Module3 : Hardware Software Co design and Program Modelling

Hardware Software Co-Design

- In the traditional embedded system development approach, the hardware software partitioning is done at an early stage.
- Engineers from the software group take care of the software architecture development and implementation, whereas engineers from the hardware group are responsible for building the hardware required for the product.
- There is less interaction between the two teams and the development happens either serially or in parallel.
- Once the hardware and software are ready, the integration is performed.
- The increasing competition in the commercial market and need for reduced 'time-to-market' the product calls for a novel approach for embedded system design in which the hardware and software are co-developed instead of independently developing both.
- During the co-design process, the product requirements captured from the customer are converted into system level needs or processing requirements. At this point of time it is not segregated as either hardware requirement or software requirement, instead it is specified as functional requirement.
- The system level processing requirements are then transferred into functions which can be simulated and verified against performance and functionality.
- The Architecture design follows the system design.
  - The partition of system level processing requirements into hardware and software takes place during the architecture design phase.
  - Each system level processing requirement is mapped as either hardware and/or software requirement.
  - The partitioning is performed based on the hardware-software trade-offs.
- The architectural design results in the detailed behavioural description of the hardware requirement and the definition of the software required for the hardware.
- The processing requirement behaviour is usually captured using computational models.
- The models representing the software processing requirements are translated into firmware implementation using programming languages.

Fundamental Issues in Hardware Software Co-Design

- The fundamental issues in hardware software co-design are:
  - Selecting the Model
  - Selecting the Architecture
  - Selecting the Language
  - Partitioning System Requirements into Hardware and Software

Selecting the Model

- In hardware software co-design, models are used for capturing and describing the system characteristics.
- A model is a formal system consisting of objects and composition rules.
- It is hard to make a decision on which model should be followed in a particular system design.
- Most often designers switch between a variety of models from the requirements specification to the implementation aspect of the system design.
  - The reason being, the objective varies with each phase.
  - For example, at the specification stage, only the functionality of the system is in focus and not the implementation information.
  - When the design moves to the implementation aspect, the information about the system components is revealed and the designer has to switch to a model capable of capturing the system's structure.

Selecting the Architecture

- A model only captures the system characteristics and does not provide information on 'how the system can be manufactured?'.
- The architecture specifies how a system is going to implement in terms of the number and types of different components and the interconnection among them.
- The commonly used architectures in system design are Controller Architecture, Datapath Architecture, Complex Instruction Set Computing (CISC), Reduced Instruction Set Computing (RISC), Very Long Instruction Word Computing (VLIW), Single Instruction Multiple Data (SIMD), Multiple Instruction Multiple Data (MIMD), etc.
  - Some of them fall into Application Specific Architecture Class (like controller architecture), while others fall into either general purpose architecture class (CISC, RISC, etc.) or Parallel processing class (like VLIW, SIMD, MIMD, etc.).

Selecting the Language

- A programming language captures a 'Computational Model' and maps it into architecture.
- There is no hard and fast rule to specify this language should be used for capturing this model.

- A model can be captured using multiple programming languages like C, C++, C#, Java, etc. for software implementations and languages like VHDL, System C, Verilog, etc. for hardware implementations.
- On the other hand, a single language can be used for capturing a variety of models.
- Certain languages are good in capturing certain computational model.
- For example, C++ is a good candidate for capturing an object oriented model.
- The only pre-requisite in selecting a programming language for capturing a model is that the language should capture the model easily.

Partitioning System Requirements into Hardware and Software

- From an implementation perspective, it may be possible to implement the system requirements in either hardware or software (firmware).
- It is a tough decision making task to figure out which one to opt.
- Various hardware software trade-offs are used for making a decision on the hardware-software partitioning.

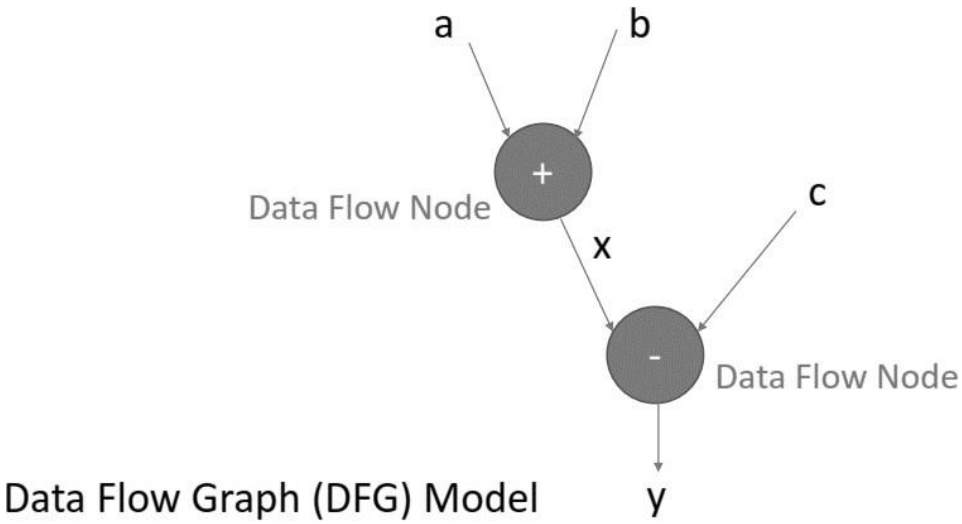
Computational Models in Embedded Design

- The commonly used computational models in embedded system design are:
  - Data Flow Graph Model
  - Control Data Flow Graph Model
  - State Machine Model
  - Sequential Program Model
  - Concurrent/Communicating Process Model
  - Object-Oriented Model

Data Flow Graph/Diagram (DFG) Model

- The Data Flow Graph (DFG) model translates the data processing requirements into a data flow graph.
- It is a data driven model in which the program execution is determined by data.
- This model emphasises on the data and operations on the data which transforms the input data to output data.
- Embedded applications which are computational intensive and data driven are modelled using the DFG model.
  - DSP applications are typical examples for it.
- Data Flow Graph (DFG) is a visual model in which the operation on the data (process) is represented using a block (circle) and data flow is represented using arrows.
- An inward arrow to the process (circle) represents input data and an outward arrow from the process (circle) represents output data in DFG notation.
- Suppose one of the functions in our application contains the computational requirement $x = a + b$ and $y = x - c$.

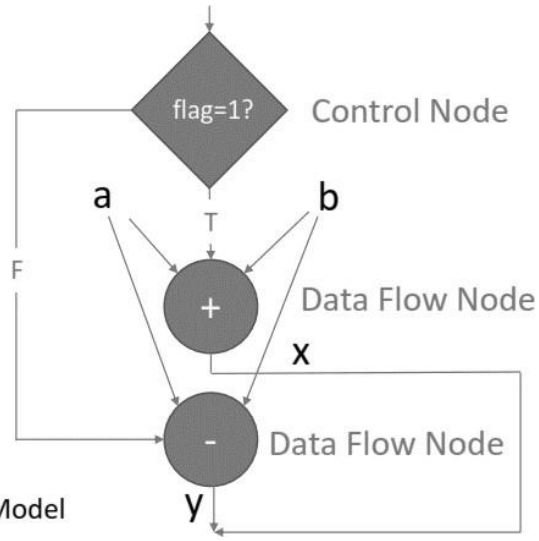
- Figure illustrates the implementation of a DFG model for implementing these requirements.



- In a DFG model, a data path is the data flow path from input to output.
- A DFG model is said to be acyclic DFG (ADFG) if it doesn't contain multiple values for the input variable and multiple output values for a given set of input(s).
    - Feedback inputs (Output is fed back to Input), events, etc. are examples for non-acyclic inputs.
- A DFG model translates the program as a single sequential process execution.

Control Data Flow Graph/Diagram (CDFG) Model

- The DFG model is a data driven model in which the execution is controlled by data and it doesn't involve any control operations (conditionals).
- The Control DFG (CDFG) model is used for modelling applications involving conditional program execution.
- CDFG models contains both data operations and control operations. The CDFG uses Data Flow Graph (DFG) as element and conditional (constructs) as decision makers.
- CDFG contains both data flow nodes and decision nodes, whereas DFG contains only data flow nodes.
- Consider the implementation of the CDFG for the following requirement.
- $If\ flag = 1, x = a + b; else\ y = a - b;$
- This requirement contains a decision making process.
- The CDFG model for the same is given in the figure.
- The control node is represented by a 'Diamond' block which is the decision making element in a normal flow chart based design.
- CDFG translates the requirement, which is modelled to a concurrent process model.
- The decision on which process is to be executed is determined by the control node.



Control Data Flow Graph (CDFG) Model

- A real world example for modelling the embedded application using CDFG is capturing and saving of the image to a format set by the user in a digital still camera.
- The decision on, in which format the image is stored (formats like JPEG, TIFF, BMP, etc.) is controlled by the camera settings, configured by the user.

State Machine Model

- The State Machine Model is used for modelling reactive or event-driven embedded systems whose processing behaviour are dependent on state transitions.
    - Embedded systems used in the control and industrial applications are typical examples for event driven systems.
- The State Machine model describes the system behaviour with 'States', 'Events', 'Actions' and 'Transitions'.
    - State is a representation of a current situation.
    - An event is an input to the state.
        - The event acts as stimuli for state transition.
    - Transition is the movement from one state to another.
    - Action is an activity to be performed by the state machine.

Finite State Machine (FSM) Model

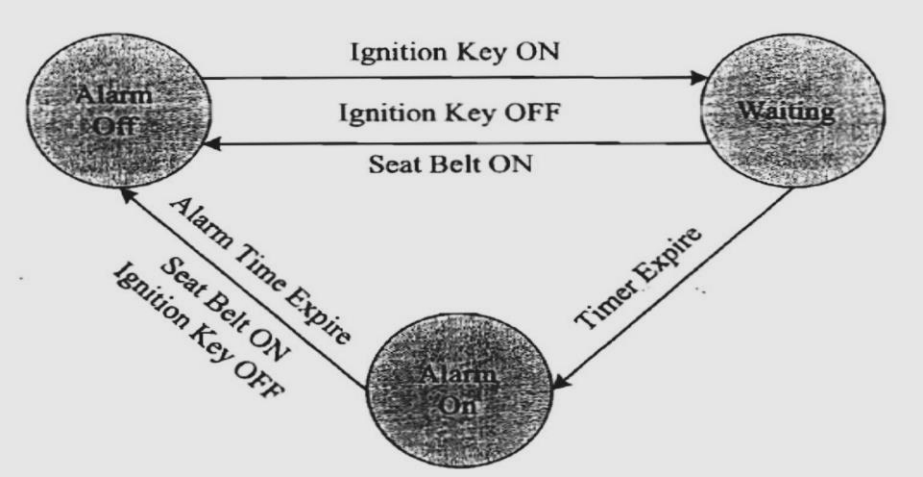
- A Finite State Machine (FSM) model is one in which the number of states are finite.
    - The system is described using a finite number of possible states.
- As an example, let us consider the design of an embedded system for driver/passenger 'Seat Belt Warning' in an automotive using the FSM model.
- The system requirements are captured as.
    - When the vehicle ignition is turned on and the seat belt is not fastened within 10 seconds of ignition ON, the system generates an alarm signal for 5 seconds.

- The Alarm is turned off when the alarm time (5 seconds) expires or if the driver/passenger fastens the belt or if the ignition switch is turned off, whichever happens first.

➢ Here the states are
- 'Alarm Off'
- 'Waiting'
- 'Alarm On'

➢ The events are
- 'Ignition Key ON'
- 'Ignition Key OFF'
- 'Timer Expire'
- 'Alarm Time Expire'
- 'Seat Belt ON'

➢ Using the FSM, the system requirements can be modeled as given in figure.



➢ The 'Ignition Key ON' event triggers the 10 second timer and transitions the state to 'Waiting'.

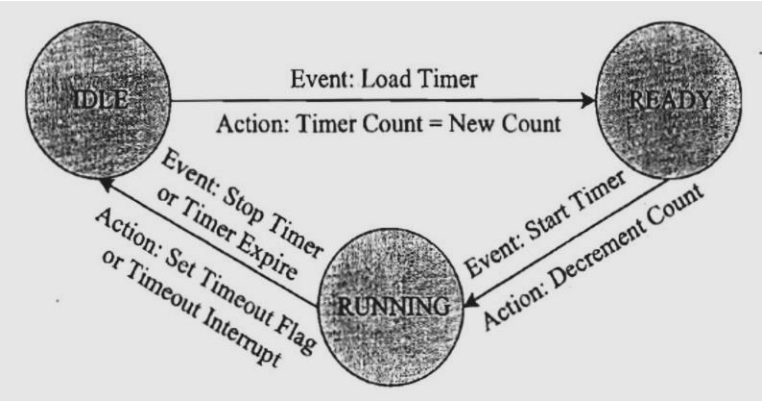
➢ If a Seat Belt ON' or 'Ignition Key OFF' event occurs during the wait state, the state transitions into 'Alarm Off'.

➢ When the wait timer expires in the waiting state, the event 'Timer Expire' is generated and it transitions the state to 'Alarm On' from the 'Waiting' state.

➢ The 'Alarm On' state continues until a 'Seat Belt ON' or 'Ignition Key OFF' event or 'Alarm Time Expire' event, whichever occurs first.

➢ The occurrence of any of these events transitions the state to 'Alarm Off'.

➢ The wait state is implemented using a timer.
- The timer also has certain set of states and events for state transitions.
- Using the FSM model, the timer can be modelled as shown in the figure.



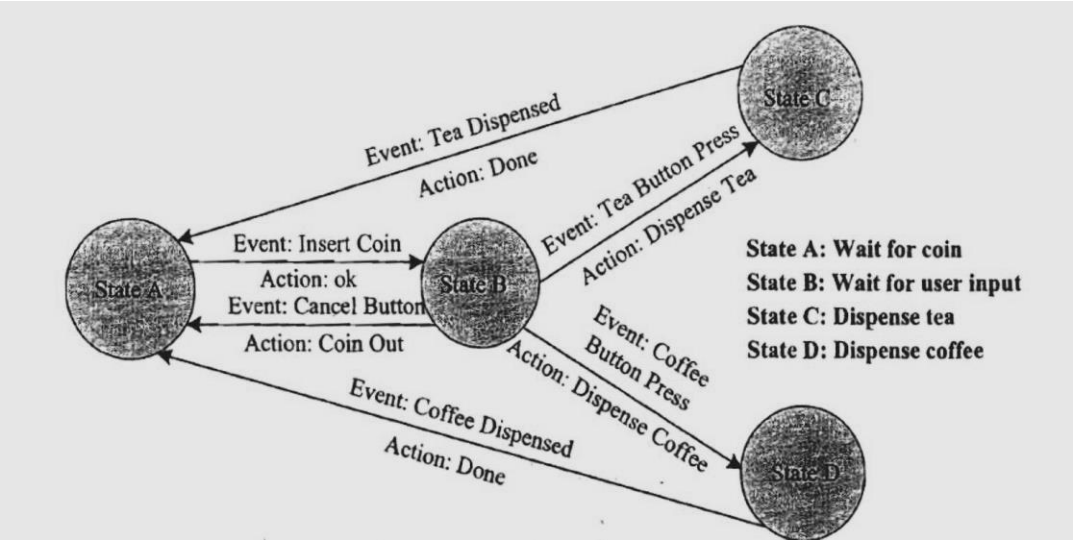
- As seen from the FSM, the timer state can be either 'IDLE' or 'READY' or 'RUNNING'.
- During the normal condition when the timer is not running, it is said to be in the 'IDLE' state.
- The timer is said to be in the 'READY' state when the timer is loaded with the count corresponding to the required time delay.
- The timer remains in the 'READY' state until a 'Start Timer' event occurs.
- The timer changes its state to 'RUNNING' from the 'READY' state on receiving a 'Start Timer' event and remains in the 'RUNNING' state until the timer count expires or a 'Stop Timer' even occurs.
- The timer state changes to 'IDLE' from 'RUNNING' on receiving a 'Stop Timer' or 'Timer Expire' event.

FSM Model – Example 1

- Design an automatic tea/coffee vending machine based on FSM model for the following requirement.
    - 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.

Solution

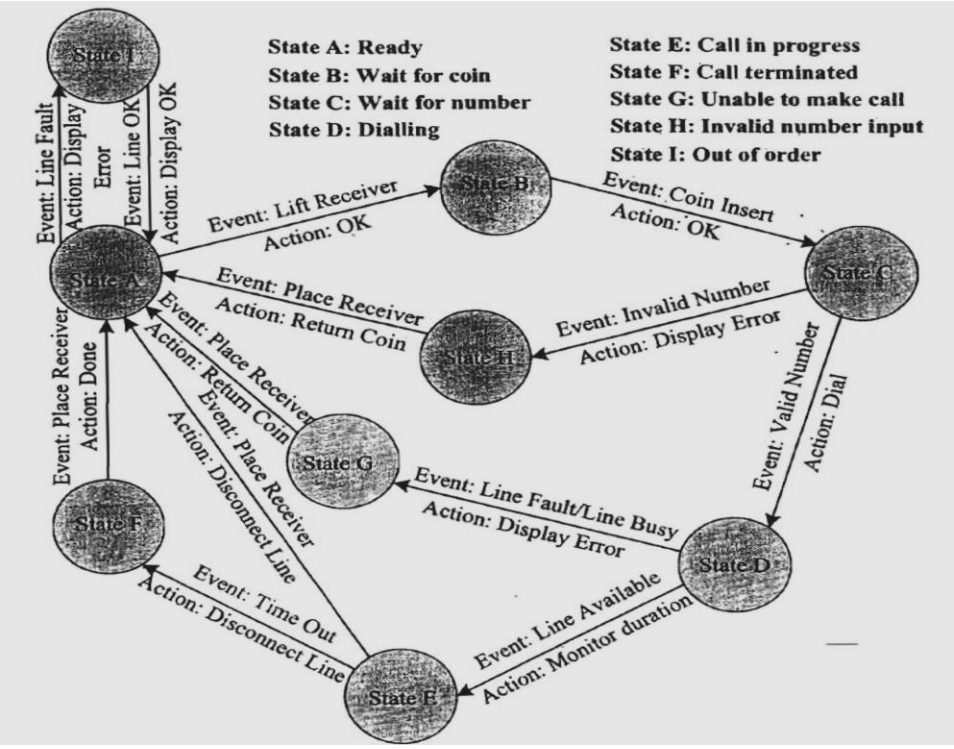
- The FSM Model contains four states namely,
    - 'Wait for coin'
    - 'Wait for User Input'
    - 'Dispense Tea'
    - 'Dispense Coffee'



- The event 'Insert Coin' (5 rupee coin insertion), transitions the state to 'Wait for User Input'.
- The system stays in this state until a user input is received from the buttons 'Cancel', 'Tea' or 'Coffee' (Tea and Coffee are the drink select button).
- If the event triggered in 'Wait State' is 'Cancel' button press, the coin is pushed out and the state transitions to 'Wait for Coin'.
- If the event received in the 'Wait State' is either 'Tea' button press, or 'Coffee' button press, the state changes to 'Dispense Tea' and 'Dispense Coffee' respectively.
- Once the coffee/tea vending is over, the respective states transition back to the 'Wait for Coin' state.

FSM Model – Example 2

- Design a coin operated public telephone unit based on FSM model for the following requirements.
    - The calling process is initiated by lifting the receiver (off-hook) of the telephone unit.
    - After lifting the phone the user needs to insert a 1 rupee coin to make the call.
    - If the line is busy, the coin is returned on placing the receiver back on the hook (on-hook).
    - If the line is through, the user is allowed to talk till 60 seconds and at the end of 45^{th} second, prompt for inserting another 1 rupee coin for continuing the call is initiated.
    - If the user doesn't insert another 1 rupee coin, the call is terminated on completing the 60 seconds time slot.
    - The system is ready to accept new call request when the receiver is placed back on the hook (on-hook).
    - The system goes to the 'Out of Order' state when there is a line fault.

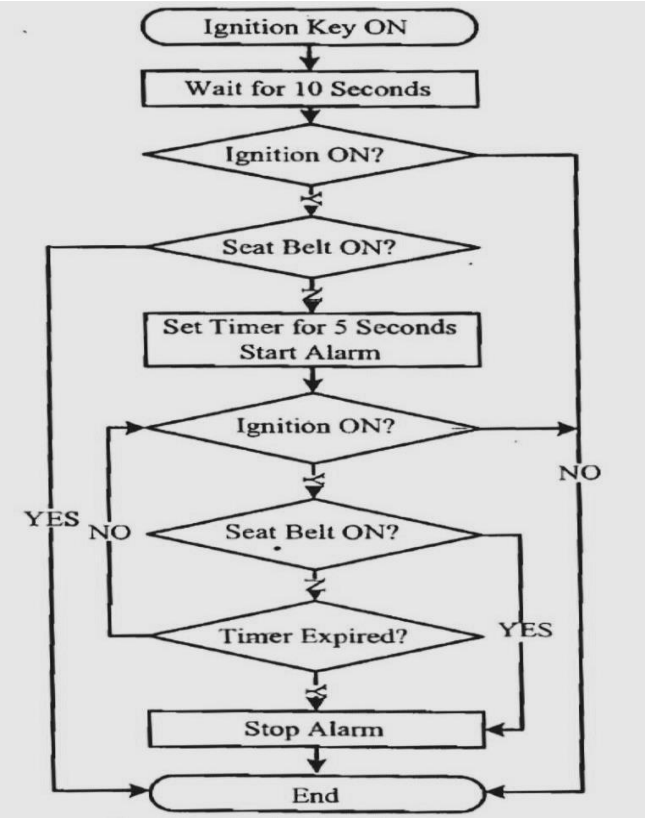


Sequential Program Model

- In the Sequential Program 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.
- Finite State Machines (FSMs) and Flow Charts are used for modelling sequential program.
  - 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
#define YES 1
#define NO 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()==ON)&&(timer_expire()==NO));
   stop_alarm();
   }
  }
 }
}
```

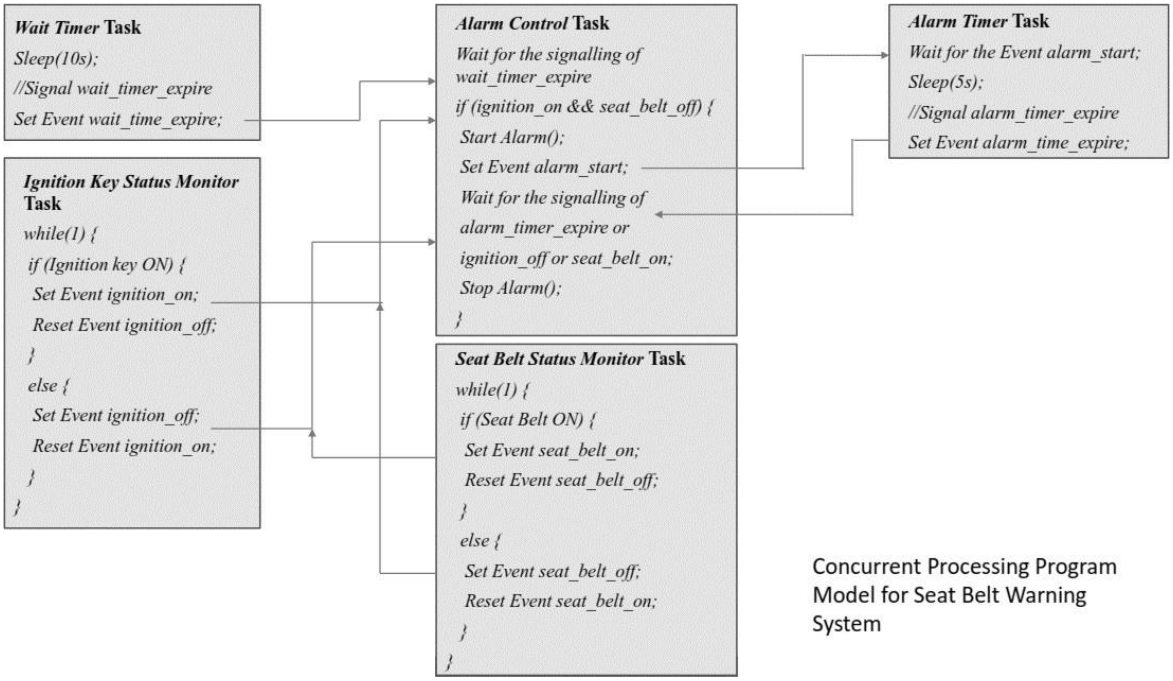
- Sequential Program Model for Seat Belt Warning System



Concurrent/Communicating Process Model

- The concurrent or communicating process model models concurrently executing tasks/processes.
- It is easier to implement certain requirements in concurrent processing model than the conventional sequential execution.
    - Sequential execution leads to a single sequential execution of task and thereby leads to poor processor utilisation, when the task involves I/O waiting, sleeping for specified duration etc.
    - If the task is split into multiple subtasks, it is possible to tackle the CPU usage effectively by switching the task execution, when the subtask under execution goes to a wait or sleep mode.
- However, concurrent processing model requires additional overheads in task scheduling, task synchronization and communication.
- As an example, consider the implementation of the 'Seat Belt Warning' system using concurrent processing model.
- We can split the tasks into:
    - Timer task for waiting 10 seconds (wait timer task)
    - Task for checking the ignition key status (ignition key status monitoring task)
    - Task for checking the seat belt status (seat belt status monitoring task)
    - Task for starting and stopping the alarm (alarm control task)
    - Alarm timer task for waiting 5 seconds (alarm timer task)

- The tasks cannot be executed them randomly or sequentially.
- We need to synchronize their execution through some mechanism.



Concurrent Processing Program Model for Seat Belt Warning System

Object-Oriented Model

- The object-oriented model is an object based model for modelling system requirements.
- It disseminates a complex software requirement into simple well defined pieces called objects.
- Object-oriented model brings re-usability, maintainability and productivity in system design.
- In the object-oriented modelling, object is an entity used for representing or modelling a particular piece of the system.
    - Each object is characterized by a set of unique behaviour and state.
- A class is an abstract description of a set of objects and it can be considered as a 'blueprint' of an object.
- A class represents the state of an object through member variables and object behaviour through member functions.
- The member variables and member functions of a class can be private, public or protected.
    - Private member variables and functions are accessible only within the class, whereas public variables and functions are accessible within the class as well as outside the class.
    - The protected variables and functions are protected from external access.
    - However, classes derived from a parent class can also access the protected member functions and variables.

Embedded Hardware Design and Development:

ANALOG ELECTRONIC COMPONENTS

- Resistors, capacitors, diodes, inductors, operational amplifiers (OpAmps), transistors, etc. are the commonly used analog electronic components in embedded hardware design.
- A resistor limits the current flowing through a circuit.
- Interfacing of LEDs, buzzer, etc. with the port pins of microcontroller through current limiting resistors is a typical example for the usage of resistors in embedded application.
- Capacitors and inductors are used in signal filtering and resonating circuits.
- Reset circuit implementation, matching circuits for RF designs, power supply decoupling, etc. are examples for the usage of capacitors in embedded hardware circuit.
- Inductors are widely used for filtering the power supply from ripples and noise signals. Inductors with inductance value in the microhenry (µH) range are commonly used in embedded applications for filter and matching circuit implementation.
- P-N Junction diode, Schottky diode, Zener diode, etc. are the commonly used diodes in embedded hardware circuits.
- Transistors in embedded applications are used for either switching or amplification purpose. In switching application, the transistor is in either ON or OFF state.
- Relay, buzzer and stepper motor driving circuits are examples for common emitter confi guration based driver circuit implementation using transistor

DIGITAL ELECTRONIC COMPONENTS

- Digital electronics deal with digital or discrete signals.
- Microprocessors, Microcontrollers, and System on Chips (SoCs) work on digital principles. They interact with the rest of the world through digital I/O interfaces and process digital data.
- Embedded systems employ various digital electronic circuits for ‘Glue logic’ implementation.
- ‘Glue logic’ is the custom digital electronic circuitry required to achieve compatible interface between two different integrated circuit chips.
- Address decoders, latches, encoders/ decoders, etc. are examples for glue logic circuits. Transistor Transistor Logic (TTL), Complementary Metal Oxide Semiconductor (CMOS) logic etc are some of the standards describing the electrical characteristics of digital signals in a digital system.

Open Collector and Tri-State Output

- Open collector is an I/O interface standard in digital system design.
- The term ‘open collector’ is commonly used in conjunction with the output of an Integrated Circuit (IC) chip.
- It facilitates the interfacing of IC output to other systems which operate at different voltage levels.
- In the open collector configuration, the output line from an IC circuit is connected to the base of an NPN transistor. The collector of the transistor is left unconnected (floating) and the emitter is internally connected to the ground signal of IC.
- Figure 8.1 illustrates an open collector output configuration.
- The output signal of the IC is fed to the base of an open collector transistor. When the base drive to the transistor is ON and the collector is in open state, the o/p pin floats. This state is also known as ‘high impedance’ state.
- Here the output is neither driven to logic ‘high’ nor logic ‘low’.
- If a pull-up resistor is connected to the o/p pin, when the base drive is ON, the o/p pin becomes at logic 0 (0V).
- With a pull-up resistor, if the base driver is 0, the o/p will be at logic high (Voltage =

Vcc).

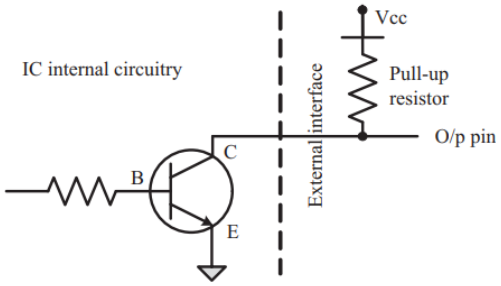


Fig. 8.1 Open collector output configuration

The advantage of open collector output in embedded system design is listed below.

(1) It eliminates the need for additional interface circuits for connecting devices at different voltage levels.

(2) An open collector configuration supports multi-drop connection, i.e., connecting more than one open collector output to a single line.

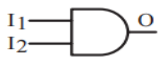
(3) It is easy to build 'Wired AND' and 'Wired OR' configuration using open collector output lines.

Logic Gates

- Logic gates are the building blocks of digital circuits.
- Logic gates control the flow of digital information by performing a logical operation of the input signals.
- Depending on the logical operation, the logic gates used in digital design are classified into–AND, OR, XOR, NOT, NAND, NOR and XNOR.
- The logical relationship between the output signal and the input signals for a logic gate is represented using a truth table. Figure 8.2 illustrates the truth table and symbolic representation of each logic gate

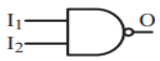
AND Gate –Truth Table

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

Fig. 8.2 Logic Gates Truth Table and Symbolic representation

Buffer

- A buffer circuit is a logic circuit for amplifying the current or power.
- It increases the driving capability of a logic circuit.
- A tri-state buffer is a buffer with Output Enable control. When the Output Enable control is active (Low for Active low enable and High for Active high enable), the tri-state buffer functions as a buffer.
- If the Output Enable is not active, the output of the buffer remains at high impedance state (Tri-stated).
- Tristate buffers are commonly used as drivers for address bus and to select the required device among multiple devices connected to a shared data bus.
- Tri-state buffers are available as either unidirectional or bi-directional buffers. 74LS244/74HC244 is an example of unidirectional octal buffer.
- It contains 8 individual buffers which are grouped into two. Each buffer group has its own output enable line.
- Figure 8.3 illustrates the 74LS244 buffer device.
- IC 74LS245 is an example of bi-directional tri-state buffer. It allows data flow in both directions, one at a time.
- The data fl ow direction can be set by the direction control line. One buffer is allocated for the data line associated with each direction.
- Figure 8.4 illustrates the 74LS245 octal bi-directional buffer.

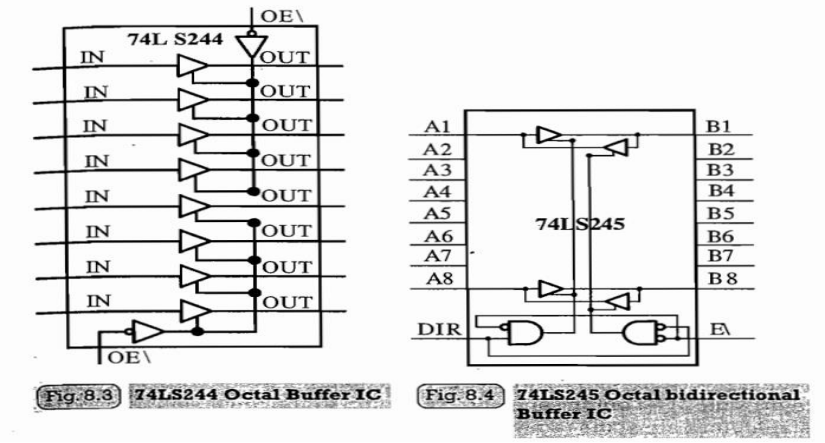


Fig. 8.3 74LS244 Octal Buffer IC

Fig. 8.4 74LS245 Octal bidirectional Buffer IC

Latch

- A latch is used for storing binary data.
- It contains an input data line, clock or gating control line for triggering the latching operation and an output line.
- The gating signal can be either a positive edge (raising edge) or a negative edge (falling edge).
- Whenever a latch trigger happens, the data present on the input line is latched.
- The latched data is available on the output line of the latch until the next trigger.
- D flip flop is a typical example of a latch.
- Latches are available as integrated circuits, IC 74LS373 being a typical example.
- It contains 8 individual D latches. The 74LS373 latch IC (Fig. 8.5) is commonly used for latching the lower order address byte in a multiplexed address data bus system.
- The Address Latch Enable (ALE) pulse generated by the processor, when the Address bits are available on the multiplexed bus, is used as the latch trigger.
- Figure 8.6 illustrates the usage of latches in address latching.

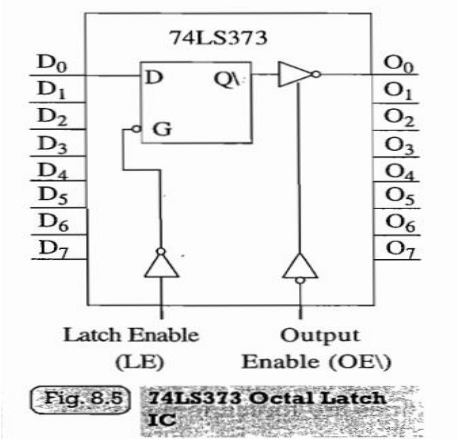
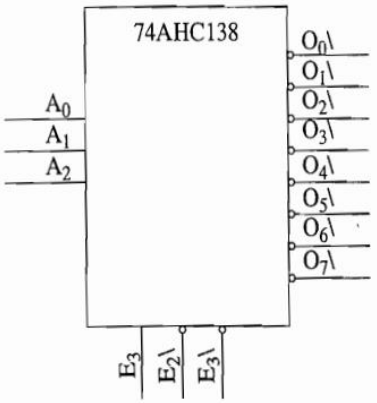


Fig. 8.5 74LS373 Octal Latch IC

Decoder

- A decoder is a logic circuit which generates all the possible combinations of the input signals.
- Examples are 2 to 4 decoder, 3 to 8 decoder and 4 to 16 decoder.
- The 3 to 8 decoder contains 3 input signal lines and it is possible to have 8 different configurations with the 3 lines.
- Depending on the input signal, the corresponding output line is asserted.
- For example, for the input state 001, the output line 2 is asserted.
- Decoders are mainly used for address decoding and chip select signal generation in electronic circuits and are available as integrated circuits.
- 74LS138/74AHC138 is an example for 3 to 8 decoder IC.
- Figure 8.7 illustrates the 74AHC138 decoder and the function table for it.
- The decoder output is enabled only when the 'Output Enable' signal lines E1\, E2\ and E3 are at logic levels 0, 0 and 1 respectively.
- If the output-enable signals are not at the required logic state, all the output lines are forced to the inactive (High) state.

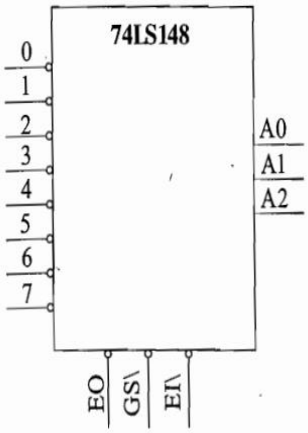


Input						Output							
A_2	A_1	A_0	E_1\	E_2\	E_3\	O_0\	O_1\	O_2\	O_3\	O_4\	O_5\	O_6\	O_7\
0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	0	1	1	1	1	1
0	1	1	0	0	1	1	1	1	0	1	1	1	1
1	0	0	0	0	1	1	1	1	1	0	1	1	1
1	0	1	0	0	1	1	1	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1	1	1	1	0	1
1	1	1	0	0	1	1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

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

Encoder

- An encoder performs the reverse operation of decoder.
- The encoder encodes the corresponding input state to a particular output format.
- The 8 to 3 encoder contains 8 input signal lines and it is possible to generate a 3 bit binary output corresponding to the input.
- For example, if the input line 1 is asserted, the output lines A0, A1 and A2 are asserted as 0, 1 and 1 respectively.
- Encoders are mainly used for address decoding and chip select signal generation.
- 74F148/74LS148 is an example of 8 to 3 encoder IC.
- Figure 8.8 illustrates the 74F148/74LS148 encoder and the function table for it
- The encoder output is enabled only when the 'Enable Input (EI)' signal line is at logic 0.
- A 'High' on the Enable Input (EI) forces all outputs to the inactive (High) state and allows new data to settle without producing erroneous information at the outputs.
- The group signal (GS) is active-Low when any input is Low.
- 74LS148/74F148 is a priority encoder and it provides priority encoding of the inputs to ensure that only the highest order data line is encoded when multiple data lines are asserted.
- Encoding of keypress in a keyboard is a typical example for an application requiring encoder. The encoder converts each keypress to a binary code

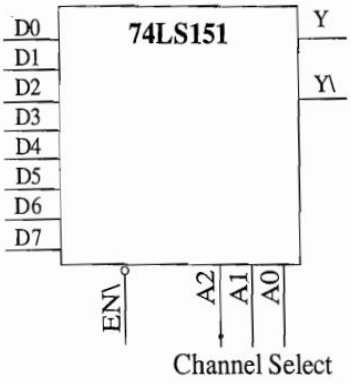


Input									Output				
7	6	5	4	3	2	1	0	E1	GS	E0	A2	A1	A0
1	1	1	1	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	1	0	0	1	1	1	0
1	1	1	1	1	0	1	1	0	0	1	1	0	1
1	1	1	1	0	1	1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	1	0	0	1	0	1	1
1	1	0	1	1	1	1	1	0	0	1	0	1	0
1	0	1	1	1	1	1	1	0	0	1	0	0	1
0	1	1	1	1	1	1	1	0	0	1	0	0	0

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

Multiplexer (MUX)

- A multiplexer (MUX) can be considered as a digital switch which connects one input line from a set of input lines, to an output line at a given point of time.
- It contains multiple input lines and a single output line.
- The inputs of a MUX are said to be multiplexed. The input line is selected through the MUX control lines.
- 74S151 is an example for 8 to 1 multiplexer IC.
- Figure 8.9 illustrates the 74S151 multiplexer and the function table for it.
- The multiplexer is enabled only when the 'Enable signal (EN)' line is at logic 0.
- A 'High' on the EN line forces the output to the inactive (Low) state.
- The input signal is switched to the output line through the channel select control lines A2, A1 and A0.
- In order to select a particular input line, apply its binary equivalent to the channel select lines A0, A1 and A2 (e.g. set A2A1A0 as 000 for selecting Input D0, and as 001 for selecting channel D1, etc.)



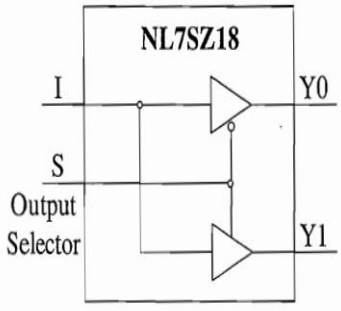
Input				Output	
A2	A1	A0	EN\	Y	Y\
0	0	0	0	D0	D0\
0	0	1	0	D1	D1\
0	1	0	0	D2	D2\
0	1	1	0	D3	D3\
1	0	0	0	D4	D4\
1	0	1	0	D5	D5\
1	1	0	0	D6	D6\
1	1	1	0	D7	D7\
x	x	x	1	0	1

x: Don't care

Fig. 8.9 8 to 1 multiplexer IC and I/O signal states

De-multiplexer (D-MUX)

- A de-multiplexer performs the reverse operation of multiplexer.
- De-multiplexer switches the input signal to the selected output line among a number of output lines.
- The output line to which the input is to be switched is selected by the output selector control lines.
- The 1 to 2 de-multiplexer, NL7SZ18 is a typical example for 1 to 2 de-multiplexer IC.
- It contains a single input line and two output lines to switch the input line.
- The output switching is controlled by the output selector control.
- When one output line is selected by the output selector control (S), the other output line remains in the High impedance state**.**
- Figure 8.10 illustrates the NL7SZ18 de-multiplexer and the function table for it.



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

Fig. 8.10 1 to 2 De-multiplexer IC and I/O signal states

Combinational Circuits

- A combinational circuit is a combination of the logic gates.
- The output of the combinational circuit, at a given point of time, is dependent only on the state of the inputs at the given point of time.
- Encoders, decoders, multiplexers, de-multiplexers, adder circuits, comparators, multiple input gates, etc. are examples of digital combinational circuits.
- In digital system design, logical functions representing a combinational circuit are expressed as either 'Sum of Products (SOP)' or 'Product of Sums (POS)' form.
- The SOP form represents the logic as the sum of the products of the logical variables whereas the POS form represents the logic as the product of sums of the logical variable.

- The expression $Y = AB + \overline{B}C + AC$ is an example for SOP form representation of the logical function. Here Y is the output signal and A, B and C are the input signals.
- The expression Y = (A+B) (B+C) (A+C) is an example for POS form representation of the logical function. Here Y is the output signal and A, B and C are the input signals.
- 'Karnaugh map' or K-map is the easiest logic simplification technique.
- Now let us try to implement a 2 input half adder combinational circuit, for adding two one-bit numbers, using the K-map technique.
- The 'Truth Table' and the corresponding K-map drawing* for the 2 input 'half adder' circuit is given below.

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

C : Carry

B \ A	0	1
0	0	1
1	1	0

K-Map for Output (O)

B \ A	0	1
0	0	0
1	0	1

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.

$$Y = \overline{A}B + A\overline{B}$$
$$C = AB$$

The logical expression $\overline{A}B + A\overline{B}$ 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

B \ A	0	1
0	0	1
1	1	0

K-Map for XOR Gate

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

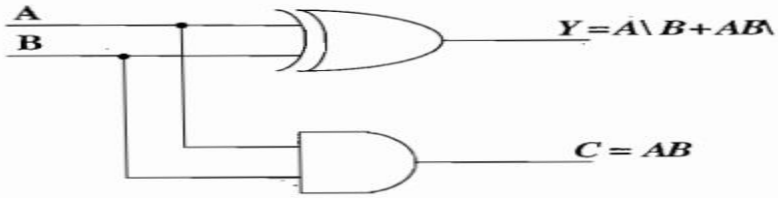


Fig. 8.13 **Half Adder Circuit Implementation**

Sequential Circuits

- In sequential circuits, output at any given point of time depends on both the present and past inputs.
- Hence, sequential circuits contain a memory element for holding the previous input states.
- Flip-flops act as the basic building blocks of sequential circuits.
- Sequential circuits are of two types, namely–synchronous (clocked) sequential circuits and asynchronous sequential circuits.

- The operation of a synchronous sequential circuit is synchronized to a clock signal, whereas an asynchronous sequential circuit does not require a clock for operation.
- For an asynchronous sequential circuit, the response depends upon the sequence in which the input signal changes.
- The memory capability to asynchronous sequential circuit is provided through feedback.
- Register, synchronous counters, etc. are examples of synchronous serial circuits,
- Ripple or asynchronous counter is an example for asynchronous sequential circuits.

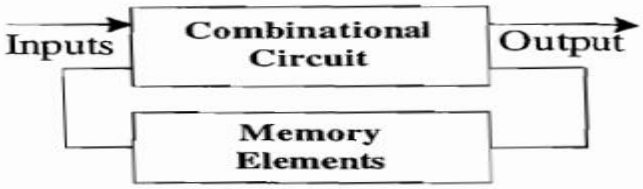
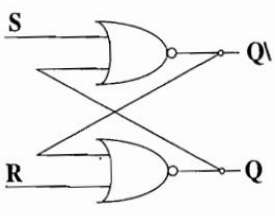


Fig. 8.14 Visualisation of Sequential Circuit

S-R flipflop

- The S-R flip-flop is built using 2 NOR gates.
- The output of each NOR gate is fed back as input to the other NOR gate.
- This ensures that if the output of one NOR gate is at logic 1, the output of the other NOR gate will be at logic 0.
- The S-R flip-flop works in the following way.
  (1) If the Set input (S) is at logic high and Reset input (R) is at logic low, the output remains at logic high regardless of the previous output state.
  (2) If the Set input (S) is at logic low and Reset input (R) is at logic high, the output remains at logic low regardless of the previous output state.
  (3) If both the Set input (S) and Reset input (R) are at logic low, the output remains at the previous logic state.
  (4) The condition Set input (S) = Reset input (R) = Logic high (1) will lead to race condition and the state of the circuit becomes undefined or indeterminate (x).
- A clock signal can be used for triggering the state change of flip-flops.
- The clock signal can be either level triggered or edge triggered.
- For level triggered flip-flops, the output responds to any changes in input signal, if the clock signal is active.
- For edge triggered flip-flops, the output state changes only when a clock trigger happens regardless of the changes in the input signal state.



Input		Output	
S	R	Previous State	Present State
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

S-R Flip-Flop and Truth Table

Clocked S-R flip-flop

- The clocked S-R flip-flop functions in the same way as that of S-R flip-flop.
- The only difference is that the output state changes only with a clock trigger.
- Even though there is a change in the input state, the output remains unchanged until the next clock trigger.
- When a clock trigger occurs, the output state changes in accordance with the values of S and R at the time of the clock trigger.

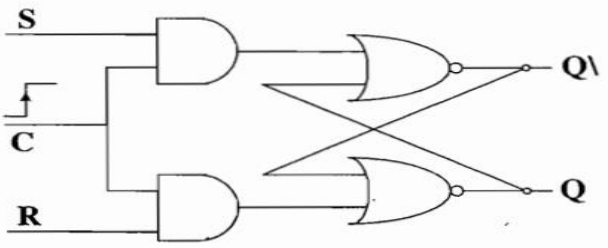
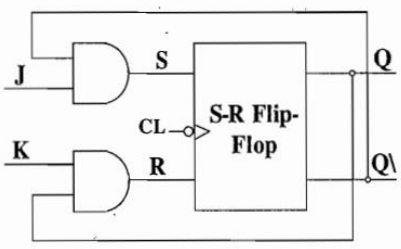


Fig. 8.16 Clocked S-R Flip-Flop

J-K flip-flop

- The J-K flip-flop augments the behavior of S-R flip by interpreting the input state S=R=1 as a toggle command.
- The logic circuit and the I/O states for a J-K flip-flop are given in Fig. below

It is clear that for a J-K flip-flop, S = JQ\ and R = KQ.
The J-K flip-flop operates in the following way:
(1) When J = 1 and K = 0, the output remains in the set state.
(2) When J = 0 and K = 1, the output remains in the reset state.
(3) When J = K = 0, the output remains at the previous logic state.
(4) When J = 1 and K = 1, the output toggles its state

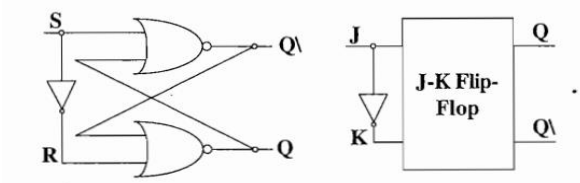


Input		Output	
J	K	Previous State	Present State
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

J-K Flip-Flop Implementation and Truth Table

D-type (Delay) flip-flop

- A D-type (Delay) flip-flop is formed by connecting a NOT gate in between the S and R inputs of an S-R flip-flop or by connecting a NOT gate between the J and K inputs of a J-K flip-flop.
- Figure 8.18 illustrates a D-type flip-flop and its I/O states.
- This flip-flop is known with the so-called name 'Delay' flip-flop for the following reason– the input to the flip-flop appears at the output at the end of the clock pulse (for falling edge triggering).



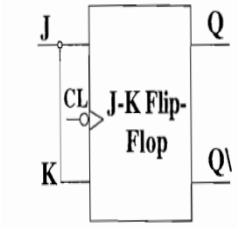
Input	Output
D	Q
0	0
1	1

3) D-Type Flip-Flop – Circuit and Truth Table

Toggle flip-flop or T flip-flop

- A Toggle flip-flop or T flip-flop is formed by combining the J and K inputs of a J-K flipflop.
- When the 'T' input is held at logic 1, the T flip-flop toggles the output with each clock signal.

Figure 8.19 illustrates a T flip-flop and its I/O states.



Input	Output
T	Q_{n+1}
0	Q_n
1	Q_n\

Q_{n+1} : Present Output
Q_n : Previous Output

T (Toggle) Flip-Flop – Circuit and Truth Table

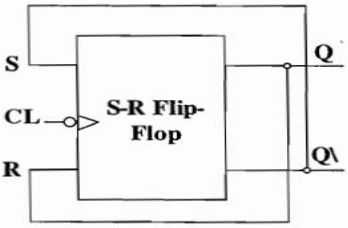


Fig. 8.20 S-R Flip-Flop acting as Toggle switch

- An S-R flip-flop cannot be converted to a 'T' flipflop by combining the inputs S and R. The logic state S=R=1 is not defined in the S-R flip-flop. However, an S-R flip-flop can be configured for toggling the output with the circuit configuration shown in Fig. 8.20.

Synchronous sequential circuit

- Uses clocked flip-flops (S-R, J-K, D, T etc.) as memory elements and a 'state' change occurs only in response to a synchronizing clock signal.
- The clock signal is common to all flip-flops and the clock pulse is applied simultaneously to all flip-flops.

  Example 1 for synchronous sequential circuit, let us consider the design of a synchronous 3-bit binary counter. The counting sequence for a 3-bit binary counter is given below.

Count	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

- From the count sequence, it is clear that the LS bit (Q0) of the binary counter toggles on each count and the next LS bit (Q1) toggles only when the LS bit (Q0) makes a transition from 1 to 0.
- The Bit (Q2) of the binary counter toggles its state only when the Q1 bit and Q0 bits are at logic 1.

- This logic circuit can be realized with 3 T flip-flops satisfying the following criteria:
  (1) All the T flip-flops are driven simultaneously by a single clock (synchronous design).
  (2) The output of the T flip-flop representing the Q0 bit is initially set at 0. The input line of Q0 flip-flop is connected to logic 1 to ensure toggling of the output with input clock signal.
  (3) Since Q1 bit changes only when Q0 makes a transition from 1 to 0, the output of the T flip-flop representing Q0 is fed as input to the T flip-flop representing the Q1 bit.
  (4) The output bit Q2 changes only when Q0 = Q1 = 1. Hence the input to the flip-flop representing bit Q2 is fed by logically ANDing Q0 and Q1.

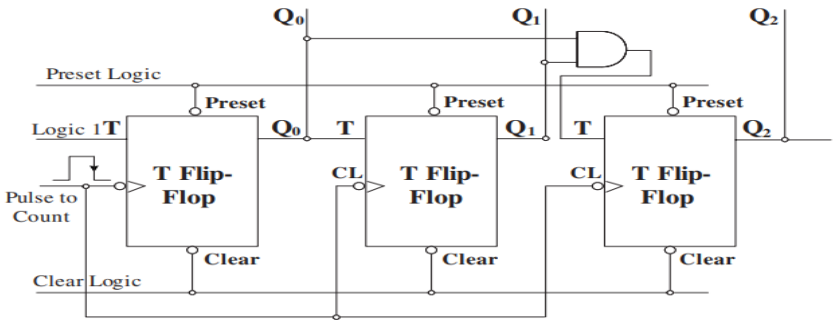


Fig. 8.21 Binary Counter Implementation using T Flip-Flops

- The Preset line is used for setting the output of flip-flop, whereas the Clear line is used for resetting the output of flip-flops.

Example 2:

- A register can be considered as a group of bits holding information.
- A D flip-flop can be used for holding a 'bit' information.
- The bit storing operation is controlled by the signal associated with a latch write (like a write to latch pulse).
- The figure given below illustrates the implementation of a 4 bit register using D flip-flops.

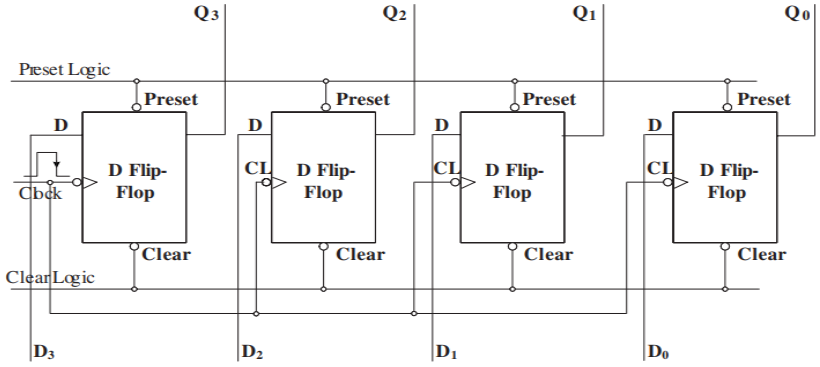


Fig. 8.22 4 bit Register Implementation using D Flip-Flops

Asynchronous sequential circuit

- The state of an asynchronous sequential circuit changes instantaneously with changes in input signal state.
- The asynchronous sequential circuit does not require a synchronizing clock signal for its operation.
- The memory element of an asynchronous sequential circuit can be either an un-clocked flip-flop or logical gate circuits with feedback loops for latching the state.

Example

3-bit binary counter using an asynchronous sequential circuit.

- From the count sequence of the 3-bit binary counter, it is clear that the least significant (LS) bit (Q0) of the binary counter toggles on each count and the next LS bit (Q1) toggles only when the LS bit (Q0) makes a transition from 1 to 0.
- The most significant (MS) Bit (Q2) of the binary counter toggles its state only when the Q1 bit makes a transition from 1 to 0.
- This logic circuit can be realized with 3 T flip-flops satisfying the following criteria:

(1) The T input of all flip-flops are connected to logic high (it is essential for the toggling condition of the T Flip-flop).

(2) The pulse for counting is applied to the clock input of the first T flip-flop representing the LS bit Q0.

(3) The clock to the T flip-flop representing bit Q1 is supplied by the output of the T flip-flop representing bit Q0. This ensures that the output of Q1 toggles only when the output of Q0 transitions from 1 to 0.

(4) The clock to the T flip-flop representing bit Q2 is supplied by the output of the T flip-flop representing bit Q1. This ensures that the output of Q2 toggles only when the output of Q1 transitions from 1 to 0.

The circuit realization of the 3-bit binary counter using 3 T flip-flops in an asynchronous sequential circuit is given below.

- The Preset line is used for setting the output of flip-flop, whereas the Clear line is used for resetting the output of flip-flops.
- Here the input line (T) of all flip-flops is tied to logic 1 permanently.

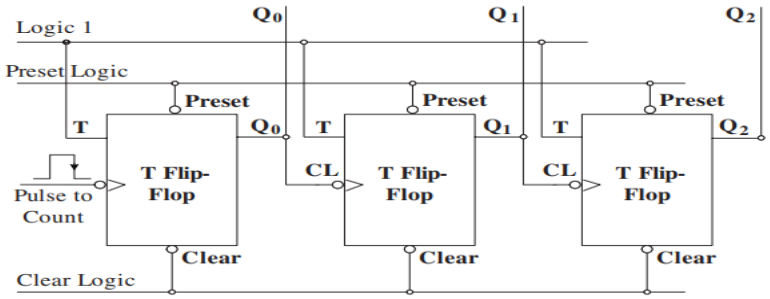


Fig. 8.23 3-Bit Binary Counter Implementation using T Flip-Flop

The table given below summarizes the characteristics, pros and cons of synchronous and asynchronous sequential circuits

Synchronous sequential circuits	Asynchronous sequential circuits
Clocked flip-flops act as the memory element in the circuit. All flip-fl ops are clocked to the same clock signal.	Un-clocked flip-flops or logic gate circuits with feedback loops act as the memory element in the circuit.
The output state of the circuit changes only with clock trigger.	The output state change happens instantaneously with changes in input state
The speed of operation depends on the maximum supported clock frequency.	Faster than synchronous sequential circuits.
Speed is less	speed of operation is high
Costlier compared to asynchronous	cheaper

VLSI AND INTEGRATED CIRCUIT DESIGN

- The first integrated circuit was designed in the 1950s. Depending on the number of integrated components, the degree of integration within an integrated circuit (IC) is known as:
- Small-Scale Integra on (SSI) Integrates one or two logic gate(s) per IC, e.g. LS7400
- Medium-Scale Integra on (MSI) Integrates up to 100 logic gates in an IC. The decade Counter 7490 is an example for MSI device
- Large-Scale Integra on (LSI) Integrates more than 1000 logic gates in an IC
- Very Large-Scale Integra on (VLSI) Integrates millions of logic gates in an IC. Pentium processor is an example of a VLSI Device

Depending on the type of circuits integrated in the IC, the IC design is categorised as:

- **Digital Design**: Deals with the design of integrated circuits handling digital signals and data. Example- Microprocessor/microcontroller, memory
- **Analog Design:** Deals with the design of integrated circuits handling analog signals and data Example- RF IC design, Op-Amp design, voltage regulator IC design.
- **Mixed Signal Design:** Mixed signal design involves design of ICs, which handle both digital and analog signals as well as data. Design of an analog-to-digital converter is an example for mixed signal IC design

VHDL for VLSI Design

- Very High Speed Integrated Circuit HDL or VHDL is a hardware description language used in VLSI design.
- VHDL is a technology independent description, which enables creation of designs targeted for a chosen technology (like CPLD, FPGA, etc.) using synthesis tools. This enables one keep up with the fast development of semiconductor technology.
- VHDL can be used for describing the functionality and behavior of the system (Behavioral representation), or describing the actual gate and register levels of the system [Register Transfer Level (RTL) representation].
- VHDL supports concurrent, sequential, hierarchical and timing modelling.
- VHDL also possess certain set of rules and characteristics. The following table gives a snapshot of the important rules and characteristics specific to VHDL

Type	Representation/Remark
Comment	-- (Start with two adjacent hyphens)
Identifier	Reserved words and programmer chosen names for entity and signal description. Supports any length and characters A-Z, a-z, numbers 0-9 and special character underscore (_). Non-case sensitive. Must start with a character
character	ASCII character in single quote. e.g. 'A', 'a', etc
string	Characters grouped in double quotes, e.g. "VHDL"
Bit strings	Arrays of bits with base Binary (B), Octal (O) or Hexadecimal (X). e.g. B"100" Only 1 and 0 allowed in string O"127" numerals only 0 to 7 are allowed X"1F9" Numerals 0 to 9 and letters A to F, and a to f are allowed in string

Numerals	*universal_integer* does not include points, e.g. 100 *universal_real* include a point, e.g. 100.1 The special character '_' can be used for increasing the readability e.g. 1000 can be represented as 1_000 Character E or e can be used for including exponent, e.g. 1E3 '#' can be used for describing the base of the numbering system, e.g. 2#011# Number base 2 and representing number 3. Base can be between 2 and 16

if statement	**if** condition **then**
	--Corresponding actions
	else
	--Corresponding actions
	end if;

case statement	**case** expression **when** case 0 --statements or case ; **when** case 1 --statements or case 1; --…….. **when others** --statements or default case; **end case**;
Loop statement	**loop** --Body of loop; **end loop;**

while loop	**while** expression **loop** --Body of loop; **end loop;**
for loop	**for** item **in** start item to end item **loop** --Body of loop; **end loop;**
Important Syntax Rules	1. Reserved words are written in bold letters 2. All statements end with a semicolon (;) 3. | is used for separating mutually exclusive alternatives 4. ( ) is used for clarifying the order in which a rule is evaluated 5. {} used for representing optional things that may occur once or many times or never 6. [ ] used for representing optional things that may occur once or never

- The basic structure of a VHDL design consists of an entity, architecture and signals.
- The entity declaration defines the name of the function being modelled and its interface ports to the outside world, their direction and type.

The basic syntax for an entity declaration is given below

```
Entity name-of-entity is
Port (list of interface ports and their types);
Entity item declarations;
Begin
Statements;
End entity name-of-entity
```

- The architecture describes the internal structure and behavior of the model.
  The basic syntax for an architecture declaration is given below

```
Architecture name-of-architecture of name-of-entity is
Begin
Statements;
End architecture name-of-architecture;
```

The library and packages must be specified in the VHDL code as

```
Library name-of-library;
Use name-of-library.name-of-package.ALL;
```

Figure 8.25 illustrates the various steps involved in a HDL based design.

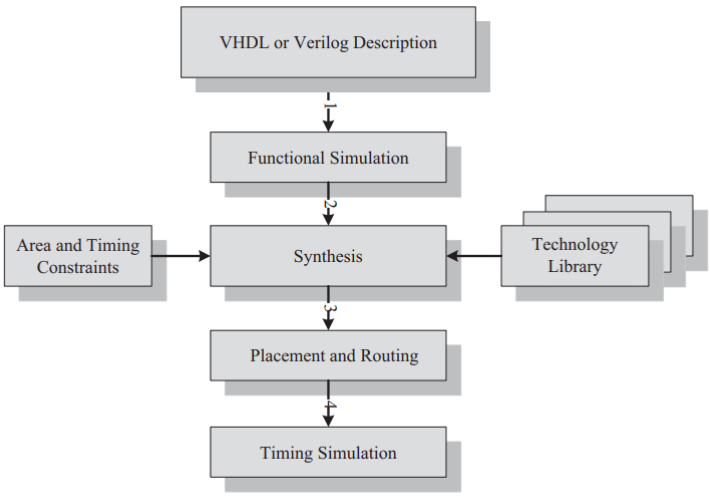


Fig. 8.25 The HDL based VLSI Design Process

ELECTRONIC DESIGN AUTOMATION (EDA) TOOLS

- The designers built the PCB with their hands, oil paper, pencil, pen, ruler and copper plates.
- The more the inter connections involved in the hardware, the more difficult was the process.
- Advances in computer technology and IT brought out highly sophisticated and automated tools for PCB design and fabrication.
- The process of manual sketching the PCB has given way to software packages that gives an automatic routing and layout for your product in a few seconds.
- These software packages are widely known as Electronic Design Automation (EDA) tools.
- EDA tool is a set of Computer Aided Design/Manufacturing (CAD/CAM) software packages which helps in designing and manufacturing the electronic hardware like integrated circuits, printed circuit board, etc.