

Lecture-2 : ASIC, FPGA, Logic Synthesis Fundamentals

ECE-111 Advanced Digital Design Project

Vishal Karna

Winter 2022

Lecture-2 Objective and Outline

- ❑ Introduction to ASIC and FPGA design flow
- ❑ Introduction to FPGA Architecture and understand how digital logic is implemented inside FPGA
- ❑ Overview of Logic Synthesis, Netlist, Simulation, Simulator, Waveform concepts
- ❑ Advantages and Dis-advantages of ASIC versus FPGA based hardware designs

What is an ASIC ?

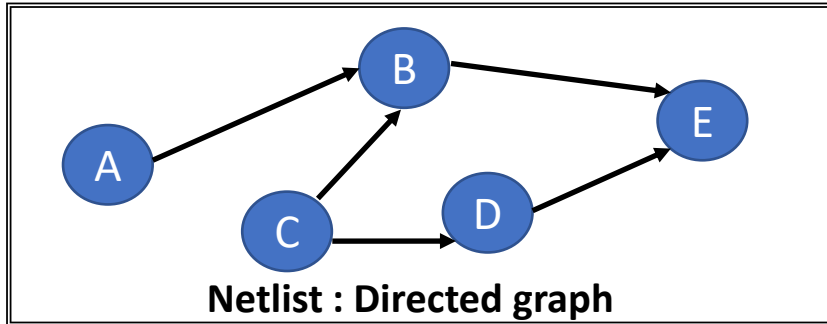
ASIC stands for Application Specific Integrated Circuit.

- As name suggests, ASIC's are **application specific circuits**
- Designed for one sole purpose and they function the same their whole operating life.
- Example : “**Mobile processor** inside a mobile phone is an ASIC. It is meant to function as a **mobile processor** for its entire life”.
- Its logic function cannot be changed since its digital circuitry is made up of permanently connected gates and flip-flops in silicon.
- Logic function is specified using **HDL** such as **SystemVerilog, Verilog, VHDL**



What is a netlist ?

- ❑ A netlist is a directed graph, where the vertices indicates components, and the edges indicate interconnections.



edge



Interconnections

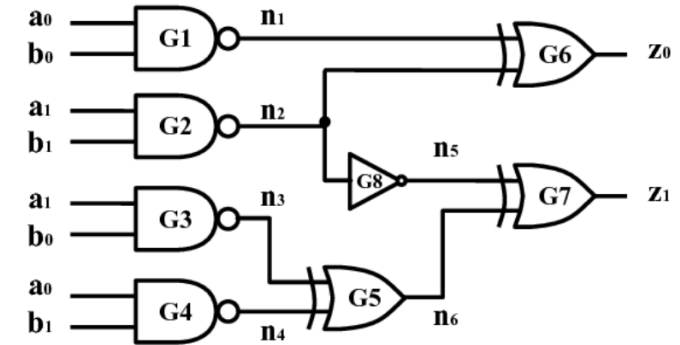


Vertice

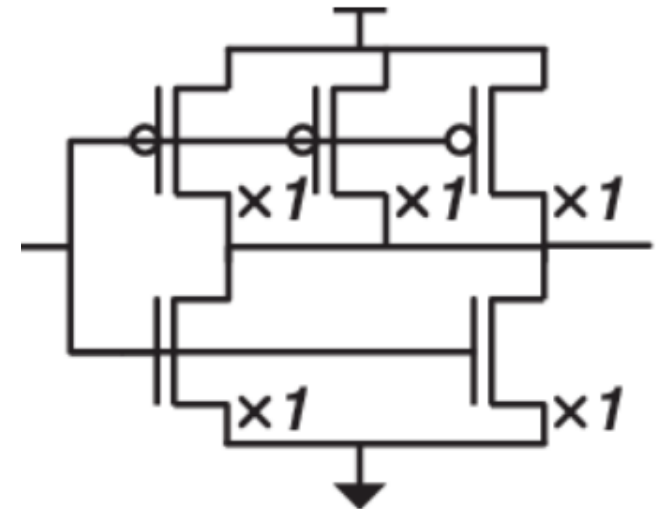
SystemVerilog modules or logic gates or transistors

- ❑ Netlist can be specified at various levels :
 - If vertices are gates, then netlist is called a **gate level netlist**
 - If vertices are transistors, then netlist is called a **transistor level netlist**

Logic Gate Level Netlist



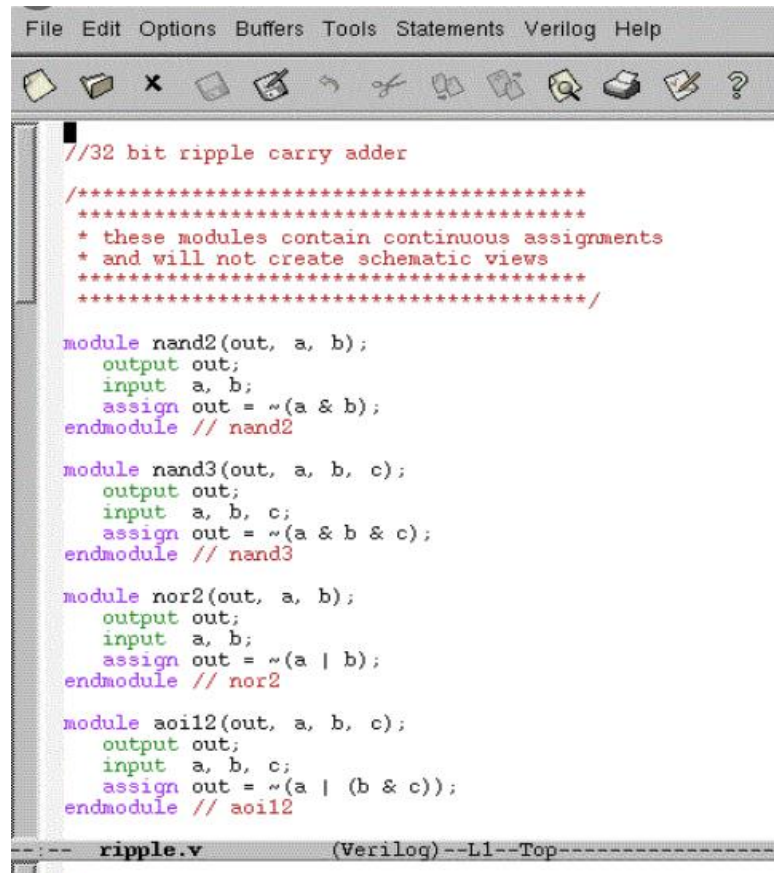
Transistor Level Netlist



What is a Netlist ?

- If vertices are modules, then netlist is called a **SystemVerilog level netlist**

SystemVerilog Level Netlist



```
File Edit Options Buffers Tools Statements Verilog Help
[Icons]

//32 bit ripple carry adder

/*****
 * these modules contain continuous assignments
 * and will not create schematic views
 *****/

module nand2(out, a, b);
    output out;
    input a, b;
    assign out = ~(a & b);
endmodule // nand2

module nand3(out, a, b, c);
    output out;
    input a, b, c;
    assign out = ~(a & b & c);
endmodule // nand3

module nor2(out, a, b);
    output out;
    input a, b;
    assign out = ~(a | b);
endmodule // nor2

module aoi12(out, a, b, c);
    output out;
    input a, b, c;
    assign out = ~(a | (b & c));
endmodule // aoi12

-- ripple.v (Verilog)--L1--Top--
```

What is Synthesis ?

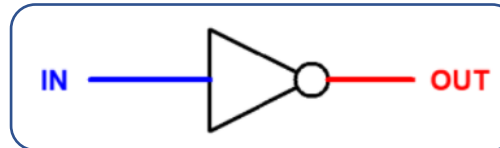
- ❑ Synthesis is a process to convert design from one form to another form
- ❑ Design Netlist can be transformed from one level to another using a process known as “Synthesis”
 - **Logic Synthesis** : Converts SystemVerilog netlist to gate-level netlist representation of design
 - **Physical Synthesis** : Converts gate-level netlist to transistor level netlist

SystemVerilog Inverter Design Netlist

```
1 module NOT_Gate(  
2   input IN,  
3   output OUT);  
4  
5   assign OUT = ~IN;  
6  
7 endmodule
```

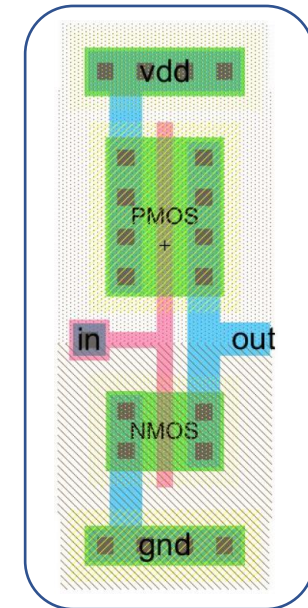
Logic Synthesis

Gate-Level Inverter Design Netlist

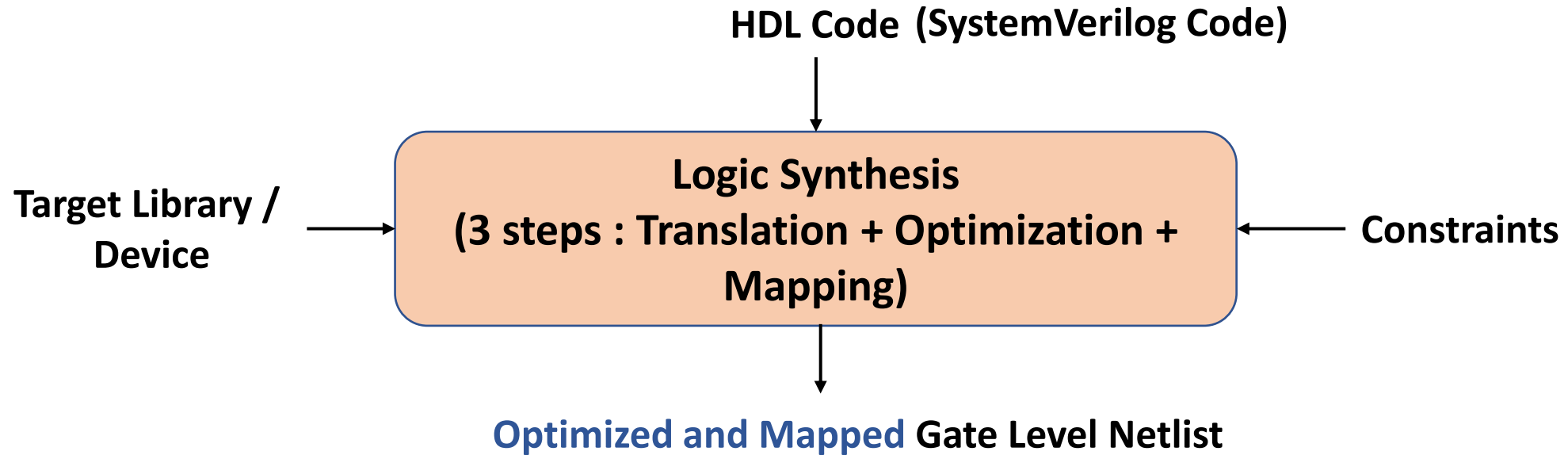


Physical Synthesis

Transistor-Level Inverter Design Netlist



What is Logic Synthesis ?



- ❑ **Logic synthesis is the process of converting HDL description of a design into an optimized structured gate-level representation using technology specific primitives :**
 - Output of logic synthesis process is optimized and mapped gate-level netlist
 - For FPGAs Synthesis uses : **Look-Up-Tables (LUTs)**, flip-flops, and RAM blocks to implement design
 - For ASICs it uses : Standard cell gates, flip-flop libraries, and memory blocks to implement design
 - Constraints are provided to synthesis tool to meet goals such as :
 - **Timing, area, performance, max dynamic transition for power, fanout**

Logic Synthesis Goals

❑ Logic Synthesis Goals :

- **Minimize area:**

- in terms of literal count, cell count, register count, etc

- **Minimize power:**

- in terms of switching activity in individual gates, deactivated circuit blocks, etc

- **Maximize performance:**

- in terms of maximal clock frequency of synchronous systems, throughput for asynchronous systems)

- **Combination of above:**

- “minimize area for clock speed >500Mhz”

- **Feedback from layout :**

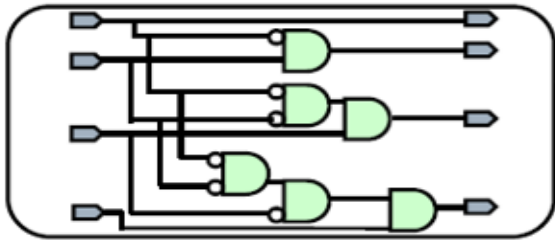
- actual physical sizes, delays, placement and routing

ASIC Logic Synthesis Flow Details

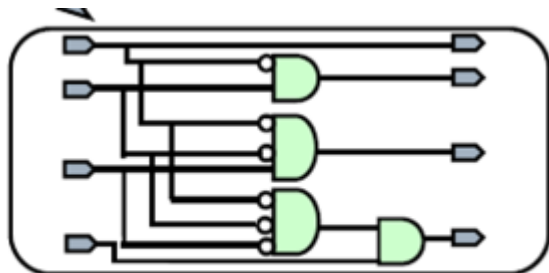
SystemVerilog .sv Source Code files

```
1 // Full Adder SystemVerilog Module
2 module fulladder(input logic a, b, cin,
3                 output logic s, cout);
4     logic p, g; // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11 endmodule
12
13
```

No Timing Details



Generic Boolean
(GTECH)



Target Technology

Timing Details

Step 1 : Translation

- Check if design provided can be synthesized
- Map RTL to unmapped gate-netlist
- Uses standard GTECH library which has “and, or, nand” type of gates and designware library which has adders, comparators type primitives

Step 2 : Optimization

- Reduce logic
- Eliminate redundant logic
- Make design smaller and faster (best Fmax)

Step 3 : Technology Mapping

- Map gates to technology dependent standard cell and macros
- Technology library is also known by transistor size (14nm, 20nm, 5nm)

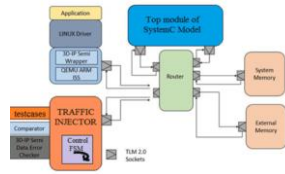
Step 4 : Optimization

- Path equalization and re-sizing
- Logic level power optimization and more

Step 5 : Test logic insertion

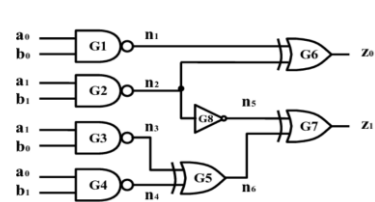
- Insertion of logic to support DFT (design for testability), such as scan chains

ASIC Digital Design Flow

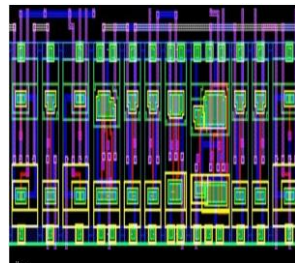


System Architecture Exploration

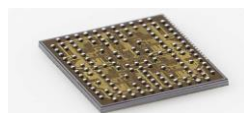
```
module Compare1 (Equal, Alarger, Blarger, A, B);
  input  A, B;
  output Equal, Alarger, Blarger;
  assign Equal = (A & B) | (~A & ~B);
  assign Alarger = (A & ~B);
  assign Blarger = (~A & B);
endmodule
```



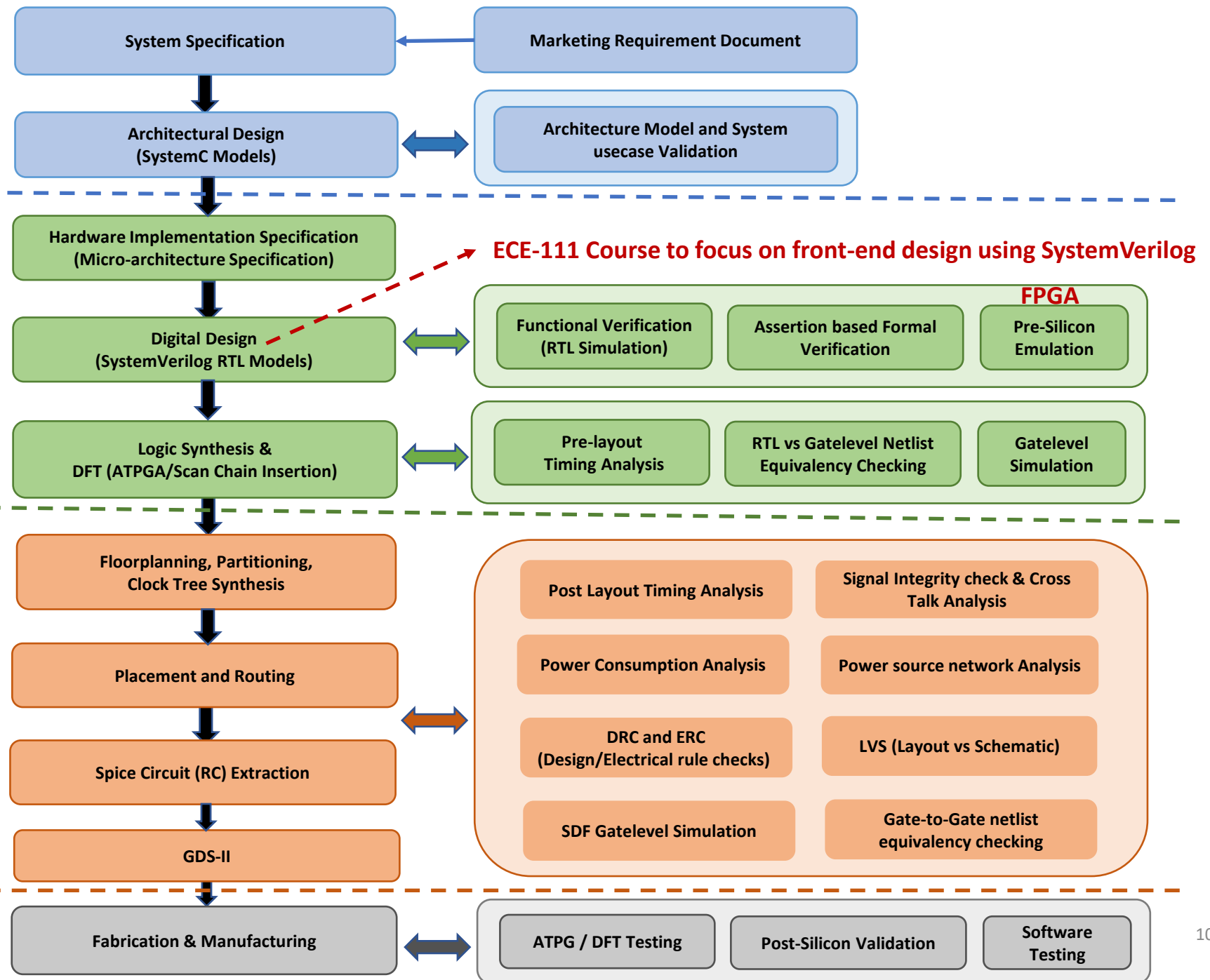
Front-End Design & Verification



Back-End Design / Physical Design / Implementation



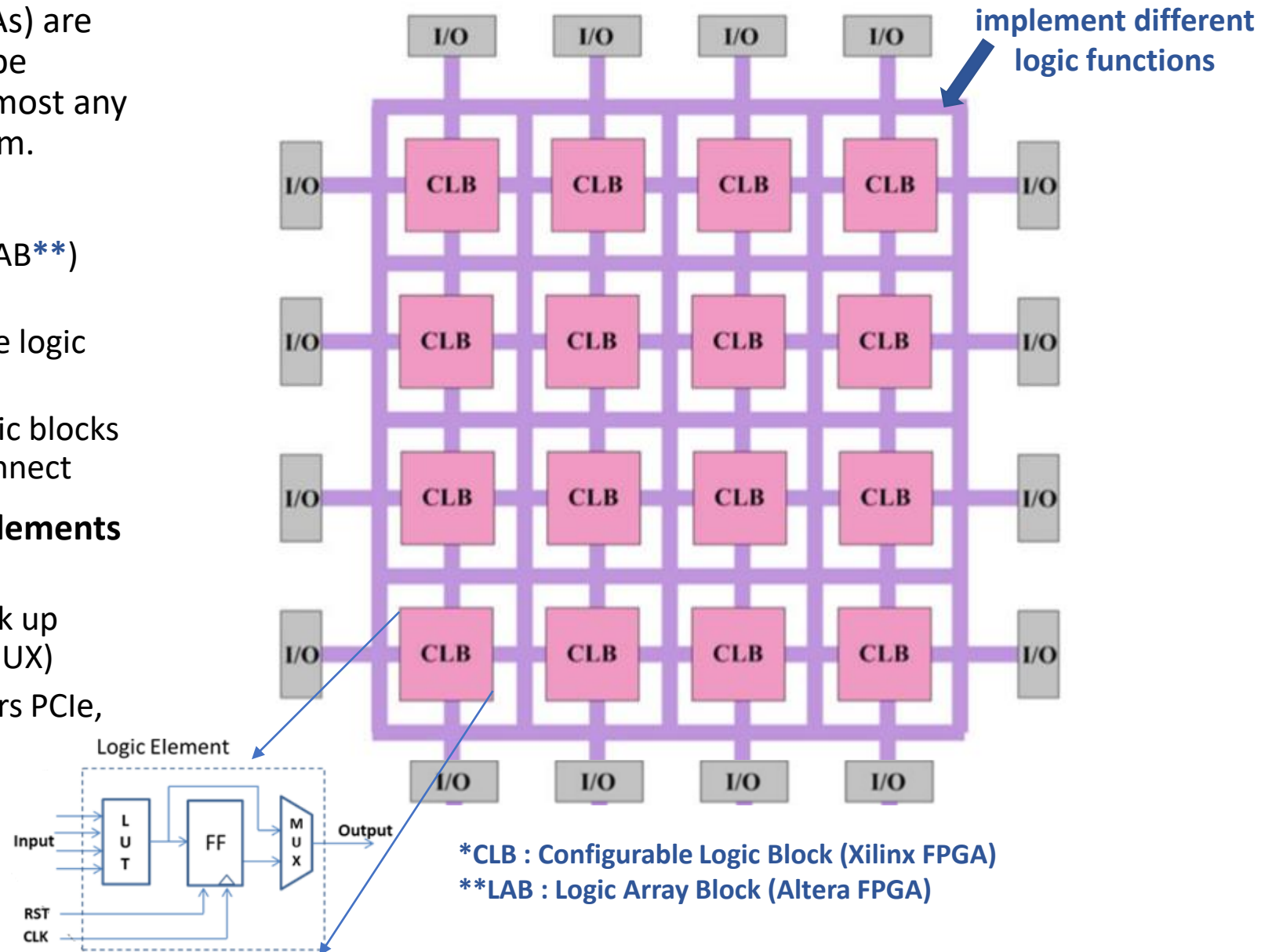
IC Fabrication



What is an FPGA ?

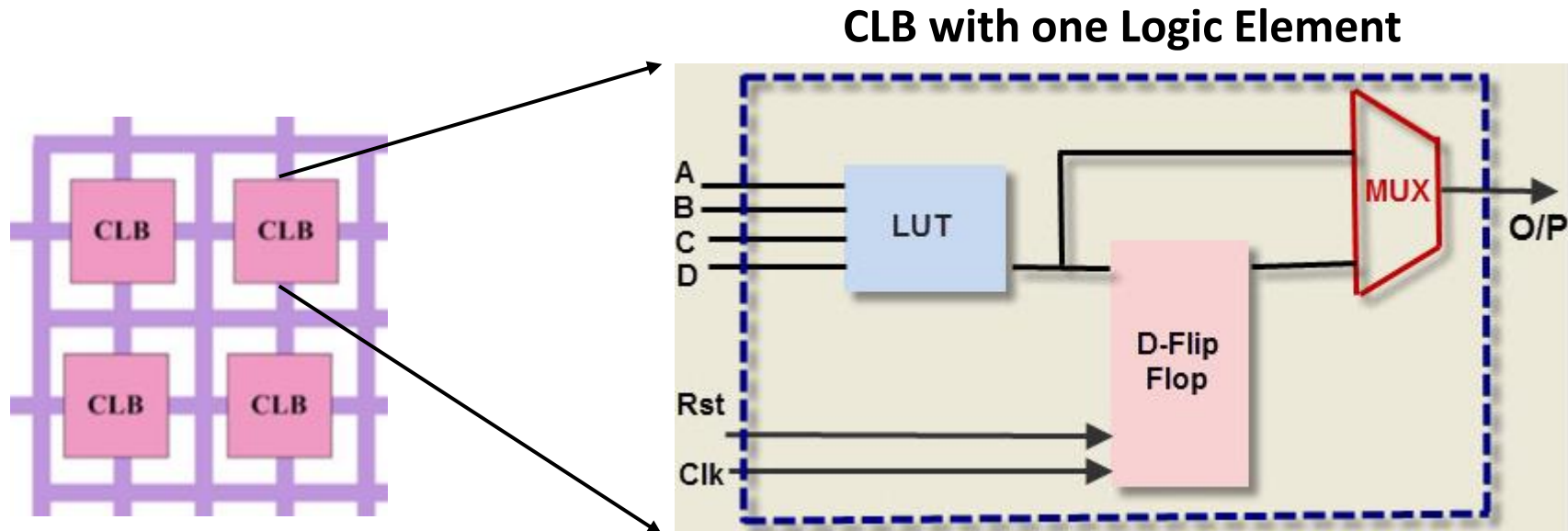
- ❑ **Field programmable Gate Arrays (FPGAs)** are pre-fabricated silicon devices that can be electrically programmed to function almost any kind of a digital circuit or a digital system.
- ❑ **FPGA Comprises of :**
 - Programmable Logic Block (CLB* or LAB**) which implement logic functions
 - Programmable routing connects these logic blocks (CLB's/LAB's)
 - I/O blocks are connected to these logic blocks (CLB's/LAB's) through routing interconnect
- ❑ **Each CLB contains one or more Logic Elements (LE) connected over interconnect**
 - Each LE consists of k-input **LUT's** (Look up tables), **Flipflop** and a **Multiplexer** (MUX)
 - Hard Blocks : DSP's, Multipliers, Adders PCIe, Serdes
- ❑ **FPGA Vendors :**
 - Xilinx, Altera, Actel, Microsemi, Lattice, QuickLogic

FPGA Generic Architecture



Configurable Logic Block (CLB)

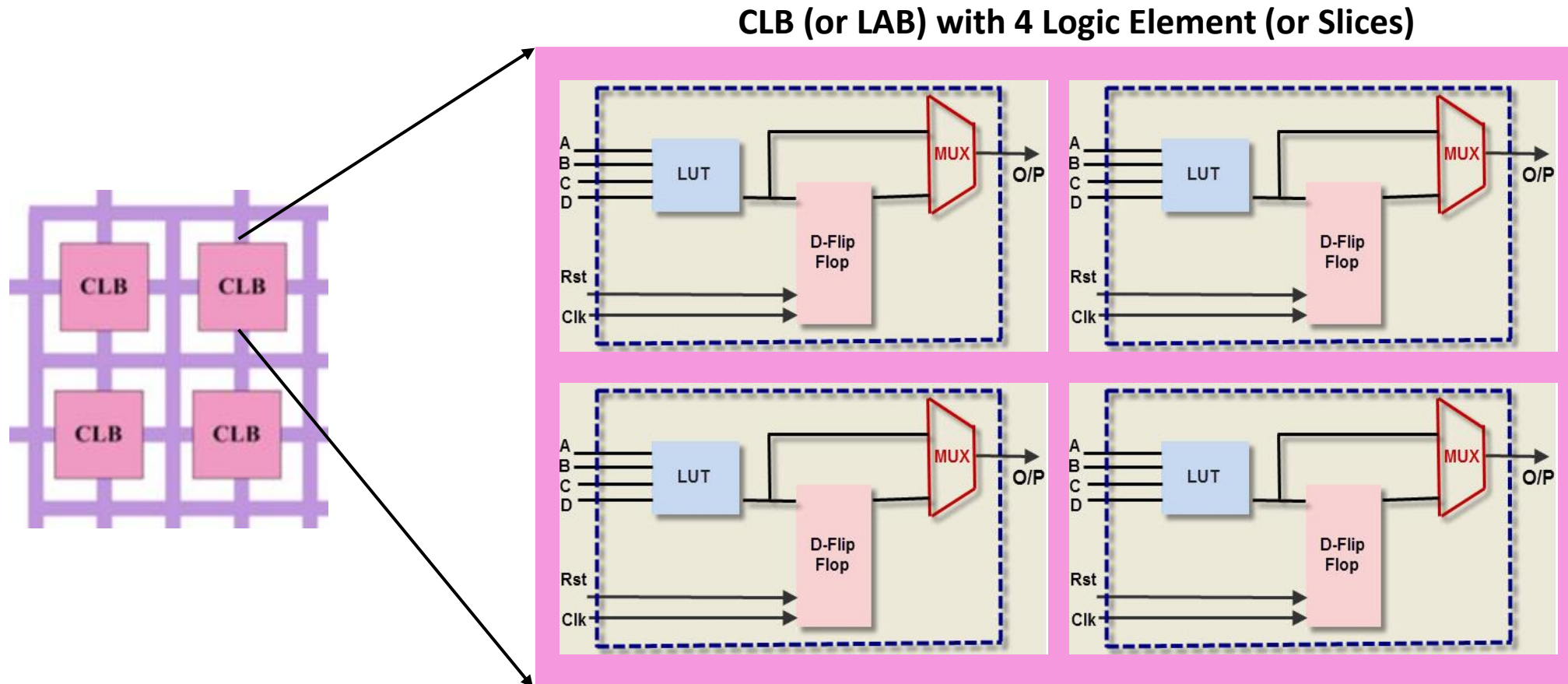
- ❑ CLB comprises of one or more Logic Element (LE) connected over an interconnect
- ❑ Each LE contains :
 - Lookup Table (LUT)
 - D-Flip Flop
 - MUX which allows selection of either the LUT output or the D-Flip Flop output
 - **Note** : In Altera FPGA, a CLB is called as **LAB** (Logic **A**rray **B**lock)



Configurable Logic Block (CLB)

❑ CLB can contain multiple Logic Elements

- In Xilinx FPGA Logic Elements are known as Slices. There can be multiple slices within single CLB.
- **Higher the count of Logic Elements means more digital logic functions can be implemented inside an FPGA**
 - In Altera Arria II GX FPGA devices there are 43,000 to 244,000 Logic Elements
 - In Altera Stratix 10 FPGA device family supports up to 5.5 Million Logic Elements



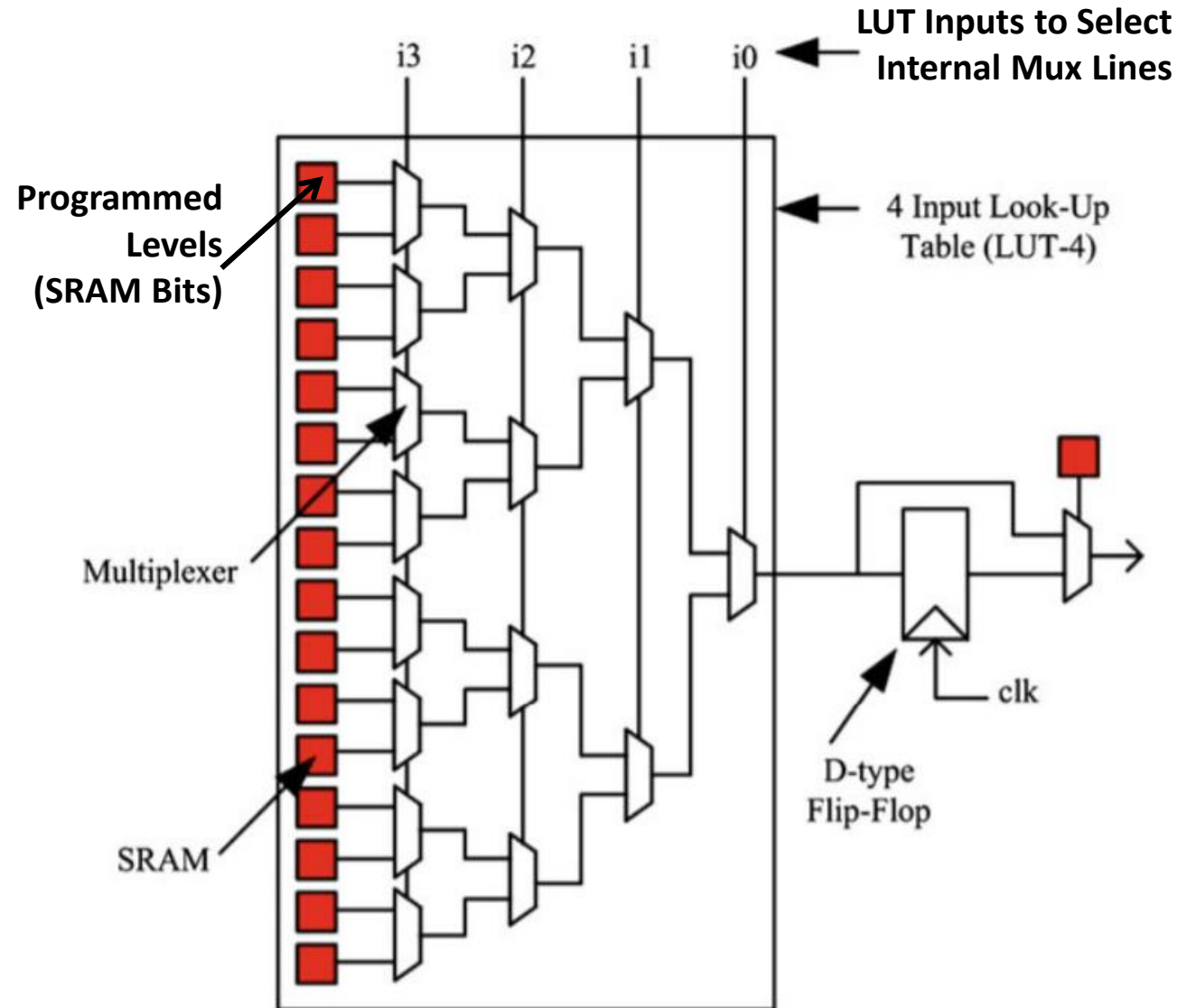
Typical LE (BLE) Example

❑ A typical LE comprises of a 4 input LUT (LUT-4) and a D-type Flip-Flop.

- LUT consists of cascaded Multiplexors
- LUT inputs are select lines for internal multiplexors (MUX)
- The LUT-4 uses 16 SRAM bits to implement any 4 input Boolean function.
- The output of LUT-4 is connected to an optional Flip-Flop. A multiplexor selects the **BLE** output to be either the output of a Flip-Flop or the LUT-4

❑ Modern FPGAs contain typically 4 to 10 BLEs in a single cluster

Note : BLE is same as LE

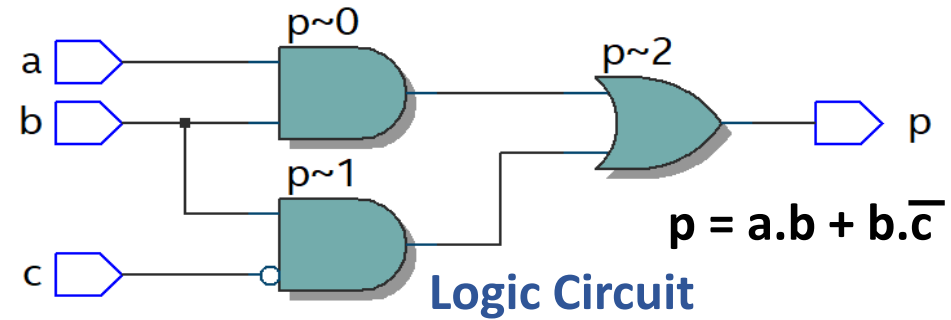


4 input LUT based **BLE (Basic Logic Element)**

LUT (Look Up Table) Operation

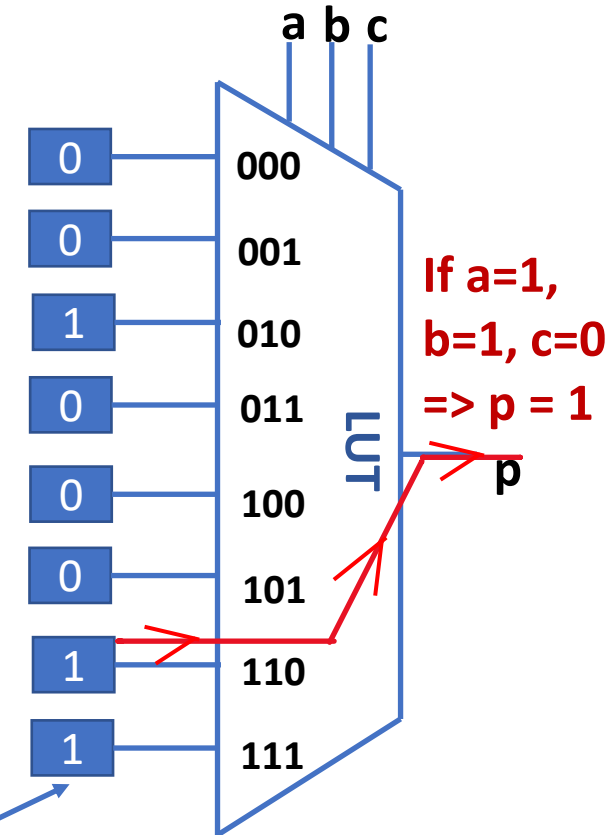
- ❑ LUT can implement any Boolean expression
- ❑ LUT is made up of two main components :
 - Series of cascaded multiplexer(s)
 - SRAM cells to program input values
- ❑ LUT inputs are used as select lines to multiplexer
- ❑ Input to multiplexer is programmed to logic 0 or 1 (these are programmed levels stored in SRAM cells within FPGA)
- ❑ logic is called a look up table because output is selected by looking at correct programming levels and input select lines
- ❑ Example on this slide is 3-to-1 LUT implementing a boolean expression :

$$p = a.b + b.\bar{c}$$



Truth Table for $p = a.b + b.\bar{c}$

a	b	c	$p = a.b + b.\bar{c}$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

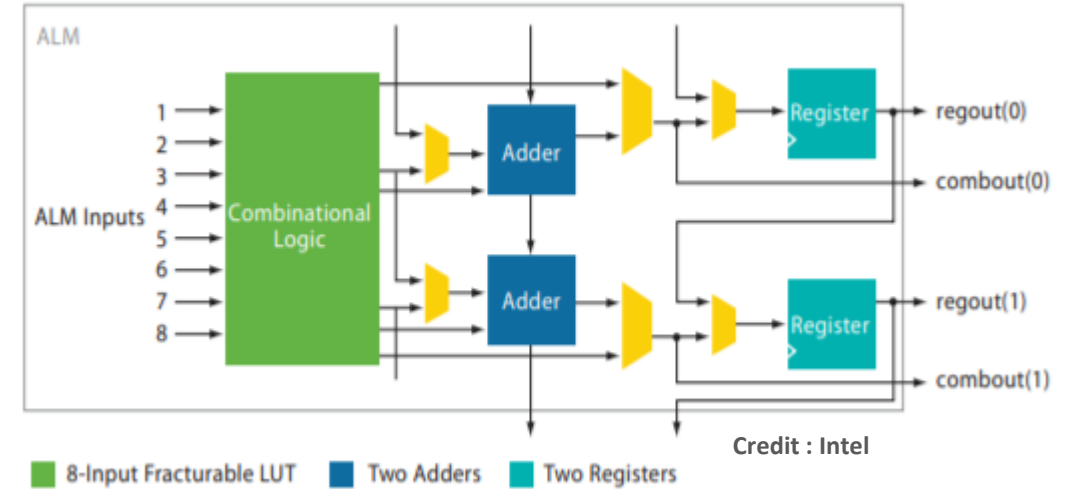


SRAM, programmed with value 0xC4 (1100_0100) to implement $p = a.b + b.\bar{c}$

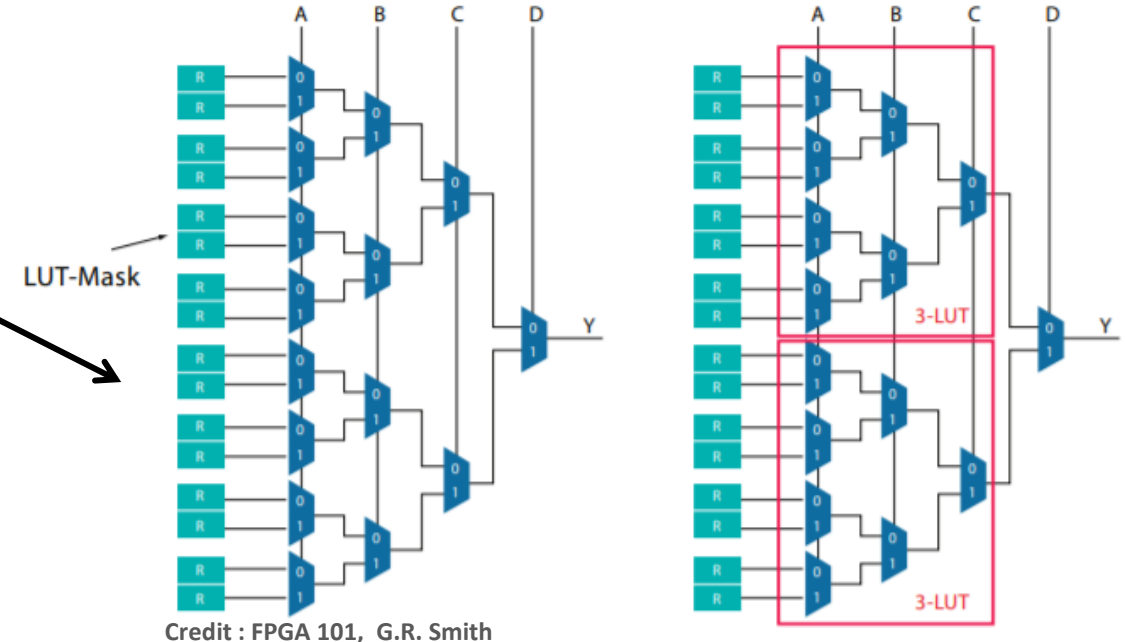
Adaptive LUT

- ❑ Altera introduced the 8-input fracturable look-up table (LUT) with the Stratix® II family.
- ❑ The ALM can also be efficiently partitioned into independent smaller LUTs to implement one or two logic functions
- ❑ It provides the performance advantage of larger LUTs and the area efficiency of smaller LUTs.
- ❑ Diagram shows, 4 input ALUT is broken into two halves to implement 3 to 1 LUT
- ❑ ALM also consists of higher efficient adder logic and two registers
- ❑ **Note :** the fast adder carry chain in ALM (does not require going out to programmable switch boxes)

8 input Adaptive Logic Module (ALM)

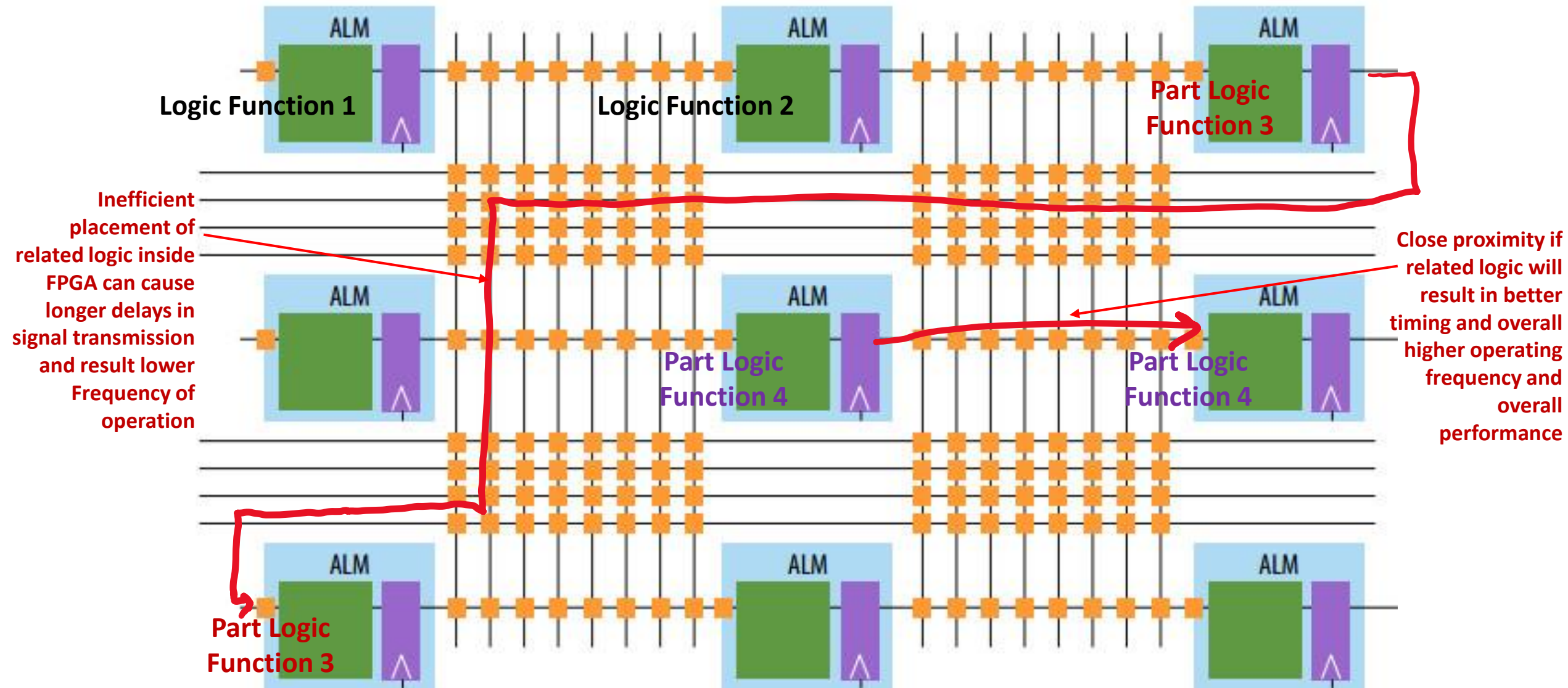


4 input Adaptive LUT divided into two 3 to 1 LUT



$$a'b'c'd' + abcd + abc'd' = 1000\ 0000\ 0000\ 1001 = 0x8009$$

Connectivity between ALMs



Altera Arria II Family Devices Capacity and Features

Table 1–1. Features in Arria II Devices

Feature	Arria II GX Devices						Arria II GZ Devices		
	EP2AGX45	EP2AGX65	EP2AGX95	EP2AGX125	EP2AGX190	EP2AGX260	EP2AGZ225	EP2AGZ300	EP2AGZ350
Total Transceivers (1)	8	8	12	12	16	16	16 or 24	16 or 24	16 or 24
ALMs	18,050	25,300	37,470	49,640	76,120	102,600	89,600	119,200	139,400
LEs	42,959	60,214	89,178	118,143	181,165	244,188	224,000	298,000	348,500
PCIe hard IP blocks	1	1	1	1	1	1	1	1	1
M9K Blocks	319	495	612	730	840	950	1,235	1,248	1,248
M144K Blocks	—	—	—	—	—	—	—	24	36
Total Embedded Memory in M9K Blocks (Kbits)	2,871	4,455	5,508	6,570	7,560	8,550	11,115	14,688	16,416
Total On-Chip Memory (M9K + M144K + MLABs) (Kbits)	3,435	5,246	6,679	8,121	9,939	11,756	13,915	18,413	20,772
Embedded Multipliers (18 x 18) (2)	232	312	448	576	656	736	800	920	1,040
General Purpose PLLs	4	4	6	6	6	6	6 or 8	4, 6, or 8	4, 6, or 8
Transceiver TX PLLs (3), (4)	2 or 4	2 or 4	4 or 6	4 or 6	6 or 8	6 or 8	8 or 12	8 or 12	8 or 12
User I/O Banks (5), (6)	6	6	8	8	12	12	16 or 20	8, 16, or 20	8, 16, or 20
High-Speed LVDS SERDES (up to 1.25 Gbps) (7)	8, 24, or 28	8, 24, or 28	24, 28, or 32	24, 28, 32	28 or 48	24 or 48	42 or 86	0 (8), 42, or 86	0 (8), 42, or 86

Modern FPGA's

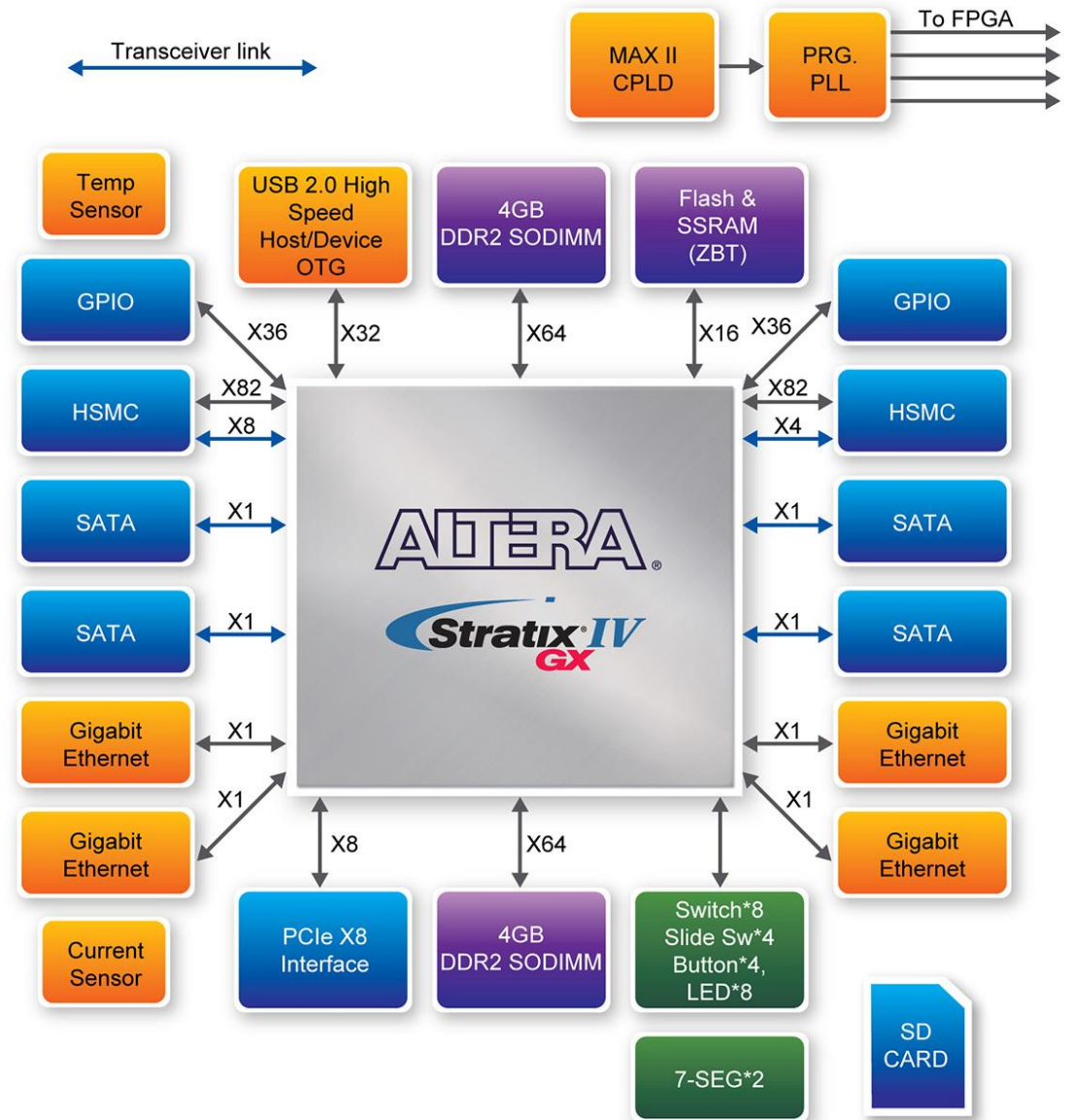
❑ Modern FPGAs have many built-in interfaces :

- DRAM
- PCI Express
- USB
- SATA (disk drives) and more

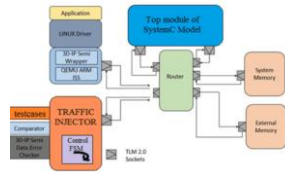
❑ Modern FGPA's such as Altera Stratix IV family them easy to integrate into compute environments due to wide variety of built-in interface

❑ Modern FPGA's contain built-in **Hard IP blocks**

- Such as : DDR memory controller, high performance multipliers, adders, DSP blocks, Quad core ARM CPU's, High speed SERDES (Serializer/de-serializer), and more
- **These Hard IP blocks** are designed optimally to efficiently perform specific functions such as multiplication, division, 3D Fourier transform function, low latency memory access, high speed serialization/de-serialization
- If hard IP blocks are unused, then they end up wasting huge amount of logic and routing resources.



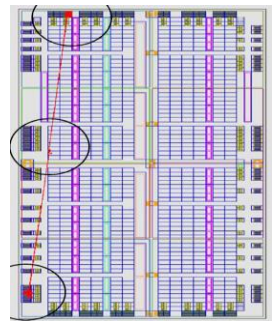
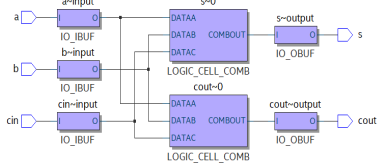
FPGA Digital Design Flow



**System
Architecture
Exploration**

```
// Full Adder SystemVerilog Module
1 module fulladder(input logic a, b, cin,
2 output logic s, cout);
3 logic p, g; // internal nodes
4
5 assign p = a ^ b;
6 assign g = a & b;
7
8 assign s = p ^ cin;
9 assign cout = g | (p & cin);
10 endmodule
11
12
13
```

**Design
Entry &
Synthesis**



Implementation

**Download
Bitstream to FPGA**

System Specification

Marketing Requirement Document

Architectural Design
(SystemC Models)

Architecture Model and System
Usecase Validation

Hardware Implementation Specification
(Micro-architecture Specification)

Digital Design (SystemVerilog RTL Models or
Schematic captures)

Functional Verification
(RTL Simulation)

Synthesis

Post Synthesis
Simulation

Mapping

Place and Route
(Design Fitting)

Post Implementation
Timing Simulation

Generate Programming Bitstream File



Run Software

This step is only
required if final
product is FPGA
based platform

Simulation Speed

FPGA Synthesis

❑ FPGA Synthesis is a 3-step process

- Design check and resource association
- Optimization
- Technology mapping

SystemVerilog .sv source files

```
1 // Full Adder SystemVerilog Module
2 module fulladder(input logic a, b, cin,
3                 output logic s, cout);
4     logic p, g; // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11 endmodule
12
13
```

FPGA Synthesis 3-Step Flow

Step 1 : Design check and resource association

- Check for syntax errors in design source files
- Check if design provided can be synthesized
- Associate design to logic cell and blocks

Step 2 : Optimization

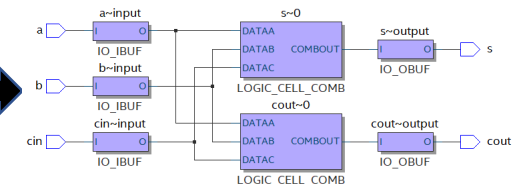
- Reduce logic
- Eliminate redundant logic
- Make design smaller and faster (best Fmax)

Step 3 : Technology Mapping

- Connect design to logic
- Predict and add timing estimates
- Create output reports and netlists

Credit : FPGA 101, G.R. Smith

Synthesized output netlist



FPGA Implementation

❑ Mapping :

- Compares the resources specified in input synthesized netlist and checks for available resources of the target FPGA
 - **Insufficient resources will result in errors !!**
- Divides the netlist circuit into sub-blocks to fit into FPGA logic blocks

❑ Place and Route

- Also known as design fitting. This is the most challenging and intensive part of FPGA design flow
- Physically places the sub-blocks in netlist generated from mapping stage to FPGA logic blocks
- Routes signals between logic blocks considering timing constraints
- FPGA tools provides choices to user to specify fitting criteria's and based on that logic will be mapped to FPGA resources. These fitting criteria are:
 - High performance (speed), Smallest area, Low power, Balanced

❑ Generate programming file

- Generate bitstream file or IEEE-1532 configuration file (.isc).
- Download to FPGA either through JTAG or downloaded to non-volatile memory on FPGA board which upon reset will automatically program FPGA

FPGA Netlist Viewer

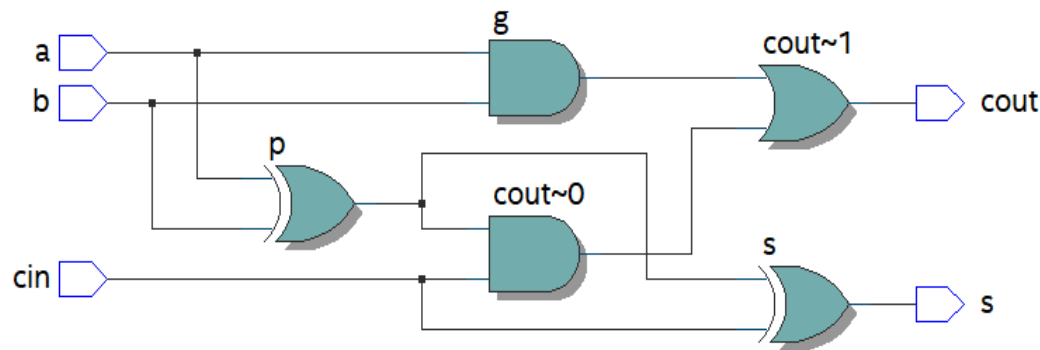
❑ FPGA Tools provides netlist viewer post synthesis and implementation :

- RTL netlist viewer
- Post-fitting netlist viewer
- Post-mapping netlist viewer

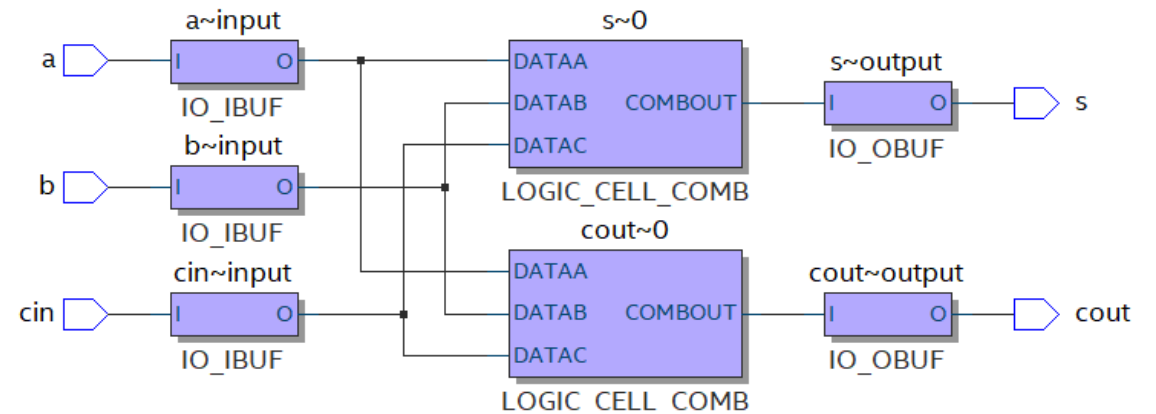
SystemVerilog .sv source files

```
1 // Full Adder SystemVerilog Module
2 module fulladder(input logic a, b, cin,
3                 output logic s, cout);
4     logic p, g; // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11 endmodule
12
13
```

RTL Netlist Viewer



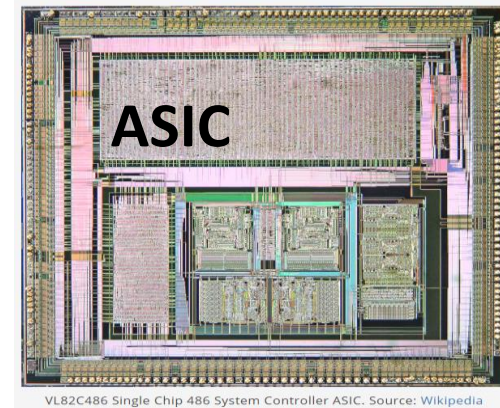
Post-Fitting Netlist Viewer



FPGA vs ASIC Comparison



Versus



❑ Advantages :

- Low Time-to-market
- High flexibility
- High reusability
- High IO count and massive parallelism
- High NRE (Net Return of Engineering Cost)

❑ Dis-advantages :

- Low Performance
- High power consumption
- High Area
- Higher Cost (can cost 4K USD)
- Cannot implement Analog blocks

- ❑ **Application :** Whenever fast turnaround time required with performance tradeoff. HW (SOC/IP) prototyping to prove concept, validate design, early SW validation platform, FPGA based Clouds, ML

❑ Advantages :

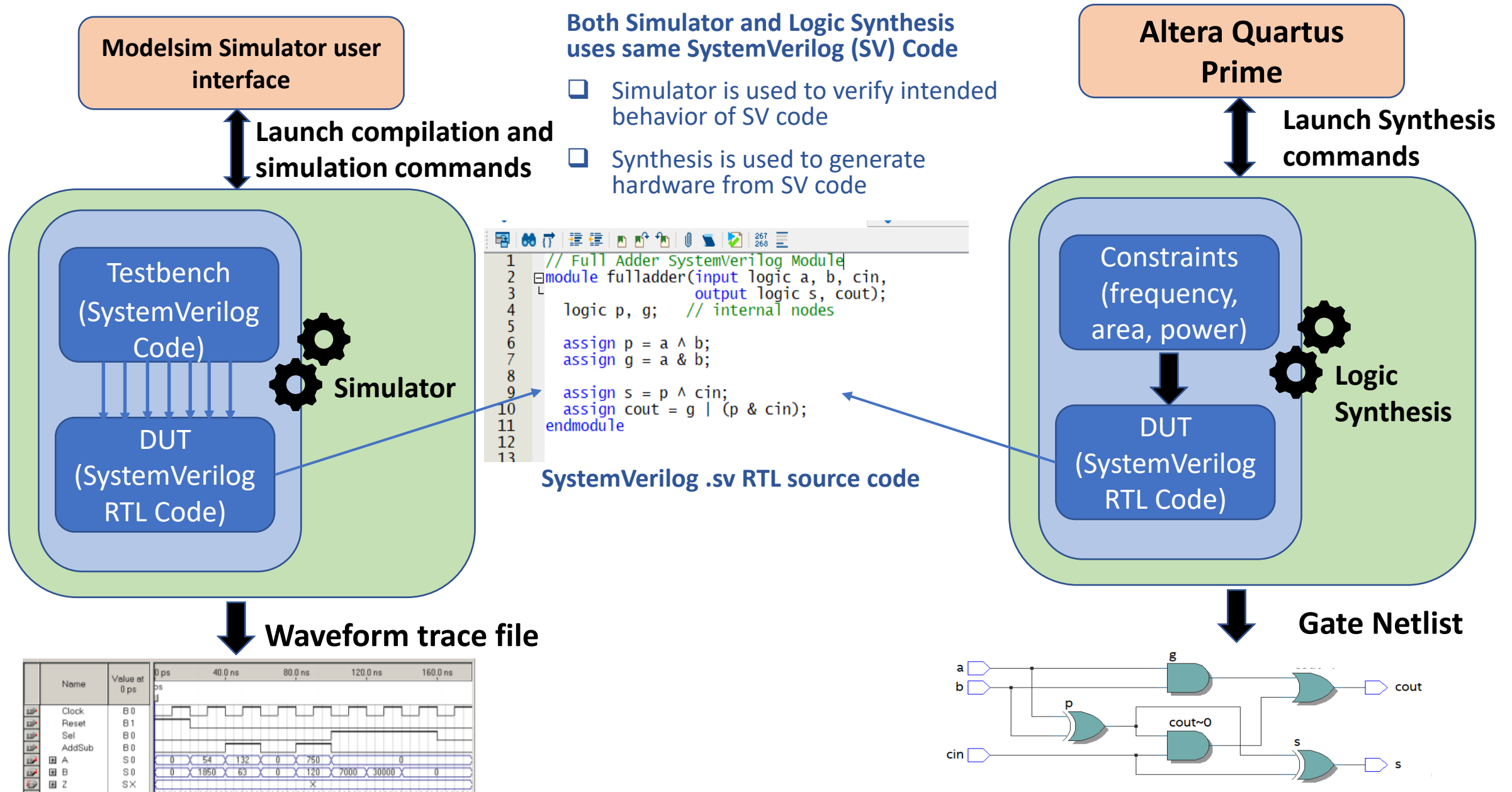
- High Performance
- Low power consumption
- Low Area
- High volume production due to lower cost per unit
- Can implement Analog blocks

❑ Dis-advantages :

- High Time-to-market
- Low flexibility
- Low reusability
- Low NRE

- ❑ **Application :** High Speed designs, high packing density, used in larger numbers (CPU, Modern mobile chipsets, Analog devices, Gigabit Serdes, etc)

Simulator and Logic Synthesis For FPGA Designing



What is Simulation ?

- ❑ **Simulation** is the process of verifying the functionality and timing of a design against its original specifications
 - **Example** : Does 32-bit ALU logic implemented using SystemVerilog perform operations such as addition, subtraction, multiplication, comparison correctly as per the intended specification ?

- ❑ Digital Designer engineer performs functional **simulation** using a **simulator** at various stages of design development flow :
 - Prior to **logic synthesis**, simulation is performed to verify functionality of **HDL** (SystemVerilog) description of design
 - After synthesis, gate level **simulation** is performed on the gate-netlist design generated by synthesis.
 - Gate level simulation is performed to verify design timing and re-check functionality of design

What is the role of a simulator ?

❑ Simulator role is approximating reality. Design realized in virtual environment !

- Simulator creates an artificial universe that mimics the future real circuit design
- It lets the designer interact with the design before it is manufactured, and enables designer to correct flaws and problem earlier
 - Simulator takes SystemVerilog code or a gate level netlist and simulates design using stimulus provided in testbench code
 - It generates waveforms to provide approximate behavior of design prior to manufacturing.
- **Note :**
 - Simulator does not correct automatically incorrect description of the design written in HDL
 - It is the responsibility of verification engineer, not the simulator, to apply legal set of stimulus to design under test based on real world usecase.

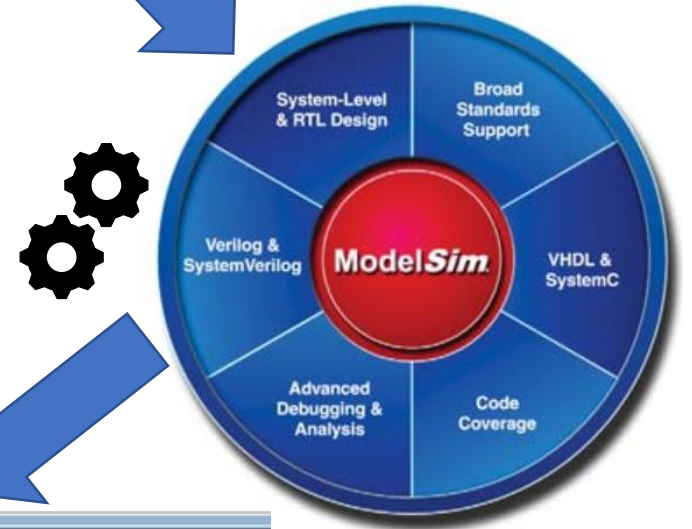
Waveform

- ❑ **Waveform** is a plot of a signal over period of time
- ❑ **Waveform** is generated by the **simulator** at the end of RTL or gate level **simulation**
- ❑ Designer and Verification engineers review waveforms to analyze the behavior of a design and also to perform debugging of potential bug in the HDL code

SystemVerilog (SV) RTL Code & Testbench Code

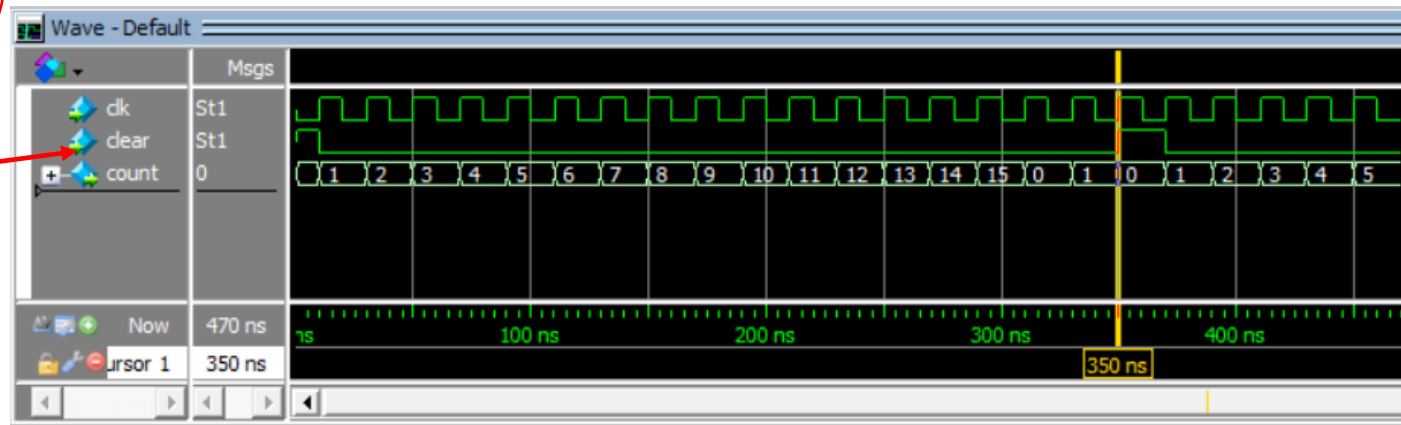
```
// SystemVerilog code for 4-bit counter
module counter ( input clk, input clear, output [3:0] count);
always @ (posedge clk) begin
    if (clear==1)
        count <= 0;
    else
        count <= count + 1;
end
endmodule
```

**ModelSim
Simulator**



**Waveform generated by
Modelsim Simulator at
end of simulation**

**Lets review
waveform to see
if SV design
working like a
4-bit counter**



Learning Resources on FPGA Architecture

- ❑ Basics of programmable logic : FPGA Architecture (Intel Altera)
 - <https://www.youtube.com/watch?v=jbOjWp4C3V4>

- ❑ Altera FPGA Architecture :
 - <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf>
 - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/arria-ii-gx/aiigx_51001.pdf