MODULE 3: Hardware Software Co design and Program Modelling

Structure

3.1 Hardware Software Co design and Program Modelling : Fundamental issues in Hardware Software Co-design,

3.2 Computational models in Embedded System Design (Chapter 7 – Text 1: 7.1, 7.2)

3.3 Embedded Hardware Design and Development: Analog Electronic Components,

3.4 Embedded Hardware Design and Development :Digital Electronic Components,

3.5 VLSI & Integrated Circuit Design,

3.6 Electronic Design Automation Tools

Objectives

Learning Objectives

1. **Understand Hardware-Software Co-Design**: Analyze key issues and architectural trade-offs.
2. **Explore Computational Models**: Familiarize with models such as FSM, DFG, and CDFG.
3. **Understand Embedded Hardware Design**: Study the role of analog and digital components.
4. **Learn VLSI Design Concepts**: Understand IC types, design steps, and methodologies.
5. **Introduction to EDA Tools**: Study modern tools like OrCAD, Eagle, and Prote

3.1 Hardware Software Co design and Program Modelling

Fundamental Issues in Hardware-Software Co-Design

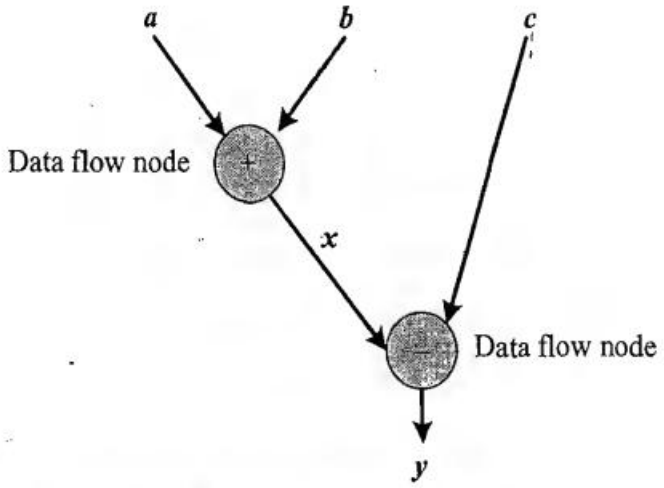
1. **Selecting the Model**
    - Models describe **system characteristics** and vary at different design stages.
    - **Specification Stage** → Focus on functionality, not implementation.
    - **Implementation Stage** → Switch to models capturing system structure.
2. **Selecting the Architecture**
    - Defines **system implementation** in terms of components & interconnections.
    - Common architectures:
        - **Controller Architecture** – Implements **finite state machine (FSM)**.
        - **Datapath Architecture** – Implements **data flow graph (DFG)**.
        - **FSMD Architecture** – Combines **controller + datapath**.
        - **CISC Architecture** – Complex instruction set, reduces memory access.
        - **RISC Architecture** – Simple instruction set, supports pipelining.
        - **VLIW Architecture** – Multiple functional units execute **parallel instructions**.
        - **SIMD (Single Instruction Multiple Data)** – Parallel execution using one controller.
        - **MIMD (Multiple Instruction Multiple Data)** – Multiple processors execute different instructions.
3. **Selecting the Language**
    - Programming languages map **computational models** into architecture.
    - **Software Languages:** C, C++, Java (object-oriented models).
    - **Hardware Languages:** VHDL, System C, Verilog (hardware description).
4. **Partitioning System Requirements**

- Deciding **which functions** should be implemented in **hardware vs. software**.
- Uses **hardware-software trade-offs** for optimal performance.

3.2 Computational models in Embedded System Design (Chapter 7 – Text 1: 7.1, 7.2)

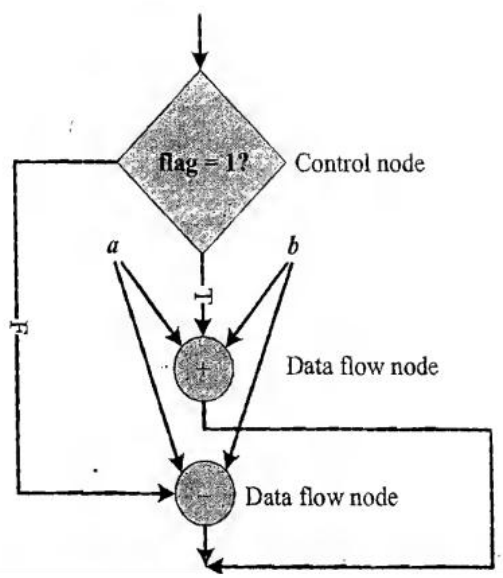
Computational Models in Embedded System Design

1. **Data Flow Graph (DFG) Model**
   - **Data-driven model**, where execution is controlled by data flow.
   - **Processes (operations) represented by circles**, arrows indicate data movement.
   - Used in **DSP applications** and **computational-intensive tasks**.
   - **Acyclic DFG (ADFG)**: No multiple values for inputs/outputs, no feedback loops.



2. **Control Data Flow Graph (CDFG) Model**
   - Extends **DFG model** by incorporating **control operations (conditionals)**.
   - Uses **decision nodes (diamonds)** to represent conditional execution.
   - Example: **Image processing in a digital camera**, where the user selects **JPEG, TIFF, BMP** format.

$$\text{If flag} = 1, x = a + b; \text{ else } y = a - b;$$



3. **State Machine Model (FSM)**
    - **Models reactive/event-driven systems** (e.g., industrial control, automotive applications).
    - System is represented by:
        - **States** (e.g., "Alarm Off", "Waiting", "Alarm On").
        - **Events** (e.g., "Ignition ON", "Seat Belt ON", "Timer Expire").
        - **Actions & Transitions** (state changes triggered by events).
    - Example: **Seat Belt Warning System in Automobiles**
        - If ignition is **ON** and the seatbelt is not fastened within **10 sec**, an **alarm sounds for 5 sec**.
        - **Alarm stops** when the belt is fastened, ignition is off, or time expires.
4. **Timer State Machine (FSM Model)**
    - Timer operates in **three states**:
        - **IDLE**: When timer is not running.
        - **READY**: When the timer is loaded with time delay value.
        - **RUNNING**: Timer counts down until expiration or a stop event occurs.

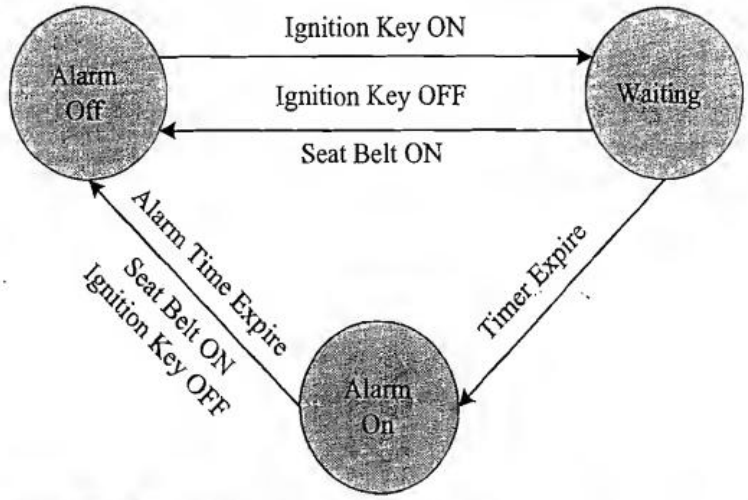


Fig. 7.3 FSM Model for Automatic seat belt warning system

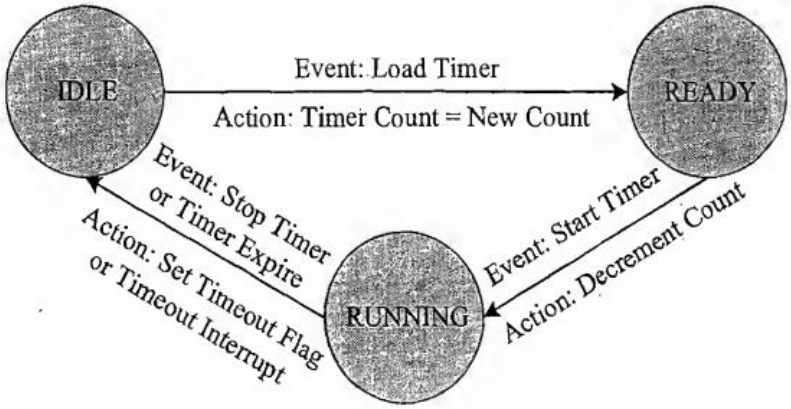


Fig. 7.4 FSM Model for timer

Example-1

Question

Design an Automatic Tea/Coffee Vending Machine using FSM Model

A **tea/coffee vending machine** operates based on a **Finite State Machine (FSM) model** with the following requirements:

1. The machine starts in **'Wait for Coin'** state.
2. The user **inserts a ₹5 coin** to initiate the vending process.
3. After inserting the coin, the system transitions to **'Wait for User Input'** state.
4. The user can select:
    - **'Tea'** → Transitions to **'Dispense Tea'** state.
    - **'Coffee'** → Transitions to **'Dispense Coffee'** state.

- **'Cancel'** → Coin is returned, and state transitions back to **'Wait for Coin'**.

5. Once the drink is dispensed, the system resets back to **'Wait for Coin'**.
6. Additional conditions may include:
   - **Timeout in 'Wait for User Input' state** (if no input, coin is returned).
   - **Error Handling** for **'Water Not Available'** or **'Mix Not Available'**.

Solution – FSM Representation

States of the FSM Model:

1. **Wait for Coin** → Initial state, waiting for a ₹5 coin.
2. **Wait for User Input** → Waiting for selection of **Tea, Coffee, or Cancel**.
3. **Dispense Tea** → Vends tea, then returns to **Wait for Coin**.
4. **Dispense Coffee** → Vends coffee, then returns to **Wait for Coin**.
5. **Error State (Optional)** → Handles **no water/mix availability** issues.

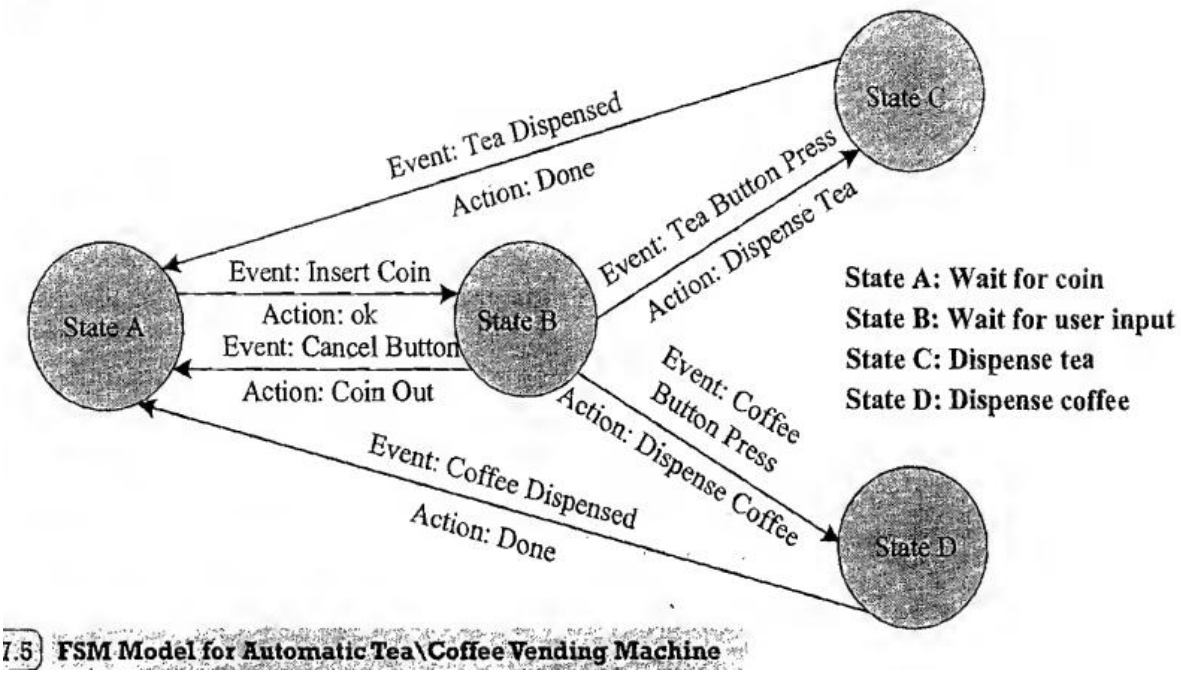
FSM State Transitions:

Current State	Event (Input)	Next State	Action
Wait for Coin	Insert ₹5 coin	Wait for User Input	Accept Coin
Wait for User Input	Press 'Tea'	Dispense Tea	Start Tea Dispensing
Wait for User Input	Press 'Coffee'	Dispense Coffee	Start Coffee Dispensing
Wait for User Input	Press 'Cancel'	Wait for Coin	Return Coin
Dispense Tea	Tea Dispensed	Wait for Coin	Reset System
Dispense Coffee	Coffee Dispensed	Wait for Coin	Reset System
Wait for User Input	Timeout (No Selection)	Wait for Coin	Return Coin
Any State	Error (No Water/Mix)	Error State	Display Error
Error State	Problem Resolved	Wait for Coin	Reset System

Enhancements:

- **Timeout Mechanism**: If the user doesn't select within a few seconds, the coin is returned.

- **Error Handling**: If water or tea/coffee mix is unavailable, the system enters an **Error State**.



7.5 **FSM Model for Automatic Tea\Coffee Vending Machine**

Example-2

Question

Design a Coin-Operated Public Telephone Unit using FSM Model

A **coin-operated public telephone** system is modeled using **Finite State Machine (FSM)** with the following requirements:

1. **Call initiation** starts when the receiver is **lifted (off-hook)**.
2. The user must **insert a ₹1 coin** to proceed with the call.
3. If the **line is busy**, the coin is returned when the receiver is placed **back on hook (on-hook)**.
4. If the **line is connected**, the user can talk for **60 seconds**.
5. At the **45th second**, the system prompts the user to **insert another ₹1 coin** to continue the call.
6. If no additional coin is inserted, the call **terminates after 60 seconds**.
7. The system **resets** when the receiver is placed back **on-hook**.
8. If there is a **line fault**, the system enters an **'Out of Order'** state.

Solution – FSM Representation

FSM States:

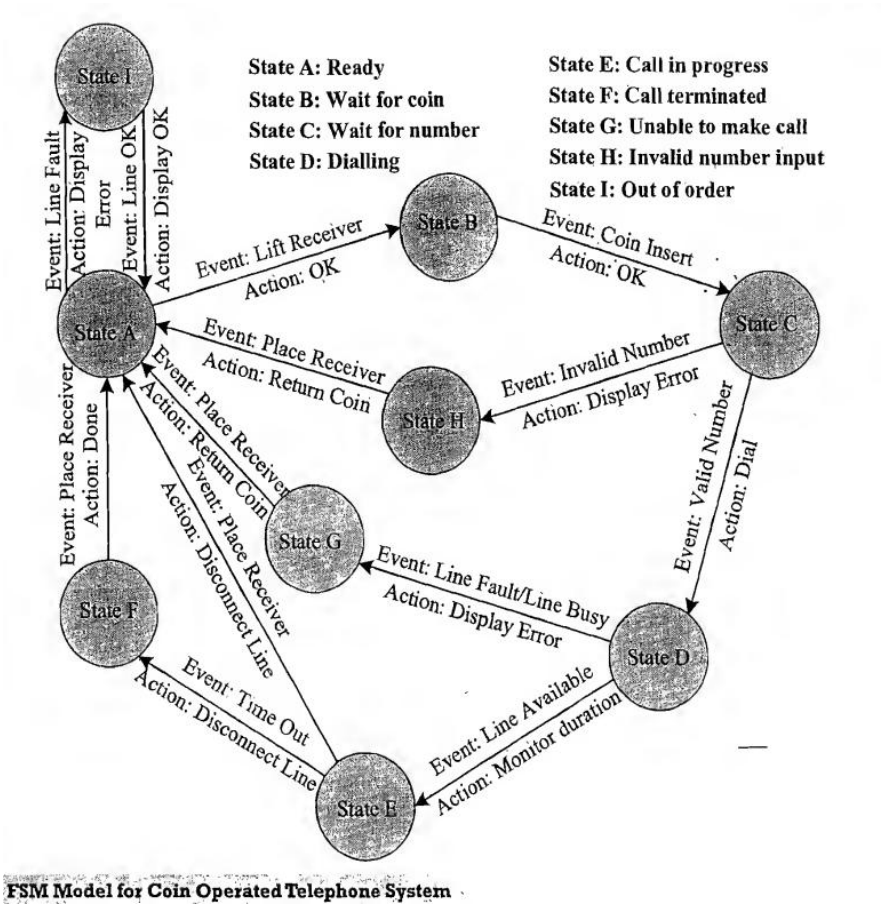
1. **Idle (Wait for Off-Hook)** → Initial state, waiting for the receiver to be lifted.
2. **Wait for Coin** → User must insert a ₹1 coin.
3. **Check Line Status** → System checks if the line is **busy or connected**.
4. **Busy Signal** → If the line is busy, the **coin is returned**, and the system resets.
5. **Call in Progress** → User is allowed to talk for **60 seconds**.
6. **Prompt for Coin** → At **45 seconds**, a prompt for another **₹1 coin** is given.
7. **Timeout/Call Termination** → If no coin is inserted, the call ends after **60 seconds**.
8. **Out of Order** → If a **line fault occurs**, the system stops working.

FSM State Transitions:

Current State	Event (Input)	Next State	Action
Idle (On-Hook)	Receiver Lifted	Wait for Coin	Wait for user input
Wait for Coin	Insert ₹1 Coin	Check Line Status	Accept Coin
Check Line	Line Busy	Busy Signal	Return Coin
Check Line	Line Connected	Call in Progress	Start 60-sec timer
Call in Progress	Timer = 45 sec	Prompt for Coin	Notify user
Prompt for Coin	Insert ₹1 Coin	Call in Progress	Extend call
Prompt for Coin	No Coin Inserted	Timeout/Terminate	End Call
Call Terminated	Receiver Placed On-Hook	Idle (Wait for Off-Hook)	Reset System
Any State	Line Fault Detected	Out of Order	Disable System
Out of Order	Line Fixed	Idle (Wait for Off-Hook)	Reset System

Enhancements:

- **Timeout Mechanism**: If the user doesn't insert a coin within a few seconds after off-hook, the receiver resets.
- **Multiple Coin Handling**: If a user inserts an **incorrect coin**, the system returns it.
- **Concurrent Process Handling (HCFSM)**: If required, a **Hierarchical/Concurrent FSM** model can be used for simultaneous state management (e.g., handling timeouts and user inputs together).



FSM Model for Coin Operated Telephone System

Sequential Program Model

1. **Definition**
    - A **sequential program model** executes **functions or processing tasks in a defined sequence**.
    - Similar to **procedural programming**, where instructions run **iteratively** or **conditionally**.
2. **Implementation Methods**
    - **Finite State Machine (FSM)**: Models execution using **states, events, transitions, and actions**.

- **Flow Charts**: Visually represent **execution flow** in sequential steps.

3. **Example – Seat Belt Warning System**
   - **Step 1:** Detect if ignition is **ON**.
   - **Step 2:** Check if seatbelt is **fastened** within **10 seconds**.
   - **Step 3:** If seatbelt **not fastened**, trigger **alarm for 5 seconds**.
   - **Step 4:** Alarm **turns OFF** if seatbelt is fastened, ignition is OFF, or time expires.

C program for the **Seat Belt Warning System** based on the flowchart:

```
#include <stdio.h>
int main() {
    int ignitionOn, seatBeltOn;
    printf("Ignition Key ON? (1 for YES, 0 for NO): ");
    scanf("%d", &ignitionOn);
    if (!ignitionOn) {
        printf("Ignition OFF. Exiting system.\n");
        return 0;
    }
    printf("Waiting for 10 seconds...\n");
    printf("Seat Belt ON? (1 for YES, 0 for NO): ");
    scanf("%d", &seatBeltOn);
    if (seatBeltOn) {
        printf("Seat belt fastened. System secure.\n");
        return 0;
    }
    printf("Starting alarm for 5 seconds...\n");
    printf("Seat Belt ON or Ignition OFF? (1 for YES, 0 for NO): ");
    scanf("%d", &seatBeltOn);
```

```
    if (seatBeltOn) {
        printf("Alarm stopped. Exiting.\n");
    } else {
        printf("Alarm timer expired. Stopping alarm.\n");
    }
    return 0;
}
```

Explanation:

- The program checks ignition and seat belt status in a few simple conditional blocks.
- It waits briefly, simulating real-time delays implicitly.
- Alarms and conditions (like seat belt fastened or ignition turned off) are handled succinctly.

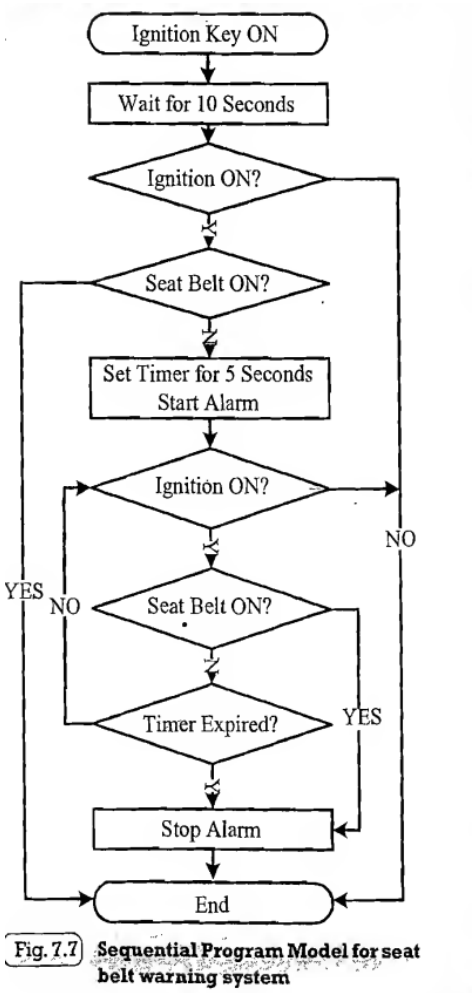


Fig. 7.7 Sequential Program Model for seat belt warning system

Concurrent/Communicating Process Model

Overview:

- The concurrent process model executes tasks/processes simultaneously, enhancing CPU utilization.
- Useful for tasks involving I/O waiting or sleep durations, unlike sequential models that cause poor utilization.
- Requires task scheduling, synchronization, and communication mechanisms.

Example: Seat Belt Warning System (Concurrent Model)

Tasks divided into:

1. **Wait Timer Task**: Waits for 10 seconds.
2. **Ignition Key Monitoring Task**: Monitors ignition status (ON/OFF).
3. **Seat Belt Monitoring Task**: Monitors seat belt status (ON/OFF).
4. **Alarm Control Task**: Starts/stops the alarm based on conditions.
5. **Alarm Timer Task**: Waits for 5 seconds after alarm activation.

Event-Based Synchronization:

- **Wait Timer Expire Event**: Set when the 10-second timer ends.
- **Ignition Events**:
  - ignition_on (Set when ignition is ON).
  - ignition_off (Set when ignition is OFF).
- **Seat Belt Events**:
  - seat_belt_on (Set when seat belt is ON).
  - seat_belt_off (Set when seat belt is OFF).
- **Alarm Timer Events**:
  - alarm_timer_start (Set when alarm starts).
  - alarm_timer_expire (Set when 5-second alarm ends).

Execution Flow:

- **Alarm Control Task**: Activates alarm only if:
  - Wait timer expires.
  - Ignition is ON.
  - Seat belt is OFF.
- Waits for events:
  - **Alarm Timer Expire**: Ends after 5 seconds.
  - **Ignition Off or Seat Belt On**: Stops alarm immediately.

Uses in Real-Time Systems:

- Synchronization techniques like shared memory, message passing, and events coordinate tasks.
- Enhances real-time system modeling by handling multiple concurrent tasks.

Object-Oriented Model (Brief Overview)

- **Definition**:
  - A system design approach that breaks complex requirements into simple, reusable components called **objects**.
- **Key Features**:

1. **Object**: Represents a specific part of the system with unique behavior and state.
2. **Class**: Abstract blueprint of objects, defining:
   - **State**: Member variables.
   - **Behavior**: Member functions.

- **Access Modifiers**:
  - **Private**: Accessible only within the class.
  - **Public**: Accessible both inside and outside the class.
  - **Protected**: Accessible to the class and derived classes.
- **Benefits**:
  - **Reusability**: Simplifies complex designs by reusing objects.
  - **Maintainability**: Easier updates and modifications.
  - **Productivity**: Streamlined system design with abstraction, encapsulation, and protection.

This flowchart represents the **Concurrent/Communicating Process Model** for implementing a **Seat Belt Warning System** in an embedded system. It illustrates how multiple tasks are executed concurrently and communicate through events to achieve the desired system behavior.

Key Components of the Flowchart:

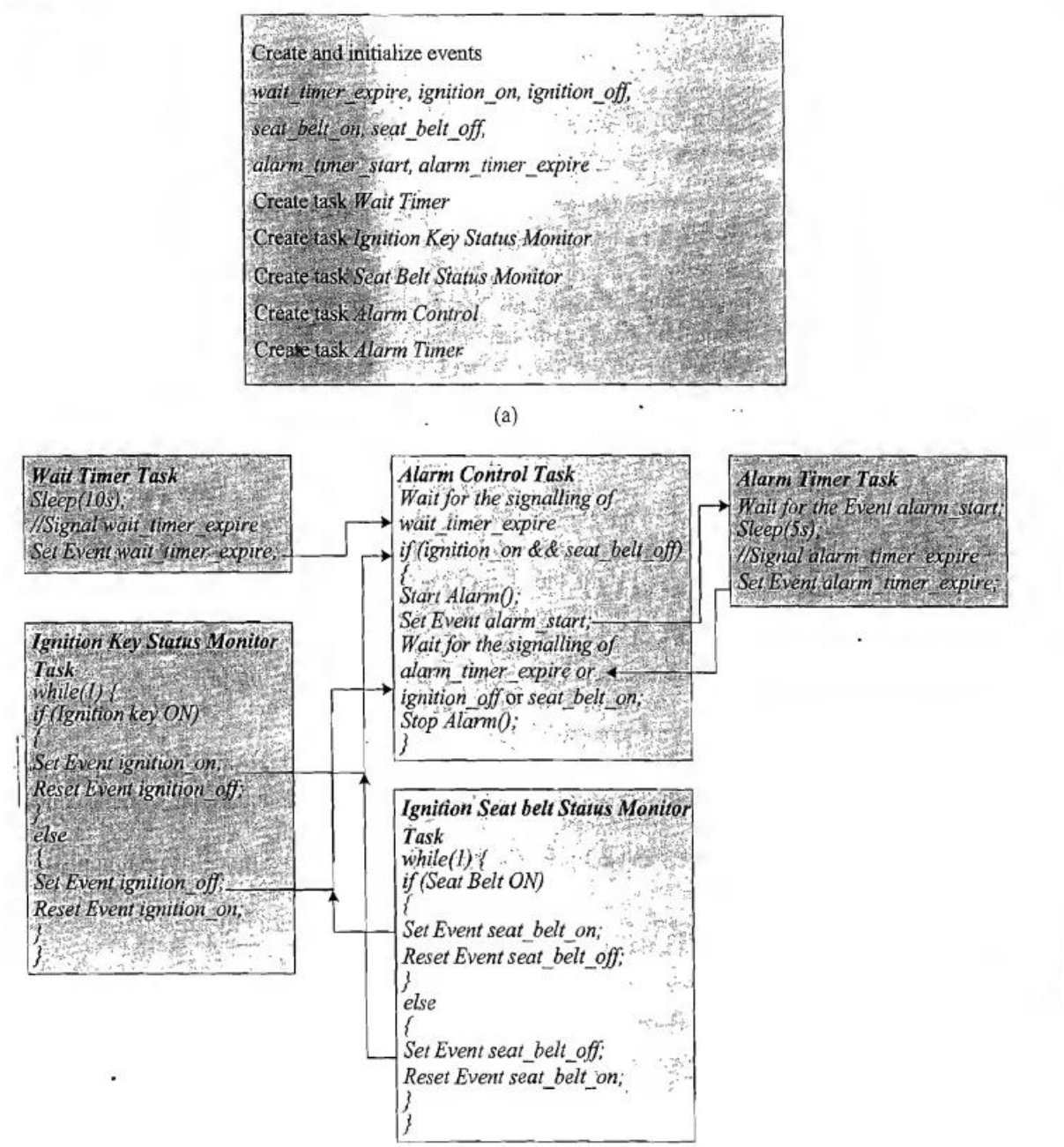
1. **Initialization**:
   - Events such as wait_timer_expire, ignition_on, ignition_off, seat_belt_on, and seat_belt_off are created and initialized.
   - Tasks like Wait Timer, Ignition Key Status Monitor, Seat Belt Status Monitor, Alarm Control, and Alarm Timer are created.
2. **Tasks**:
   - **Wait Timer Task**:
     - Waits for 10 seconds and signals the wait_timer_expire event.
   - **Ignition Key Status Monitor Task**:
     - Continuously monitors the ignition key status (ON or OFF) and sets or resets corresponding events (ignition_on or ignition_off).
   - **Seat Belt Status Monitor Task**:
     - Monitors if the seat belt is fastened or not and sets/resets events (seat_belt_on or seat_belt_off).
   - **Alarm Control Task**:
     - Waits for wait_timer_expire and checks if the ignition is ON and the seat belt is OFF.
     - Starts the alarm and waits for alarm_timer_expire or events like ignition_off or seat_belt_on to stop the alarm.

- **Alarm Timer Task**:
  - Waits for 5 seconds and signals the alarm_timer_expire event.

Purpose:

This model demonstrates:

- **Concurrent Task Execution**: Tasks like monitoring ignition status, seat belt status, and managing the alarm run independently but coordinate through events.
- **Task Synchronization and Communication**: Tasks communicate via events to ensure logical system behavior, such as triggering alarms only under specific conditions.



3.3 Embedded Hardware Design and Development: Analog Electronic Components

Summary of Embedded System Hardware Components

Analog Electronic Components

1. **Resistors**:
   - Limit current in circuits.
   - Example: Current-limiting resistors for LEDs and buzzers.
2. **Capacitors**:
   - Used for signal filtering, reset circuits, RF matching, and power supply decoupling.
   - Types: Electrolytic, ceramic, tantalum.
3. **Inductors**:
   - Filter power supply ripples and noise.
   - Commonly used in matching circuits and filters.
4. **Diodes**:
   - Types:
     - **Schottky Diode**: Low voltage drop, fast switching.
     - **Zener Diode**: Allows reverse current flow above breakdown voltage, used for voltage clamping.
   - Applications: Reverse polarity protection, voltage rectification, freewheeling, etc.
5. **Transistors**:
   - Applications: Switching (ON/OFF) or amplification.
   - Common emitter NPN transistor configuration is widely used in driver circuits for relays, motors, etc.

3.4 Embedded Hardware Design and Development :Digital Electronic Components,

Digital Electronic Components

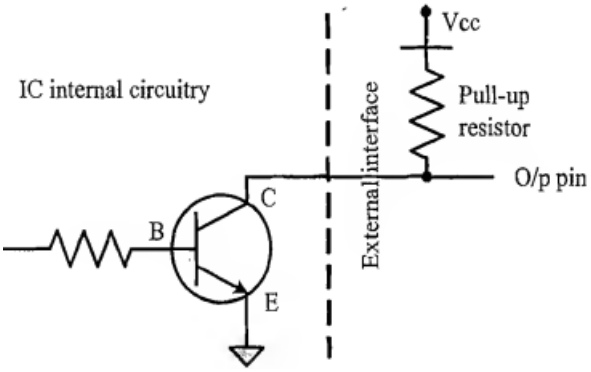
1. **Digital Circuits**:
   - Used for processing discrete digital signals in microcontrollers, microprocessors, and SoCs.
   - Examples: Address decoders, latches, encoders/decoders.
2. **Glue Logic**:
   - Custom circuits for interfacing incompatible ICs.
3. **Standards**:
   - **Transistor-Transistor Logic (TTL)** and **CMOS Logic** describe the electrical characteristics of digital signals.

This combination of analog and digital components forms the backbone of embedded systems, providing the foundation for diverse applications.

Brief Summary of Key Concepts

Open Collector and Tri-State Output

1. **Open Collector:**
    - Used for interfacing devices operating at different voltage levels.
    - Requires a pull-up resistor to define the output state.
    - Supports multi-drop connections (e.g., I2C) and enables "Wired AND/OR" configurations.



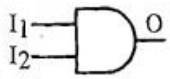
2. **Tri-State Output:**
    - Adds a third state, "High Impedance," to Logic 0 and Logic 1.
    - Includes a "Device Enable" line to activate or deactivate the device.
    - Useful for shared buses, ensuring only one device drives the bus at a time.

Logic Components

1. **Logic Gates:** Perform basic logical operations (AND, OR, XOR, etc.) and are represented by truth tables.
2. **Buffer:**
    - Amplifies current or power.
    - Tri-state buffers used for address/data buses allow device selection.
3. **Latch:**
    - Stores binary data, triggered by a clock signal.
    - Example: 74LS373 IC for address latching.

AND Gate –Truth Table

I_1	I_2	O
0	0	0
0	1	0
1	0	0
1	1	1

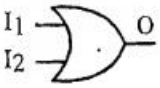


NAND Gate –Truth Table

I_1	I_2	O
0	0	1
0	1	1
1	0	1
1	1	0

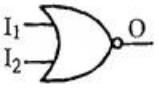
OR Gate –Truth Table

I_1	I_2	O
0	0	0
0	1	1
1	0	1
1	1	1



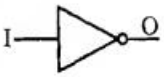
NOR Gate –Truth Table

I_1	I_2	O
0	0	1
0	1	0
1	0	0
1	1	0



NOT Gate –Truth Table

I	O
0	1
1	0

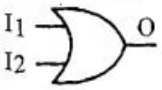


XOR Gate –Truth Table

I_1	I_2	O
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate –Truth Table

I_1	I_2	O
0	0	1
0	1	0
1	0	0
1	1	1



Logic Gates Truth Table and Symbolic representation

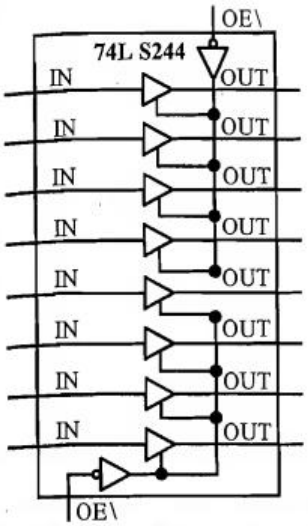


Fig. 8.3 74LS244 Octal Buffer IC

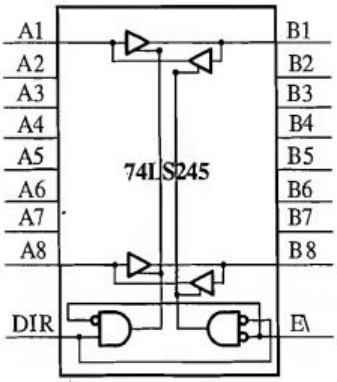


Fig. 8.4 74LS245 Octal bidirectional Buffer IC

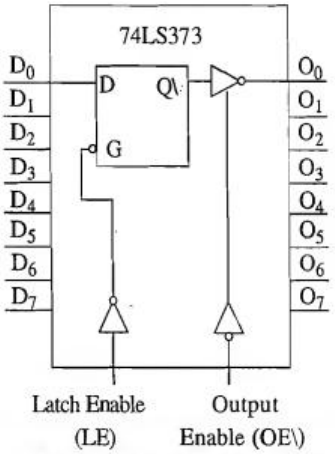


Fig. 8.5 74LS373 Octal Latch IC

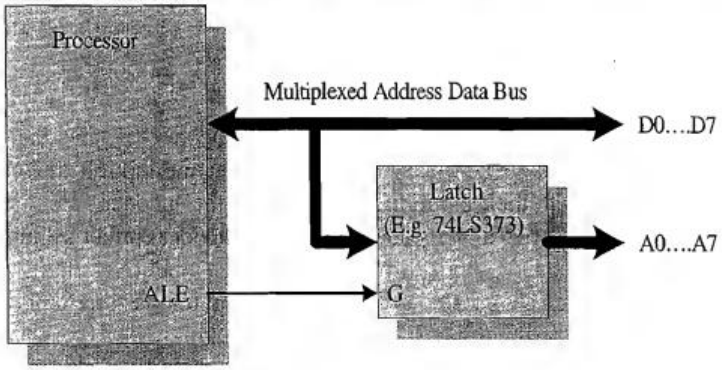
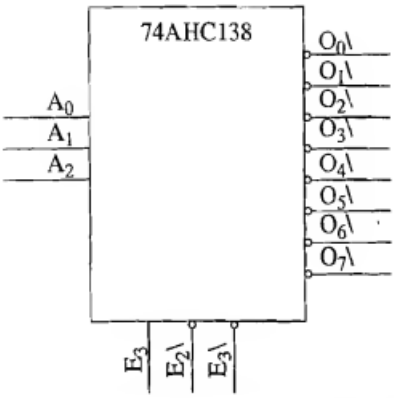


Fig. 8.6 Latch IC for address latching in multiplexed address data-bus

4. **Decoder:**
   - Converts input signals into multiple outputs.
   - Example: 74LS138 (3-to-8 decoder) for address decoding.

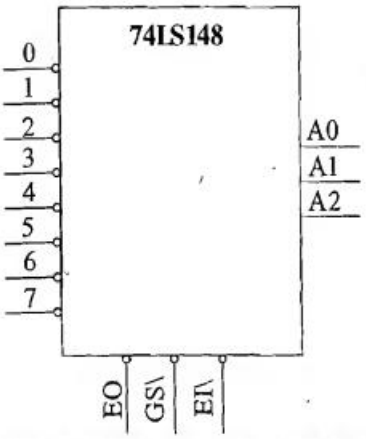


Input						Output							
A_2	A_1	A_0	E_1\	E_2\	E_3\	O_0\	O_1\	O_2\	O_3\	O_4\	O_5\	O_6\	O_7\
0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1	1	0	1
1	1	1	0	0	1	1	1	1	1	1	1	1	0

Fig. 8.7 3 to 8 Decoder IC and I/O signal states

5. **Encoder:**
   - Performs the reverse of a decoder, encoding inputs to outputs.

- Example: 74LS148 (8-to-3 encoder) for keyboard encoding.

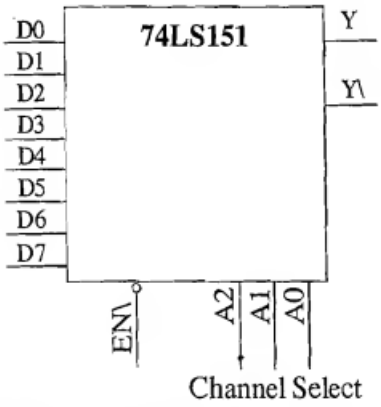


Input									Output				
7	6	5	4	3	2	1	0	EI	GS	EO	A2	A1	A0
1	1	1	1	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	1	0	0	1	1	1	0
1	1	1	1	1	0	1	1	0	0	1	1	0	1
1	1	1	1	0	1	1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	1	0	0	1	0	1	1
1	1	0	1	1	1	1	1	0	0	1	0	1	0
1	0	1	1	1	1	1	1	0	0	1	0	0	1
0	1	1	1	1	1	1	1	0	0	1	0	0	0

Fig. 8.8 8 to 3 Encoder IC and I/O signal states

Multiplexers and De-Multiplexers

1. **Multiplexer (MUX):**
   - Connects multiple inputs to a single output, controlled by selector lines.
   - Example: 74S151 (8-to-1 MUX).

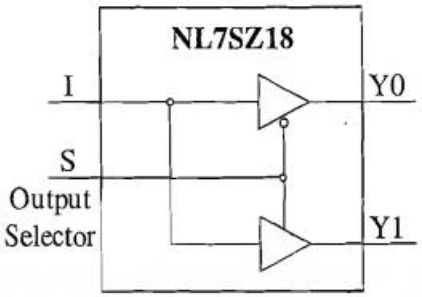


Input				Output	
A2	A1	A0	EN\	Y	Y\
0	0	0	0	D0	D0\
0	0	1	0	D1	D1\
0	1	0	0	D2	D2\
0	1	1	0	D3	D3\
1	0	0	0	D4	D4\
1	0	1	0	D5	D5\
1	1	0	0	D6	D6\
1	1	1	0	D7	D7\
x	x	x	1	0	1

x: Don't care

8 to 1 multiplexer IC and I/O signal states

2. **De-Multiplexer (D-MUX):**
   - Routes a single input to one of many outputs, controlled by selector lines.
   - Example: NL7SZ18 (1-to-2 D-MUX).



Input		Output	
S	I	Y0	Y1
0	0	0	Z
0	1	1	Z
1	0	Z	0
1	1	Z	1

Z : High Impedance

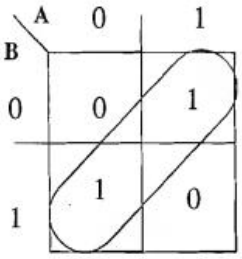
1 to 2 De-multiplexer IC and I/O signal states

Combinational and Sequential Circuits

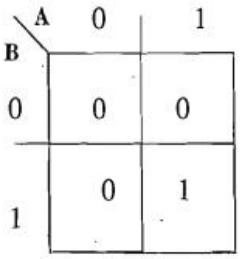
1. **Combinational Circuits:**
   - Output depends only on current inputs.
   - Examples: Encoders, decoders, multiplexers, adders.
   - Logic simplified using techniques like Karnaugh Map (K-Map).

Input		Output	
B	A	O	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

C : Carry



K-Map for Output (O)



K-Map for Carry (C)

Fig. 8.11 Truth Table and K-map representation for Half Adder

The simplified logical expression for the output (Y) and carry (C) of half adder is given below.

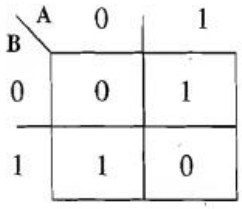
$$Y = \bar{A}B + A\bar{B}$$

$$C = AB$$

The logical expression $\bar{A}B + A\bar{B}$ represents an *XOR* gate. Please refer to the 'truth table' of *XOR* gate given under the '*Logic Gates*' section. Using K-map it can be represented as:

XOR Gate - T ruth Table

I_1	I_2	O
0	0	0
0	1	1
1	0	1
1	1	0



K-Map for XOR Gate

Fig. 8.12 Truth Table and K-map representation for XOR Gate

2. **Sequential Circuits:**
   - Output depends on current and past inputs.
   - Require memory elements (e.g., flip-flops).
   - Types:
     - **Synchronous:** Uses a clock signal (e.g., registers, counters).
     - **Asynchronous:** Relies on input changes (e.g., ripple counters).

3.5 VLSI & Integrated Circuit Design,

Brief Summary of VLSI and EDA Concepts

Integrated Circuit (IC) and VLSI Design

1. **IC Evolution:**
   - Transitioned from vacuum tubes to transistors to ICs.
   - IC types:
     - **SSI (Small-Scale Integration):** 1-2 logic gates (e.g., LS7400).
     - **MSI (Medium-Scale Integration):** Up to 100 gates (e.g., 7490 decade counter).
     - **LSI (Large-Scale Integration):** >1000 gates.
     - **VLSI (Very Large-Scale Integration):** Millions of gates (e.g., Pentium processor).
   - IC types include **Digital Design**, **Analog Design**, and **Mixed Signal Design**.
2. **VLSI Design Steps:**
   - **System Specification:** Defines chip functionality, input/output, timing, and power requirements.
   - **Design Entry:** Use block diagrams, truth tables, state diagrams, or HDLs like VHDL/Verilog.
   - **Functional Simulation:** Verifies functionality using simulation tools.
   - **Logic Synthesis:** Optimizes the design for speed, power, and area.
   - **Physical Design:** Maps circuits to target hardware (FPGA, ASIC, etc.).
   - **Timing Simulation:** Analyzes delays and validates timing requirements.
3. **VHDL for VLSI:**
   - Describes hardware behavior or structure.
   - Supports **Concurrent**, **Sequential**, **Hierarchical**, and **Timing Modeling**.
   - Enables portability across technologies like CPLD and FPGA.

3.6 Electronic Design Automation Tools

Electronic Design Automation (EDA) Tools

1. **Evolution:**
   - Shift from manual PCB design to automated tools.
   - Early PCBs relied on manual sketches; now replaced by CAD/CAM software.
2. **Capabilities of EDA Tools:**
   - Automate PCB design, routing, and layout.
   - Ensure precision and faster development cycles.
3. **Popular EDA Tools:**
   - **Key Players:** Cadence, Protel, Altium, Cadsoft.
   - **Common Tools:** OrCAD, Eagle, Protel, Cadstar.

- **OrCAD:** Widely used, flexible, user-friendly, with evaluation versions available.

This summary encapsulates IC and VLSI design methodologies, VHDL applications, and the role of modern EDA tools in simplifying PCB and hardware design.

Outcomes

At the end of the module, students will be able to:
CO-3: Analyse embedded system software and hardware requirements. [L3]

TEXT BOOKS:

Shibu K V, "Introduction to Embedded Systems", Second Edition, McGraw Hill Education

Reference Books/ Link

NPTL Lectures: https://nptel.ac.in/courses/108102045
Embedded Systems, IIT Delhi, Prof. Santanu Chaudhary