


CSCE 611

Introduction to Behavioral Logic Design



Instructor: Jason D. Bakos



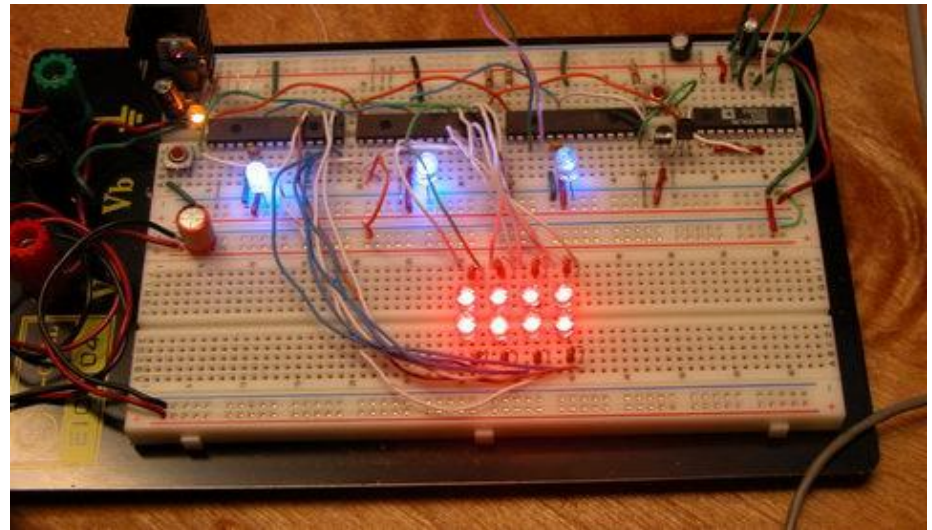
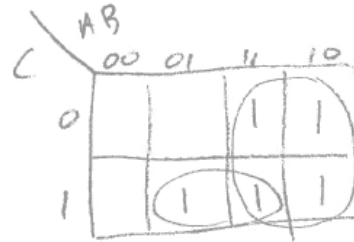
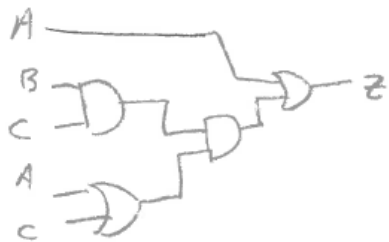
Introduction to CSCE 611

- Teaching assistant:
 - Scottie Scott (fscott@email.sc.edu)
 - 3D15, left side
- 3D22 combination:
 - 5-1-2-4-3

Digital Design vs. Advanced Digital Design

$$Z = A + (B \cdot c) \cdot (A + c)$$

$$\begin{aligned} Z &= A + (ABC + CBc) \\ &= A + (ABC + BC) \\ &= A + BC(A + 1) \\ &= A + BC \end{aligned}$$



Design Complexity

- Modern CPUs have over a billion transistors
 - At 4 transistors/gate, this is > 250 million gates
- Using CSCE 211 technology, this would require:
 - 62 million 7400-series chips
 - 8 million breadboards
 - 1010 miles long, laid end-to-end

Introduction to CSCE 611

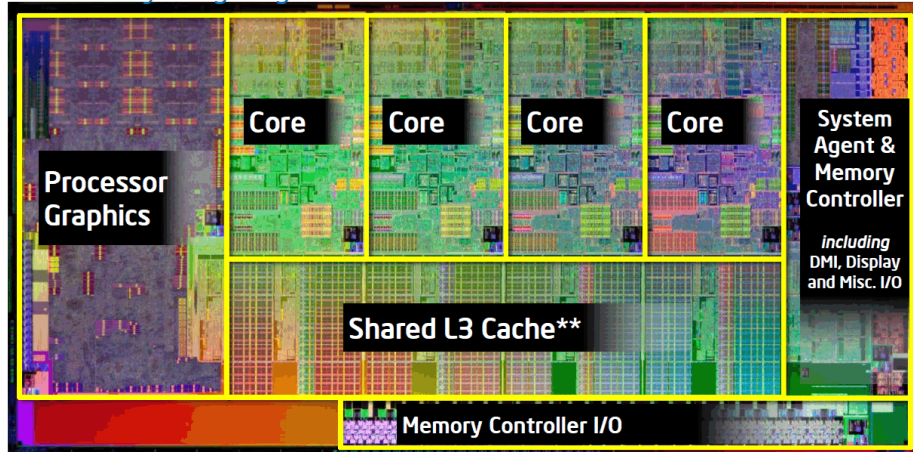
- Problems:
 1. How can we design large-scale digital systems?
 2. How can the designs be re-used and extended?
 3. How can we test and debug designs?
- Answer: Conceptualize the design process into a text-based programming language
 - Hardware Description Language
 - Use for **modeling** and **synthesis**



Why Learn HDL?

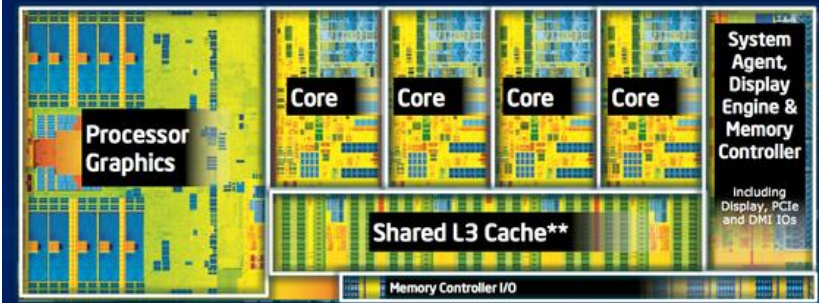
2nd Generation Intel® Core™ Processor Die Map

32nm Sandy Bridge High-k + Metal Gate Transistors



4th Generation Intel® Core™ Processor Die Map

22nm Tri-Gate 3-D Transistors



Quad core die shown above

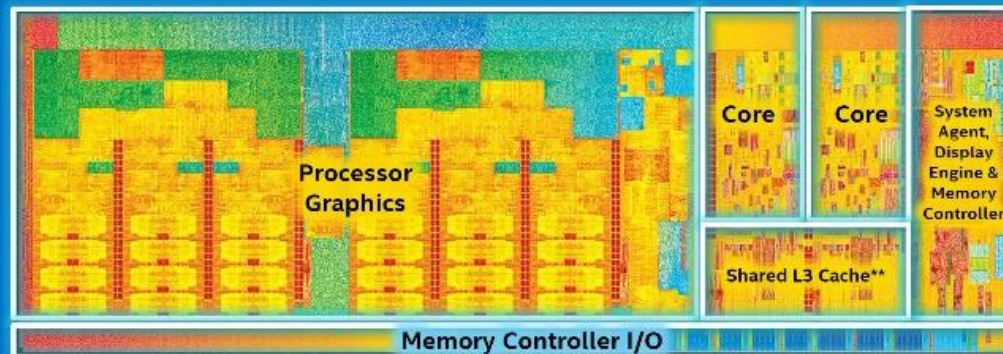
Transistor count: 1.4 Billion

Die size: 177mm²

** Cache is shared across all 4 cores and processor graphics

5th Gen Intel® Core™ Processor Die Map

Intel® HD Graphics 6000 or Intel® Iris™ Graphics 6100



Dual Core Die Shown Above

Transistor Count: 1.9 Billion

Die Size: 133 mm²

4th Gen Core Processor (U series): 1.3B
** Cache is shared across both cores and processor graphics

4th Gen Core Processor (U series): 181 mm²

Summary of Course Activities

- Objective: combine material from 211 and 212 to design a CPU
- All lab work performed in 2-person groups
 - Both members receive same grade for projects
- Short quizzes given in class
- Final exam: 2 hour design practicum
- Series of labs designed to incrementally design a MIPS CPU using Verilog hardware description language
 - Each lab will require:
 1. Hardware design
 2. Corresponding test bench or MIPS program
 3. Implementation on FPGA board



Labs

- List of lab assignments (subject to change):
 1. 7-segment Decoder and PWM
 2. MIPS ALU testbench
 3. MIPS register file and interface to switches and LEDs
 4. MIPS fetch stage
 5. 3-stage MIPS CPU with arithmetic and move instructions
 6. 3-stage MIPS CPU with load/store instructions
 7. 3-stage MIPS CPU with branch/jump instructions
 8. Peripheral interface (VGA/keys)
 9. 2-issue VLIW CPU (graduate credit)



Implementing Large-Scale Digital Designs

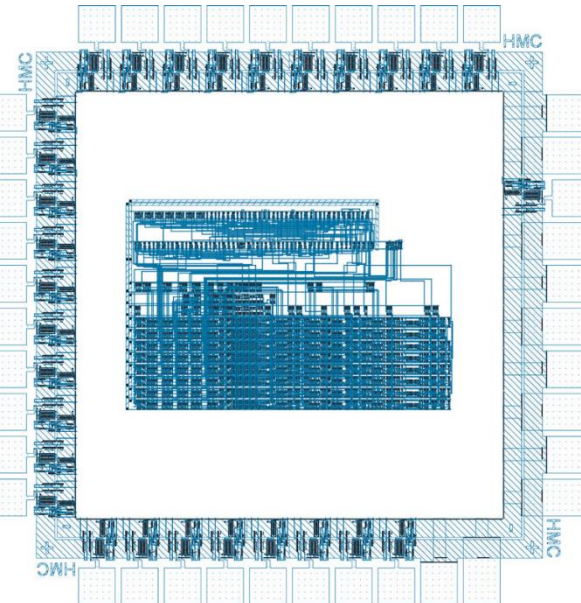
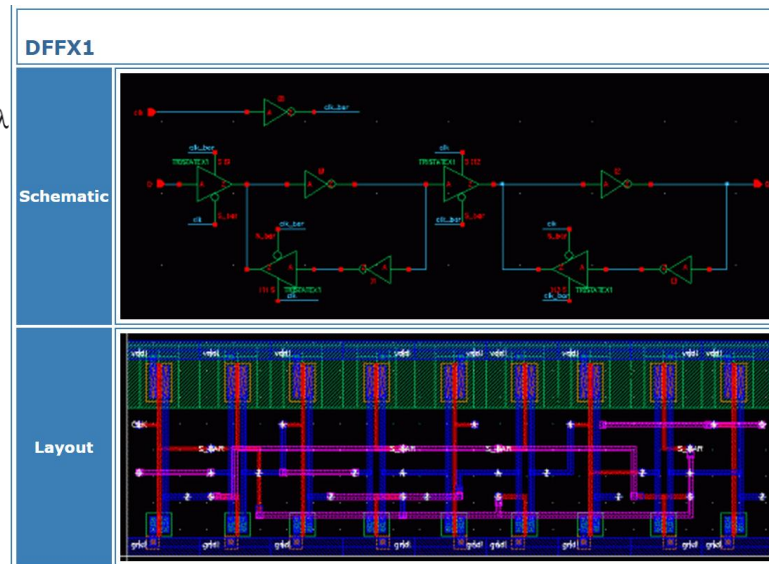
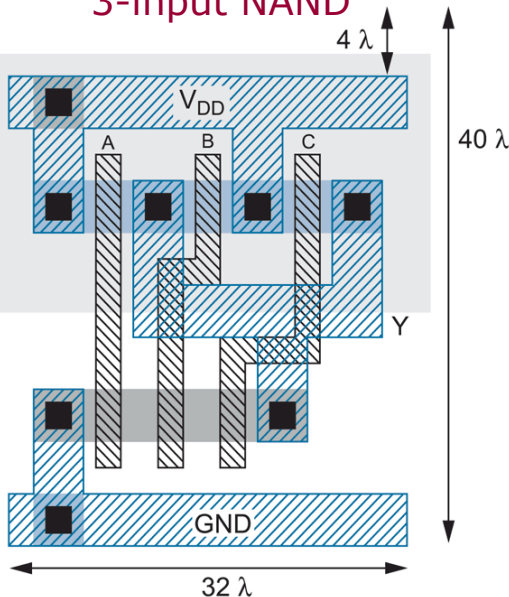
- Three options:
 - Option 1: Use TTL chips and breadboards



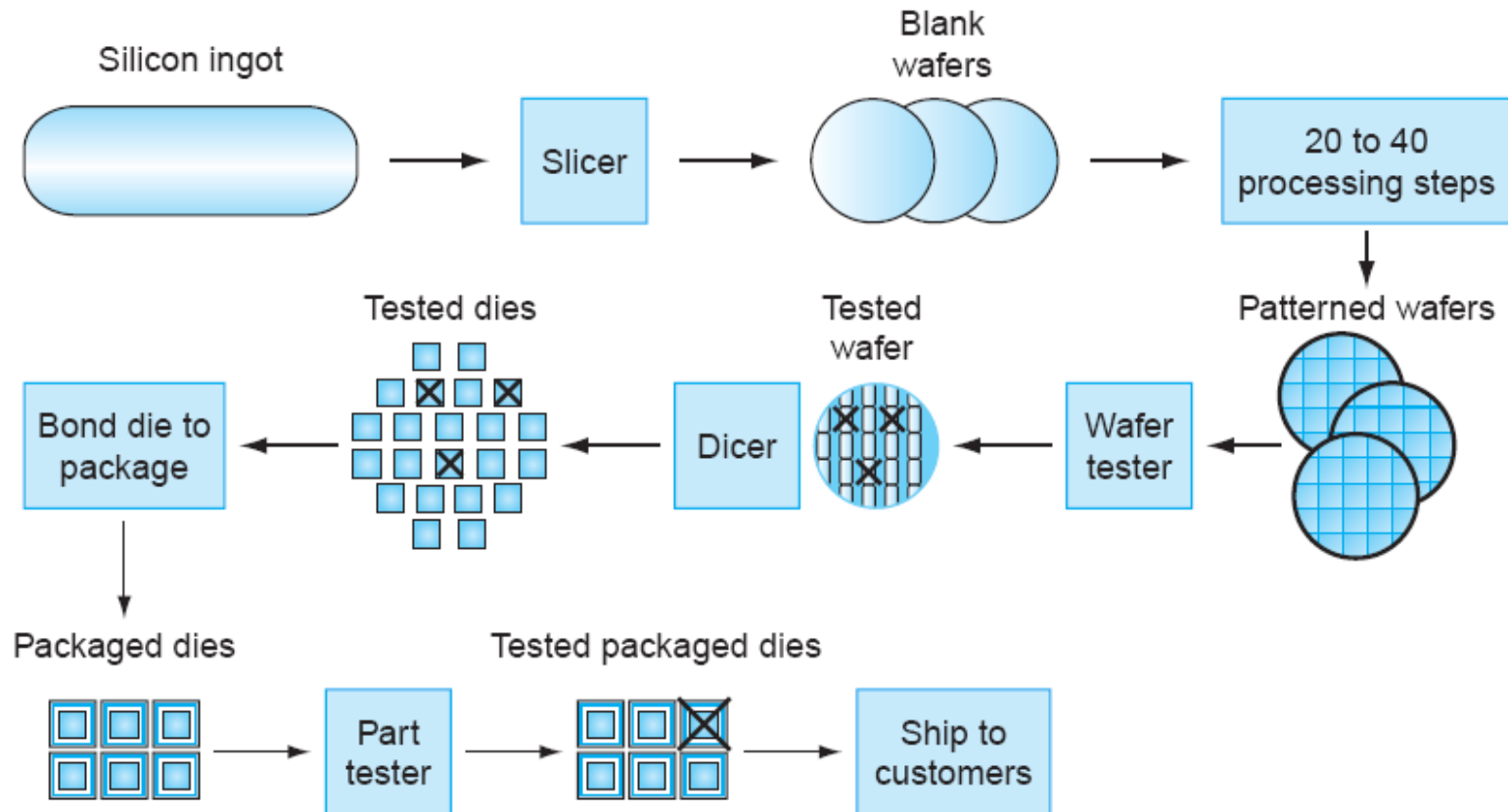
Implementing Large-Scale Digital Designs

- Option 2: Design and fabricate integrated circuit
 - Design-for-manufacturing: requires IC mask design

3-input NAND

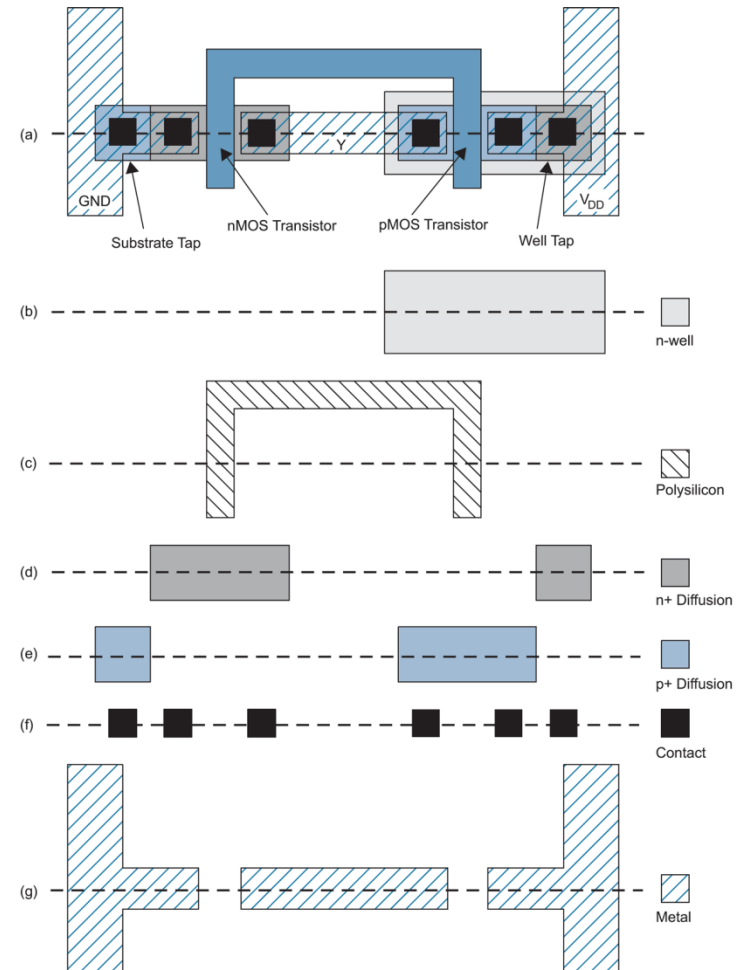


IC Fabrication

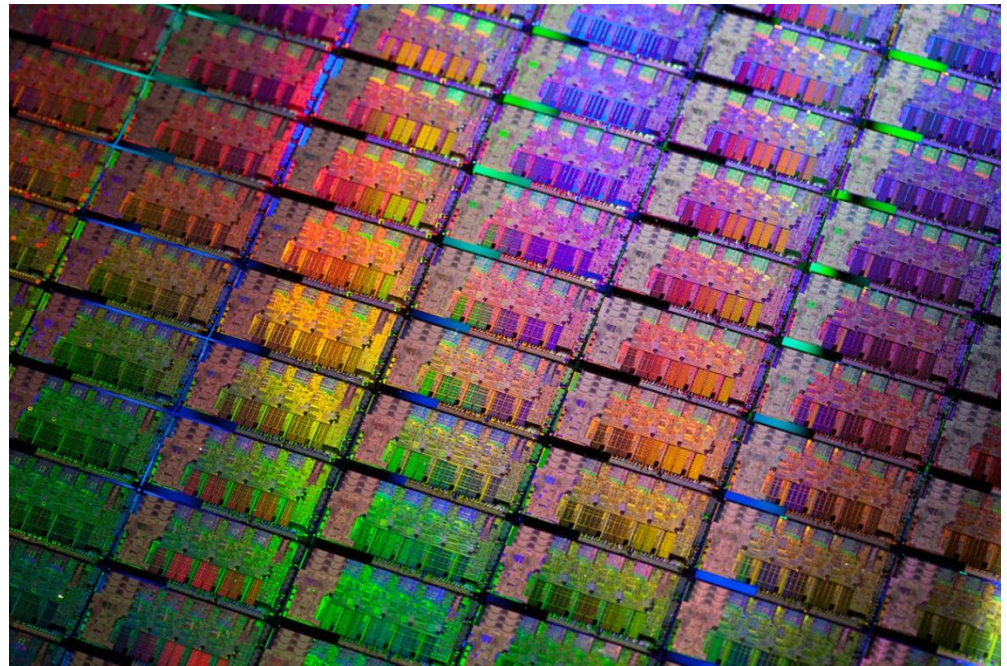
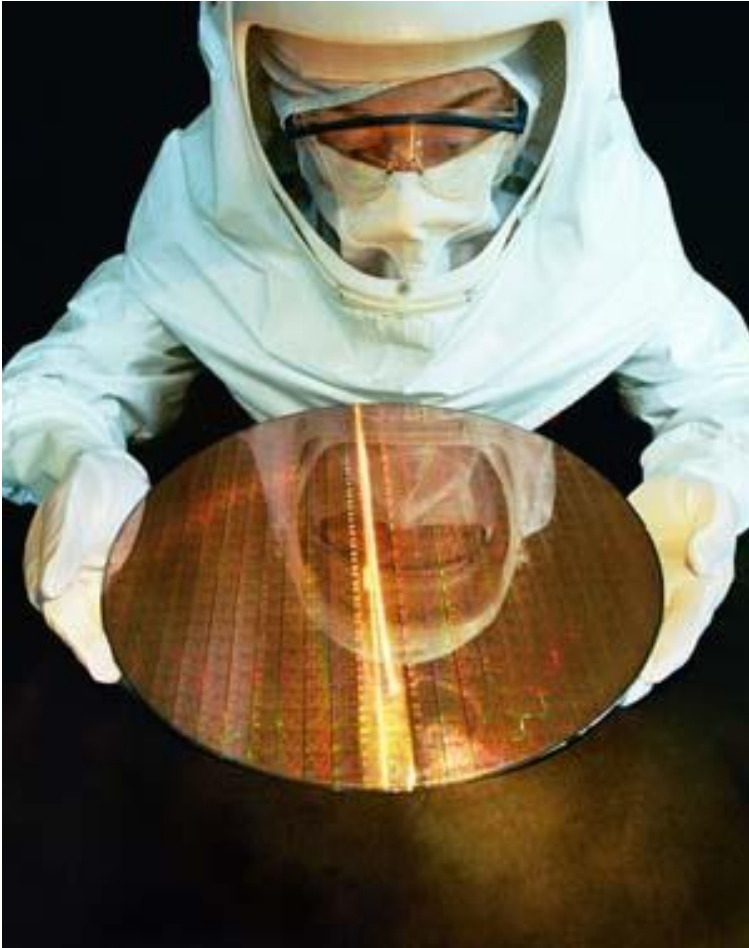


IC Fabrication

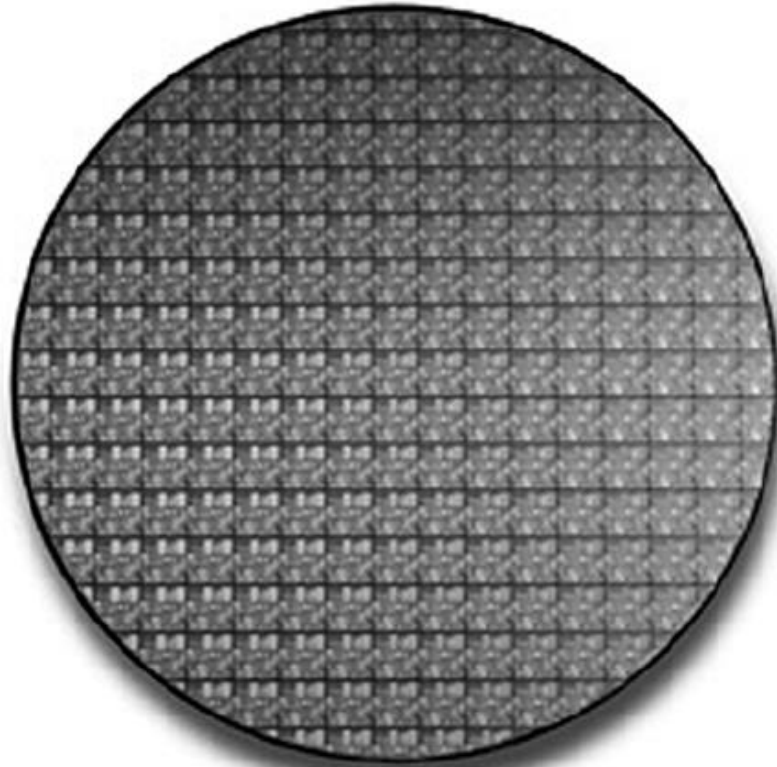
- Chips are fabricated using set of masks
 - Photolithography
- Basic steps
 - oxidize
 - apply photoresist
 - remove photoresist with mask
 - HF acid eats oxide but not photoresist
 - pirana acid eats photoresist
 - ion implantation (diffusion, wells)
 - vapor deposition (poly)
 - plasma etching (metal)



Silicon Wafer



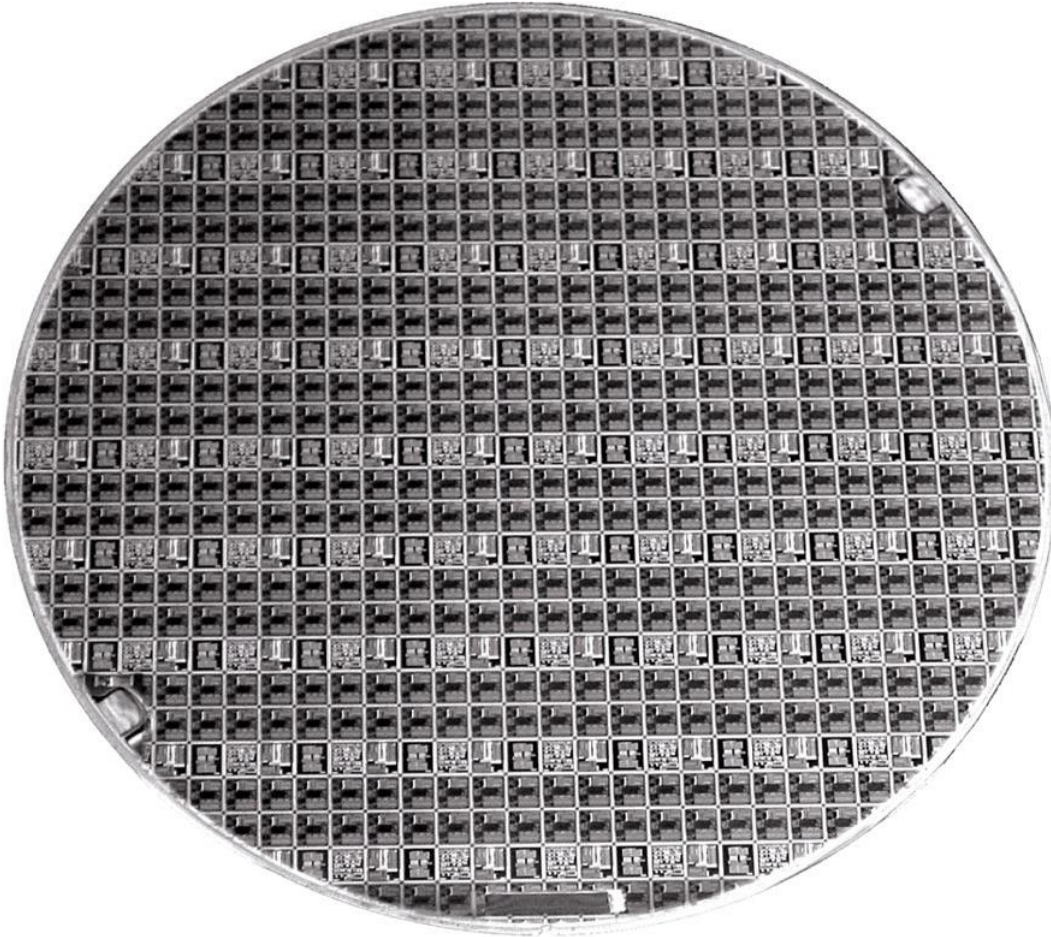
8" (200mm) Wafer



- 8 inch (200 mm) wafer containing Pentium 4 processors
 - 165 dies, die area = 250 mm², 55 million transistors, .18μm



Another 8" Wafer



Modern wafers are 12"
(300mm)

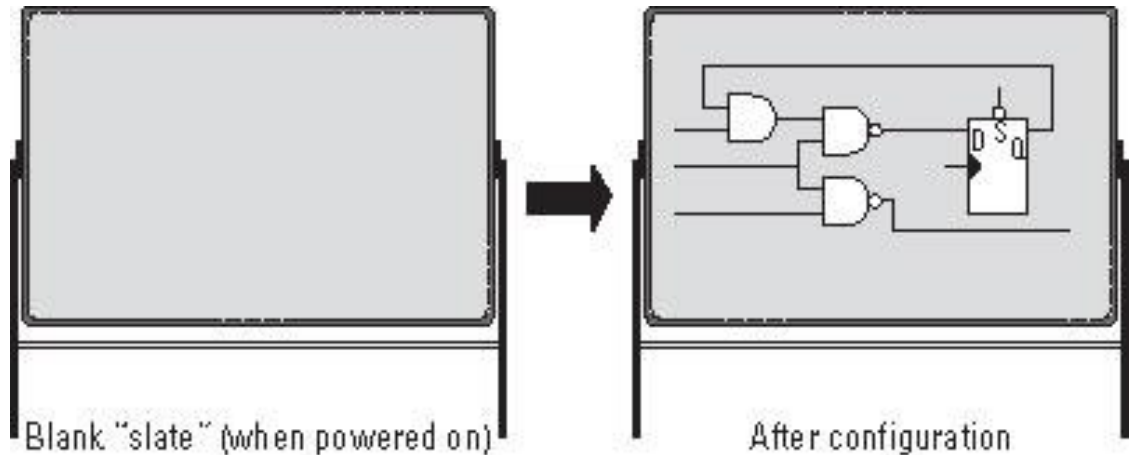
18" (450mm) are
forthcoming



Chip Fabrication

- Chip fabrication not practical for class:
 1. Requires high emphasis on analog design and requires slow analog circuit simulators
 2. Tedious design rules regarding feature placement and connecting substrates, wells, and diffusion areas
 3. Layout tools require at least one semester to learn
 4. Practical design considerations (I/O pads, power supply capacitors, signal integrity, etc.)
 5. Fabrication turnaround is several months
 6. Testing the chip is difficult
 - Need specialized equipment to mount chip and to generate inputs signals and read output signals

Option 3: Use Programmable Logic Device

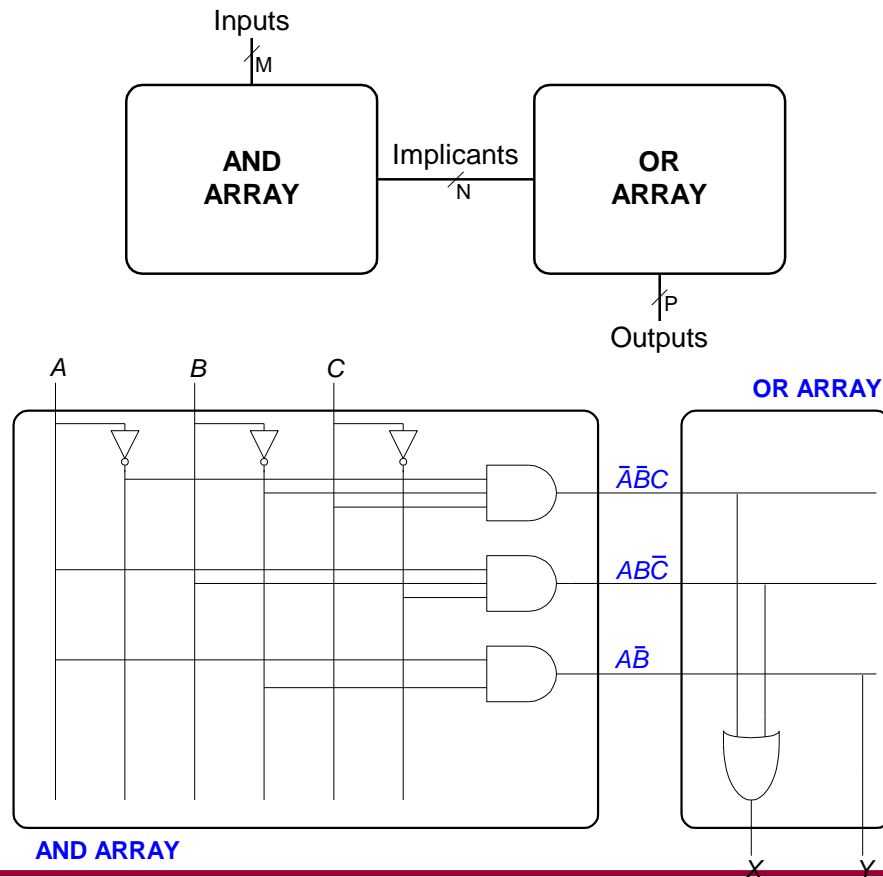


Programmable Logic Devices

- Programmable logic arrays (PLAs)
 - AND array followed by OR array
 - Perform combinational logic only
 - Fixed internal connections
- Field programmable gate arrays (FPGAs)
 - Array of configurable logic blocks (CLBs)
 - Perform combinational and sequential logic
 - Programmable internal connections

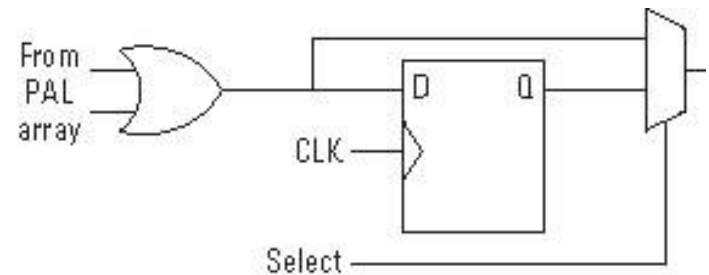
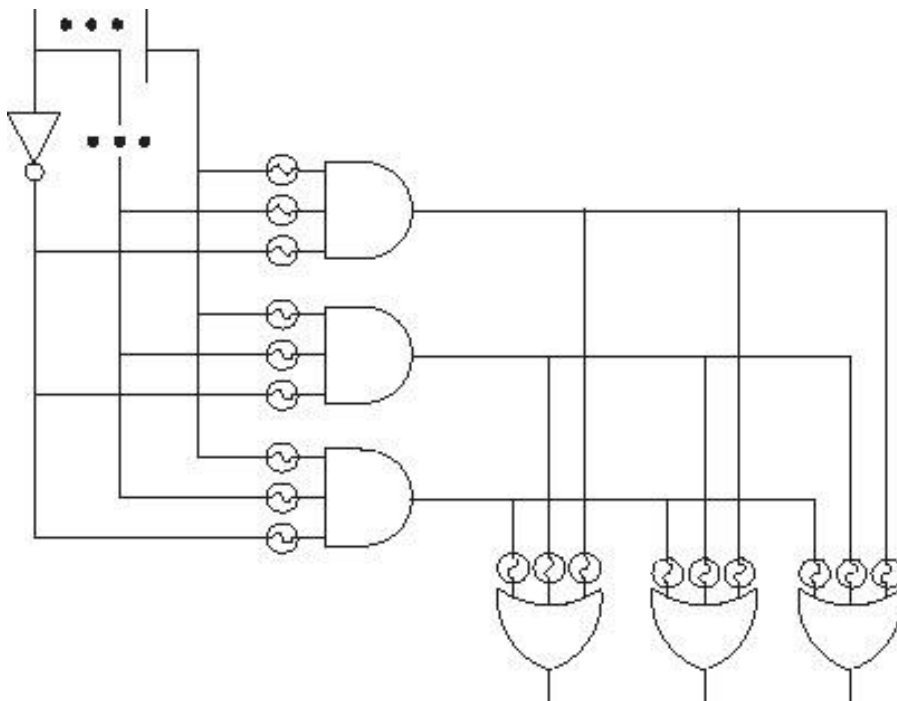
Programmable Logic Array

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

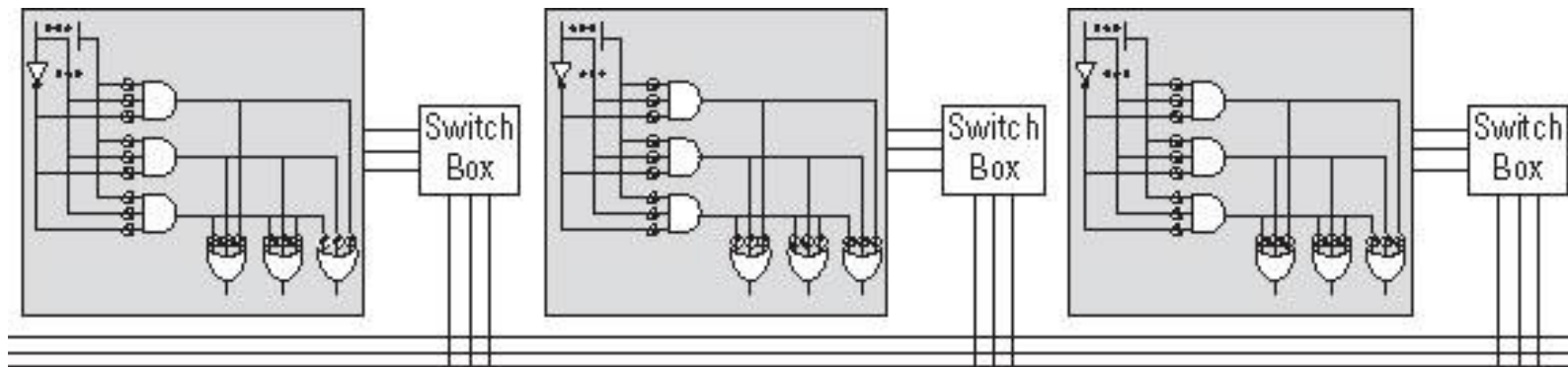


Programmable Logic Array

- Programmable Logic Array (PLA) Cell



Programmable Logic Array

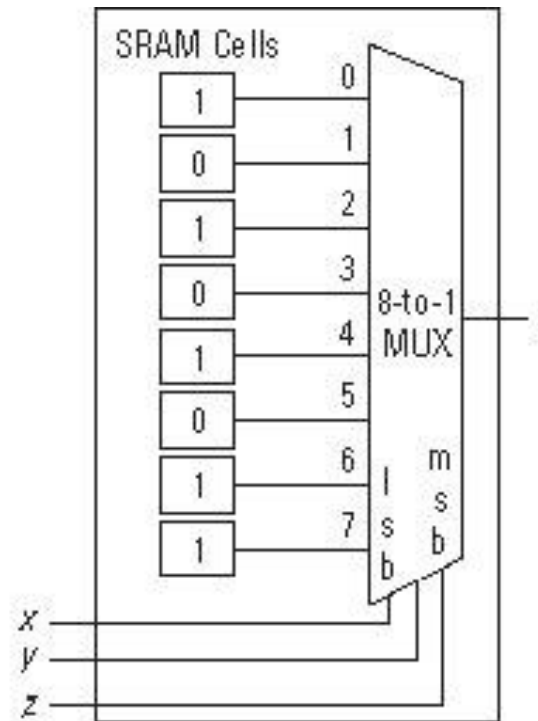


FPGA Lookup Table

- Function generator:

x	y	z	$xy + z'$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

(a)

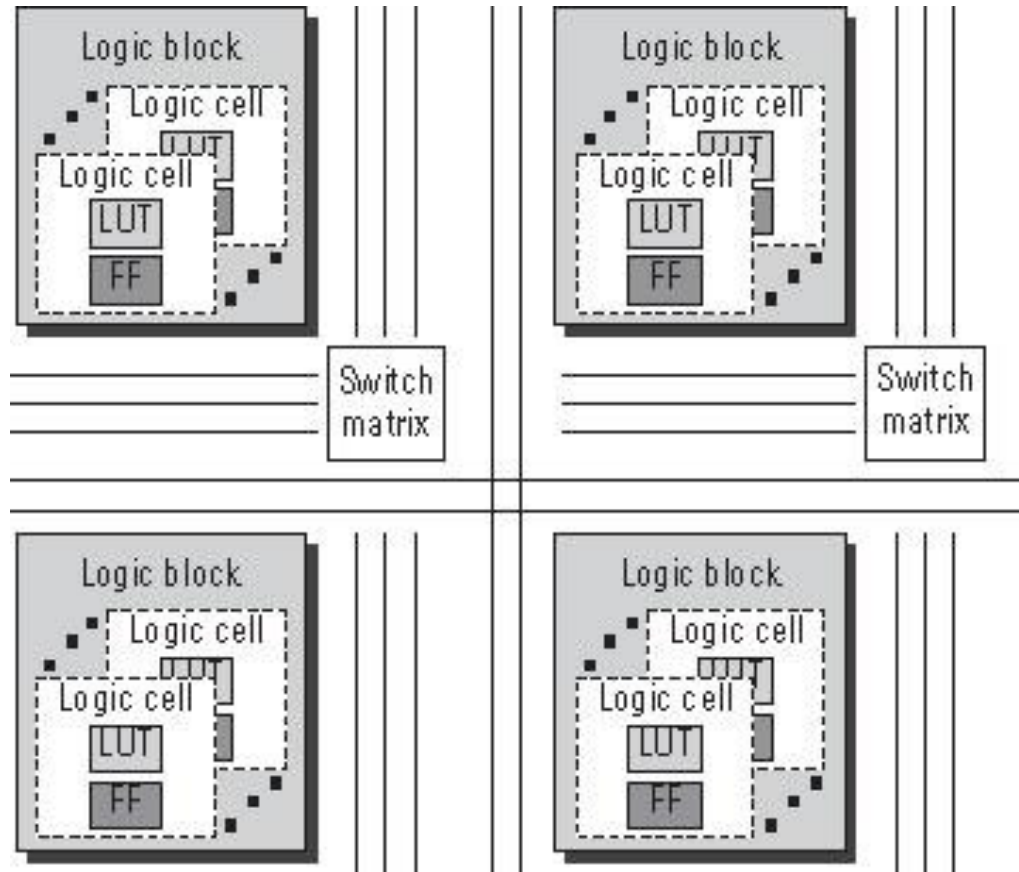


(b)

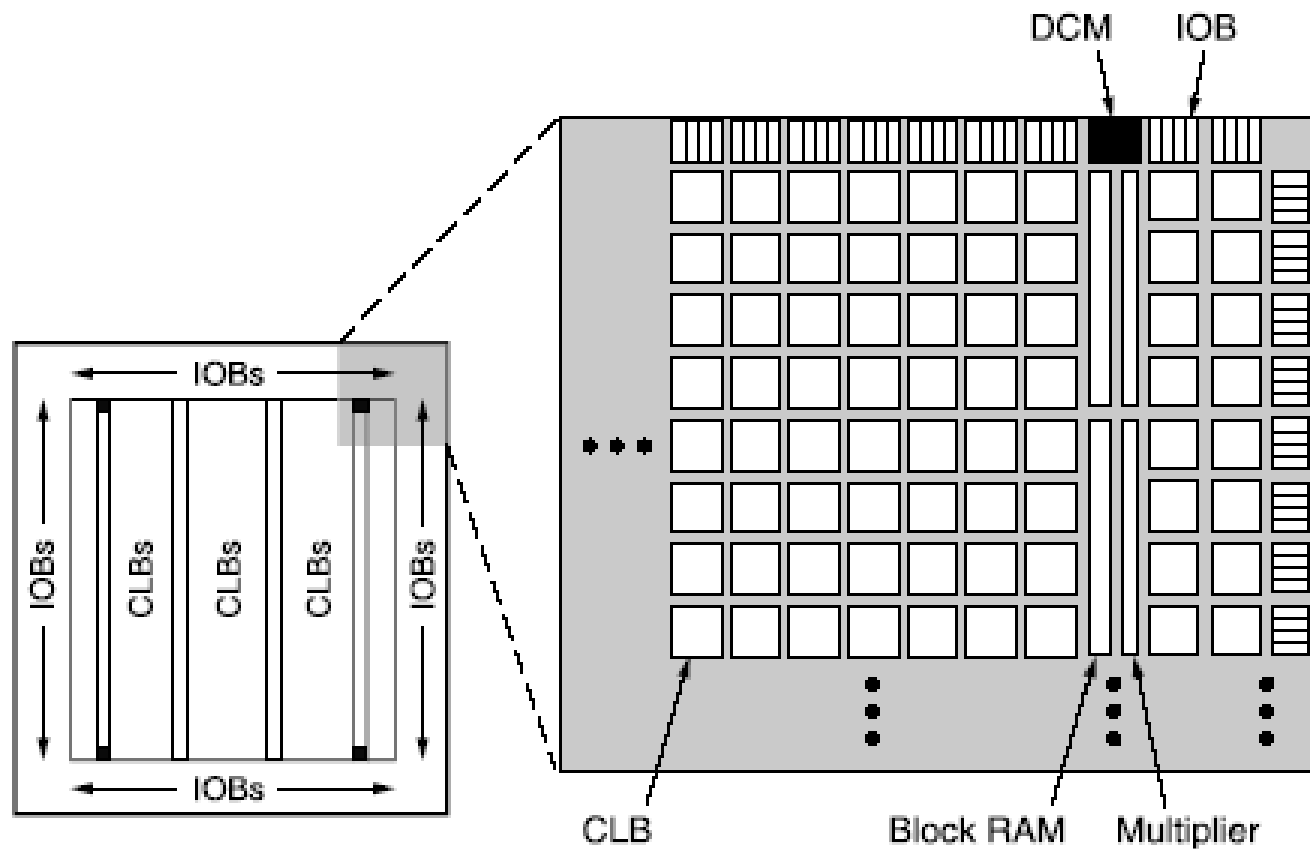


FPGA Fabric

- FPGA fabric:



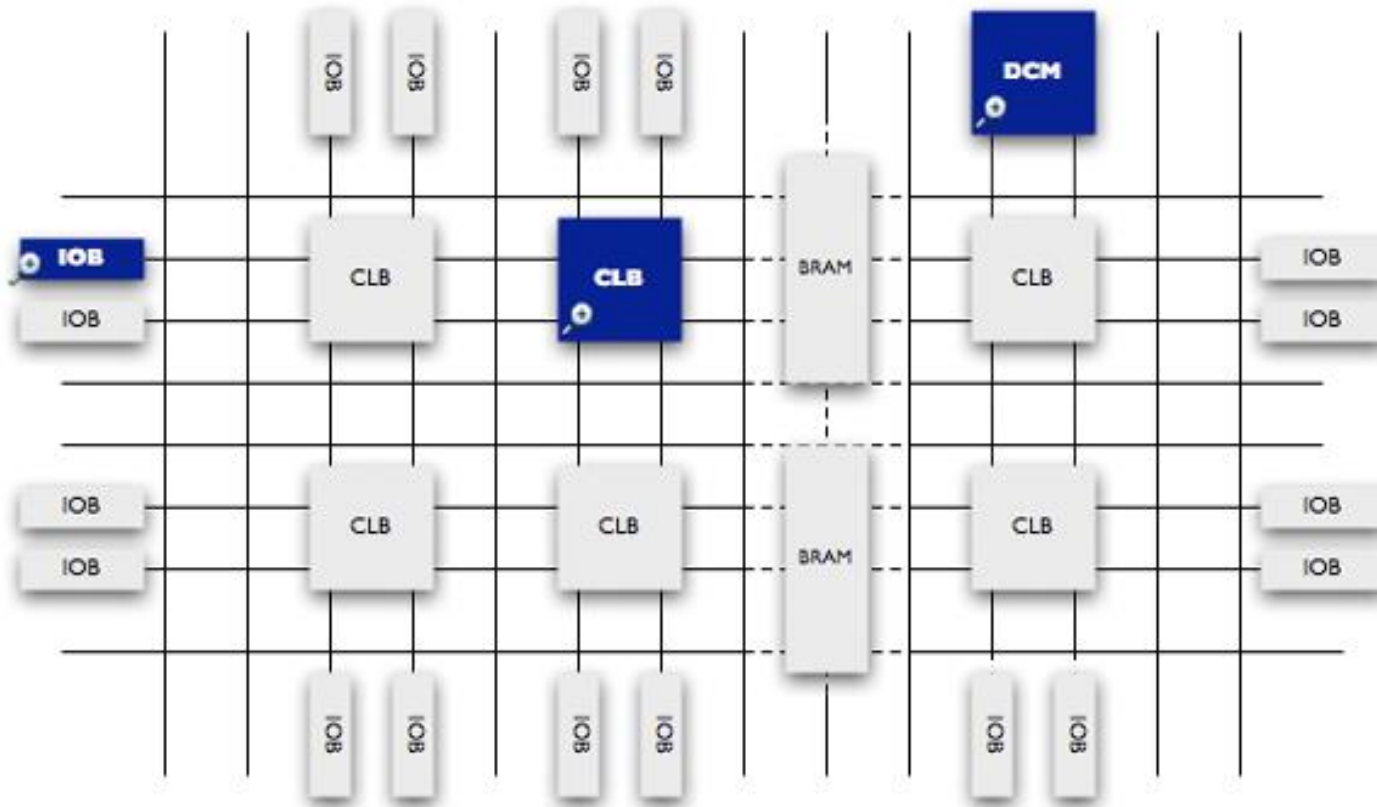
Xilinx Spartan 3 FPGA Schematic



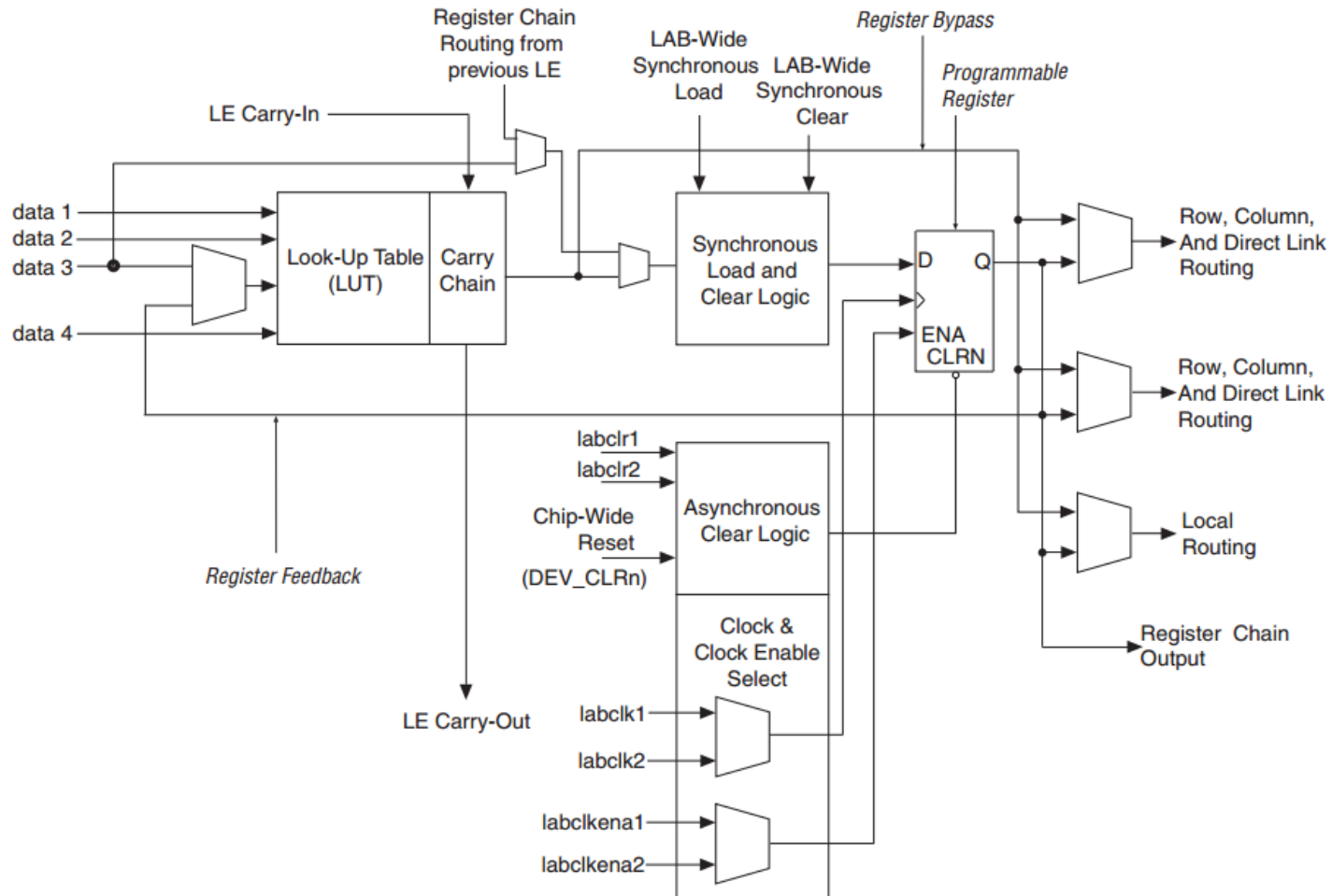
D6099-1_01_092703



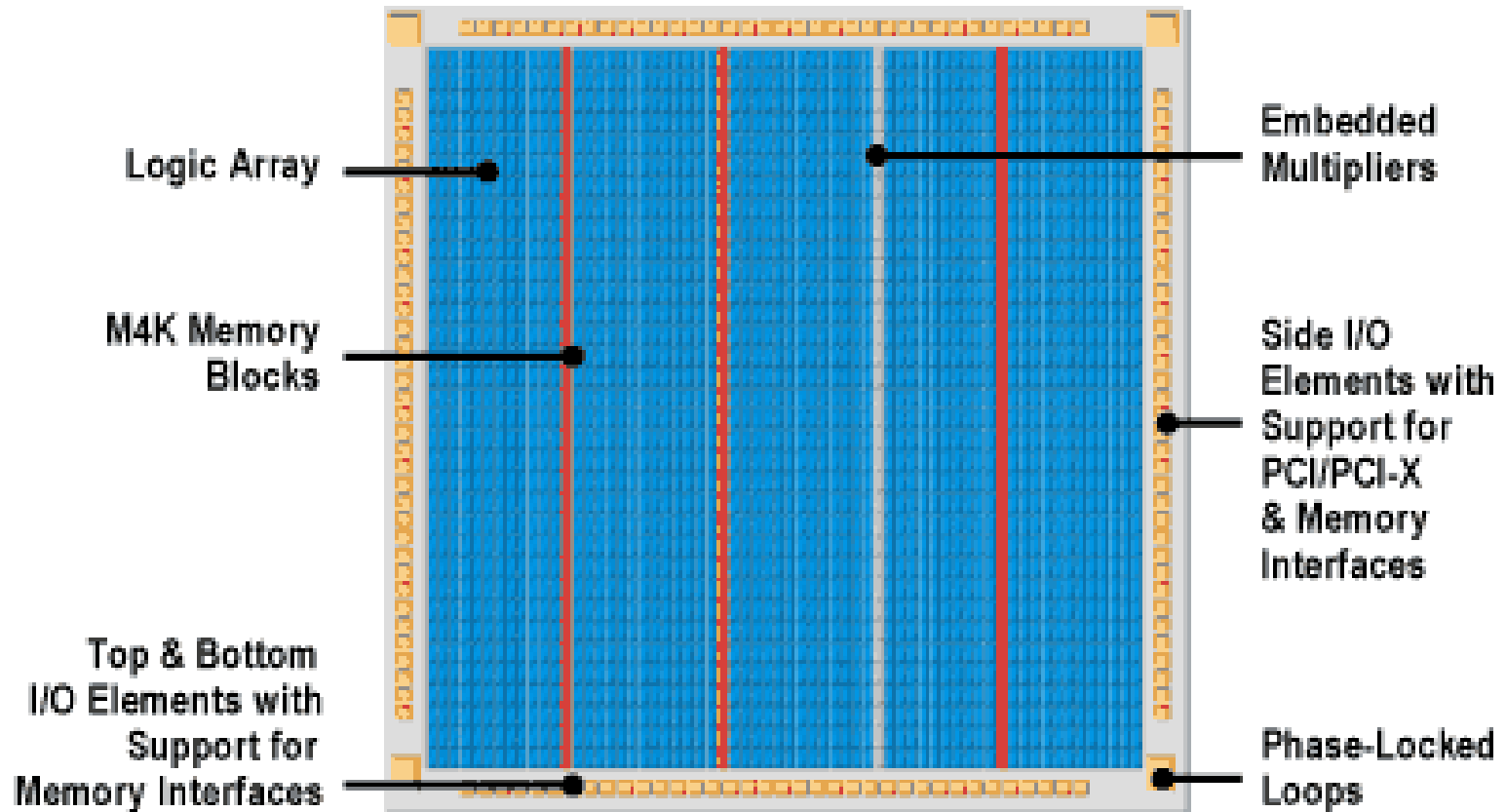
Field Programmable Gate Arrays



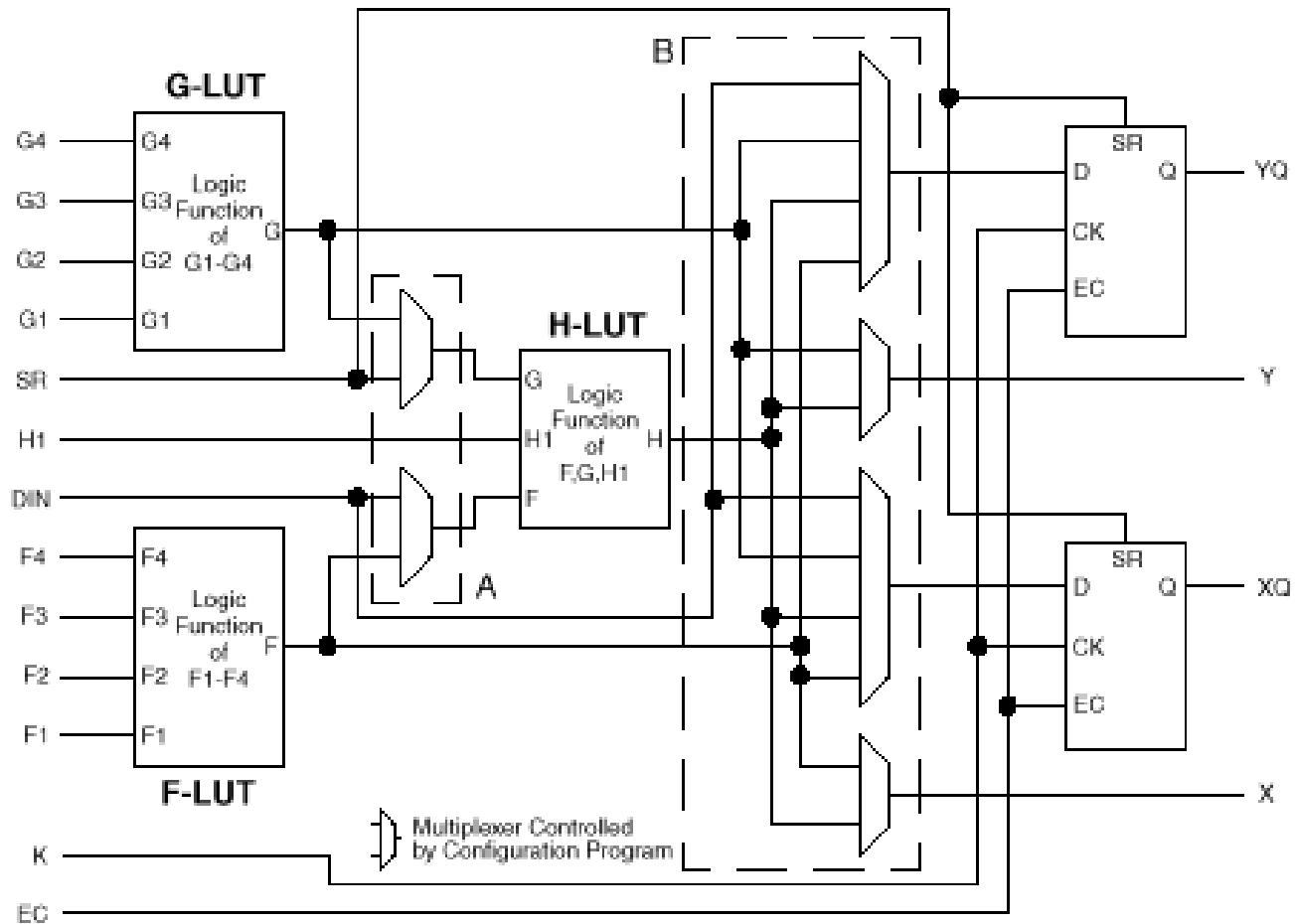
Cyclone 4 Logic Element



Cyclone 2 Design



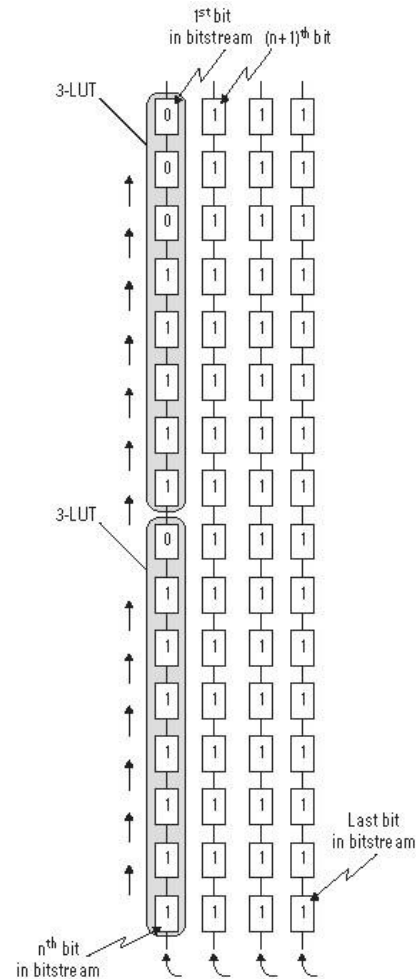
Xilinx Spartan CLB



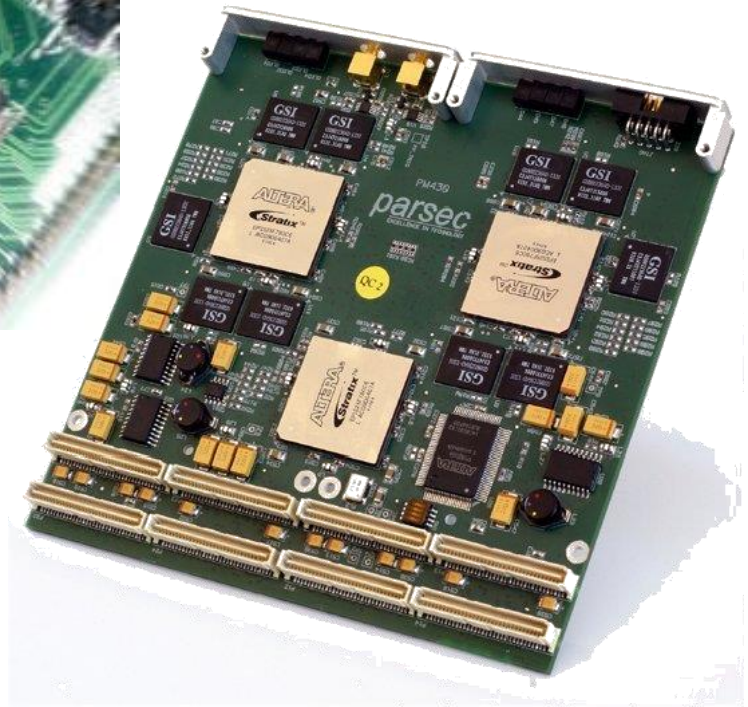
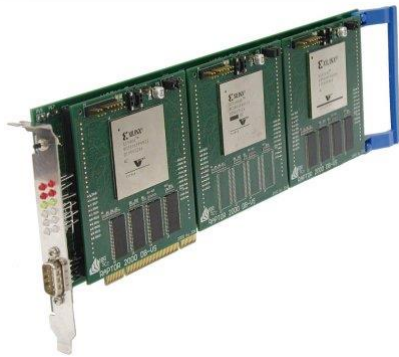
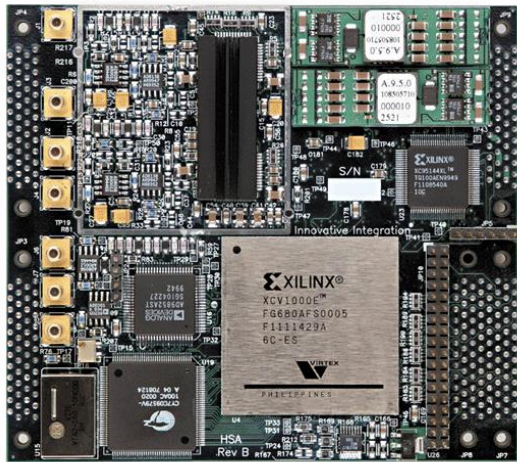
Rev 1.0



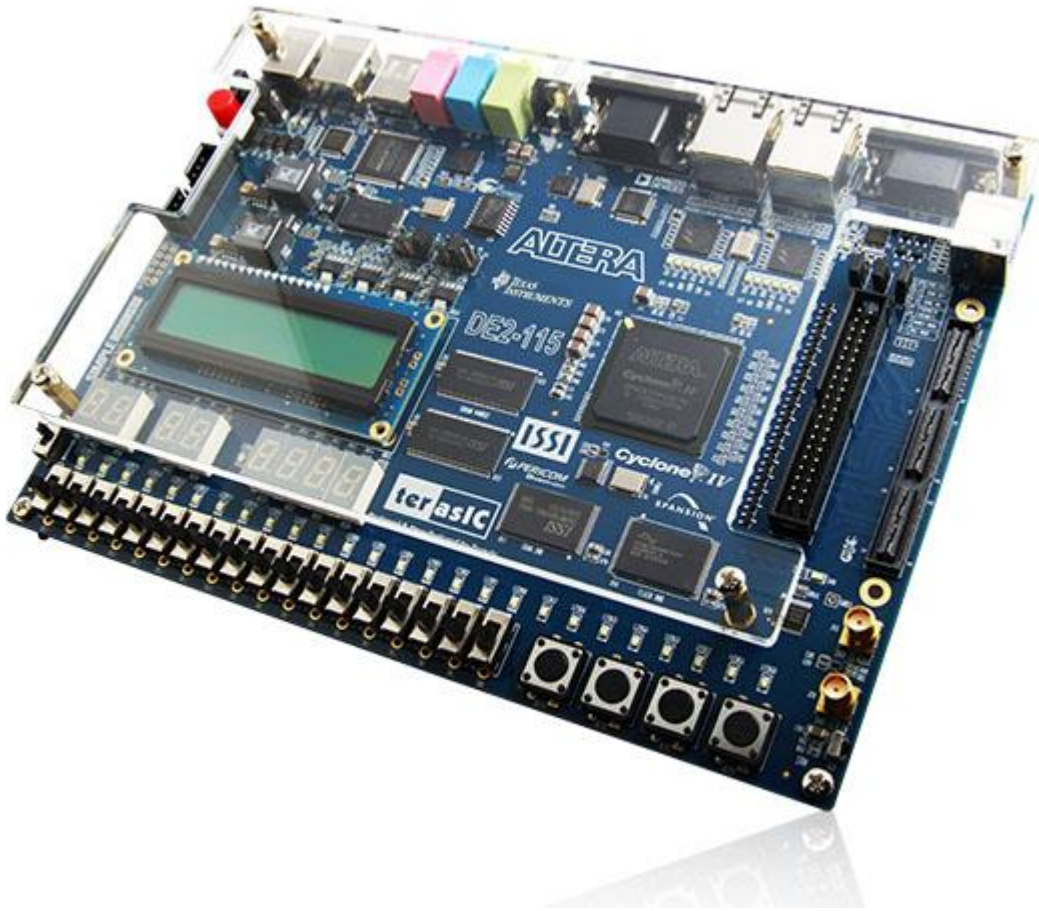
FPGA Configuration



Field Programmable Gate Arrays



Terasic DE2-115



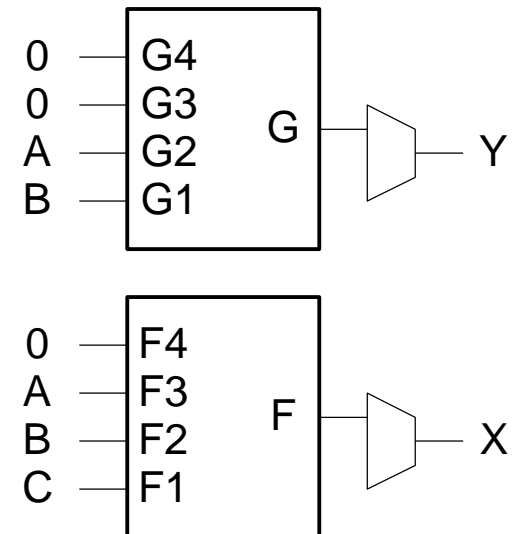
- DE2 board:
 - Altera Cyclone 4 FPGA with 115K gates

CLB Configuration Example

- Show how to configure the Spartan CLB to perform the following functions:
 - $X = \overline{A}BC + A\overline{B}C$
 - $Y = A\overline{B}$

	(A)	(B)	(C)	(X)
F4	F3	F2	F1	F
X	0	0	0	0
X	0	0	1	1
X	0	1	0	0
X	0	1	1	0
X	1	0	0	0
X	1	0	1	0
X	1	1	0	1
x	1	1	1	0

		(A)	(B)	(Y)
G4	G3	G2	G1	G
X	X	0	0	0
X	X	0	1	0
X	X	1	0	1
X	X	1	1	0



Field Programmable Gate Arrays

- Originally developed for “**glue logic**”
 - Interface between board-level components
 - Example: PCI interface chip, embedded microprocessor, ethernet interface chip
- Now used as **embedded platforms**, or **system-on a-programmable chip (SoPC)**
- Idea:
 - Implement:
 - customized “softcore” processor,
 - memory/cache subsystem,
 - I/O interfaces,
 - off-chip memory interfaces
 - ...entirely on an FPGA
 - Only board-level components needed are:
 - analog devices (i.e. analog, VGA), connector receptacles, memory chips, power components (voltage regulators, capacitors), digital interface that have a faster clock than is possible on an FPGA (i.e. USB2 interface)

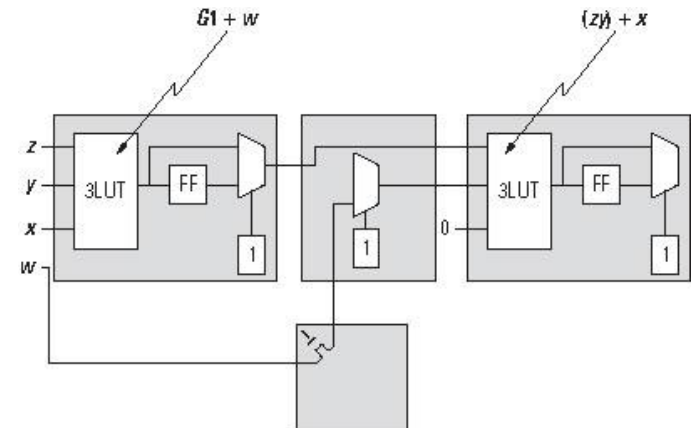
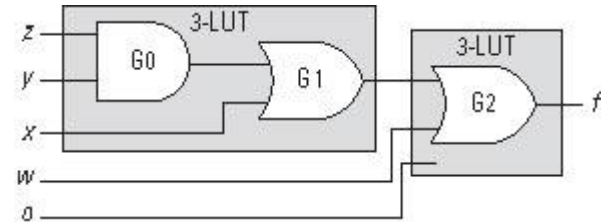


FPGA Design Flow

- A CAD tool is used to design and implement a digital system
- The user **enters the design** using schematic entry or an HDL
- The user **simulates** the design
- A **synthesis, map, place, and route** tool converts the code into hardware and maps it onto the FPGA
- The user uses the CAD tool to **download the configuration** onto the FPGA
- This configures the CLBs and the connections between them and the IOBs

Mapping, Placing, and Routing

- $f(w,x,y,z)=w+x+yz$
- Assume LUT3s only, so
 - $f(w,x,y,z)=f_2(w, f_1(x,y,z))$ where:
 - $f_2(w,t)=w+t$ and $f_1(x,y,z)=x+yz$
- Mapping: map gates to primitives
- Placement: assign primitives to specific LUTS on FPGA array
- Routing: configure interconnect



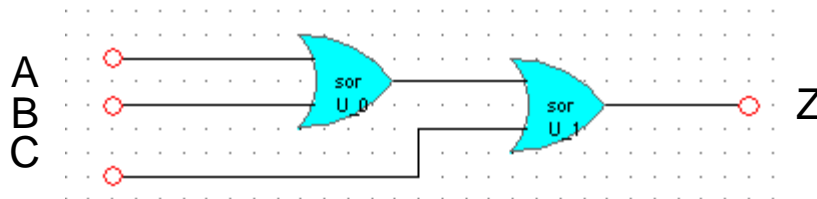
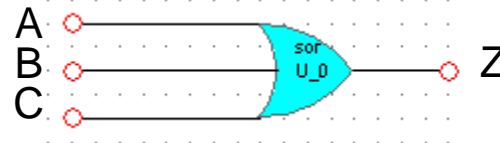
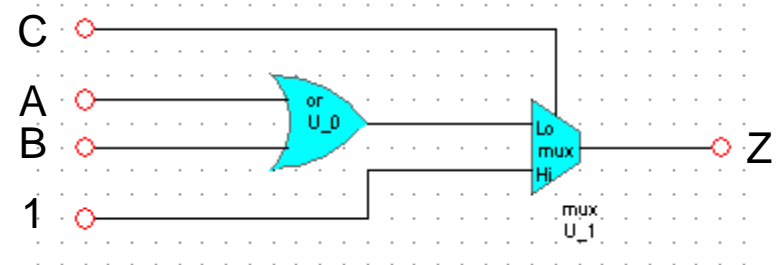
Synthesis

- Example:**

assign Z = A | B | C;

assign Z = C ? 1'b1 : A | B;

A	B	C	Z
0	0	0	0
0	1	0	1
1	0	0	1
1	1	0	1
0	0	1	1
0	1	1	1
1	0	1	1
1	1	1	1



Introduction

- Hardware description language (HDL): allows designer to specify logic function only
 - A computer-aided design (CAD) tool produces or *synthesizes* the optimized gates
 - Most commercial designs built using HDLs
- Two leading HDLs:

- **VHDL**

- VHSIC (“Very High Speed Integrated Circuit”) HDL
- Developed in 1981 by the Department of Defense to specify the behavior of custom chips, simulators and synthesizers followed
- Became an IEEE standard (1076) in 1987
- Based on ADA (looks like Pascal), designed to be verbose and easy-to-read

- **Verilog**

- developed in 1984 by Gateway Design Automation
- became an IEEE standard (1364) in 1995
- Based on C, easier to write than VHDL



HDL to Gates

- **Simulation**

- Input values are applied to the circuit
- Outputs checked for correctness
- Millions of dollars saved by debugging in simulation instead of hardware

- **Synthesis**

- Transforms HDL code into a *netlist* describing the hardware (i.e., a list of gates and the wires connecting them)

IMPORTANT:

When describing circuits using an HDL, it's critical to think of the **hardware** the code should produce.



Verilog Example

- Full adder:

```
module full_adder (input a,b,cin,output s,cout);  
    assign s = a ^ b ^ cin;  
    assign cout = (a & b) | (a & cin) | (b & cin);  
endmodule
```

- Synthesize:
(Compile)

A	B	CIN	S	COUT
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Mapping

- Assume our target FPGA has LUT2s
 - Can't map an 3-input function to one LUT2...

$$S = A \text{ xor } B \text{ xor } C_{in}$$

$$= A\bar{B}\bar{C}_{in} + \bar{A}B\bar{C}_{in} + \bar{A}\bar{B}C_{in} + AB C_{in}$$

$$= \bar{A}(B\bar{C}_{in} + \bar{B}C_{in}) + A(\bar{B}\bar{C}_{in} + BC_{in})$$

$$= \bar{A}(B\bar{C}_{in} + \bar{B}C_{in}) + A(\bar{B}\bar{C}_{in} + BC_{in} + \bar{B}B + \bar{C}_{in}C_{in})$$

$$= \bar{A}(B\bar{C}_{in} + \bar{B}C_{in}) + A((\bar{B} + C_{in})(B + \bar{C}_{in}))$$

$$= \bar{A}(B\bar{C}_{in} + \bar{B}C_{in}) + A\overline{(B\bar{C}_{in} + \bar{B}C_{in})}$$

$$S_0 = B\bar{C}_{in} + \bar{B}C_{in}$$

$$= \bar{A}S_0 + A\bar{S}_0$$



Verilog Example

A	B	CIN	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

A	B	CIN	S0	S
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

A	B	CIN	S0
X	0	0	0
X	0	1	1
X	1	0	1
X	1	1	0

A	B	CIN	S0	S
0	X	X	0	0
0	X	X	1	1
1	X	X	0	1
1	X	X	1	0

Rules:

1. S0 computed as a function of B and CIN
2. S computed as a function of A and S0



Place and Route

B	CIN	S0
0	0	0
0	1	1
1	0	1
1	1	0

A	S0	S
0	0	0
0	1	1
1	0	1
1	1	0

