
# Welcome to 6.111!

- Introductions, course mechanics
- Course overview
- Digital signaling
- Combinational logic
- 4 Handouts: slides, LP #1, info form, kit signout with safety information

Lecture material: Prof Anantha Chandrakasan and Dr. Chris Terman.



## Introductions

Gim Hom
*Lectures*

Shawn Jain
*TA*

Weston Braun
UTA

Mitchell Gu
UTA

Alex Sloboda
UTA

Joe Steinmeyer
*Course Assistant*

David Gomez

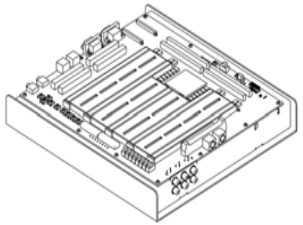
Valerie Sarge

*LA's*

Madeline Waller



## Introductions – The Hardware



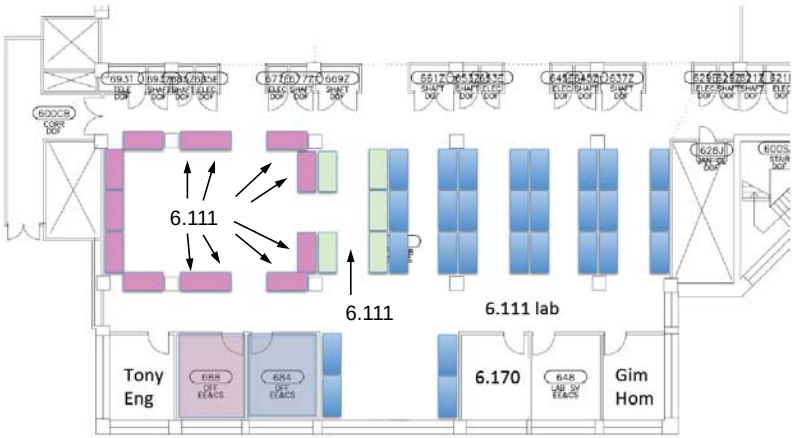
6.111 Labkit
*6M-gate FPGA + audio + video + memories + …*

Nexys 4 –DDR
Analog Input ,PWM Audio,
ADX362 3-axisaccelerometer,
*ADI temp sensor …*



## 38-600

41 Stations



**Course Website:** web.mit.edu/6.111

Fall 2016

Home
Announcements

Labkit
Handouts
»Lectures

»Final Project info

Final Projects
»Project list
»Schedule

*MIT cert required
*On-line Grades
*Submit PDFs
*Submit Verilog
*Staffed Lab Hours

Course info
Course objectives
Course calendar

Past projects

Tools
Verilog Editors (new tab)

Piazza (new tab)

6.111 Introductory Digital Systems Laboratory

Announcements, updates, etc

Online copies of lecture notes, lpsets and labs

Final project info

Lab: 38-600

On-line grades

PDF submissions

Verilog submissions

Policies and important dates

Tools

On-line Q&A

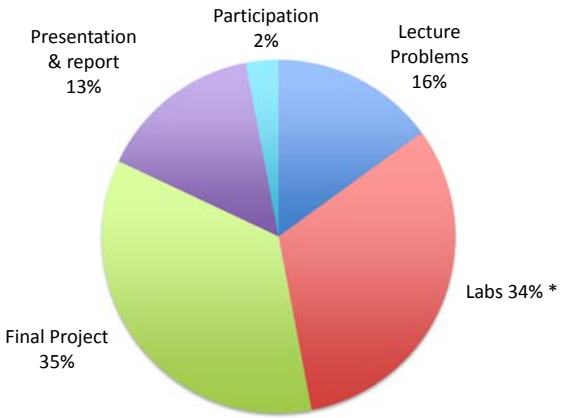


# 6.111 in lieu of 6.UAP

- Under the pre-2015 EECS curriculum, a department CI-M lab can be used to meet the 6.UAP requirement. With the change to the new EECS curriculum, 6.111 starting Fall 2016 will continue to be a AUS/AUS2 Lab subject but not a CI-M. However, EECS and SOCR (Subcommittee on Communications Requirements) have approved 6.111 Fall 2016 as a substitution for 6.UAP.
- 6.111 will continue to be a 12 unit subject.
- Prior to add date, will submit a list of students to have this substitution approved.



# Assignments




A large number of students do "A" level work and are, indeed, rewarded with a grade of "A". The corollary to this is that, since average performance levels are so high, punting any part of the subject can lead to a disappointing grade.



# Project Presentation & Report (13%)

- Design proposal (2%)
- Design presentation (6%)
- Final Report (5%)

# Labs: learning the ropes

- Lab 1
  - Experiment with gates, design & implement some logic
  - Learn about lab equipment in the Digital Lab (38-600): oscilloscopes and logic analyzers
- Lab 2
  - Introduction to Verilog, ModelSim & the labkit
- Lab 3
  - Video circuits: a simple Pong game
    - Use Verilog to program an FPGA
- Lab 4
  - Design and implement a Finite State Machine (FSM) – Car Alarm *
- Lab 5
  - Design a complicated system with multiple FSMs (Major/Minor FSM)
    - Voice recorder using AC97 codec and SRAMs or
    - Build your own remote control *
- All labs must be completed before starting final project.

* 6.111 labkit or Nexys 4 implementation



# Final Project

- Done in groups of two or three; one person project by exception
- Open-ended
- You and the staff negotiate a project proposal
  - Must emphasize digital concepts, but inclusion of analog interfaces (e.g., data converters, sensors or motors) common and often desirable
  - Proposal Conference, several Design Reviews
- Design presentation to staff
- Staff will provide help with project definition and scope, design, debugging, and testing
- It is extremely difficult for a student to receive an A without completing the final project. Sorry, but we don't give incompletes.

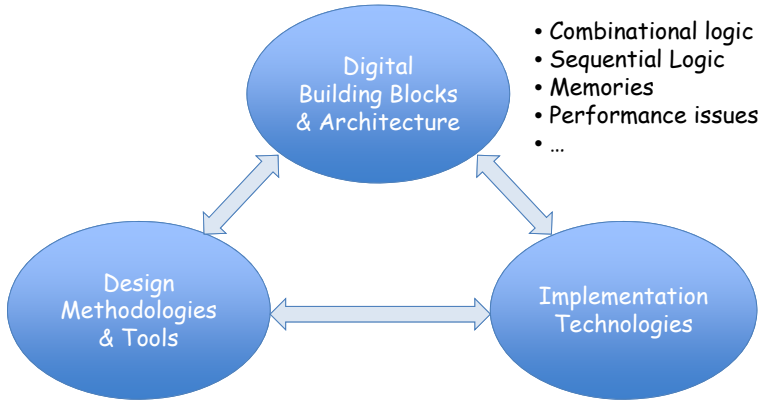


# Collaboration

- Labs must be done independently but students may seek help from other students.
- Work submitted for review must be their own



# 6.111 Topics



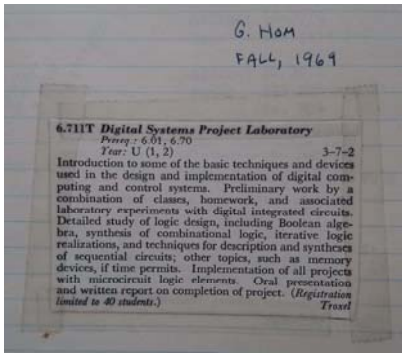

- Combinational logic
- Sequential Logic
- Memories
- Performance issues
- …

- Design metrics
- HDL: Verilog
- Simulation tools
- Synthesis, Place & Route
- …

- FPGAs
- Flash, ZBT ram
- AC97, TripleDAC
- …

# 6.111 Evolution

Fall 1969


G. Hom
FALL, 1969

**6.711T *Digital Systems Project Laboratory***
*Prereq*: 6.01, 6.70
*Year*: U (1, 2) 3–7–2
Introduction to some of the basic techniques and devices used in the design and implementation of digital computing and control systems. Preliminary work by a combination of classes, homework, and associated laboratory experiments with digital integrated circuits. Detailed study of logic design, including Boolean algebra, synthesis of combinational logic, iterative logic realizations, and techniques for description and syntheses of sequential circuits; other topics, such as memory devices, if time permits. Implementation of all projects with microcircuit logic elements. Oral presentation and written report on completion of project. (*Registration limited to 40 students.*)
*Travel*

Fall 2016

**6.111 Introductory Digital Systems Laboratory**

U(🕈) 🧪
Prereq: 6.002, 6.071, or 16.004
Units: 3-7-2
**Lecture**: *TR2.30-4* (32-124) **Lab**: *TBA*

Lectures and labs on digital logic, flip flops, PALs, FPGAs, counters, timing, synchronization, and finite-state machines prepare students for the design and implementation of a final project of their choice: games, music, digital filters, wireless communications, video, or graphics. Extensive use of Verilog for describing and implementing digital logic designs



# The First Computer

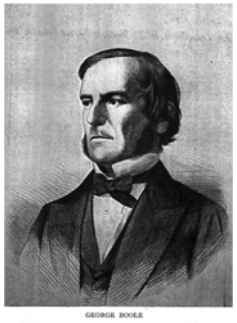
The Babbage Difference Engine (1834)

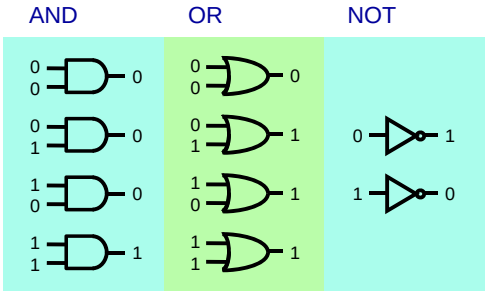
25,000 parts
cost: £17,470

- The first digital systems were mechanical and used base-10 representation.
- Most popular applications: arithmetic and scientific computation



# Meanwhile, in the World of Theory…

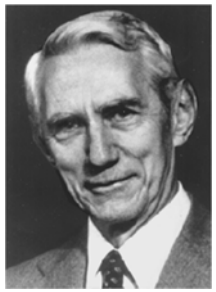

GEORGE BOOLE

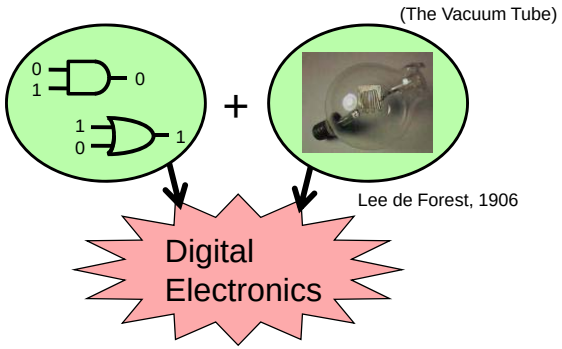



- 1854: George Boole shows that logic is math, not just philosophy!
- Boolean algebra: the mathematics of binary values



# Key Link Between Logic and Circuits






- Despite existence of relays and introduction of vacuum tube in 1906, *digital* electronics did not emerge for thirty years!
- Claude Shannon notices similarities between Boolean algebra and electronic telephone switches
- Shannon's 1937 MIT Master's Thesis introduces the world to binary digital electronics

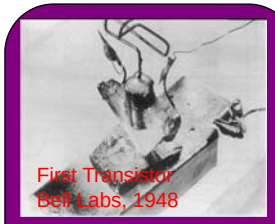
# Evolution of Digital Electronics

Vacuum Tubes | Transistors | VLSI Circuits

ENIAC, 1946

UNIVAC, 1951

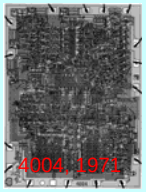
1900 adds/sec



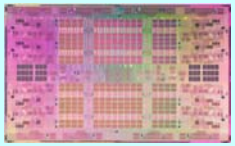
First Transistor
Bell Labs, 1948

IBM System/360, 1964

500,000 adds/sec

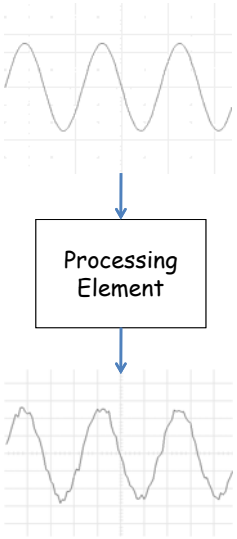


4004, 1971



Intel Poulson, 2013
8 Cores
>>2 Billion adds/sec

# The trouble with analog signaling




The real world is full of continuous-time continuous-value (aka "analog") signals created by physical processes: sound vibrations, light fields, voltages and currents, phase and amplitudes, …

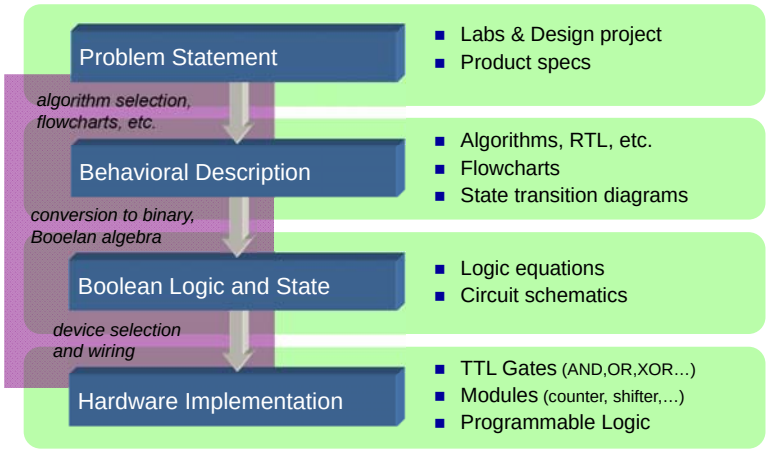
But if we build processing elements to manipulate these signals we must use non-ideal components in real-world environments, so some amount of error (aka "noise") is introduced. The error comes from component tolerances, electrical phenomenon (e.g., IR and LdI/dt effects), transmission losses, thermal noise, etc. Facts of life that can't be avoided…

And the more analog processing we do, the worse it gets: signaling errors accumulate in analog systems since we can't tell from looking at signal which wiggles were there to begin with and which got added during processing.



# Building Digital Systems

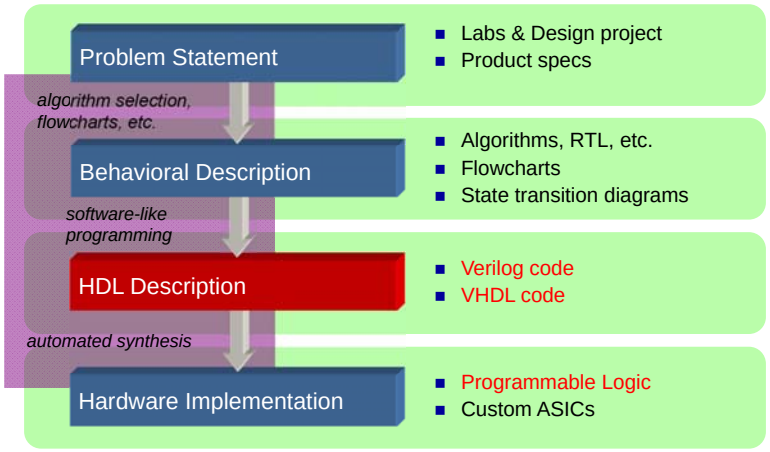
- Goal of 6.111: Building binary digital solutions to computational problems






# Building Digital Systems with HDLs

- Logic synthesis using a Hardware Description Language (HDL) automates the most tedious and error-prone aspects of design



# Verilog and VHDL

| VHDL | Verilog |
|---|---|
| ■ Commissioned in 1981 by Department of Defense; now an IEEE standard | • Created by Gateway Design Automation in 1985; now an IEEE standard |
| ■ Initially created for ASIC synthesis | • Initially an interpreted language for gate-level simulation |
| ■ Strongly typed; potential for verbose code | • Less explicit typing (e.g., compiler will pad arguments of different widths) |
| ■ Strong support for package management and large designs | • No special extensions for large designs |

Hardware structures can be modeled effectively in either VHDL and Verilog. Verilog is similar to c and a bit easier to learn.



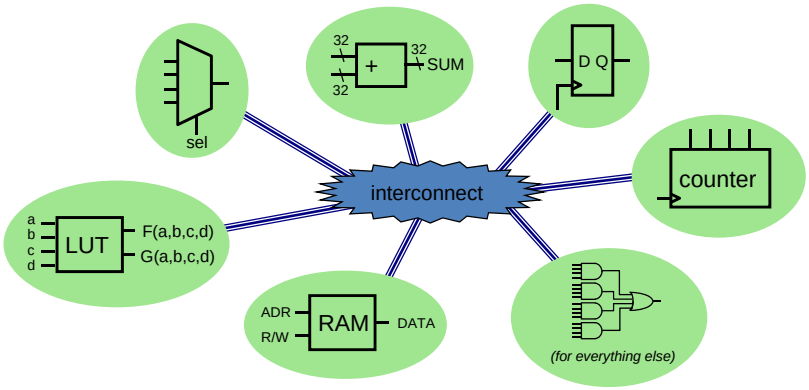
# Verilog HDL

- Misconceptions
  - The coding style or clarity does not matter as long as it works
  - Two different Verilog encodings that simulate the same way will synthesize to the same set of gates
  - Synthesis just can't be as good as a design done by humans
    - Shades of assembly language versus a higher level language
- What can be Synthesized
  - Combinational Functions
    - Multiplexors, Encoders, Decoders, Comparators, Parity Generators, Adders, Subtractors, ALUs, Multipliers
    - Random logic
  - Control Logic
    - FSMs
- What can't be Synthesized
  - Precise timing blocks (e.g., delay a signal by 2ns)
  - Large memory blocks (can be done, but very inefficient)
- Understand what constructs are used in simulation vs. hardware mapping



# The FPGA: A Conceptual View

- An FPGA is like an electronic breadboard that is wired together by an automated synthesis tool
- Built-in components are called macros





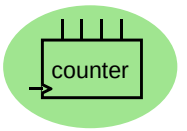

# Synthesis and Mapping for FPGAs

- Infer macros: choose the FPGA macros that efficiently implement various parts of the HDL code

```
...
always @ (posedge clk)
begin
    count <= count + 1;
end
...
```

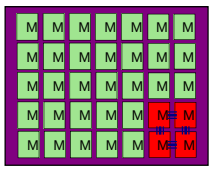
HDL Code

*"This section of code looks like a counter. My FPGA has some of those..."*




Inferred Macro

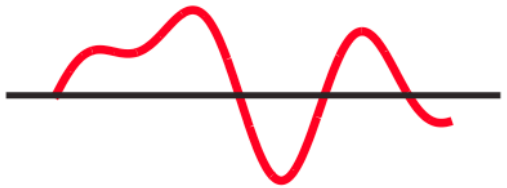
- Place-and-route: with area and/or speed in mind, choose the needed macros by location and route the interconnect

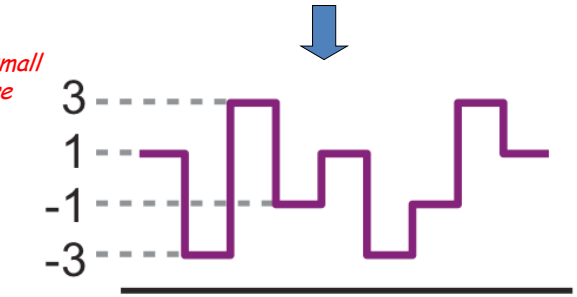



*"This design only uses 10% of the FPGA. Let's use the macros in one corner to minimize the distance between blocks."*

# Solution: go digital!

Continuous values
Continuous time






*So we can detect small changes and restore original values*

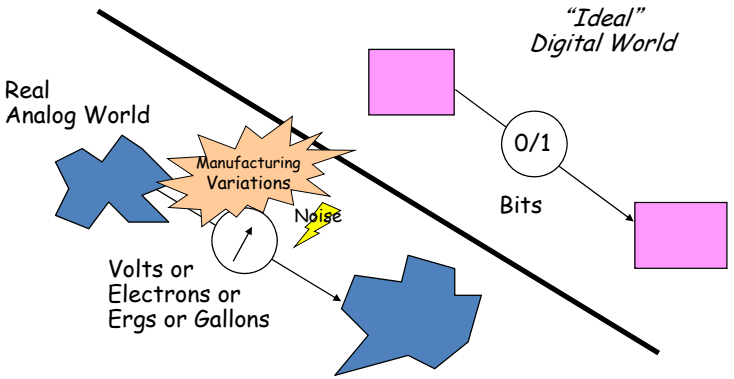
Discrete values
Discrete time

*So we don't look while it's changing*



# The Digital Abstraction

Noise and inaccuracy are inevitable; we can't reliably engineer perfect components – we must design our system to tolerate some amount of error if it is to process information reliably.



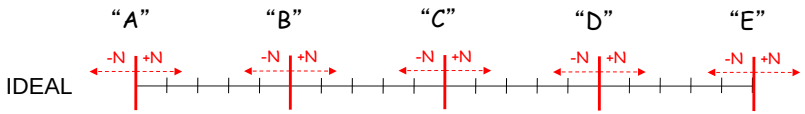

Keep in mind that the world is not digital, we would simply like to engineer it to behave that way. Furthermore, we must use real physical phenomena to implement digital designs!



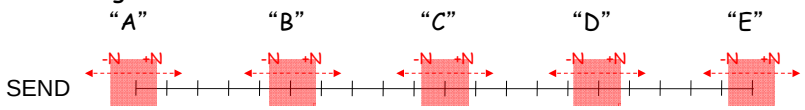
# Digital Signaling: sending

To ensure we can distinguish signal from noise, we'll encode information using a fixed set of discrete values called symbols.

Given a bound N on the size of possible errors, if the analog representations for the symbols are chosen to be at least 2N apart, we should be able to detect and eliminate errors of up to ±N.



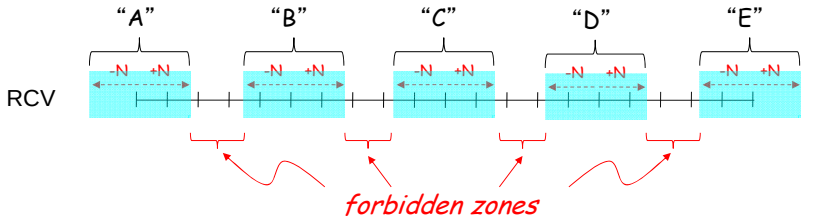

Since we will use non-ideal components in the sender, we allow each transmitted symbol to be represented by a (small) range of analog values.





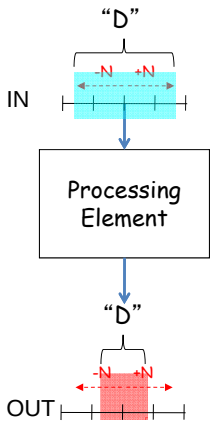

# Digital Signaling: receiving

Since the channel/wire is imperfect and we will use non-ideal components in the receiver, we require the receiver to accept a (larger) range of analog values for each symbol.




To avoid hard-to-make decisions at the boundaries between symbol representations, insert a "forbidden zone" between symbols so that some ranges of received values are not required to be mapped to a specific symbol.

# Digital processing elements



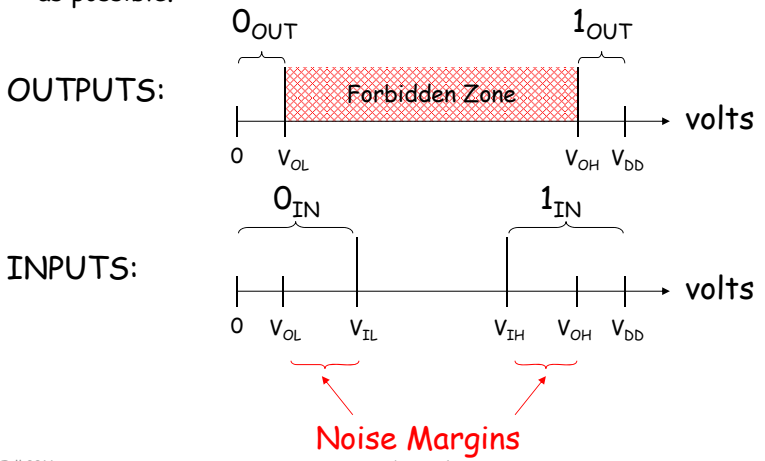

Digital processing elements *restore* noisy input values to legal output values – signaling errors don't accumulate in digital systems. So the number of processing elements isn't limited by noise problems!

The "trick" is that we've defined our signaling convention so that we can tell from looking at a signal which wiggles were there to begin with and which got added during processing.



# Using voltages to encode binary values

We'll keep things simple by designing our processing elements to use voltages to encode binary values (0 or 1). To ensure robust operation we'd like to make the noise margins as large as possible.






# Digital Signaling Specification

Digital input: $V_{IN} < V_{IL}$ or $V_{IN} > V_{IH}$

Digital output: $V_{OUT} < V_{OL}$ or $V_{OUT} > V_{OH}$

Noise margins: $V_{IL} - V_{OL}$ and $V_{OH} - V_{IH}$

Where $V_{OL}$, $V_{IL}$, $V_{IH}$ and $V_{OH}$ are part of the specification for a particular family of digital components.

Now that we have a way of encoding information as a signal, we can define what it means to be *digital device*.



# Sample DC (signaling) Specification

| I/O Standard | $V_{IL}$ | | $V_{IH}$ | | $V_{OL}$ | $V_{OH}$ | $I_{OL}$ | $I_{OH}$ |
|---|---|---|---|---|---|---|---|---|
| | V, Min | V, Max | V, Min | V, Max | V, Max | V, Min | mA | mA |
| LVTTL | –0.3 | 0.8 | 2.0 | 3.45 | 0.4 | 2.4 | Note(3) | Note(3) |
| LVCMOS33, LVDCI33 | –0.3 | 0.8 | 2.0 | 3.45 | 0.4 | $V_{CCO}$ – 0.4 | Note(3) | Note(3) |
| LVCMOS25, LVDCI25 | –0.3 | 0.7 | 1.7 | $V_{CCO}$ + 0.3 | 0.4 | $V_{CCO}$ – 0.4 | Note(3) | Note(3) |
| LVCMOS18, LVDCI18 | –0.3 | 35% $V_{CCO}$ | 65% $V_{CCO}$ | $V_{CCO}$ + 0.3 | 0.45 | $V_{CCO}$ – 0.45 | Note(4) | Note(4) |
| LVCMOS15, LVDCI15 | –0.3 | 35% $V_{CCO}$ | 65% $V_{CCO}$ | $V_{CCO}$ + 0.3 | 25% $V_{CCO}$ | 75% $V_{CCO}$ | Note(4) | Note(4) |
| LVCMOS12 | –0.3 | 35% $V_{CCO}$ | 65% $V_{CCO}$ | $V_{CCO}$ + 0.3 | 25% $V_{CCO}$ | 75% $V_{CCO}$ | Note(6) | Note(6) |
| PCI33_3[5] | –0.2 | 30% $V_{CCO}$ | 50% $V_{CCO}$ | $V_{CCO}$ | 10% $V_{CCO}$ | 90% $V_{CCO}$ | Note(5) | Note(5) |
| PCI66_3[5] | –0.2 | 30% $V_{CCO}$ | 50% $V_{CCO}$ | $V_{CCO}$ | 10% $V_{CCO}$ | 90% $V_{CCO}$ | Note(5) | Note(5) |
| PCI-X[5] | –0.2 | 35% $V_{CCO}$ | 50% $V_{CCO}$ | $V_{CCO}$ | 10% $V_{CCO}$ | 90% $V_{CCO}$ | Note(5) | Note(5) |

Source: Xilinx Virtex 5 Datasheet

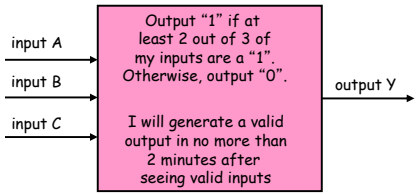
# A Digital Processing Element

A *combinational device* is a processing element that has

- one or more digital *inputs* — *One of two discrete values*
- one or more digital *outputs*

Static discipline

- a *functional specification* that details the value of each output for every possible combination of valid input values
- a *timing specification* consisting (at minimum) of an upper bound $t_{pd}$ on the required time for the device to compute the specified output values from an arbitrary set of stable, valid input values



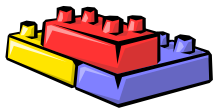



# Why have processing blocks?

- The goal of modular design:

## ABSTRACTION

- What does that mean anyway:
  - Rules simple enough for a 6-3 to follow…
  - Understanding BEHAVIOR without knowing IMPLEMENTATION
  - Predictable composition of functions
  - Tinker-toy assembly
  - Guaranteed behavior under REAL WORLD circumstances



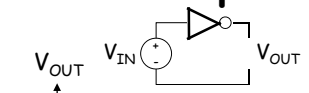


# A Combinational Digital System

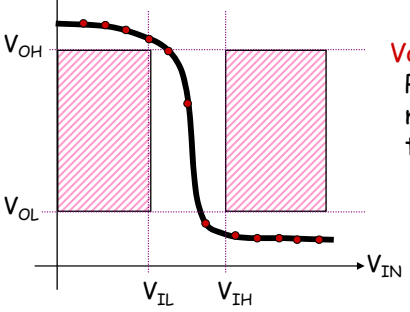
- A set of interconnected elements is a *combinational device* if
  - each circuit element is a combinational device
  - every input is connected to exactly one output or a constant (e.g., some vast supply of 0's and 1's)
  - the circuit contains no directed cycles
- Why is this true?
  - Given an acyclic circuit meeting the above constraints, we can derive functional and timing specs for the input/output behavior from the specs of its components!
  - We'll see lots of examples soon. But first, we need to build some combinational devices to work with…



# Example Device: An Inverter




0 → 1 1 → 0



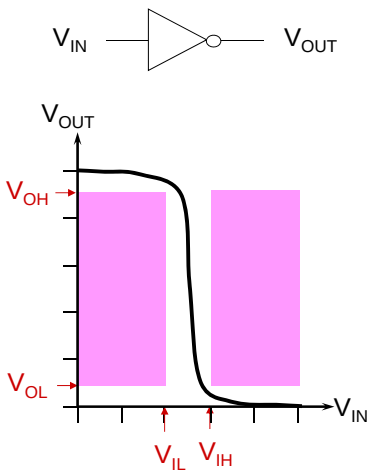

Voltage Transfer Characteristic: Plot of $V_{OUT}$ vs. $V_{IN}$ where each measurement is taken after any transients have died out.

*Note: VTC does not tell you anything about how fast a device is—it measures static behavior not dynamic behavior*

Static Discipline requires that we avoid the shaded regions (aka "forbidden zones"), which correspond to *valid* inputs but *invalid* outputs. Net result: combinational devices must have GAIN > 1 and be NONLINEAR.

$$\frac{\partial V_{OUT}}{\partial V_{IN}}$$

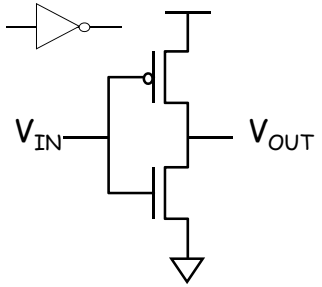
# Combinational Device Wish List

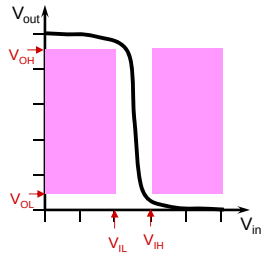


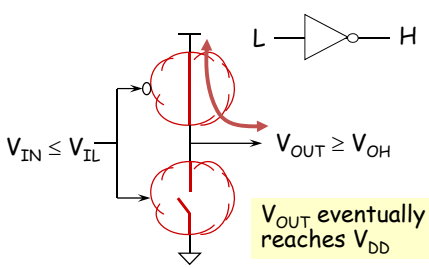

- ✓ Design our system to tolerate some amount of error
  - ⇒ Add positive noise margins
  - ⇒ VTC: gain>1 & nonlinearity
- ✓ Lots of gain ⇒ big noise margin
- ✓ Cheap, small
- ✓ Changing voltages will require us to dissipate power, but if no voltages are changing, we' d like zero power dissipation
- ✓ Want to build devices with useful functionality (what sort of operations do we want to perform?)

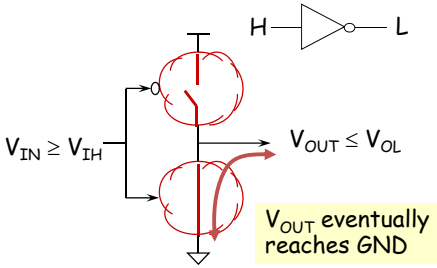


# Wishes Granted: CMOS



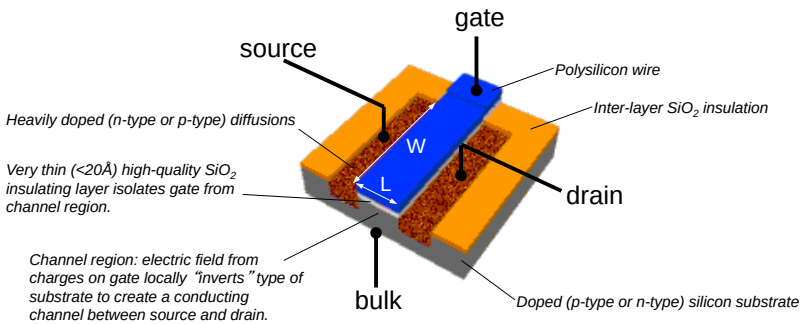






# MOSFETS: Gain & Non-linearity

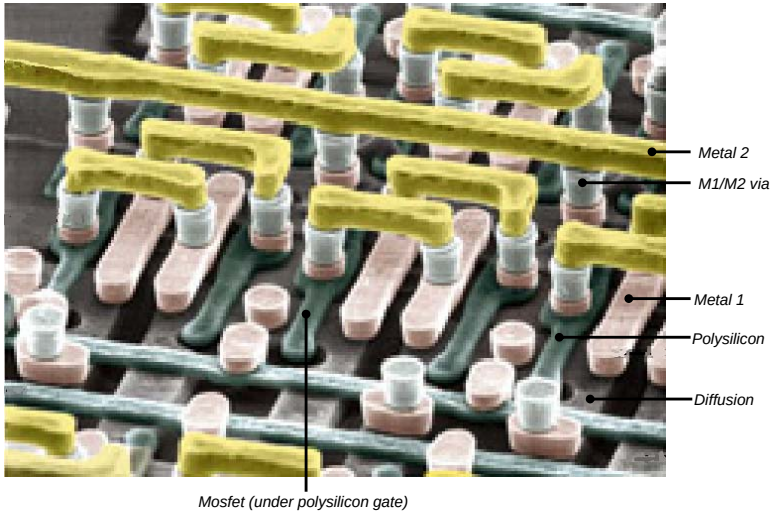



MOSFETs (metal-oxide-semiconductor field-effect transistors) are four-terminal voltage-controlled switches. Current flows between the diffusion terminals if the voltage on the gate terminal is large enough to create a conducting "channel", otherwise the mosfet is off and the diffusion terminals are not connected.

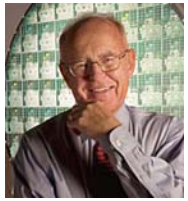


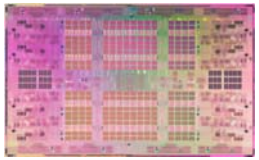
# Digital Integrated Circuits

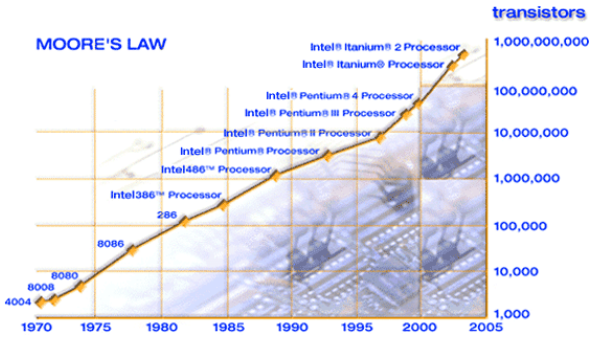
IBM photomicrograph ($SiO_2$ has been removed!)



# CMOS Forever!?








2013 Intel Poulson Processor

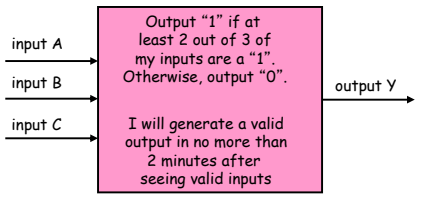
- 32nm process
- 3.1 billion transistors
- 18.2mm x 29.9mm
- 8 multithreaded cores
- 170 watts

Aug 2016 SPARC M7

- 20nm process
- 10 billion transistors
- 32 multithreaded cores
- ? watts



# Functional Specifications




| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

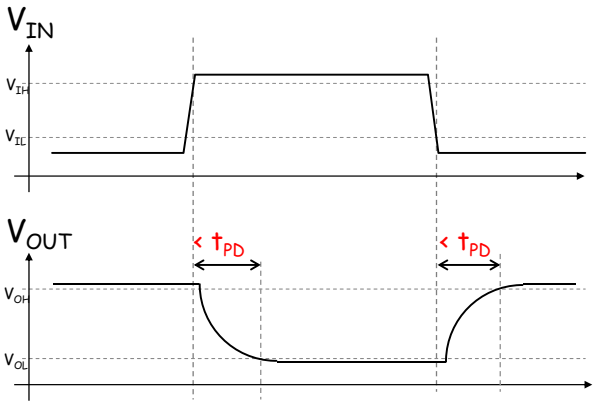
*3 binary inputs*
*so $2^3$ = 8 rows in our truth table*

An concise, unambiguous technique for giving the functional specification of a combinational device is to use a *truth table* to specify the output value for each possible combination of input values (N binary inputs -> $2^N$ possible combinations of input values).



# Timing Specifications

Propagation delay ($t_{PD}$): An *upper bound* on the delay from valid inputs to valid outputs (aka "$t_{PD,MAX}$")



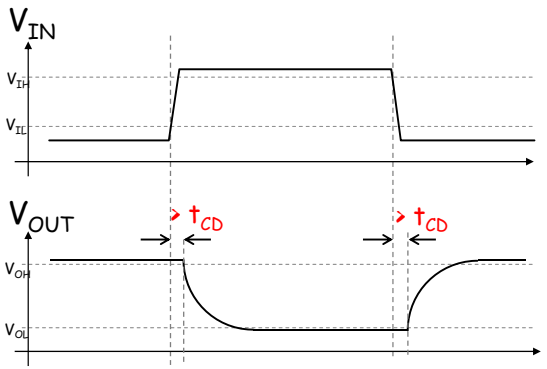

Design goal: *minimize* propagation delay



# Contamination Delay

***an optional, additional timing spec***

Contamination delay($t_{CD}$): A *lower bound* on the delay from invalid inputs to invalid outputs (aka "$t_{PD,MIN}$")




Do we really need $t_{CD}$?

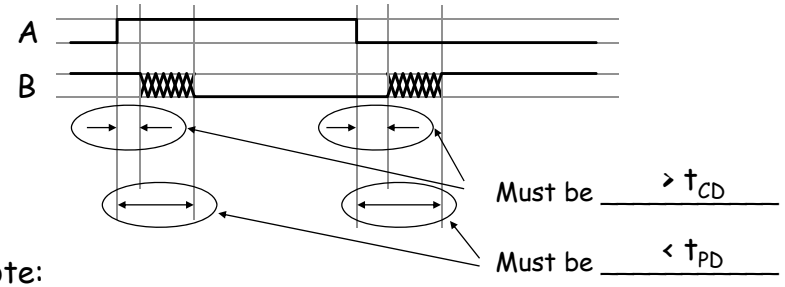
Usually not… it' ll be important when we design circuits with registers (coming soon!)

If $t_{CD}$ is not specified, safe to assume it' s 0.

# The Combinational Contract


| A | B |
|---|---|
| 0 | 1 |
| 1 | 0 |

$t_{PD}$ propagation delay
$t_{CD}$ contamination delay




Note:
1. *No Promises* during ▨
2. Default (conservative) spec: $t_{CD} = 0$



# Summary

- Use voltages to encode information
- "Digital" encoding
  - valid voltage levels for representing "0" and "1"
  - forbidden zone avoids mistaking "0" for "1" and vice versa
- Noise
  - Want to tolerate real-world conditions: NOISE.
  - Key: tougher standards for output than for input
  - devices must have gain and have a non-linear VTC
- Combinational devices
  - Each logic family has Tinkertoy-set simplicity, modularity
  - predictable composition: "parts work → whole thing works"
  - static discipline
    - digital inputs, outputs; restore marginal input voltages
    - complete functional spec, e.g., a truth table
    - valid inputs lead to valid outputs in bounded time (<$t_{PD}$)



# Tektronix Logic Analyzer -Demo

- 4 Sets of 16 channels plus clock = 68 channels
- Align probes with flying leads correctly
- Screen capture
- redundant keyboard/cursor/mouse controls
- cursor1/2 locator
- fastest sampling rate is 2ghz, magniview is 8ghz
- sampling can be clocked externally or internally (select judiciously)
- triggering modes – simple events, complex multiple events
- waveforms – customize via right mouse click: expand channels, change radix, rename, delete, add …
- Future labs will have LA directly connected via analyzer ports.



# Hand in Background Informatin