CS152
Computer Architecture and Engineering
Lecture 3

Logic Design, Technology, and Delay

January 28, 2004

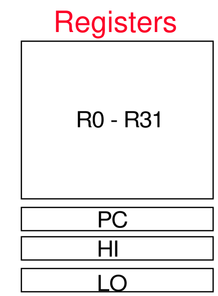
John Kubiatowicz (www.cs.berkeley.edu/~kubitron)

lecture slides: http://www-inst.eecs.berkeley.edu/~cs152/

Review:MIPS R3000 Instruction Set Architecture

° Register Set
  • 32 general 32-bit registers
  • Register zero ($R0) always zero
  • Hi/Lo for multiplication/division

° Instruction Categories
  • Load/Store
  • Computational
    - Integer/Floating point
  • Jump and Branch
  • Memory Management
  • Special



° 3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	sa	funct

OP	rs	rt	immediate

OP	jump target

The Design Process

"To Design Is To Represent"

Design activity yields description/representation of an object

-- Traditional craftsman does not distinguish between the conceptualization and the artifact

-- Separation comes about because of *complexity*

-- The concept is captured in one or more *representation languages* *VERILOG, Schematics, etc.*

-- This process IS design

Design Begins With Requirements

-- Functional Capabilities: what it will do

-- Performance Characteristics: Speed, Power, Area, Cost, . . .

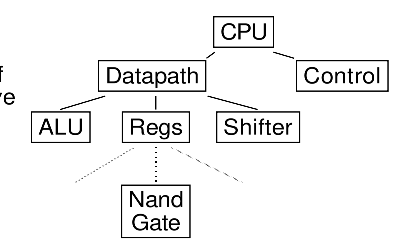
Design Process (cont.)

Design Finishes As Assembly

-- Design understood in terms of components and how they have been assembled

-- Top Down *decomposition* of complex functions (behaviors) into more primitive functions

-- bottom-up *composition* of primitive building blocks into more complex assemblies



Design is a "creative process," not a simple method

Design Refinement

Informal System Requirement

Initial Specification

Intermediate Specification

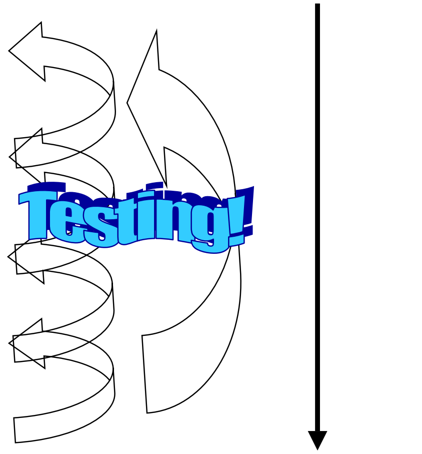
Final Architectural Description

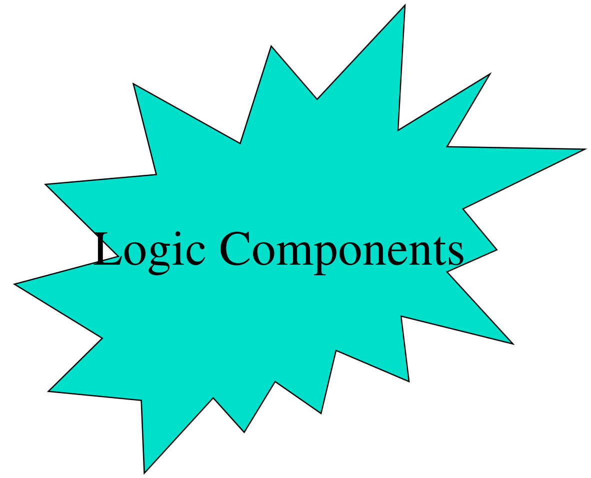
Intermediate Specification of Implementation

Final Internal Specification

Physical Implementation

refinement
increasing level of detail

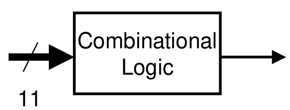




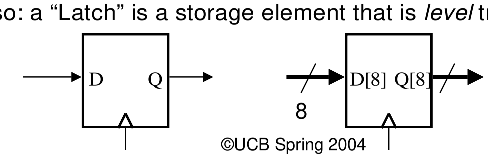
Elements of the design zoo

- Wires: Carry signals from one point to another
  - Single bit (no size label) or multi-bit bus (size label)

- Combinational Logic: Like function evaluation
  - Data goes in, Results come out after some propagation delay



- Flip-Flops: Storage Elements
  - After a clock edge, input copied to output
  - Otherwise, the flip-flop holds its value
  - Also: a "Latch" is a storage element that is *level* triggered



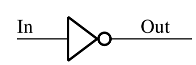
Basic Combinational Elements+DeMorgan Equivalence

Wire

Out = In

In	Out
0	0
1	1

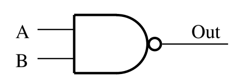
Inverter



$Out = \overline{In}$

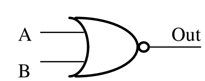
In	Out
0	1
1	0

NAND Gate



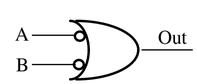
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

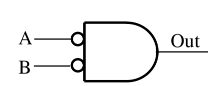


A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

$Out = \overline{A \bullet B} = \overline{A} + \overline{B}$ ← DeMorgan's Theorem → $Out = \overline{A + B} = \overline{A} \bullet \overline{B}$

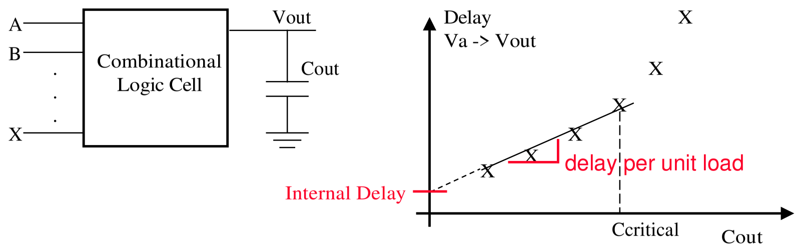


A	B	$\overline{A}$	$\overline{B}$	Out
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0



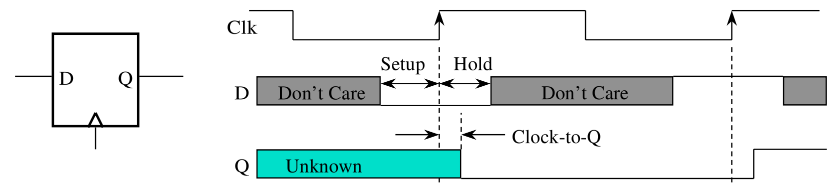
A	B	$\overline{A}$	$\overline{B}$	Out
0	0	1	1	1
0	1	1	0	0
1	0	0	1	0
1	1	0	0	0

General C/L Cell Delay Model



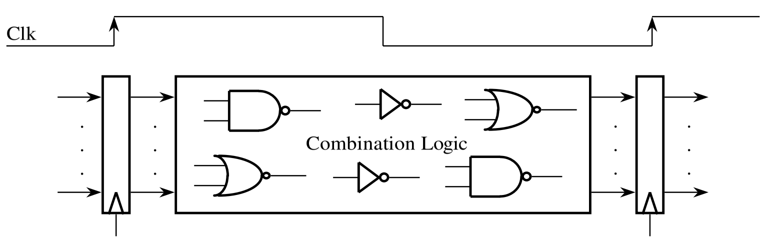
- Combinational Cell (symbol) is fully specified by:
  - functional (input -> output) behavior
    - truth-table, logic equation, VHDL
  - Input load factor of each input
  - Propagation delay from each input to each output for each transition
    - $T_{HL}(A, o)$ = Fixed Internal Delay + Load-dependent-delay x *load*
- Linear model composes

Storage Element's Timing Model



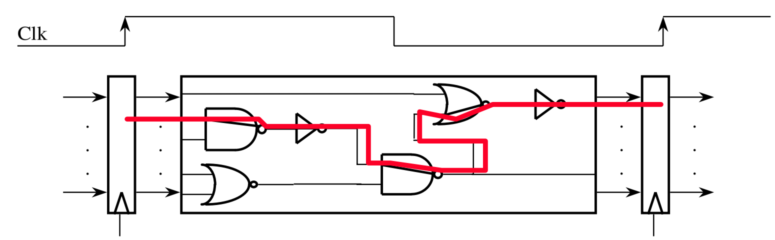
- Setup Time: Input must be stable BEFORE trigger clock edge
- Hold Time: Input must REMAIN stable after trigger clock edge
- Clock-to-Q time:
  - Output cannot change instantaneously at the trigger clock edge
  - Similar to delay in logic gates, two components:
    - Internal Clock-to-Q
    - Load dependent Clock-to-Q

Clocking Methodology



- All storage elements are clocked by the same clock edge
- The combination logic blocks:
  - Inputs are updated at each clock tick
  - All outputs MUST be stable before the next clock tick

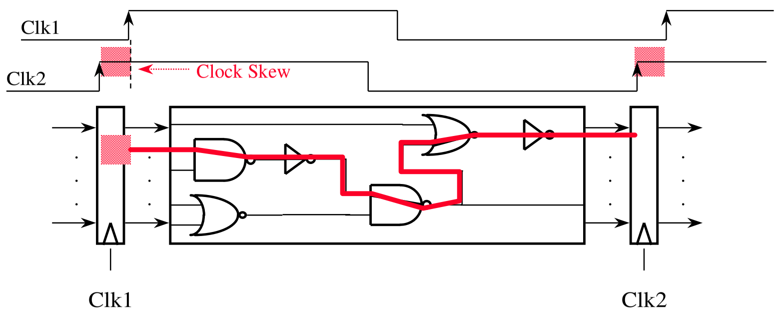
Critical Path & Cycle Time



- Critical path: the slowest path between any two storage devices
- Cycle time is a function of the critical path
- must be greater than:

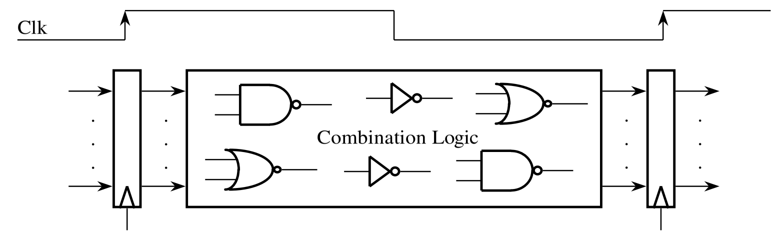
  Clock-to-Q + Longest Path through Combination Logic + Setup

Clock Skew's Effect on Cycle Time



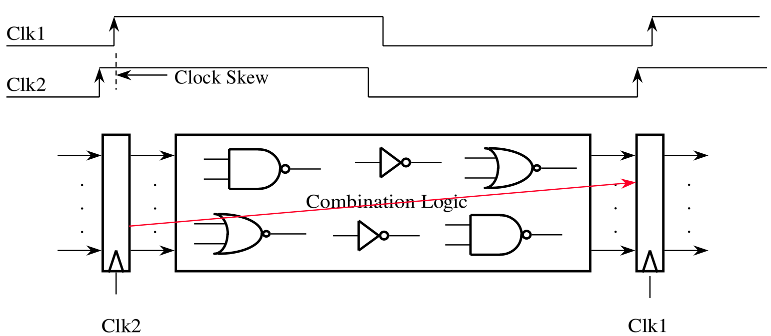
- The worst case scenario for cycle time consideration:
  - The input register sees CLK1
  - The output register sees CLK2
- Cycle Time - Clock Skew ≥ CLK-to-Q + Longest Delay + Setup
  Cycle Time ≥ CLK-to-Q + Longest Delay + Setup + Clock Skew

How to Avoid Hold Time Violation?



- Hold time requirement:
  - Input to register must NOT change immediately after the clock tick
- This is usually easy to meet in the "edge trigger" clocking scheme
- Hold time of most FFs is <= 0 ns
- CLK-to-Q + Shortest Delay Path must be greater than Hold Time

Clock Skew's Effect on Hold Time



- The worst case scenario for hold time consideration:
  - The input register sees CLK2
  - The output register sees CLK1
  - fast FF2 output must not change input to FF1 for same clock edge
- (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time

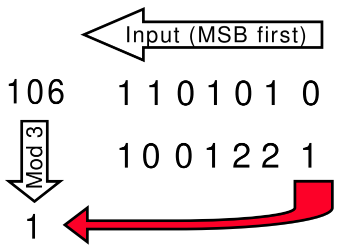
Administrative Matters

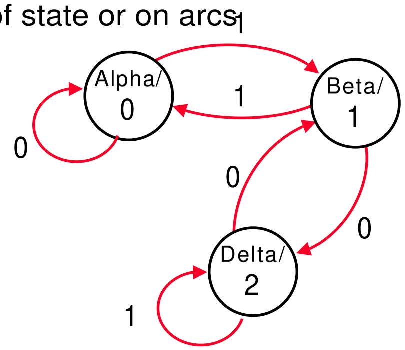
- Sections start tomorrow!
  - 2:00 – 4:00, 4:00 – 6:00 in 3107 Etcheverry
- Want announcements directly via EMail?
  - Look at information page to sign up for "cs152-announce" mailing list.
- Prerequisite quiz will be Monday 2/2 during class:
  - Review Sunday (2/1), 7:30 – 9:00 pm here (306 Soda)
  - Review Chapters 1-4, 7.1-7.2, Ap A, Ap, B of COD, Second Edition
  - Turn in survey form (with picture!) [Can't get into class without one!]
- Homework #1 also due Monday 2/2 at beginning of lecture!
  - No homework quiz this time (Prereq quiz may contain homework material, since this is supposed to be review)
- Lab 1 Due Wednesday 2/4

Finite State Machines:

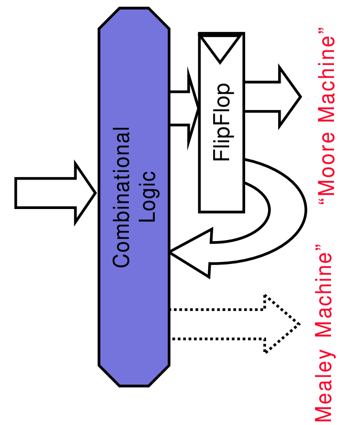
- System state is explicit in representation
- Transitions between states represented as arrows with inputs on arcs.
- Output may be either part of state or on arcs

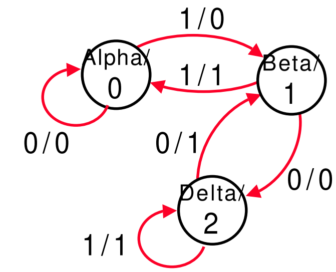
"Mod 3 Machine"





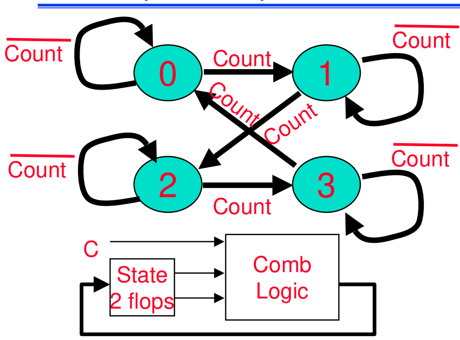
Implementation as Combinational logic + Latch





Input	State$_{old}$	State$_{new}$	Div
0	00	00	0
0	01	10	0
0	10	01	1
1	00	01	0
1	01	00	1
1	10	10	1

Example: Simplification of logic



S_1	S_0	C	S_1'	S_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

$$S_0' = (\overline{S_1}\cdot\overline{S_0}\cdot C) + (\overline{S_1}\cdot S_0\cdot\overline{C}) + (S_1\cdot\overline{S_0}\cdot C) + (S_1\cdot S_0\cdot\overline{C})$$
$$= (\overline{S_0}\cdot C) + (S_0\cdot\overline{C})$$

$$S_1' = (\overline{S_1}\cdot S_0\cdot C) + (S_1\cdot\overline{S_0}\cdot\overline{C}) + (S_1\cdot\overline{S_0}\cdot C) + (S_1\cdot S_0\cdot\overline{C})$$
$$= (\overline{S_1}\cdot S_0\cdot C) + (S_1\cdot\overline{C}) + (S_1\cdot\overline{S_0})$$

Karnaugh Map for easier simplification

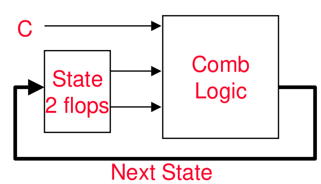
S_1	S_0	C	S_1'	S_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	1	0
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

S_0	00	01	11	10
0	0	1	1	0
1	1	0	0	1

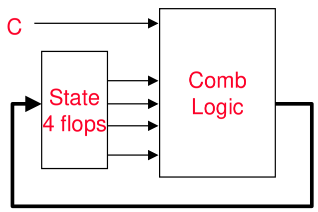
$$S_0' = (\overline{S_0}\cdot C) + (S_0\cdot\overline{C})$$

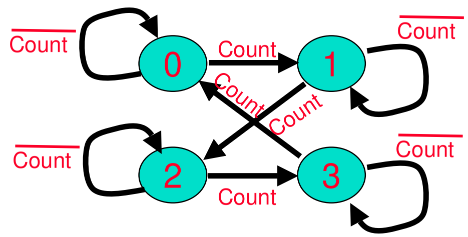
S_1	00	01	11	10
0	0	0	1	1
1	0	1	0	1

$$S_1' = (\overline{S_1}\cdot S_0\cdot C) + (S_1\cdot\overline{C}) + (S_1\cdot\overline{S_0})$$



One-Hot Encoding

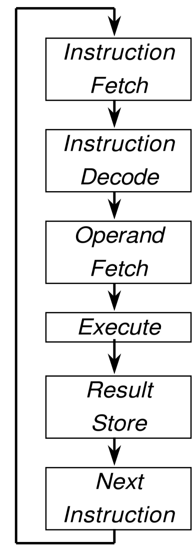




- One Flip-flop per state
- Only one state bit = 1 at a time
- Much faster combinational logic
- Tradeoff: Size ⇔Speed

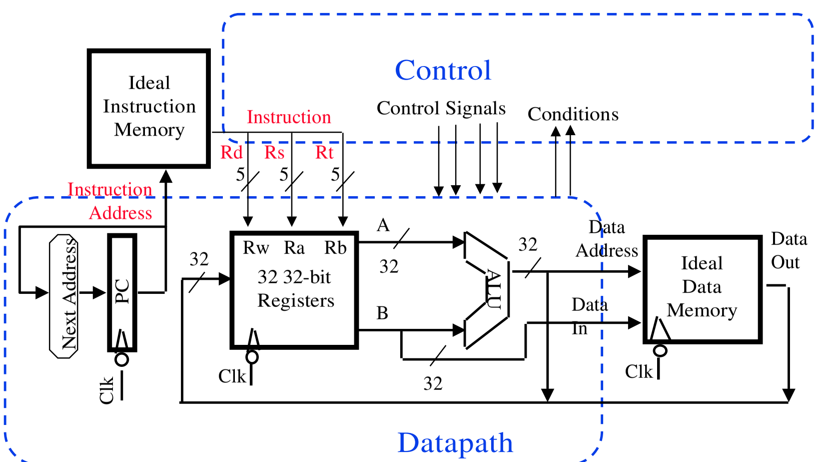
$$S_0' = (S_0 \cdot \overline{C}) + (S_3 \cdot C)$$
$$S_1' = (S_1 \cdot \overline{C}) + (S_0 \cdot C)$$
$$S_2' = (S_2 \cdot \overline{C}) + (S_1 \cdot C)$$
$$S_3' = (S_3 \cdot \overline{C}) + (S_2 \cdot C)$$

Review: The loop of control (is there a statemachine?)



- Instruction Format or Encoding
  - how is it decoded?
- Location of operands and result
  - where other than memory?
  - how many explicit operands?
  - how are memory operands located?
  - which can or cannot be in memory?
- Data type and Size
- Operations
  - what are supported
- Successor instruction
  - jumps, conditions, branches
  - *fetch-decode-execute is implicit!*

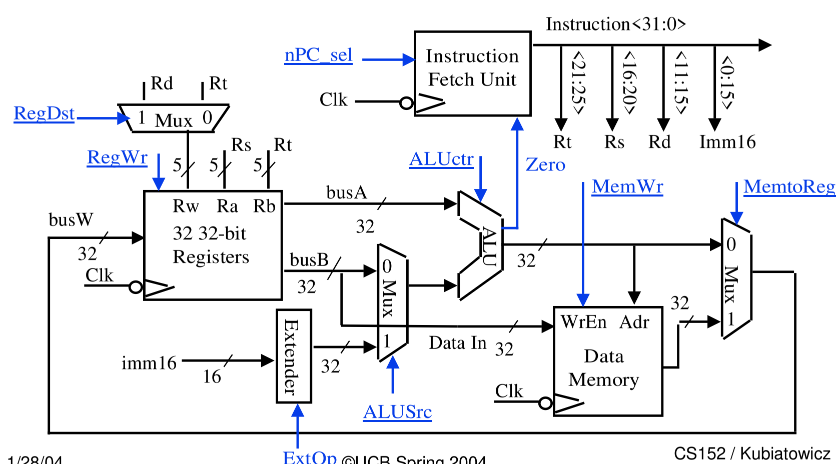
Designing a machine that executes MIPS



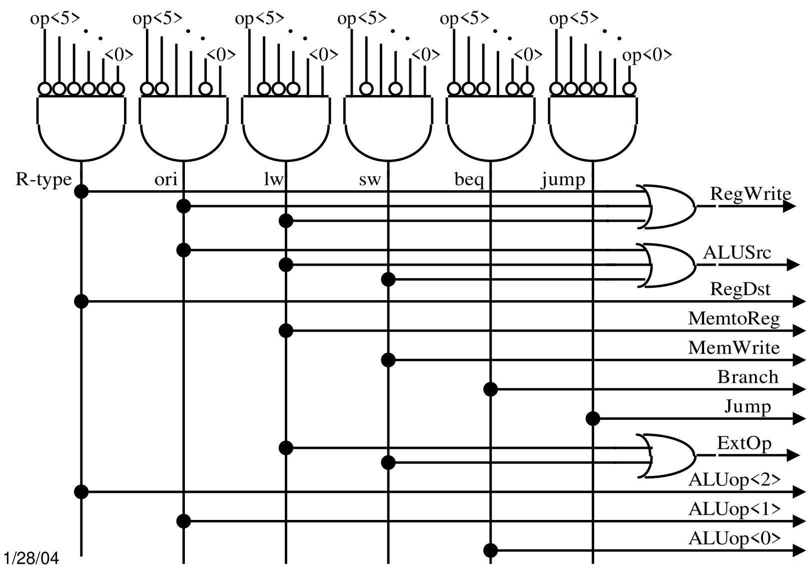
If you don’t fully remember this, it is ok! (Don’t need for prereq quiz)

A peek: A Single Cycle Datapath

- Rs, Rt, Rd and Imed16 hardwired from Fetch Unit
- Combinational logic for decode and lookup



A peek: PLA Implementation of the Main Control

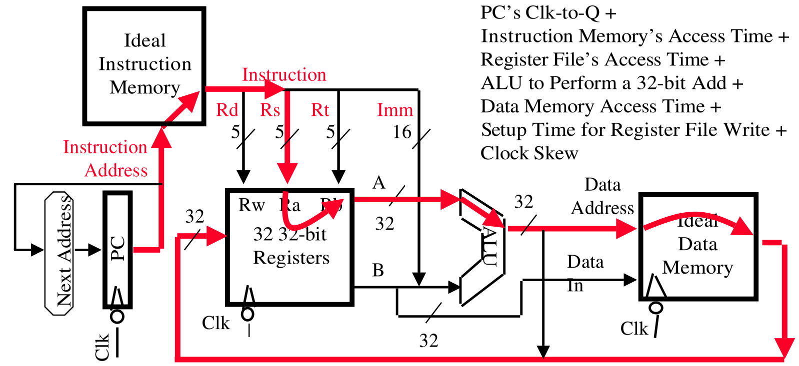


A peek: An Abstract View of the Critical Path (Load)

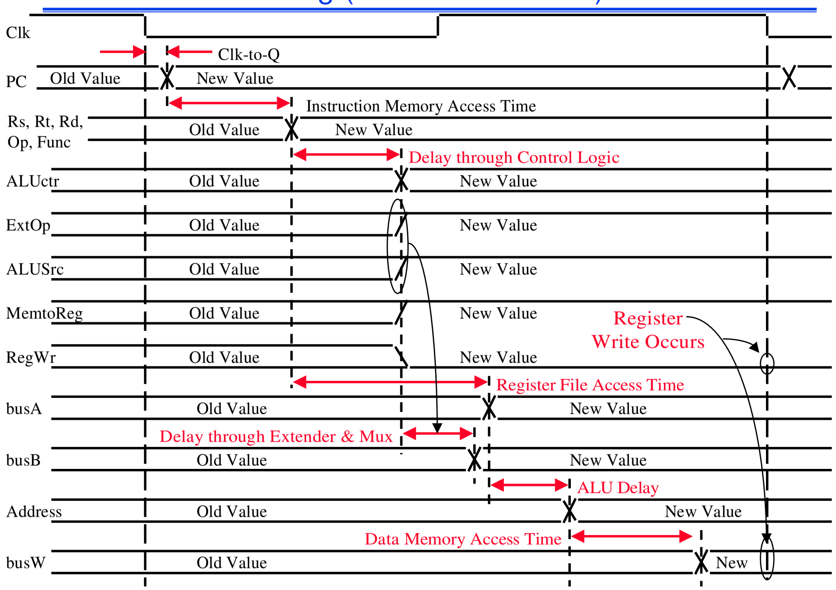
° Register file and ideal memory:

- The CLK input is a factor ONLY during write operation
- During read operation, behave as combinational logic:
  - Address valid => Output valid after "access time."

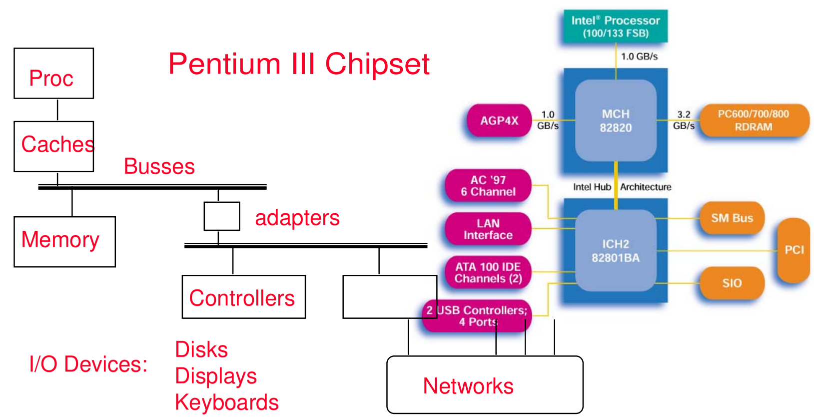
Critical Path (Load Operation) =
PC's Clk-to-Q +
Instruction Memory's Access Time +
Register File's Access Time +
ALU to Perform a 32-bit Add +
Data Memory Access Time +
Setup Time for Register File Write +
Clock Skew



Worst Case Timing (Load Instructions)



Ultimately: It's all about communication

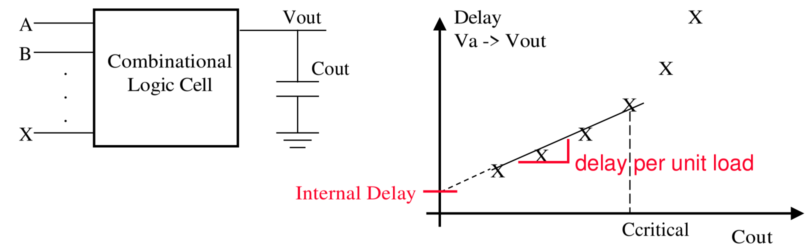


° All have interfaces & organizations

° New Pentium Chip: 30 cycle pipeline

- Pipeline stages for communication? I would bet it's true!

Review: General C/L Cell Delay Model



° Combinational Cell (symbol) is fully specified by:
- functional (input -> output) behavior
  - truth-table, logic equation, VHDL
- load factor of each input
- critical propagation delay from each input to each output for each transition
  - T_{HL}(A, o) = Fixed Internal Delay + Load-dependent-delay x *load*

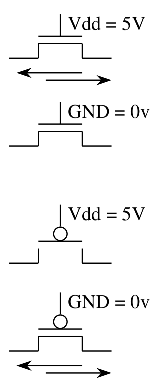
° Linear model composes

Basic Technology: CMOS

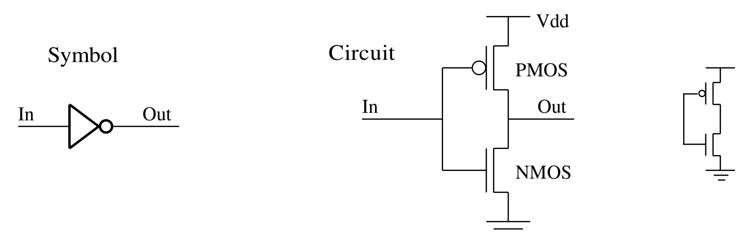
° CMOS: Complementary Metal Oxide Semiconductor
- NMOS (N-Type Metal Oxide Semiconductor) transistors
- PMOS (P-Type Metal Oxide Semiconductor) transistors

° NMOS Transistor
- Apply a HIGH (Vdd) to its gate turns the transistor into a "conductor"
- Apply a LOW (GND) to its gate shuts off the conduction path

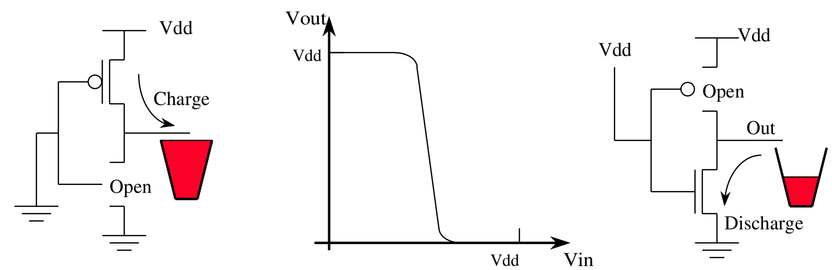
° PMOS Transistor
- Apply a HIGH (Vdd) to its gate shuts off the conduction path
- Apply a LOW (GND) to its gate turns the transistor into a "conductor"



Basic Components: CMOS Inverter

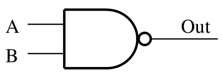


° Inverter Operation



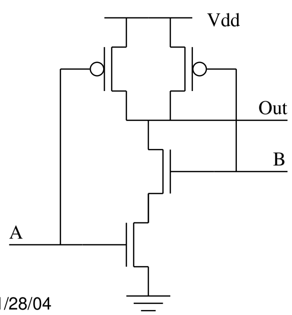
Basic Components: CMOS Logic Gates

NAND Gate

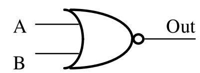


A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

Out = $\overline{A \bullet B}$

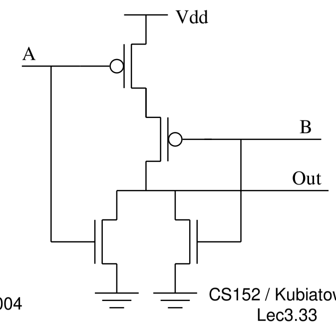


NOR Gate



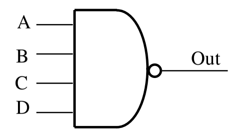
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

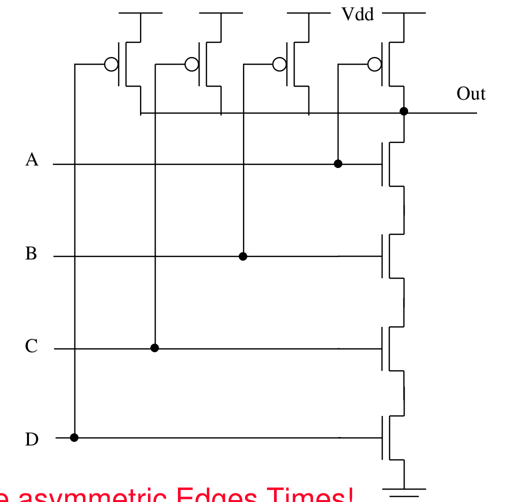
Out = $\overline{A + B}$



Basic Components: CMOS Logic Gates

4-input NAND Gate

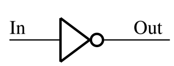


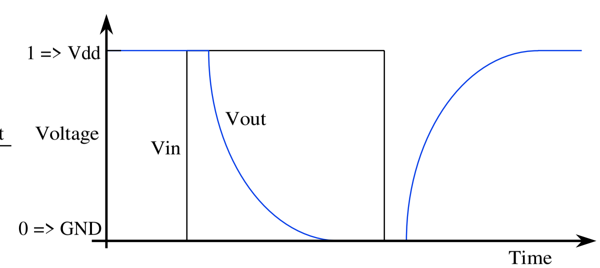


More Inputs   More asymmetric Edges Times!

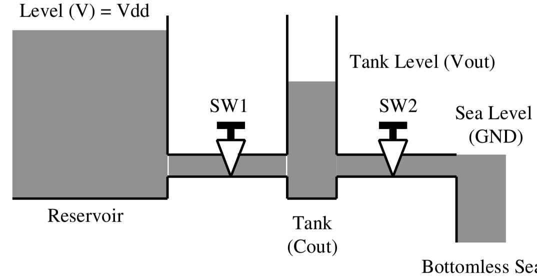
Ideal versus Reality

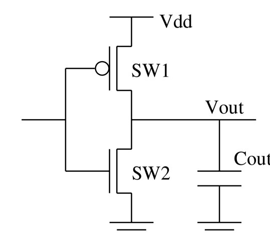
- When input 0 -> 1, output 1 -> 0 but NOT instantly
  - Output goes 1 -> 0: output voltage goes from Vdd (5v) to 0v
- When input 1 -> 0, output 0 -> 1 but NOT instantly
  - Output goes 0 -> 1: output voltage goes from 0v to Vdd (5v)
- Voltage does not like to change instantaneously





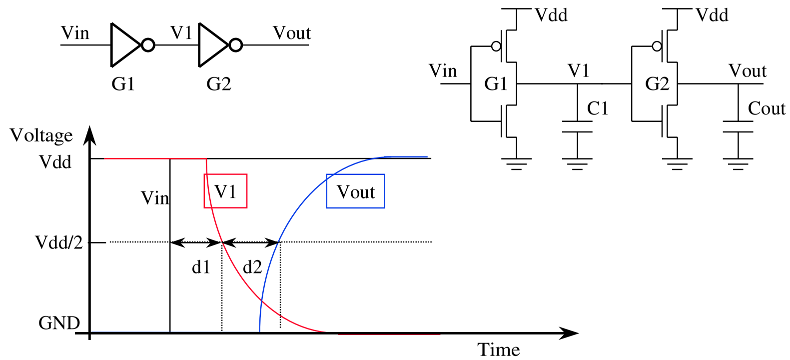
Fluid Timing Model





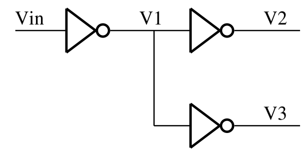
- Water ⇔ Electrical Charge   Tank Capacity ⇔ Capacitance (C)
- Water Level ⇔ Voltage   Water Flow ⇔ Charge Flowing (Current)
- Size of Pipes ⇔ Strength of Transistors (G)
- Time to fill up the tank proportional to C / G

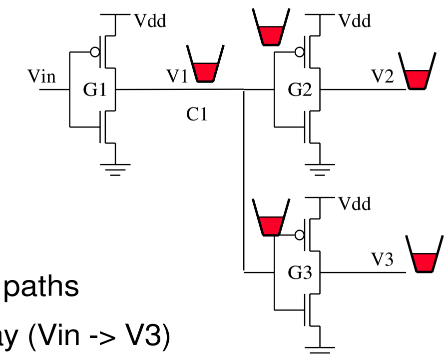
Series Connection



- Total Propagation Delay = Sum of individual delays = d1 + d2
- Capacitance C1 has two components:
  - Capacitance of the wire connecting the two gates
  - Input capacitance of the second inverter

Calculating Aggregate Delays

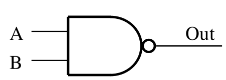




- Sum delays along serial paths
- Delay (Vin -> V2) ! = Delay (Vin -> V3)
  - Delay (Vin -> V2) = Delay (Vin -> V1) + Delay (V1 -> V2)
  - Delay (Vin -> V3) = Delay (Vin -> V1) + Delay (V1 -> V3)
- Critical Path = The longest among the N parallel paths
- C1 = Wire C + Cin of Gate 2 + Cin of Gate 3

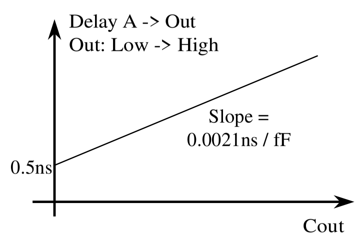
Characterize a Gate

- Input capacitance for each input
- For each input-to-output path:
  - For each output transition type (H->L, L->H, H->Z, L->Z ... etc.)
    - Internal delay (ns)
    - Load dependent delay (ns / fF)
- Example: 2-input NAND Gate

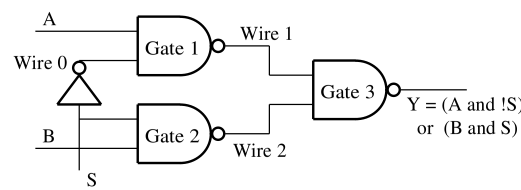


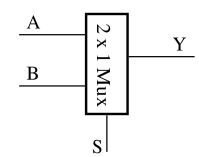
For A and B: Input Load (I.L.) = 61 fF

For either A -> Out or B -> Out:
Tlh = 0.5ns Tlhf = 0.0021ns / fF
Thl = 0.1ns Thlf = 0.0020ns / fF



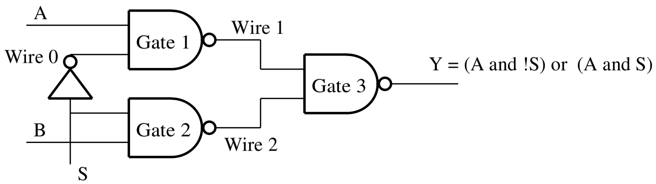
A Specific Example: 2 to 1 MUX





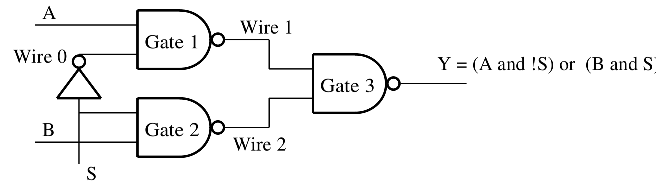
- Input Load (I.L.)
  - A, B: I.L. (NAND) = 61 fF
  - S: I.L. (INV) + I.L. (NAND) = 50 fF + 61 fF = 111 fF
- Load Dependent Delay (L.D.D.): Same as Gate 3
  - TAYlhf = 0.0021 ns / fF   TAYhlf = 0.0020 ns / fF
  - TBYlhf = 0.0021 ns / fF   TBYhlf = 0.0020 ns / fF
  - TSYlhf = 0.0021 ns / fF   TSYlhf = 0.0020 ns / fF

2 to 1 MUX: Internal Delay Calculation



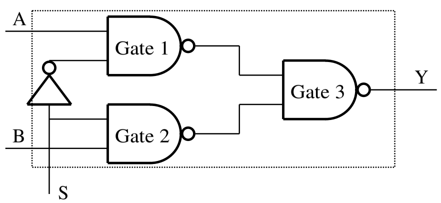
- Internal Delay (I.D.):
  - A to Y: I.D. G1 + (Wire 1 C + G3 Input C) * L.D.D G1 + I.D. G3
  - B to Y: I.D. G2 + (Wire 2 C + G3 Input C) * L.D.D. G2 + I.D. G3
  - S to Y (Worst Case): I.D. Inv + (Wire 0 C + G1 Input C) * L.D.D. Inv + Internal Delay A to Y
- We can approximate the effect of "Wire 1 C" by:
  - Assume Wire 1 has the same C as all the gate C attached to it.

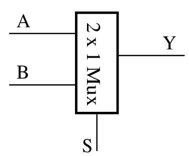
2 to 1 MUX: Internal Delay Calculation (continue)



- Internal Delay (I.D.):
  - A to Y: I.D. G1 + (Wire 1 C + G3 Input C) * L.D.D G1 + I.D. G3
  - B to Y: I.D. G2 + (Wire 2 C + G3 Input C) * L.D.D. G2 + I.D. G3
  - S to Y (Worst Case): I.D. Inv + (Wire 0 C + G1 Input C) * L.D.D. Inv + Internal Delay A to Y
- Specific Example:
  - TAYlh = TPhl G1 + (2.0 * 61 fF) * TPhlf G1 + TPlh G3
    = 0.1ns + 122 fF * 0.0020 ns/fF + 0.5ns = 0.844 ns

Abstraction: 2 to 1 MUX

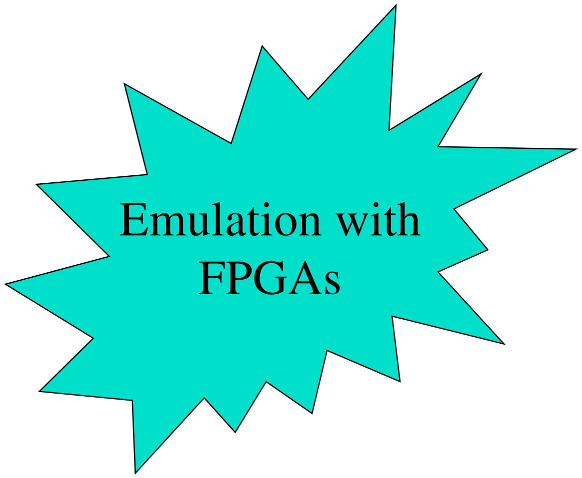




- Input Load: A = 61 fF, B = 61 fF, S = 111 fF
- Load Dependent Delay:
  - TAYlhf = 0.0021 ns / fF TAYhlf = 0.0020 ns / fF
  - TBYlhf = 0.0021 ns / fF TBYhlf = 0.0020 ns / fF
  - TSYlhf = 0.0021 ns / fF TSYlhf = 0.0020 ns / f F
- Internal Delay:
  - TAYlh = TPhl G1 + (2.0 * 61 fF) * TPhlf G1 + TPlh G3
    = 0.1ns + 122 fF * 0.0020ns/fF + 0.5ns = 0.844ns
  - Fun Exercises: TAYhl, TBYlh, TSYlh, TSYlh

KISS RULE: "Keep It Simple, Stupid!"

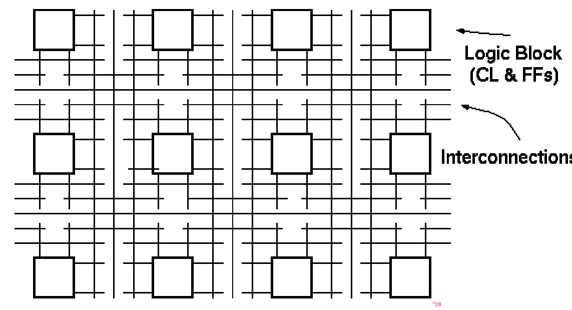
- Simple designs:
  - Can be debugged easier
  - Have lower capacitance on any one output (less *fan-out*)
  - Have fewer gates in the critical path (complexity more gates)
  - Less Power consumption
- Complex designs:
  - More gates/capacitance (probably slower clock rate!)
  - More functionality per cycle (may occasionally win out!)
  - More Power
  - More Bugs!
- Which is better? Better evaluate carefully



FPGA Overview

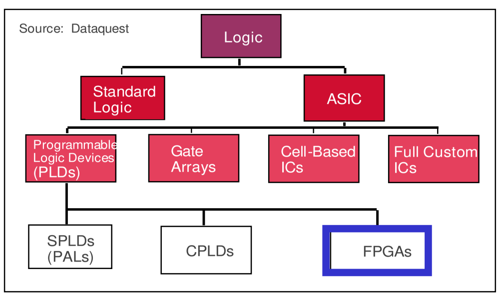
° Basic idea: 2D array of combination logic blocks (CL) and flip-flops (FF) with a means for the user to configure both:

1. the interconnection between the logic blocks,
2. the function of each block.



Simplified version of FPGA internal architecture

Where are FPGAs in the IC Zoo?



Acronyms
SPLD = Simple Prog. Logic Device
PAL = Prog. Array of Logic
CPLD = Complex PLD
FPGA = Field Prog. Gate Array

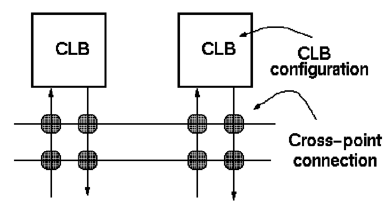
(Standard logic is SSI or MSI buffers, gates)

Common Resources
Configurable Logic Blocks (CLB)
Memory Look-Up Table
AND-OR planes
Simple gates
Input / Output Blocks (IOB)
Bidirectional, latches, inverters, pullup/pulldowns
Interconnect or Routing
Local, internal feedback, and global

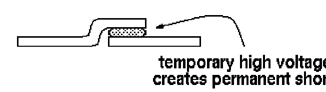
FPGA Variations

° Families of FPGA's differ in:
- physical means of implementing user programmability,
- arrangement of interconnection wires, and
- basic functionality of logic blocks

° Most significant difference is in the method for providing flexible blocks and connections:



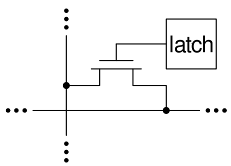
° Anti-fuse based (ex: Actel)



+ Non-volatile, relatively small
- fixed (non-reprogrammable)
(Almost used in 150 Lab: only 1-shot at getting it right!)

User Programmability

- Latch-based (Xilinx, Altera, …)



+ reconfigurable

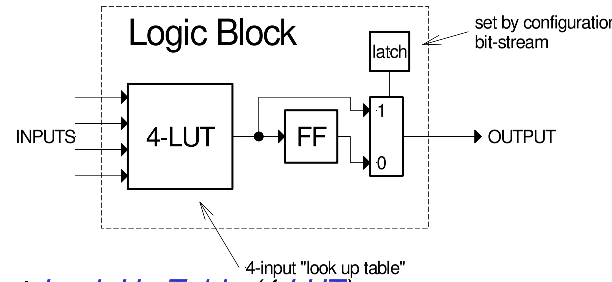
- volatile

- relatively large die size

- Note: Today 90% die is interconnect, 10% is gates

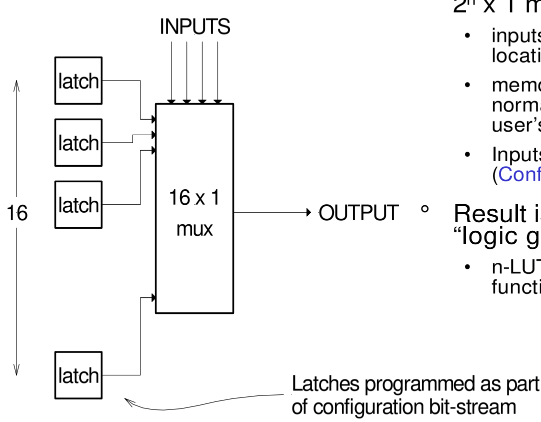
- Latches are used to:
  1. make or break cross-point connections in interconnect
  2. define function of logic blocks
  3. set user options:
     - within the logic blocks
     - in the input/output blocks
     - global reset/clock
- "Configuration bit stream" loaded under user control:
  - All latches are strung together in a shift chain
  - "Programming" => creating bit stream

Idealized FPGA Logic Block



- 4-input *Look Up Table (4-LUT)*
  - implements combinational logic functions
- Register
  - optionally stores output of LUT
  - Latch determines whether read reg or LUT

4-LUT Implementation



- n-bit LUT is actually 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 (Configurable Logic Block) 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 value of 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.

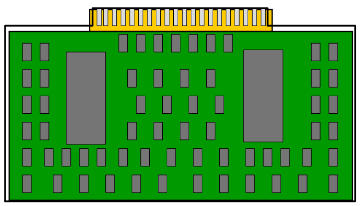
How many functions of n inputs?

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		

Why FPGAs? (1 / 5)

- By the early 1980's most of logic circuits in typical systems were absorbed by a handful of standard large scale integrated circuits (LSI ICs).
  - Microprocessors, bus/IO controllers, system timers, ...
- Every system still needed random small "glue logic" ICs 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.

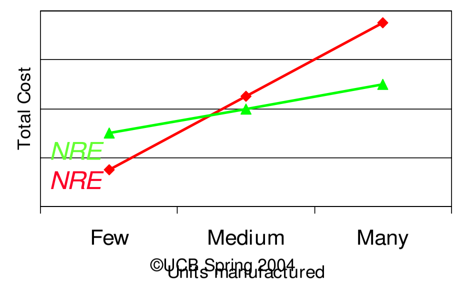


Printed Circuit (PC) board with many small SSI and MSI ICs and a few LSI ICs

Why FPGAs? (2 / 5)

- Custom ICs sometimes designed to replace glue logic:
  - reduced complexity/manufacturing cost, improved performance
  - But custom ICs expensive to develop, and delay introduction of product ("time to market") because of increased design time
- Note: need to worry about two kinds of costs:
  1. cost of development, "Non-Recurring Engineering (NRE)", fixed
  2. cost of manufacture per unit, variable

  Usually tradeoff between NRE cost and manufacturing costs



Why FPGAs? (3 / 5)

- Therefore custom IC approach was only viable for products with very high volume (where NRE could be amortized), and not sensitive in time to market (TTM)
- FPGAs introduced as alternative to custom ICs for implementing glue logic:
  - improved PC board density vs. discrete SSI/MSI components (within around 10x of custom ICs)
  - computer aided design (CAD) tools meant circuits could be implemented quickly (no physical layout process, no mask making, no IC manufacturing), relative to Application Specific ICs (ASICs) (3-6 months for these steps for custom IC)
    - lowers NREs (Non Recurring Engineering)
    - shortens TTM (Time To Market)
- 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.

Why FPGAs? (4 / 5)

- FPGAs continue to compete with custom ICs for special processing functions (and glue logic) but now try to 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)
- MICRO: Highest NRE, SW: fastest TTM
- ASIC: Highest performance, worst TTM
- FPGA: Highest cost per chip (unit cost)

Why FPGAs? (5 / 5)

- As Moore's Law continues, FPGAs work for more applications as both can do more logic in 1 chip and faster
- Can easily be "patched" vs. ASICs
- Perfect for courses:
  - Can change design repeatedly
  - Low TTM yet reasonable speed
- With Moore's Law, now can do full CS 152 project easily inside 1 FPGA

Summary

- Design = translating specification into physical components
  - Combinational, Sequential (FlipFlops), Wires
- Timing is important
  - Critical path: maximum time between clock edges
- Clocking Methodology and Timing Considerations
  - Simplest clocking methodology
    - All storage elements use the SAME clock edge
  - Cycle Time $\geq$ CLK-to-Q + Longest Delay Path + Setup + Clock Skew
  - (CLK-to-Q + Shortest Delay Path - Clock Skew) > Hold Time
- Algebraic Simplification
  - Karnaugh Maps
  - Speed $\Leftrightarrow$ Size tradeoffs! (Many to be shown
- Performance and Technology Trends
  - Keep the design simple (KISS rule) to take advantage of the latest technology
  - CMOS inverter and CMOS logic gates
- Delay Modeling and Gate Characterization
  - Delay = Internal Delay + (Load Dependent Delay x Output Load)
- FPGAs: programmable logic