:

- It can be difficult to detect all memory problems with a simple test.
- Many Embedded Systems include Memory Tests only to detect catastrophic memory failures which might not even notice memory chips removal.
- Memory Problems rarely occur with the chip itself, but due to a variety of post production tests to check quality this possibility is ruled out.
- Catastrophic Failure is a memory problem that occurs due to physical and electrical damage, it is uncommon and easily detectable.

A common source of memory problems is associated with the circuit board. Typical circuit board problems are:

- Circuit board wiring between Processor & Memory device.
- Missing Memory chip.
- Improperly inserted Memory chip.

Circuit board wiring between Processor & Memory device.

These are usually caused by,

- An error in design
- An error in production of the board
- Any damage after manufacture
Q.2(e) What is control and status registers? [5]

Ans.: Control and status registers are the basic interface between and embedded processor and peripheral device.

These registers are a part of peripheral hardware and their location size and individual meanings are feature of the peripheral.

For example, The registers vary from device to device: example the registers within a serial controller are very different from those in a timer.

Depending upon the design of the processor and target board, peripheral devices are located either in the processor’s memory space or within the I/O space.

It is common for Embedded Systems to include some peripherals of each type. These are called Memory-Mapped and I/O-mapped peripherals.

Of the two types, memory-mapped peripherals are generally easier to work with and are increasingly popular.

Memory-mapped control and status registers can be used just like ordinary variables.

Q.2(f) Short note on watchdog timer. [5]

Ans.: It is hardware equipment.

It is special purpose hardware that protects the system from software hangs.

Watchdog timer always counts down from some large number to zero

This process takes a few seconds to reset, in the meantime, it is possible for embedded software to “kick” the watchdog timer, to reset its counter to the original large number.

If the timer expires i.e. counter reaches zero, the watchdog timer will assume that the system has entered a state of software hang, then resets the embedded processor and restarts the software

It is a common way to recover from unexpected software hangs

The figure below diagrammatically represents the working of the watchdog timer

Q.3 Attempt the following (any THREE) [15]

Q.3(a) List down Features of 8051 Microcontroller. [5]

Ans.: An 8051 microcontroller comes bundled with the following features:

- 64K bytes on-chip program memory (ROM)
- 128 bytes on-chip data memory (RAM)
- Four register banks
- 128 user defined software flags
- 8-bit bidirectional data bus
- 16-bit unidirectional address bus
- 32 general purpose registers each of 8-bit
- 16 bit Timers (usually 2, but may have more or less)
Q.3(b) Draw block diagram of 8051 Microcontroller.

Ans.:

Q.3(c) Explain Flag Bits and PSW Register.

Ans.: The program status word (PSW) register is an 8-bit register, also known as flag register. It is of 8-bit wide but only 6-bit of it is used. The two unused bits are user-defined flags. Four of the flags are called conditional flags, which means that they indicate a condition which results after an instruction is executed. These four are CY (Carry), AC (auxiliary carry), P (parity), and OV (overflow). The bits RS0 and RS1 are used to change the bank registers.

The following figure shows the program status word register.

The PSW Register contains that status bits that reflect the current status of the CPU.

<table>
<thead>
<tr>
<th>CY</th>
<th>AC</th>
<th>F0</th>
<th>RS1</th>
<th>RS0</th>
<th>OV</th>
<th>-</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY</td>
<td>PSW.7</td>
<td>Carry Flag</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AC</td>
<td>PSW.6</td>
<td>Auxiliary Carry Flag</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>PSW.5</td>
<td>Flag 0 available to user for general purpose.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RS1</td>
<td>PSW.4</td>
<td>Register Bank selector bit 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RS0</td>
<td>PSW.3</td>
<td>Register Bank selector bit 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OV</td>
<td>PSW.2</td>
<td>Overflow Flag</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>PSW.1</td>
<td>User definable FLAG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>PSW.0</td>
<td>Parity FLAG. Set/cleared by hardware during instruction cycle to indicate even/odd number of 1 bit in accumulator.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We can select the corresponding Register Bank bit using RS0 and RS1 bits.

<table>
<thead>
<tr>
<th>RS1</th>
<th>RS2</th>
<th>Register Bank</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>00H-07H</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>08H-0FH</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>10H-17H</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>18H-1FH</td>
</tr>
</tbody>
</table>
• **CY, the carry flag** - This carry flag is set (1) whenever there is a carry out from the D7 bit. It is affected after an 8-bit addition or subtraction operation. It can also be reset to 1 or 0 directly by an instruction such as "SETB C" and "CLR C" where "SETB" stands for set bit carry and "CLR" stands for clear carry.

• **AC, auxiliary carry flag** - If there is a carry from D3 and D4 during an ADD or SUB operation, the AC bit is set; otherwise, it is cleared. It is used for the instruction to perform binary coded decimal arithmetic.

• **P, the parity flag** - The parity flag represents the number of 1's in the accumulator register only. If the A register contains odd number of 1's, then P = 1; and for even number of 1's, P = 0.

• **OV, the overflow flag** - This flag is set whenever the result of a signed number operation is too large causing the high-order bit to overflow into the sign bit. It is used only to detect errors in signed arithmetic operations.

Q.3(d) Write an 8051 C program to get a byte of data form P0. If it is less than 100, send it to P1; otherwise, send it to P2.

**Ans.:**

```c
#include <reg51.h>

void main(void)
{
    unsigned char mybyte;
    P0 = 0xFF; //make P0 input port
    while (1)
    {
        mybyte = P0; //get a byte from P0
        if (mybyte<100)
            P1 = mybyte; //send it to P1
        else
            P2 = mybyte; //send it to P2
    }
}
```

Q.3(e) The data pins of an LCD are connected to P1. The information is latched into the LCD whenever its Enable pin goes from high to low. Write an 8051 C program to send "The Earth is but One Country" to this LCD.

**Ans.:**

```c
#include <reg51.h>
#define LCDData P1 //LCDData declaration
sbit En = P2^0; //the enable pin

void main(void)
{
    unsigned char message[] = "The Earth is but One Country";
    unsigned char z;
    for (z=0;z<28;z++) //send 28 characters
    {
        LCDData = message[z];
        En=1; //a high-
        En=0; //to-low pulse to latch data
    }
}
```

Q.3(f) Write an 8051 C program to convert ASCII digits of '4' and '7' to packed BCD and display them on P1.

**Ans.:**

```c
#include <reg51.h>

void main(void)
{
```
unsigned char bcdbyte;
unsigned char w = '4';
unsigned char z = '7';
w = w & 0x0F;
w = w << 4;
z = z & 0x0F;
bcdbyte = w | z;
P1 = bcdbyte;
}

Q.4 Attempt the following (any THREE) [15]
Q.4(a) Design Interface of 8051 with 8 LED and send 0 to 255 on it. [5]
Ans.:

Q.4(b) What are the selection criteria for Controller? [5]
Ans.: Selection of a microcontroller for any application depends on design factors. A good designer finalizes his selection based on a comparative study of the design factors. The important factors to be considered in the selection process of a microcontroller are listed below.

- **Feature Set**
  The important queries related to the feature set are: Does the microcontroller support all the peripherals required by the applications, say serial interface, parallel interface, etc.? Does it satisfy the general I/O port requirements by the application? Does the controller support sufficient number of timer and counters? Does the controller support built-in ADC/DAC hardware in case of signal processing applications? Does the controller provide the required performance?

- **Code Memory Space**
  If the target processor/controller application is written in C or any other high level language, does the controller support sufficient code memory space to hold the compiled hex code (In case of controllers with internal code memory)?

- **Data Memory Space**
  Does the controller support sufficient internal data memory (on chip RAM) to hold run time variables and data structures?

- **Development Support**
  Development support is another import factor for consideration. It deals with- Does the controller manufacture provide cost-effective development tools? Does the manufacture provide product samples for prototyping and sample development stuffs to
alleviate the development pains? Does the controller support third party development tools? Does the manufacture provide technical support if necessary?

- **Availability**
Availability is another important factor that should be taken into account for the selection process. Since the product is entirely dependent on the controller, the product development time and time to market the product solely depends on its availability. By technical terms it is referred to as Lead time. Lead time is the time clapsed between the purchase order approval and the supply of the product.

- **Power Consumption**
The power consumption of the controller should be minimal. It is a crucial factor since high power requirement leads to bulky supply design. The high power dissipation also demands for cooling fans and it will make the overall system messy and expensive. Controllers should support idle and power down modes of operation to reduce power consumption.

- **Cost**
Last but not least, cost is a big deciding factor in selecting a controller. The cost should be within the reachable limit of the end user and the targeted user should not be high tech. Remember the ultimate aim of a product is to gain marginal benefit.

**Q.4(c) Why to select 8051 microcontroller?**

**Ans.:** 8051 is a very versatile microcontroller featuring powerful Boolean processor which supports bit manipulation instructions for real time industrial control applications. The standard 8051 architecture supports 6 interrupts (2 external interrupts, 2 timer interrupts and 2 serial interrupts), two 16 bit timer/ counters, 32 I/O lines and a programmable full duplex serial interface. Another fascinating feature of 8051 is the way it handles interrupts. The interrupts have two priority levels and each interrupt is allocated fixed 8 bytes of code memory. This approach is very efficient in real time application. Though 8051 is invented by Intel, today it is available in the market from more than 20 vendors and with more than 100 variants of the original 8051 flavour, supporting CAN, USB, SPI and ICP/IP interfaces, integrated ADC/DAC, LCD Controller and extended number of I/O ports. Another remarkable feature of 8051 is its low cost. The 8051 flash microcontroller (AT89C51) from Atmel is available in the market for less than 1 USS per piece. So imagine its cost for high volume purchases.

**Q.4(d) What is Simulator?**

**Ans.:** Simulators are used for embedded firmware debugging. Simulator simulates the target hardware, while the code execution can be inspected. Simulators have the following characteristics which make them very much favorable:

- Purely software based
- No need of target system (hardware)
- Support only for basic operations
- Cannot Support or lack real time behavior

**Advantages**

(i) **Simple and straight forward.**
Simulators are a software utility with assumptions about the underlying hardware. So it only requires concentrating on debugging of the code, hence straight forward.

(ii) **No Hardware**
Simulators are purely software oriented.
The IDE simulates the target CPU. The user needs to know only about the target specific details like memory map of various devices.
Since no hardware is required the code can be written and tested even before the hardware prototype is ready thus saving development time

(iii) Simulation options
Simulators provide various simulation options like I/O peripherals or CRO or Logic analyzers.
Simulators I/O support can be used to edit values for I/O registers.

(iv) Simulation of abnormal conditions
Using simulator the code can be tested for any desired value.
This helps to study the code behavior in abnormal conditions without actually testing it on the hardware.

Disadvantages
(i) Lack of real time behavior
A simulator assumes the ideal condition for code execution.
Hence the developer may not be able to debug the code under all possible combinations of input.
The results obtained in simulation may deviate from actual results on target hardware.

(ii) Lack of real timeliness
The I/O condition in hardware is unpredictable. So the output of simulation is usually under ideal condition and hence lacks timeliness.

Q.4(e) What is process of converting source code to hex file? [5]
Ans.:
Definition: The process which converts source code to executable code is called as the build process.

The build process for embedded systems is different. This is because the code to be run on an embedded system is written on one platform i.e. general purpose computer and executed on another platform i.e. the target hardware.

An Embedded system would also use tools such as a Compiler, Linker, Locater and Debugger to perform the entire build process. These tools would be a part of a larger IDE.

A compiler which produces the executable code to be run on a different platform is called a cross-compiler; else it is called a native compiler.

Ex. Turbo C++ is a native compiler. The compiler in case of embedded systems development is a cross compiler.

The build process involves three steps:
Compiling
Linking
Locating
Q.4(f) What is Infinite loop and explain with help of LED blinking Example? [5]

Ans.: Blinking Led Program and Infinite Loop

The substitute for hello world program could be a program that blinks an LED. LEDs are used in almost every embedded system. Also the code used to program an LED would be very small.

To blink an LED we would require the following hardware: • LED

• A microcontroller or microprocessor

The LED can be connected to any available port i.e P1, P2, P3, P4 on the microprocessor. Assuming the LED is connected to port 2, i.e. P2 its state is controlled by a bit in a register called the Port 2 I/O Latch Register, also known the P2LTC2.

The structure of the program would be like this:

```c
void main ()
{
    while (1)
    {
        toggle(LED_1); /* change the state of the LED. */
        delay (500); /* Pause for 500 milliseconds. */
    }
}
```

The above piece of code is hardware independent hence can be implemented for any circuit. It contains two functions namely: toggle() & delay()

• toggle(): This function is used to toggle the state of the LED.
• delay(): This function is used to introduce a delay of 500 ms every time the LED is toggled

The implementation of toggle() and delay() is hardware specific.

Infinite Loop

The code for every embedded program is written in an infinite loop. This is because the embedded system is supposed to run every time it is turned on till the time its power goes off or it stops functioning.

The code for blinking LED is also enclosed in an infinite loop. The functions toggle() and delay() run infinite number of times.

An application of an embedded system has an infinite loop around its code. It’s just like the program you did to implement switch case where the program has to run continuously until the user selects to exit.

Q.5 Attempt the following (any THREE) [15]

Q.5(a) What is selection process of OS? [5]

Ans.: The process of selecting the best commercial operating system that best fits the needs of one’s project depends on various factors. Commercial operating systems form a continuum of functionality, performance and price.

Operating Systems that offer only a basic scheduler and a few other system calls are inexpensive and come with the source code that one can modify and do not require payment of royalties.

While on the other hand operating systems that include a lot of useful functionality beyond just the scheduler are quite expensive and royalties due on every copy shipped in ROM and they might also make a stronger guarantees about real-time performance.
Two important points to be considered while selecting an operating system: Put your processor, real time performance and budgetary requirements first. Contact all of the vendors of the remaining operating systems for more detailed technical information.

Q.5(b) What is real time characteristic? [5]
Ans.: An Operating system is called "Real-Time Operating System" (RTOS) only if it has following characteristics:

(i) **Deterministic**
An OS is said to be deterministic if the worst case execution time of each of the system calls is calculable.

The data sheet of an OS should publish the real-time behavior of its RTOS provides average, minimum and maximum number of clock cycles required by each system call.

(ii) **Interrupt Latency**
Interrupt Latency is the total length of time from an interrupt signal's arrival at the processor to the start of the associated interrupt service routine.

(iii) **Context Switch**
Context Switch is important because it represents overhead across your entire system.

Q.5(c) What operating system and it’s types? [5]
Ans.: An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.

Types of operating systems

(i) **Single- and multi-tasking**
A single-tasking system can only run one program at a time, while a multi-tasking operating system allows more than one program to be running in concurrency. This is achieved by time-sharing, dividing the available processor time between multiple processes that are each interrupted repeatedly in time slices by a task-scheduling subsystem of the operating system. Multi-tasking may be characterized in preemptive and co-operative types. In preemptive multitasking, the operating system slices the CPU time and dedicates a slot to each of the programs. Unix-like operating systems, e.g., Solaris, Linux, as well as AmigaOS support preemptive multitasking. Cooperative multitasking is achieved by relying on each process to provide time to the other processes in a defined manner. 16-bit versions of Microsoft Windows used cooperative multi-tasking. 32-bit versions of both Windows NT and Win9x, used preemptive multi-tasking.

(ii) **Single- and multi-user**
Single-user operating systems have no facilities to distinguish users, but may allow multiple programs to run in tandem. A multi-user operating system extends the basic concept of multi-tasking with facilities that identify processes and resources, such as disk space, belonging to multiple users, and the system permits multiple users to interact with the system at the same time. Time-sharing operating systems schedule tasks for efficient use of the system and may also include accounting software for cost allocation of processor time, mass storage, printing, and other resources to multiple users.

(iii) **Distributed**
A distributed operating system manages a group of distinct computers and makes them appear to be a single computer. The development of networked computers that could be linked and communicate with each other gave rise to distributed computing. Distributed computations are carried out on more than one machine. When computers in a group work in cooperation, they form a distributed system.
(iv) **Templated**

In an OS, distributed and cloud computing context, templating refers to creating a single virtual machine image as a guest operating system, then saving it as a tool for multiple running virtual machines. The technique is used both in virtualization and cloud computing management, and is common in large server warehouses.

(v) **Embedded**

Embedded operating systems are designed to be used in embedded computer systems. They are designed to operate on small machines like PDAs with less autonomy. They are able to operate with a limited number of resources. They are very compact and extremely efficient by design. Windows CE and Minix 3 are some examples of embedded operating systems.

(vi) **Real-time**

A real-time operating system is an operating system that guarantees to process events or data by a specific moment in time. A real-time operating system may be single- or multi-tasking, but when multitasking, it uses specialized scheduling algorithms so that a deterministic nature of behavior is achieved. An event-driven system switches between tasks based on their priorities or external events while time-sharing operating systems switch tasks based on clock interrupts.

(vii) **Library**

A library operating system is one in which the services that a typical operating system provides, such as networking, are provided in the form of libraries and composed with the application and configuration code to construct a unikernel: a specialized, single address space, machine image that can be deployed to cloud or embedded environments.

Q.5(d) **What is Emulator/Simulator/Disassembler/Decompiler?** [5]

**Ans.:** Emulators

The terms simulators and emulators are very confusing but their basic functionality is the same i.e. to debug the code. There is a difference in which this is achieved by both the tools.

A simulator is a utility program that duplicates the target CPU and simulates the features and instructions supported by target CPU whereas an Emulator is a self contained hardware device which emulates the target CPU.

The Emulator hardware contains the necessary emulation logic and is connected to the debugging application that runs on the host PC.

The Simulator 'simulates' while the Emulator 'emulates'.

**Simulator**

Simulators are used for embedded firmware debugging. Simulators have the following characteristics which make them very much favorable:

- Purely software based
- No need of target system (hardware)
- Support only for basic operations
- Cannot Support or lack real time behavior

**Disassembler/Decompiler**

A Disassembler/Decompiler is a reverse engineering tool. Reverse Engineering is used in embedded system to find out the secret behind the working of a proprietary product. A DISASSEMBLER is a utility program which converts machine codes into target processor specific assembly code/instruction. The process of converting machine codes to assembly code is called disassembling.
A DECOMPIILER is a utility program for translating machine codes into corresponding high level language instruction. A decompiler performs the reverse operation of a compiler/cross-compiler.

Q.5(e) Short note on EDLC. [5]
Ans.: Embedded Product Development Life Cycle (Edlc)
EDLC is Embedded Product Development Life Cycle
It is an Analysis - Design - Implementation based problem solving approach for embedded systems development.

There are three phases to Product development:

Analysis involves understanding what product needs to be developed
Design involves what approach to be used to build the product
Implementation is developing the product by realizing the design.

Need for EDLC
EDLC is essential for understanding the scope and complexity of the work involved in embedded systems development. It can be used in any developing any embedded product. EDLC defines the interaction and activities among various groups of a product development phase.
Example: project management, system design

Objectives of EDLC
The ultimate aim of any embedded product in a commercial production setup is to produce Marginal benefit. Marginal is usually expressed in terms of Return On Investment. The investment for product development includes initial investment, manpower, infrastructure investment etc.

EDLC has three primary objectives are:
1. Ensure that high quality products are delivered to user
2. Risk minimization defect prevention in product development through project management
3. Maximize the productivity

DIFFERENT PHASES OF EDLC
The following figure depicts the different phases in EDLC:
Q.5(f) What is Firmware Debugging?

Ans.: Debugging in embedded application is the process of diagnosing the firmware execution, monitoring the target processor's registers and memory while the firmware is running and checking the signals on various buses of hardware.

Debugging is classified into Hardware Debugging and Firmware Debugging. Hardware Debugging deals with debugging the various aspects of hardware involved in the embedded system.

The various tools used for hardware debugging are Multimeter, CRO, Logic Analyzers and Function Generators.

Firmware Debugging involves inspecting the code, its execution flow, changes to different registers on code execution.

It is done to find out the bugs or errors in code which produces unexpected behavior in the system.

There is a wide variety of firmware debugging techniques available that have advanced from basic to advanced.

Some of the tools used are Simulators and Emulators.

**Simulators**
The terms simulators and emulators are very confusing but their basic functionality is the same i.e. to debug the code. There is a difference in which this is achieved by both the tools.

A simulator is a utility program that duplicates the target CPU and simulates the features and instructions supported by target CPU whereas an Emulator is a self contained hardware device which emulates the target CPU.

The Emulator hardware contains the necessary emulation logic and is connected to the debugging application that runs on the host PC.

The Simulator 'simulates' while the Emulator 'emulates'