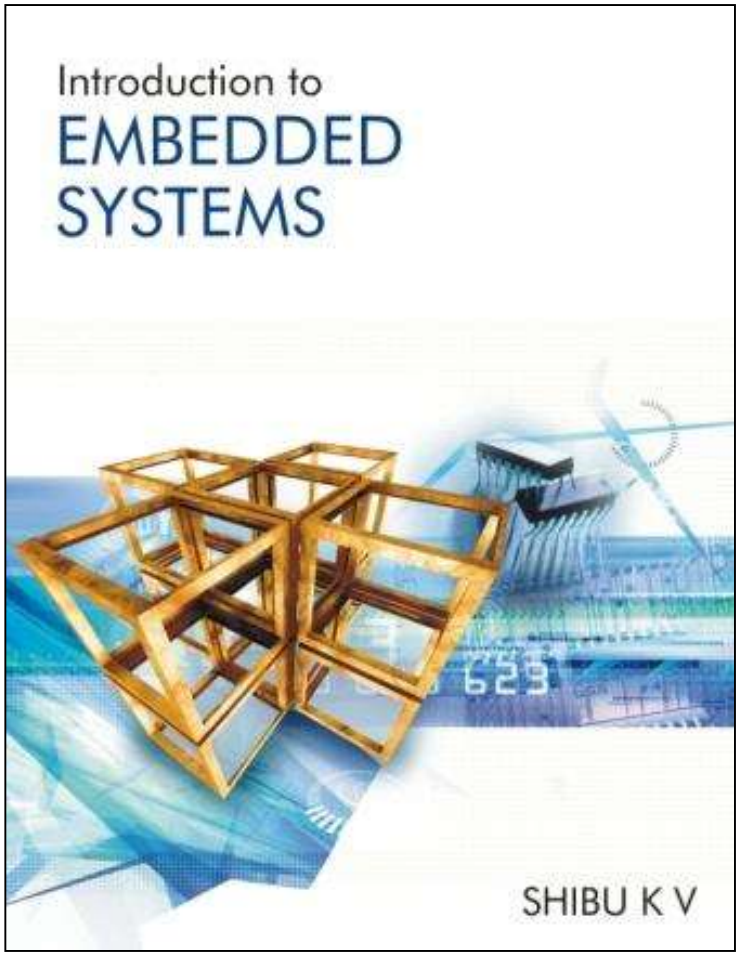
Mc Graw Hill Education



Hardware Software Co-Design and Program Modeling

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, and 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 and once the hardware and software are ready, the integration is performed

Hardware Software Co-design Approach for Embedded System Development

- ✓ The product requirements captured from the customer are converted into system level needs or processing requirements rather than partitioning them to either h/w or s/w
- ✓ 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 this 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

Hardware Software Co-Design and Program Modeling

Fundamental issues in H/w S/w Co-design

- **Model Selection**
  - ✓ A Model captures and describes 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 objectives vary with each phase
- **Architecture Selection**
  - ✓ 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
  - ✓ 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 are the commonly used architectures in system design

Hardware Software Co-Design and Program Modeling

Fundamental issues in H/w S/w Co-design

❑ Language Selection

- ✓ A programming Language captures a 'Computational Model' and maps it into architecture
- ✓ 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
- ✓ 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 of System Requirements into H/w and S/w

- ✓ Implementation aspect of a System level Requirement
- ✓ It may be possible to implement the system requirements in either hardware or software (firmware)
- ✓ Various hardware software trade-offs like performance, re-usability, effort etc are used for making a decision on the hardware-software partitioning

Hardware Software Co-Design and Program Modeling

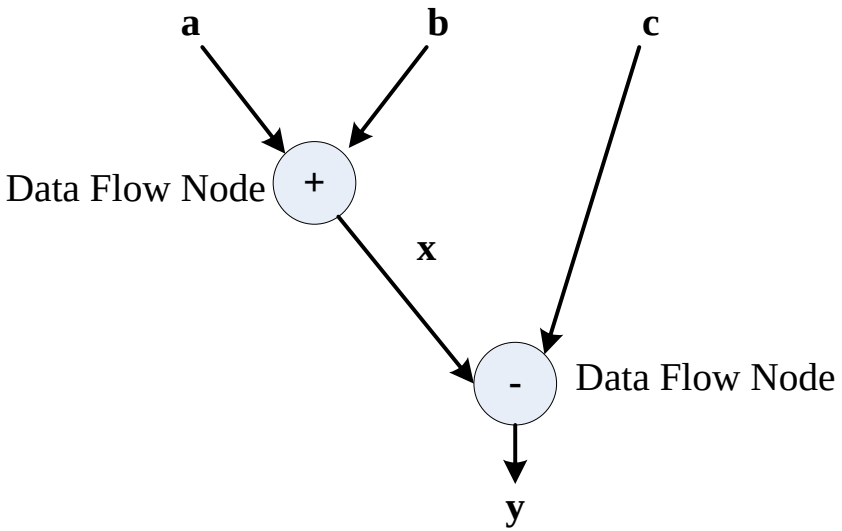
Computational Models in Embedded Design

- **Data Flow Graph/Diagram (DFG) Model**
  - ✓ Translates the data processing requirements into a data flow graph
  - ✓ A data driven model in which the program execution is determined by data.
  - ✓ Emphasizes on the data and operations on the data which transforms the input data to output data.
  - ✓ 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
  - ✓ Best suited for modeling Embedded systems which are computational intensive (like DSP applications)

Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design – Data Flow Graph/Diagram (DFG) Model

E.g. Model the requirement x = a + b; and y = x - c;



Data path: 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 feed back to Input), events etc are examples for non-acyclic inputs. A DFG model translates the program as a single sequential process execution.

Hardware Software Co-Design and Program Modeling

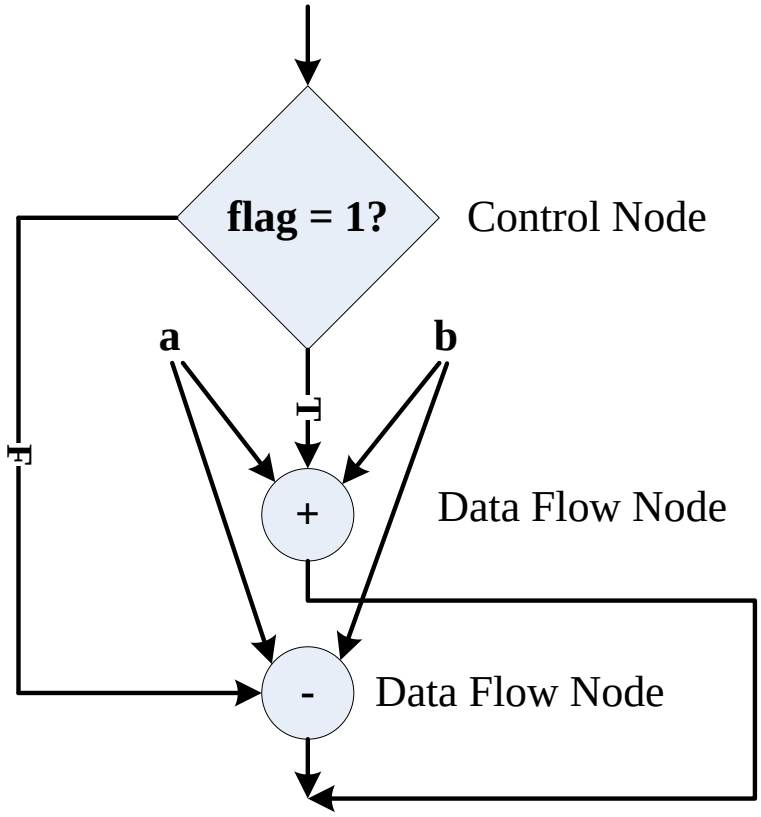
Computational Models in Embedded Design

- **Control Data Flow Graph/Diagram (CDFG) Model**
  - ✓ Translates the data processing requirements into a data flow graph
  - ✓ Model applications involving conditional program execution
  - ✓ Contains both data operations and control operations
  - ✓ 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
  - ✓ 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.
  - ✓ The control node is represented by a 'Diamond' block which is the decision making element in a normal flow chart based design
  - ✓ Translates the requirement, which is modeled to a concurrent process model
  - ✓ The decision on which process is to be executed is determined by the control node
  - ✓ Capturing of image and storing it in the format selected (bmp, jpg, tiff, etc.) in a digital camera is a typical example of an application that can be modeled with CDFG

Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design – Control Data Flow Graph/Diagram (CDFG) Model

E.g. Model the requirement If flag = 1, x = a + b; else y = a-b;



Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design

❑ State Machine Model

- ✓ Based on 'States' and 'State Transition'
- ✓ Describes the system behavior 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.
- ✓ A Finite State Machine (FSM) Model is one in which the number of states are finite. In other words the system is described using a finite number of possible states
- ✓ Most of the time State Machine model translates the requirements into sequence driven program
- ✓ The Hierarchical/Concurrent Finite State Machine Model (HCFSM) is an extension of the FSM for supporting concurrency and hierarchy
- ✓ HCFSM extends the conventional state diagrams by the AND, OR decomposition of States together with inter level transitions and a broadcast mechanism for communicating between concurrent processes
- ✓ HCFSM uses statecharts for capturing the states, transitions, events and actions. The Harel Statechart, UML State diagram etc are examples for popular statecharts used for the HCFSM modeling of embedded systems

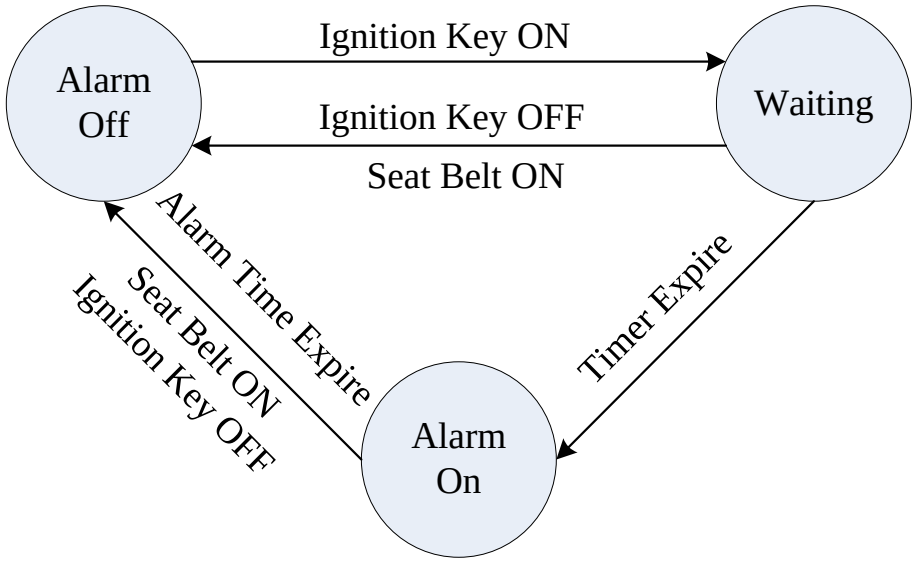
Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design – Finite State Machine (FSM) Model

E.g. Automatic 'Seat Belt Warning' in an automotive

Requirement:

- 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.



Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design

- **Sequential Program Model**
  - ✓ The functions or processing requirements are executed in sequence
  - ✓ 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.
  - ✓ Flow Charts is another important tool used for modeling sequential program
  - ✓ The FSM approach represents the states, events, transitions and actions, whereas the Flow Chart models the execution flow

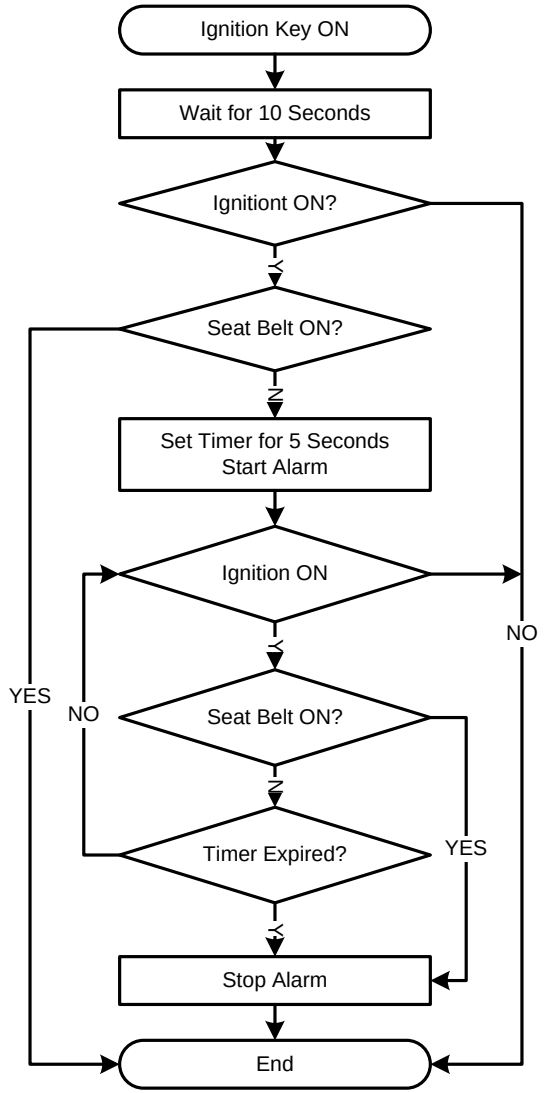
Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design –Sequential Program Model

E.g. Automatic 'Seat Belt Warning' in an automotive

Requirement:

- 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.



Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design –Sequential Program Model

E.g. Automatic 'Seat Belt Warning' in an automotive

```
#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()==OFF )&& (timer_expire()==NO));
stop_alarm();
}
}
}
```

Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design

- **Concurrent/Communicating Process Model**
  - ✓ Models concurrently executing tasks/processes The program instructions are iterated and executed conditionally and the data gets transformed through a series of operations
  - ✓ Certain processing requirements are easier to model 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 utilization, 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, when the subtask under execution goes to a wait or sleep mode, by switching the task execution.
  - ✓ Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication

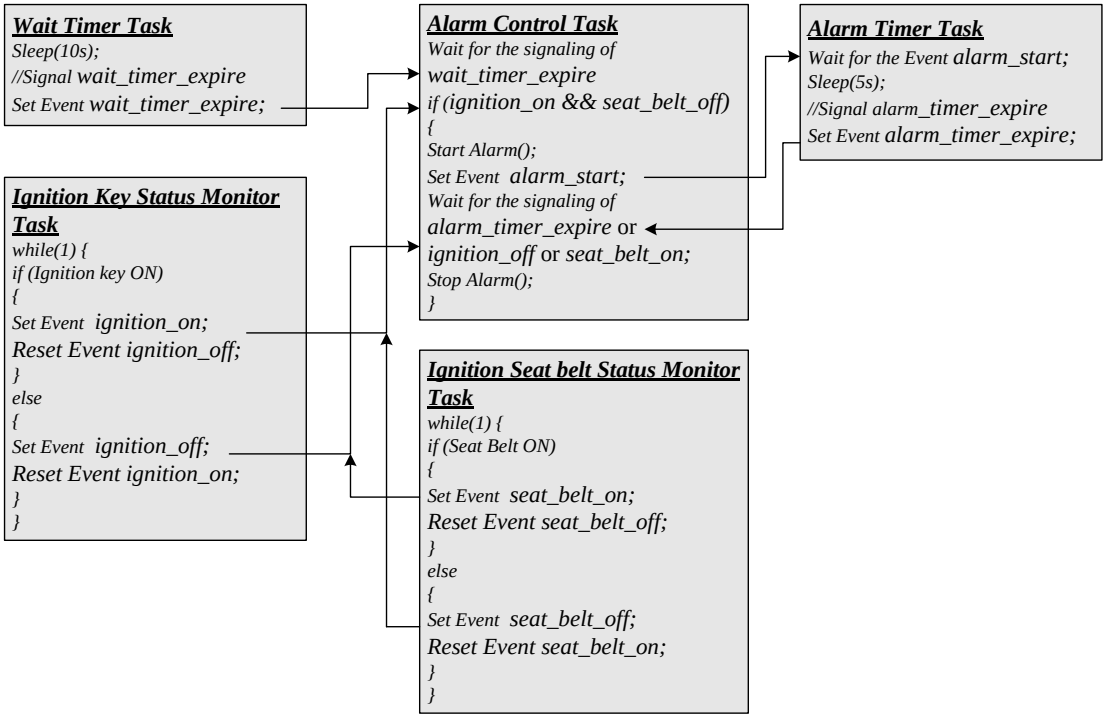
Hardware Software Co-Design and Program Modeling

Computational Models in Embedded Design – Concurrent Processing Model

E.g. Automatic 'Seat Belt Warning' in an automotive

Create and initialize events
wait_timer_expire, ignition_on, ignition_off, seat_belt_on, seat_belt_off, alarm_timer_start, alarm_timer_expire
Create task *Wait Timer*
Create task *Ignition Key Status Monitor*
Create task *Seat Belt Status Monitor*
Create task *Alarm Control*
Create task *Alarm Timer*

(a)



(b)

- ✓ The processing requirements are split in to multiple tasks
- ✓ Tasks are executed concurrently
- ✓ 'Events' are used for synchronizing the execution of tasks

Embedded Hardware Design and Development

Analog Electronic Components

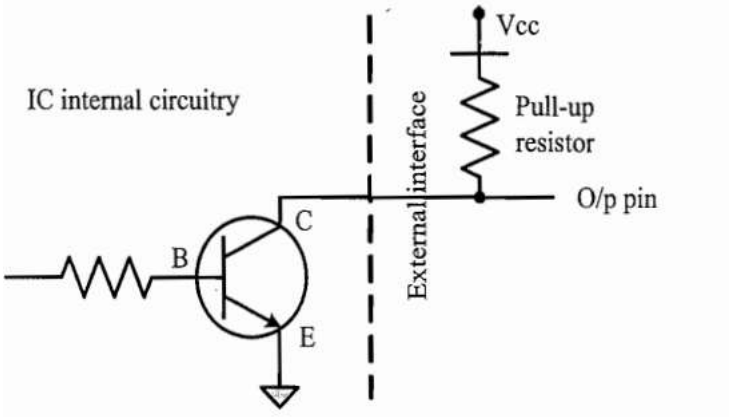
- Resistors, capacitors, diodes, Inductors, operational amplifiers, transistors are used as electronic components in Embedded hardware design.
- Function of each component

Digital Electronic component

- Digital electronics make use of digital or discrete signals.
- Microprocessor/Microcontroller and system on chips, Adders, decoders, latches, encoders/decoders, TTL, CMOS works on digital principle.
- Glue logic – custom digital electronic circuitry required to achieve compatible interface between two different integrated circuit chips.

Open Collector and Tri state output

- The output line from an IC circuit is connected to the base of an NPN transistor.
- The collector is left unconnected and the emitter is internally connected to the ground signal of IC.
- High Impedance state:
- Below figure shows Open collector output configuration.



Advantage of Open Collector Output in embedded system

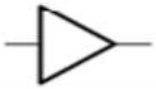
- It eliminates additional interface circuits for connecting devices at different voltage levels
- An open collector configuration supports multi-drop connection.
- It is easy to build Wired AND and OR configuration.

Logic gates

- Logic gates are the building blocks of digital circuits.
- It will control the flow of digital information by performing a logical operation of the input signals.
- Logic gates- AND, OR,XOR,NOT,NAND,NOR,XNOR.
- Truth table-A truth table is a tabular representation of all the combinations of values for inputs and their corresponding outputs

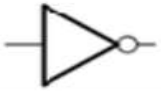
Below figure illustrates the TT and symbolic representation of each logic gates

YES



INPUT	OUTPUT
A	
0	0
1	1

NOT



INPUT	OUTPUT
A	
0	1
1	0

AND

INPUT		OUTPUT
A	B	
0	0	0
1	0	0
0	1	0
1	1	1

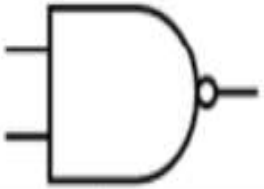
OR

INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	1

XOR

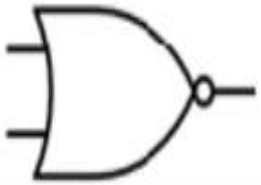
INPUT		OUTPUT
A	B	
0	0	0
1	0	1
0	1	1
1	1	0

NAND



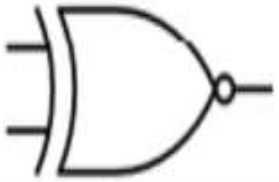
INPUT		OUTPUT
A	B	
0	0	1
1	0	1
0	1	1
1	1	0

NOR



INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	0

XNOR



INPUT		OUTPUT
A	B	
0	0	1
1	0	0
0	1	0
1	1	1

Buffer

- Logic circuit for amplifying the power or current.
- Tri state buffer is a buffer with Output Enable Control.
- When OEC is active tri state buffer acts as a buffer.
- When OEC is not active output of the buffer remains in high impedance state.
- Either unidirectional or bi-directional buffers.
- 74LS244/74HC244 – unidirectional octal buffer
- 74LS245- bi-directional tri state buffer. Allow data flow in both directions, one at a time.
- It contains 8 individual buffers which are grouped into two. Each buffer group has its own output enable line

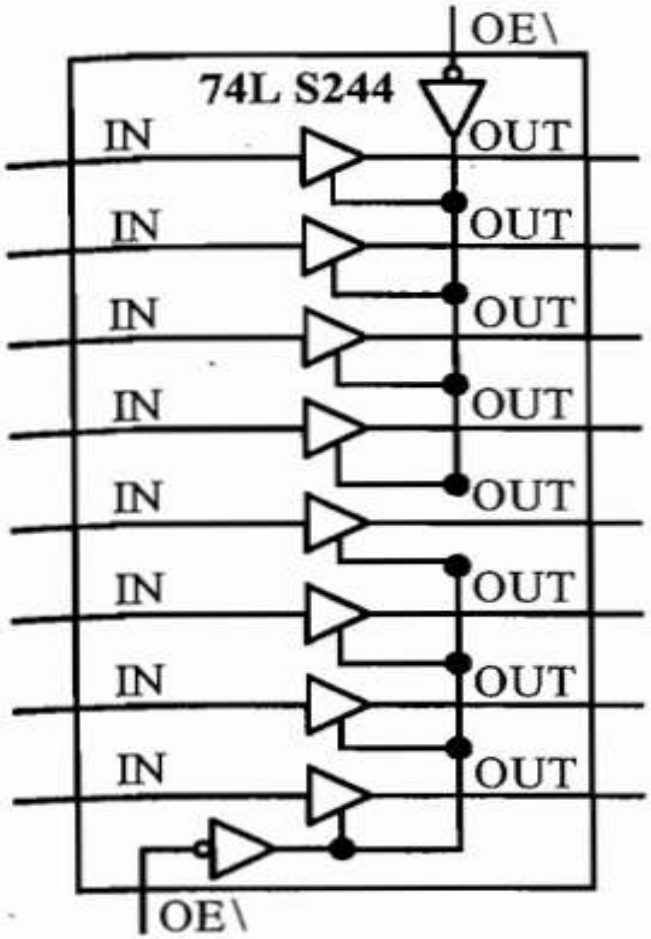


Fig. 8.3 74LS244 Octal Buffer IC

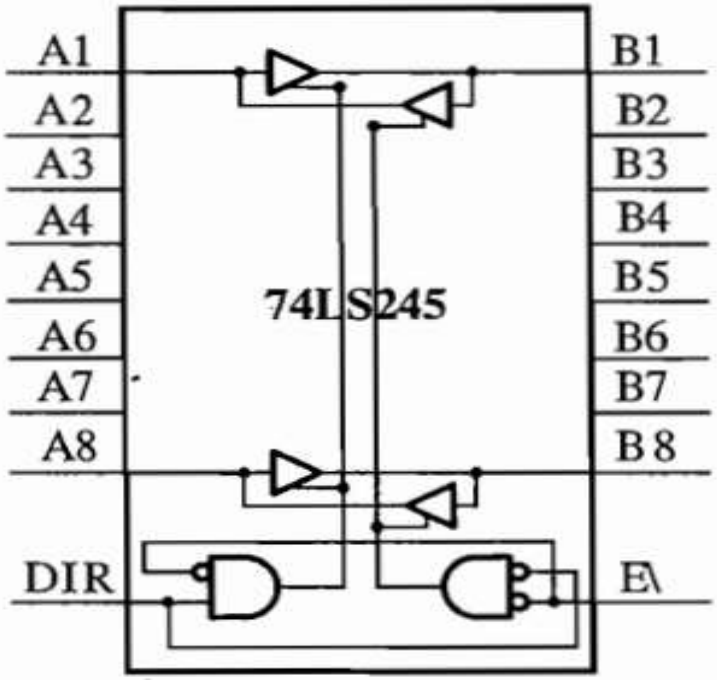


Fig. 8.4 74LS245 Octal bidirectional Buffer IC

Latch

- Stores 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).
- Example- D flip flop.
- Latches are available as integrated circuits, IC 74LS373 contains 8 individual D latches ie used for latching the lower order address byte in a multiplexed address data bus system.
- The Address Latch Enable pulse generated by the processor, when the address bits are available it is used as the latch trigger.
-

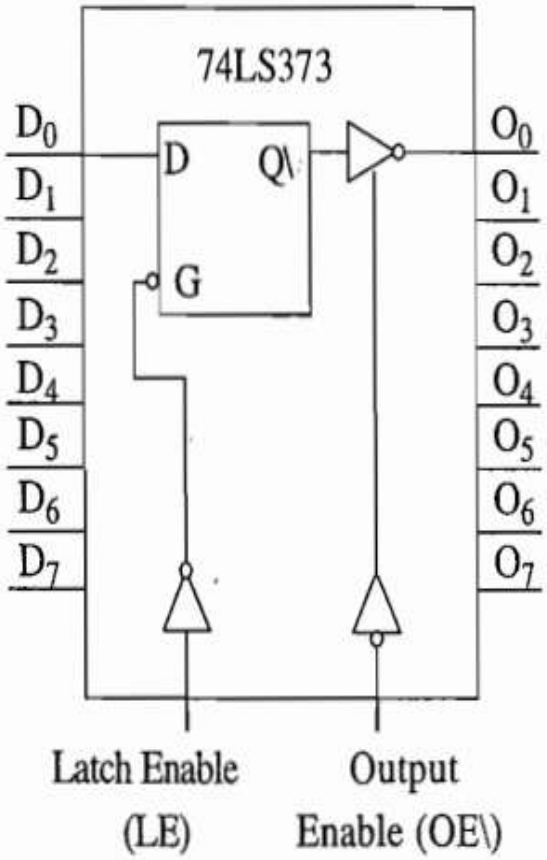


Fig. 8.5 74LS373 Octal Latch IC

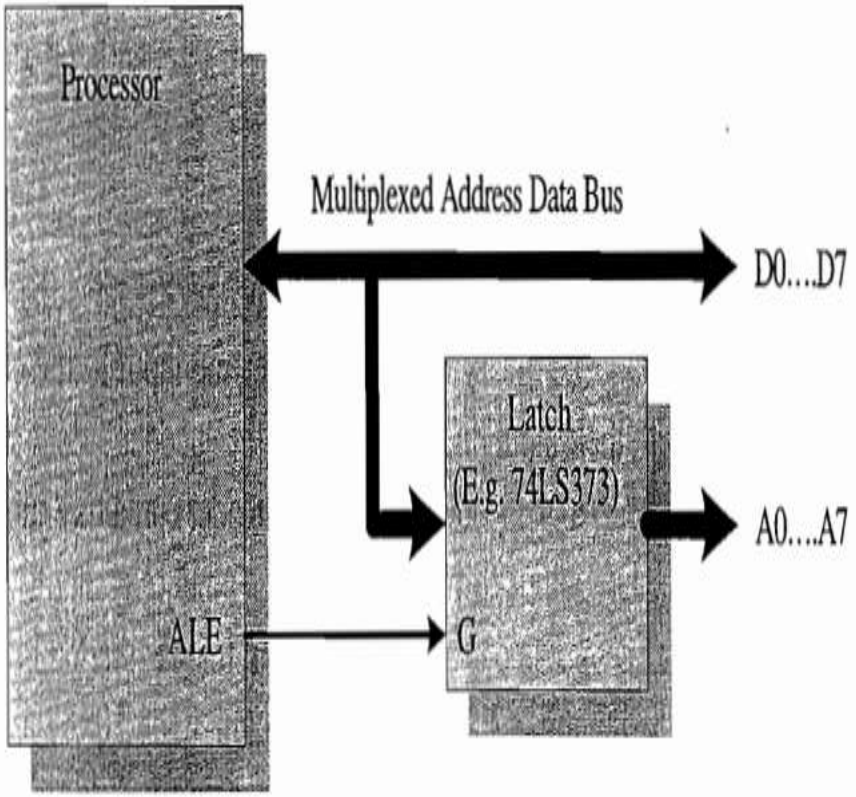
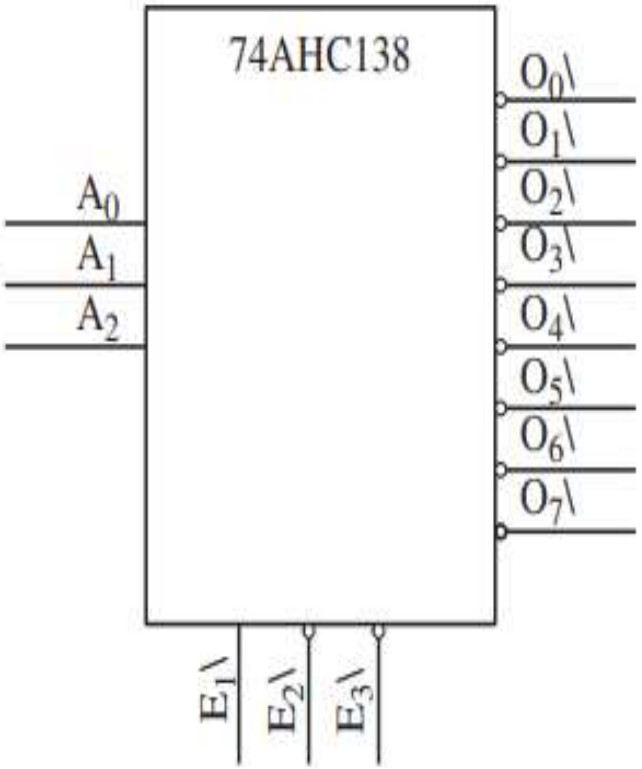


Fig. 8.6 Latch IC for address latching in multiplexed address data-bus

Decoder

- Decoder is a logic circuit which generates all possible combinations of input signals.
- 3 to 8 decoder contains 3 input signal lines and it is possible to have 8 different configurations with the 3 lines.
- Decoders are mainly used for address decoding and chip select signal generation in electronic circuits and are available as integrated circuits.
- Example-74LS138/74AHC138
- The decoder output is enabled only when the output Enable signal lines E1, E2 and E3 are at logic levels 0,0,1.
- If the output enable signals are not at the required logic state all the output lines are forced to the inactive state.
- The output line corresponding to the input state is asserted LOW when the output Enable signal lines are at the required logic state

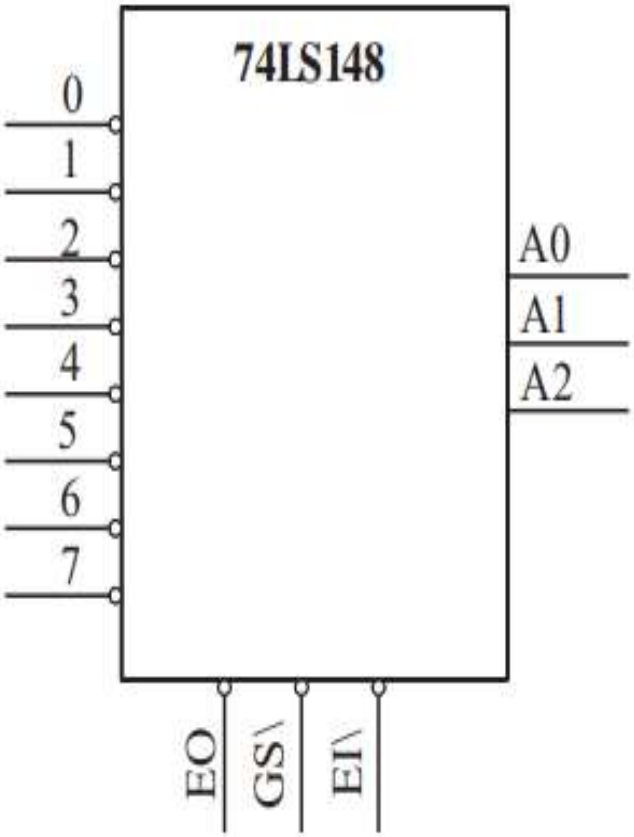


Input						Output							
A_2	A_1	A_0	$E_1\backslash$	$E_2\backslash$	$E_3\backslash$	$O_0\backslash$	$O_1\backslash$	$O_2\backslash$	$O_3\backslash$	$O_4\backslash$	$O_5\backslash$	$O_6\backslash$	$O_7\backslash$
0	0	0	0	0	1	0	1	1	1	1	1	1	1
0	0	1	0	0	1	1	0	1	1	1	1	1	1
0	1	0	0	0	1	1	1	0	1	1	1	1	1
0	1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

- Performs reverse operation of decoder.
- The encoder encodes the corresponding input state to a particular output format.
- The 8 to 3 encoder contains 8 input and it is possible to generate a 3 bit binary output.
- 74F148/74LS148 – 8 to 3 encoder IC
- The encoder output is enabled when the Enable input signal line is at logic 0.
- A high on Enable input forces all outputs to the inactive state. Allows new data to settle without producing erroneous information at the outputs.
- The group signal is active low when any input is low
- The Enable output is active low when all inputs are at logic high.

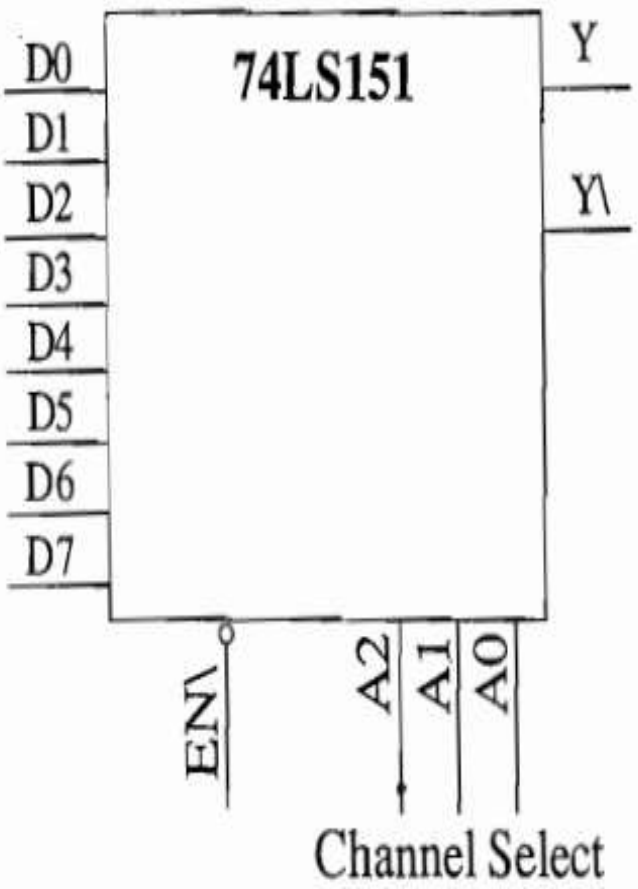


Input									Output				
7	6	5	4	3	2	1	0	EI	GS	EO	A2	A1	A0
1	1	1	1	1	1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	1	0	0	1	1	1	0
1	1	1	1	1	0	1	1	0	0	1	1	0	1
1	1	1	1	0	1	1	1	0	0	1	1	0	0
1	1	1	0	1	1	1	1	0	0	1	0	1	1
1	1	0	1	1	1	1	1	0	0	1	0	1	0
1	0	1	1	1	1	1	1	0	0	1	0	0	1
0	1	1	1	1	1	1	1	0	0	1	0	0	0

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

Multiplexer

- Digital switch which connects one input line from a set of input lines, to an output line at a given point of time.
- It is enabled only when the “Enable signal” line is at logic 0.
- High on the EN line forces the output to the low state.
- The input signal is switched to the output line through the channel select control lines A2,A1 and A0.



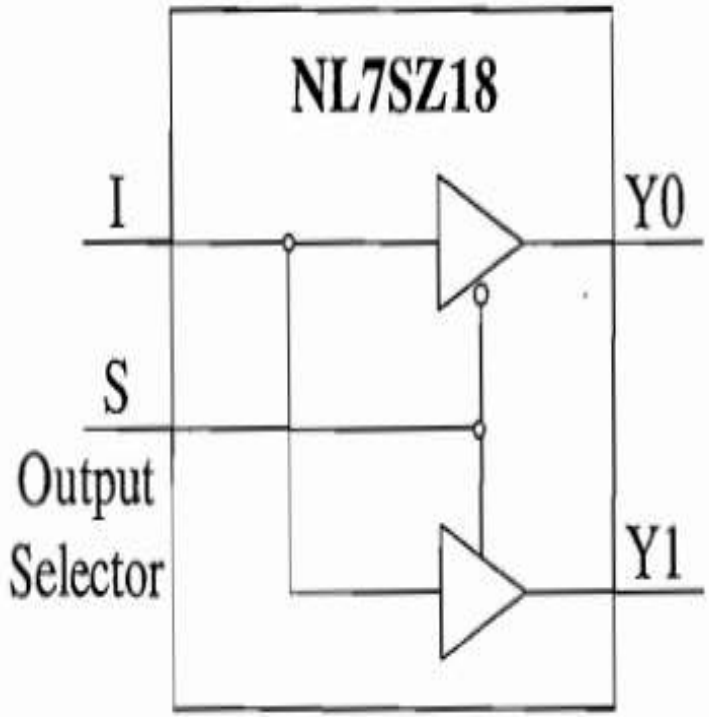
Input				Output	
A2	A1	A0	EN\	Y	Y\
0	0	0	0	D0	D0\
0	0	1	0	D1	D1\
0	1	0	0	D2	D2\
0	1	1	0	D3	D3\
1	0	0	0	D4	D4\
1	0	1	0	D5	D5\
1	1	0	0	D6	D6\
1	1	1	0	D7	D7\
x	x	x	1	0	1

x: Don't care

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

De- multiplexer(D-MUX)

- It performs reverse operation of multiplexer.
- The output line to which the input is to be switched is selected by the output selector control lines.
- NL7SZ18 – example for 1 to 2 D- Mux IC.
- When one output line is selected by the output selector control(S), the other output line remains in the high impedance state(The signal is left open).



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**

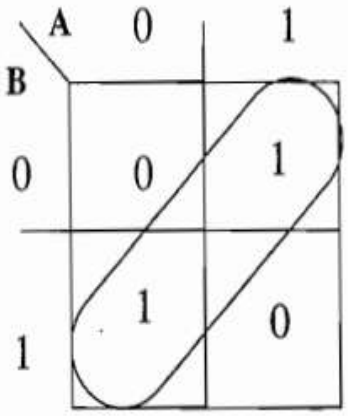
Combinational circuits

- a combinational circuit is a circuit in which the output depends on the present combination of inputs. Combinational circuits are made up of logic gates. The output of each logic gate is determined by its logic function.
- Adder, Subtractor, Converter, and Encoder/Decoder, MUX/D-Mux
- K-map, Algebraic method, Variable entered mapping , Quine- McCluskey method used to simplify the logic expression.
- Logical function is represented as either SOP(Sum of Products) or $Y = AB + \overline{B}C + AC$ s(POS) form
- Ex -> SOP - $Y = (A+B)(B+C)(A+C)$
- Ex -> POS -

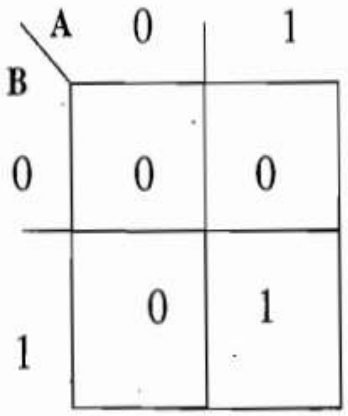
K-map is the easiest technique to simplify the logical expression when the TT was given.

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.

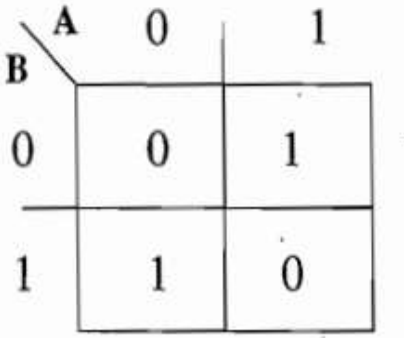
$$Y = \bar{A}B + A\bar{B}$$

$$C = AB$$

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**

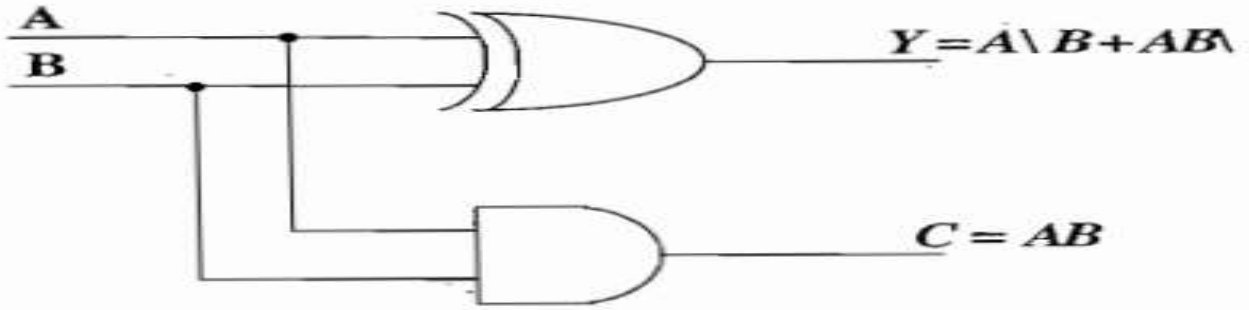


Fig. 8.13 **Half Adder Circuit Implementation**

Sequential circuits

- Output depends on both present and past input.
- Memory element holds the previous input states.
- Flipflop- basic building blocks of sequential circuits.
- Two types of SC are synchronous(clocked) and asynchronous sequential circuits.
- Example of synchronous sequential circuits is synchronous counters.
- Example of asynchronous sequential circuits is Ripple counters.

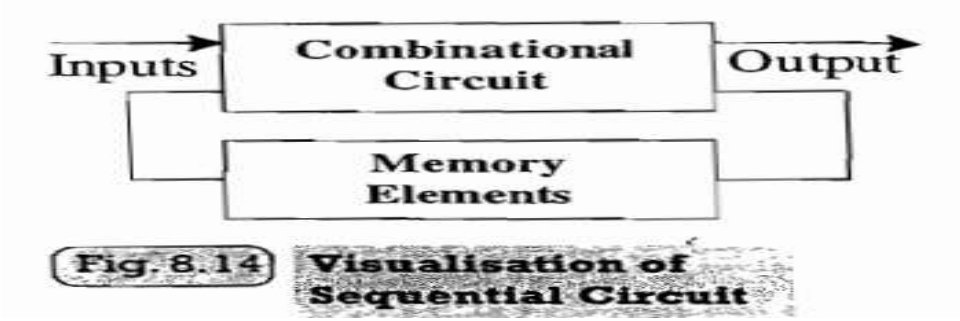
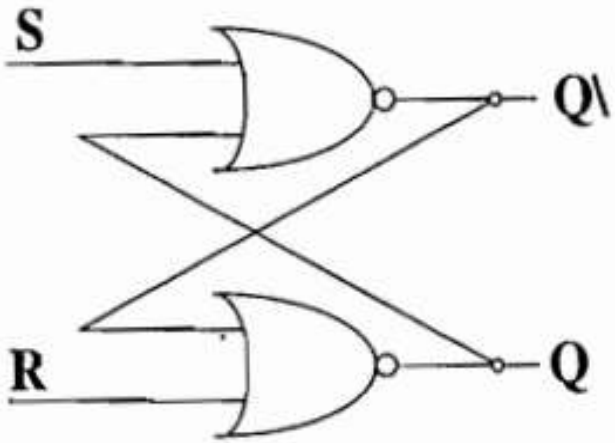


Fig. 8.14 Visualisation of Sequential Circuit

S-R flipflop



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

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 (i.e., if the clock signal is at logic 1 for 'HIGH' level triggered and at logic 0 for 'LOW' level triggered clock signal). For edge triggered flip-flops, the output state changes only when a clock trigger happens regardless of the changes in the input signal state. The clock trigger signal can be either a positive edge (A 0 to 1 transition) or a negative edge (A 1 to 0 transition).

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.

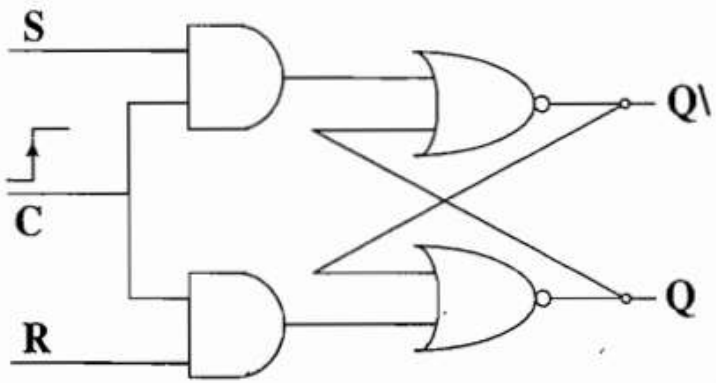
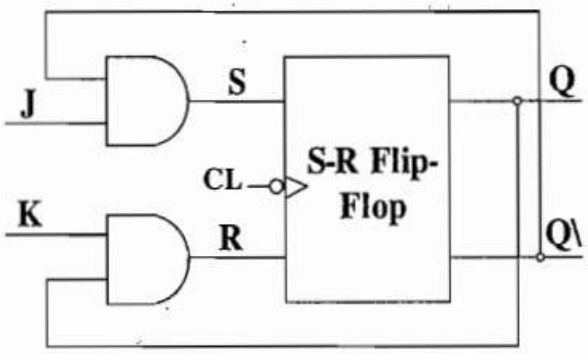


Fig. 8.16 Clocked S-R Flip-Flop

J-K flipflop

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.

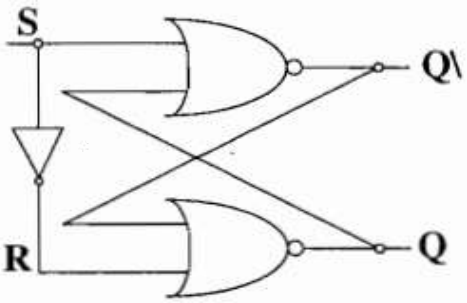


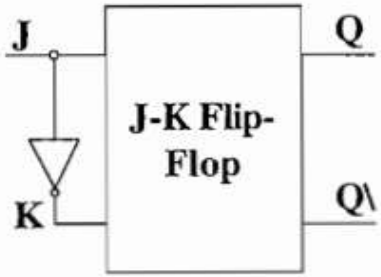
Input		Output	
J	K	Previous State	Present State
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

J-K Flip-Flop Implementation and Truth Table

D- Flipflop

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.





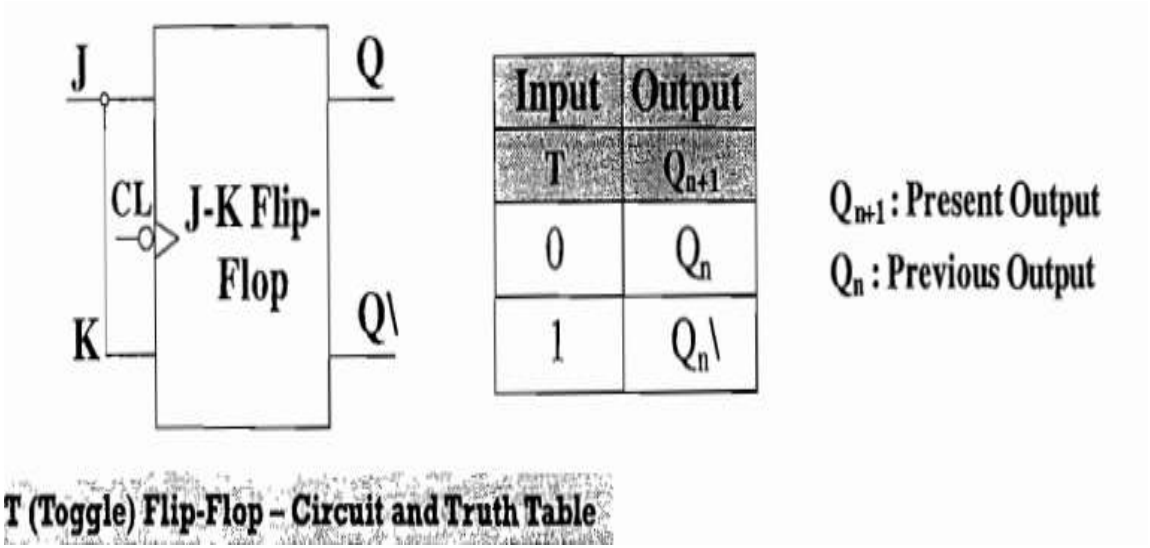
Input	Output
D	Q
0	0
1	1

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

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).

T- flipflop

- T flipflop is formed by combining the J and K inputs of a J-K flipflop.
- When the T input is held at logic 1, T flipflop toggles the output with each clock signal.



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

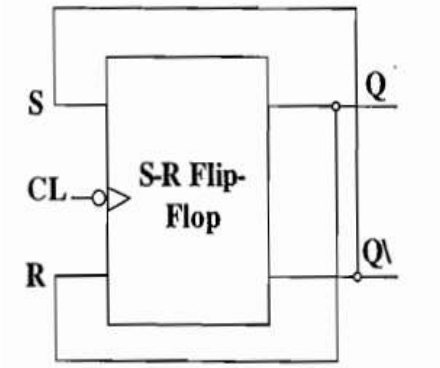
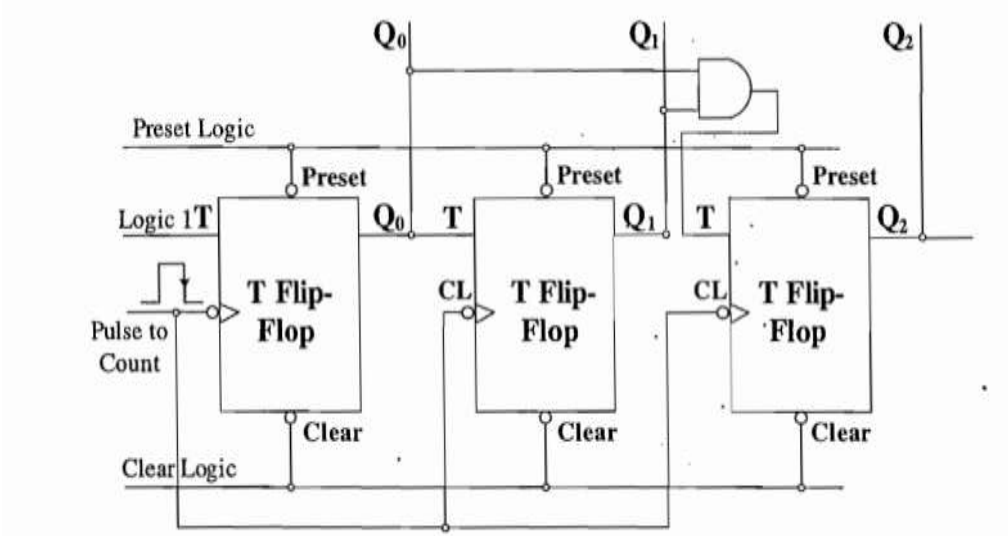


Fig. 8.20 S-R Flip-Flop acting as Toggle switch

3 bit Binary counter



1) Binary Counter Implementation using T Flip-Flops

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

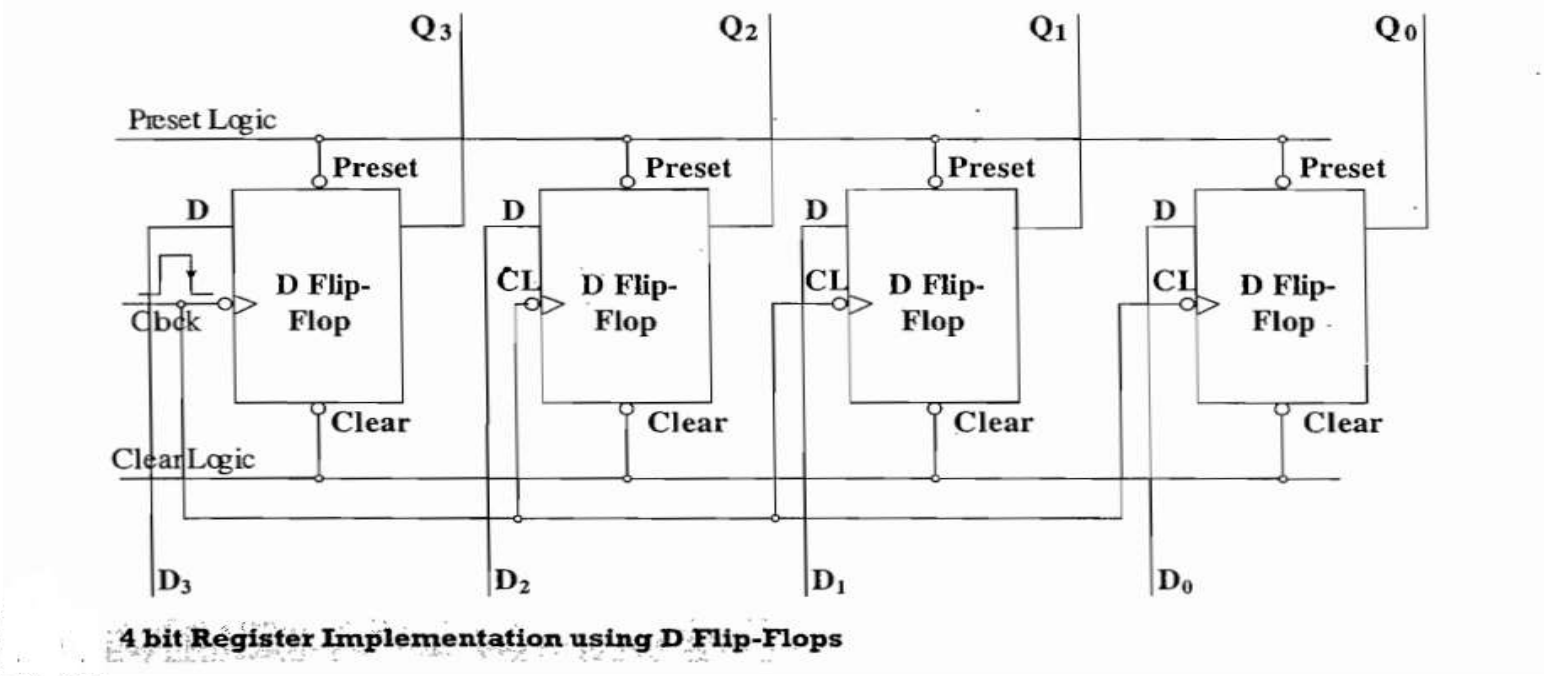
From the count sequence, it is clear that the LS bit (Q_0) of the binary counter toggles on each count and the next LS bit (Q_1) toggles only when the LS bit (Q_0) makes a transition from 1 to 0. The Bit (Q_2) of the binary counter toggles its state only when the Q_1 bit and Q_0 bits are at logic 1.

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 Q_0 bit is initially set at 0. The input line of Q_0 flip-flop is connected to logic 1 to ensure toggling of the output with input clock signal.
3. Since Q_1 bit changes only when Q_0 makes a transition from 1 to 0, the output of the T flip-flop representing Q_0 is fed as input to the T flip-flop representing the Q_1 bit.
4. The output bit Q_2 changes only when $Q_0 = Q_1 = 1$. Hence the input to the flip-flop representing bit Q_2 is fed by logically ANDing Q_0 and Q_1.

Synchronous Sequential Circuit	Asynchronous Sequential Circuit
Clocked flip-flops act as the memory element in the circuit. All flip-flops 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.

4 bit register using D flipflop

Another example for synchronous sequential circuit is 'register'. A register can be considered as a group of bits holding information. A D flip-flop can be used for holding a 'bit' information. An 8 bit wide register can be visualised as the combination of 8 D flip-flops. The bit storing operation is controlled by the signal associated with a latch write (like a write to latch pulse). The figure given below



4 bit Register Implementation using D Flip-Flops

VLSI AND INTEGRATED CIRCUIT DESIGN

- Small-Scale Integration (SSI) Integrates one or two logic gate(s) per IC, e.g. LS7400
- Medium-Scale Integration (MSI) Integrates up to 100 logic gates in an IC. The decade Counter 7490 is an example for MSI device.
- Large-Scale Integration (LSI) Integrates more than 1000 logic gates in an IC.
- Very Large-Scale Integration (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

- 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;**

-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 of architecture body is given below

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

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.