EE108A

Lecture 1:
The Digital Abstraction
Combinational Logic
Boolean Algebra
Verilog Introduction

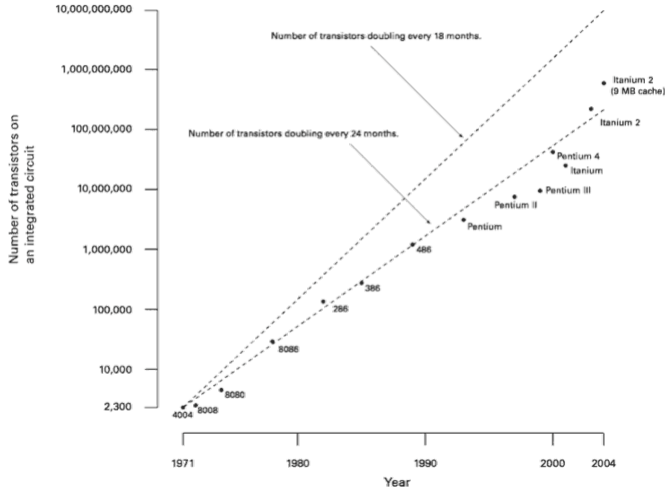
Administrivia

- Course Notes Chapters 1-12 (on web)
  - Read 1, 3, 6.1-6.4 for today
  - Read 6.5-6.11, 7 for Monday
- Handouts
  - Lecture slides
  - Homework 1
  - Lab 0
  - Course Policy Sheet
  - Lab/Section Questionnaire
- Web page http://ee108a.stanford.edu, EEClass
- E102E - writing in the major
- Photos

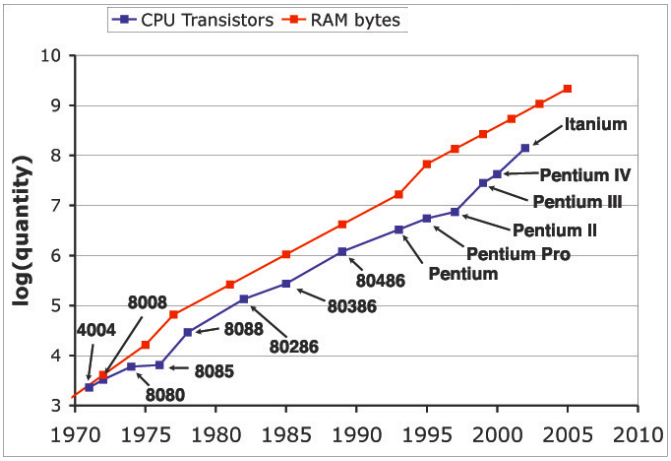
Today

- The Digital Abstraction
  - Representation
  - Noise
- Combinational Logic
  - Output function of current input
  - Acyclic composition
  - Representations - description, truth table, equation
  - Boolean algebra
- Verilog

Moore's Law

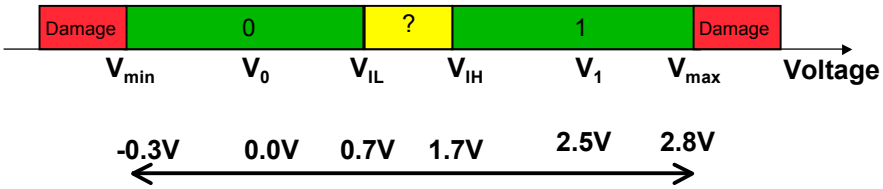


Moore's Law, Broadly Defined



The Digital Abstraction

Voltages Ranges of Binary Signals



2.5V LVCMOS Logic

Representing Information with Digital Signals

Binary Information **Light On/Off: 1/0**

Element of a set

000	White	○	100	Purple	●
001	Red	●	101	Orange	●
010	Blue	●	110	Green	●
011	Yellow	●	111	Black	●

Continuous Quantities

000	68	100	76	0000000	68	0001111	76
001	70	101	78	0000001	70	0011111	78
010	72	110	80	0000011	72	0111111	80
011	74	111	82	0000111	74	1111111	82

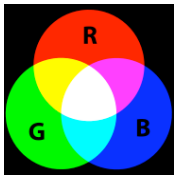
How would you represent

- A Date ?
  - Days since Jan 1 0000 (20 bits)
  - Month, day, year (4 + 5 + 12 bits)
  - Month, day, year, day of week (4 + 5 + 12 + 3 bits)
    - This representation is _____
- A playing card ?
  - Number in deck (6 bits)
  - Suit, Number (2 + 4 bits)
  - Decoded suit, decoded number (4 + 13 bits)
- A 5-card hand ?
- Pick a representation based on operations required

Example

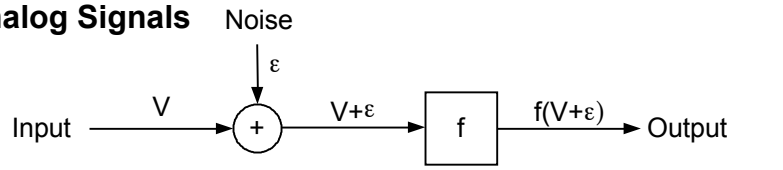
- Pick representation of colors to support the operation of subtractive color mixing (e.g., red + blue = purple)
  - No colors: white
  - Primary colors: red, blue, yellow
  - Derived colors: orange, purple, green, black

- How about an additive color model?
  - No colors: black
  - Primary: red, green, blue
  - Derived: yellow, orange, purple, green, white



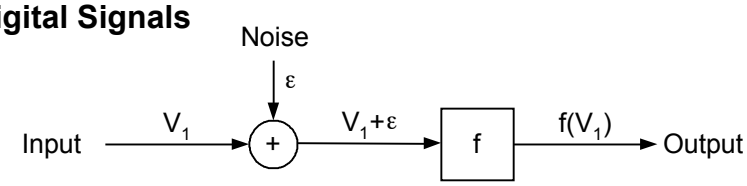
Effect of Noise on Analog & Digital Signals

Analog Signals



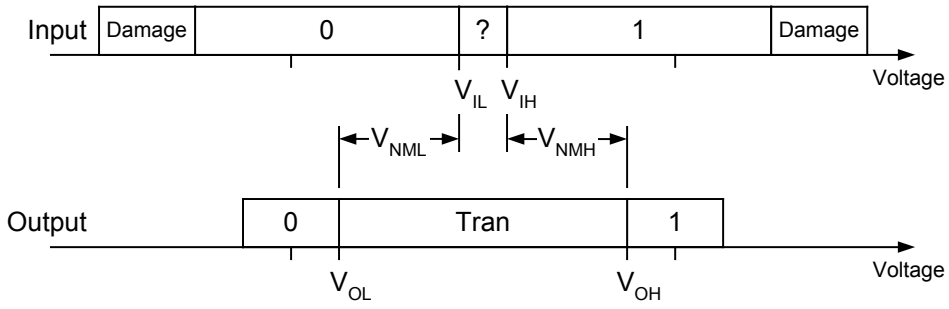
Noise adds to input **Error at output**

Digital Signals



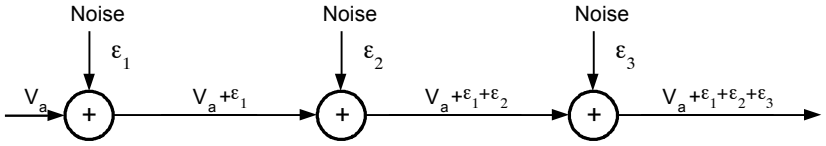
Noise added to input < noise margin **Correct value at output**

Input & Output Voltage Ranges

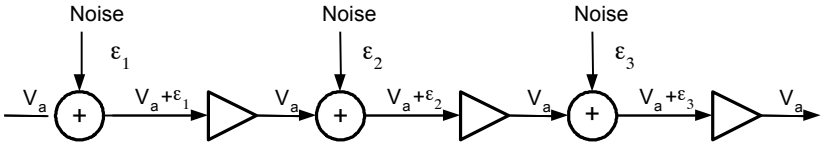


Restoration of Digital Signals

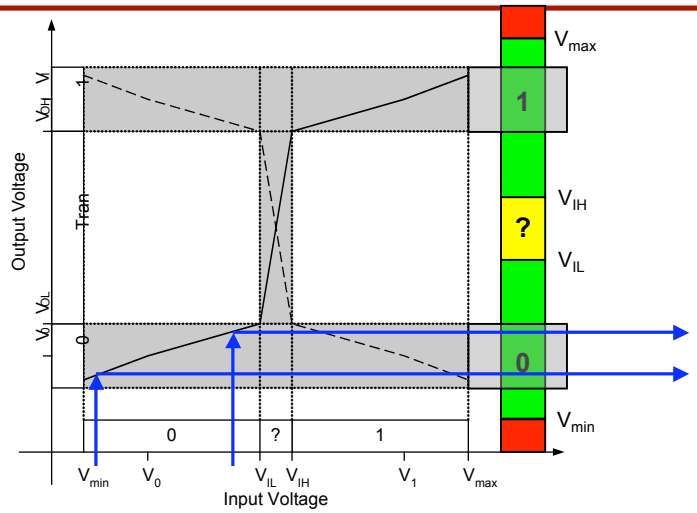
Noise Accumulation



Signal Restoration



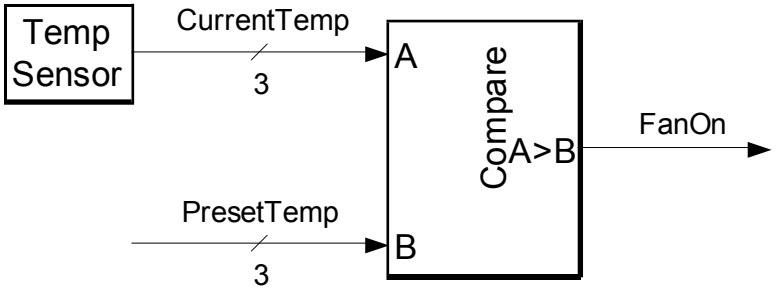
DC Transfer Curve for a Logic Module



Combinational Logic

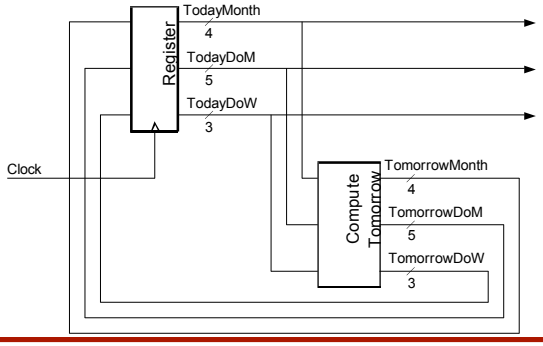
Combinational Logic Circuit

- Output is a function of current input
- Example – digital thermostat

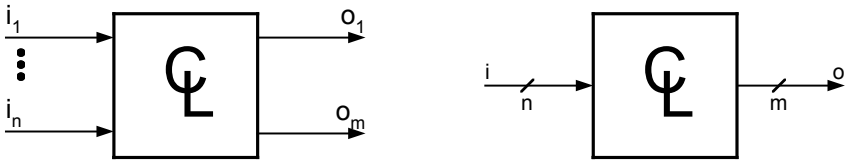


Sequential logic circuit

- Includes state (memory, storage)
- Makes output a function of history as well as current inputs
- Synchronous sequential logic uses a *clock*
- Example, calendar circuit

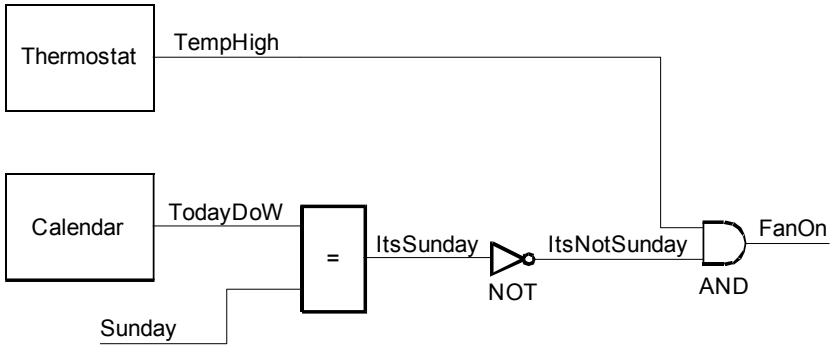


Combinational logic is memoryless



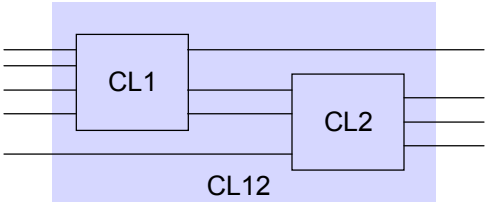
o = f(i)

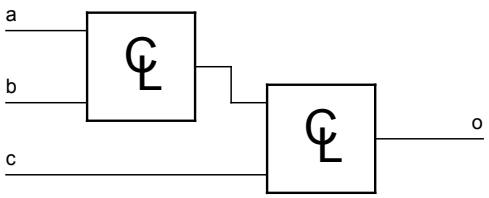
Can compose digital circuits



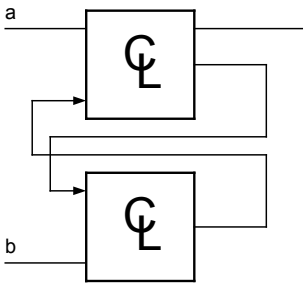
Closure

- Combinational logic circuits are closed under acyclic composition.





YES



NO

Combinational NOT Combinatorial

Combinational not Combinatorial

Combinational

↓

combinational logic circuit combines inputs to generate an output

Combinatorial

↓

mathematics of counting

Several ways to describe a combinational logic function

Example – Majority Circuit

1. English Language Description
   - Outputs 1 if more inputs are 1 than are 0
2. Logic equation
   - $q = (a \wedge b) \vee (a \wedge c) \vee (b \wedge c)$
3. Truth table

abc	q
000	0
001	0
010	0
011	1
100	0
101	1
110	1
111	1

Boolean Algebra

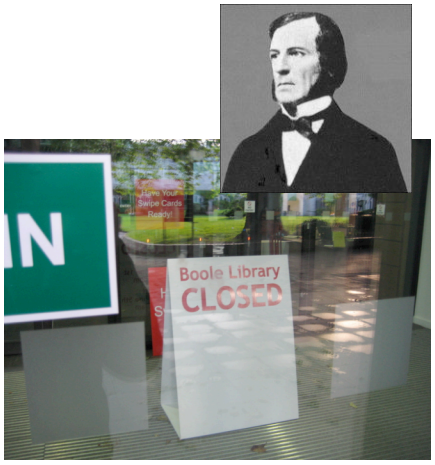
- Boolean Algebra:
  - algebra over 2 elements: {0,1}
  - 3 operators: AND (∧), OR (∨), NOT (¬)
- AND Operation:
  - a ∧ b = 1 iff a=1 and b=1
- OR Operation:
  - a ∨ b = 1 if a=1 or b=1
- NOT Operation:
  - ¬a = 1 iff a=0

a	b	a&b	a|b
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

a	~a
0	1

Note on Notation for Boolean Algebra

- For AND logic function:
  - use ∧ or &
  - don't use x or .
- For OR logic function:
  - use ∨ or |
  - don't use +



Axioms of Boolean Algebra

- *Axiom: a mathematical statement we assert to be true.*
- Identity:
  - $0 \wedge x = 0$
  - $1 \vee x = 1$
- Idempotence:
  - $1 \wedge x = x$
  - $0 \vee x = x$
- Negation:
  - $\neg 0 = 1$
  - $\neg 1 = 0$

Useful Boolean Properties (all can be derived from the axioms)

Communicative	$x \wedge y = y \wedge x$	$x \vee y = y \vee x$
Associative	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \vee (y \vee z) = (x \vee y) \vee z$
Distributive	$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$	$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$
Idempotence	$x \wedge x = x$	$x \vee x = x$
Absorption	$x \wedge (x \vee y) = x$	$x \vee (x \wedge y) = x$
Combining	$(x \wedge y) \vee (x \wedge \neg y) = x$	$(x \vee y) \wedge (x \vee \neg y) = x$
DeMorgan's	$\neg(x \wedge y) = \neg x \vee \neg y$	$\neg(x \vee y) = \neg x \wedge \neg y$

DeMorgan's Law

- $\neg(x \wedge y) = \neg x \vee \neg y$ $\quad\quad$ $\neg(x \vee y) = \neg x \wedge \neg y$
- Proof by perfect induction

x	y	¬(x ∧ y)	¬x ∨ ¬y
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

DeMorgan Graphically

Applying Boolean Properties

f(a,b,c) = (a ∧ c) ∨ (a ∧ b ∧ c) ∨ (¬a ∧ b ∧ c) ∨ (a ∧ b ∧ ¬c)

f(a,b,c) = (a ∧ c) ∨ (a ∧ b ∧ c) ∨
(¬a ∧ b ∧ c) ∨ (a ∧ b ∧ c) ∨
(a ∧ b ∧ ¬c) ∨ (a ∧ b ∧ c) ← Apply absorption property

f(a,b,c) = (a ∧ c) ∨
(¬a ∧ b ∧ c) ∨ (a ∧ b ∧ c) ∨
(a ∧ b ∧ ¬c) ∨ (a ∧ b ∧ c) ← Apply combining property

f(a,b,c) = (a ∧ c) ∨ (b ∧ c) ∨ (a ∧ b)

Verilog

Verilog

- A *hardware description language (HDL)*
- Used as an input to:
  - Synthesis – implement hardware with gates, cells, or FPGAs
  - Simulation – see what your hardware will do before you build it
- Basic unit is a *module*
- Modules have
  - Module declaration
  - Input and output declarations
  - Internal signal declarations
  - Logic definition
    - Assign statements
    - Case statements
    - Module instantiations

Example, Verilog for Thermostat

```
module Thermostat(presetTemp, currentTemp, fanOn) ;
  // temperature inputs, 3 bits each
  input [2:0] presetTemp, currentTemp ;


  // one bit, true when current > preset
  output fanOn ;

  wire fanOn ;
  assign fanOn = (currentTemp > presetTemp) ;
endmodule
```

Example, Verilog for Thermostat

Module declaration

I/O list

```
module Thermostat(presetTemp, currentTemp, fanOn) ;
  // temperature inputs, 3 bits each
  input [2:0] presetTemp, currentTemp ;

  // one bit, true when current > preset
  output fanOn ;

  wire fanOn ;
  assign fanOn = (currentTemp > presetTemp) ;
endmodule
```

Declare I/Os

3-bit wide signals

A wire is a signal set with an assign statement or connected to a module

An assign statement defines a signal with an equation

Example, Days in Month Function

```
module DaysInMonth(month, days) ;
  input [3:0] month ; // month of the year 1 = Jan, 12 = Dec
  output [4:0] days ; // number of days in month

  reg [4:0] days ;

  always @(month) begin // evaluate whenever month changes
    case(month)
      2: days = 5'd28 ;        // February has 28
      4,6,9,11: days = 5'd30 ; // 30 days have September ...
      default: days = 5'd31 ;  // all the rest have 31...
    endcase
  end
endmodule
```

Example, Days in Month Function

```
module DaysInMonth(month, days) ;
  input [3:0] m                    ar 1 = Jan, 12 = Dec
  output [4:0]                     n month

  reg [4:0] days ;

  always @(month) beg              changes
    case(month)
      2: days = 5'd28 ;
      4,6,9,11: days = 5'd30 ;
      default: days = 5'd31 ; // all the rest have 31...
    endcase
  end
endmodule
```

Reg defines a signal set in an *always* block. It does **NOT** define a register.

Always block evaluates each time activity list changes. In this case each time *month changes*

Case statement selects statement depending on value of argument. Like a truth table.

Can have multiple values per statement

Default covers values not listed. Always have one!

Verilog design style – for synthesizable modules

1. Combinational modules use only
   1. Assign statements
   2. Case or Casex statements (with default)
   3. If statements – only if all signals have a default assignment
   4. Instantiations of other combinational modules
2. Sequential modules use only
   1. Combinational logic
   2. Explicitly declared registers (flip-flops)
3. Do not use
   1. Loops
   2. Always blocks except with case or casex and If
4. Do use
   1. Signal concatenation, e.g., {a, b} = {c, d}
   2. Signal subranges, e.g., a[7:1] = b[6:0] ;
5. Organize logic into small modules
   1. Leaf modules not more than 40 lines
   2. If it could be made two modules, it should be

Verilog design style – for synthesizable modules (page 2)

6. Use lots of comments
   1. Comments themselves
   2. Meaningful signal names – tempHigh, not th
   3. Meaningful module names – DaysInMonth, not Mod3
7. Activation lists for case statements include **ALL** inputs
8. Constants
   1. All constants are `defined if used more than once
   2. Width of all constants is specified 5'd31, not 31
9. Signals
   1. Buses (multi-bit signals) are numbered high to low
      - e.g., wire [31:0] bus
   2. All signals should be high-true (except primary inputs and outputs)
10. Visualize the logic your Verilog will generate.
    - If you can't visualize it, the result will not be pretty

Make your code easy to read, understand, and reason about.

Summary

- The world is digital
  - Systems are digital at their core, only analog on the edges
- Representing information
  - Binary signals, sets, continuous values
- Digital signals reject noise
  - Voltage range divided into discrete states
  - Noise margins
- Combinational logic
  - Output depends only on input (no state)
- Boolean algebra
  - Rules to manipulate logic equations
- Verilog
  - Describe hardware for simulation and synthesis