Ques1. What is embedded system? Give Characteristics and Applications of Embedded systems

Ans. An embedded device is a machine that has software program embedded in PC hardware. It makes a device devoted to a selected a part of an application or fabricated from a bigger device. Depending on the application, embedded device can be programmable or non-programmable.

CHARACTERISTICS OF EMBEDDED SYSTEM

- Embedded systems are designed for a specific challenge. Even though they use laptop strategies, they cannot be used as a widespread reason computer the use of a spread of different programs for one of a kind project. On this manner, their function can be focused on what they need to do, and they can consequently be made inexpensive and greater correctly.
- The software program for embedded systems is usually called firmware. In preference to being saved on a disc, where many applications can be stored, the single programmed for an embedded gadget is usually saved on the chip and its miles called firmware.
- Commonly, an embedded device executes a particular operation and does the similar always. As an example: a pager is continuously functioning as a pager.
- All of the computing structures have boundaries on layout metrics, but the ones can be in particular tight. Design calculation is a measure of executive functions like length, strength, price and also performance.
- It should carry out rapid sufficient and eat much less strength to increase battery existence.
- It has to be based totally on a microcontroller or microprocessor primarily based.
- An embedded system is built in with hardware and software program where the hardware is used for safety and performance and software is used for more flexibility and capabilities.

EMBEDDED SYSTEM APPLICATIONS

The programs of an embedded system fundamentals consist of smart cards, PC networking, satellites, telecommunications, digital consumer electronics, missiles, etc.

- Embedded systems in cars include motor control, Cruise manipulates, frame safety, engine safety, robotics in a meeting line, automobile multimedia, automobile enjoyment, e-com get entry to, mobiles etc.
- Embedded structures in telecommunications consist of networking, mobile computing, and wireless communications, and so on.
- Embedded structures in smart playing cards consist of banking, smartphone and protection systems.
- Embedded systems in satellites and missiles include defense, conversation, and aerospace.
- Embedded structures in computer networking & peripherals consist of photo processing, networking systems, printers, community cards, monitors, and shows.
- Embedded structures in virtual patron electronics include set-pinnacle boxes, dads, high definition TVs and digital cameras.

Consequently, that is all approximately the fundamentals of embedded device fundamentals and applications. We all recognize that embedded systems are gorgeous structures that play a crucial position in lots of packages like gadget, industrial instrumentation, and so forth.

Ques2. Write and explain various components of Embedded systems

Ans. The embedded structures fundamentals include the components of embedded machine hardware, embedded machine sorts and numerous characteristics. An embedded system has three foremost components: embedded gadget hardware, embedded gadget software program and operating gadget

EMBEDDED SYSTEMS HARDWARE

As with every electronic device, an embedded device calls for a hardware platform in which it plays the operation. Embedded device hardware is built with a microprocessor or microcontroller. The embedded device hardware has factors like entering output (I/O) interfaces, consumer interface, reminiscence and the display. Usually, an embedded system consists of:

- Power supply
- Processor
- Timers
- Serial exchange ports
- Input/output circuits
- System service specific circuits

EMBEDDED SYSTEM SOFTWARE

The embedded device software is written to carry out a selected feature. It is commonly written in a high degree format after which compiled right down to provide code that can be lodged inside a non-risky reminiscence within the hardware. An embedded gadget software program is designed to keep in view of the 3 limits:

- Availability of machine memory
- Availability of processor's speed
- When the system runs continuously, there may be a want to limit power dissipation for events like prevent, run and wake up.
-

REAL TIME OPERATING SYSTEM

A device is said to be actual time if it's miles vital to complete its work and supply its service on time. Actual time running system manages the software and provides a mechanism to let the processor run. The actual time operating gadget is accountable for handling the hardware sources of a computer and host programs which run on the PC.

MEMORY

In an embedded gadget, there are special kinds of memories. The extraordinary varieties of processors utilized in an embedded system consisting of digital signal processor (DSP), microprocessor, RISC processor, microcontroller, ASSP processor, ASIP processor.

PROCESSORS

Amazing processors utilized in embedded systems are microprocessor, (DSP) virtual sign processor, microcontroller, RISC processor, ASIP processor, arm processor and ASSP processor.

Ques3. Explain various issues and challenges in Embedded system design

Ans. **Challenges in Embedded System Design:**

a) Optimizing the Design Metrics and
b) Formalism of System Metrics

- Amount and type of hardware needed
- Taking into account the design metrics
- Optimizing the Power Dissipation
- Disable use of certain structural units of the processor to reduce power dissipation
- Process Deadlines
- Flexibility and Upgradeability
- Reliability
- Testing, Verification and Validation

Ques4. What is the difference between RISC and CISC Architectures

Ans. The main difference between RISC and CISC is in the number of computing cycles each of their instructions take. The difference the number of cycles is based on the complexity and the goal of their instructions.

	RISC	**CISC**
Acronym	It stands for 'Reduced Instruction Set Computer'.	It stands for 'Complex Instruction Set Computer'.
Definition	The RISC processors have a smaller set of instructions with few addressing nodes.	The CISC processors have a larger set of instructions with many addressing nodes.
Memory unit	It has no memory unit and uses a separate hardware to implement instructions.	It has a memory unit to implement complex instructions.
Program	It has a hard-wired unit of programming.	It has a micro-programming unit.
Design	It is a complex complier design.	It is an easy complier design.
Calculations	The calculations are faster and precise.	The calculations are slow and precise.

Decoding	Decoding of instructions is simple.	Decoding of instructions is complex.
Time	Execution time is very less.	Execution time is very high.
External memory	It does not require external memory for calculations.	It requires external memory for calculations.
Pipelining	Pipelining does function correctly.	Pipelining does not function correctly.
Stalling	Stalling is mostly reduced in processors.	The processors often stall.
Code expansion	Code expansion can be a problem.	Code expansion is not a problem.
Disc space	The space is saved.	The space is wasted.
Applications	Used in high end applications such as video processing, telecommunications and image processing.	Used in low end applications such as security systems, home automations, etc.

Ques5. Write and Explain the Processor Architecture of ARM processor

Ans. ARM, previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of reduced instruction set computing (RISC) architectures for computer processors, configured for various environments. ARM Holdings periodically releases updates to architectures and core designs.

Following are the units of architecture

Register Bank – 2 read ports, 1 write ports, access any register –1 additional read port, 1 additional write port for r15 (PC)

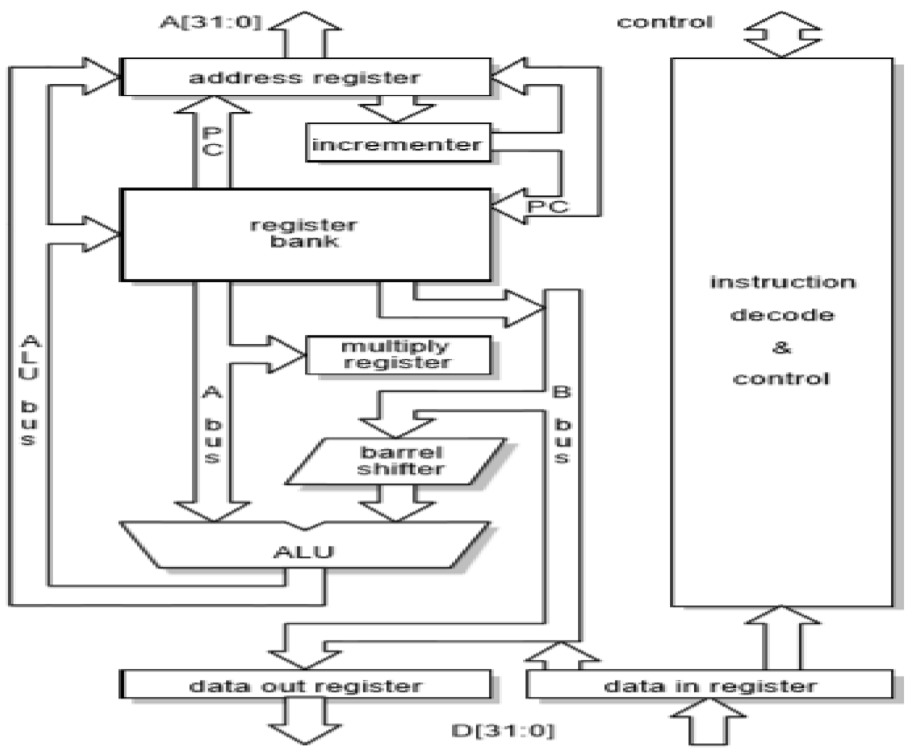
Barrel Shifter –Shift or rotate the operand by any number of bits

Arithmetic and Logical Unit

Address register and incrementer

Data Registers –Hold data passing to and from memory

Instruction Decoder and Control



Ques6. Explain data operations and flow control within the instruction set of ARM processor

Ans. ARM Instruction Set

Data processing instructions

Data transfer instructions

Control flow instructions

Writing simple assembly language programs

Data processing and operations:

- They are move, arithmetic, logical, comparison and multiply instructions.
- Most data processing instructions can process one of their operands using the barrel shifter.
- General rules: –
  - ❖ All operands are 32-bit, coming from registers or literals.
  - ❖ The result, if any, is 32-bit and placed in a register
  - ❖ 3-address format

Flow control instructions:

Branch instruction

Conditional execution

Ques7. Write and Explain the Processor Architecture and Memory organization of SHARC processor

Ans. Analog Devices 32-Bit Floating-Point SHARC Processors are based on a Super Harvard architecture that balances exceptional core and memory performance with outstanding I/O throughput capabilities. This "Super" Harvard architecture extends the original concepts of separate program and data memory busses by adding an I/O processor with its associated dedicated busses. In addition to satisfying the demands of the most computationally intensive, real-time signal-processing applications, SHARC processors integrate large memory arrays and application-specific peripherals designed to simplify product development and reduce time to market.

Common Architectural Features

32/40-Bit IEEE Floating-Point Math

32-Bit Fixed-Point Multipliers with 64-Bit Product & 80-Bit Accumulation

No Arithmetic Pipeline; All Computations Are Single-Cycle

Circular Buffer Addressing Supported in Hardware

32 Address Pointers Support 32 Circular Buffers

Six Nested Levels of Zero-Overhead Looping in Hardware

Rich, Algebraic Assembly Language Syntax

Instruction Set Supports Conditional Arithmetic, Bit Manipulation, Divide & Square Root, Bit Field Deposit and Extract

DMA Allows Zero-Overhead Background Transfers at Full Clock Rate Without Processor Intervention

Ques8. Write short note on parallelism within instruction

Ans. Pipelining can overlap the execution of instructions when they are independent of one another. This potential overlap among instructions is called instruction-level parallelism (ILP) since the instructions can be evaluated in parallel.

The amount of parallelism available within a basic block (a straight-line code sequence with no branches in and out except for entry and exit) is quite small. The average dynamic branch frequency in integer programs was measured to be about 15%, meaning that about 7 instructions execute between a pair of branches. Since the instructions are likely to depend upon one another, the amount of overlap we can exploit within a basic block is likely to be much less than 7.

To obtain substantial performance enhancements, we must exploit ILP across multiple basic blocks. The simplest and most common way to increase the amount of parallelism available among instructions is to exploit parallelism among iterations of a loop. This type of parallelism is often called loop-level parallelism.

Ques9. Describe DMA in embedded systems

Ans. Direct memory access (DMA) is a means of having a peripheral device control a processor's memory bus directly. DMA permits the peripheral, such as a UART, to transfer data directly to or from memory without having each byte (or word) handled by the processor. Thus DMA enables more efficient use of interrupts, increases data throughput, and potentially reduces hardware costs by eliminating the need for peripheral-specific FIFO buffers.

In a typical DMA transfer, some event (such as an incoming data-available signal from a UART) notifies a separate device called the DMA controller that data needs to be transferred to memory. The DMA controller then asserts a DMA request signal to the CPU, asking its permission to use the bus. The CPU completes its current bus activity, stops driving the bus, and returns a DMA acknowledge signal to the DMA controller. The DMA controller then reads and writes one or more memory bytes, driving the address, data, and control signals as if it were itself the CPU. (The CPU's address, data, and control outputs are tri-stated while the DMA controller has control of the bus.) When the transfer is complete, the DMA controller stops driving the bus and de-asserts the DMA request signal. The CPU can then remove its DMA acknowledge signal and resume control of the bus.

Each DMA cycle will typically result in at least two bus cycles: either a peripheral read followed by a memory write or a memory read followed by a peripheral write, depending on the transfer base addresses. The DMA controller itself does no processing on this data. It just transfers the bytes as instructed in its configuration registers.

It's possible to do a flyby transfer that performs the read and write in a single bus cycle. However, though supported on the ISA bus and its embedded cousin PC/104, flyby transfers are not typical.

Processors that support DMA provide one or more input signals that the bus requester can assert to gain control of the bus and one or more output signals that the processor asserts to indicate it has relinquished the bus. A typical output signal might be named HLDA (short for HOLD Acknowledge).

When designing with DMA, address buffers must be disabled during DMA so the bus requester can drive them without bus contention. To avoid bus contention, the bus buffer used by the DMA device must not drive the address bus until after HLDA goes active to indicate that the CPU has stopped driving the bus signals, and it must stop driving the bus before the CPU drives HLDA inactive. The system design may also need pull up resistors or terminators on control signals (such as read and write strobes) so the control signals don't float to the active state during the brief period when neither the processor nor the DMA controller is driving them.

DMA controllers require initialization by software. Typical setup parameters include the base address of the source area, the base address of the destination area, the length of the block, and whether the DMA controller should generate a processor interrupt once the block transfer is complete.

It's typically possible to have the DMA controller automatically increment one or both addresses after each byte (word) transfer, so that the next transfer will be from the next memory location. Transfers between peripherals and memory often require that the peripheral address not be incremented after each transfer. When the address is not incremented, each data byte will be transferred to or from the same memory location.

Ques10. Explain Timer and counters in ARM bus

Ans. Counter/timer hardware is a crucial component of most embedded systems. In some cases a timer is needed to measure elapsed time; in others we want to count or time some external events. Here's a primer on the hardware.

Counter/timer hardware is a crucial component of most embedded systems. In some cases, a timer measures elapsed time (counting processor clock ticks). In others, we want to count or time external events. The names counter and timer can be used interchangeably when talking about the hardware. The difference in terminology has more to do with how the hardware is used in a given application.

A timer with automatic reload capability will have a latch register to hold the count written by the processor. When the processor writes to the latch, the count register is written as well. When the timer later overflows, it first generates an output signal. Then, it automatically reloads the contents of the latch into the count register. Since the latch still holds the value written by the processor, the counter will begin counting again from the same initial value.

Such a timer will produce a regular output with the same accuracy as the input clock. This output could be used to generate a periodic interrupt like a real-time operating system (RTOS) timer tick, provide a baud rate clock to a UART, or drive any device that requires a regular pulse.

A variation of this feature found in some timers uses the value written by the processor as the endpoint rather than the initial count. In this case, the processor writes into a terminal count register that is constantly compared with the value in the count register. The count register is always reset to zero and counts up. When it equals the value in the terminal count register, the output signal is asserted. Then the count register is reset to zero and the process repeats. The terminal count remains the same. The overall effect is the same as an overflow counter. A periodic signal of a pre-determined length will then be produced.

Ques11. Write and explain the software design patterns for embedded system

Ans. Embedded System Design Patterns

- Object Design Patterns
- State Design Patterns
- Hardware Interface Design Patterns
- Protocol Design Patterns
- Architecture Design Patterns
- Implementation Design Patterns

Object Design Patterns

- Half Call Design Pattern Half Call design pattern helps in simplifying systems which support interworking of multiple protocols.
- Manager Design Pattern Real-time software generally manages multiple entities of the same type. Manager Design Pattern is used to control these entities.
- Resource Manager Pattern Resource Manager keeps track of allocated and free resources.

- Message Factory and Message Interface Design Pattern Message interfaces and the rest of the logic can be decoupled using this design pattern
- Publish-Subscribe Design Patterns Decoupling of publisher and subscriber of information can be achieved by applying these design patterns.

State Design Patterns

- Hierarchical State Machine Hierarchical State Machine design is introduced and compared with conventional state design.
- State Machine Inheritance This article discusses several ways in which new state machines can be defined by inheriting from existing state machines.
- Collector State Pattern This state pattern is used when the recipient has to collect similar messages before it can initiate action.
- Parallel Wait State Pattern State Pattern to handle parallel operations in Real-time systems.
- Serial Wait State Pattern State Pattern to handle sequential operations in Real-time systems.

Hardware Interface Design Patterns

- Serial Port Design Pattern This design pattern is described in terms of a class that completely encapsulates the interface with a serial port device.
- High Speed Serial Port Design Pattern We consider the design of a DMA based high speed serial interface. The classes involved in this pattern interact with the device to setup buffers for DMA operations.
- Hardware Device Design Pattern Encapsulate the hardware device register access in a class.
- Synchronizer Design Pattern The Synchronizer Design Pattern is used to look at the raw incoming bit or byte stream and detect and align to the frame structure. The frame structure is detected by searching for a sync pattern in the frame.

Protocol Design Patterns

- Transmit Protocol Handler Design Pattern Sliding window transmit protocol design pattern is described here.
- Receive Protocol Handler Design Pattern Sliding window receive protocol design pattern is described here.
- Protocol Packet Design Pattern Simplify buffer management in protocol stacks by supporting a single buffer that allows addition and extraction of different protocol layers.
- Protocol Layer Design Pattern Provide a common framework for implementing different layers of a protocol stack.
- Protocol Stack Design Pattern Manages different layers of a protocol stack. Allows dynamic addition and removal of protocol layers.

Architecture Design Patterns

- Processor Architecture Patterns Typical processor patterns found in embedded and distributed systems are covered here
- Processor Architecture Patterns II Comparison of processor architecture patterns.
- Feature Coordination Patterns This article covers different design patterns for feature coordination.
- Task Design Patterns Typical design patterns in Embedded systems are compared here.
- Resource Allocation Patterns Resource allocation is a very important part of Embedded system design. Here we discuss important Resource allocation patterns.
- Timer Management Design Patterns Various Timer Management Design Patterns used in Real-time systems are covered in this article.

Implementation Design Patterns

- C++ Header File Include Patterns Header file management in complex Realtime projects can get very complicated. Here are some of the rules to simplify that.

- STL Design Patterns Here we put Standard Template Library (STL) to work with examples of typical design patterns using the STL map.
- STL Design Patterns II Queue Management and Resource Management design patterns are implemented using STL primitives.

Ques12. Explain Data flow graphs and control flow graphs

Ans. Control Flow Graph :

- A CFG is a graphical representation of a program unit.
- Three symbols are used to construct a control flow graph which includes a rectangle used to represent a sequential computation, a decision box labelled with T and F to represent True and False evaluations respectively and a merge point.
- A control flow graph is process oriented.
- It doesn’t manage or pass data between components.
- A control flow diagram illustrates how different programs, applications, services, or endpoints act on and process information to achieve certain ends within the context of a system.

Data Flow Graph :

- A data flow graph is a graphical representation of the "flow" of data through an information system.
- A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.
- A data flow graph is information oriented
- It passes data between other components.
- A data flow diagram illustrates how data flows from logical point to point in a system. Consider the following example which illustrates the difference between control flow graph and data flow graph