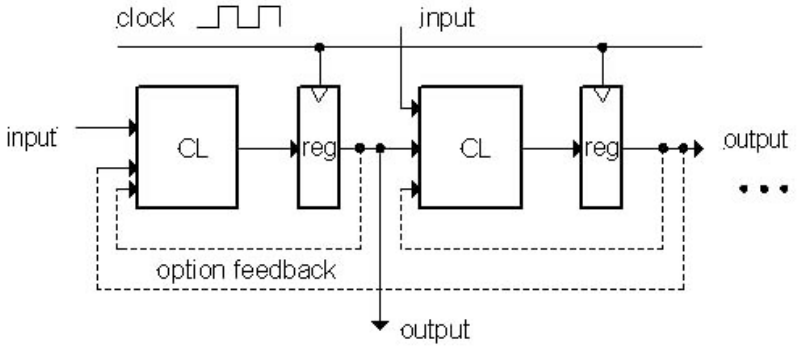
EECS150 – Digital Design
Lecture 2 – Combinational Logic Review and FPGAs

January 19, 2012

John Wawrzynek
Electrical Engineering and Computer Sciences
University of California, Berkeley

http://www-inst.eecs.berkeley.edu/~cs150

General Model for Synchronous Systems

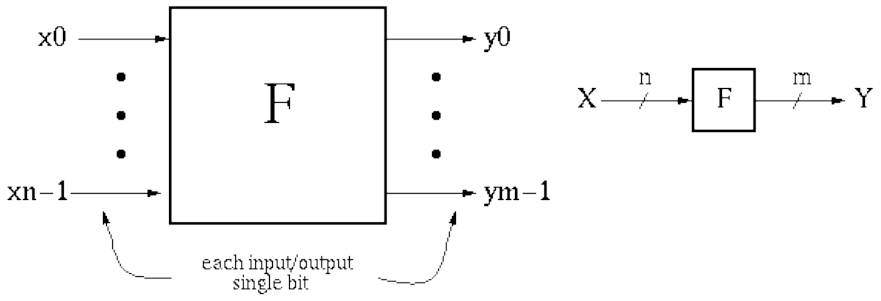


- All synchronous digital systems fit this model:
  - Collections of combinational logic blocks and state elements connected by signal wires. These form a directed graph with only two types of nodes (although the graph need not be bi-partite.)
  - Instead of simple registers, sometimes the state elements are large memory blocks.

Outline

- Review of three representations for combinational logic: truth tables, gate diagrams, algebraic equations
  - Laws of Boolean Algebra
  - Canonical Forms
  - Boolean Simplification
  - multi-level logic
  - NAND/NOR Networks
- Field Programmable Gate Arrays (FPGAs) Introduction

Combinational Logic (CL) Defined

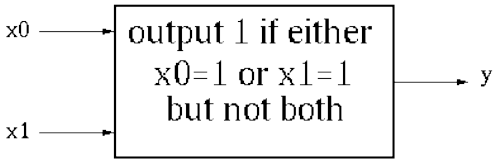


$y_i = f_i(x0, \ldots, xn-1)$, where x, y are {0,1}.

Y is a function of only X.

- If we change X, Y will change immediately (well almost!).
- There is an implementation dependent delay from X to Y.

CL Block Example #1



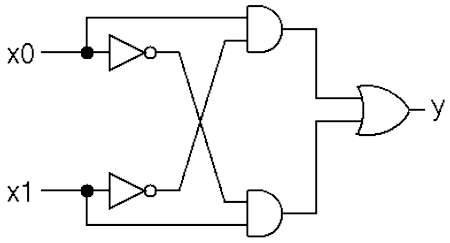
Boolean Equation:

$y_0 = [x_0 \text{ AND } \text{not}(x_1)]$
$\quad\quad \text{OR } [\text{not}(x_0) \text{ AND } x_1]$
$y_0 = x_0 x_1' + x_0' x_1$

Truth Table Description:

x0	x1	y
0	0	0
0	1	1
1	0	1
1	1	0

Gate Representation:



How would we *prove* that all three representations are equivalent?

Boolean Algebra/Logic Circuits

- Why are they called "logic circuits"?
- Logic: The study of the principles of reasoning.
- The 19th Century Mathematician, George Boole, developed a math. system (algebra) involving logic, Boolean Algebra.
- His variables took on TRUE, FALSE
- Later Claude Shannon (father of information theory) showed (in his Master's thesis!) how to map Boolean Algebra to digital circuits:
- Primitive functions of Boolean Algebra:

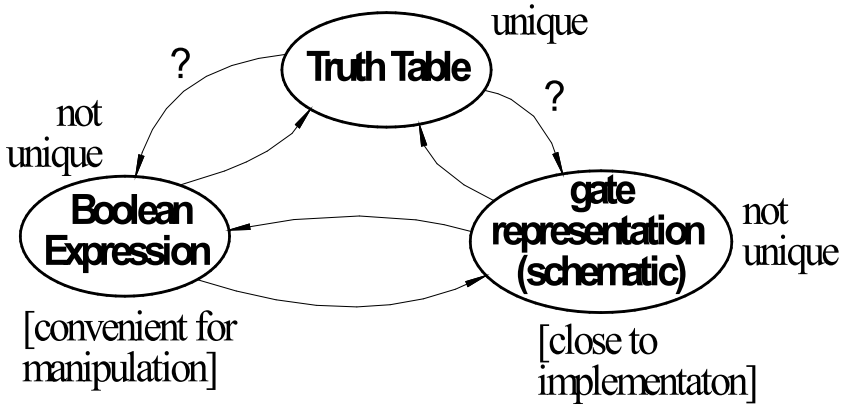
a b	AND
0 0	0
0 1	0
1 0	0
1 1	1

a b	OR
0 0	0
0 1	1
1 0	1
1 1	1

a	NOT
0	0
1	0

Relationship Among Representations

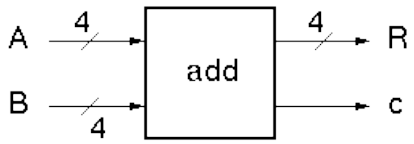
* Theorem: Any Boolean function that can be expressed as a truth table can be written as an expression in Boolean Algebra using AND, OR, NOT.



How do we convert from one to the other?

CL Block Example #2

- 4-bit adder:



R = A + B,
c is carry out

- Truth Table Representation:

a3	a2	a1	a0	b3	b2	b1	b0	r3	r2	r1	r0	c
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	1	1	0	0	1	1	0
0	0	0	0	0	1	0	0	0	1	0	0	0
•												
•												
•												
0	0	1	0	0	0	1	0	0	1	0	0	0
0	0	1	0	0	0	1	1	0	1	0	1	0
•												
•												
•												
0	0	0	1	1	1	1	1	0	0	0	0	1

In general: 2^n rows for n inputs. 256 rows!
Is there a more efficient (compact) way to specify this function?

4-bit Adder Example

- Motivate the adder circuit design by hand addition:

	a3	a2	a1	a0
+	b3	b2	b1	b0
c	r3	r2	r1	r0

- Add a0 and b0 as follows:

a	b	r	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

carry to next stage

r = a XOR b = a ⊕ b
c = a AND b = ab

	a3	a2	a1	a0
+	b3	b2	b1	b0
c	r3	r2	r1	r0

- Add a1 and b1 as follows:

ci	a	b	r	co
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

$r = a \oplus b \oplus c_i$

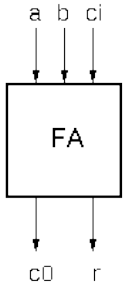
$co = ab + ac_i + bc_i$

4-bit Adder Example

- In general:

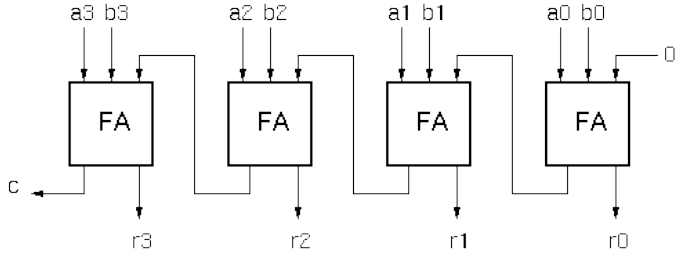
$r_i = a_i \oplus b_i \oplus c_{in}$

$c_{out} = a_i c_{in} + a_i b_i + b_i c_{in} = c_{in}(a_i + b_i) + a_i b_i$



"Full adder cell"

- Now, the 4-bit adder:



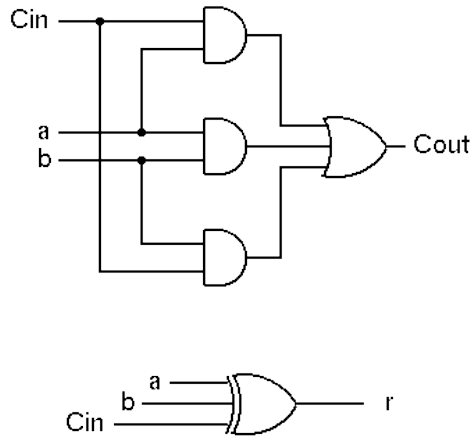
"ripple" adder

4-bit Adder Example

- Graphical Representation of FA-cell

  $r_i = a_i \oplus b_i \oplus c_{in}$

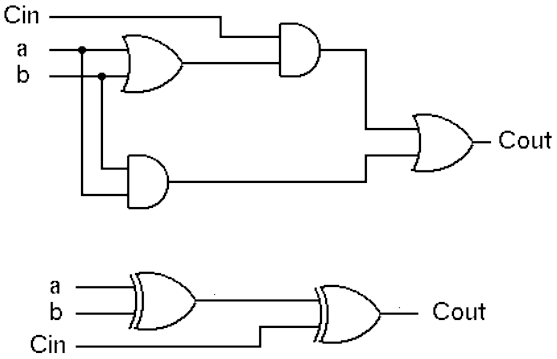
  $c_{out} = a_i c_{in} + a_i b_i + b_i c_{in}$



- Alternative Implementation (with 2-input gates):

  $r_i = (a_i \oplus b_i) \oplus c_{in}$

  $c_{out} = c_{in}(a_i + b_i) + a_i b_i$



Boolean Algebra

Defined as:

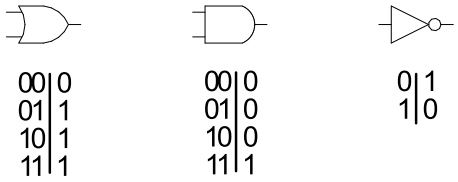
Set of elements B, binary operators $\{+, \bullet\}$, unary operation $\{'\}$, such that the following axioms hold:

1. B contains at least two elements a, b such that $a \neq b$.
2. Closure: a, b in B,
   $a + b$ in B, $a \bullet b$ in B, a' in B.
3. Communitive laws:
   $a + b = b + a$, $a \bullet b = b \bullet a$.
4. Identities: $0, 1$ in B
   $a + 0 = a$, $a \bullet 1 = a$.
5. Distributive laws:
   $a + (b \bullet c) = (a + b) \bullet (a + c)$, $a \bullet (b + c) = a \bullet b + a \bullet c$.
6. Complement:
   $a + a' = 1$, $a \bullet a' = 0$.

Logic Functions

$B = \{0,1\}$, + = OR, • = AND, $'$ = NOT

is a valid Boolean Algebra.



00	0	00	0	0	1
01	1	01	0	1	0
10	1	10	0		
11	1	11	1		

Do the axioms hold?

- Ex: communitive law: a+b = b+a?

Some Laws of Boolean Algebra

Duality: A dual of a Boolean expression is derived by interchanging OR and AND operations, and 0s and 1s (literals are left unchanged).

$$\{F(x_1, x_2, ..., x_n, 0, 1, +, \bullet)\}^D = \{F(x_1, x_2, ..., x_n, 1, 0, \bullet, +)\}$$

Any law that is true for an expression is also true for its dual.

Operations with 0 and 1:

1. x + 0 = x x * 1 = x
2. x + 1 = 1 x * 0 = 0

Idempotent Law:

3. x + x = x x x = x

Involution Law:

4. (x')' = x

Laws of Complementarity:

5. x + x' = 1 x x' = 0

Commutative Law:

6. x + y = y + x x y = y x

Laws of Boolean Algebra (cont.)

Associative Laws:

$(x + y) + z = x + (y + z)$ $\quad x\ y\ z = x\ (y\ z)$

Distributive Laws:

$x\ (y + z) = (x\ y) + (x\ z)$ $\quad x + (y\ z) = (x + y)(x + z)$

"Simplification" Theorems:

$x\ y + x\ y' = x$ $\quad (x + y)\ (x + y') = x$

$x + x\ y = x$ $\quad x\ (x + y) = x$

DeMorgan's Law:

$(x + y + z + \ldots)' = x'y'z'$ $\quad (x\ y\ z\ \ldots)' = x' + y' + z'$

Theorem for Multiplying and Factoring:

$(x + y)\ (x' + z) = x\ z + x'\ y$

Consensus Theorem:

$x\ y + y\ z + x'\ z = (x + y)\ (y + z)\ (x' + z)$

$x\ y + x'\ z = (x + y)\ (x' + z)$

Proving Theorems via axioms of Boolean Algebra

Ex: prove the theorem: $x\ y + x\ y' = x$

$x\ y + x\ y' = x\ (y + y')$ distributive law

$x\ (y + y')\ = x\ (1)$ complementary law

$x\ (1)\ = x$ identity

Ex: prove the theorem: $x + x\ y = x$

$x + x\ y\ = x\ 1 + x\ y$ identity

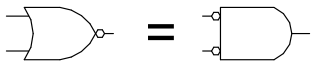
$x\ 1 + x\ y\ = x\ (1 + y)$ distributive law

$x\ (1 + y)\ = x\ (1)$ identity

$x\ (1)\ = x$ identity

DeMorgan's Law

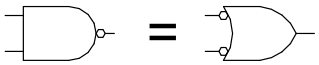
$(x + y)' = x' y'$



Exhaustive Proof

x	y	x'	y'	(x+y)'	x'y'
0	0	1	1	1	1
0	1	1	0	0	0
1	0	0	1	0	0
1	1	0	0	0	0

$(x y)' = x' + y'$

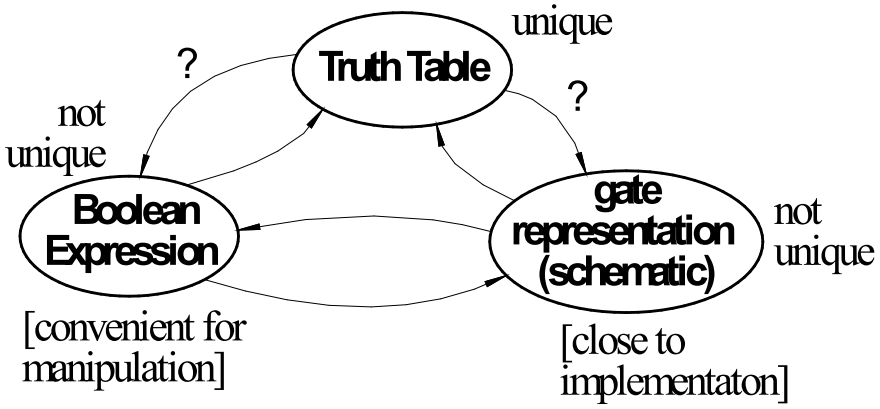


Exhaustive Proof

x	y	x'	y'	(xy)'	x'+y'
0	0	1	1	1	1
0	1	1	0	1	1
1	0	0	1	1	1
1	1	0	0	0	0

Relationship Among Representations

* Theorem: Any Boolean function that can be expressed as a truth table can be written as an expression in Boolean Algebra using AND, OR, NOT.



How do we convert from one to the other?

Canonical Forms

- Standard form for a Boolean expression - unique algebraic expression directly from a true table (TT) description.
- Two Types:
  * Sum of Products (SOP)
  * Product of Sums (POS)

- **Sum of Products** (disjunctive normal form, minterm expansion). Example:

minterms	a b c	f f'
a'b'c'	0 0 0	0 1
a'b'c	0 0 1	0 1
a'bc'	0 1 0	0 1
a'bc	0 1 1	1 0
ab'c'	1 0 0	1 0
ab'c	1 0 1	1 0
abc'	1 1 0	1 0
abc	1 1 1	1 0

One product (**and**) term for each 1 in f:

f = a'bc + ab'c' + ab'c +abc' +abc

f' = a'b'c' + a'b'c + a'bc'

Canonical Forms

- **Product of Sums** (conjunctive normal form, maxterm expansion). Example:

maxterms	a b c	f f'
a+b+c	0 0 0	0 1
a+b+c'	0 0 1	0 1
a+b'+c	0 1 0	0 1
a+b'+c'	0 1 1	1 0
a'+b+c	1 0 0	1 0
a'+b+c'	1 0 1	1 0
a'+b'+c	1 1 0	1 0
a'+b'+c'	1 1 1	1 0

One sum (**or**) term for each 0 in f:

f = (a+b+c)(a+b+c')(a+b'+c)

f' = (a+b'+c')(a'+b+c)(a'+b+c')
(a'+b'+c)(a+b+c')

Mapping from SOP to POS (or POS to SOP): Derive truth table then proceed.

Algebraic Simplification Example

Ex: full adder (FA) carry out function (in canonical form):

Cout = a'bc + ab'c + abc' + abc

Algebraic Simplification

Cout = a'bc + ab'c + abc' + abc

= a'bc + ab'c + abc' + **abc + abc**

= a'bc + **abc** + ab'c + abc' + **abc**

= **(a' + a)bc** + ab'c + abc' + abc

= **(1)bc** + ab'c + abc' + abc

= bc + ab'c + abc' + **abc + abc**

= bc + ab'c + **abc** + abc' + **abc**

= bc + **a(b' +b)c** + abc' +abc

= bc + **a(1)c** + abc' + abc

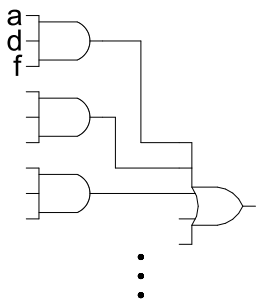
= bc + ac + **ab(c' + c)**

= bc + ac + **ab(1)**

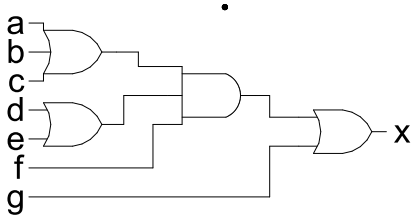
= bc + ac + ab

Multi-level Combinational Logic

- Example: reduced sum-of-products form
  x = adf + aef + bdf + bef + cdf + cef + g
- Implementation in 2-levels with gates:
  cost: 1 7-input OR, 6 3-input AND
  => 50 transistors
  delay: 3-input AND gate delay + 7-input OR gate delay



- Factored form:
  x = (a + b +c)(d + e)f + g
  cost: 1 3-input OR, 2 2-input OR, 1 3-input AND
  => 20 transistors
  delay: 3-input OR + 3-input AND + 2-input OR



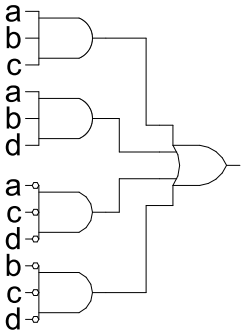
Footnote: NAND would be used in place of all ANDs and ORs.

Which is faster?

In general: Using multiple levels (more than 2) will reduce the cost. Sometimes also delay. Sometimes a tradeoff between cost and delay.

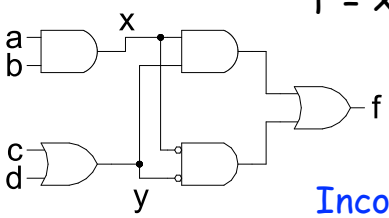
Multi-level Combinational Logic

Another Example: F = abc + abd +a'c'd' + b'c'd'



let x = ab y = c+d

f = xy + x'y'



Incorporates fanout.

No convenient hand methods exist for multi-level logic simplification:

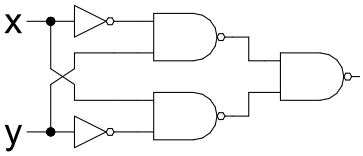
a) CAD Tools use sophisticated algorithms and heuristics
b) Humans and tools often exploit some special structure (example adder)

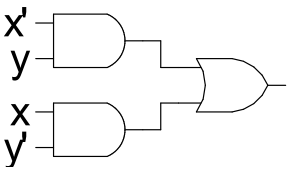
EXOR Function

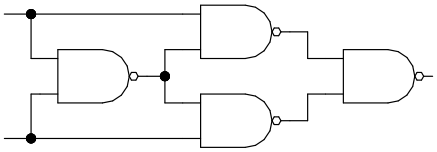
Parity, addition mod 2

$x \oplus y = x'y + xy'$

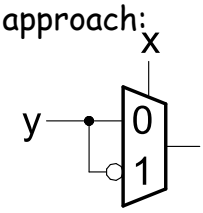
x y	xor	xnor
0 0	0	1
0 1	1	0
1 0	1	0
1 1	0	1





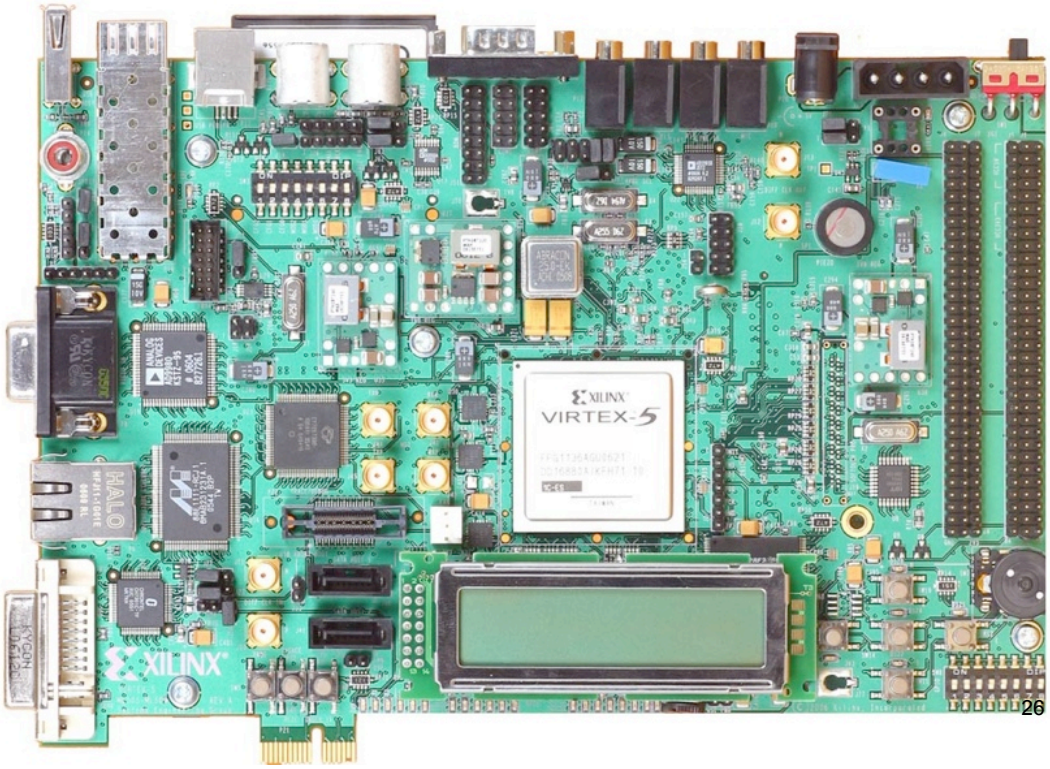


Another approach:

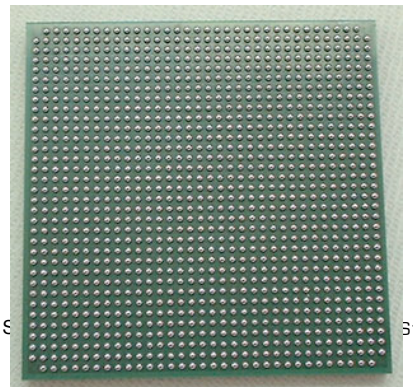


if x=0 then y else y'

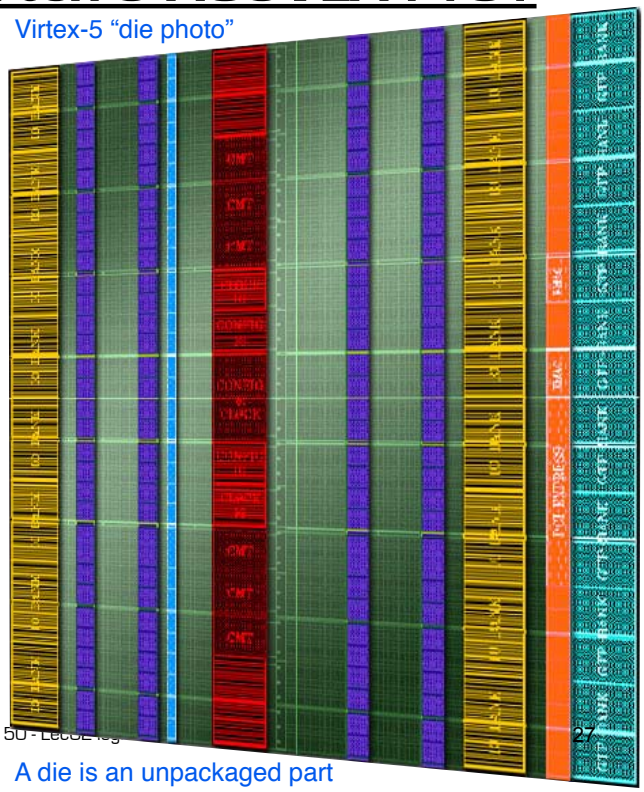
Project platform: Xilinx ML505-110



FPGA: Xilinx Virtex-5 XC5VLX110T



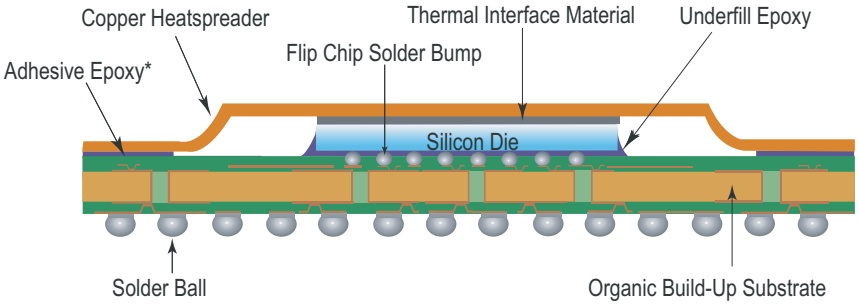
Virtex-5 “die photo”

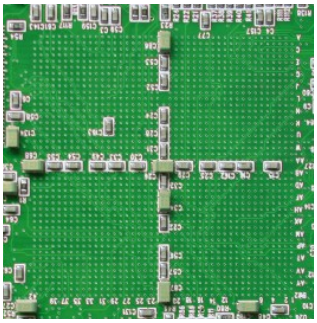


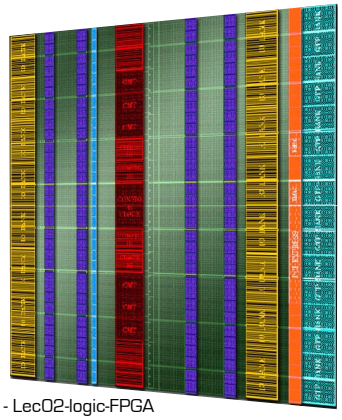
A die is an unpackaged part

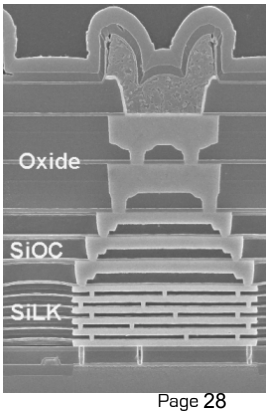
From die to PC board …

Ball Grid Array (BGA) Flip-Chip Package



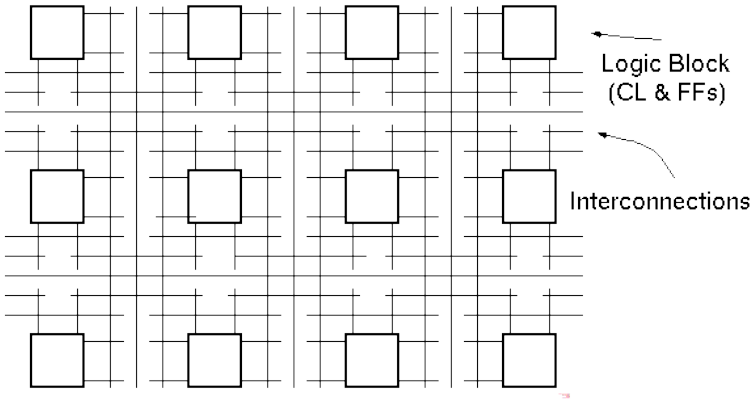






FPGA Overview

- Basic idea: two-dimensional array of logic blocks and flip-flops with a means for the user to configure (program):
  1. the interconnection between the logic blocks,
  2. the function of each block.



Simplified version of FPGA internal architecture:

Why are FPGAs Interesting?

- Technical viewpoint:
  - For hardware/system-designers, like ASICs only better! "Tape-out" new design every few minutes/hours.
  - Does the "reconfigurability" or "reprogrammability" offer other advantages over fixed logic?
  - Dynamic reconfiguration? In-field reprogramming? Self-modifying hardware, evolvable hardware?

Why are FPGAs Interesting?

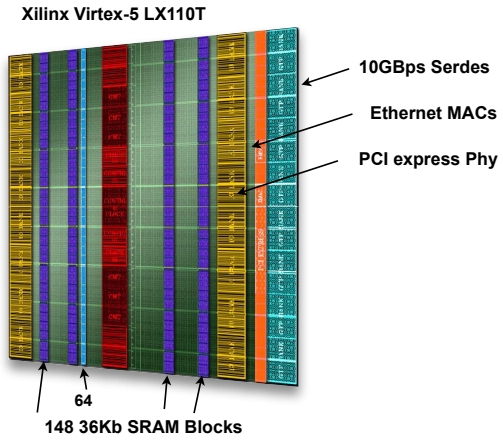
- Staggering logic capacity growth (10000x):

Year Introduced	Device	Logic Cells	"logic gate equivalents"
1985	XC2064	128	1024
2011	XC7V2000T	1,954,560	15,636,480

  - FPGAs have tracked Moore's Law better than any other programmable device.

Why are FPGAs Interesting?

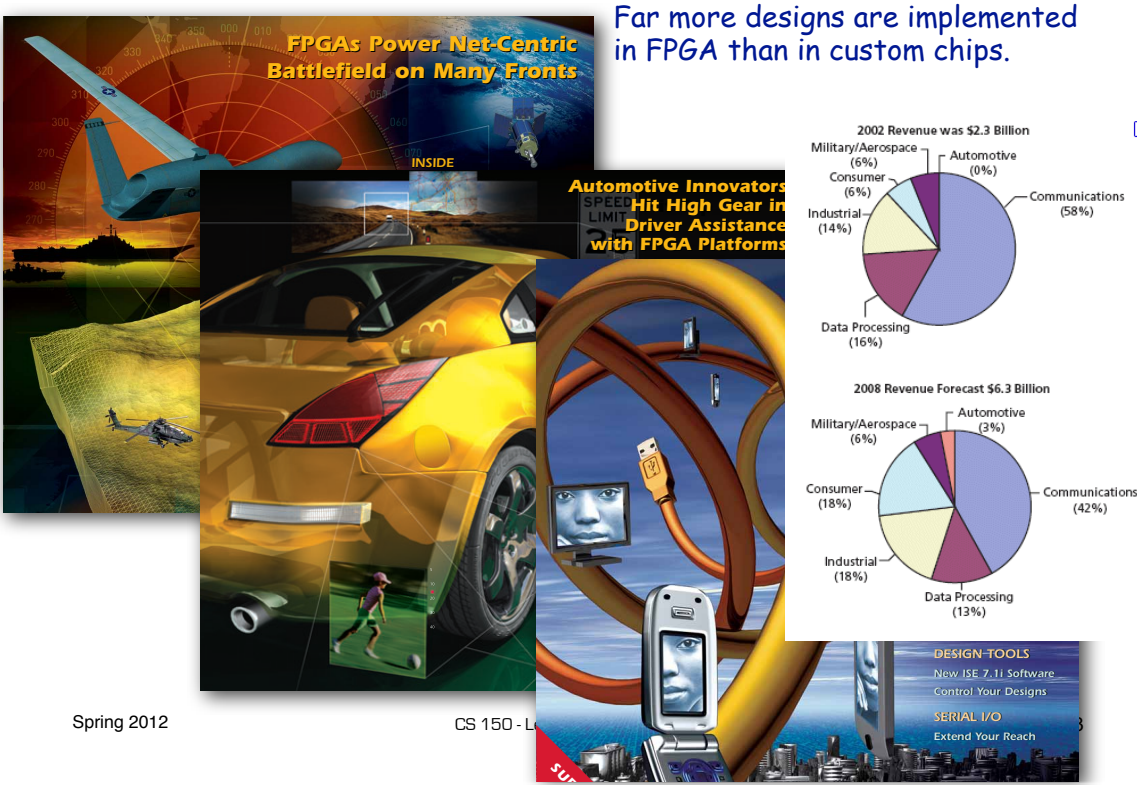
  - Logic capacity now only part of the story: on-chip RAM, high-speed I/Os, "hard" function blocks, …
  - Modern FPGAs are "reconfigurable systems"



But, the heterogeneity erodes the "purity" argument. Mapping is more difficult. Introduces uncertainty in efficiency of solution.

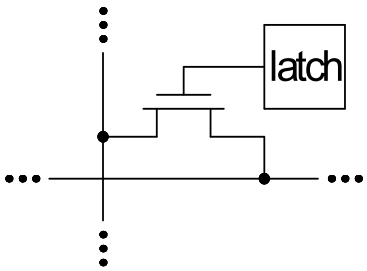
FPGAs are in widespread use

Far more designs are implemented in FPGA than in custom chips.



User Programmability

- Latch-based (Xilinx, Altera, ...)

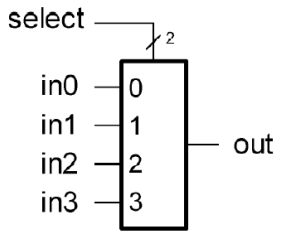


  + reconfigurable
  – volatile
  – relatively large.

- Latches are used to:
  1. control a switch to make or break cross-point connections in the interconnect
  2. define the function of the logic blocks
  3. set user options:
     - within the logic blocks
     - in the input/output blocks
     - global reset/clock
- "Configuration bit stream" is loaded under user control

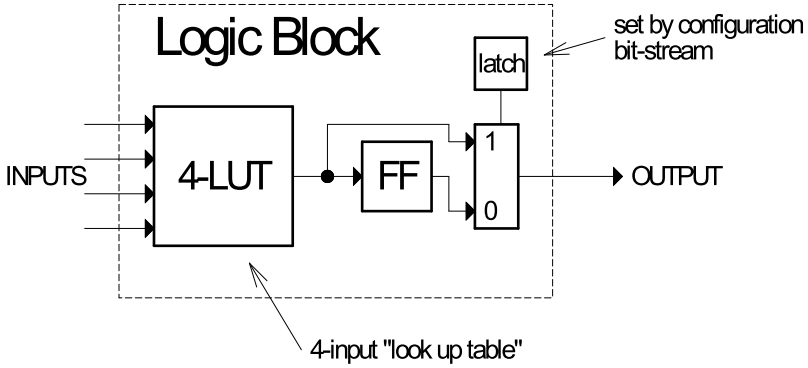
Background (review) for upcoming

- A MUX or multiplexor is a combinational logic circuit that chooses between 2^N inputs under the control of N control signals.



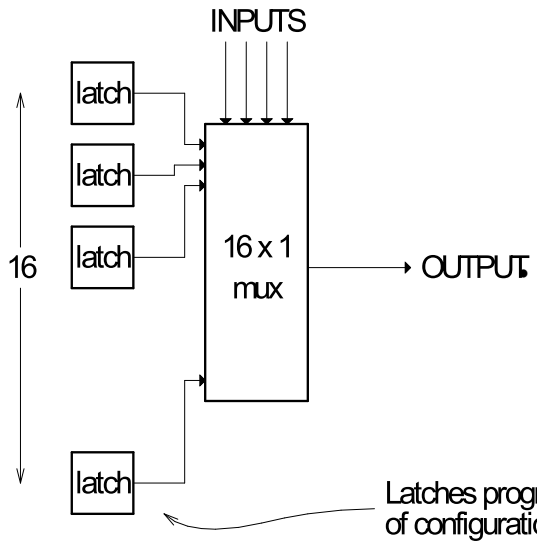
- A latch is a 1-bit memory (similar to a flip-flop).

Idealized FPGA Logic Block



- 4-input look up table (LUT)
  - implements combinational logic functions
- Register
  - optionally stores output of LUT

4-LUT Implementation



- n-bit LUT is implemented as a 2^n x 1 memory:
  - inputs choose one of 2^n memory locations.
  - memory locations (latches) are normally loaded with values from user's configuration bit stream.
  - Inputs to mux control are the CLB inputs.

Result is a general purpose "logic gate".

- n-LUT can implement any function of n inputs!

LUT as general logic gate

- An n-lut as a direct implementation of a function **truth-table**.
- Each latch location holds the value of the function corresponding to one input combination.

Example: 2-lut

INPUTS	AND	OR
00	0	0
01	0	1
10	0	1
11	1	1

Implements *any* function of 2 inputs.

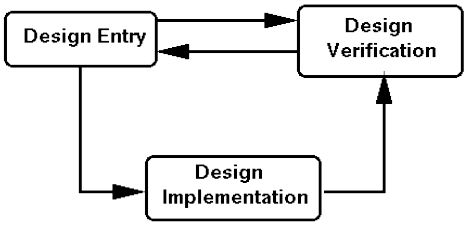
How many of these are there?

How many functions of n inputs?

Example: 4-lut

INPUTS	
0000	F(0,0,0,0) ← store in 1st latch
0001	F(0,0,0,1) ← store in 2nd latch
0010	F(0,0,1,0) ←
0011	F(0,0,1,1) ←
0011	
0100	•
0101	•
0110	•
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

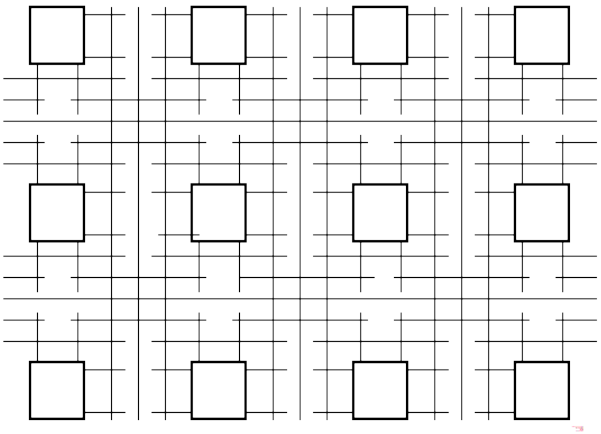
FPGA Generic Design Flow



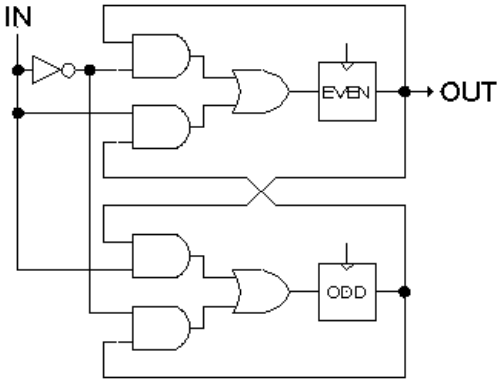
- Design Entry:
  - Create your design files using:
    - schematic editor or
    - HDL (hardware description languages: Verilog, VHDL)
- Design Implementation:
  - Logic synthesis (in case of using HDL entry) followed by,
  - Partition, place, and route to create configuration bit-stream file
- Design verification:
  - Optionally use simulator to check function,
  - Load design onto FPGA device (cable connects PC to development board), optional "logic scope" on FPGA
    - check operation at full speed in real environment.

Example Partition, Placement, and Route

- Idealized FPGA structure:



- Example Circuit:
  - collection of gates and flip-flops



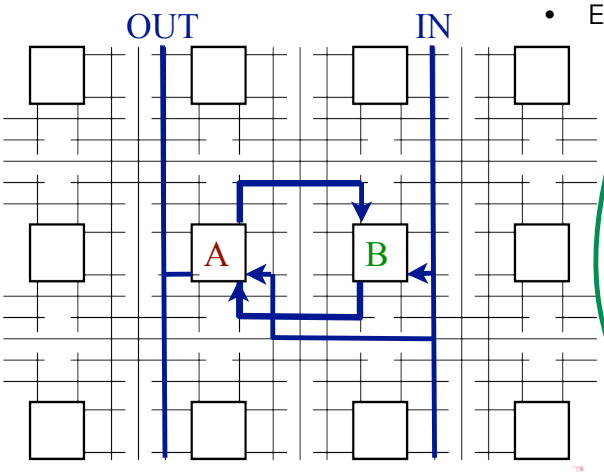
Circuit combinational logic must be "covered" by 4-input 1-output LUTs.

Flip-flops from circuit must map to FPGA flip-flops.
(Best to preserve "closeness" to CL to minimize wiring.)

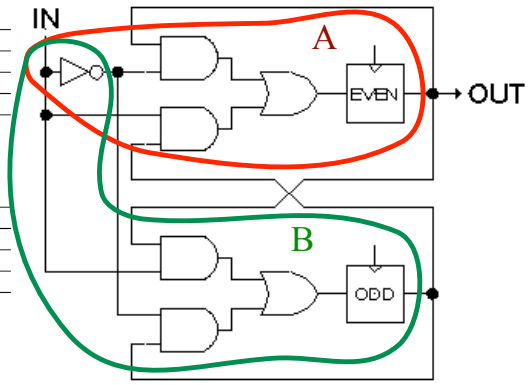
Best placement in general attempts to minimize wiring.

Vdd, GND, clock, and global resets are all "prewired".

Example Partition, Placement, and Route



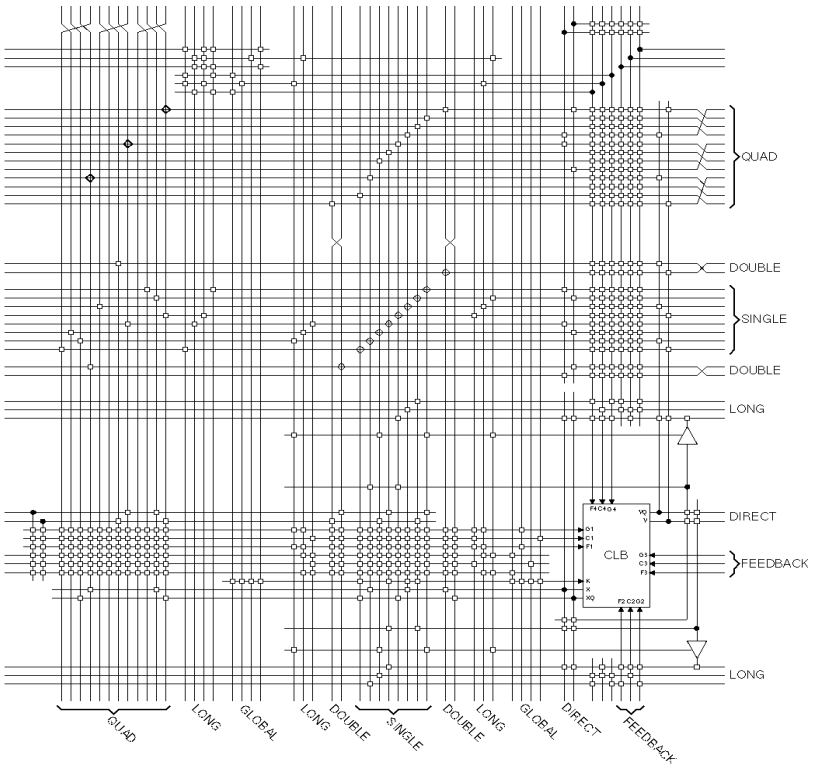
- Example Circuit:
  - collection of gates and flip-flops



Two partitions. Each has single output, no more than 4 inputs, and no more than 1 flip-flop. In this case, inverter goes in both partitions.

Note: the partition can be arbitrarily large as long as it has not more than 4 inputs and 1 output, and no more than 1 flip-flop.

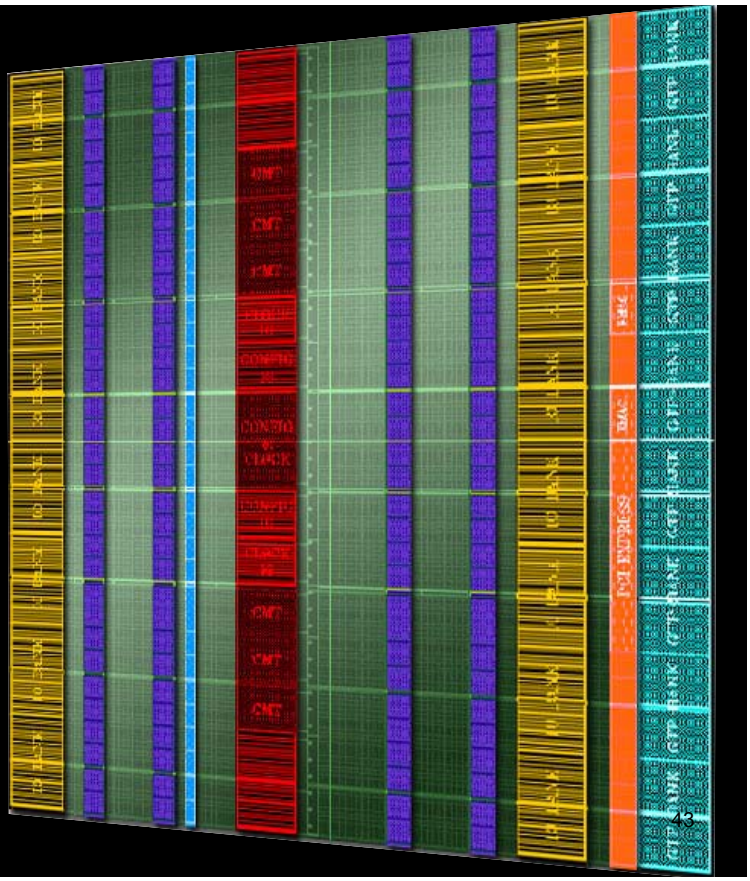
Xilinx FPGAs (interconnect detail)



Colors represent different types of resources:

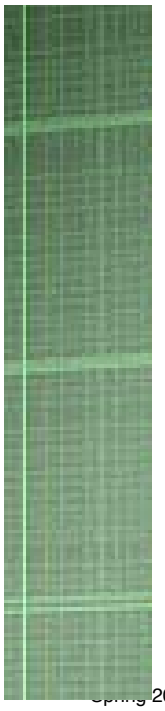
Logic
Block RAM
DSP (ALUs)
Clocking
I/O
Serial I/O + PCI

A routing fabric runs throughout the chip to wire everything together.

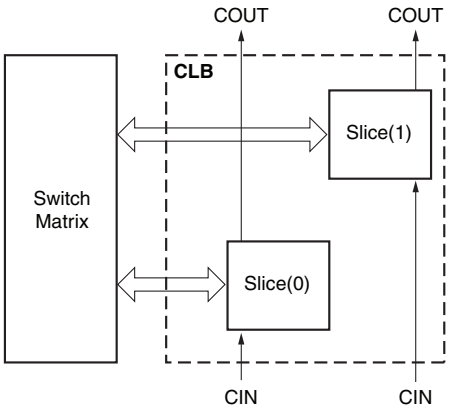


43

Configurable Logic Blocks (CLBs)



Slices define regular connections to the switching fabric, and to slices in CLBs above and below it on the die.



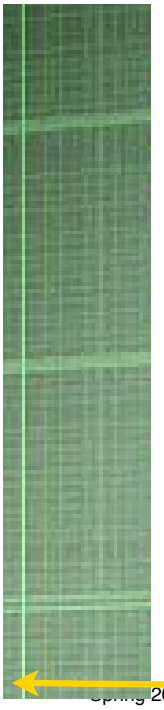
The LX110T has 17,280 slices.

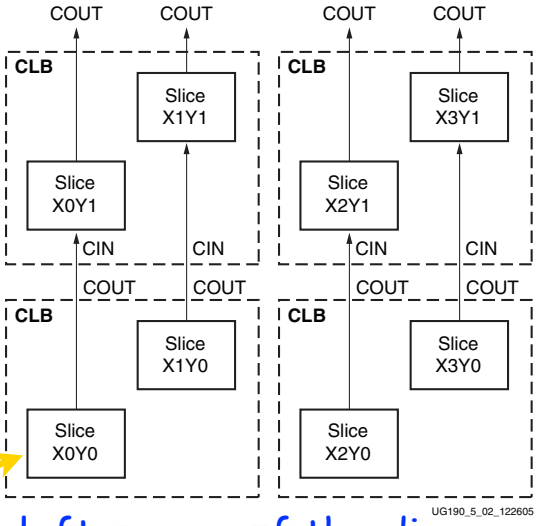
X-Y naming convention for slices

X0, X2, ... are lower CLB slices.

X1, X3, ... are upper CLB slices.

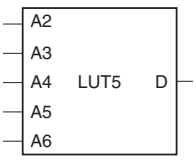
Y0, Y1, ... are CLB column positions.



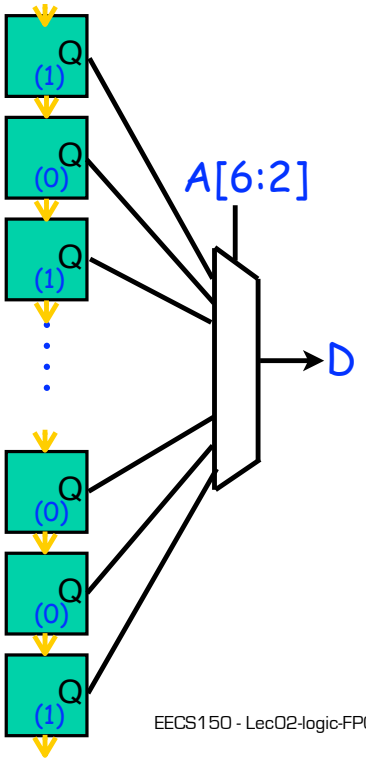


Lower-left corner of the die.

Atoms: 5-input Look Up Tables (LUTs)



A[6:2]	D
00000	1
00001	0
00010	1
⋮	
11101	0
11110	0
11111	1



Computes any 5-input logic function.

Timing is independent of function.

Latches set during configuration.

Virtex-5 6-LUTs: Composition of 5-LUTs

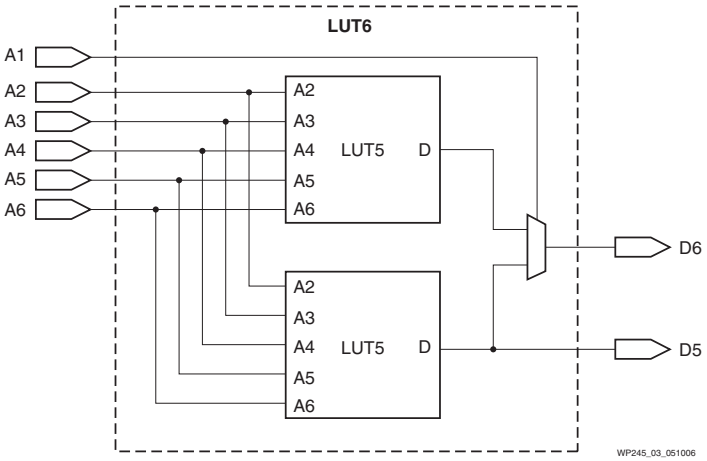


Figure 3: **Block Diagram of a Virtex-5 6-Input LUT**

May be used as one 6-input LUT (D6 out) ...

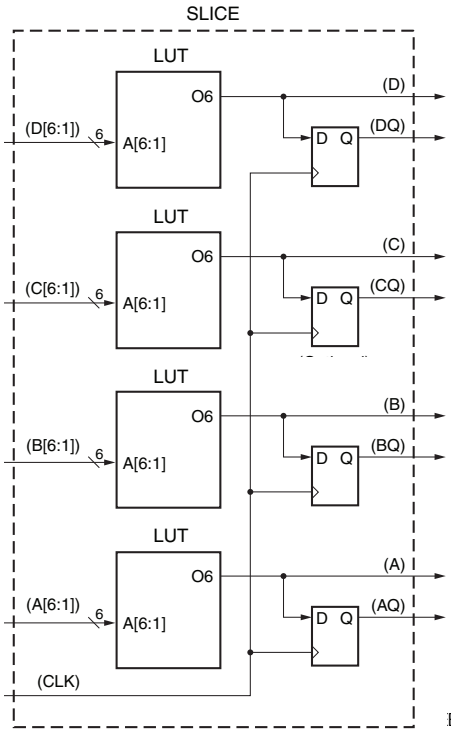
... or as two 5-input LUTS (D6 and D5)

The LX110T has 69,120 6-LUTs

6-LUT delay is 0.9 ns

Combinational logic (post configuration)

The simplest view of a slice



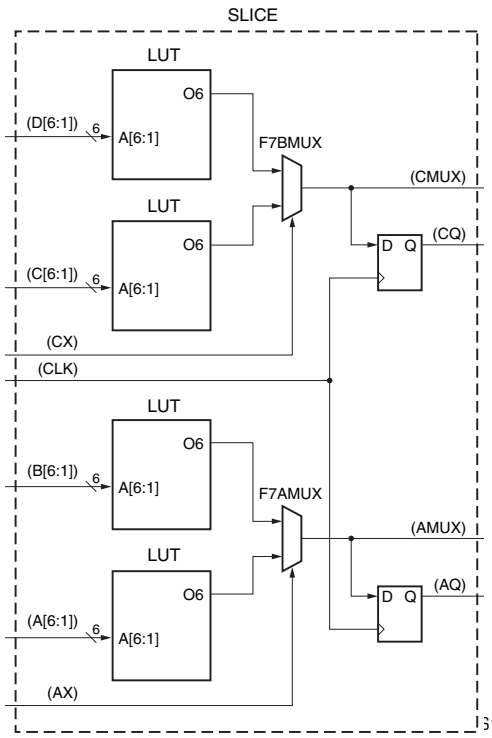
Four 6-LUTs

Four Flip-Flops

Switching fabric may see combinational and registered outputs.

An actual Virtex-5 slice adds many small features to this simplified diagram. We show them one by one ...

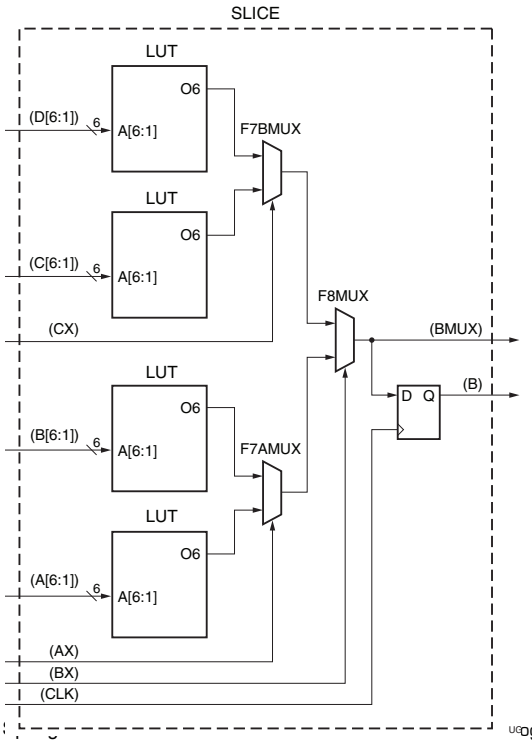
Two 7-LUTs per slice ...



Extra multiplexers(F7AMUX, F7BMUX)

Extra inputs (AX and CX)

Or one 8-LUTs per slice ...

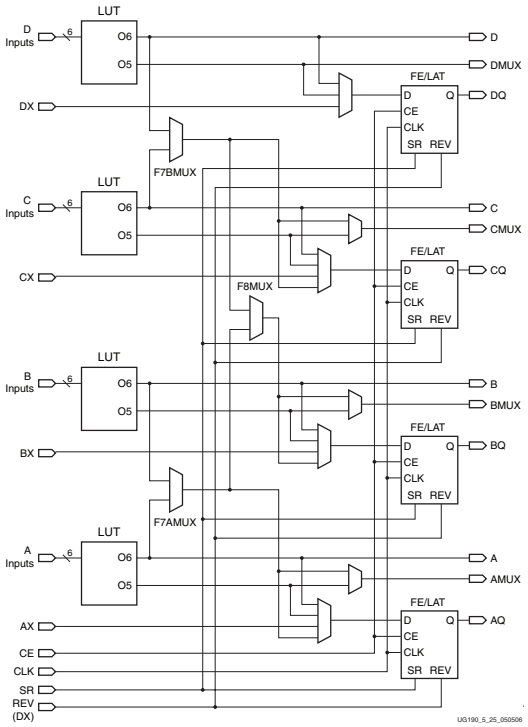


Third multiplexer(F8MUX)

Third input (BX)

Configuring the "n" of an n-LUT ...

Extra muxes to chose LUT option ...

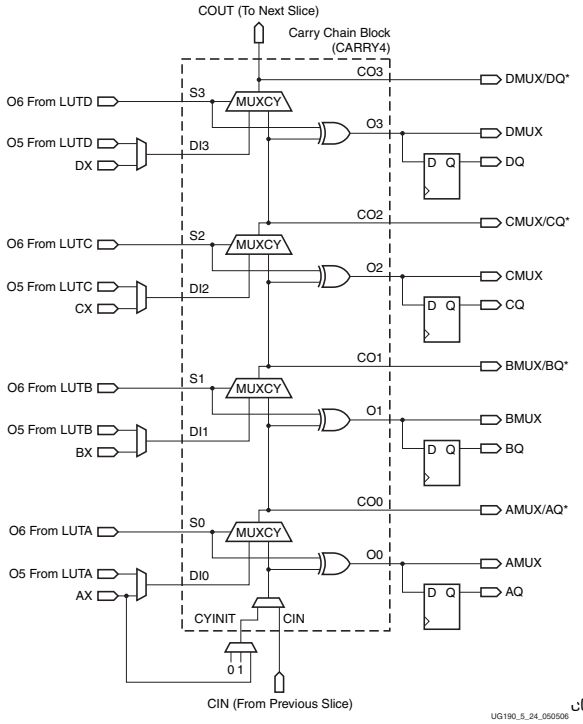


From eight 5-LUTs
... to one 8-LUT.

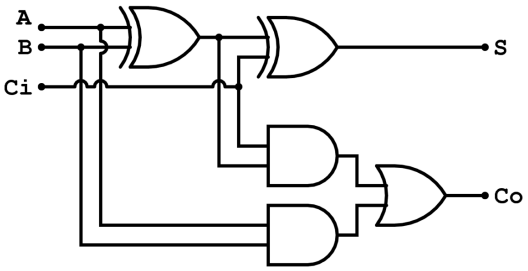
Combinational
or registered outs.

Flip-flops unused by
LUTs can be used
standalone.

Virtex 5 Verical Logic

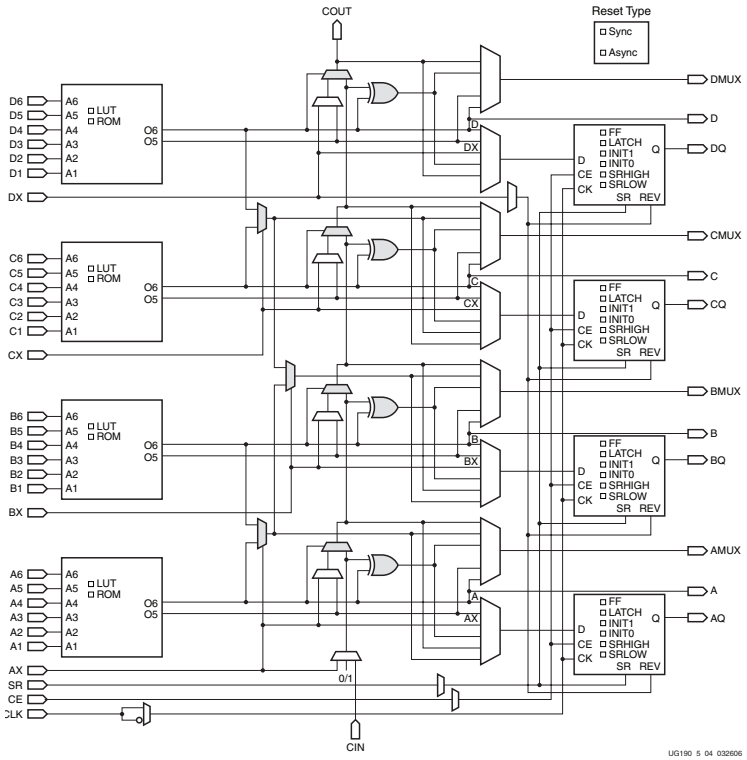


We can map
ripple-carry addition onto
carry-chain block.



The carry-chain block
also useful for speeding
up other adder
structures and counters.

Putting it all together … a SLICEL.

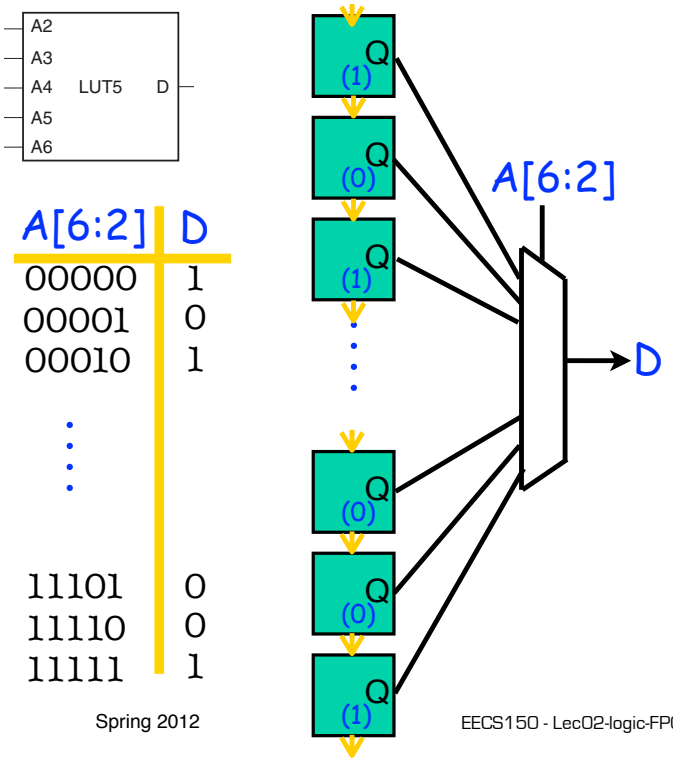


The previous slides explain all SLICEL features.

About 50% of the 17,280 slices in an LX110T are SLICELs.

The other slices are SLICEMs, and have extra features.

Recall: 5-LUT architecture …

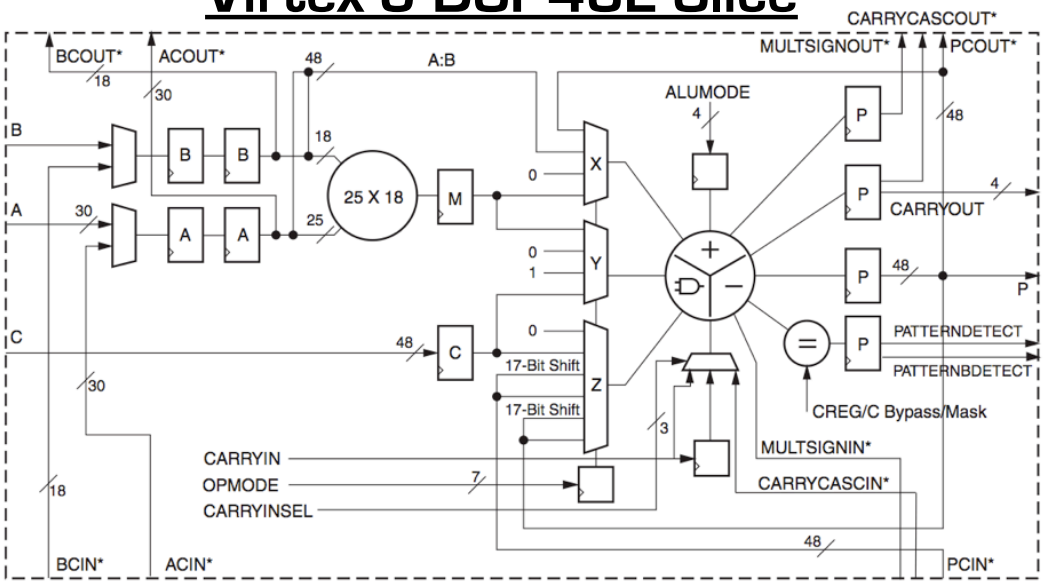


- 32 Latches.
- Configured to 1 or 0.

Some parts of a logic design need many state elements.

SLICEMs replace normal 5-LUTs with circuits that can act like 5-LUTs, but can alternatively use the 32 latches as RAM, ROM, shift registers.

Virtex-5 DSP48E Slice



*These signals are dedicated routing paths internal to the DSP48E column. They are not accessible via fabric routing resources.

UG193_c1_01_032806

Efficient implementation of multiply, add, bit-wise logical.

LX110T has 64 in a single column.

Table 1: **Virtex-5 FPGA Family Members**

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices[2]	Block RAM Blocks			CMTs[4]	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs[5]	Max RocketIO Transceivers[6]		Total I/O Banks[8]	Max User I/O[7]
	Array (Row x Col)	Virtex-5 Slices[1]	Max Distributed RAM (Kb)		18 Kb[3]	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480
XC5VLX85T	120 x 54	12,960	840	48	216	108	3,888	6	N/A	1	4	12	N/A	15	480
XC5VLX110T	160 x 54	17,280	1,120	64	296	148	5,328	6	N/A	1	4	16	N/A	20	680
XC5VLX155T	160 x 76	24,320	1,640	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX220T	160 x 108	34,560	2,280	128	424	212	7,632	6	N/A	1	4	16	N/A	20	680
XC5VLX330T	240 x 108	51,840	3,420	192	648	324	11,664	6	N/A	1	4	24	N/A	27	960
XC5VSX35T	80 x 34	5,440	520	192	168	84	3,024	2	N/A	1	4	8	N/A	12	360
XC5VSX50T	120 x 34	8,160	780	288	264	132	4,752	6	N/A	1	4	12	N/A	15	480
XC5VSX95T	160 x 46	14,720	1,520	640	488	244	8,784	6	N/A	1	4	16	N/A	19	640
XC5VSX240T	240 x 78	37,440	4,200	1,056	1,032	516	18,576	6	N/A	1	4	24	N/A	27	960
XC5VTX150T	200 x 58	23,200	1,500	80	456	228	8,208	6	N/A	1	4	N/A	40	20	680
XC5VTX240T	240 x 78	37,440	2,400	96	648	324	11,664	6	N/A	1	4	N/A	48	20	680
XC5VFX30T	80 x 38	5,120	380	64	136	68	2,448	2	1	1	4	N/A	8	12	360
XC5VFX70T	160 x 38	11,200	820	128	296	148	5,328	6	1	3	4	N/A	16	19	640
XC5VFX100T	160 x 56	16,000	1,240	256	456	228	8,208	6	2	3	4	N/A	16	20	680
XC5VFX130T	200 x 56	20,480	1,580	320	596	298	10,728	6	2	3	6	N/A	20	24	840
XC5VFX200T	240 x 68	30,720	2,280	384	912	456	16,416	6	2	4	8	N/A	24	27	960

To be continued ...

Throughout the semester, we will look at different Virtex-5 features in-depth.

Switch fabric

Block RAM

DSP48 (ALUs)

Clocking

I/O

Serial I/O + PCI