

Lecture 2: ASIC, FPGA Architecture, and Logic Synthesis

UCSD ECE 111 Advanced Digital Design Project

Prof. Farinaz Koushanfar

Fall 2024

Some slides are courtesy of Prof. Lin, Prof.
Turakia. or Prof. Karna ECE111 courses

Weekly Announcement(s)

Week 1 (oct 1 2024)

Important announcements (Details in Canvas)

- **TA office hours:** are now MWF 9-11am for Fall'24
 - Zoom Meeting ID: 948 6397 0932; Passcode 004453
 - <https://ucsd.zoom.us/j/94863970932?pwd=iXcQXbLYjOaAQTdOJlrmglCYMSyeir.1>
- **TA Discussion session:** Wed 4-4:50PM (starting Oct 9) Location: CNTRE 214
- **September 30:** The #FinAid survey has been added to the Quizzes section on Canvas for the commencement of academic activity reporting.
 - Due date this coming Friday, **10/4/24**.
- **Oct 1 (Today):** Homework 1 will be posted on Canvas
 - Due in 1 week from now on Tuesday, **10/8/24**

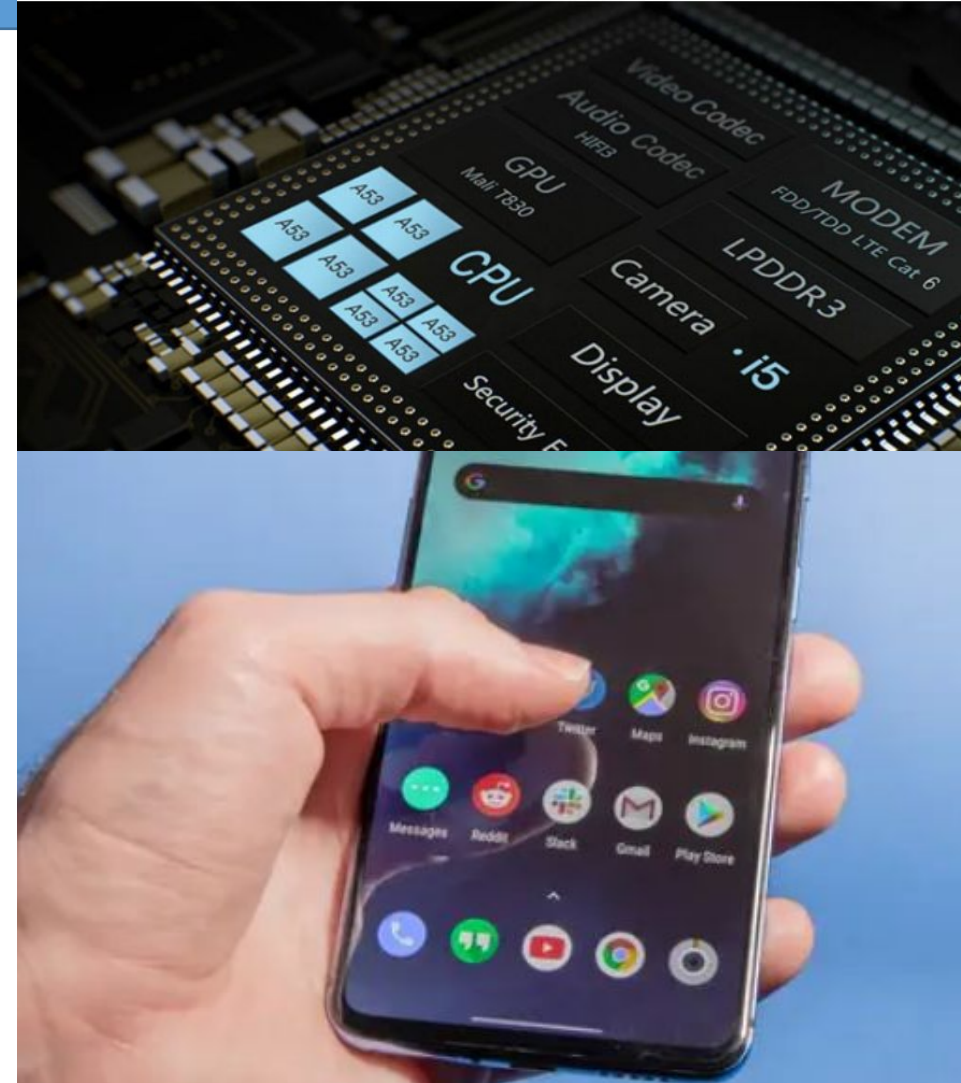
Homework 1 overview

- This assignment will help you familiarize with digital design at three different abstraction levels
 - Behavioral, dataflow and gate
 - You will also learn to synthesize models and review resource
- Learn how to find usage summary for different models
 - Such as the number of Adaptive Look Up Tables (ALUTs) and inputs that are used to implement Boolean functions).
- Learn to simulate all the different models separately in Modelsim-Altera using the testbenches we have provided
- Learn to review the simulation results

What is ASIC?

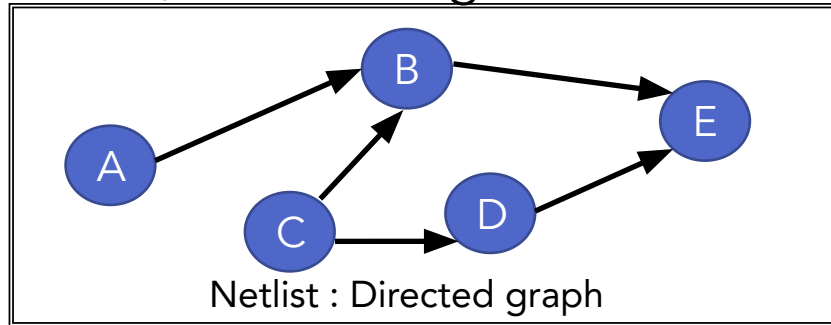
ASIC (Application Specific Integrated Circuit)

- As name suggests, ASICs are application specific circuits
- Designed for one sole purpose for its entire operating lifetime
 - Example: A mobile processor inside a mobile phone is an ASIC and is meant to function as a mobile processor for its entire life
- Logic function of an ASIC is typically specified using HDLs, such as SystemVerilog, Verilog or VHDL



What is a netlist?

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



edge



Interconnections

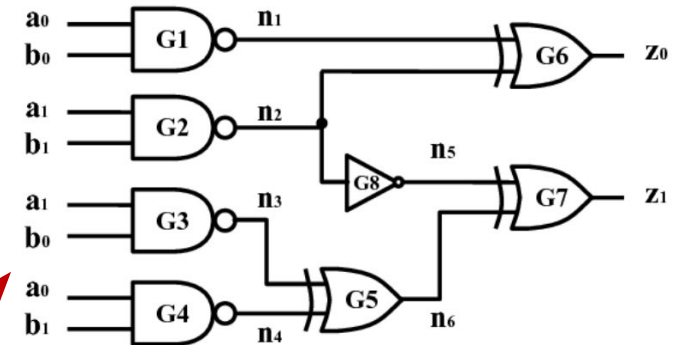


Vertex

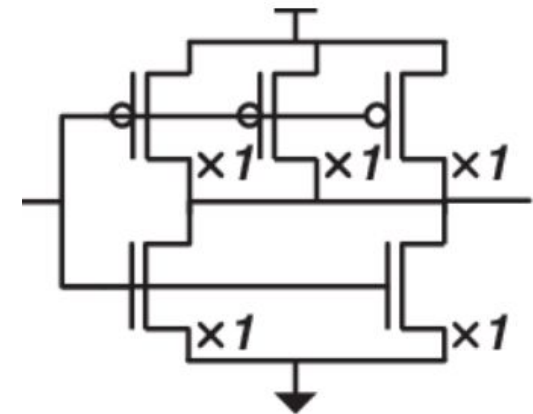
SystemVerilog modules or logic gates or transistors

- 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



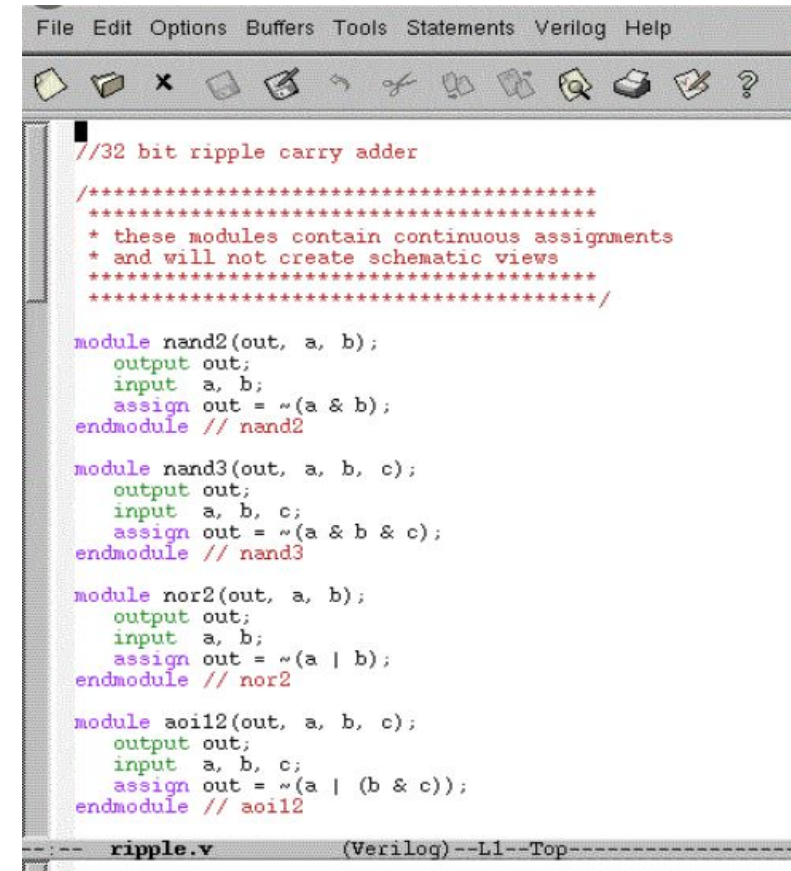
Transistor Level Netlist



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 aoii2(out, a, b, c);
    output out;
    input  a, b, c;
    assign out = ~(a | (b & c));
endmodule // aoii2

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


What is synthesis?

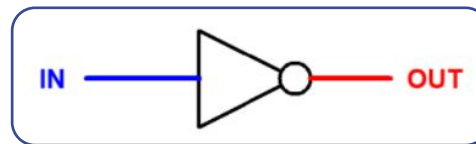
- Synthesis is a process to convert design from one form to another
- Design netlist can be transformed from one level to another using “Synthesis”
 - Logic Synthesis : Converts SystemVerilog netlist to gate-level netlist representation of design
 - Physical Synthesis : Converts gate-level netlist to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

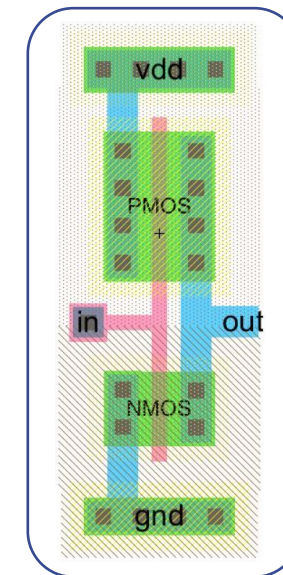
Logic Synthesis

Gate-Level Inverter Design Netlist

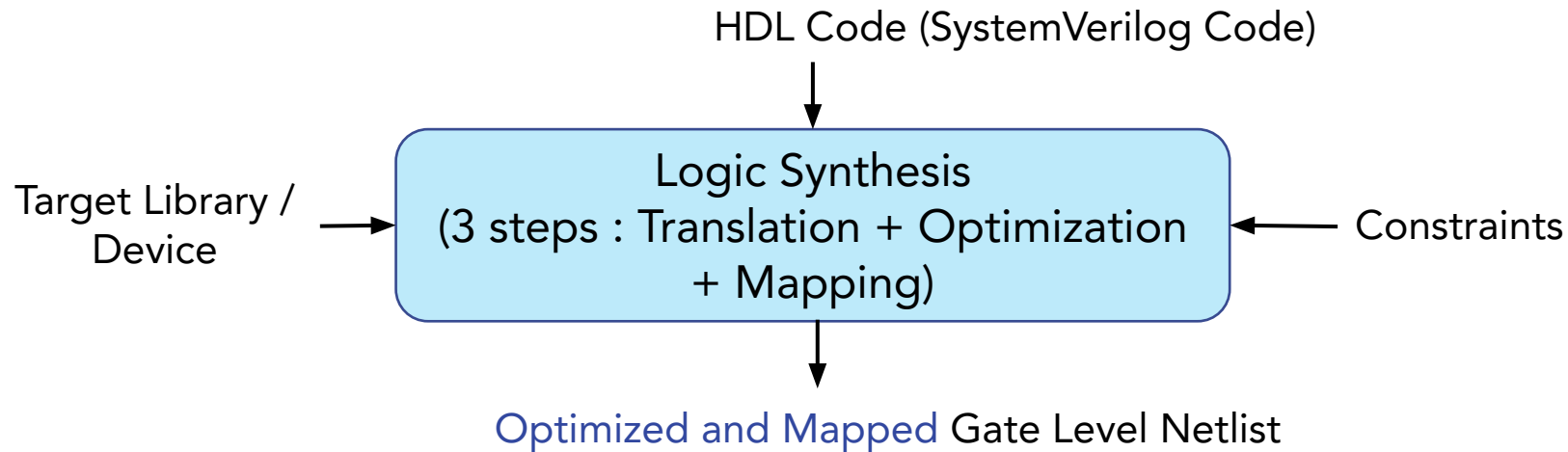


Physical Synthesis

Transistor-Level Inverter Design Netlist



What is logic synthesis?



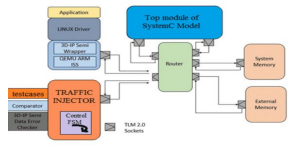
- Logic synthesis is the process of converting HDL description of a design into an optimized structured gate-level representation using technology specific primitives:
 - Output of logic synthesis process is optimized and mapped gate-level netlist
 - For FPGAs Synthesis uses : Look-Up-Tables (LUTs), 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

What is Logic Synthesis ?

- **Logic Synthesis Objectives and 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)

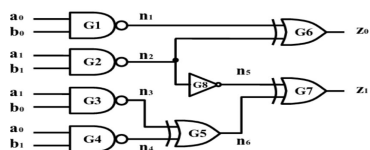
ASIC Digital Design Flow

Design Accuracy

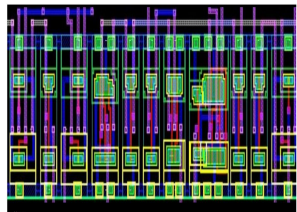


**System
Architecture
Exploration**

```
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
```

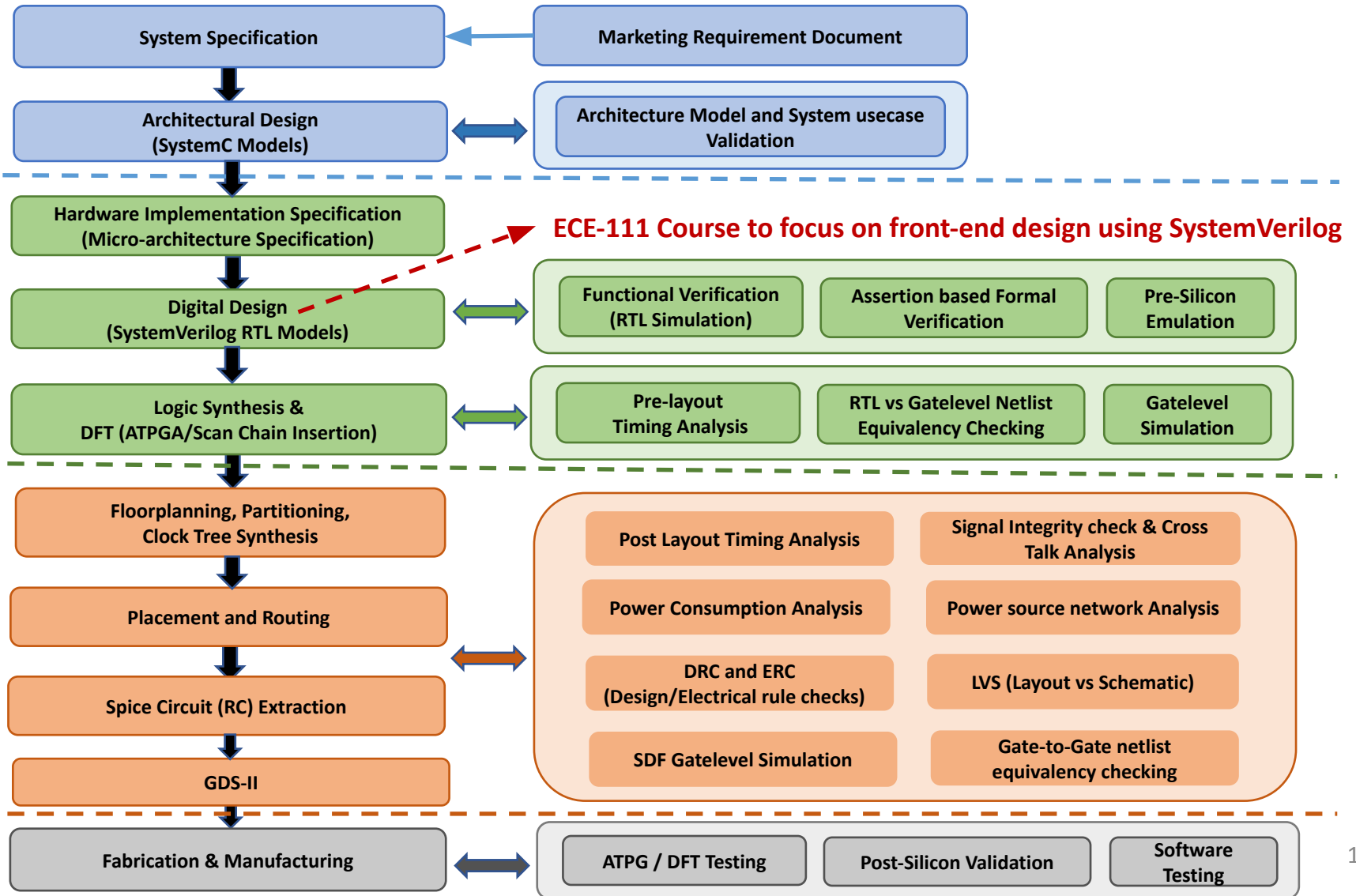


Front-End Design



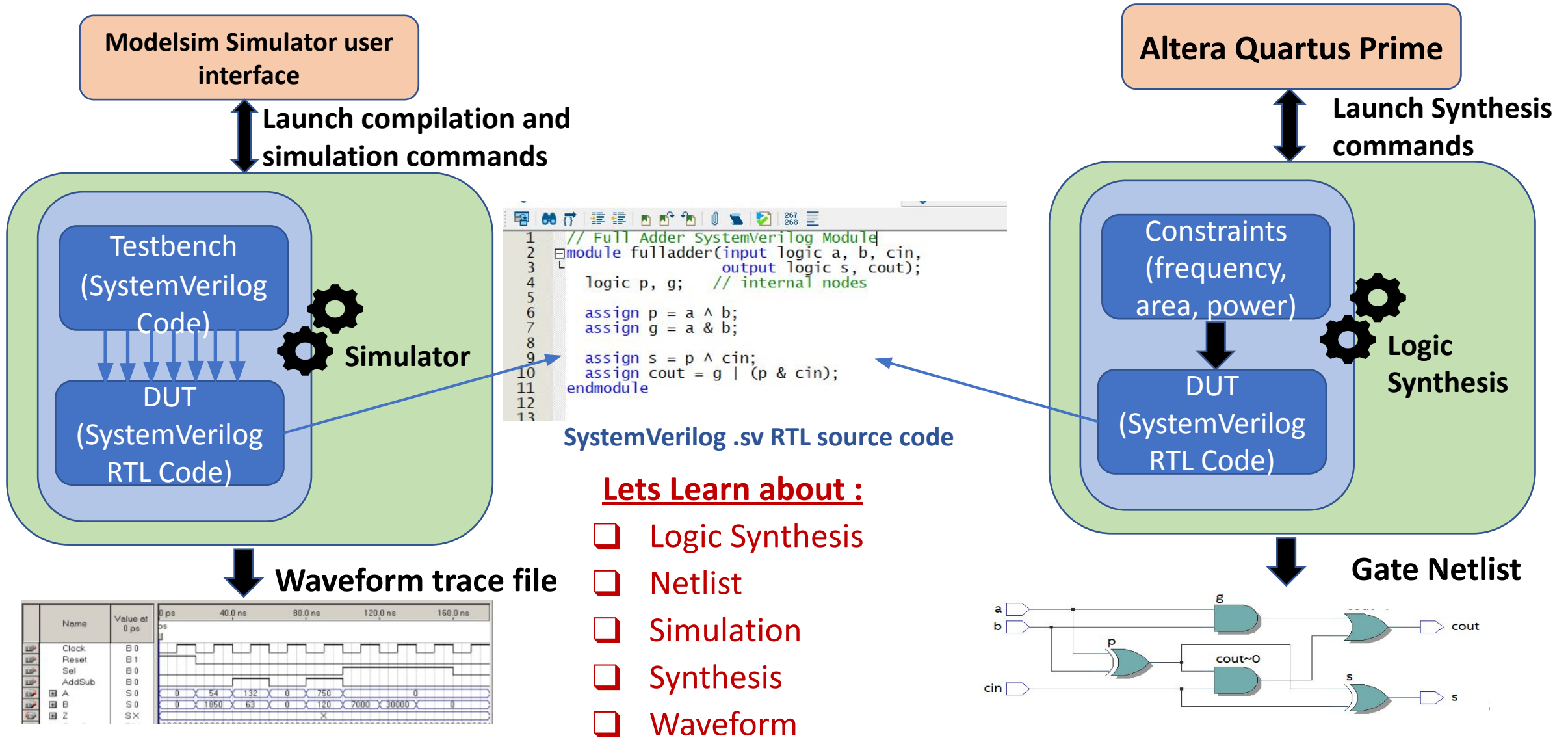
**Back-End
Design / Physical
Design /
Implementation**

IC Fabrication



Simulation Speeds

Simulation and Synthesis

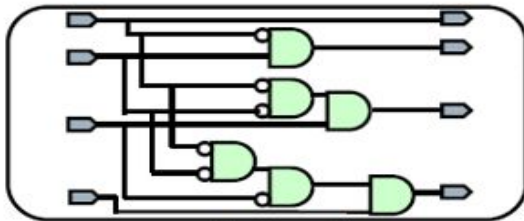


ASIC Synthesis Flow

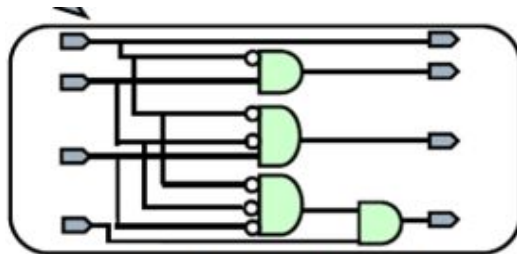
SystemVerilog .sv source files

```
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
```

No Timing Info



Generic Boolean
(GTECH)



Target Technology

Timing Info

Step 1 : Translation

- Check if design provided can be synthesized
- Map RTL to unmapped gate-netlist
- Uses standard GTECH library which has and, or, etc gates and designware library which has adders, comparators, etc 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

Step 4 : Optimization

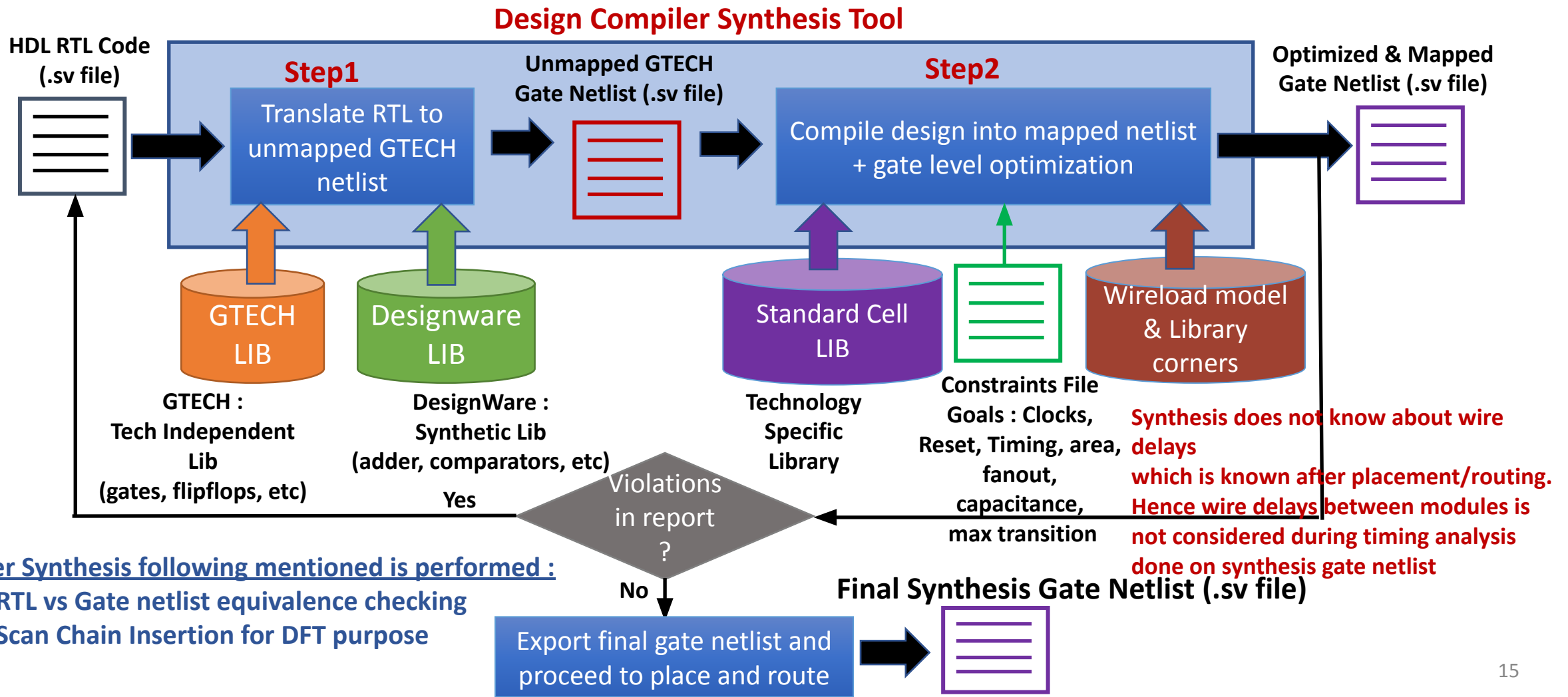
- Path equalization and re-sizing
- Logic level power optimization and more

Step 5 : Test logic insertion

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

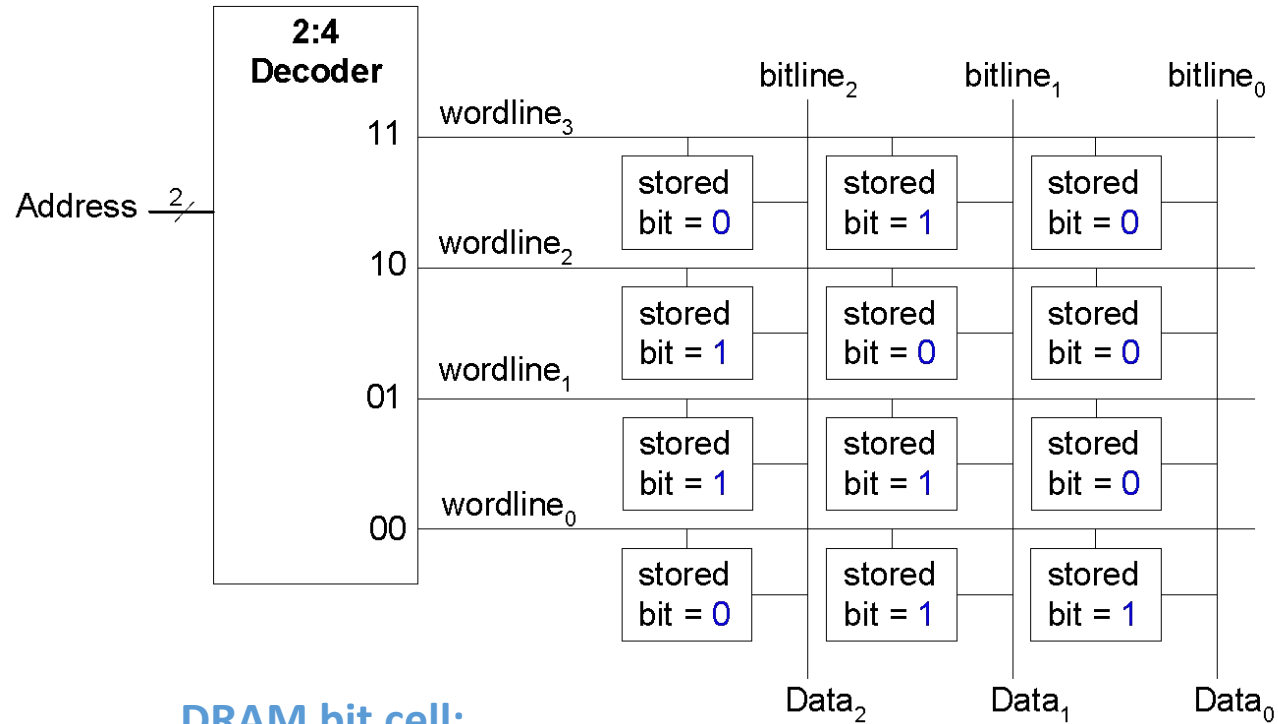
ASIC Synthesis Flow (Design Compiler Tool)

- ❑ Synthesis flow to convert RTL description into a structural gate level mapped and optimized netlist.

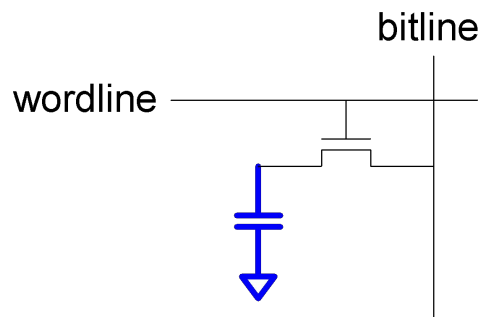


What is Programmable Logic?

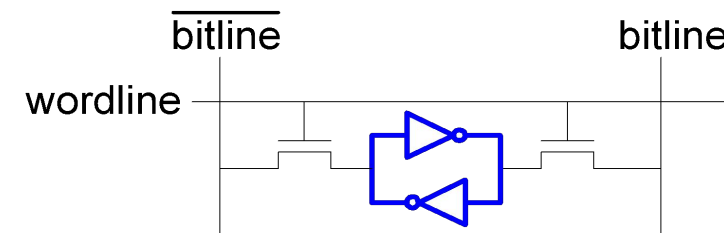
Memory Arrays Review



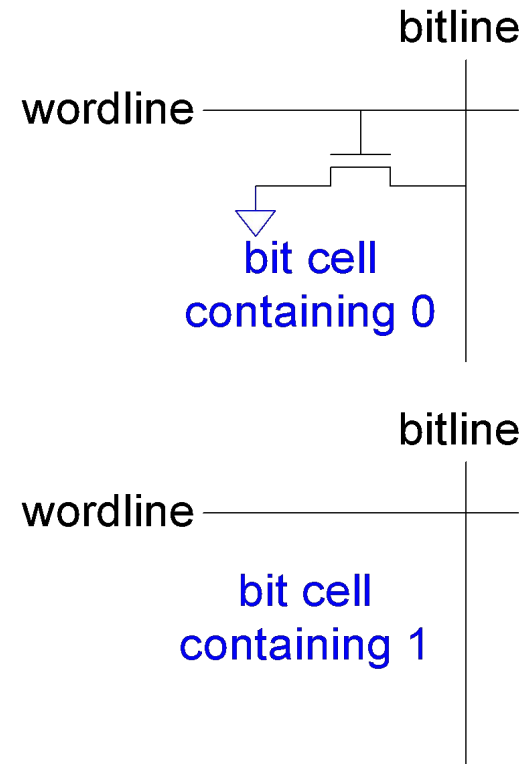
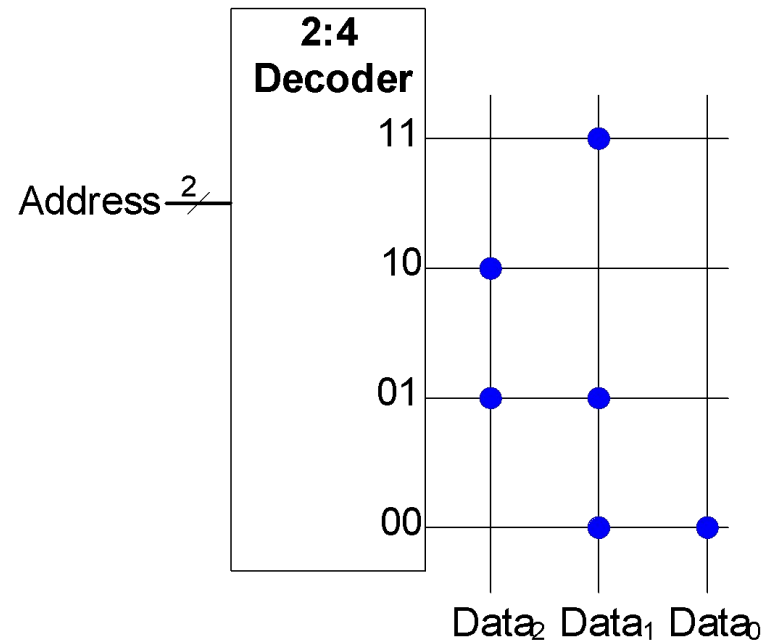
DRAM bit cell:



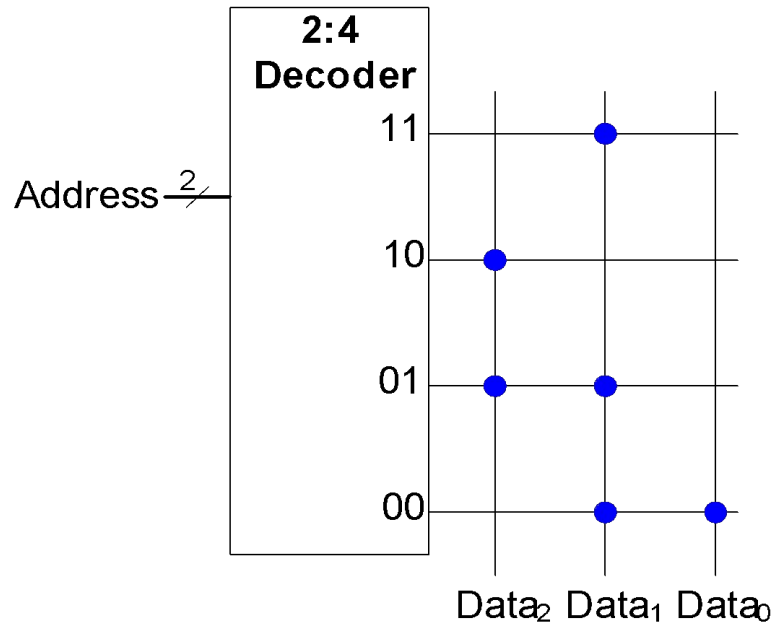
SRAM bit cell:



ROM: Dot Notation

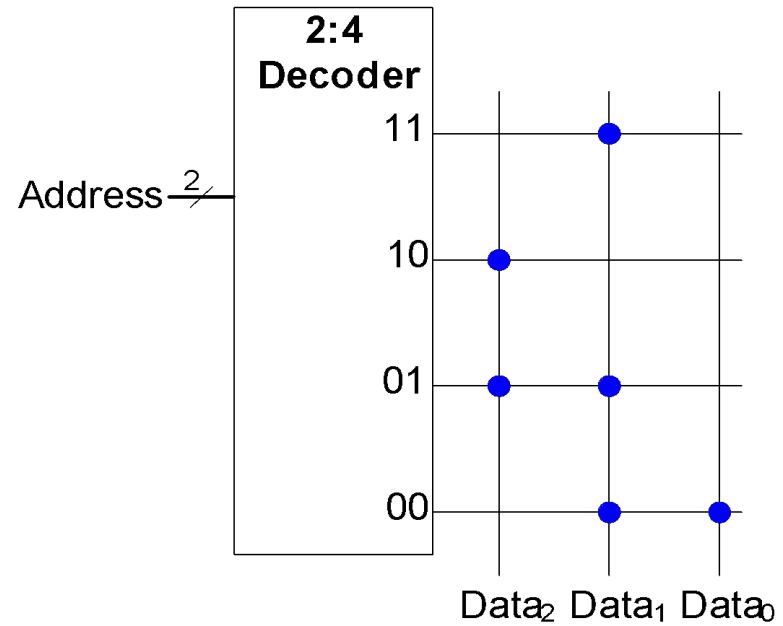


ROM Storage



Address	Data			depth ↑ ↓
11	0	1	0	
10	1	0	0	
01	1	1	0	
00	0	1	1	
width ←→				

ROM Logic



$$Data_2 = A_1 \oplus A_0$$

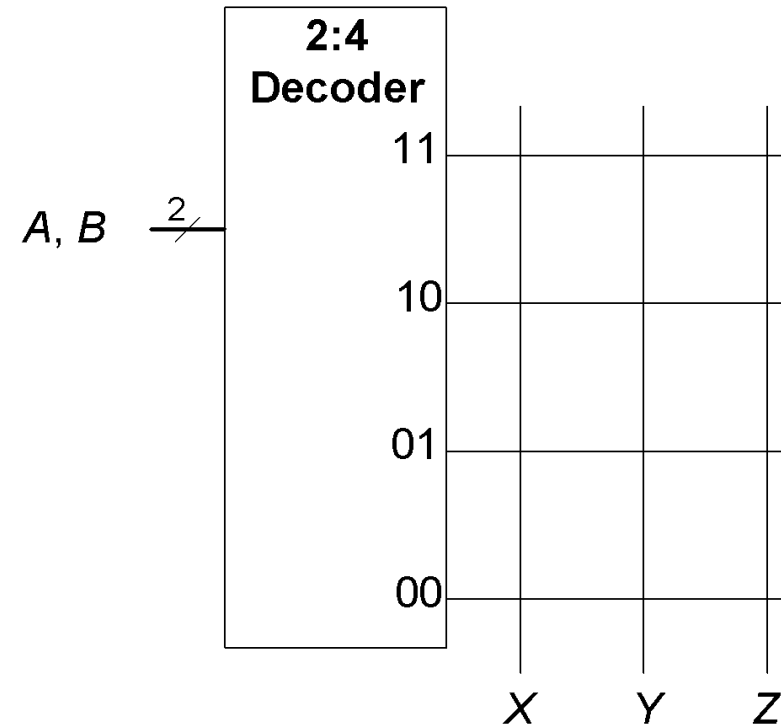
$$Data_1 = A_1 + \overline{A_0}$$

$$Data_0 = A_1 \overline{A_0}$$

Example: Logic with ROMs

Implement the following logic functions using a $2^2 \times 3$ -bit ROM:

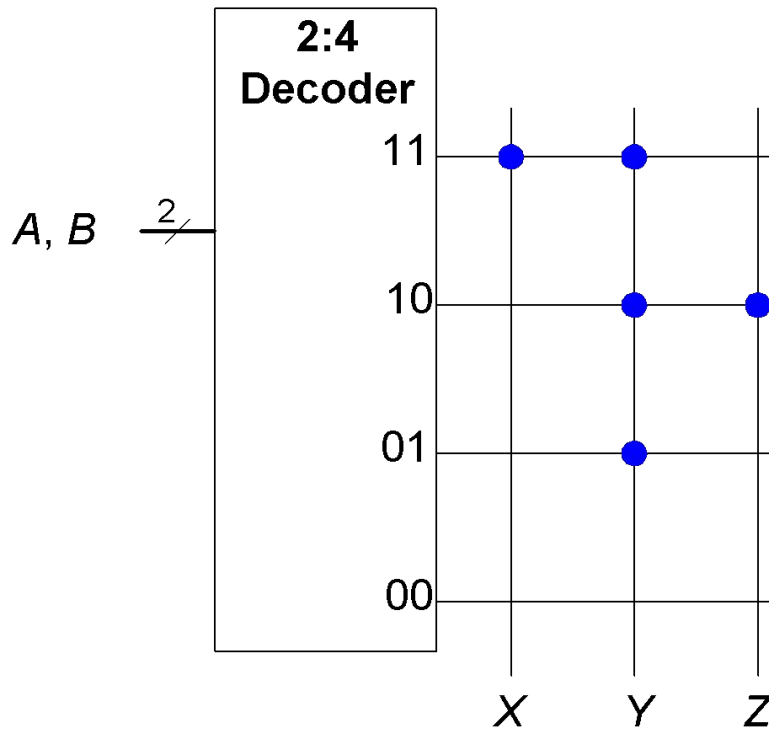
- $X = AB$
- $Y = A + B$
- $Z = A - B$



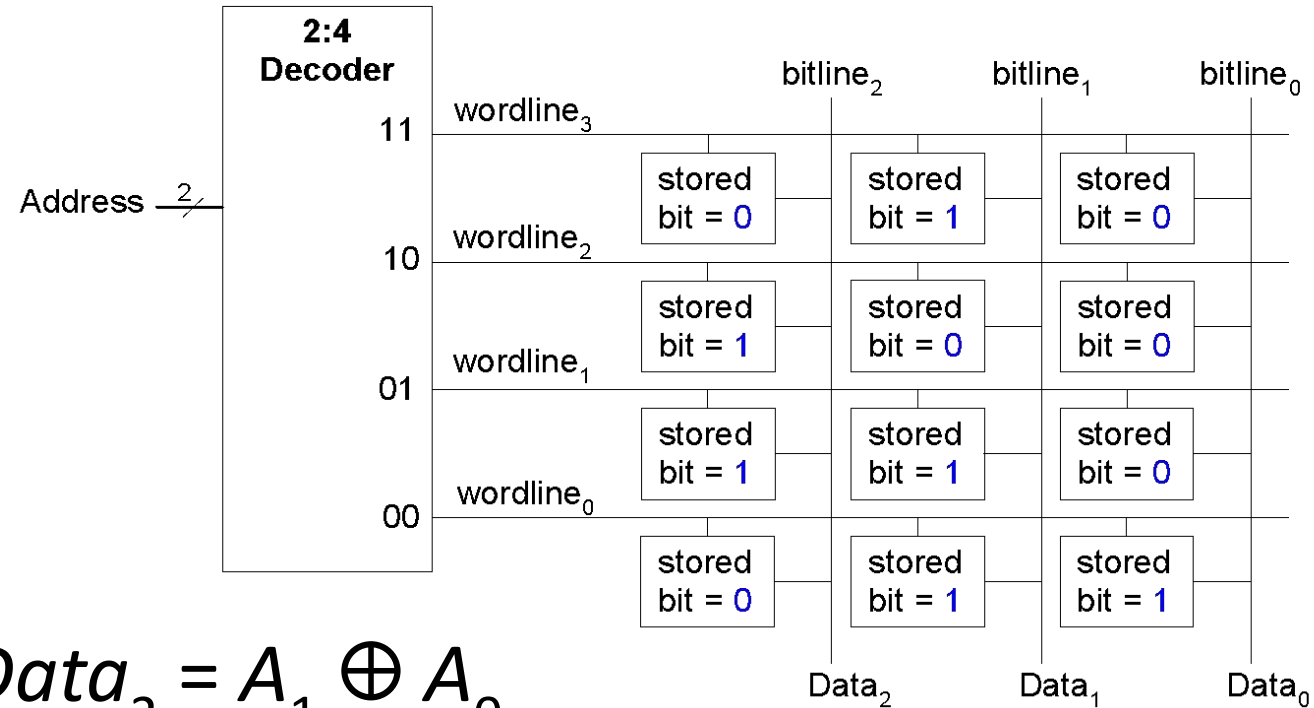
Example: Logic with ROMs

Implement the following logic functions using a $2^2 \times 3$ -bit ROM:

- $X = AB$
- $Y = A + B$
- $Z = A - B$



Logic with Any Memory Array



$$Data_2 = A_1 \oplus A_0$$

$$Data_1 = A_1 + \overline{A_0}$$

$$Data_0 = A_1 \overline{A_0}$$

Logic with Memory Arrays

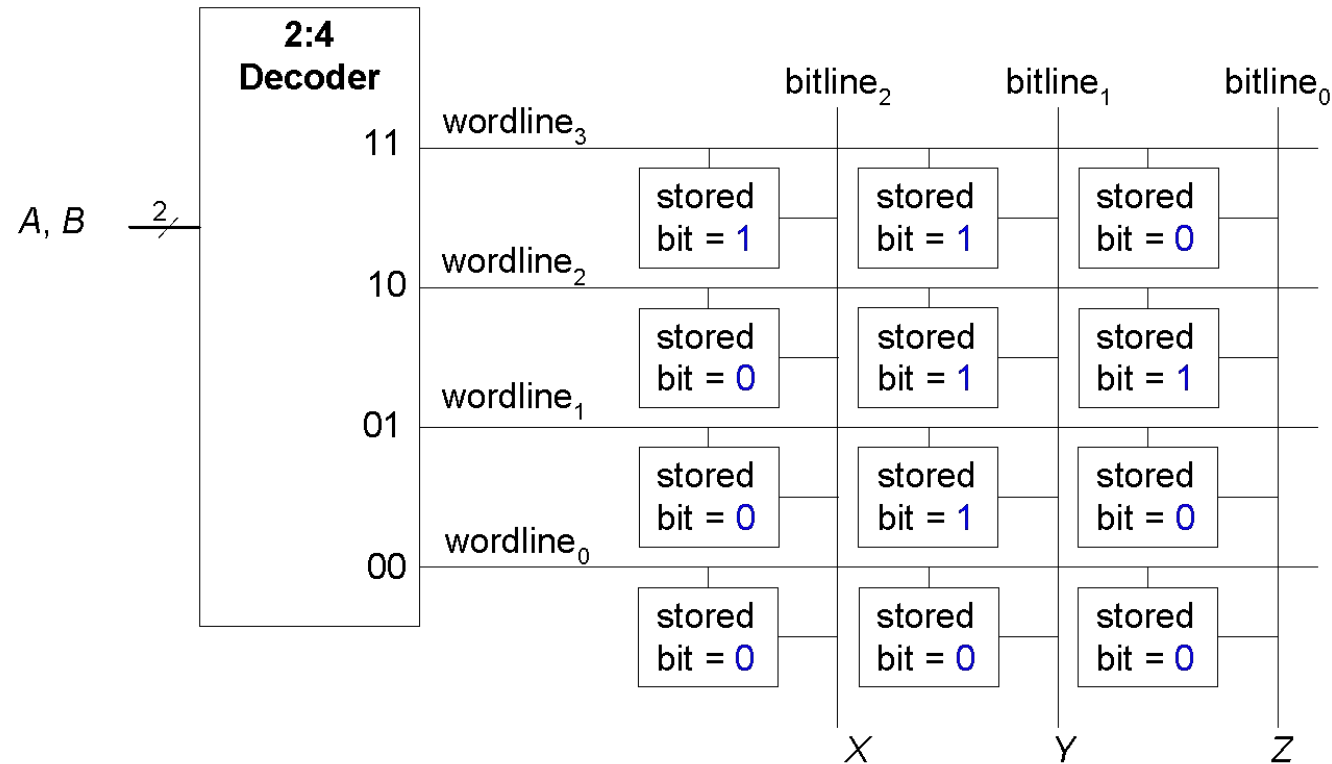
Implement the following logic functions using a $2^2 \times 3$ -bit memory array:

- $X = AB$
- $Y = A + B$
- $Z = A - B$

Logic with Memory Arrays

Implement the following logic functions using a $2^2 \times 3$ -bit memory array:

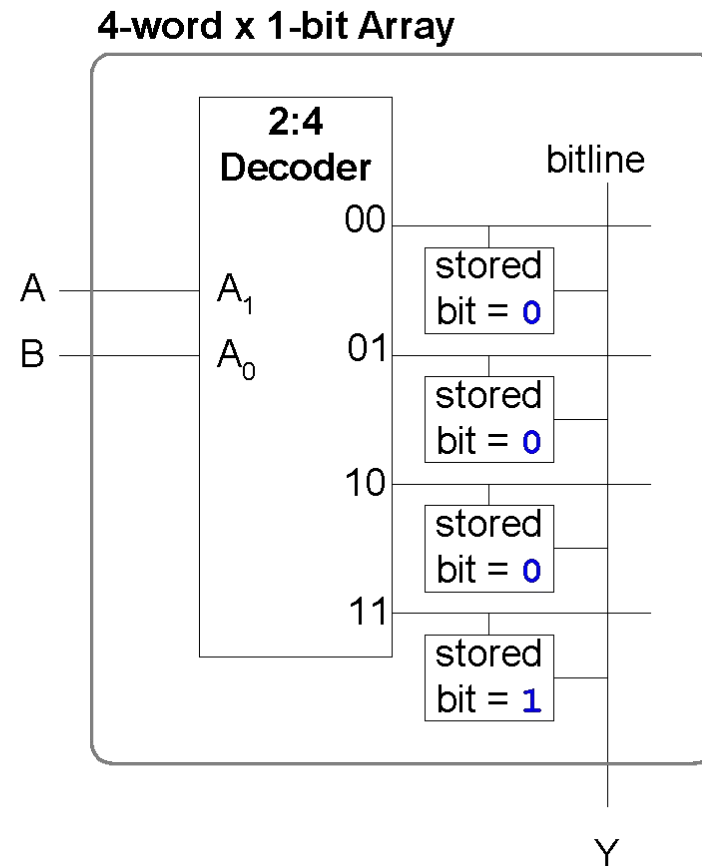
- $X = AB$
- $Y = A + B$
- $Z = A - B$



Logic with Memory Arrays

Called *lookup tables* (LUTs): look up output at each input combination (address)

Truth Table		
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

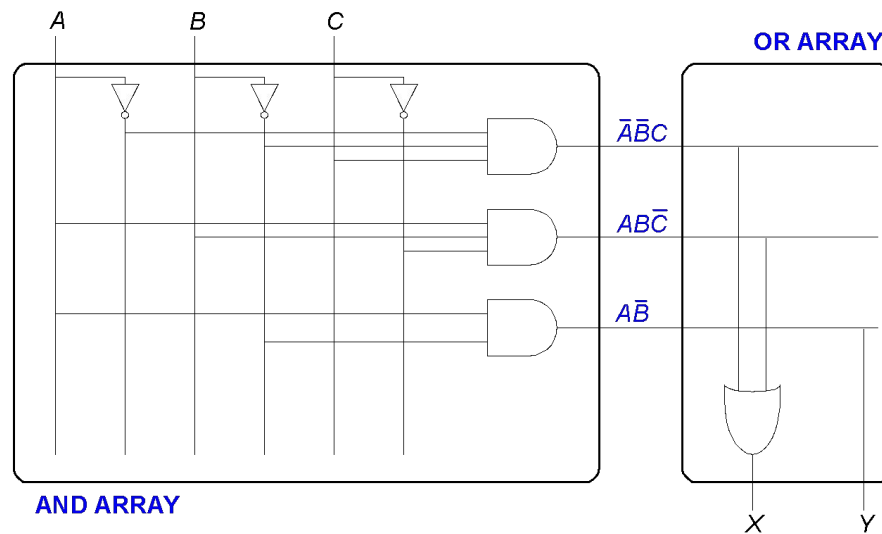
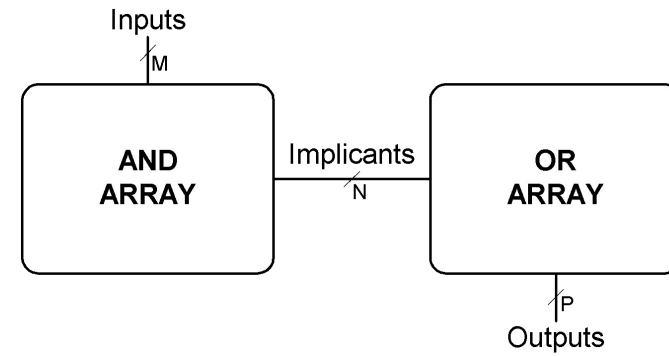


Logic Arrays

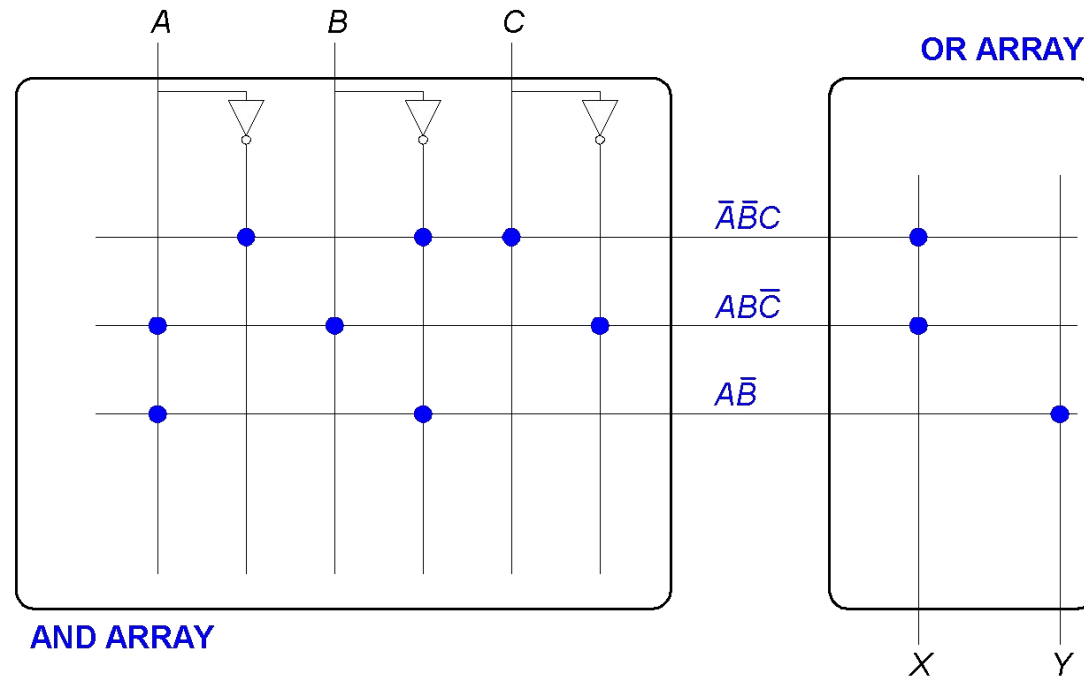
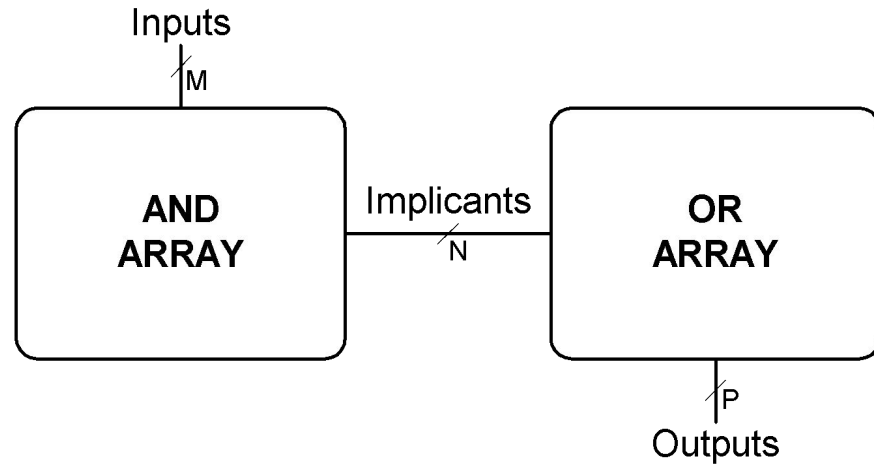
- **PLAs** (Programmable logic arrays)
 - AND array followed by OR array
 - Combinational logic only
 - Fixed internal connections
- **FPGAs** (Field programmable gate arrays)
 - Array of Logic Elements (LEs)
 - Combinational and sequential logic
 - Programmable internal connections

PLAs

- $X = A\bar{B}\bar{C} + AB\bar{C}$
- $Y = A\bar{B}$



PLAs: Dot Notation

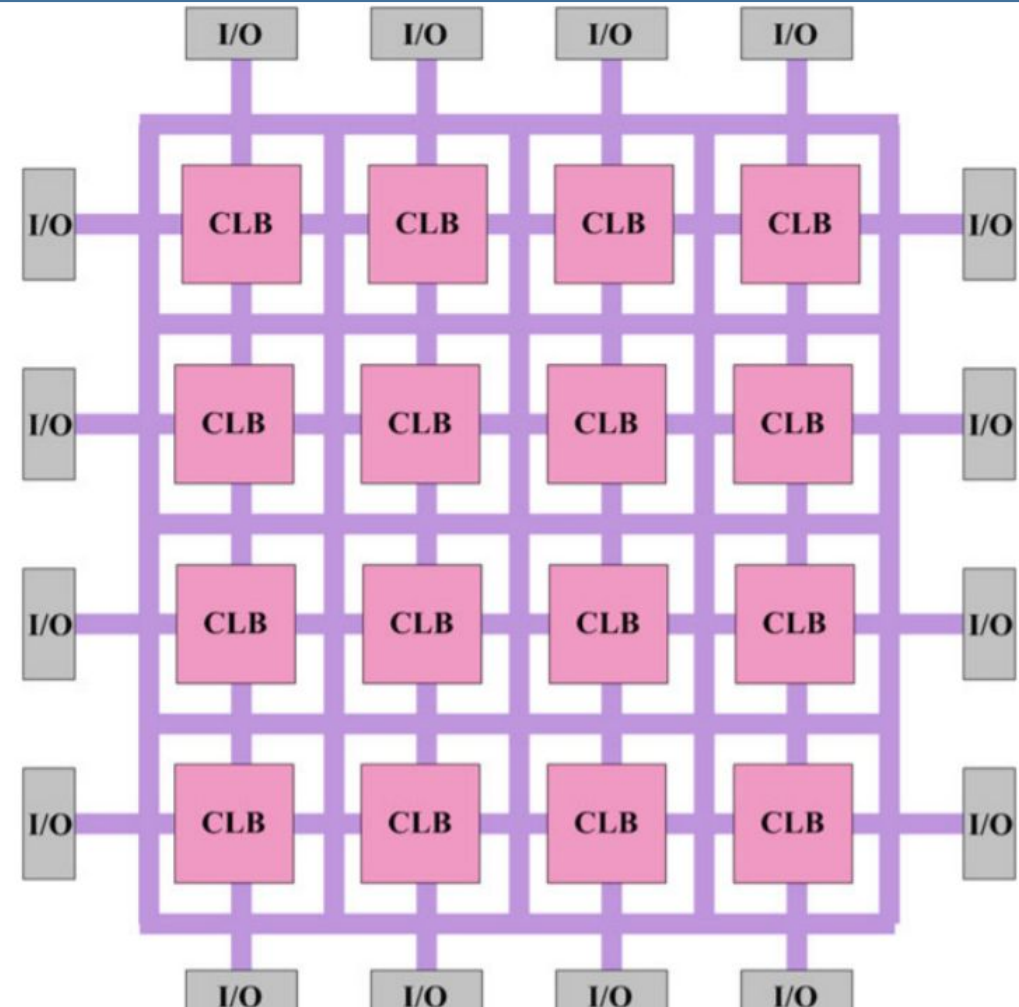


FPGA
(Field Programmable Gate Array)

What is a FPGA ?

FPGA Generic Architecture

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

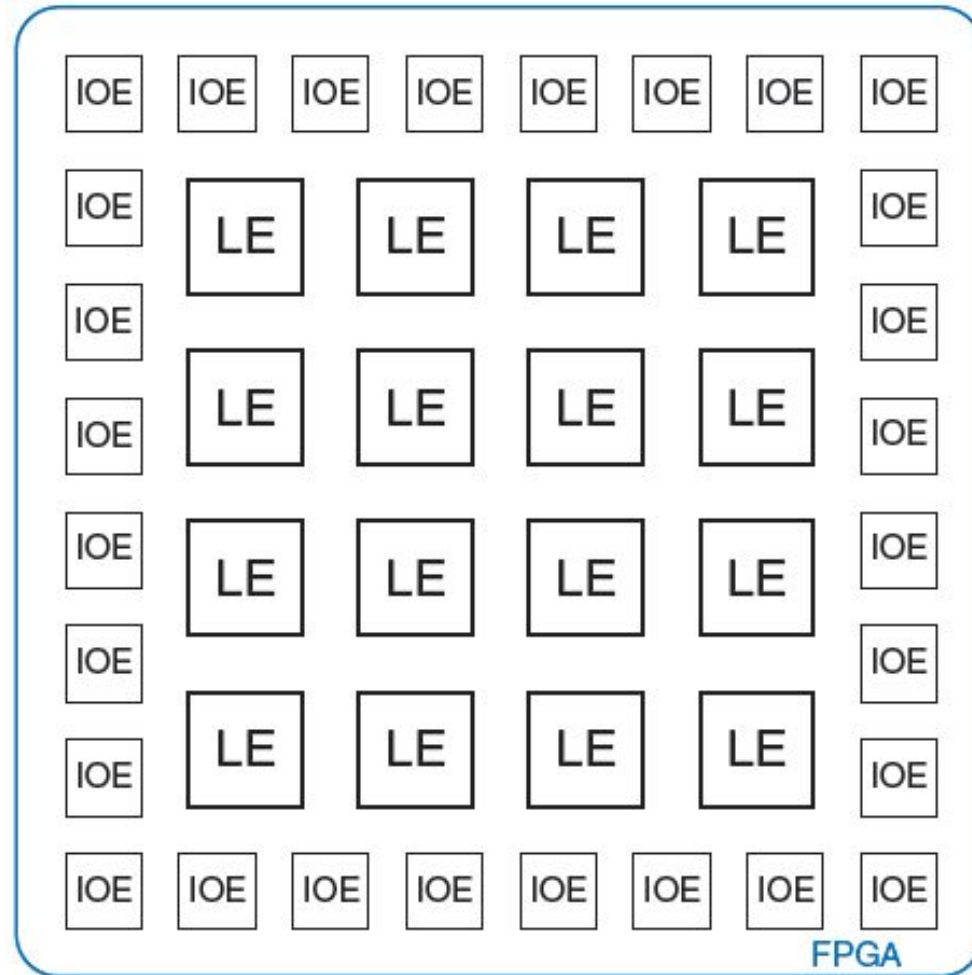


CLB : Configurable Logic Block (Xilinx FPGA)
LAB : Logic Array Block (Altera FPGA)

FPGA: Field Programmable Gate Array

- Composed of:
 - **LEs/CLB** (Logic elements or Configurable Logic Block): perform logic
 - **IOEs** (Input/output elements): interface with outside world
 - **Programmable interconnection:** connect LEs and IOEs
 - Some FPGAs include other building blocks such as multipliers and RAMs

General FPGA Layout



LE: Logic Element

- Composed of:
 - **LUTs** (lookup tables): perform combinational logic
 - **Flip-flops**: perform sequential logic
 - **Multiplexers**: connect LUTs and flip-flops

Example of a CLB

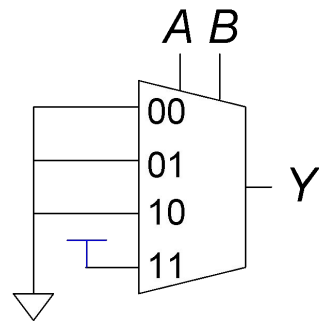
- The Spartan CLB has:
 - 1 four-input LUT
 - 1 registered output
 - 1 combinational output

Logic Design Using Multiplexers

Using the mux as a lookup table

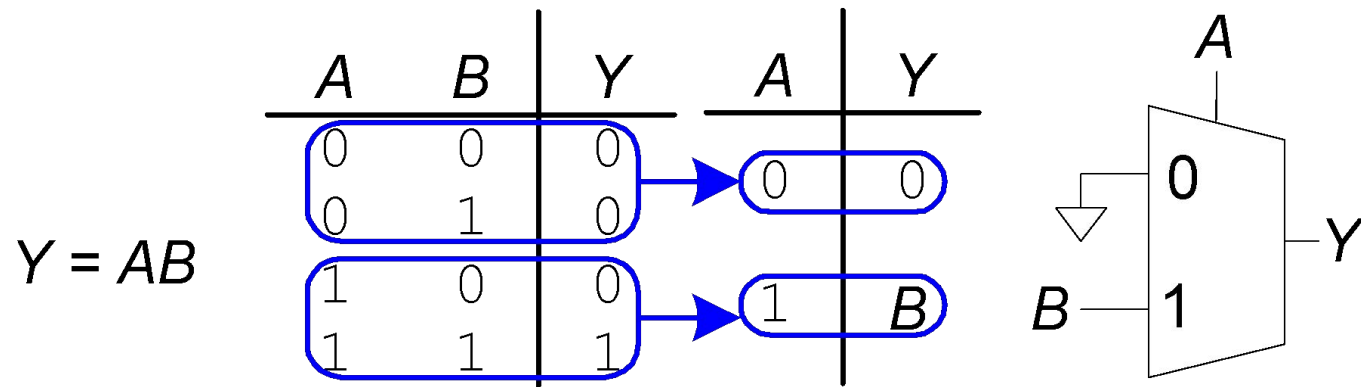
<i>A</i>	<i>B</i>	<i>Y</i>
0	0	0
0	1	0
1	0	0
1	1	1

$$Y = AB$$

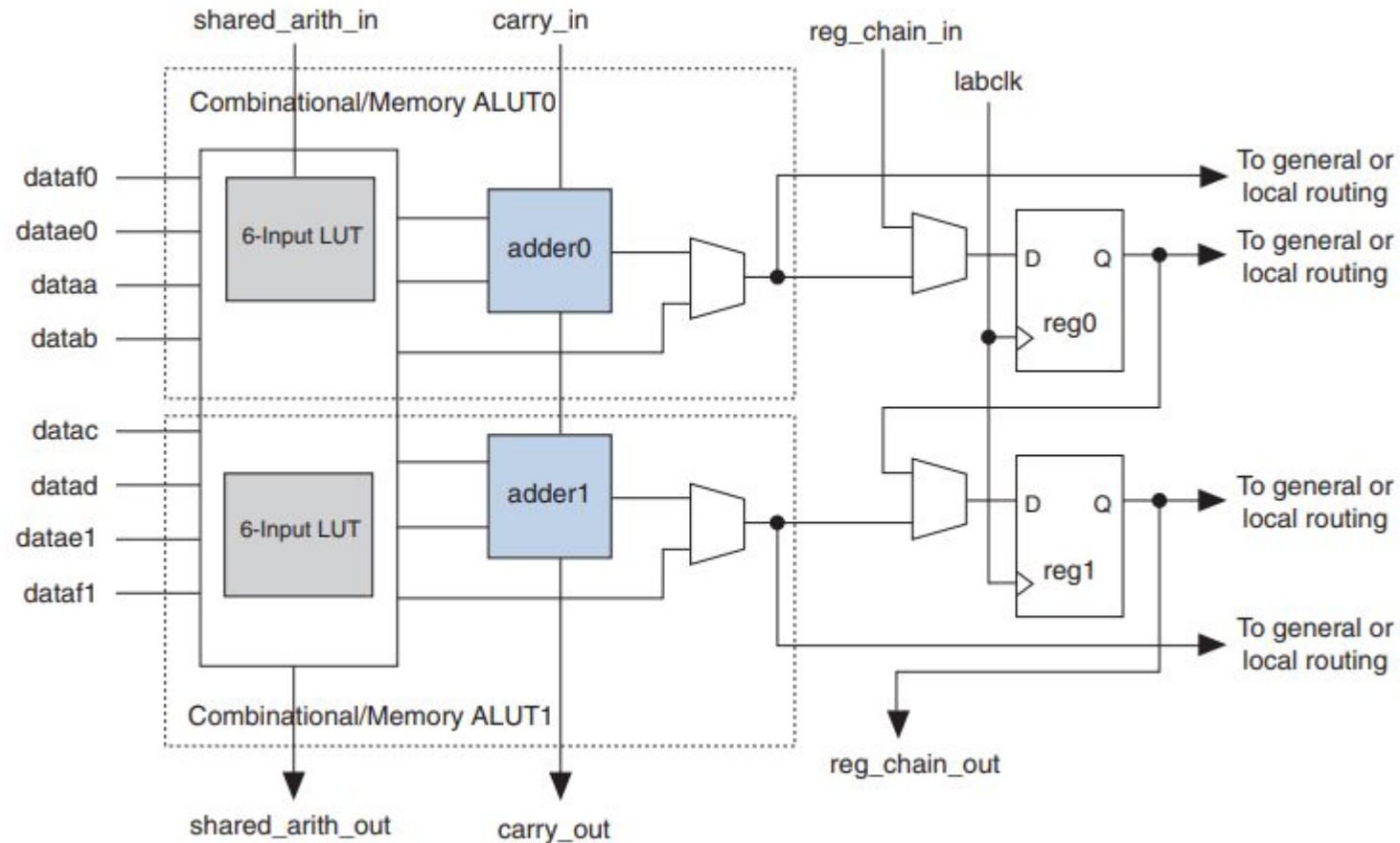


Logic using multiplexers

Reducing the size of the mux



Arria II Device Family

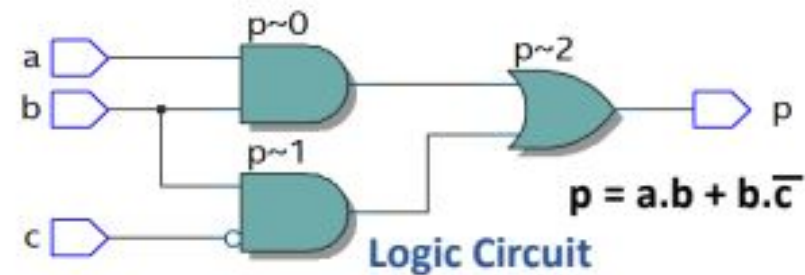


Look Up Tables (LUTs)

- A computation can be represented as a Boolean equation
- A Boolean equation can be represented as a truth table
- Truth tables are the computational heart of the FPGAs
- A LUT is a hardware element that can easily implement a truth table

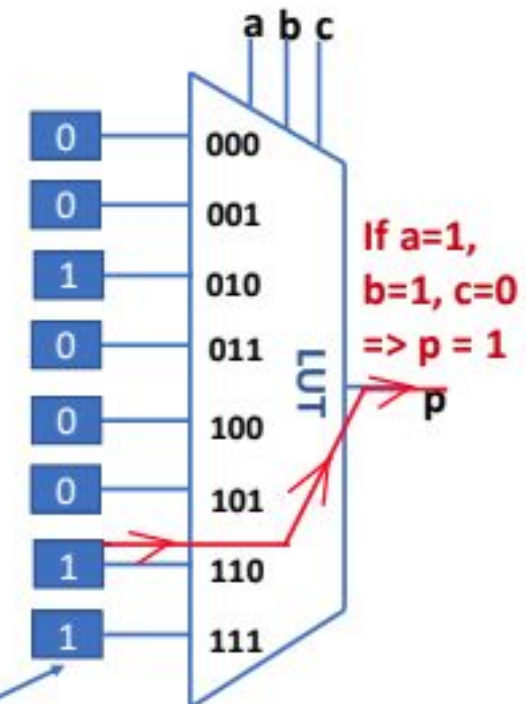
Example of Look Up Table:

Example of a 3-to-1
LUT implementing a
boolean expression
 $p = a.b + b.c$



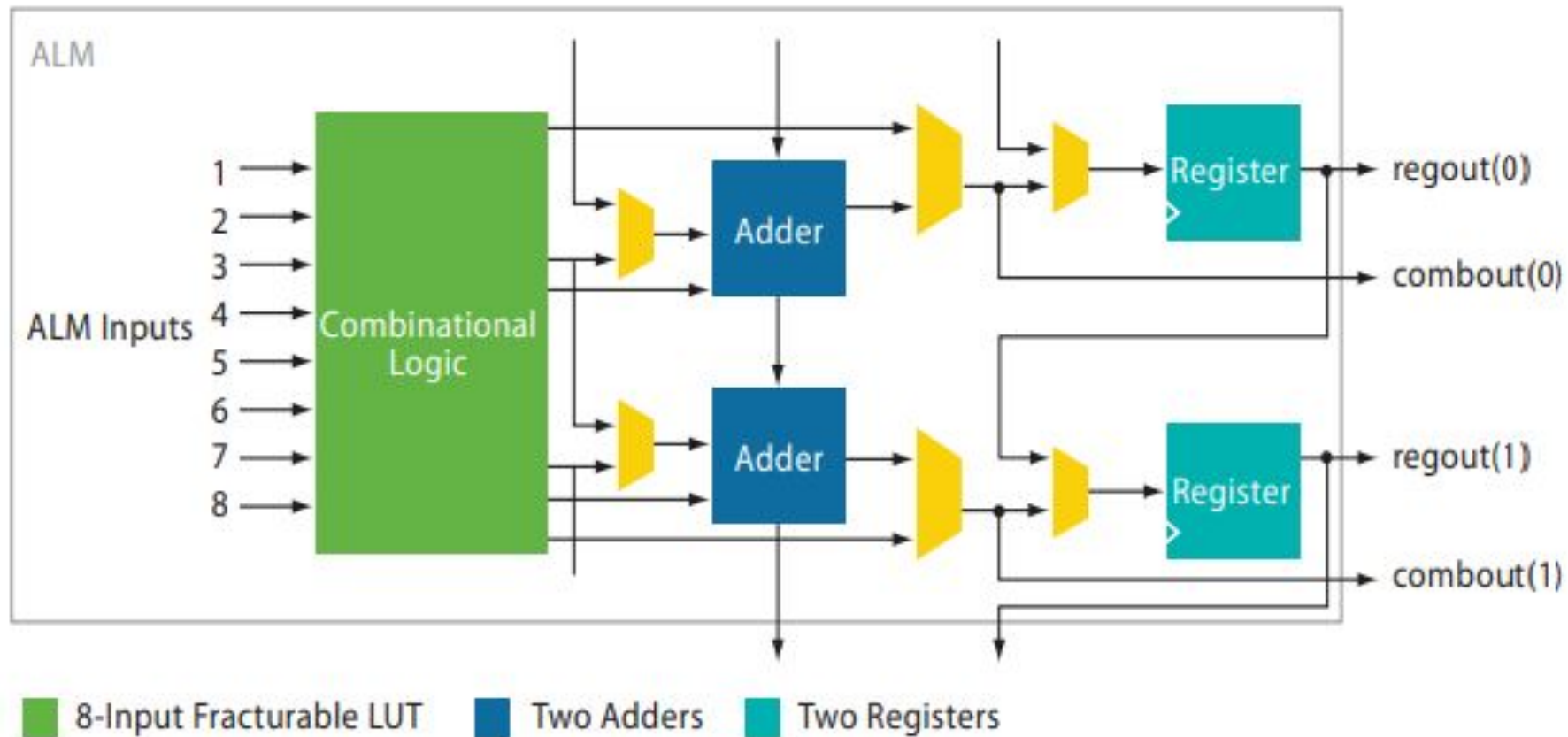
Truth Table for $p = a.b + b.c$

a	b	c	$p = a.b + b.c$
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

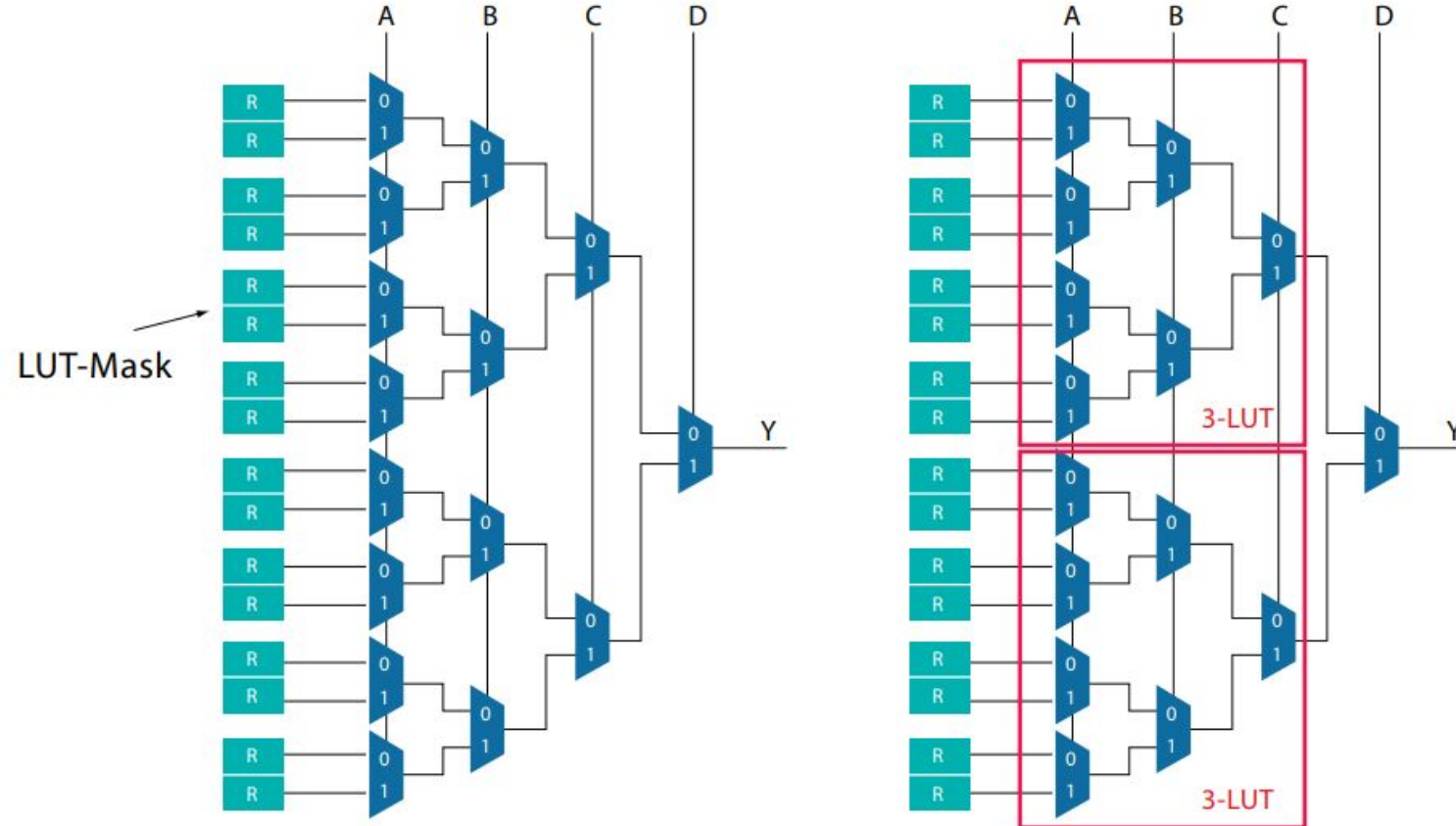


SRAM, programmed with value 0xC4
(1100_0100) to implement $p = a.b + b.c$

Altera FPGA ALM



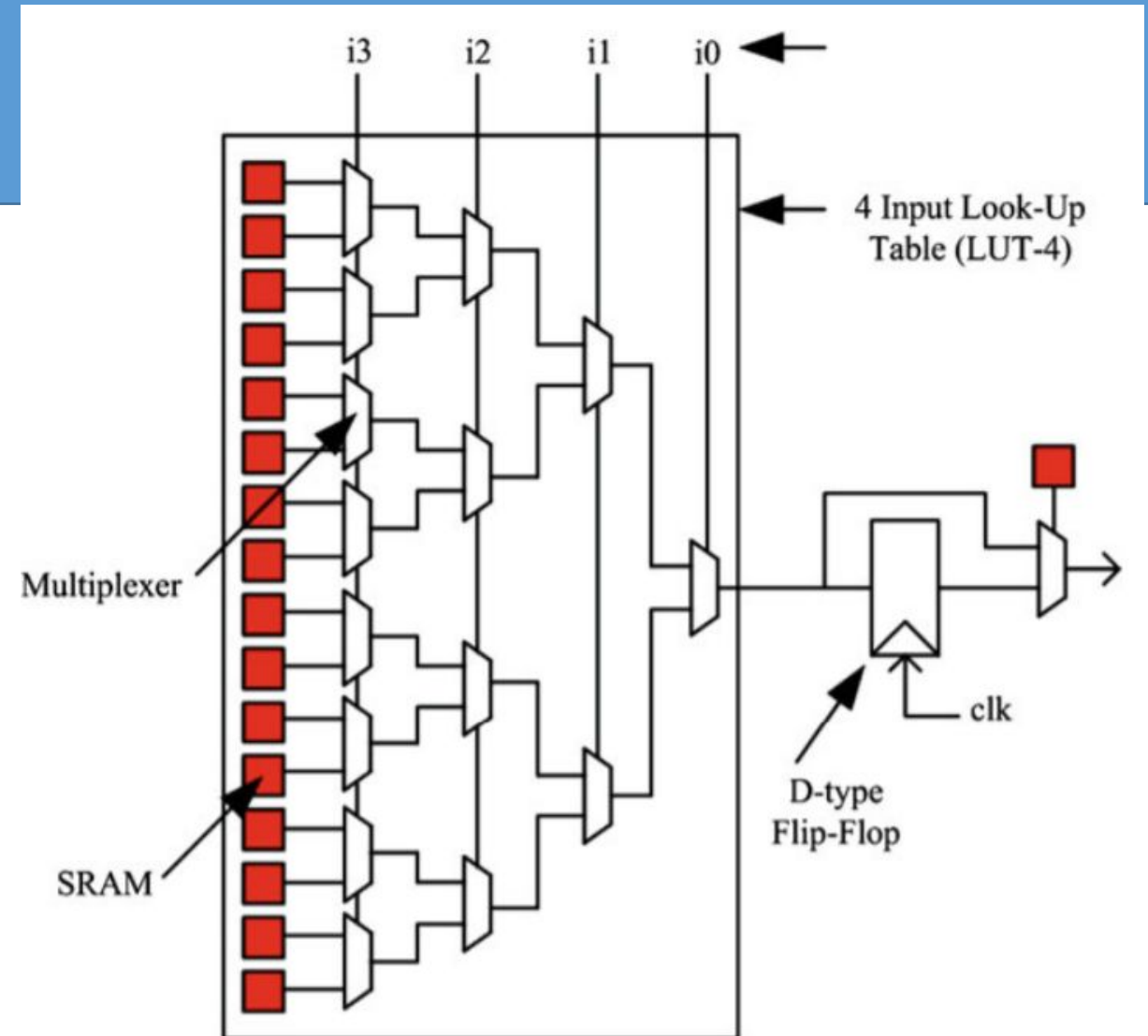
Altera FPGA LUT divided into two 3 to 1 LUT



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

What's inside FPGA ?

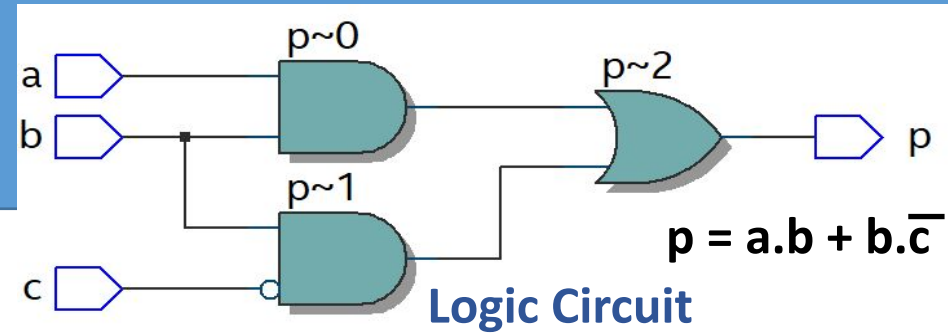
- A simple BLE comprising of a 4 input LUT (LUT-4) and a D-type Flip-Flop.
 - The LUT-4 uses 16 SRAM bits to implement any 4 inputs boolean function.
 - The output of LUT-4 is connected to an optional Flip-Flop. A multiplexor selects the BLE output to be either the output of a Flip-Flop or the LUT-4
- Modern FPGAs contain typically 4 to 10 BLEs in a single cluster.
- Many modern FPGAs contain a heterogeneous mixture of blocks, some of which can only be used for specific purposes.
 - These specific purpose blocks, also referred here as hard blocks, include memory, multipliers, adders and DSP blocks etc.
 - Hard blocks are very efficient at implementing specific functions as they are designed optimally to perform these functions, yet they end up wasting huge amount of logic and routing resources if unused



4 input LUT based BLE (Basic Logic Element)

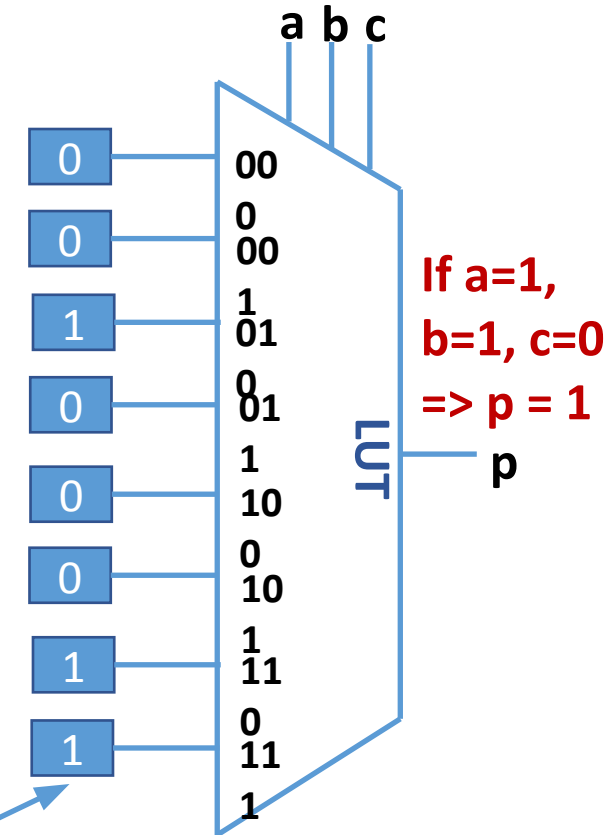
LUT (Look Up Table)

- 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)
- 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



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

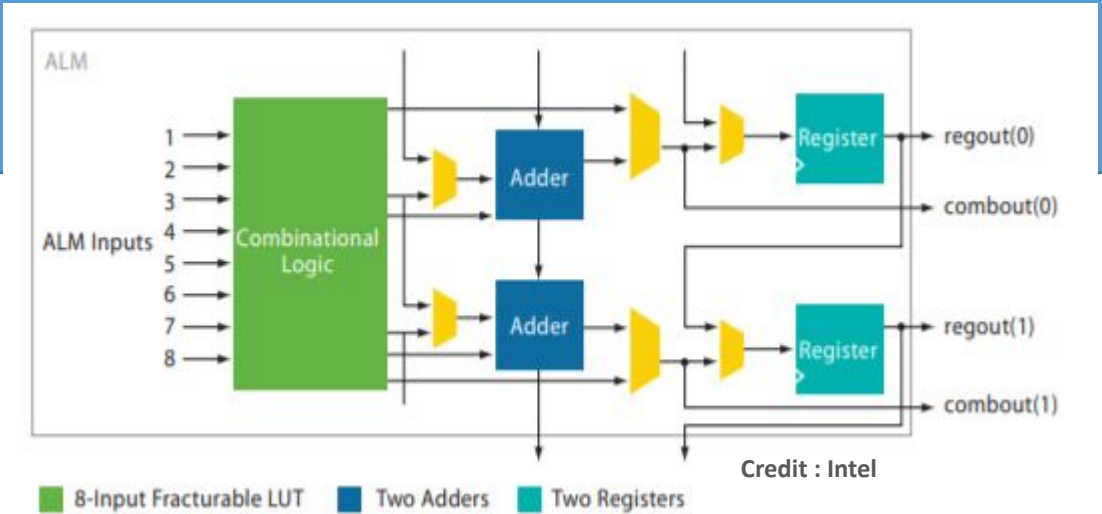


SRAM, programmed with value 0xC4 (1100_0100) to implement

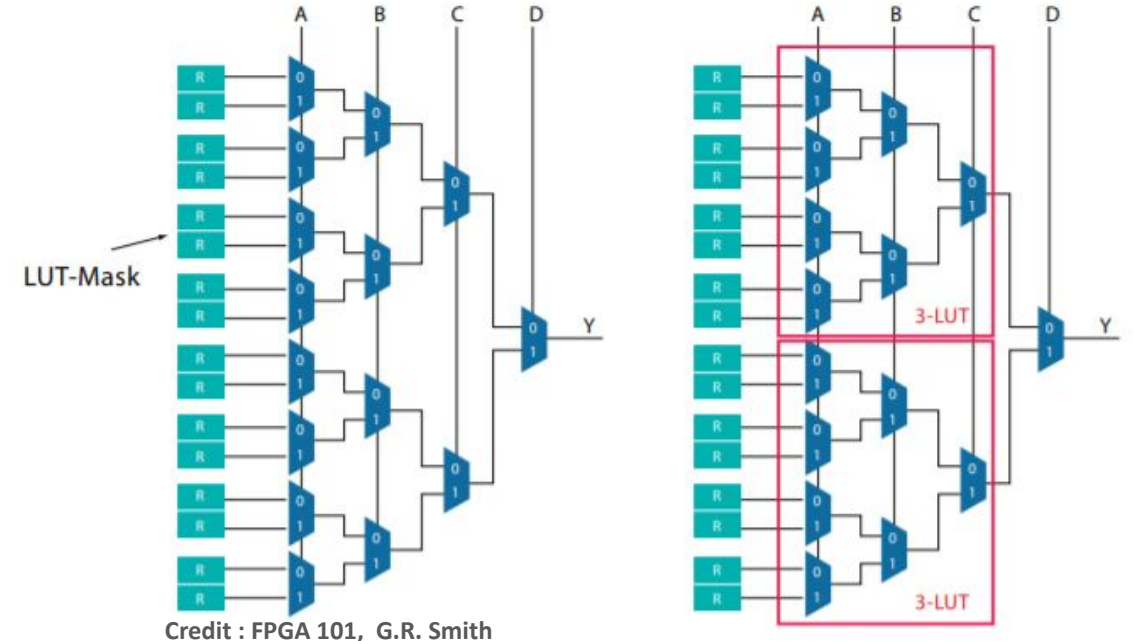
ALUT?

- Altera introduced the 8-input fracturable look-up table (LUT) with the Stratix® II family.
- Its adaptive logic module (ALM) has 8 inputs, which can implement a full 6-input LUT (6-LUT) or select 7-input functions.
- The ALM can also be efficiently partitioned into independent smaller LUTs, providing the performance advantage of larger LUTs and the area efficiency of smaller LUTs.
- ALM also consists of higher efficient adder logic and two registers
- Diagram shows, 4 input ALUT is broken into two halves to implement 3 to 1 LUT

8 input Adaptive Logic Module (ALM)



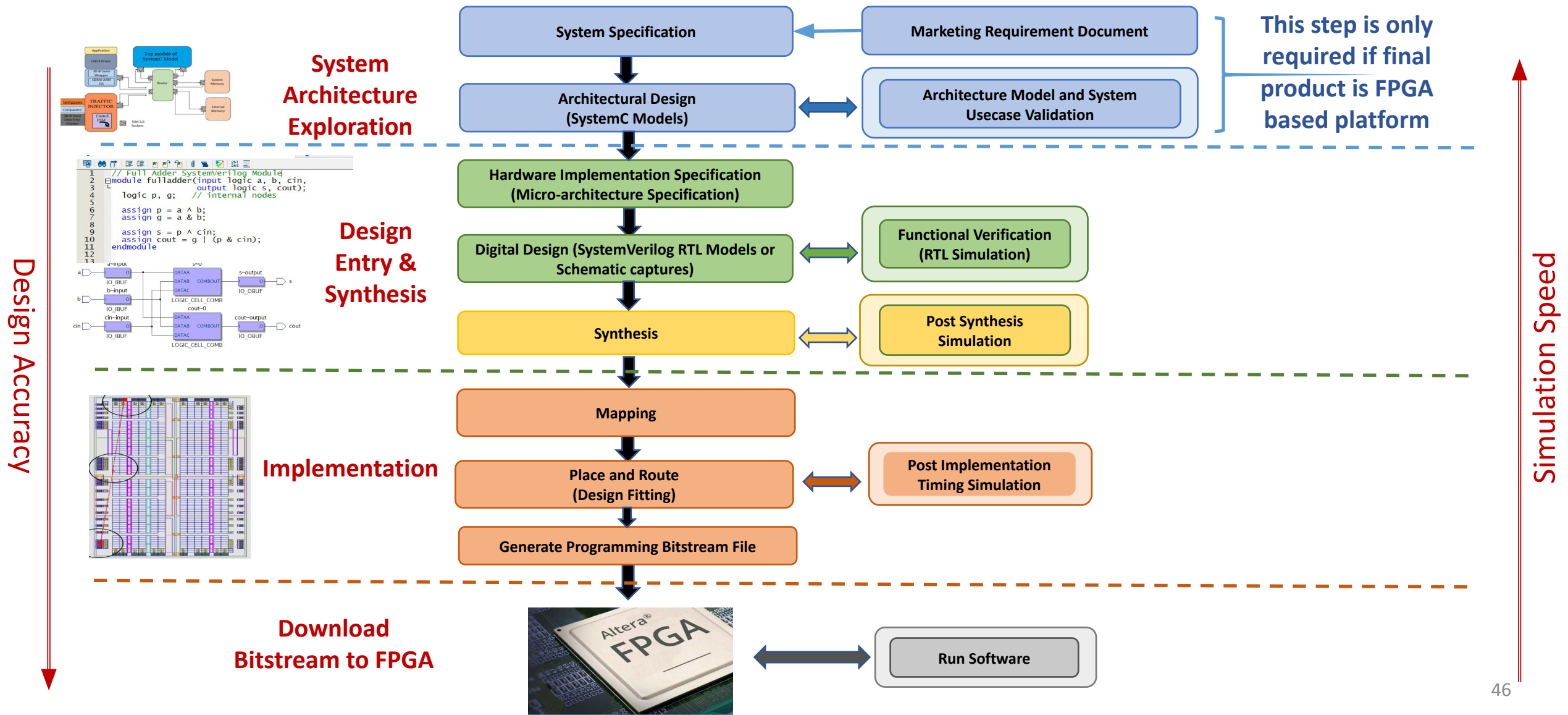
4 input Adaptive LUT divided into two 3 to 1 LUT



Credit : FPGA 101, G.R. Smith

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

FPGA Digital Design Flow



FPGA Synthesis

- **FPGA Synthesis is a three 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

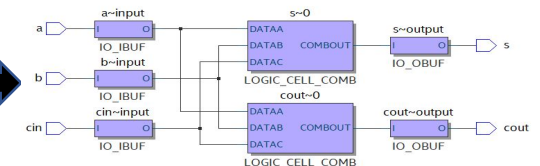
- Reduce logic
- Eliminate redundant logic
- Make design smaller and faster (best Fmax)

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's 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 :**

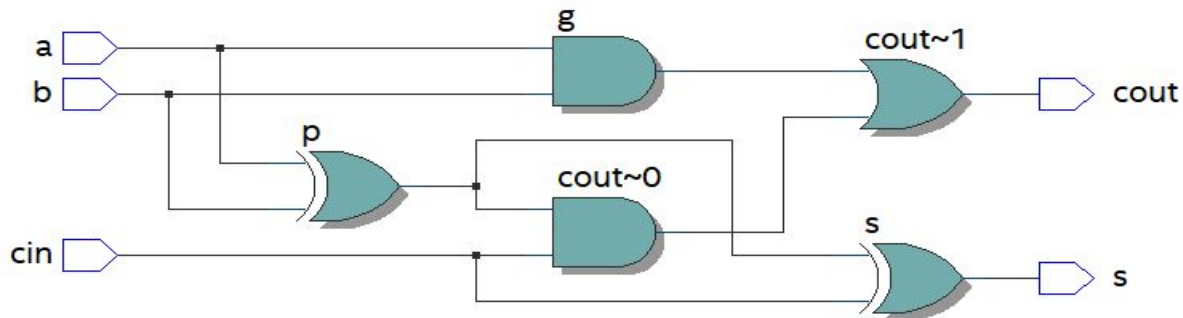
- RTL netlist viewer
- Post-fitting netlist viewer
- Post-mapping netlist viewer

SystemVerilog .sv source files

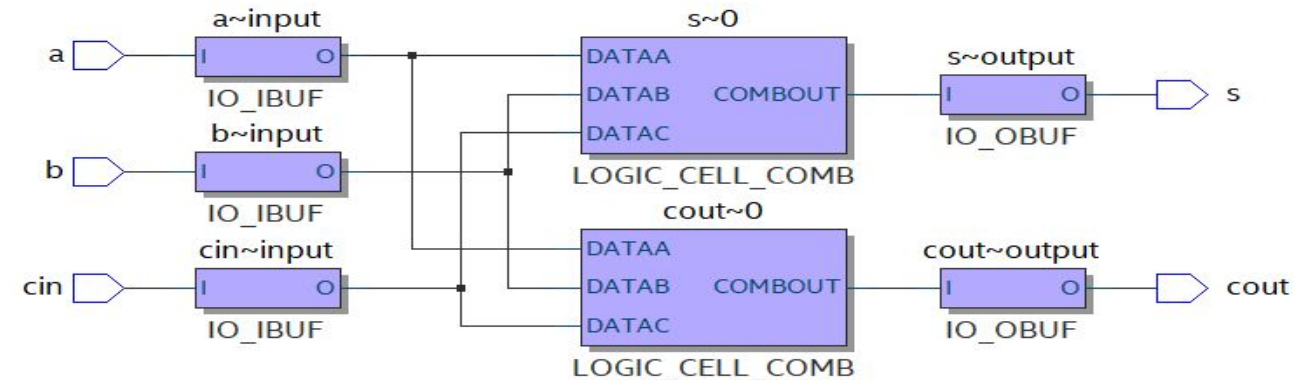
```
1 // Full Adder SystemVerilog Module
2 module fulladder(input logic a, b, cin,
3                 output logic s, cout);
4     logic p, g; // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11 endmodule
12
13
```



RTL Netlist Viewer



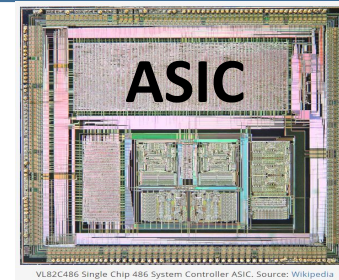
Post-Fitting Netlist Viewer



FPGA vs ASIC Comparison



Versus



- **Advantages :**
 - Low Time-to-market
 - High flexibility
 - High reusability
 - High NRE
 - **Disadvantages :**
 - Low Performance
 - High power consumption
 - High Area
 - Low volume production due to lower cost per unit
 - 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, Machine learning, etc
- **Advantages :**
 - High Performance
 - Low power consumption
 - Low Area
 - High volume production due to lower cost per unit
 - Can implement Analog blocks
 - **Disadvantages :**
 - 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)

Learning Resources on FPGA Architecture

- Modern FPGA Architecture (Intel Altera)
 - <https://youtu.be/EVy4KEj9kZg?si=A6RLPiGacFoB54gA>
- Altera FPGA Architecture :
 - <https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf>