EECS 150 - Components and Design Techniques for Digital Systems

Lec 03 – Field Programmable Gate Arrays
9-4-07

David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley

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

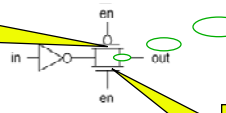
Review

- Building blocks of computer systems
  - ICs (Chips), PCBs, Chassis, Cables & Connectors
- CMOS Transistors
  - Voltage controlled switches
  - Complementary forms (nmos, pmos)
- Logic gates from CMOS transistors
  - Logic gates implement particular boolean functions
    - » N inputs, 1 output
  - Serial and parallel switches
  - Dual structure
  - P-type "pull up" transmit 1
  - N-type
- Complex gates: mux
- Synchronous Sequential Elements [today]
  - D FlipFlops

Question from Thurs

Transmission Gate

- Transmission gates are the way to build "switches" in CMOS.
- Both transistor types are needed:
  - nFET to pass zeros.
  - pFET to pass ones.
- The transmission gate is bi-directional (unlike logic gates and tri-state buffers).
- Functionally it is similar to the tri-state buffer, but do not connect to Vdd and GND, so must be combined logic gates or buffers.



How does nFET behave
• when EN is Hi, S = D = Hi?
• when EN is Hi, S = D = lo?

Together they go Rail-to-Rail ☺

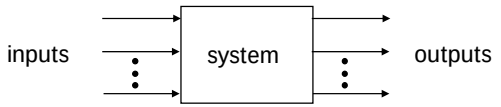
Is it self restoring?

How does nFET behave
• when EN is Hi, S = D = lo?
• when EN is Hi, S = D = Hi?

8/30/2007 © UC Berkeley EECS150-F05 CMOS lec02

Combinational vs. Sequential Digital Circuits

- Simple model of a digital system is a unit with inputs and outputs:



- Combinational means "memory-less"
  - Digital circuit is combinational if its output values only depend on its inputs

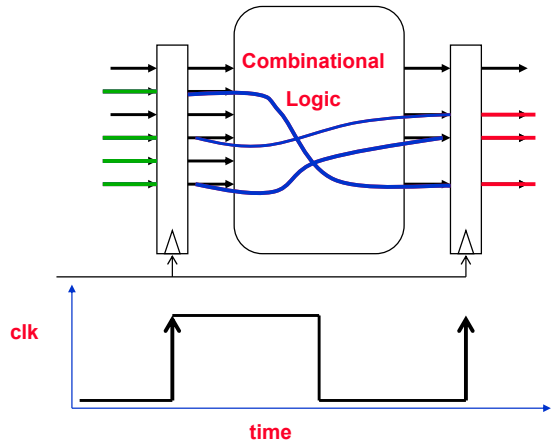
Sequential logic

- Sequential systems
  - Exhibit behaviors (output values) that depend on current *as well as* previous inputs
- All real circuits are sequential
  - Outputs do not change instantaneously after an input change
  - Why not, and why is it then sequential?
- Fundamental abstraction of digital design is to reason (mostly) about steady-state behaviors
  - Examine outputs only after sufficient time has elapsed for the system to make its required changes and settle down

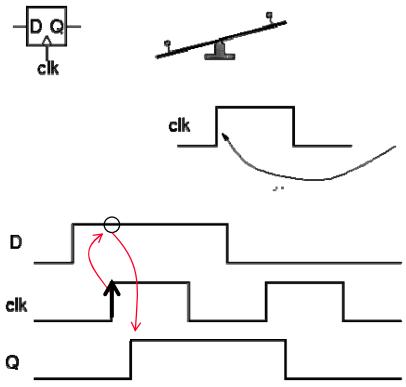
Synchronous sequential digital systems

- Combinational circuit outputs depend *only* on current inputs
  - After sufficient time has elapsed
- Sequential circuits have *memory*
  - Even after waiting for transient activity to finish
- Steady-state abstraction: most designers use it when constructing sequential circuits:
  - Memory of system is its state
  - Changes in system state only allowed at specific times controlled by an external periodic signal (the *clock*)
  - Clock period is elapsed time between state changes sufficiently long so that system reaches steady-state before next state change at end of period

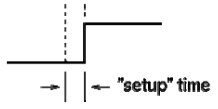
Recall: What makes Digital Systems tick?



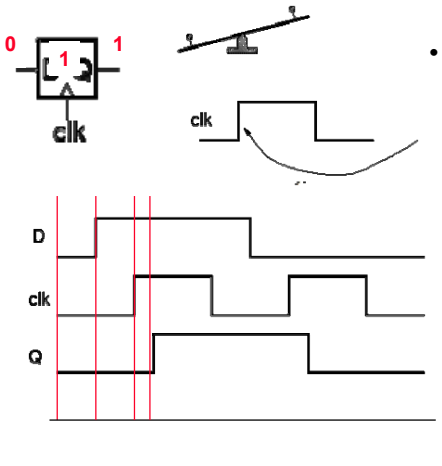
D-type edge-triggered flip-flop



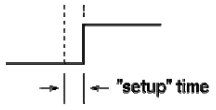
- The edge of the clock is used to *sample* the "D" input & send it to "Q" (positive edge triggering).
  - At all other times the output Q is independent of the input D (just stores previously sampled value).
  - The input must be stable for a short time before the clock edge.



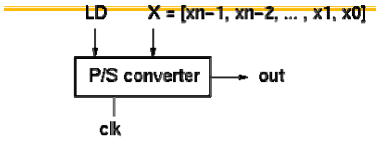
D-type edge-triggered flip-flop



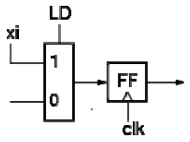
- The edge of the clock is used to *sample* the "D" input & send it to "Q" (positive edge triggering).
  - At all other times the output Q is independent of the input D (just stores previously sampled value).
  - The input must be stable for a short time before the clock edge.

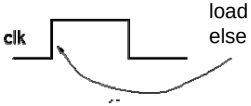


Parallel to Serial Converter Example



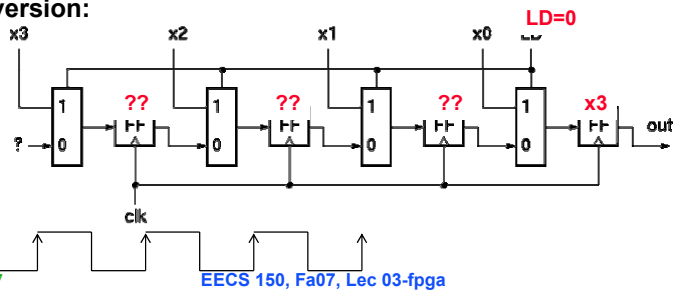
- Operation:
  - cycle 1: load x, output x_0
  - cycle i: output x_i
- Each stage:



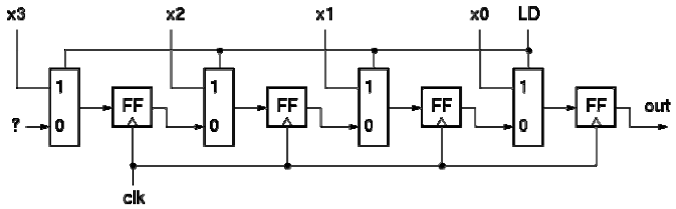


if LD=1
load FF from x_i
else from previous stage.

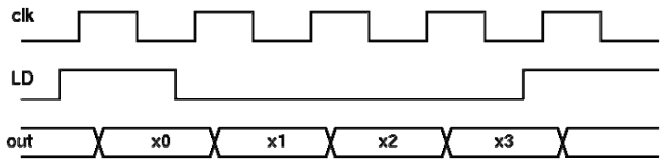
- 4-bit version:



Parallel to Serial Converter Example

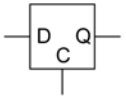


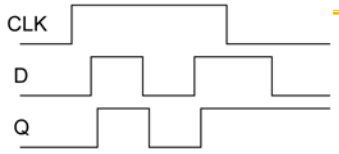
- timing:

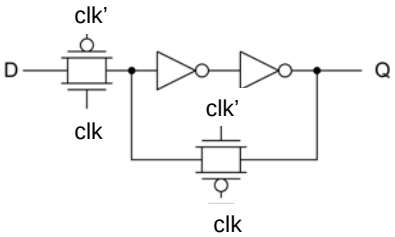


Transistor-level Logic Circuits - Latch

- Positive Level-sensitive latch



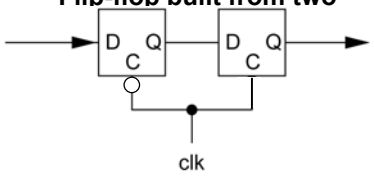




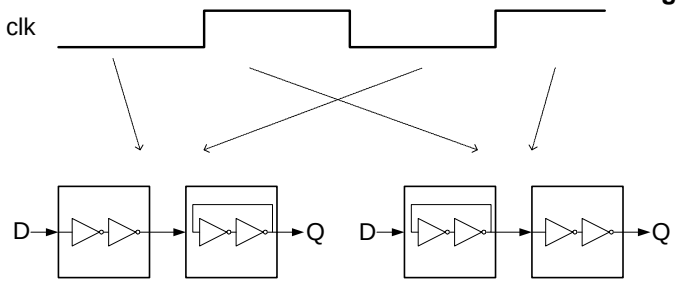
D FlipFlop

Positive Edge-triggered Flip-flop

- Flip-flop built from two



- When clk low, left latch acts as feedthrough, and Q is stored value of right latch.
- When clk high left latch stores values and right latch acts as feedthrough.



Summary: Representation of digital designs

- Physical devices (transistors, relays)
- ***Switches***
- ***Truth tables***
- ***Boolean algebra***
- ***Gates***
- Waveforms
- Finite state behavior
- Register-transfer behavior
- Concurrent abstract specifications

scope of CS 150

more depth than 61C

focus on building systems

Outline

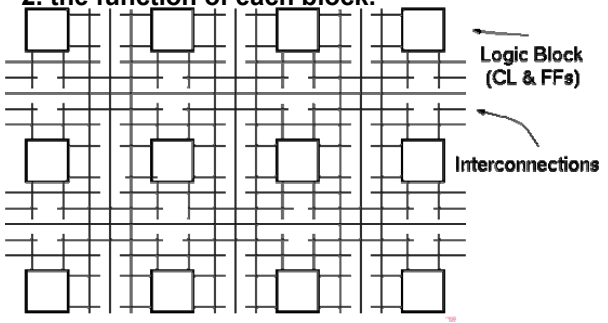
- Review
- What are FPGAs?
- Why use FPGAs (a short history lesson).
- Canonical Forms => Programmable Logic
- FPGA variations
- Internal logic blocks.
- Designing with FPGAs.
- Specifics of Xilinx Virtex-E series.

Today's reading

- Katz: 9.4 pp 428-447 (especially 9.4.4)
- XILINX Virtex-E FPGA data sheet (first 10 pages)

FPGA Overview

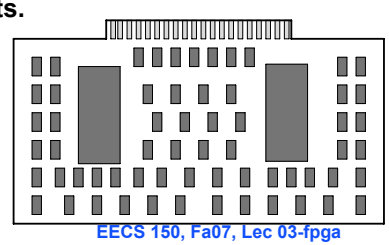
- Basic idea: two-dimensional array of logic blocks and flip-flops with a means for the user to configure:
  1. the interconnection between the logic blocks,
  2. the function of each block.



Simplified version of FPGA internal architecture:

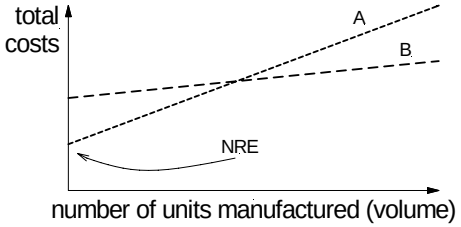
Why FPGAs?

- By the early 1980's most of the logic circuits in typical systems where absorbed by a handful of standard large scale integrated circuits (LSI).
  - Microprocessors, bus/IO controllers, system timers, ...
- Every system still had the need for random "glue logic" to help connect the large ICs:
  - generating global control signals (for resets etc.)
  - data formatting (serial to parallel, multiplexing, etc.)
- Systems had a few LSI components and lots of small low density SSI (small scale IC) and MSI (medium scale IC) components.



Why FPGAs?

- Custom ICs sometimes designed to replace the large amount of glue logic:
  - reduced system complexity and manufacturing cost, improved performance.
  - However, custom ICs are very expensive to develop, and delay introduction of product to market (time to market) because of increased design time.
- Note: need to worry about two kinds of costs:
  1. cost of development, sometimes called non-recurring engineering (NRE)
  2. cost of manufacture
  - A tradeoff usually exists between NRE cost and manufacturing costs

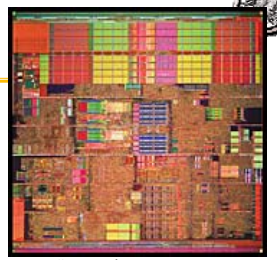


Why FPGAs?

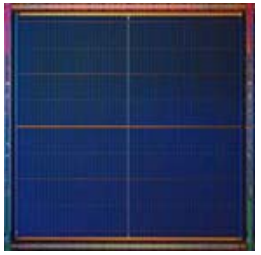
- Custom IC approach viable for products that are …
  - very high volume (where NRE could be amortized),
  - not time-to-market sensitive.
- FPGAs introduced as an alternative to custom ICs for implementing glue logic:
  - improved density relative to discrete SSI/MSI components (within around 10x of custom ICs)
  - with the aid of computer aided design (CAD) tools circuits could be implemented in a short amount of time (no physical layout process, no mask making, no IC manufacturing), relative to ASICs.
    - » lowers NREs
    - » shortens TTM
- Because of Moore's law the density (gates/area) of FPGAs continued to grow through the 80's and 90's to the point where major data processing functions can be implemented on a single FPGA.

Programmable Logic

- Regular logic
  - Programmable Logic Arrays
  - Multiplexers/Decoders
  - ROMs
- Field Programmable Gate Arrays
  - Xilinx Vertex



"Random Logic"
Full Custom Design



"Regular Logic"
Structured Design

Canonical Forms

- Truth table is the unique signature of a Boolean function
- Many alternative gate realizations may have the same truth table
- Canonical forms
  - Standard forms for a Boolean expression
  - Provides a unique algebraic signature

Sum-of-Products Canonical Forms

- Also known as disjunctive normal form
- Also known as minterm expansion

F = 001 011 101 110 111

F = A'B'C + A'BC + AB'C + ABC' + ABC

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

F' = A'B'C' + A'BC' + AB'C'

Sum-of-Products Canonical Form (cont'd)

- Product term (or minterm)
  - ANDed product of literals – input combination for which output is true
  - Each variable appears exactly once, in true or inverted form (but not both)

A	B	C	minterms
0	0	0	A'B'C' m0
0	0	1	A'B'C m1
0	1	0	A'BC' m2
0	1	1	A'BC m3
1	0	0	AB'C' m4
1	0	1	AB'C m5
1	1	0	ABC' m6
1	1	1	ABC m7

short-hand notation for minterms of 3 variables

F in canonical form:

F(A, B, C) = Σm(1,3,5,6,7)
= m1 + m3 + m5 + m6 + m7
= A'B'C + A'BC + AB'C + ABC' + ABC

canonical form ≠ minimal form

F(A, B, C) = A'B'C + A'BC + AB'C + ABC + ABC'
= (A'B' + A'B + AB' + AB)C + ABC'
= ((A' + A)(B' + B))C + ABC'
= C + ABC'
= ABC' + C
= AB + C

Product-of-Sums Canonical Form

- Also known as conjunctive normal form
- Also known as maxterm expansion

F = 000 010 100

F = (A + B + C) (A + B' + C) (A' + B + C)

A	B	C	F	F'
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	0

F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')

Product-of-Sums Canonical Form (cont'd)

- **Sum term (or maxterm)**
  - **ORed sum of literals – input combination for which output is false**
  - **Each variable appears exactly once, in true or inverted form (but not both)**

A	B	C	maxterms	
0	0	0	A+B+C	M0
0	0	1	A+B+C'	M1
0	1	0	A+B'+C	M2
0	1	1	A+B'+C'	M3
1	0	0	A'+B+C	M4
1	0	1	A'+B+C'	M5
1	1	0	A'+B'+C	M6
1	1	1	A'+B'+C'	M7

short-hand notation for maxterms of 3 variables

F in canonical form:

$F(A, B, C) = \Pi M(0,2,4)$
$= M0 \cdot M2 \cdot M4$
$= (A + B + C)(A + B' + C)(A' + B + C)$

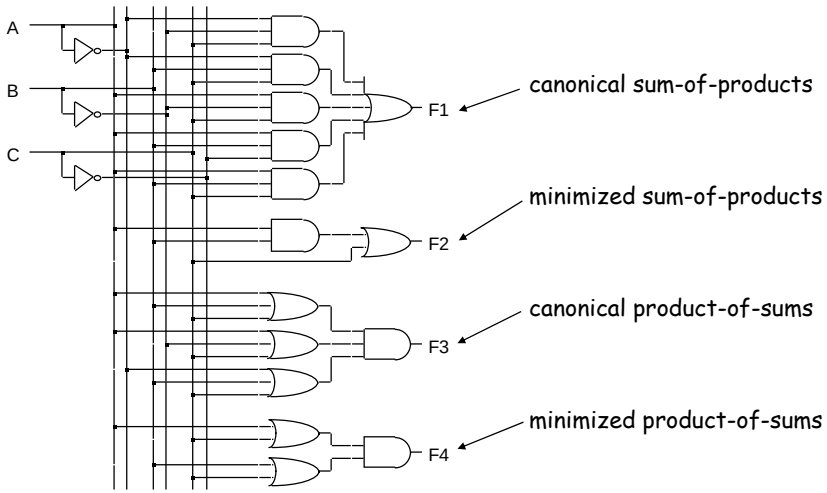
canonical form ≠ minimal form

$F(A, B, C) = (A + B + C)(A + B' + C)(A' + B + C)$
$= (A + B + C)(A + B' + C)$
$(A + B + C)(A' + B + C)$
$= (A + C)(B + C)$

S-o-P, P-o-S, and deMorgan's Theorem

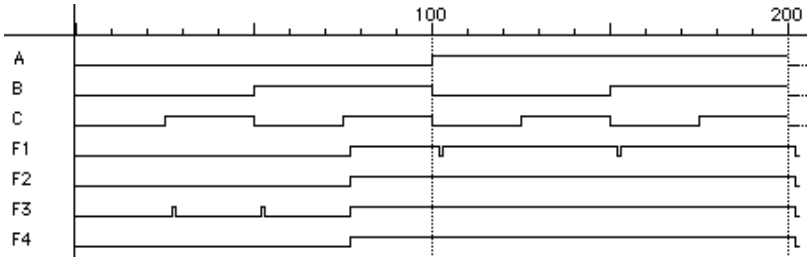
- **Sum-of-products**
  - **F' = A'B'C' + A'BC' + AB'C'**
- **Apply de Morgan's**
  - **(F')' = (A'B'C' + A'BC' + AB'C')'**
  - **F = (A + B + C) (A + B' + C) (A' + B + C)**
- **Product-of-sums**
  - **F' = (A + B + C') (A + B' + C') (A' + B + C') (A' + B' + C) (A' + B' + C')**
- **Apply de Morgan's**
  - **(F')' = ((A + B + C')(A + B' + C')(A' + B + C')(A' + B' + C)(A' + B' + C'))'**
  - **F = A'B'C + A'BC + AB'C + ABC' + ABC**

Four Alternative Two-level Implementations of F = AB + C



Waveforms for the Four Alternatives

- **Waveforms are essentially identical**
  - **Except for timing hazards (glitches)**
  - **Delays almost identical (modeled as a delay per level, not type of gate or number of inputs to gate)**



Mapping Between Canonical Forms

- Minterm to maxterm conversion
  - Use maxterms whose indices do not appear in minterm expansion
  - e.g., F(A,B,C) = Σm(1,3,5,6,7) = ΠM(0,2,4)
- Maxterm to minterm conversion
  - Use minterms whose indices do not appear in maxterm expansion
  - e.g., F(A,B,C) = ΠM(0,2,4) = Σm(1,3,5,6,7)
- Minterm expansion of F to minterm expansion of F'
  - Use minterms whose indices do not appear
  - e.g., F(A,B,C) = Σm(1,3,5,6,7) F'(A,B,C) = Σm(0,2,4)
- Maxterm expansion of F to maxterm expansion of F'
  - Use maxterms whose indices do not appear
  - e.g., F(A,B,C) = ΠM(0,2,4) F'(A,B,C) = ΠM(1,3,5,6,7)

Incompletely Specified Functions

- Example: binary coded decimal increment by 1
  - BCD digits encode decimal digits 0 – 9 in bit patterns 0000 – 1001

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

off-set of W

on-set of W

don't care (DC) set of W

these inputs patterns should never be encountered in practice - "don't care" about associated output values, can be exploited in minimization

Notation for Incompletely Specified Functions

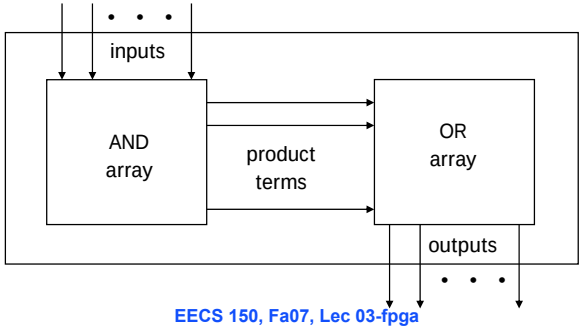
- Don't cares and canonical forms
  - So far, only represented on-set
  - Also represent don't-care-set
  - Need two of the three sets (on-set, off-set, dc-set)
- Canonical representations of the BCD increment by 1 function:
  - Z = m0 + m2 + m4 + m6 + m8 + d10 + d11 + d12 + d13 + d14 + d15
  - Z = Σ [m(0,2,4,6,8) + d(10,11,12,13,14,15)]
  - Z = M1 • M3 • M5 • M7 • M9 • D10 • D11 • D12 • D13 • D14 • D15
  - Z = Π [M(1,3,5,7,9) • D(10,11,12,13,14,15)]

Simplification of Two-level Combinational Logic

- Finding a minimal sum of products or product of sums realization
  - Exploit don't care information in the process
- Algebraic simplification
  - Not an algorithmic/systematic procedure
  - How do you know when the minimum realization has been found?
- Computer-aided design tools
  - Precise solutions require very long computation times, especially for functions with many inputs (> 10)
  - Heuristic methods employed – "educated guesses" to reduce amount of computation and yield good if not best solutions
- Hand methods still relevant
  - Understand automatic tools and their strengths and weaknesses
  - Ability to check results (on small examples)

Programmable Logic Arrays (PLAs)

- **Pre-fabricated building block of many AND/OR gates**
  - **Actually NOR or NAND**
  - **"Personalized" by making or breaking connections among gates**
  - **Programmable array block diagram for sum of products form**



Enabling Concept

- **Shared product terms among outputs**

example:

F0 = A + B' C'
F1 = A C' + A B
F2 = B' C' + A B
F3 = B' C + A

personality matrix

product term	inputs A	B	C	outputs F0	F1	F2	F3
AB	1	1	–	0	1	1	0
B'C	–	0	1	0	0	0	1
AC'	1	–	0	0	1	0	0
B'C'	–	0	0	1	0	1	0
A	1	–	–	1	0	0	1

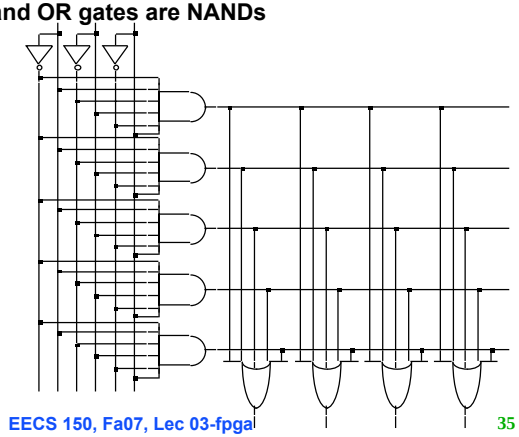
reuse of terms

input side:
1 = uncomplemented in term
0 = complemented in term
– = does not participate

output side:
1 = term connected to output
0 = no connection to output

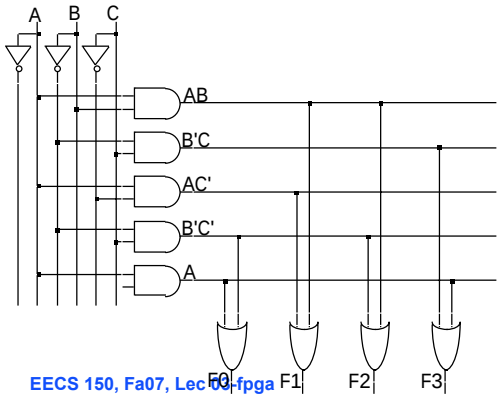
Before Programming

- **All possible connections available before "programming"**
  - **In reality, all AND and OR gates are NANDs**



After Programming

- **Unwanted connections are "blown"**
  - **Fuse (normally connected, break unwanted ones)**
  - **Anti-fuse (normally disconnected, make wanted connections)**



Announcements

- Homework #1 due Friday 2pm. (#2 out thurs)
- Please do the reading (the earlier the better).
- Attend discussions!
  - Held this week. Propose Fridays 11-12 and 1-2, Take vote
- Homework is an important part of the class:
  - It goes beyond what is covered in class.
  - High correlation to exam questions.
  - Work on it seriously.
  - We'll try to post it early.
  - Discussion is a good place to get hints about homework.
- Unlike some of our lower division classes we will not necessarily tell you everything you need to know. Some of it will come from readings and homework.

Why FPGAs?

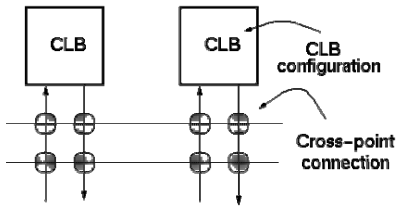
- Much more general form of programmable logic
- FPGAs continue to compete with custom ICs for special processing functions (and glue logic) but now also compete with microprocessors in dedicated and embedded applications.
  - Performance advantage over microprocessors because circuits can be customized for the task at hand. Microprocessors must provide special functions in software (many cycles).
- Summary:

	performance	NREs	Unit cost	TTM
↑	ASIC	ASIC	FPGA	ASIC
	FPGA	FPGA	MICRO	FPGA
	MICRO	MICRO	ASIC	MICRO

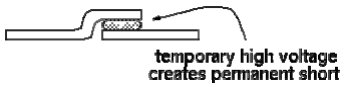
ASIC = custom IC, MICRO = microprocessor + SW

FPGA Variations

- Families of FPGA's differ in:
  - physical means of implementing user programmability,
  - arrangement of interconnection wires, and
  - the basic functionality of the logic blocks.
- Most significant difference is in the method for providing flexible blocks and connections:



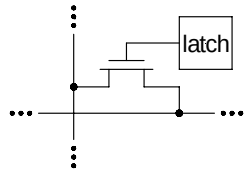
- Anti-fuse based (ex: Actel)



+ Non-volatile, relatively small
– fixed (non-reprogrammable)

User Programmability

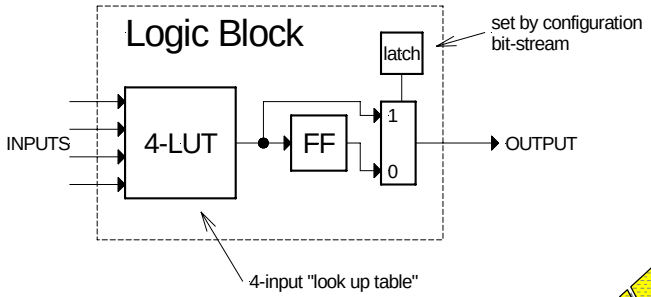
- Latch-based (Xilinx, Altera, …)



+ reconfigurable
– volatile
– relatively large.

- Latches are used to:
  1. 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" can be loaded under user control:
  - All latches are strung together in a shift chain:

Idealized FPGA Logic Block



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

what's a LUT?

Boolean Functions: 1 variable

A	False
0	0
1	0

A	$\overline{A}$
0	1
1	0

A	A
0	0
1	1

A	TRUE
0	1
1	1

A	False	$\overline{A}$	A	True
0	0	1	0	1
1	0	0	1	1

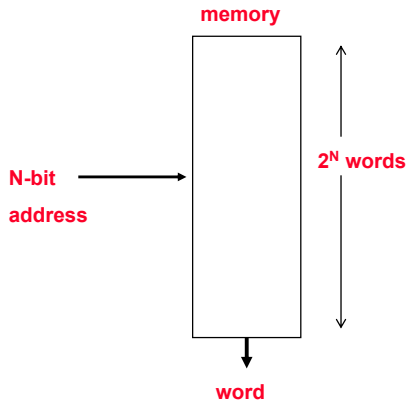
- **What are the possible boolean functions of two variable?**

Interactive Quiz: Boolean Functions of 2 variables?

A	B	False ??		$\overline{A}B$		$A\overline{B}$		$\overline{A}\overline{B}$			True
0	0	0	1	0	1	0	1	0	1	…	1
0	1	0	0	1	1	0	0	1	1	…	1
1	0	0	0	0	0	1	1	1	1	…	1
1	1	0	0	0	0	0	0	0	0	…	1

- **What are the possible boolean functions of 3, 4 variables?**

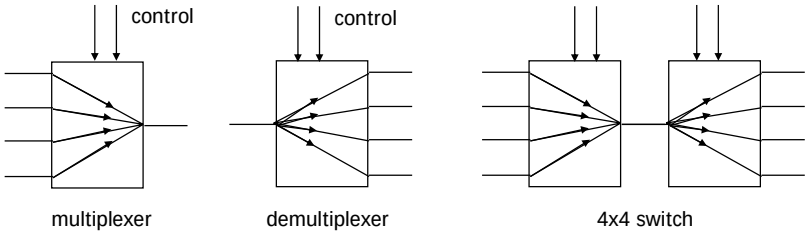
How could you build a generic boolean logic circuit?



- **1-bit memory to hold boolean value**
- **Address is vector of boolean input values**
- **Contents encode a boolean function**
- **Read out logical value (col) for associated row**

Recall: Multiplexer/Demultiplexer

- *Multiplexer:* route one of many inputs to a single output
- *Demultiplexer:* route single input to one of many outputs



Multiplexers/Selectors – a logical function

- Multiplexers/Selectors: general concept
  - 2^n data inputs, n control inputs (called "selects"), 1 output
  - Used to connect 2^n points to a single point
  - Control signal pattern forms binary index of input connected to output

$Z = A' I_0 + A I_1$

A	Z
0	I_0
1	I_1

functional form

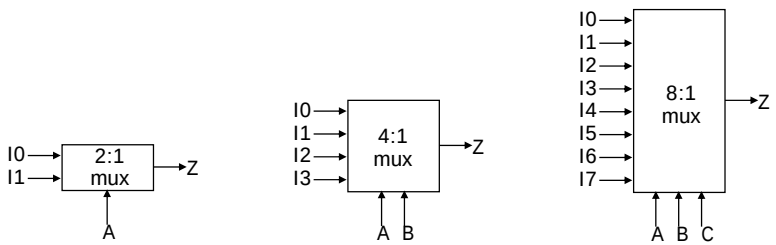
logical form

I_1	I_0	A	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

two alternative forms for a 2:1 Mux truth table

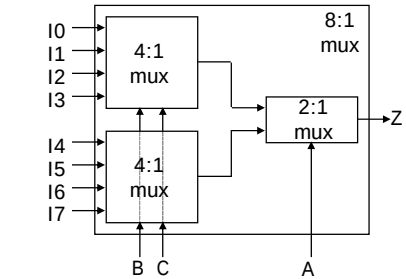
Multiplexers/Selectors: to implement logic

- 2:1 mux: Z = A' I0 + A I1
- 4:1 mux: Z = A' B' I0 + A' B I1 + A B' I2 + A B I3
- 8:1 mux: Z = A'B'C'I0 + A'B'CI1 + A'BC'I2 + A'BCI3 + AB'C'I4 + AB'CI5 + ABC'I6 + ABCI7
- In general, $Z = \sum_{k=0}^{2^n-1} (m_k I_k)$
  - in minterm shorthand form for a 2^n:1 Mux



Cascading Multiplexers

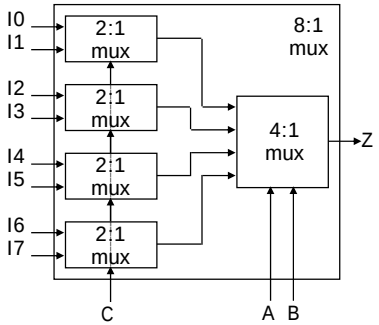
- Large multiplexers implemented by cascading smaller ones



control signals B and C simultaneously choose one of I0, I1, I2, I3 and one of I4, I5, I6, I7

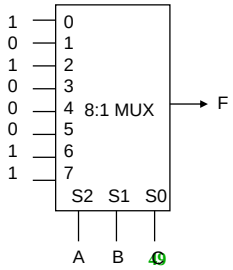
control signal A chooses which of the upper or lower mux's output to gate to Z

alternative implementation



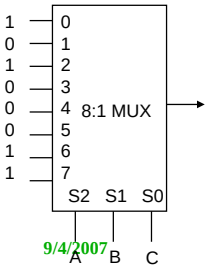
Multiplexers as Lookup Tables (LUTs)

- 2^n:1 multiplexer implements any function of n variables
  - With the variables used as control inputs and
  - Data inputs tied to 0 or 1
  - In essence, *a lookup table*
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)



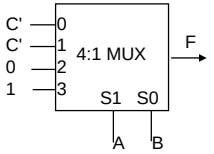
Multiplexers as LUTs (cont'd)

- 2^{n-1}:1 mux can implement any function of n variables
  - With n-1 variables used as control inputs and
  - Data inputs tied to the last variable or its complement
- Example:
  - F(A,B,C) = m0 + m2 + m6 + m7
    = A'B'C' + A'BC' + ABC' + ABC
    = A'B'(C') + A'B(C') + AB'(0) + AB(1)



F

A	B	C	F	
0	0	0	1	C'
0	0	1	0	
0	1	0	1	C'
0	1	1	0	
1	0	0	0	0
1	0	1	0	
1	1	0	1	1
1	1	1	1	



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.

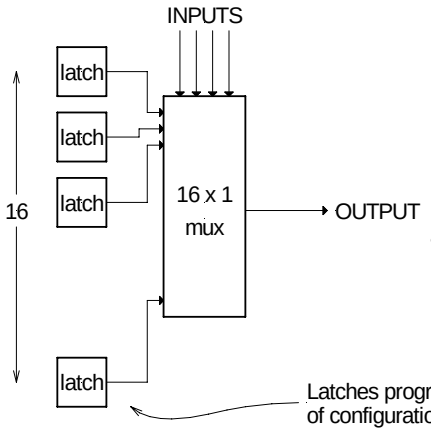
How many of these are there?

How many functions of n inputs?

Example: 4-lut

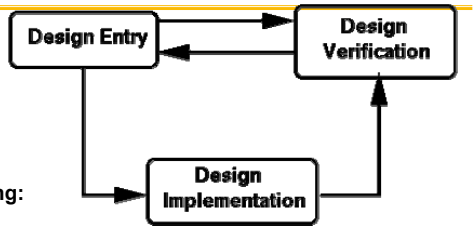
INPUTS	
0000	F(0,0,0,0) ← store in 1st latch
0001	F(0,0,0,1) ← store in 2nd latch
0010	F(0,0,1,0) ←
0011	F(0,0,1,1) ←
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

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!

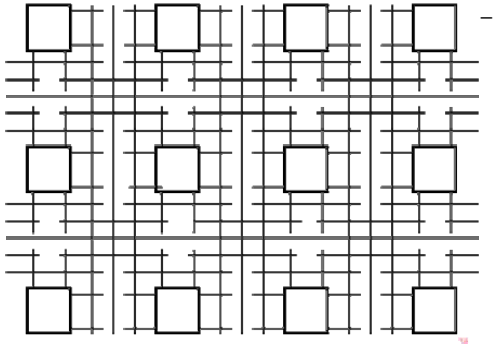
FPGA Generic Design Flow



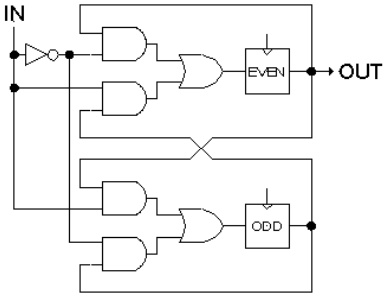
- **Design Entry:**
  - Create your design files using:
    - » schematic editor or
    - » hardware description language (Verilog, VHDL)
- **Design "implementation" on FPGA:**
  - *Partition, place, and route* to create bit-stream file
- **Design verification:**
  - Use Simulator to check function,
  - other software determines max clock frequency.
  - Load onto FPGA device (cable connects PC to development board)
    - » check operation at full speed in real environment.

Example Partition, Placement, and Route

- **Idealized FPGA structure:**



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



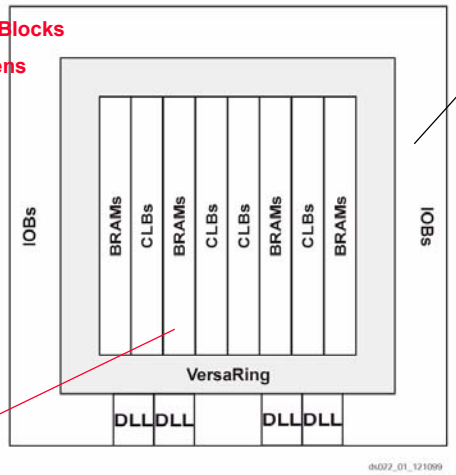
Circuit combinational logic must be "covered" by 4-input 1-output "gates".

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

Placement in general attempts to minimize wiring.

Xilinx Virtex-E Floorplan

Configurable Logic Blocks
- 4-input function gens
- buffers
- flipflop



Input/Output Blocks
- combinational, latch, and flipflop output
- sampled inputs

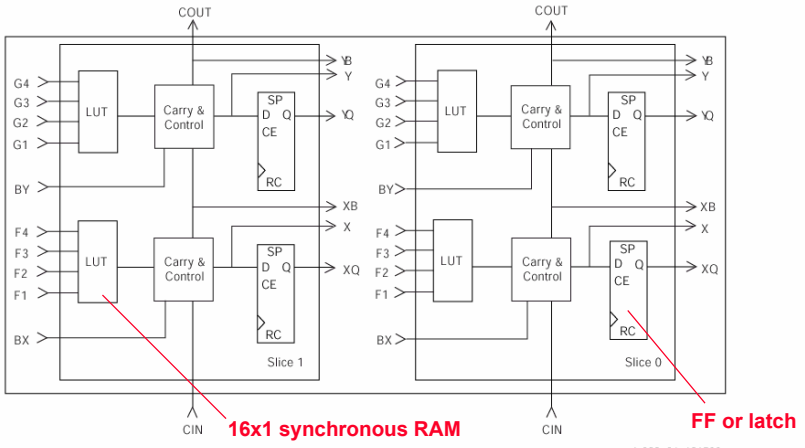
Block RAM
- 4096 bits each
- every 12 CLB columns

Virtex-E Configurable Logic Block (CLB)

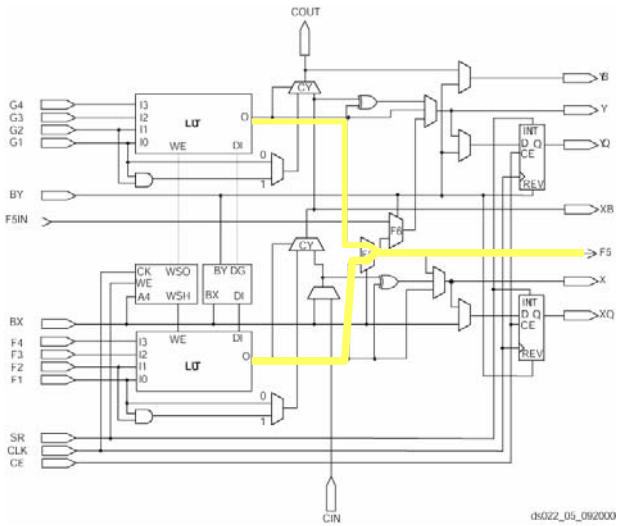
CLB = 4 logic cells (LC) in two slices

LC: 4-input function generator, carry logic, storage ele't

80 x 120 CLB array on 2000E

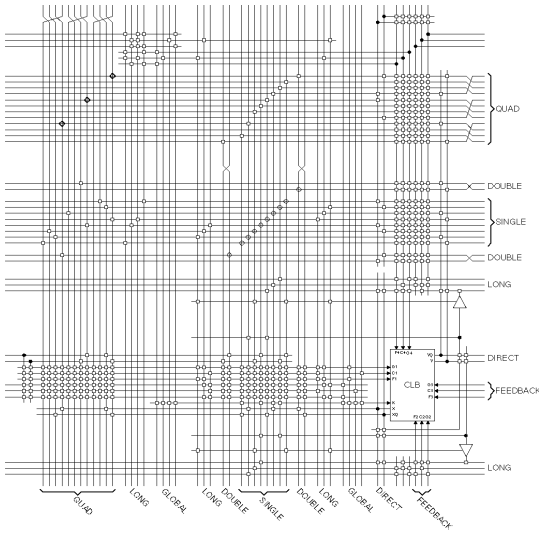


Details of Virtex-E Slice

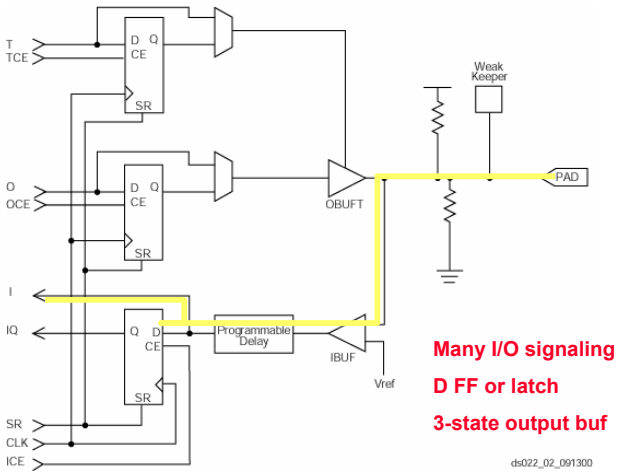


- LUT
- •4-input fun
- •16x1 sram
- •32x1 or 16x2 in slice
- •16 bit shift register
- Storage element
- • D flipflip
- • latch
- Combinational outputs
- 5 and 6 input functions
- Carry chain
- • arithmetic along row or col

Xilinx FPGAs (interconnect detail)



Virtex-E Input/Output block (IOB) detail



- Many I/O signaling stds
- D FF or latch
- 3-state output buf

Virtex-E Family of Parts

Table 1: **Virtex-E Field-Programmable Gate Array Family Members**

Device	System Gates	Logic Gates	CLB Array	Logic Cells	Differential I/O Pairs	User I/O	BlockRAM Bits	Distributed RAM Bits
XCV50E	71,693	20,736	16 x 24	1,728	83	176	65,536	24,576
XCV100E	128,236	32,400	20 x 30	2,700	83	196	81,920	38,400
XCV200E	306,393	63,504	28 x 42	5,292	119	284	114,688	75,264
XCV300E	411,955	82,944	32 x 48	6,912	137	316	131,072	98,304
XCV400E	569,952	129,600	40 x 60	10,800	183	404	163,840	153,600
XCV600E	985,882	186,624	48 x 72	15,552	247	512	294,912	221,184
XCV1000E	1,569,178	331,776	64 x 96	27,648	281	660	393,216	393,216
XCV1600E	2,188,742	419,904	72 x 108	34,992	344	724	589,824	497,664
XCV2000E	2,541,952	518,400	80 x 120	43,200	344	804	655,360	614,400
XCV2600E	3,263,755	685,584	92 x 138	57,132	344	804	753,664	812,544
XCV3200E	4,074,387	876,096	104 x 156	73,008	344	804	851,968	1,038,336

Xilinx FPGAs

- How they differ from idealized array:
  - In addition to their use as general logic "gates", LUTs can alternatively be used as general purpose RAM.
    - » Each 4-lut can become a 16x1-bit RAM array.
  - Special circuitry to speed up "ripple carry" in adders and counters.
    - » Therefore adders assembled by the CAD tools operate much faster than adders built from gates and luts alone.
  - Many more wires, including tri-state capabilities.

Summary

- Logic design process influenced by available technology AND economic drivers
  - Volume, Time to Market, Costs, Power
- Fundamental understanding of digital design techniques carry over
  - Specifics on design trade-offs and implementation differ
- FPGA offer a valuable new sweet spot
  - Low TTM, medium cost, tremendous flexibility
- Fundamentally tied to powerful CAD tools
- Build everything (simple or complex) from one set of building blocks
  - LUTs + FF + routing + storage + IOs