ABOUT SRI SHASHA PRAYATHI TECHNOLOGIES:

Sri Shasha Prayathi Technologies is incubated by the National Institute of Technology Karnataka Science and

Technology Entrepreneurship Park (NITK-STEP). We offer an Internship Program in the recent trending areas of Electronics and Communication Engineering for undergraduate and post-graduate students.

Prerequisites: Digital Logic Design

COURSE OUTLINE:

WEEK 1 AND WEEK 2:

- Introduction to VLSI design
- Types of design flows (ASIC, FPGA)
- Introduction to Verilog
- Types of modeling (Data Flow, Structural and Behavioral)

WEEK 3 AND WEEK 4:

- Introduction to Combinational and Sequential circuits
- Basic components Verilog coding for Combinational and Sequential circuits.

WEEK 5 :

- VLSI Signal processing basics
- Pipelining concepts on basic FIR digital filters.

MINI-PROJECTS (WEEK 6):

- Mini project-1 (on Digital circuits)
- Mini project-2 (on Digital module designing of Signal processing algorithm)
- Submission of Report to concerned college.

LEARNING OUTCOMES (LO):

Week 1 and 2:

Describe VLSI process flow of creating an IC.

To learn Verilog HDL and illustrate modeling types

Week 3 and 4:

To learn basic elements in Verilog coding.

Week 5 and week 6:

Mini-Projects understanding.

ACTIVITY LOG FOR THE FIRST WEEK

DAY	BRIEF DESCRIPTION OF THE DAILY	LEARNING OUTCOME	SIGNATURE OF THE FACULTY
Day-1	Basics of Digital Design	Recalculation of Digital Design	
Day-2	Introduction to VLSI design	Introducing the concepts of VLSI	
Day-3	Introduction to VLSI design	Describe VLSI process flow of creating an IC	
Day-4	Introduction to VLSI design	Describe VLSI process flow of creating an IC	
Day-5	VLSI Design - FPGA Technology	Understand FPGA Technology	
Day-6	Introduction to ASIC Design	Understand ASIC Design	

INTRODUCTION TO VLSI DESIGN

Very-large-scale integration (VLSI) is the process of creating an integrated circuit (IC) by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. The microprocessor is a VLSI device.

Before the introduction of VLSI technology, most ICs had a limited set of functions they could perform. An electronic circuit might consist of a CPU, ROM, RAM and other glue logic. VLSI lets IC designers add all of these into one chip.

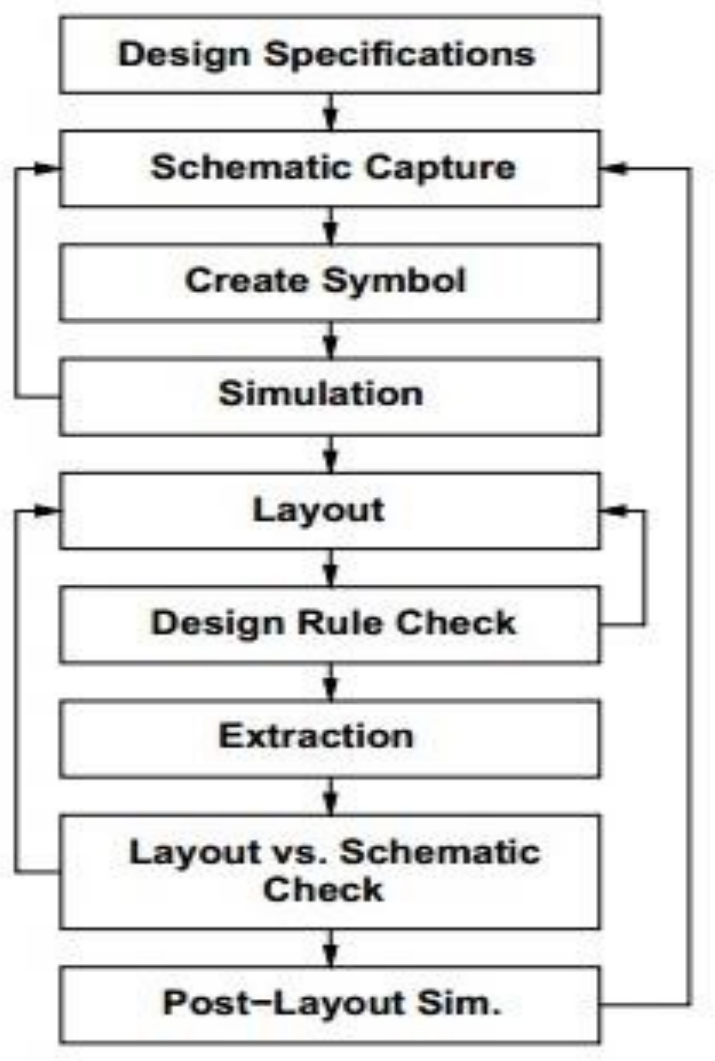
The electronics industry has achieved a phenomenal growth over the last few decades, mainly due to the rapid advances in large scale integration technologies and system design applications. With the advent of very large scale integration (VLSI) designs, the number of applications of integrated circuits (ICs) in high-performance computing, controls, telecommunications, image and video processing, and consumer electronics has been rising at a very fast pace.

The current cutting-edge technologies such as high resolution and low bit-rate video and cellular communications provide the end-users a marvelous amount of applications, processing power and portability. This trend is expected to grow rapidly, with very important implications on VLSI design and systems design.

VLSI DESIGN FLOW

The VLSI IC circuits design flow is shown in the figure below. The various levels of design are numbered and the blocks show processes in the design flow.

Specifications comes first, they describe abstractly, the functionality, interface, and the architecture of the digital IC circuit to be designed.



Behavioral description is then created to analyze the design in terms of functionality, performance, compliance to given standards, and other specifications.

RTL description is done using HDLs. This RTL description is simulated to test functionality. From here onwards we need the help of EDA tools.

RTL description is then converted to a gate-level netlist using logic synthesis tools. A gatelevel netlist is a description of the circuit in terms of gates and connections between them, which are made in such a way that they meet the timing, power and area specifications.

Finally, a physical layout is made, which will be verified and then sent to fabrication.

Y CHART

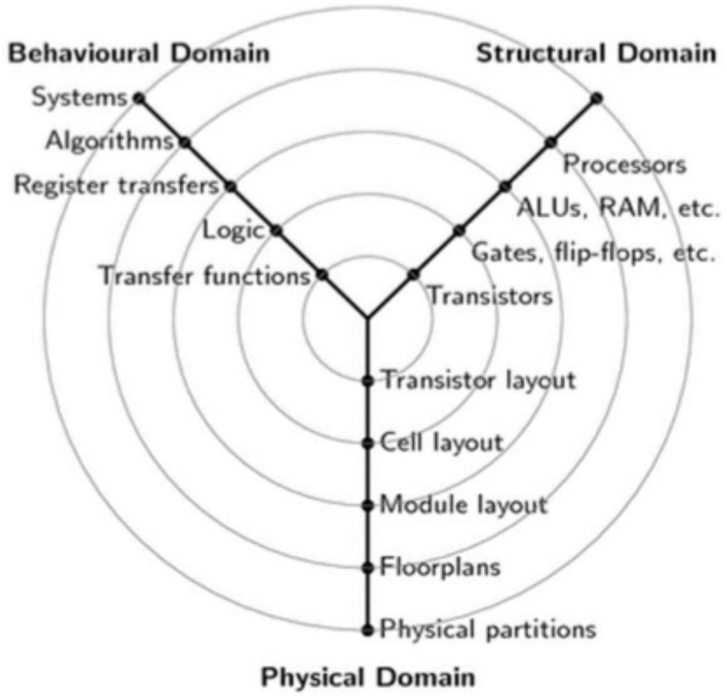
The Gajski-Kuhn Y-chart is a model, which captures the considerations in designing semiconductor devices.

The three domains of the Gajski-Kuhn Y-chart are on radial axes. Each of the domains can be divided into levels of abstraction, using concentric rings.

At the top level (outer ring), we consider the architecture of the chip; at the lower levels (inner rings), we successively refine the design into finer detailed implementation −

Creating a structural description from a behavioral one is achieved through the processes of high-level synthesis or logical synthesis.

Creating a physical description from a structural one is achieved through layout synthesis.

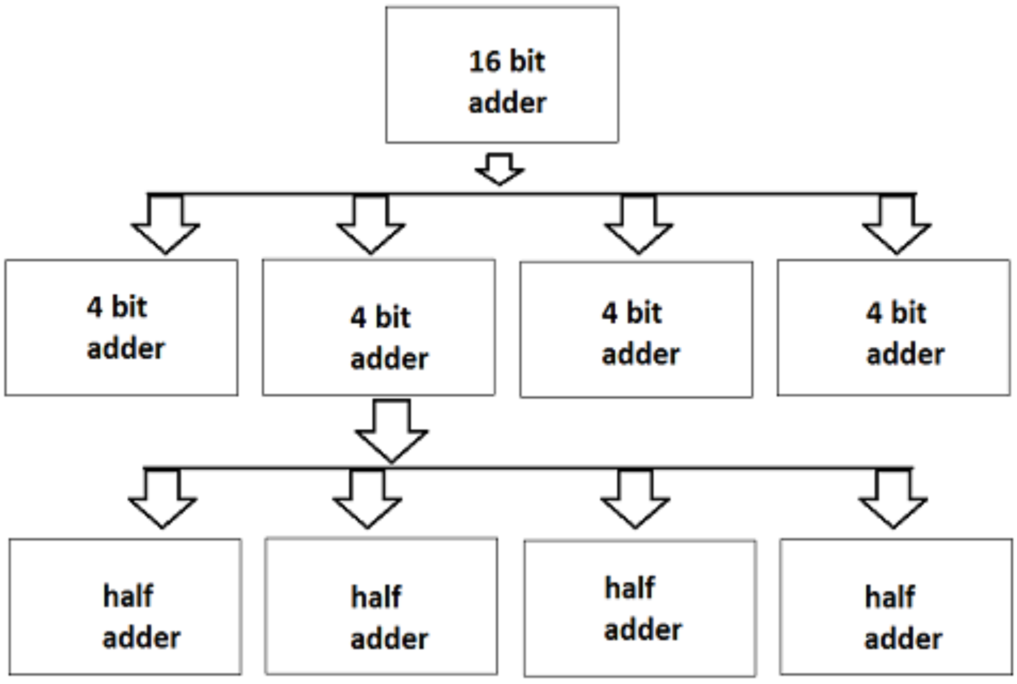


Gajski-Kuhn Y-chart

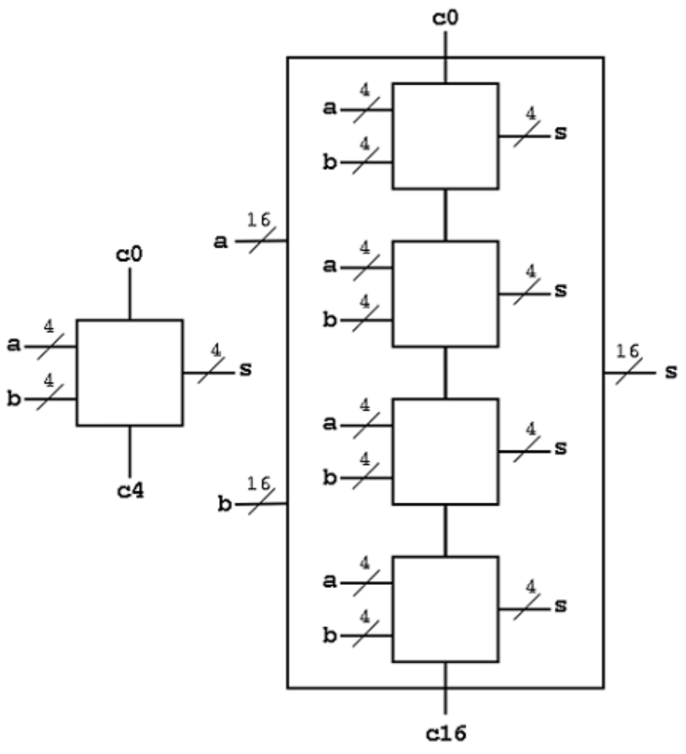
DESIGN HIERARCHY-STRUCTURAL

The design hierarchy involves the principle of "Divide and Conquer." It is nothing but dividing the task into smaller tasks until it reaches to its simplest level. This process is most suitable because the last evolution of design has become so simple that its manufacturing becomes easier.

We can design the given task into the design flow process's domain (Behavioral, Structural, and Geometrical). To understand this, let’s take an example of designing a 16-bit adder, as shown in the figure below.



Here, the whole chip of 16-bit adder is divided into four modules of 4-bit adders. Further, dividing the 4-bit adder into 1-bit adder or half adder. 1 bit addition is the simplest designing process and its internal circuit is also easy to fabricate on the chip. Now, connecting all the last four adders, we can design a 4-bit adder and moving on, we can design a 16-bit adder.



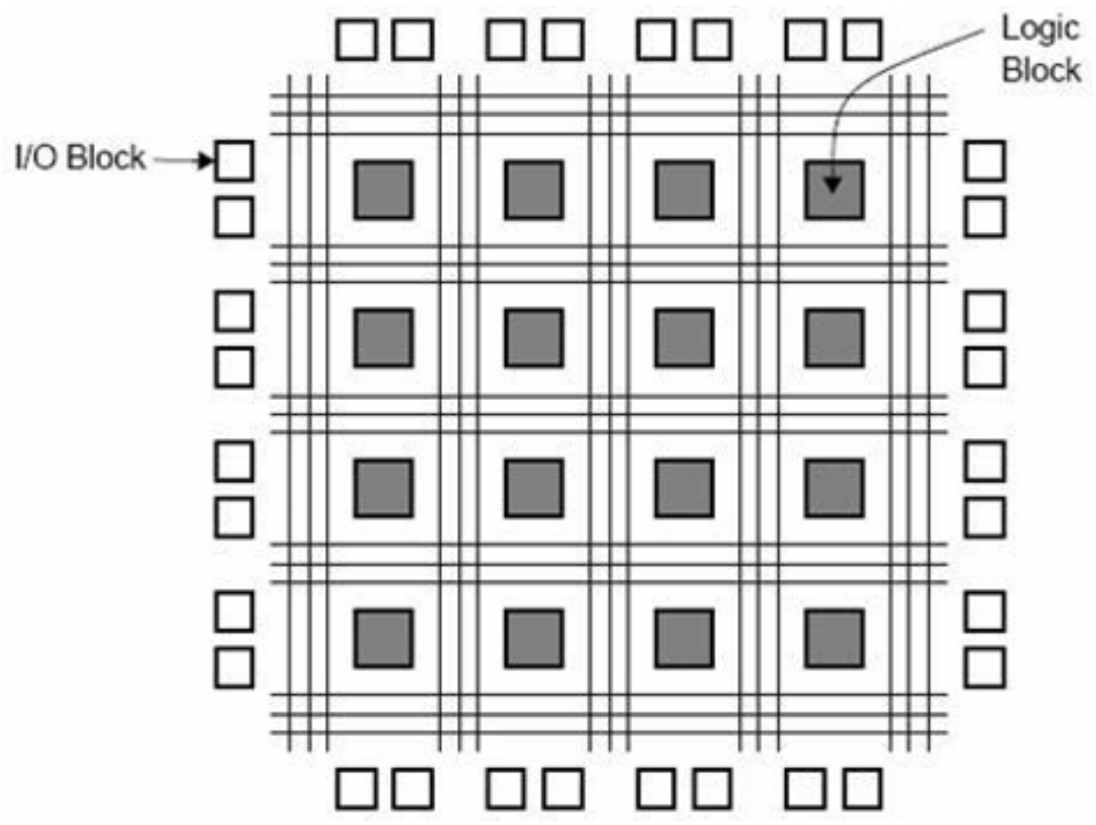
VLSI DESIGN - FPGA TECHNOLOGY

FPGA – INTRODUCTION

The full form of FPGA is “Field Programmable Gate Array”. It contains ten thousand to more than a million logic gates with programmable interconnection. Programmable interconnections are available for users or designers to perform given functions easily. A typical model FPGA chip is shown in the given figure. There are I/O blocks, which are designed and numbered according to function. For each module of logic level composition, there are CLB’s (Configurable Logic Blocks).

CLB performs the logic operation given to the module. The inter connection between CLB and I/O blocks are made with the help of horizontal routing channels, vertical routing channels and PSM (Programmable Multiplexers).

The number of CLB it contains only decides the complexity of FPGA. The functionality of CLB’s and PSM are designed by VHDL or any other hardware descriptive language. After programming, CLB and PSM are placed on chip and connected with each other with routing channels.



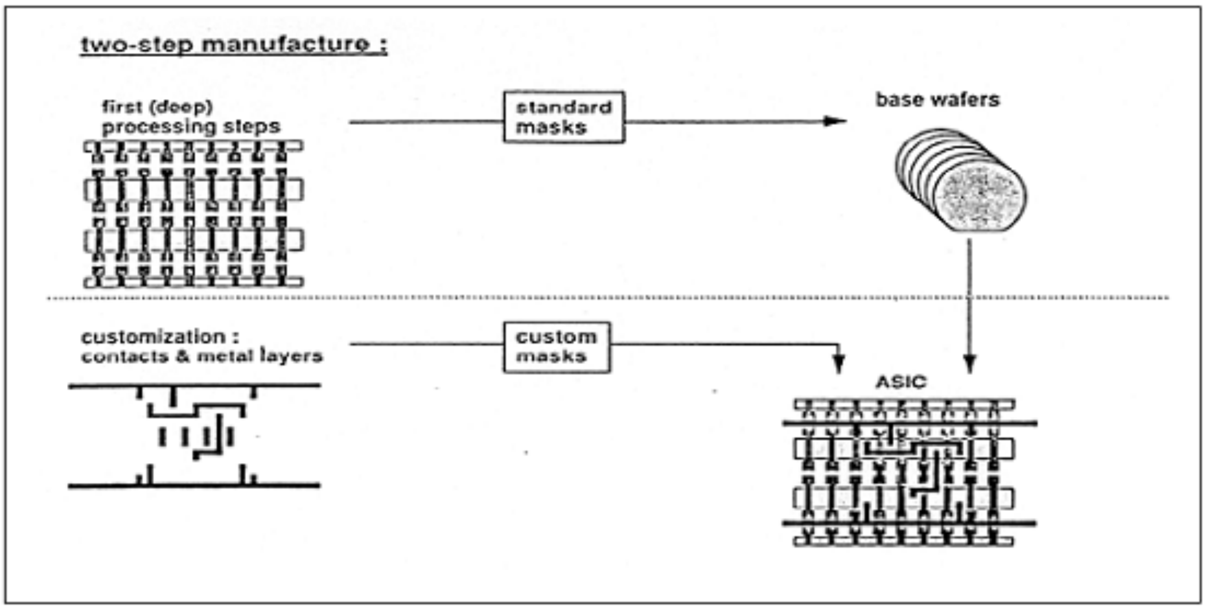
Advantages

- It requires very small time; starting from design process to functional chip.
- No physical manufacturing steps are involved in it.
- The only disadvantage is, it is costly than other styles.

GATE ARRAY DESIGN

The **gate array (GA)** ranks second after the FPGA, in terms of fast prototyping capability. While user programming is important to the design implementation of the FPGA chip, metal mask design and processing is used for GA. Gate array implementation requires a two-step manufacturing process.

The first phase results in an array of uncommitted transistors on each GA chip. These uncommitted chips can be stored for later customization, which is completed by defining the metal interconnects between the transistors of the array. The patterning of metallic interconnects is done at the end of the chip fabrication process, so that the turn-around time can still be short, a few days to a few weeks. The figure given below shows the basic processing steps for gate array implementation.

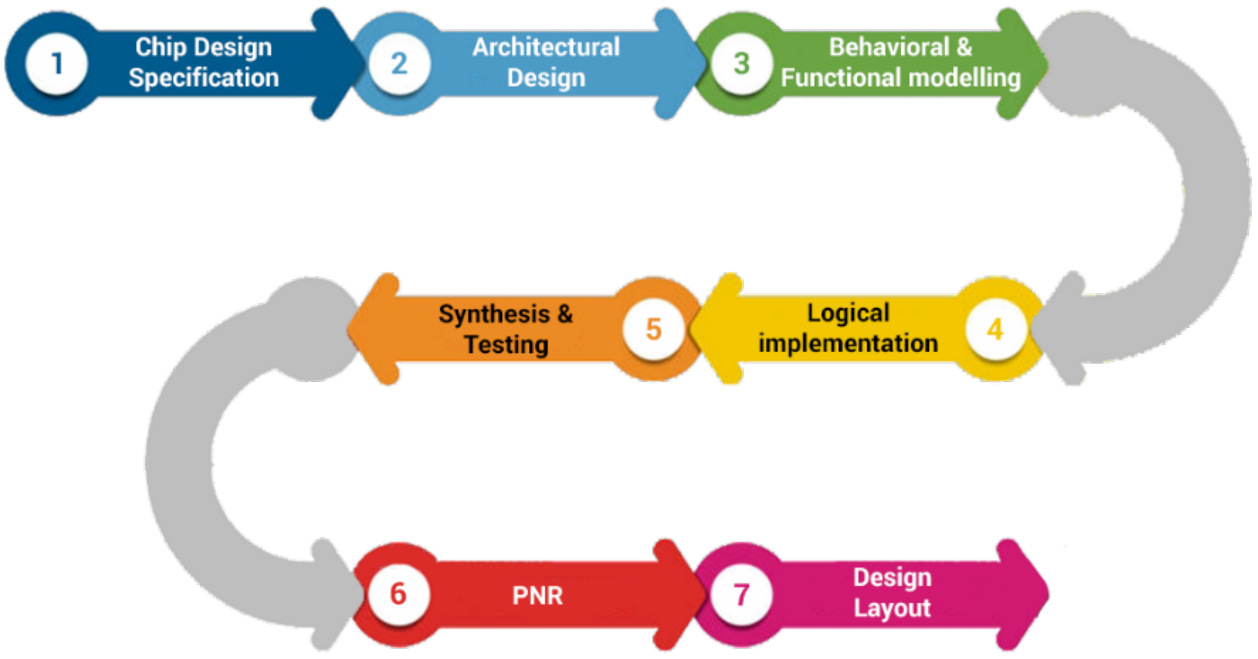

two-step manufacture :
first (deep) processing steps
standard masks
base wafers
customization : contacts & metal layers
custom masks
ASIC

Typical gate array platforms use dedicated areas called channels, for inter-cell routing between rows or columns of MOS transistors. They simplify the interconnections. Interconnection patterns that perform basic logic gates are stored in a library, which can then be used to customize rows of uncommitted transistors according to the netlist.

In most of the modern GAs, multiple metal layers are used for channel routing. With the use of multiple interconnected layers, the routing can be achieved over the active cell areas; so that the routing channels can be removed as in Sea-of-Gates (SOG) chips. Here, the entire chip surface is covered with uncommitted nMOS and pMOS transistors. The neighboring transistors can be customized using a metal mask to form basic logic gates.

For inter cell routing, some of the uncommitted transistors must be sacrificed. This design style results in more flexibility for interconnections and usually in a higher density. GA chip utilization factor is measured by the used chip area divided by the total chip area. It is higher than that of the FPGA and so is the chip speed.

ASIC-INTRODUCTION

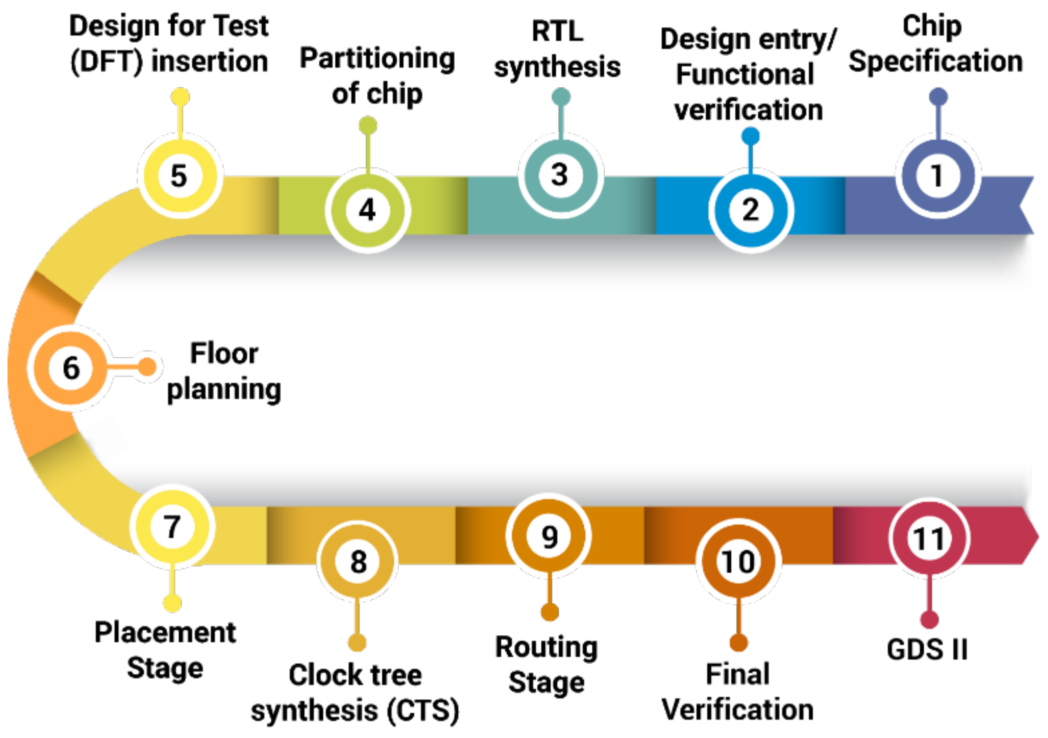


To ensure successful ASIC design, engineers must follow a proven ASIC design flow which is based on a good understanding of ASIC specifications, requirements, low power design and performance, with a focus on meeting the goal of right time to market. Every stage of ASIC design cycle has EDA tools that can help to implement ASIC design with ease.

HOW DOES THE ASIC DESIGN CYCLE WORK?

In order to fulfill futuristic demands of chip design, changes are required in design tools, methodologies, and software/hardware capabilities. For those changes, ASIC design flow adopted by engineers for efficient structured ASIC chip architecture and focus on its design functionalities

ASIC design flow is a mature and silicon-proven IC design process which includes various steps like design conceptualization, chip optimization, logical/physical implementation, and design validation and verification. Let's have an overview of each of the steps involved in the process.



Step 1. Chip Specification

This is the stage at which the engineer defines features, microarchitecture, functionalities (hardware/software interface), specifications (Time, Area, Power, Speed) with design guidelines of ASIC. Two different teams are involved at this juncture.

Step 2. Design Entry / Functional Verification

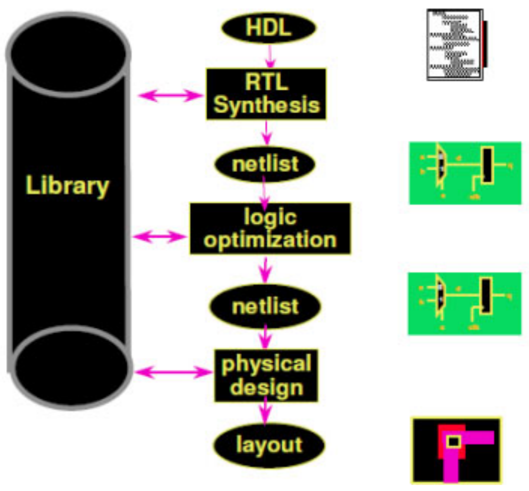
Functional verification confirms the functionality and logical behavior of the circuit by simulation on a design entry level. This is the stage where the design team and verification team come into the cycle where they generate RTL code using test-benches. This is known as behavioral simulation.

In this simulation, once the RTL code (RTL code is a set of code that checks whether the RTL implementation meets the design verification) is done in HDL, a lot of code coverage metrics proposed for HDL. Engineers aim to verify correctness of the code with the help of test vectors and trying to achieve it by 95% coverage test. This code coverage includes statement coverage, expression coverage, branch coverage, and toggle coverage.

There are two types of simulation tools:

- Functional simulation tools: After the testbench and design code, functional simulation verifies logical behavior and its implementation based on design entry.
- Timing simulation tools: Verifies that circuit design meets the timing requirements and confirms the design is free of circuit signal delays.

Step 3. RTL block synthesis / RTL Function



Once the RTL code and testbench are generated, the RTL team works on RTL description – they translate the RTL code into a gate-level netlist using a logical synthesis tool that meets required timing constraints. Thereafter, a synthesized database of the ASIC design is created in the system. When timing constraints are met with the logic synthesis, the design proceeds to the design for testability (DFT) techniques.

Step 4. Chip Partitioning

This is the stage wherein the engineer follows the ASIC design layout requirement and specification to create its structure using EDA tools and proven methodologies. This design structure is going to be verified with the help of HLL programming languages like C++ or System C.

After understanding the design specifications, the engineers partition the entire ASIC into multiple functional blocks (hierarchical modules), while keeping in mind ASIC's best performance, technical feasibility, and resource allocation in terms of area, power, cost and time. Once all the functional blocks are implemented in the architectural document, the engineers need to brainstorm ASIC design partitioning by reusing IPs from previous projects and procuring them from other parties.

Step 5. Design for Test (DFT) Insertion

With the ongoing trend of lower technology nodes, there is an increase in system-on-chip variations like size, threshold voltage and wire resistance. Due to these factors, new models and techniques are introduced to high-quality testing.

ASIC design is complex enough at different stages of the design cycle. Telling the customers that the chips have fault when you are already at the production stage is embarrassing and disruptive. It's a situation that no engineering team wants to be in. In order to overcome this situation, design for test is introduced with a list of techniques:

- Scan path insertion: A methodology of linking all registers elements into one long shift register (scan path). This can help to check small parts of design instead of the whole design in one go.
- Memory BIST (built-in Self-Test): In the lower technology node, chip memory requires lower area and fast access time. MBIST is a device which is used to check RAMs. It is a comprehensive solution to memory testing errors and self-repair proficiencies.

Step 6. Floor Planning (blueprint your chip)

After, DFT, the physical implementation process is to be followed. In physical design, the first step in RTL-to-GDSII design is floorplanning. It is the process of placing blocks in the chip. It includes: block placement, design portioning, pin placement, and power optimization.

Floorplan determines the size of the chip, places the gates and connects them with wires. While connecting, engineers take care of wire length, and functionality which will ensure signals will not interfere with nearby elements. In the end, simulate the final floor plan with post-layout verification process.

A good floorplanning exercise should come across and take care of the below points; otherwise, the life of IC and its cost will blow out:

- Minimize the total chip area
- Make routing phase easy (routable)

- Improve signal delays

Step 7. Placement

Placement is the process of placing standard cells in row. A poor placement requires larger area and also degrades performance. Various factors, like the timing requirement, the net lengths and hence the connections of cells, power dissipation should be taken care. It removes timing violation.

Step 8. Clock tree synthesis

Clock tree synthesis is a process of building the clock tree and meeting the defined timing, area and power requirements. It helps in providing the clock connection to the clock pin of a sequential element in the required time and area, with low power consumption.

In order to avoid high power consumption, increase in delays and a huge number of transitions, certain structures can be used for optimizing CTS structure such as Mesh Structure, H-Tree Structure, X-Tree Structure, Fishbone Structure and Hybrid structure. With the help of these structures, each flop in the clock tree gets the clock connection. During the optimization, tools insert the buffer to build the CTS structure. Different clock structures will build the clock tree with a minimum buffer insertion and lower power consumption of chips.

ACTIVITY LOG FOR THE SECOND WEEK

DAY	BRIEF DESCRIPTION OF THE DAILY ACTIVITY	LEARNING OUTCOME	SIGNATURE OF THE FACULTY
Day-1	Introduction to Verilog	Understand Verilog Basics	
Day-2	Introduction to Verilog	Learn Verilog HDL	
Day-3	Introduction to Verilog	Learn Verilog HDL	
Day-4	Types of modeling (Data Flow, Structural and Behavioral)	Illustrate modeling types	
Day-5	Types of modeling (Data Flow, Structural and Behavioral)	Illustrate modeling types	
Day-6	Types of modeling (Data Flow, Structural and Behavioral)	Illustrate modeling types	

TYPES OF MODELING (DATA FLOW, STRUCTURAL AND BEHAVIORAL)

DATA FLOW MODELING

Dataflow modeling makes use of the functions that define the working of the circuit instead of its gate structure.

Dataflow modeling has become a popular design approach, as logic synthesis tools became sophisticated. This approach allows the designer to focus on optimizing the circuit in terms of the flow of data.

Dataflow modeling uses several operators that act on operands to produce the desired results. Verilog provides about 30 operator types. Dataflow modeling describes hardware in terms of the flow of data from input to output.

The dataflow modeling style is mainly used to describe combinational circuits. The primary mechanism used is a continuous assignment.

Continuous Assignments

A value is assigned to a data type called net, which is used to represent a physical connection between circuit elements in a continuous assignment. The value assigned to the net is specified by an expression that uses operands and operators.

A continuous assignment replaces gates in the circuit's description and describes the circuit at a higher level of abstraction. A continuous assignment statement starts with the keyword assign.

Syntax

The syntax of a continuous assignment is

1. assign [delay] LHS_net = RHS_expression;

LHS_net is a destination net of one or more bit, and RHS_expression is an expression of various operators.

The statement is evaluated at any time any of the source operand value changes, and the result is assigned to the destination net after the delay unit.

- The LHS of the assign statement must always be a scalar or vector net or a concatenation. It cannot be a register.
- Continuous statements are always active statements, which means that if any value on the RHS changes, LHS changes automatically.
- Registers or nets or function calls can come in the RHS of the assignment.
- The RHS expression is evaluated whenever one of its operands changes. Then the result is assigned to the LHS.
- Delays can be specified in the assign statement.

Example

1. assign out1 = in1 & in2; // perform and function on in1 and in2 and assign the result to out1
2. assign out2 = not in1;
3. assign #2 z[0] = ~(ABAR & BBAR & EN); // perform the desired function and assig n the result after 2 units

The target in the continuous assignment expression can be one of the following:

1. A scalar net
2. Vector net
3. Constant bit-select of a vector
4. Constant part-select of a vector
5. Concatenation of any of the above

Let us take another set of examples in which a scalar and vector nets are declared and used

```
wire COUNT, CIN;               // scalar net declaration
wire [3:0] SUM, A, B;          // vector nets declaration
assign {COUT,SUM} = A + B + CIN;           // A and B vectors are added with CIN
```

Continuous Assignment on Vectors

As described in the characteristics, the continuous assignment can be performed on vector nets.

```
module adder(a,b,sum);
input [2:0] a,b;
output [3:0] sum;

assign sum = a + b;
$display("a = %b, b = %b, sum=%b", a,b,sum);
endmodule
```

The above code describes a 3-bit adder. The MSB of the sum is dedicated to carry in the above module. It generates the following output:

```
a = 100, b = 111, sum = 1011                    // (a = 4, b = 7, sum = 011, carry = 1)
```

The concatenation of vector and scalar nets is also possible. The same example for 3-bit adder is shown by using concatenation:

```
module adder(a,b,sum);
input [2:0] a,b;
output [2:0] sum; //sum is a vector
output carry; // carry is a scalar

assign {carry,sum} = a + b; // assigning result to a concatenation of scalar and vector
```

7. $display("a = %b, b = %b, sum=%b, carry = %b", a,b,sum,carry);
8. endmodule

The output is:

```
a = 100, b = 111, sum = 011, carry = 1
```

1. Regular Continuous Assignment

It follows the following steps, such as:

Step 1: Declare net.

Step 2: Write a continuous assignment on the net.

The below code follows Regular continuous assignment:

```
wire out; // net 'out' is declared
assign out = a&b; //continuous assignment on declared net
```

2. Implicit Continuous Assignment

We can also place a continuous assignment on a net when it is declared. The format will look like the below:

```
wire out = a & b; // net declaration and assignment together
```

3. Implicit Net Declaration

In Verilog, during an implicit assignment, if LHS is declared, it will assign the RHS to the declared net, but if the LHS is not defined, it will automatically create a net for the signal name.

```
Wire in0, in1;
Assign out = in0 ^ in1;
```

In the above example, out is undeclared, but Verilog makes an implicit net declaration for out.

Behavioral Modeling :

Behavioral models in Verilog contain procedural statements, which control the simulation and manipulate variables of the data types. These all statements are contained within the procedures. Each of the procedure has an activity flow associated with it.

During simulation of behavioral model, all the flows defined by the 'always' and 'initial' statements start together at simulation time 'zero'. The initial statements are executed once, and the always statements are executed repetitively. In this model, the register variables a and b are initialized to binary 1 and 0 respectively at simulation time 'zero'. The initial statement is then completed and is not executed again during that simulation run. This initial statement is containing a begin-end block (also called a sequential block) of statements. In this begin-end type block, a is initialized first followed by b.

Example of Behavioral Modeling

```
module behave;
reg [1:0]a,b;

initial
begin
  a = 'b1;
  b = 'b0;
end

always
begin
  #50 a = ~a;
end

always
begin
  #100 b = ~b;
end
End module
```

Structural model:

A structural model is a description of a circuit at the abstraction level of logic gates. This type of model could be seen as a textual representation of a schematic. Modelling at such a low-level is to be contrasted with dataflow modelling and behavioural modelling which are done at a higher level of abstraction.

Example

Consider a half adder which adds bits A and B giving the sum S and carry C. The truth table below gives the specification:

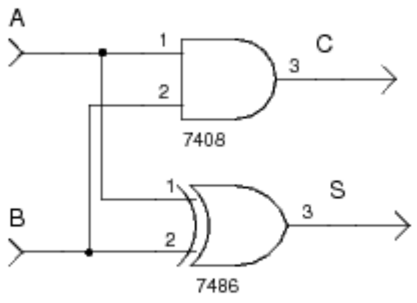
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0

1 1 0 1

The simplified equations for the half adder are:

- S = A xor B
- C = A and B

In schematic form this is:



Below is a Verilog structural model which shows just how closely a schematic and structural model match each other. In the code, the first argument to xor and and is the gate output, the other arguments are gate inputs.

```
module half_adder(S, C, A, B);
output S;output C;
input  A;input  B;xor(S, A, B);
and(C, A, B);
endmodule
```

ACTIVITY LOG FOR THE THIRD WEEK

DAY	BRIEF DESCRIPTION OF THE DAILY ACTIVITY	LEARNING OUTCOME	SIGNATURE OF THE FACULTY
Day-1	Introduction to Combinational circuits	Understand to Combinational circuits	
Day-2	Introduction to Combinational circuits	Study various Combinational circuits	
Day-3	Introduction to Combinational circuits	Study various Combinational circuits	
Day-4	Basic components Verilog coding for Combinational circuits	Learn basic elements in Verilog coding	
Day-5	Verilog coding for Combinational circuits	Implement Combinational circuits	
Day-6	Verilog coding for Combinational circuits	Implement Combinational circuits	

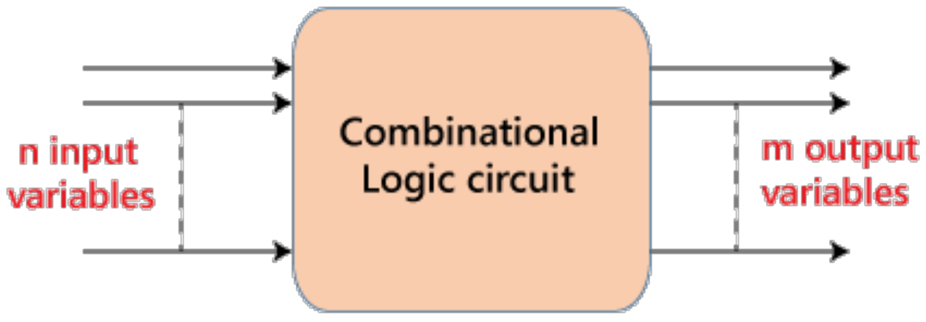
INTRODUCTION TO COMBINATIONAL CIRCUITS

Combinational Logic circuits

The combinational logic circuits are the circuits that contain different types of logic gates. Simply, a circuit in which different types of logic gates are combined is known as a **combinational logic circuit**. The output of the combinational circuit is determined from the present combination of inputs, regardless of the previous input. The input variables, logic gates, and output variables are the basic components of the combinational logic circuit. There are different types of combinational logic circuits, such as Adder, Subtractor, Decoder, Encoder, Multiplexer, and De-multiplexer.

There are the following characteristics of the combinational logic circuit:

- At any instant of time, the output of the combinational circuits depends only on the present input terminals.
- The combinational circuit doesn't have any backup or previous memory. The present state of the circuit is not affected by the previous state of the input.
- The n number of inputs and m number of outputs are possible in combinational logic circuits.



Block diagram of a combinational circuit

The 'n' input variable comes from the external source while the 'm' output variable goes to the external destination. In many applications, the source or destinations are storage registers.

Half Adder

The half adder is a basic building block having two inputs and two outputs. The adder is used to perform OR operation of two single bit binary numbers. The **carry** and **sum** are two output states of the half adder.

Full Adder

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed. The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Half Subtractors

The half subtractor is also a building block of subtracting two binary numbers. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The **'diff'** and **'borrow'** are the two output state of the half adder.

Full Subtractors

The Half Subtractor is used to subtract only two numbers. To overcome this problem, full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are **minuend, subtrahend**, and **borrow,** respectively. The full subtractor has three input states and two output states i.e., diff and borrow.

Multiplexers

The multiplexer is a combinational circuit that has n-data inputs and a single output. It is also known as the **data selector** which selects one input from the inputs and routes it to the output. With the help of the selected inputs, one input line from the n-input lines is selected. The enable input is denoted by E, which is used in cascade.

De-multiplexers

A De-multiplexer performs the reverse operation of a multiplexer. The de-multiplexer has only one input, which is distributed over several outputs. One output line is selected at a time by selecting lines. The input is transmitted to the selected output line.

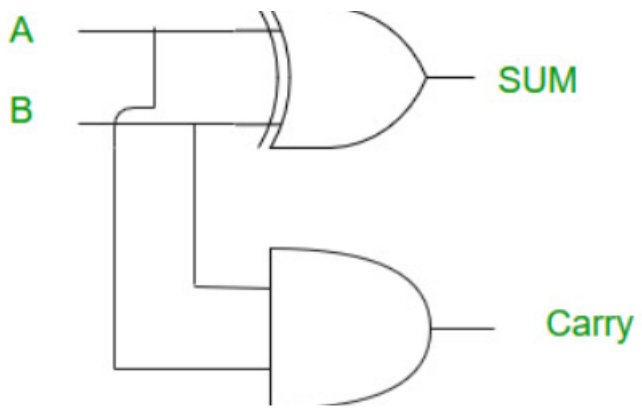
Decoder

A decoder is a combinational circuit having n inputs and to a maximum of m = 2n outputs. The decoder is the same as the de-multiplexer. The only difference between de-multiplexer and decoder is that in the decoder, there is no data input. The decoder performs an operation that is completely opposite of an encoder.

Encoder

The encoder is used to perform the reverse operation of the decoder. An encoder having n number of inputs and m number of outputs is used to produce m-bit binary code which is related to the digital input number. The encoder takes the digital word and converts it into another digital word

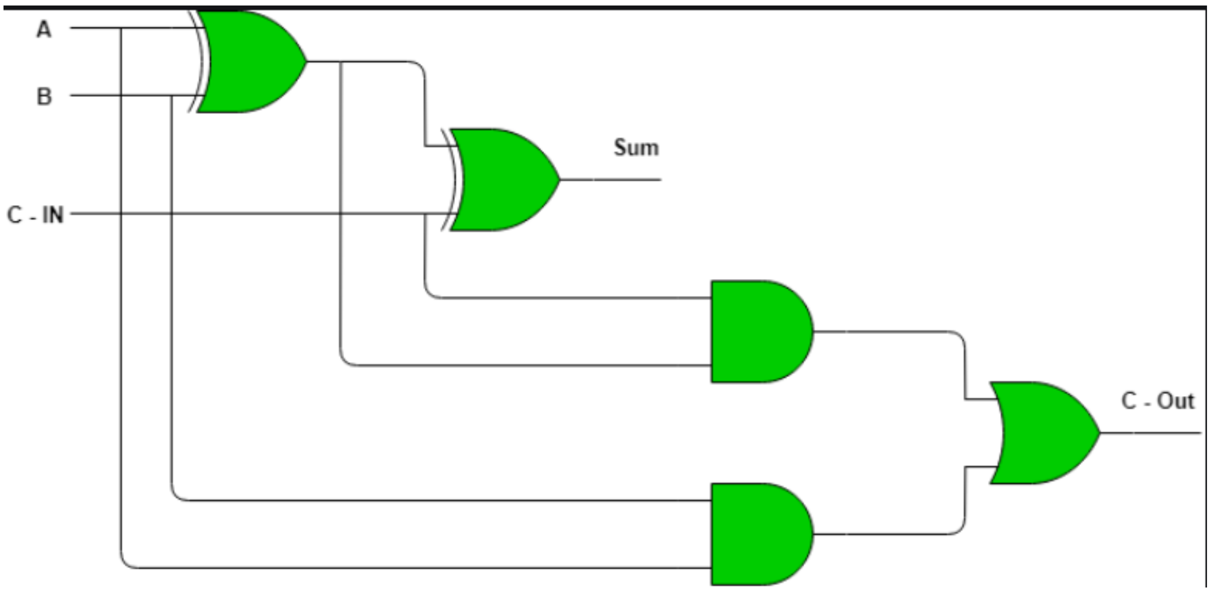
HALF ADDER



Code:

```
module TestModule;
//Inputs
reg a;
reg b;

//Outputs
wire sum;
wire carry;

HalfAdder uut (
.a(a),
.b(b),
.sum(sum),
.carry(carry)
);

initial begin
a = 0;
b = 0;
// Wait 100 ns for global reset to finish
#100

a = 1;
b = 0;
end
endmodule
```

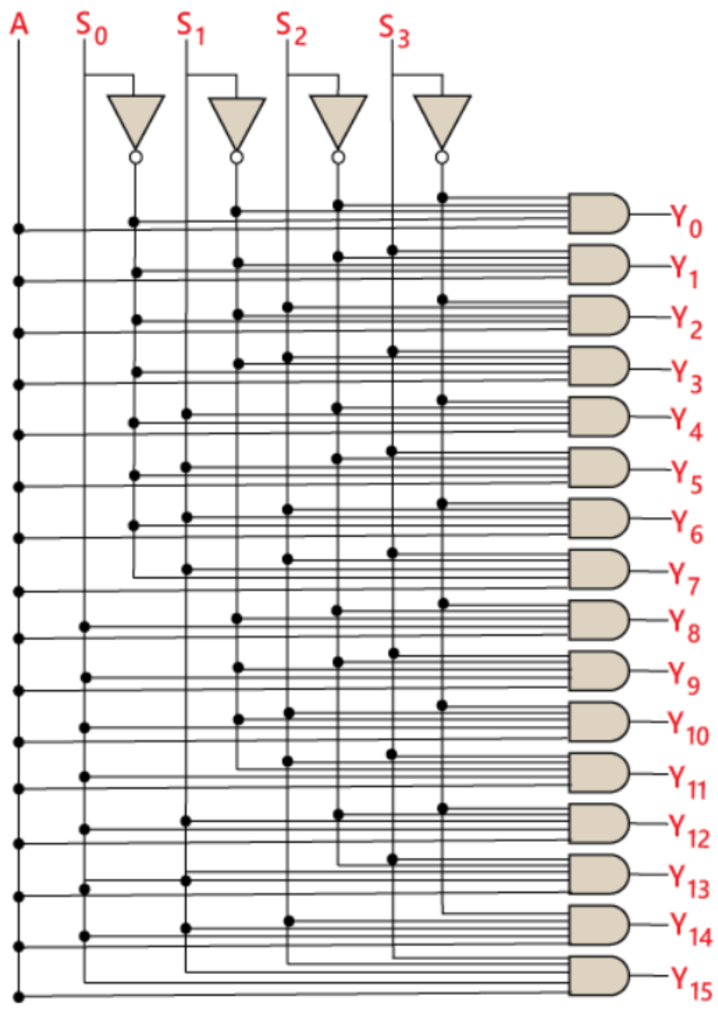
FULL ADDER:



Code:

```
module fulladder (  input [3:0] a,
                    input [3:0] b,
                        input c_in,
                       output reg c_out,
                       output reg [3:0] sum);


              always @ (a or b or c_in) begin
    {c_out, sum} = a + b + c_in;
  end
endmodule
```

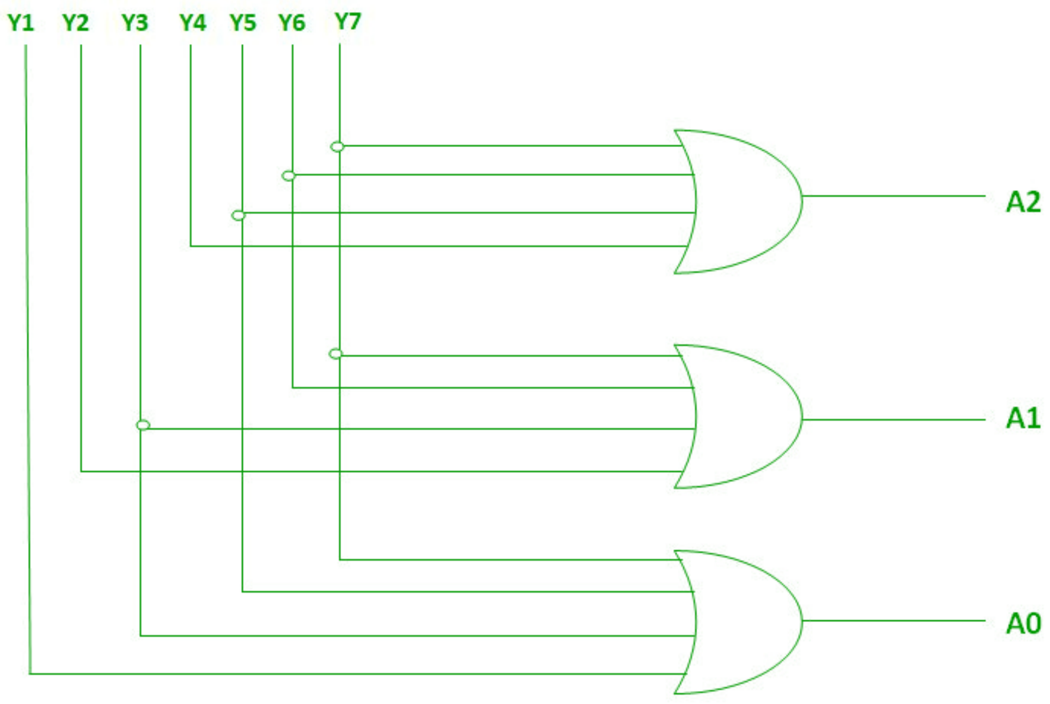
3*1 DE-MULTIPLEXER:



Code:

```
module Demultiplexer(in,s0,s1,s2,d0,d1,d2,d3,d4,d5,d6,d7);
input in,s0,s1,s2;
output d0,d1,d2,d3,d4,d5,d6,d7;
assign d0=(in & ~s2 & ~s1 &~s0),
d1=(in & ~s2 & ~s1 &s0),
d2=(in & ~s2 & s1 &~s0),
d3=(in & ~s2 & s1 &s0),
d4=(in & s2 & ~s1 &~s0),
d5=(in & s2 & ~s1 &s0),
d6=(in & s2 & s1 &~s0),
d7=(in & s2 & s1 &s0);
endmodule
```

8*3 ENCODER:



Code:

```
module Encoder(d0,d1,d2,d3,d4,d5,d6,d7,a,b,c);
input d0,d1,d2,d3,d4,d5,d6,d7;
```

```
output a,b,c;
or(a,d4,d5,d6,d7);
or(b,d2,d3,d6,d7);
or(c,d1,d3,d5,d7);
endmodule
```

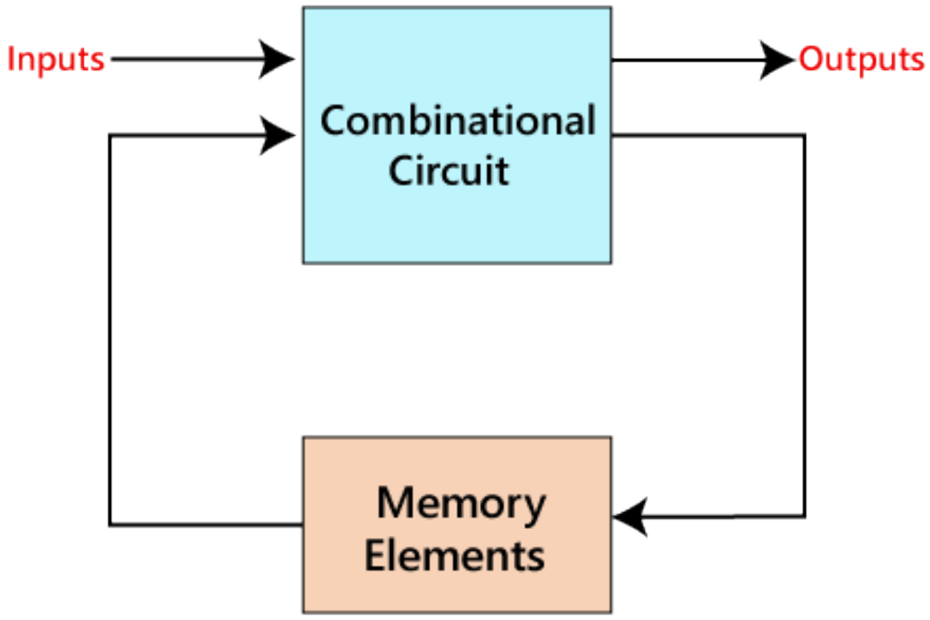
ACTIVITY LOG FOR THE FOURTH WEEK

DAY	BRIEF DESCRIPTION OF THE DAILY ACTIVITY	LEARNING OUTCOME	SIGNATURE OF THE FACULTY
Day-1	Introduction to Sequential circuits	Understand to Sequential circuits	
Day-2	Introduction to Sequential circuits	Study various Sequential circuits	
Day-3	Introduction to Sequential circuits	Study various Sequential circuits	
Day-4	Basic components Verilog coding for Sequential circuits	Learn basic elements in Verilog coding	
Day-5	Verilog coding for Sequential circuits	Implement Sequential circuits	
Day-6	Verilog coding for Sequential circuits	Implement Sequential circuits	

INTRODUCTION TO SEQUENTIAL CIRCUITS

Sequential circuit

In our previous sections, we learned about combinational circuit and their working. The combinational circuits have set of outputs, which depends only on the present combination of inputs. Below is the block diagram of the synchronous logic circuit.



The sequential circuit is a special type of circuit that has a series of inputs and outputs. The outputs of the sequential circuits depend on both the combination of present inputs and previous outputs. The previous output is treated as the present state. So, the sequential circuit contains the combinational circuit and its memory storage elements. A sequential circuit doesn't need to always contain a combinational circuit. So, the sequential circuit can contain only the memory element.

TYPES OF SEQUENTIAL CIRCUITS

Asynchronous sequential circuits

The clock signals are not used by the **Asynchronous sequential circuits**. The asynchronous circuit is operated through the pulses. So, the changes in the input can change the state of the circuit. The asynchronous circuits do not use clock pulses. The internal state is changed when the input variable is changed. The un-clocked flip-flops or time-delayed are the memory elements of asynchronous sequential circuits. The asynchronous sequential circuit is similar to the combinational circuits with feedback.

Synchronous sequential circuits

In synchronous sequential circuits, synchronization of the memory element's state is done by the clock signal. The output is stored in either flip-flops or latches(memory devices). The synchronization of the outputs is done with either only negative edges of the clock signal or only positive edges.

DESIGN OF FLIP-FLOPS WITH GATE PRIMITIVES

The basic RS latch can be designed using gate primitives. Two instantiations of NAND or NOR gates suffice here. More involved flip-flops, registers, etc., can be built around these. Some of the level triggered versions of such flip-flops are taken up for design. Subsequently, the edge-triggered flip-flop of the 7474 type is developed in a skeletal form

RS Flip-Flop

module srff(s,r,q,qb); input s,r; output q,qb; wire ss,rr; not(ss,s),(rr,r); nand(q,ss,qb); nand(qb,rr,q); endmodule

module tstsrff; //test-bench reg s,r; wire q,qb; srff ff(s,r,q,qb); initial

begin s =1'b1; r =1'b0;

end always begin

#2 s =1'b0;r =1'b0;

#2 s =1'b0;r =1'b1;

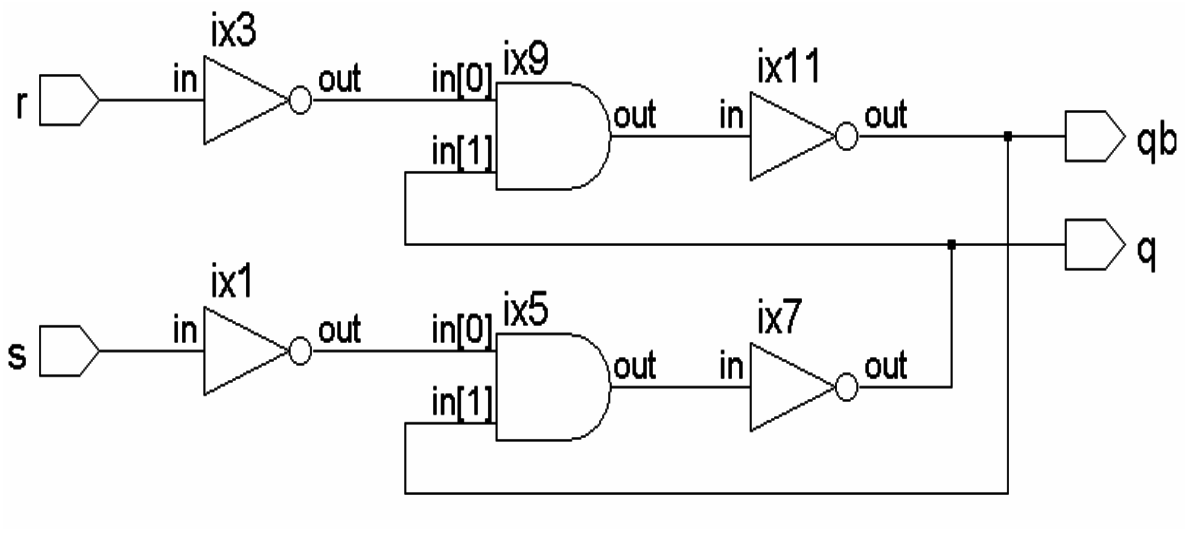
#2 s =1'b0;r =1'b0;

#2 s =1'b1;r =1'b0;

#2 s =1'b0;r =1'b0; end initial $monitor($time, " s = %b, r = %b, q = %b, qb =

%b ",s,r,q,qb); initial #20 $stop; endmodule

```
#  0 s = 1 , r = 0 , q = 1 , qb  = 0
#  2 s = 0 , r = 0 , q = 1 , qb  = 0
#  4 s = 0 , r = 1 , q = 0 , qb  = 1
#  6 s = 0 , r = 0 , q = 0 , qb  = 1
#  8 s = 1 , r = 0 , q = 1 , qb  = 0
# 10 s = 0 , r = 0 , q = 1 , qb  = 0
# 14 s = 0 , r = 1 , q = 0 , qb  = 1 # 16 s = 0 , r = 0 , q = 0 , qb  = 1
# 18 s = 1 , r = 0 , q = 1 , qb  = 0
```



D-LATCH

module dlatch(en,d,q,qb); input d,en; output q,qb; wire dd; wire s,r; not n1(dd,d); nand (sb,d,en); nand g2(rb,dd,en);

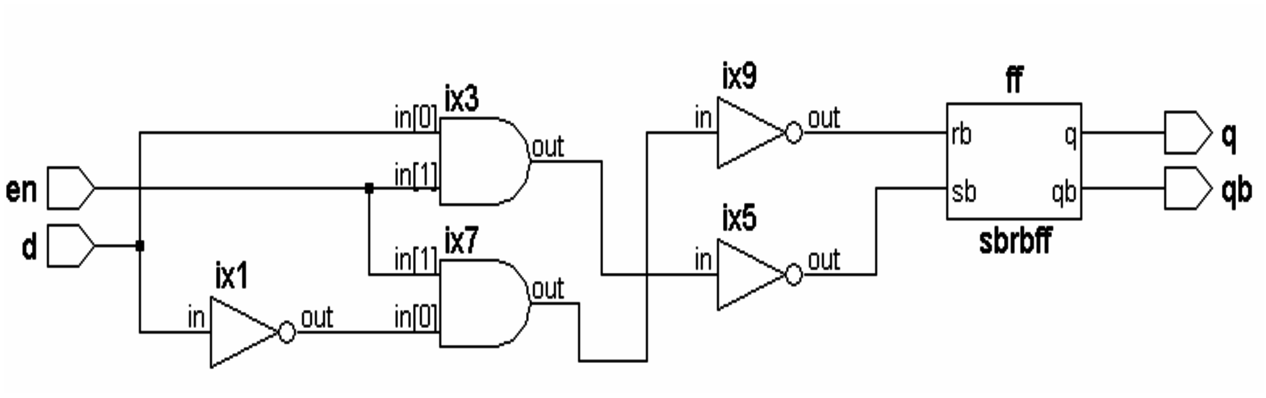
sbrbff ff(sb,rb,q,qb);//Instantiation of the sbrbff endmodule

module tstdlatch; //test-bench reg d,en; wire q,qb; dlatch ff(en,d,q,qb); initial begin d = 1'b0; en = 1'b0;

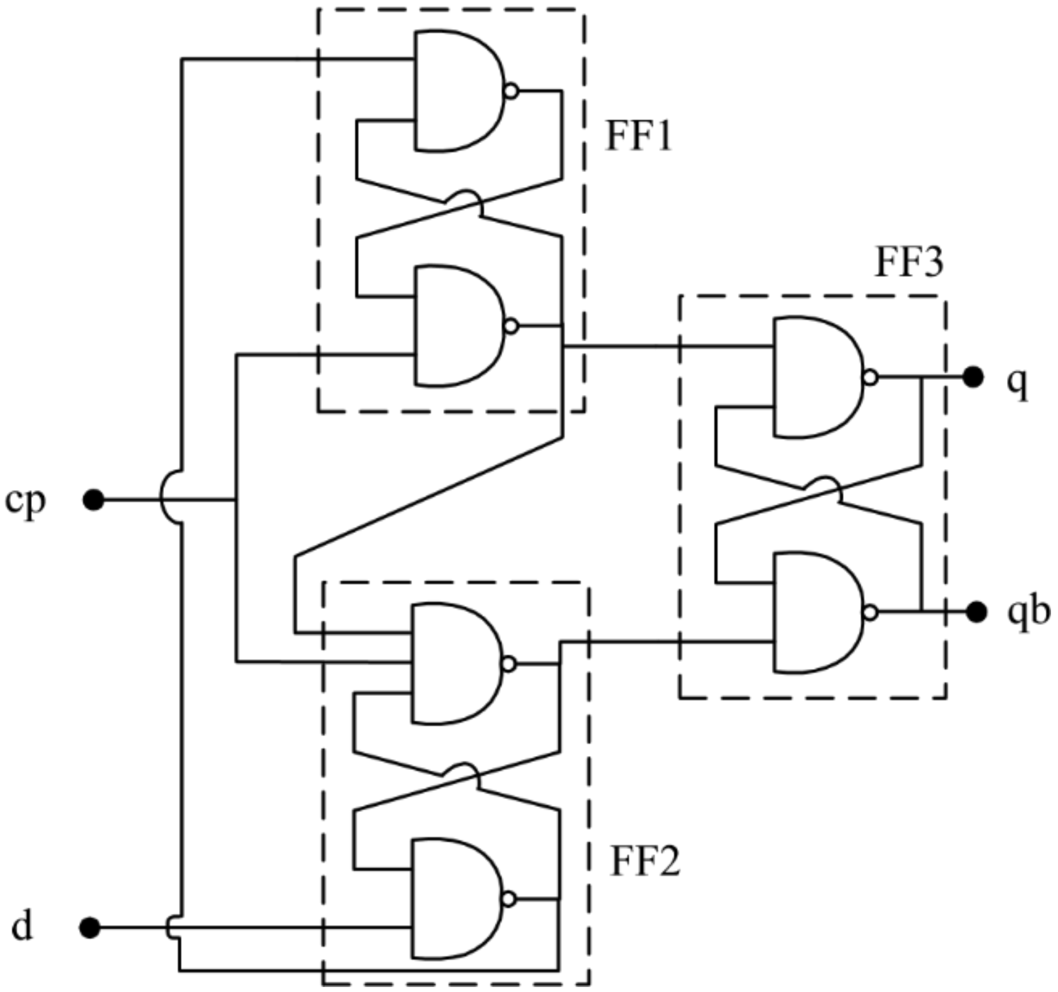
end always #4 en =~en; always #8 d=~d; initial $monitor($time," en = %b , d = %b , q = %b , qb

= %b " , en,d,q,qb); initial #40 $stop; endmodule

```
#  0 en = 0, d = 0, q = x, qb  = x
#  4 en = 1, d = 0, q = 0, qb  = 1
#  8 en = 0, d = 1, q = 0, qb  = 1
# # 12 en = 1, d = 1, q = 1, qb  = 0
16 en = 0, d = 0, q = 1, qb  = 0
# 20 en = 1, d = 0, q = 0, qb  = 1
# 24 en = 0, d = 1, q = 0, qb  = 1
# 28 en = 1, d = 1, q = 1, qb  = 0
# 32 en = 0, d = 0, q = 1, qb  = 0
# 36 en = 1, d = 0, q = 0, qb  = 1
```



EDGE-TRIGGERED FLIP-FLOP



Circuit of a skeletal edge-triggered flip-flop.

ASYNCHRONOUS COUNTER

```
module asynchronouscountermod(clk, clear, q);

input clk;

input clear;

output [3:0] q;
```

```
reg [3:0] q;

always @(negedge clk or posedge clear)
q[0]<=~q[0];
always @(negedge q[0] or posedge clear)
q[1]<=~q[1];
always @(negedge q[1] or posedge clear)
q[2]<=~q[2];
always @(negedge q[2] or posedge clear)
begin
if(clear)
q <=4'b0000;
else
q[3]<=~q[3];
end
endmodule
```

JOHNSON COUNTER

```
module johnson_ctr #(parameter WIDTH=4)
  (
    input clk,
    input rstn,
    output reg [WIDTH-1:0] out );


  always @ (posedge clk) begin
      if (!rstn)
         out <= 1;
      else begin
        out[WIDTH-1] <= ~out[0];
        for (int i = 0; i < WIDTH-1; i=i+1) begin
```

```
          out[i] <= out[i+1];
        end
      end
  end
endmodule
```

ACTIVITY LOG FOR THE FIFTH WEEK

DAY	BRIEF DESCRIPTION OF THE DAILY ACTIVITY	LEARNING OUTCOME	SIGNATURE OF THE FACULTY
Day-1	VLSI Signal processing: Basics	Understand the basics of VLSI signal processing	
Day-2	VLSI Signal processing architecture	Understand the architecture of VLSI signal processing	
Day-3	VLSI Signal processing applications	Learn the applications of VLSI signal processing	
Day-4	Pipelining concepts on basic FIR digital	Understand the pipelining concepts	
Day-5	Pipelining concepts on basic FIR digital	Learn Applications and Approaches in Pipelining	
Day-6	Pipelining concepts on basic FIR digital	Learn Pipelining in FIR filters	

VLSI SIGNAL PROCESSING

Introduction

Throughout the history of computing, digital signal processing applications have augmented the limits of compute power, especially in terms of real-time computation. While processed signals have broadly ranged from media-driven video, audio, and speech waveforms to specialized sonar and radar data, most of the calculations performed by signal processing systems have essentially exhibited similar basic computational characteristics. The inherent data parallelism found in many DSP functions has made DSP algorithms suitable for hardware implementation, leveraging expanding VLSI capabilities. Recently, DSP has witness a rapid surge in productivity due to rapid advancements in multimedia computing and high-speed wired and wireless communications.

Evolution of Hardware for DSP

Three goals have constantly driven the development of DSP implementations: 1. Data parallelism 2. Application-specific specialization 3. Functional flexibility.

In general, design decisions regarding DSP system implementation require a balanced tradeoffs between these three system goals. As a result, an extensive variety of specialized hardware implementations and associated design tools have been developed for DSP including associative processing, bit-serial processing, on-line arithmetic, and systolic processing. As implementation technologies have become available, these basic approaches have advanced to meet the needs of application designers. In the above table, various cost metrics have been developed to compare the quality of different DSP implementations. Performance has almost always been the most critical system requirement since DSP systems often have demanding real-time constraints. In the past three decades, however, cost has become more significant as DSP has transformed from predominantly military and scientific applications into numerous low-cost consumer applications. In the past decade, energy consumption has become an important measure as DSP techniques have been widely applied in portable, battery-operated systems such as cell-phones, CD players, and laptops.

Finally, flexibility has proved its significance as one of the key differentiators in DSP implementations since it allows changes to system functionality at various points in the design life cycle.

Choosing an optimal architecture

Optimal VLSI architecture is technology dependent, which requires characterization of primary functional blocks for speed, power, and area. This information is used to steer the architectural optimization procedure that is based on balancing the algorithm throughput requirement with the capability of the underlying basic building blocks. Data throughput and latency are primary constraints in chip realizations. Data throughput is interesting for optimization since, for a given architecture, the throughput relates to the frequency of operation.

The key information that provides grounds for optimization is technology specific energy-delay tradeoff in datapath logic as shown in the below figure. This tradeoff exists because the energy needed to operate digital logic gates is influences their speed. The tradeoff is obtained by negotiating the design parameters such as gate size, supply and threshold voltage. Introduction of a new technology shifts the entire Energy-Delay curve towards lower energy and delay. The architecture is said to be energy optimal when the slope of its E-D tradeoff curve is similar to that of the underlying datapath logic. A good tradeoff point is indicated in figure below.

By using the concepts of parallelism and time multiplexing, an algorithm can be mapped into a range of architectures with widely varying throughput and latency. Architectural transformations such as data-stream interleaving, loop retiming and folding enable more complex operations with concurrent or time-serial execution, which may involve feedback loops.

1. Parallelism & Time Multiplexing: Parallelism along with adjustment in the supply voltage improves the energy by slowing down the clock and distributing computation power over several parallel branches computing together.

2. Data-Stream Interleaving: Data-stream is a way of time multiplexing the data. Interleaving essentially improves the area efficiency by sharing data-path logic across the independent streams of data.

3. Folding: Folding too reduces the area. However, it raises the issue of how to optimally distribute pipeline registers around the loop in order to maximize throughput.
4. Loop Retiming: Loop retiming is a technique of distributing pipeline registers around recursive loops by assigning the right amount of latency to basic functional building blocks and then distributing the pipeline registers inside the blocks such that all internal datapath logic blocks lay at the same point.
5. Delayed Iteration: Delayed iteration occurs when majority of loops have somewhat similar latency, while only a few loops need longer computation.

Reconfigurability

Most reconfigurable devices and systems contain SRAM-programmable memory to allow full logic and interconnect reconfiguration in the field. Despite a wide range of system characteristics, most DSP systems need reconfiguration under a variety of constraints. [7] These constraints may include ever changing environmental factors such as changes in statistics of signals and noise, channel, weather, transmission rates, and communication standards. While factors such as data traffic and interference often change at a high speed, other factors such as location and weather change relatively slowly. Still infrequent variation of some other factors regarding communication standards across time and geography limit the need for rapid reconfiguration.

• Field customization—The reconfigurability of programmable devices allows periodic updates of product functionality on introduction of advanced vendor firmware versions or detection of product defects. Field customization is particularly essential in the face of changing standards and communication protocols. Unlike ASIC implementations, reconfigurable hardware solutions can be rapidly updated based on the application demands without requiring manual field upgrades or hardware swaps.

• Slow adaptation—Signal processing systems based on reconfigurable logic may need to be periodically updated in the course of daily operation based on a number of constraints. These may include issues such as the ever changing weather and operating parameters for mobile communication and structural support for multiple, time-varying standards in stationary receivers.

• Fast adaptation—Many communication processing protocols may benefit from rapid reset of computing parameters and require constant re-evaluation of operating parameters.

Some of these issues include adaptation to time-varying noise in communication channels, adaptation to network congestion in network configurations, and speculative computation based on changing data sets.

Parallelism

The abundance of programmable logic smoothly facilitates the creation of a number of functional units directly in hardware. Certain characteristics of FPGA devices, in particular, make them especially the preffered choice for use in digital signal processing systems. The fine-grained parallelism found in these devices is at par with the high-sample rates and distributed computation often required of signal processing applications in image, audio, and speech processing. Plentiful FPGA flip flops and the motivation to achieve accelerated system clock rates have led designers to focus on heavily pipelined implementations of functional blocks and inter-block communication. Given the highly pipelined and parallel nature of many DSP tasks, such as image and speech processing, these implementations have exhibited remarkably better performance than standard PDSPs. In conclusion, these systems have been successfully implemented using both task and functional unit pipelining.

Referred to as Ring Connected Trees or RCT, a two-dimensional lattice of (NxN) PEs is used for an (NxN) RCT. In the proposed architecture, a parallel processing environment for signal processing is provided by the multiple processing elements working in a coordinated way to do the computation. Multiprocessing along with pipelining results in lower computation time and faster results. The machine was tested for the parallel working of a number of computation problems from the signal processing domain at the same time. RCT has better VLSI area complexity as compared to that of Mesh-of-Tree and has linear time performance for signal processing computations. This research largely misses out on signal processing applications like autocorrelation, IIR filtering, Hadamard transform, Walsh transform and also towards multidimensional cases. However, a tree-based architecture has some obvious advantages like simplicity and regularity, area-efficient VLSI design, embedded hierarchic organization, ease of mapping signal processing algorithms to Tree-based structures, etc.

Some Major Applications

A) Sigma Delta ADC Nearly all the signals perceived naturally are continuous and hence analog in nature. These analog signals need to be converted to discrete time-discrete valued digital signals so as to be processed by digital computational systems. Analog-to-Digital (ADC) and Digital-to-Analog (DAC) converters are therefore quintessential for bit-conversion and processing of continuous data in any discrete digital system. Any analog signal is firstly sampled, then quantized to obtain its discrete form. One of the more advanced ADC technologies is the so-called delta-sigma, or ΔΣ (using the proper Greek letter notation). In mathematics and physics, the capital Greek letter delta (Δ) represents difference or change, while the capital letter sigma (Σ) represents summation. In a sigma-delta converter, the analog input voltage signal is connected to the input of an integrator, producing a voltage rate-of-change, or slope, at the output corresponding to input magnitude. This ramping voltage is then compared against ground potential (0 volts) by a comparator. The comparator acts as a sort of 1- bit ADC, producing 1 bit of output ("high" or "low") depending on whether the integrator output is positive or negative. The comparator's output is then latched through a D-type flip-flop clocked at a high frequency, and fed back to another input channel on the integrator, to drive the integrator in the direction of a 0 volt output.

B) Wireless Communication There has been a lot of advancement in wireless communication due to Very-Large Scale Integration System design. The change in VLSI circuits which are implemented for wireless communications is remarkable as the analysis of new VLSI systems continues. The transfer of radio waves through base stations is achieved by virtue of Radio Frequency power amplifiers, and the power amplifiers most commonly in application are the MOSFET's (Metal Oxide Semiconductor Field Effect Transistor), which are the base of VLSI design systems. Some of the distinguished technologies for Radio Frequency (RF)

Si-Ge BiCMOS technology is being evaluated today for use in higher frequency applications such as emerging WLAN, automotive radar and collision avoidance products in the 24 to 77 GHz range as well as wire-line communications at 40 Gigabyte per second and beyond. Another technology that is profound for RF applications is the RF CMOS technology, it is capable of integrating various functions on a single chip, hence deducting the cost of the chip. Certain wide-ranging applications for wireless communications such as the sensitivity, gain and noise of the device are balanced with the help of RF CMOS.

The choice of RF technologies for wide-ranging communication applications can be broadly separated by the optimization for cost and/or performance. Since the basis of RFCMOS is digital CMOS, the attributes of RF technology are very dependent on the digital CMOS roadmap. The future trends of VLSI systems for wireless communications will be analyzed and the state of the art technologies might be surpassed by the upcoming advancements of VLSI Circuits.

C) CORDIC Irrespective of the complexity, all Deterministic and even a few Random signals can be represented by a family of mathematical equations using trigonometric and conic equations. The CORDIC processor helps us realize these equations. The coordinate Rotation Digital Computer (CORDIC) algorithm is a hardware efficient iterative algorithm which allows a simple shift and add operation to calculate hyperbolic, exponential, and logarithmic and trigonometric functions like sine, cosine, magnitude and phase with great precision for Digital Signal Processing (DSP) applications especially during modulation and demodulation phases.

As number of gates and ROM memory required starts increasing at an alarming rate for any application, implementation of the CORDIC becomes increasingly difficult due to higher quantization errors. One of the suggested methods to deal with this issue is development of a proper Application Specific ‘pipeline’ in Verilog HDL. Such optimization not only saves area on silicon substrate but also helps in reducing the computed quantization error. The CORDIC architecture is efficiently coded using Verilog HDL. The architecture is pipelined to have an internal critical path of a single adder. To minimize angle approximations error, numbers of micro-rotations have been adjusted. To reduce the total quantization error including scale factor error, the pipelined CORDIC architecture has been optimized.

D) Speech Processing The low cost VLSI architectures come under use to deal with architecture to solve complex problems. In mobile phones speech processing plays a crucial role in solving complex DSP procedures as these involve, speech recognition, noise suppression, silence detection, pitch analysis and many more. FPGA is recommended for low price VLSI which is also widely used in the market. Here there is a specific architecture for suppressing surrounding noise in the mobile communication.

- In case of noisy speech pertaining to nonparametric model based methods, noise is estimated and removed from degraded using subtractive algorithms.

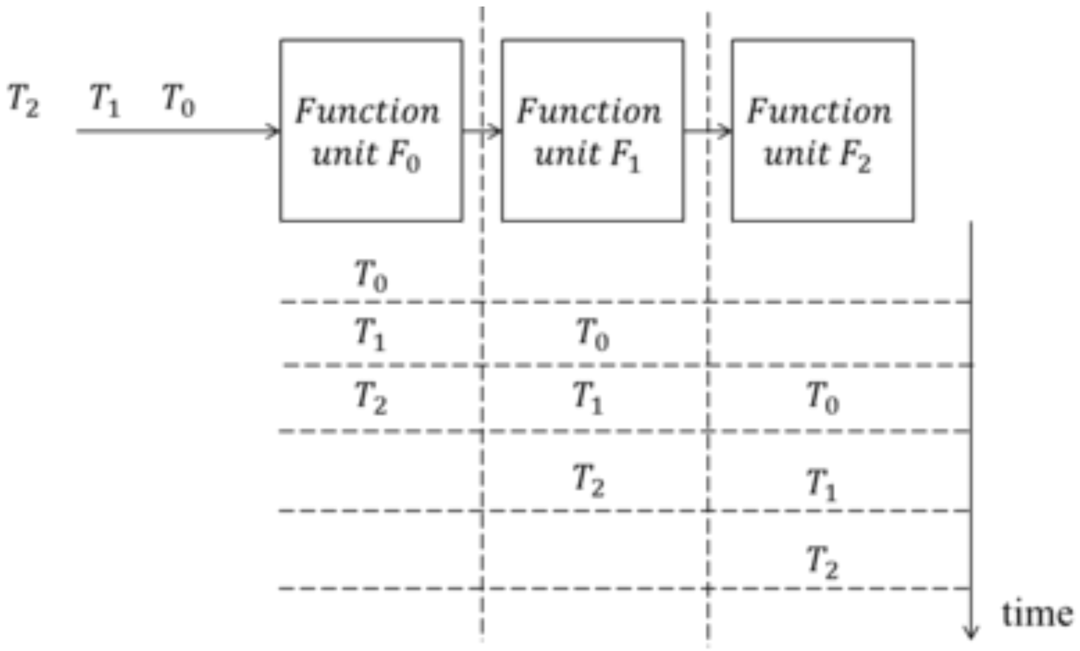
- FIR filters crucially influence the operation and performance in noise estimation and thereby on the complete system. To process real time signals it is necessary to design filters with low power operation for a given throughput requirement. Efficient Filters can be designed using a MAC circuit that consumes less processing time and less hardware. Implementation of FIR in FPGA can be a simple MAC, Parallel or Semi-Parallel, and Multi-channel FIR.

PIPELINING CONCEPTS ON BASIC FIR DIGITAL

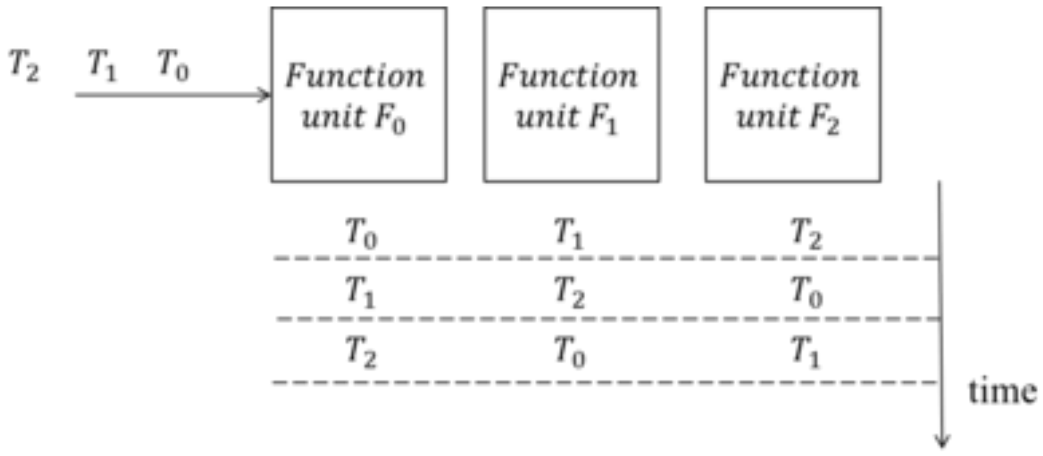
Pipelining is an important technique used in several applications such as digital signal processing (DSP) systems, microprocessors, etc. It originates from the idea of a water pipe with continuous water sent in without waiting for the water in the pipe to come out. Accordingly, it results in speed enhancement for the critical path in most DSP systems. For example, it can either increase the clock speed or reduce the power consumption at the same speed in a DSP system.

Pipelining allows different functional units of a system to run concurrently. Consider an informal example in the following figure. A system includes three sub-function units (F_0, F_1 and F_2). Assume that there are three independent tasks (T_0, T_1 and T_2) being performed by these three function units. The time for each function unit to complete a task is the same and will occupy a slot in the schedule.

If we put these three units and tasks in a sequential order, the required time to complete them is five slots.



However, if we pipeline T_0 to T_2 concurrently, the aggregate time is reduced to three slots.



Therefore, it is possible for an adequate pipelined design to achieve significant enhancement on speed.

COSTS AND DISADVANTAGES

Pipelining cannot decrease the processing time required for a single task. The advantage of pipelining is that it increases the throughput of the system when processing a stream of tasks.

Applying too many pipelined functions can lead to increased latency - that is, the time required for a single task to propagate through the full pipe is prolonged. A pipelined system may also require more resources (buffers, circuits, processing units, memory etc.), if the reuse of resources across different stages is restricted.

COMPARISON WITH PARALLEL APPROACHES

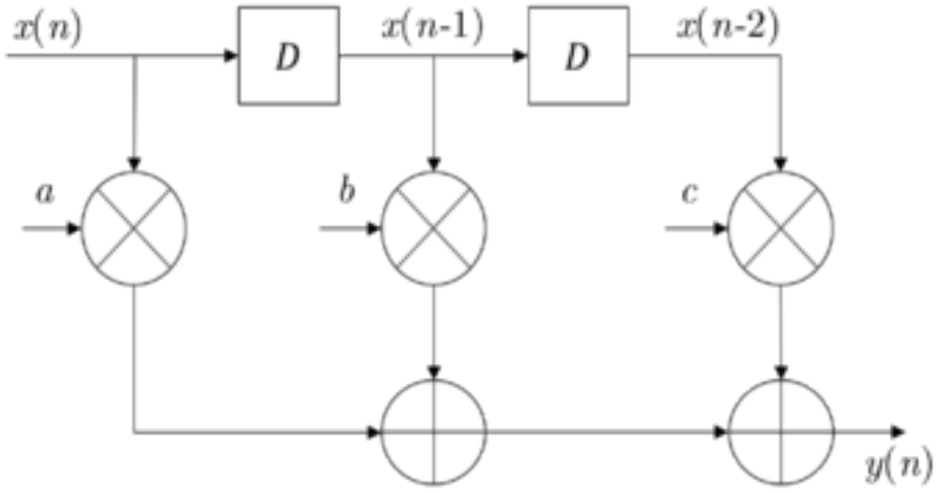
Another technique to enhance the efficiency through concurrency is parallel processing. The core difference is that parallel techniques usually duplicate function units and distribute multiple input tasks at once amongst them. Therefore, it can complete more tasks per unit time but may suffer more expensive resource costs.

For the previous example, the parallel technique duplicates each function units into another two. Accordingly, all the tasks can be operated upon by the duplicated function units with the same function simultaneously. The time to complete these three tasks is reduced to three slots.

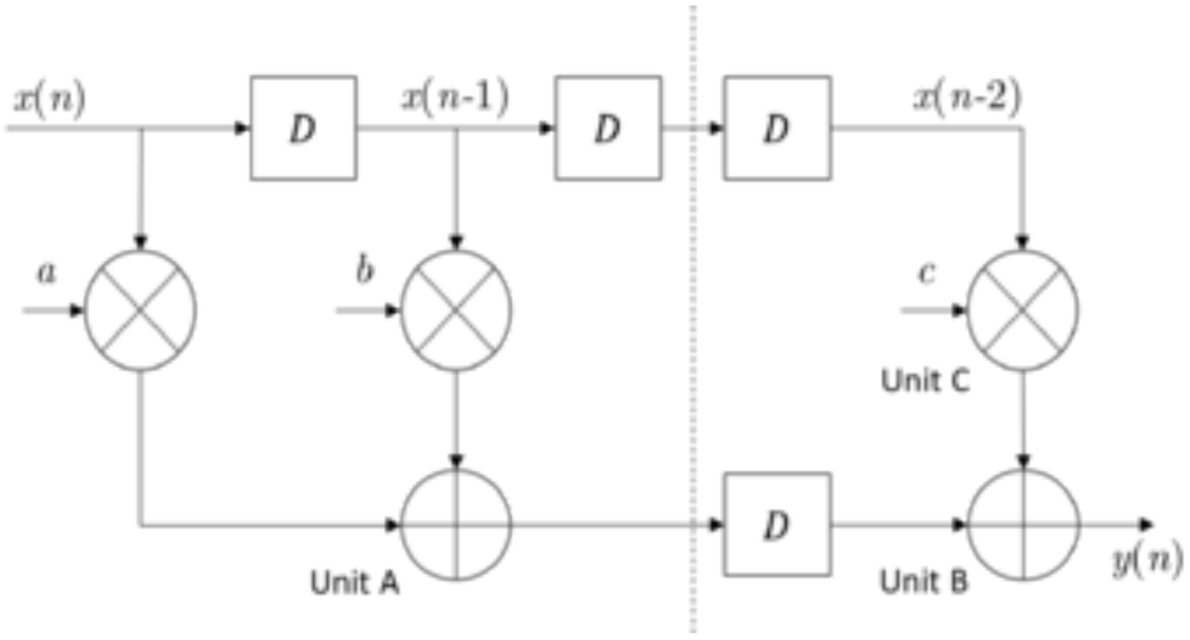
PIPELINING IN FIR FILTERS

Consider a 3-tap FIR filter:

Assume the calculation time for multiplication units is T_m and T_a for add units. The critical path, representing the minimum time required foprocessing a new sample, is limited by 1 multiplication and 2 add function units. Therefore, the sample period is given by



However, such structure may not be suitable for the design with the requirement of high speed. To reduce the sampling period, we can introduce extra pipelining registers along the critical data path. Then the structure is partitioned into two stages and the data produced in the first stage will be stored in the introduced registers, delaying one clock to the second stage. The data in first three clocks is recorded in the following table. Under such pipelined structure, the sample period is reduced to



Clock	Input	Unit A	Unit B	Unit C	Output
0	$x(0)$	$ax(0) + bx(-1)$	–	–	–
1	$x(1)$	$ax(1) + bx(0)$	$ax(0) + bx(-1)$	$cx(-2)$	$y(0)$
2	$x(2)$	$ax(2) + bx(1)$	$ax(1) + bx(0)$	$cx(-1)$	$y(1)$
3	$x(3)$	$ax(3) + bx(2)$	$ax(2) + bx(1$	$cx(0)$	$y(2)$

Pipelining in 1st-order IIR filters

By combining look-ahead techniques and pipelining,[2] we are able to enhance the sample rate of target design. Look-ahead pipelining will add canceling poles and zeroes to the transfer function such that the coefficients of the following terms in the denominator of the transfer function are zero.

Then, the output sample y(n) can be computed in terms of the inputs and the output sample y(n − M) such that there are M delay elements in the critical loop. These elements are then used to pipeline the critical loop by M stages so that the sample rate can be increased by a factor M.

$$\{z^{-1}, \ldots, z^{-(M-1)}\}$$

Consider the 1st-order IIR filter transfer function

$$H(z) = \frac{1}{1 - az^{-1}}$$

The output y(n) can be computed in terms of the input u(n) and the previous output.

$$y(n) = ay(n-1) + u(n)$$

In a straightforward structure to design such function, the sample rate of this recursive filter is restricted by the calculation time of one multiply-add operation.

$$z = a, a \leq 1$$

To pipeline such design, we observe that H has a pole at

Therefore, in a 3-stage pipelined equivalent stable filter, the transfer function can be derived by adding poles and zeros at

$$z = ae^{\pm(\frac{2j\pi}{3})}$$

and is given by

$$H(z) = \frac{1 + az^{-1} + a^2 z^{-2}}{1 - a^3 z^{-3}}$$

Therefore, the corresponding sample rate can be increased by a factor 3.