Module 4

1. What are the operational and non-operational qualities attributes of an embedded systems.

Operational Quality Attributes

The operational quality attributes pertain to the performance and functionality of embedded systems when they are in operation or 'online'. The following are the key attributes:

- **Response**
    - **Definition**: Measure of the system's quickness in tracking input changes.
    - **Importance**: Many embedded systems require fast, real-time responses.
        - **Example**: Flight control systems must respond in real time to ensure safety.
    - **Exceptions**: Not all systems need real-time responses.
        - **Example**: Electronic toys do not have critical response time requirements.
- **Throughput**
    - **Definition**: Efficiency of the system, measured as the rate of production or operation over time.
    - **Measurement**: Units of products, batches, etc.
        - **Example**: For a Card Reader, throughput is measured by the number of transactions it can perform in a given time.
    - **Benchmark**: Reference point for performance criteria. Used to compare products within the same line.
- **Reliability**
    - **Definition**: Measure of how dependable the system is.
    - **Metrics**:
        - **Mean Time Between Failures (MTBF)**: Frequency of failures.
        - **Mean Time to Repair (MTTR)**: Time allowed for the system to be out of order.
    - **Application**: Critical systems need high reliability with minimal downtime.
- **Maintainability**
    - **Definition**: Support and maintenance for technical issues and routine checkups. Higher reliability reduces the need for corrective maintenance.
    - **Types of Maintenance**:
        - **Scheduled/Periodic (Preventive)**: Regular checkups and replacements. Eg: Replacing printer ink cartridges.
        - **Corrective**: Addressing unexpected failures. Eg: Repairing the paper feeding part of a printer.
- **Security**
    - **Aspects**: Confidentiality, Integrity, Availability (distinct from availability in maintainability).
        - **Confidentiality**: Protecting data from unauthorized access.
        - **Integrity**: Ensuring data is not altered without authorization.
        - **Availability**: Ensuring authorized users have access to data.
        - **Example**: Personal Digital Assistants (PDAs). User profiles accessible via username and password. Some data may be read-only for users. Administrators have different access levels than regular users.
- **Safety**
    - **Definition**: Protection from potential damage to operators, the public, and the environment.
    - **Concerns**:
        - **Hardware or Firmware Failures**: Can lead to system breakdowns.
        - **Hazardous Emissions**: Potential for gradual or sudden damage.
    - **Safety Analysis**: Essential to evaluate and mitigate potential damages.

<u>Non-operational Quality Attributes</u>

These quality attributes pertain to aspects of the embedded system that are not directly related to its operational performance. The following are the key non-operational quality attributes:

- **Testability & Debug-ability**
  - **Testability**: Ease of testing the design, application, embedded hardware, and firmware.
    - **Hardware Testing**: Ensures peripherals and hardware function correctly.
    - **Firmware Testing**: Ensures firmware operates as expected.
  - **Debug-ability**: Ability to debug the product to identify sources of unexpected behavior.
    - **Hardware Debugging**: Identifies issues caused by hardware problems.
    - **Firmware Debugging**: Identifies errors in the firmware.
- **Evolvability**
  - **Definition**: Ease of modifying the embedded product to incorporate new firmware or hardware technologies.
  - **Context**: Similar to biological evolvability, which refers to non-heritable variation.
- **Portability**
  - **Definition**: Measure of 'system independence'.
  - **Characteristics**:
    - Functionality across various environments, processors, and operating systems.
    - Flexibility and ease of porting the product to new platforms.
  - **Porting Example**: Migrating firmware from one target processor to another (e.g., from Intel x86 to ARM Cortex M3).
    - **High-Level Language (e.g., C)**: Easier to port with minimal target-specific adjustments.
    - **Assembly Language**: Difficult to port due to processor-specific instructions.
  - **Comparison with Desktop Applications**:
    - **Microsoft Visual C++**: Runs only on Microsoft platforms.
    - **Java**: Runs on any operating system supporting Java standards.
- **Time to Prototype and Market**
  - **Definition**: Time between product conceptualization and readiness for selling or use.
  - **Importance**: Critical in a competitive market with rapid technology changes.
  - **Prototyping**: Informal rapid development of key product features.
    - **Benefits**: Reduces overall development time and time-to-market.
    - **Strategies**: Use off-the-shelf components and reusable assets to speed up prototyping.
- **Per Unit and Total Cost**
  - **Definition**: Costs monitored by end users and manufacturers.
  - **Commercial Sensitivity**: Pricing affects product success in the market.
  - **Manufacturer's Perspective**: Aim for marginal profit by balancing budget and system cost.
  - **Market Study**: Conduct cost-benefit analysis to determine optimal pricing.

2. Explain the fundamental issues in hardware software co-design.

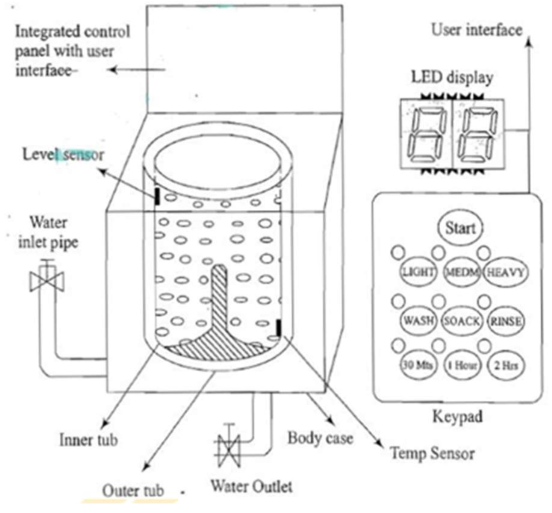
When addressing the problem statement of hardware-software co-design in real-life scenarios, several fundamental issues may arise. The key issues include:

- **Selecting the Model**
    - **Definition**: Models capture and describe system characteristics.
    - **Decision Making**: Difficult to choose the right model for each design phase.
        - **Example**: During specification, the focus is on system functionality. During implementation, focus shifts to system components and structure.
- **Selecting the Architecture**
    - **Definition**: Architecture specifies the number, types, and interconnection of system components.
    - **Types of Architectures**:
        - **Controller Architecture**: Implements the finite state machine model using a state register and combinational circuits.
        - **Datapath Architecture**: Implements the data flow graph model using registers, counters, memories, and ports.
        - **Finite State Machine Datapath (FSMD)**: Combines controller and datapath architectures.
        - **CISC (Complex Instruction Set Computing)**: Uses complex operations with single instructions, requiring additional silicon for microcode decoders.
        - **RISC (Reduced Instruction Set Computing)**: Uses simple operations with multiple instructions for complex operations, supporting extensive pipelining.
        - **VLIW (Very Long Instruction Word)**: Implements multiple functional units in the data path, packaging one instruction per unit.
        - **Parallel Processing**: Implements multiple concurrent processing elements (PEs) with local memory.
            - **SIMD (Single Instruction Multiple Data)**: Executes single instruction in parallel across PEs with a single controller.
            - **MIMD (Multiple Instruction Multiple Data)**: Executes different instructions concurrently across PEs, forming the basis of multiprocessor systems with shared memory or message passing.
- **Selecting the Language**
    - **Definition**: Programming languages capture computational models and map them into architecture.
    - **Language Choice**: No strict rules; multiple languages can capture models, and some languages are better suited for specific models.
        - **Example**: C++ is suitable for object-oriented models. Hardware implementations often use VHDL, SystemC, or Verilog.
- **Partitioning System Requirements into Hardware and Software**
    - **Decision Making**: Tough to decide whether to implement requirements in hardware or software.
    - **Trade-offs**: Various hardware-software trade-offs are used to make partitioning decisions.

These issues are crucial in the process of hardware-software co-design, affecting the overall system performance, flexibility, and efficiency.

3. **With the functional block diagram, explain the operation of Washing Machine as Application-Specific Embedded system.**

- **Washing Machine as an Embedded System**: A washing machine is a typical example of an embedded system used in home automation applications.
- **Components**: It contains sensors, actuators, a control unit, and application-specific user interfaces like keyboards and display units, some of which are visible while others are not.
- **Actuators**: The actuator part includes a motorized agitator, tumble tub, water drawing pump, and inlet valve to control water flow into the unit.
- **Sensors**: The sensor part consists of a water temperature sensor, level sensor, etc.
- **Control Unit**: The control part contains a microprocessor/controller-based board with interfaces to the sensors and actuators.
- **Feedback Mechanism**: Sensor data is fed back to the control unit, which generates the necessary actuator outputs.
- **User Interface**: The control unit connects to user interfaces like a keypad for setting washing time and selecting material types, with user feedback shown through display units and LEDs.
- **Washing Machine Models**: Washing machines come in two models, top loading and front loading.
- **Top Loading Models**: The agitator twists back and forth, pulling clothes down to the bottom of the tub, where they work their way back up.
- **Front Loading Models**: Clothes are tumbled and plunged into the water repeatedly.
- **Washing Phases**: The first phase involves washing the clothes, while the second phase, called the 'spin phase', uses centrifugal force to wring out more water by spinning the tub at high RPM.
- **Keyboard Panel**: The keyboard panel has buttons like Wash, Spin, and Rinse for configuring washing stages.
- **Spin Cycle**: The inner drum, with multiple holes, spins during the spin cycle, forcing water out through the holes into the stationary outer tub, from which it is drained.
- **Design Variations**: While designs vary among manufacturers, the general working principle remains the same.
- **Basic Controls**: These include a timer, cycle selector, water temperature selector, load size selector, and start button.
- **Mechanism**: The mechanism consists of a motor, transmission, clutch, pump, agitator, inner tub, outer tub, and water inlet valve, which connects to the home water supply.
- **Control Panel**: The integrated control panel includes a microprocessor/controller-based board with I/O interfaces and a control algorithm.
- **Input Interface**: The keyboard allows settings for wash type (Wash, Spin, Rinse), cloth type (Light, Medium, Heavy duty), and washing time.
- **Output Interface**: It includes LED/LCD displays and status indication LEDs connected to the controller's I/O bus.
- **Invisible Interfaces**: These include sensor interfaces (water temperature, water level) and actuator interfaces (motor control for agitator, tub movement control, and water flow control).



4. **Explain unique characteristics of embedded systems.**

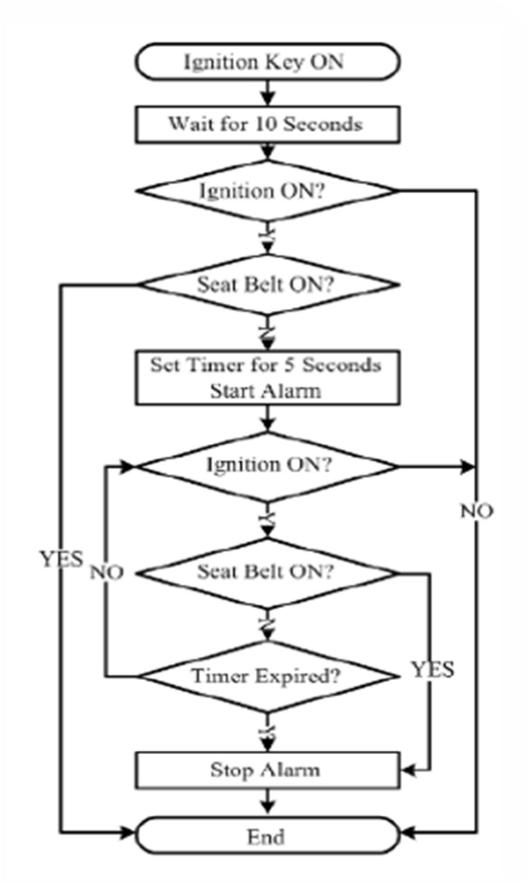
- **Application and Domain Specific**:
  - Embedded systems perform specific functions and cannot be used for other purposes.
  - For example, the embedded control unit of a microwave oven cannot replace that of an air conditioner.
- **Reactive and Real-Time**:
  - Constantly interact with the real world through sensors and user-defined input devices.
  - Systems capture real-time events and react to maintain controlled output.
  - They are reactive systems and may need to operate in real-time, especially for mission-critical applications like flight control systems and Antilock Brake Systems (ABS).
- **Operates in Harsh Environments**:
  - Embedded systems may be deployed in dusty, high-temperature, or high-vibration areas.
  - Design must consider operating conditions, component durability, power supply fluctuations, corrosion, and component aging.
- **Distributed**:
  - Embedded systems can be part of larger systems, forming a single large control unit.
  - Examples include vending machines, ATMs, and SCADA systems in control and instrumentation applications.
- **Small Size and Weight**:
  - Compactness and low weight are significant factors in product aesthetics and convenience.
  - Many applications demand small, lightweight embedded systems.
- **Power Concerns**:
  - Design should minimize heat dissipation to avoid the need for bulky cooling solutions.
  - Use ultra-low-power components like low dropout regulators and processors with power-saving modes.
  - Critical for battery-operated applications where power consumption affects battery life.

5. **What is sequential processing model? Draw a sequential processing model for car seat belt warning system using flow chart.**

Sequential Program Model: In the sequential programming Model, the functions or processing requirements are executed in sequence. It is same as the conventional procedural programming.

- Here the program instructions are iterated and executed conditionally, and the data gets transformed through a series of operations.
- FSMs are good choice for sequential program modeling.
- Another important tool used for modeling sequential program is Flow Charts.
- The FSM approach represents the states, events, transitions and actions, whereas the Flow Chart models the execution flow.
- The execution of functions in a sequential program model for the 'Seat Belt Warning' system is illustrated below.

```
#define ON 1
#define OFF 0
void seat_belt_warn() {
      wait_10sec();
      if (check_ignition_key()==ON) {
            if (check_seat_belt()==OFF) {
                  set_timer(5);
                  start_alarm();
                  while((check_seat_belt()==OFF)&&(check_ignition_key()==OFF)&&
                  (timer_expire()==ON))
                        stop_alarm();
            }
      }
}
```

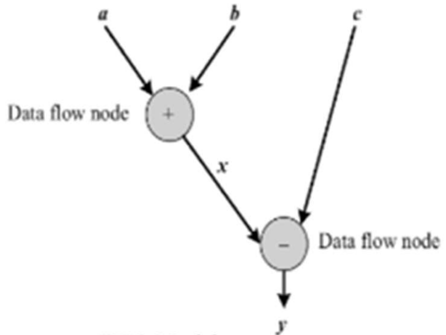


6. Explain Data flow graph and control data flow graph computational model with neat diagram.

Data Flow Graph/ Diagram (DFG) Model: The Data Flow Graph (DFG) model translates data processing requirements into a visual graph where program execution is determined by data flow. In this model, operations on the data (processes) are represented by circles, and data flow is depicted with arrows. An inward arrow to a process represents input data, while an outward arrow represents output data. This model is particularly suited for computationally intensive, data-driven embedded applications.

For example, for computations like x = a + b and y = x - c, the DFG model visually represents these operations with processes and data flow arrows. A data path in a DFG model is the flow from input to output.

A DFG model is said to be acyclic (ADFG) if it doesn't have multiple values for input variables and output values for given inputs. Non-acyclic inputs include feedback loops where output is fed back to input. The DFG model translates the program into a single sequential process execution.

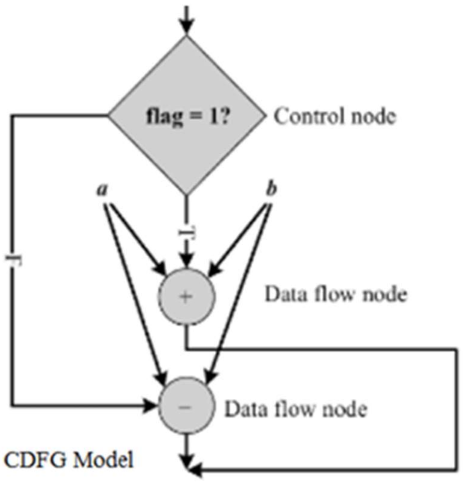


DFG Model

Control Data Flow Graph/ Diagram (CDFG) Model: The Control DFG (CDFG) model is used for applications involving conditional program execution, incorporating both data and control operations. Unlike the Data Flow Graph (DFG) model, which is solely data-driven and lacks conditionals, the CDFG contains both data flow nodes and decision nodes.

In the CDFG model, a decision-making process is represented by a 'Diamond' block, similar to decision elements in flow charts. For instance, the requirement "If flag = 1, x = a + b; else y = a - b" involves a control node that determines which process to execute.

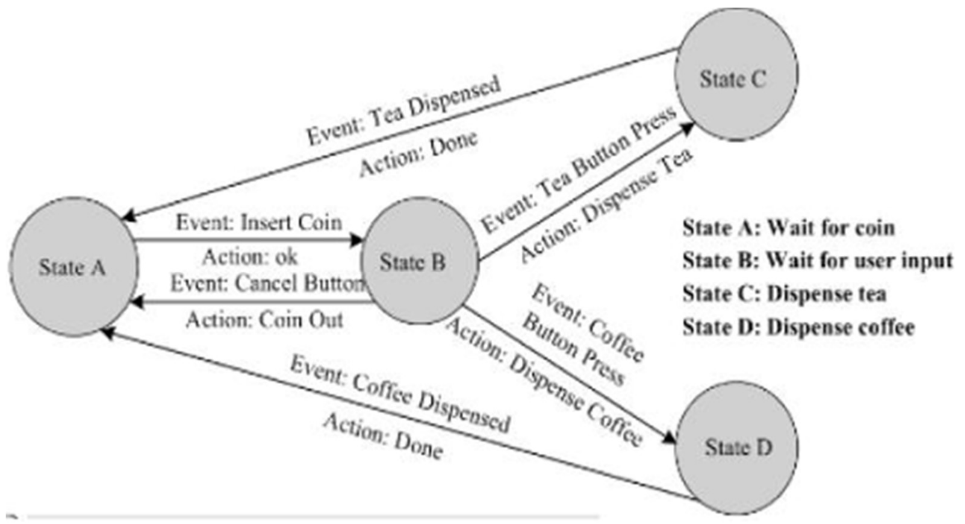
A practical example of the CDFG model is in digital still cameras. The process of capturing and saving images is data-driven, with operations like analog-to-digital conversion and media processing for auto-correction and white balance. The decision on the image format (e.g., JPEG, TIFF, BMP) is controlled by the camera settings configured by the user.



CDFG Model

7. **Design an FSM model for Tea/Coffee vending machine.**

- 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 the following Figure.
- The FSM representation 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'.
- 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' or 'Dispense Coffee' respectively.
- Once the coffee/ tea vending is over, the respective states transitions back to the 'Wait for Coin' state.



Module 5

1. Explain the process of choosing an RTOS.

The decision of choosing an RTOS for an embedded design is very crucial. A lot of factors need to be analyzed carefully before deciding on the selection of an RTOS. These factors can be either functional or non-functional.

Functional Requirements:

- **Processor Support:** Ensure the RTOS supports the required processor architecture.
- **Memory Requirements:** Evaluate the minimal ROM and RAM requirements.
- **Real-time Capabilities:** Analyze task/process scheduling policies and real-time standards met by the OS.
- **Kernel and Interrupt Latency:** Minimal interrupt latency is crucial for systems with high response requirements.
- **Inter Process Communication and Task Synchronization:** Availability and implementation of IPC and synchronization mechanisms, including priority inversion handling.
- **Modularization Support:** Ability to choose and compile essential modules to fit the embedded product's needs.
- **Support for Networking and Communication:** Ensure the OS provides stack implementation and driver support for necessary communication interfaces.
- **Development Language Support:** Availability of runtime libraries and virtual machines for languages like Java and C#.

Non-functional Requirements:

- **Custom Developed or Off the Shelf:** Decide between a custom-built OS or an off-the-shelf (commercial or open source) OS based on cost, licensing, development time, and resource availability.
- **Cost:** Evaluate the total cost of development, purchase, and maintenance.
- **Development and Debugging Tools Availability:** Ensure the availability of necessary development and debugging tools.
- **Ease of Use:** Consider the user-friendliness of the commercial RTOS.
- **After Sales Support:** Analyze the availability and quality of after-sales support for bug fixes, critical updates, and production issues.

2. Explain the working of target hardware debugging

Hardware debugging is not similar to firmware debugging. Hardware debugging involves the monitoring of various signals of the target board (address/ data lines, port pins, etc.), checking the inter connection among various components, circuit continuity checking, etc. The various hardware debugging tools used in Embedded Product Development are explained below.

- **Magnifying Glass (Lens):**
  - Used for visual inspection of the target board.
  - Helps identify dry soldering, missing components, improper placement, soldering issues, and track damage.
- **Multimeter:**
  - Measures electrical quantities: voltage (AC and DC), current (AC and DC), resistance, capacitance, and continuity.
  - Essential for physical contact-based debugging and initial hardware diagnostics.
- **Digital Cathode Ray Oscilloscope (CRO):**
  - Captures and analyzes waveforms and signal strength.
  - Useful for analyzing interference noise, power supply lines, and crystal oscillator signals.
  - Digital versions offer high frequency support, advanced waveform recording, and measurement features.
- **Logic Analyzer:**
  - Captures digital data (logic 1 and 0) from digital circuitry.
  - Measures states of port pins, address bus, and data bus.
  - Provides detailed insights into firmware behavior by capturing address and data line logic.
- **Function Generator:**
  - Simulates periodic waveforms (sine, square, saw-tooth) with various frequencies and amplitudes.
  - Used to provide required input signals to the target board during debugging.

3. Show the working of Emulators, Simulator and Debugging

<u>Simulators</u>

Working:

- **Purpose:** Simulators provide a software-based representation of the target hardware. They allow developers to test and debug firmware without needing the actual hardware.
- **Operation:**
  - **Modeling:** The simulator models the CPU and peripherals of the target hardware in software. This model replicates the functionality of the actual hardware.
  - **Firmware Execution:** Developers load the firmware into the simulator. The simulator executes the firmware as if it were running on the real hardware.
  - **Interface:** Simulators typically provide a graphical user interface (GUI) that allows developers to interact with and test the firmware, including I/O operations and UI elements.
  - **Debugging:** Debugging features may include setting breakpoints, stepping through code, and inspecting memory and register states.

Advantages:

- **No Hardware Required:** Firmware development can start without the actual hardware.
- **Simulate Peripherals:** Allows simulation of I/O peripherals and manipulation of I/O values.

Limitations:

- **Deviates from Real Behavior:** Simulated environments might not perfectly replicate real hardware conditions.
- **Lack of Real-Time Behavior:** Simulators cannot fully replicate real-time constraints and variations in hardware behavior.

<u>Emulators</u>

Working:

- **Purpose:** Emulators are hardware devices that replicate the functionality of the target CPU and enable real-time debugging of firmware on the actual hardware.
- **Operation:**
    - **Emulation Device:** Contains hardware that mimics the target CPU's functionality. It receives signals from the target board and executes firmware under debug control.
    - **Emulation Memory:** The emulator has its own RAM that replaces the target board's ROM. This allows for dynamic code updates and avoids the need for ROM burning.
    - **Control Logic:** Implements advanced debugging features such as hardware breakpoints, trace buffers, and logic analyzer functions.
    - **Device Adapters:** Connect the emulator to the target board, providing physical and electrical connections for communication and signal routing.

Advantages:

- **Real-Time Debugging:** Allows debugging of firmware in a real hardware environment.
- **ROM Emulation:** Facilitates code testing and modification without physical ROM changes.
- **Advanced Features:** Supports complex debugging features like hardware breakpoints and trace analysis.

Limitations:

- **Cost:** Emulators can be expensive due to their advanced hardware and capabilities.
- **Setup Complexity:** Requires proper setup and calibration with the target hardware.

<u>Debugging</u>

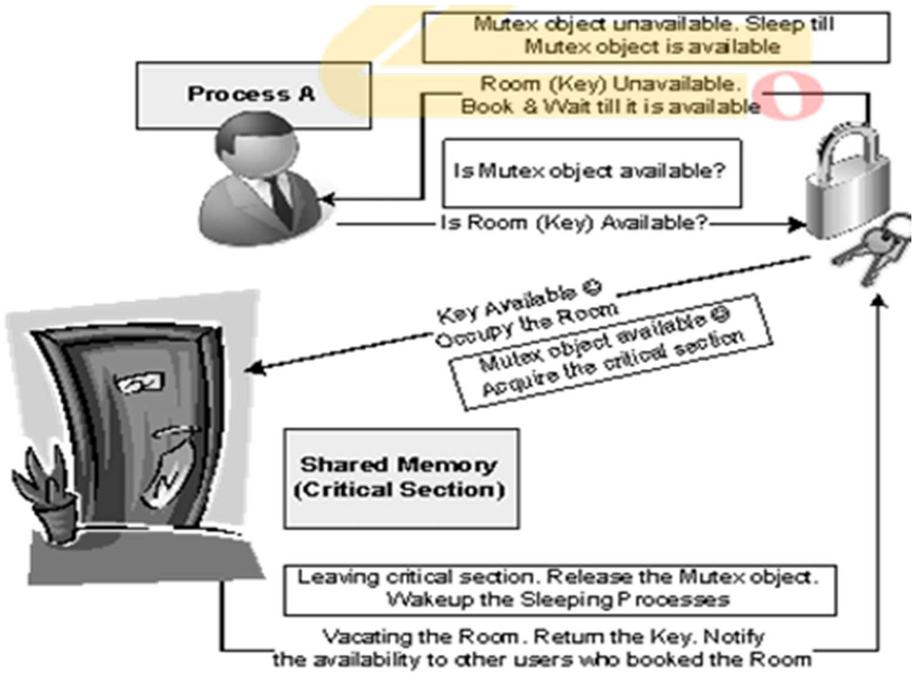
Types and Techniques:

- **Incremental EEPROM Burning:**
    - **Process:** Code is divided into functional units and burned into EEPROM incrementally. This approach simplifies debugging by isolating functional blocks.
    - **Use Case:** Early-stage firmware development where the complete firmware isn't ready.
- **Inline Breakpoint Debugging:**
    - **Process:** Involves inserting printf() statements or similar debug code into the firmware. This code helps in tracking execution flow and verifying behavior at specific points.
    - **Use Case:** Simple debugging to ensure code execution reaches desired points.
- **Monitor Program Based Debugging:**
    - **Process:** A monitor program runs on the target hardware to control firmware download, register/memory inspection, and single stepping. Communication typically happens via a serial interface.
    - **Features:** Includes command set interface, firmware download, register inspection, and debug information transfer.
- **In-Circuit Emulator (ICE) Based Debugging:**
    - **Process:** Uses an emulator hardware device to interface with the target board. The emulator replicates the target CPU and facilitates real-time debugging.
    - **Components:** Emulation device, memory, control logic, and device adapters.
- **On-Chip Debugging (OCD):**
    - **Process:** Utilizes built-in debug modules within the processor/controller for efficient debugging. Features vary by chip vendor and may include proprietary technologies like Background Debug Mode (BDM) or OnCE.
    - **Advantages:** Provides integrated, efficient debugging support directly within the chip.

4. Explain the concept of Binary Semaphore.

Binary Semaphore: Implements exclusive access to shared resource by allocating the resource to a single process at a time and not allowing the other processes to access it when it is being used by a process.

- Only one process/ thread can own the binary semaphore at a time.
- The state of a binary semaphore object is set to signaled when it is not owned by any process/ thread and set to non-signaled when it is owned by any process/ thread.
- The implementation of binary semaphore is OS kernel dependent. Under certain OS kernel it is referred as mutex.



Mutex object unavailable. Sleep till Mutex object is available
Process A
Room (Key) Unavailable. Book & Wait till it is available
Is Mutex object available?
Is Room (Key) Available?
Key Available ☺ Occupy the Room
Mutex object available ☺ Acquire the critical section
Shared Memory (Critical Section)
Leaving critical section. Release the Mutex object. Wakeup the Sleeping Processes
Vacating the Room. Return the Key. Notify the availability to other users who booked the Room

The concept of Binary Semaphore

5. Explain the role of Integrated Development Environment (IDE) for Embedded Software development.

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

- Text Editor (Source Code Editor)
- Cross-complier (for cross platform development and complier for same platform development)
- Linker
- 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.