Welcome to 6.111!

- Introductions, course mechanics
- Course overview
- Digital signaling
- Combinational logic

Handouts
- lecture slides,
- LPset #1,
- info form

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

6.111 Introductory Digital Systems Lab

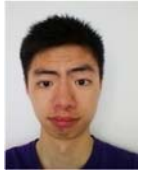
- Learn digital systems through lectures, labs and a final project using FPGAs and Verilog.
- Gain experience with hands on course with real hardware – nothing in the cloud.
- 6.111 satisfies
  - Course 6: DLAB2, II, EECS, AUS2
  - Course 16: Laboratory requirement, Professional Area subject

Introductions

Gim Hom
Lectures

Joe Steinmeyer
Lectures

Sarah Flanagan
TA


Mike Wang
TA

Diana Wofk
TA

Sam Cherna

Isabelle Liu

Lydia Sun

Mark Theng

August Trollback

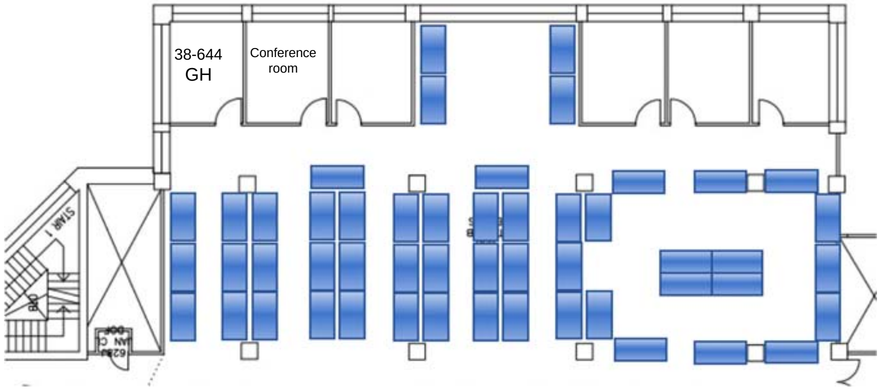
LA's

38-600

Lab hours:
S 1-11:45p
M-R 9-11:45p
F 9-5p

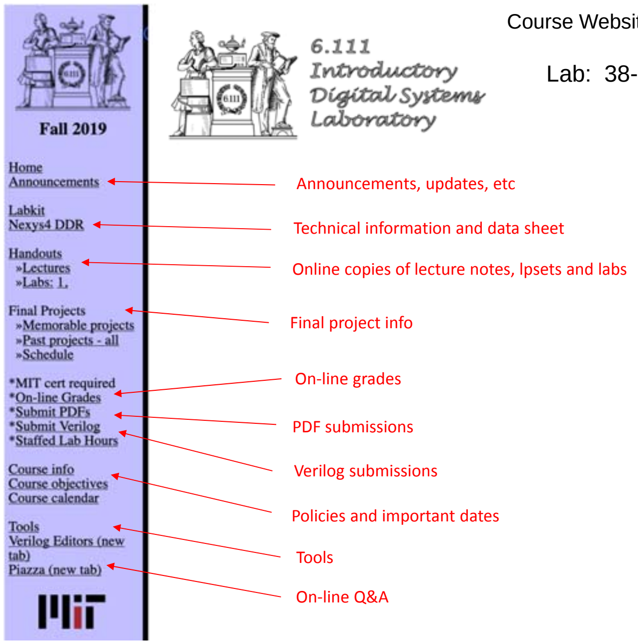
← Stata

49 Stations



Course Website: web.mit.edu/6.111

Lab: 38-600



6.111 Introductory Digital Systems Laboratory

Fall 2019

Home
Announcements

Labkit
Nexys4 DDR

Handouts
»Lectures
»Labs: 1,

Final Projects
»Memorable projects
»Past projects - all
»Schedule

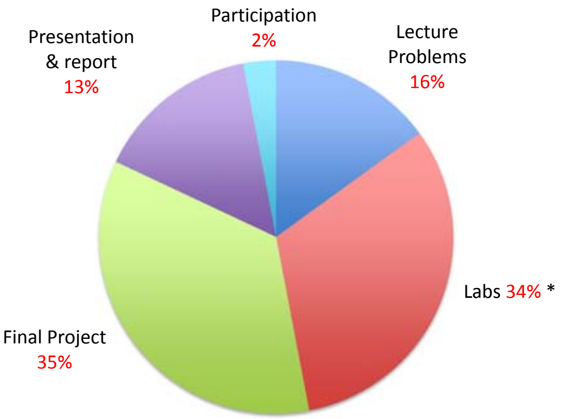
*MIT cert required
*On-line Grades
*Submit PDFs
*Submit Verilog
*Staffed Lab Hours

Course info
Course objectives
Course calendar

Tools
Verilog Editors (new tab)
Piazza (new tab)

Announcements, updates, etc
Technical information and data sheet
Online copies of lecture notes, lpsets and labs
Final project info
On-line grades
PDF submissions
Verilog submissions
Policies and important dates
Tools
On-line Q&A

Assignments - Grading



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.

Labs: learning the ropes

- Lab 1 (2%)
  - Experiment with gates, design & implement some logic with FPGA
  - Learn about simulation and using Verilog design suite: Vivado
- Lab 2 (5%)
  - Introduction to clocks, counters and more hardware
  - Serial communications
- Lab 3 (8%)
  - Video circuits: a simple Pong game
    - Use Verilog to program an FPGA
    - Display images
- Lab 4 (11%)
  - Design and implement a Finite State Machine (FSM) – Car Alarm
- Lab 5 (8%)
  - Design a complicated system with multiple FSMs (Major/Minor FSM)
    - Audio processing
    - Digital bubble level using IMU

- You must have a *non-zero* score for each of the labs and all the labs must be checked off as a prerequisite for passing the course. A missing lab will result in a failing grade.

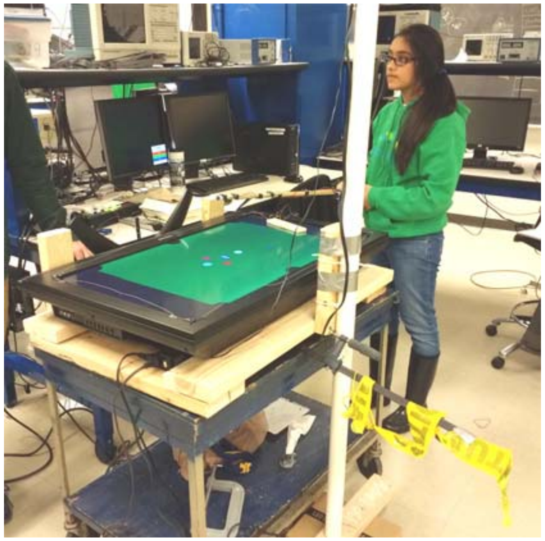
Final Project

- Grading: 35%
- 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
  - Have fun!
- 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.

Project Presentation & Report (13%)

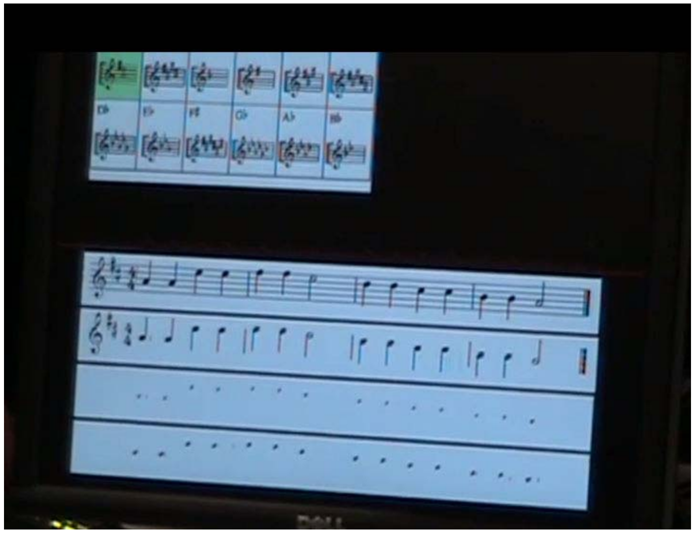
- Design proposal   (2%)
- Design presentation (6%)
- Final Report (5%)

Virtual Pool – La PC Na



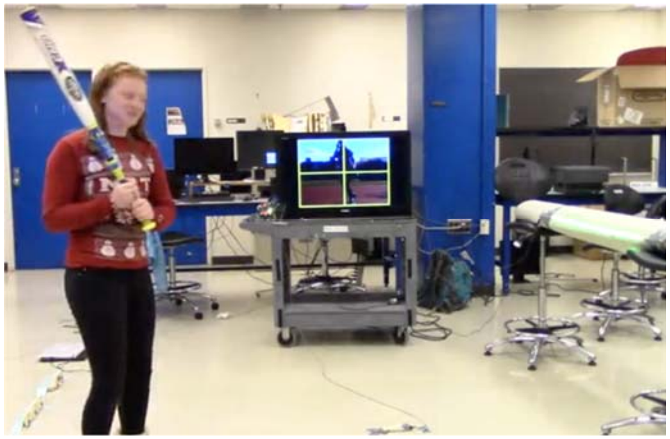
Fall 2016
Matt Basile
Zareen Choudhury

FPGA Beethoven



Fall 2016: Henry Love, Mark Yang

Virtual Softball



Fall 2017 Katherine Shade, Melinda Szabo

FPGA Passport

Fall 2016 Lorenzo Vigano, Diana Wofk

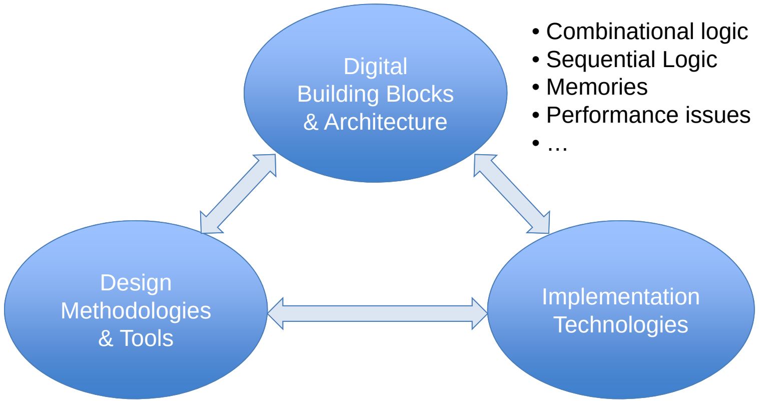
Gesture Controlled Drone

Fall 2014 Lee Gross, Ben Schrenk

Collaboration

- Labs and lpset must be done independently but students may seek help from other students and of course staff.
- Work submitted for review must be your own

6.111 Topics



Digital Building Blocks & Architecture

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

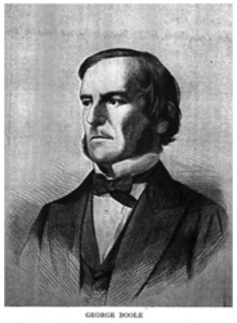
Design Methodologies & Tools

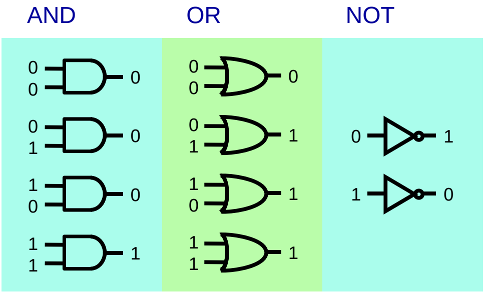
Implementation Technologies

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

- FPGAs
- Flash, ram
- ADC, DAC,
- Sensors …

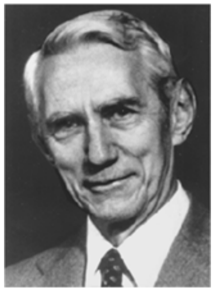
Boolean Algebra

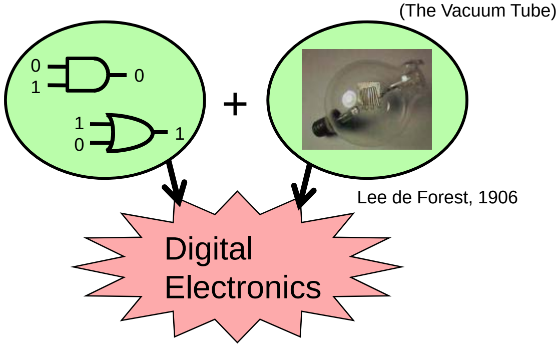




- 1854: George Boole shows that logic is math, not just philosophy!
- Boolean algebra: the mathematics of binary values

Key Link Between Logic and Circuits




(The Vacuum Tube)

Lee de Forest, 1906

Digital Electronics

- 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

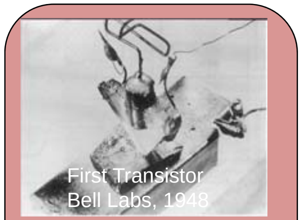
Vacuum Tubes

ENIAC, 1946

UNIVAC, 1951

1900 adds/sec

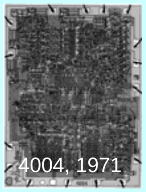
Transistors

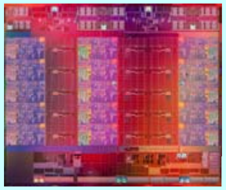

First Transistor
Bell Labs, 1948

IBM System/360, 1964

500,000 adds/sec

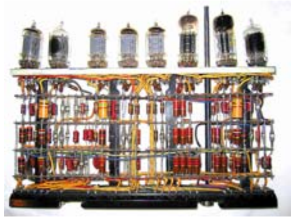
VLSI Circuits

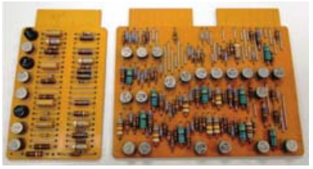

4004, 1971


Intel Cascade Lake
2018 - 22 Cores
>>7 Billion 14nm

Digital Systems Thru the Ages

Vacuum tube
computer
Circa 1950




IBM 1401 Computer
Circa 1962

DEC PDP 11
Circa 1980

6.111 Thru the Ages

Lab kit 1990 aka “digital death”

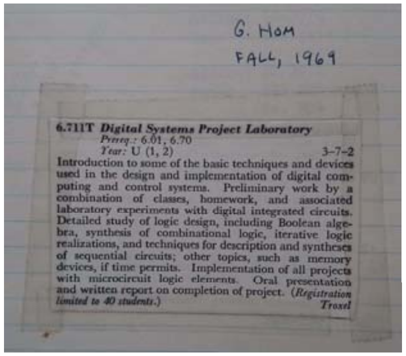
Labkit 2005

Nexys 4 - 2016

PYNQ - 2019

6.111 Evolution

Fall 1969

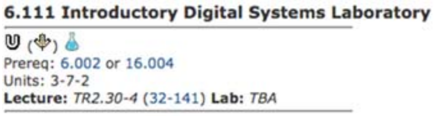


G. Hom
FALL, 1969

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.*)
Troxel

Fall 2019

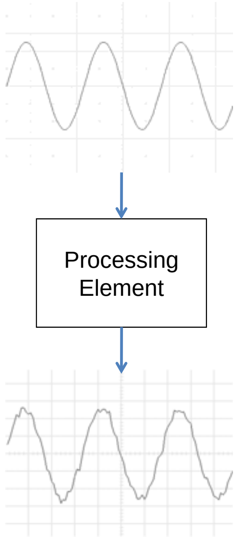


6.111 Introductory Digital Systems Laboratory

Prereq: 6.002 or 16.004
Units: 3-7-2
Lecture: *TR2.30-4* (32-141) **Lab:** *TBA*

Introduces digital systems with lectures and labs on logic, flip flops, FPGAs, counters, timing, synchronization, and finite-state machines. Includes overview of accelerometers, gyros, time of light and other modern sensors. Prepares 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 trouble with analog signaling



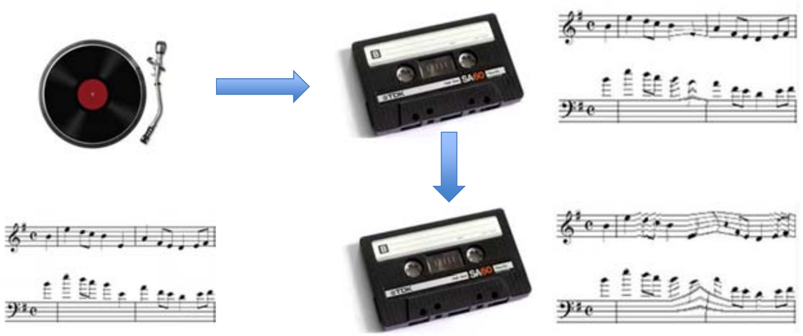
Processing Element

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, …

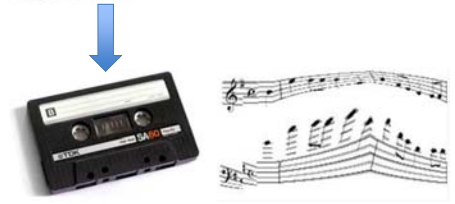
But if we build processing elements to manipulate these signals we must use non-ideal components in real-world environments, so some amount of error (aka “noise”) is introduced. The error comes from component tolerances, electrical phenomenon (e.g., IR and Ldl/dt effects), transmission losses, thermal 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.

Music at MIT circa 1970s



Need an architecture that is noise tolerant, inexpensive, reproducible.



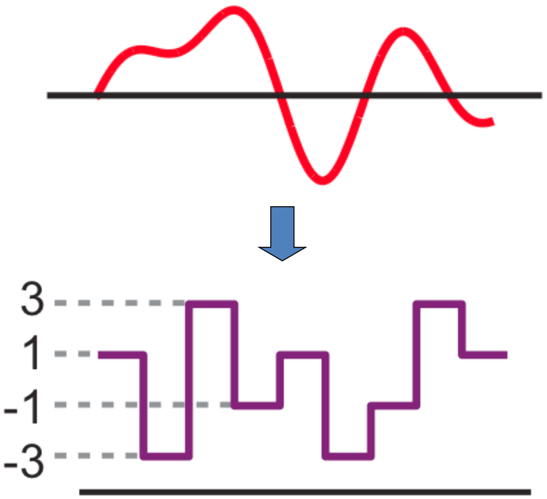
Solution: go digital!

Continuous values
Continuous time

So we can detect small changes and restore original values

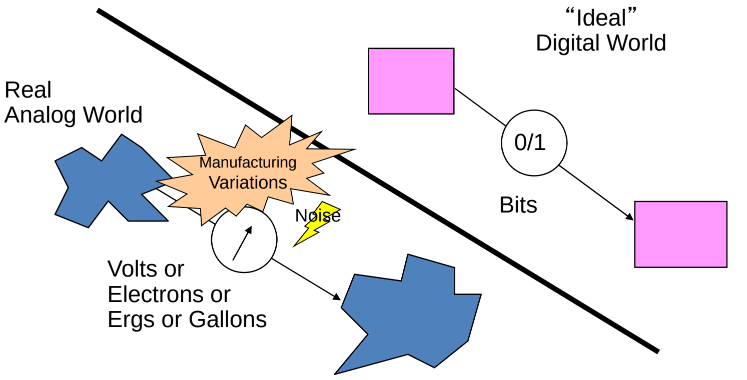
Discrete values
Discrete time

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.



"Ideal" Digital World

Real Analog World

Manufacturing Variations

Noise

0/1

Bits

Volts or Electrons or Ergs or Gallons

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 Encoding

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

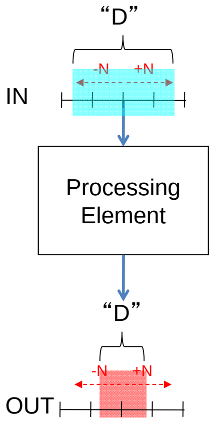
voltages	phase
currents	frequency

For 6.111, we'll use voltages to encode information. Current, phase and frequency encoding have uses in other applications.

Why voltage?
easily generated, well understood
historically used, lots of circuits
with CMOS, almost zero power in steady state

No free lunch:
noise sensitivity, non-ideal wires (RC time constant),

Digital processing elements



"D"

IN

-N +N

Processing Element

"D"

OUT

-N +N

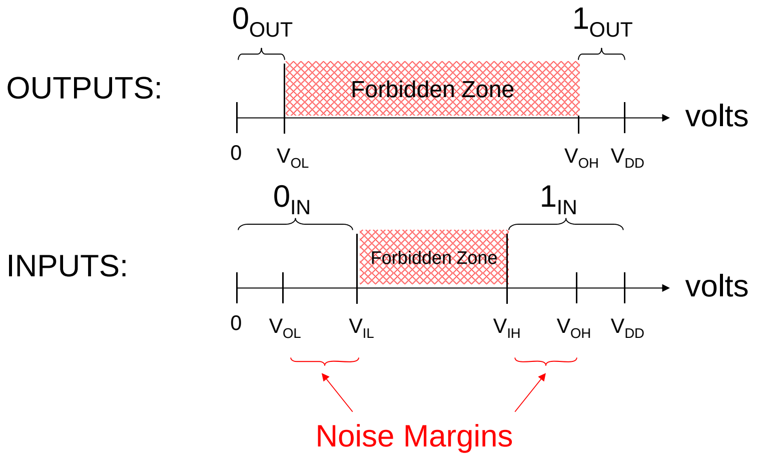
Digital processing elements restore noisy input values to legal output values – signaling errors don't accumulate in digital systems. So the number of processing elements isn't limited by noise problems!

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.

We'll keep things simple by designing our processing elements to use voltages to encode binary values (0 or 1).

Using voltages to encode binary values

To avoid hard-to-make decisions at the boundaries between voltages, insert a “forbidden zone” between levels. 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 $V_{IN} > V_{IH}$

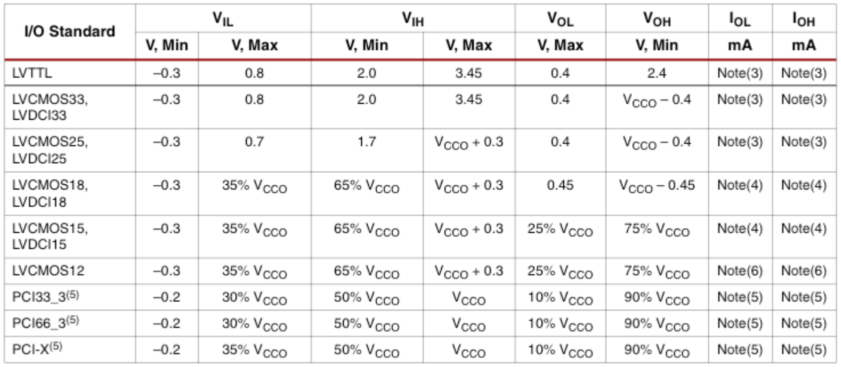
Digital output: $V_{OUT} < V_{OL}$ or $V_{OUT} > V_{OH}$

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.

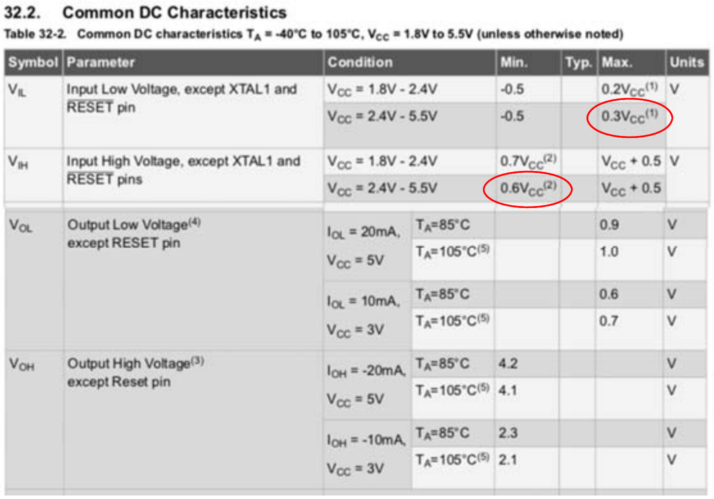
Sample DC (signaling) Specification



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

Arduino Processor DC Specification



32.2. Common DC Characteristics

Table 32-2. Common DC characteristics T_A = -40°C to 105°C, V_{CC} = 1.8V to 5.5V (unless otherwise noted)

Symbol	Parameter	Condition		Min.	Typ.	Max.	Units
V_{IL}	Input Low Voltage, except XTAL1 and $\overline{RESET}$ pin	V_{CC} = 1.8V - 2.4V		-0.5		$0.2V_{CC}$(1)	V
		V_{CC} = 2.4V - 5.5V		-0.5		$0.3V_{CC}$(1)	
V_{IH}	Input High Voltage, except XTAL1 and $\overline{RESET}$ pins	V_{CC} = 1.8V - 2.4V		$0.7V_{CC}$(2)		V_{CC} + 0.5	V
		V_{CC} = 2.4V - 5.5V		$0.6V_{CC}$(2)		V_{CC} + 0.5	
V_{OL}	Output Low Voltage(4) except RESET pin	I_{OL} = 20mA, V_{CC} = 5V	T_A=85°C			0.9	V
			T_A=105°C(5)			1.0	V
		I_{OL} = 10mA, V_{CC} = 3V	T_A=85°C			0.6	V
			T_A=105°C(5)			0.7	V
V_{OH}	Output High Voltage(3) except Reset pin	I_{OH} = -20mA, V_{CC} = 5V	T_A=85°C	4.2			V
			T_A=105°C(5)	4.1			V
		I_{OH} = -10mA, V_{CC} = 3V	T_A=85°C	2.3			V
			T_A=105°C(5)	2.1			V

Source: ATmega328P Datasheet

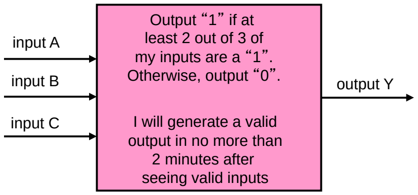
A Digital Processing Element

A combinational device is a processing element that has

Static discipline

- – one or more digital inputs
- – one or more digital outputs
- – 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

One of two discrete values



Output "1" if at least 2 out of 3 of my inputs are a "1". Otherwise, output "0".

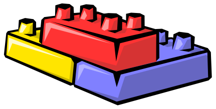
I will generate a valid output in no more than 2 minutes after seeing valid inputs

Why have processing blocks?

- The goal of modular design:

ABSTRACTION

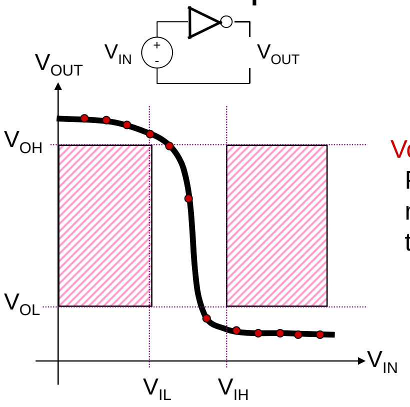
- What does that mean anyway:
  - – Rules simple enough for a 6-3 to follow…
  - – Understanding BEHAVIOR without knowing IMPLEMENTATION
  - – Predictable composition of functions
  - – Tinker-toy assembly
  - – Guaranteed behavior under REAL WORLD circumstances



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



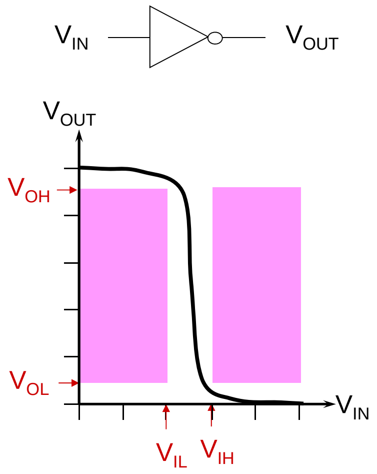
Voltage Transfer Characteristic:
Plot of V_{OUT} vs. V_{IN} where each measurement is taken after any transients have died out.

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.

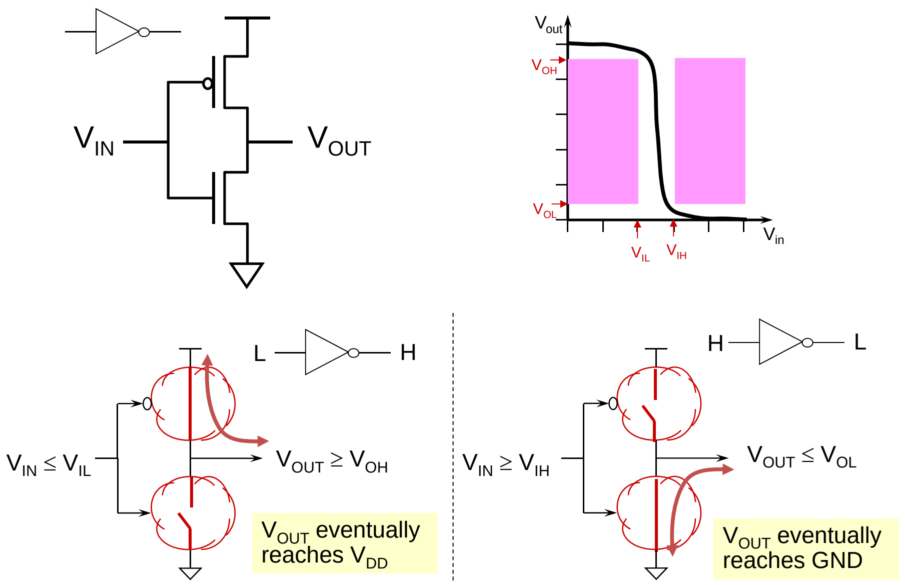
$$\frac{\partial V_{OUT}}{\partial V_{IN}}$$

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?)

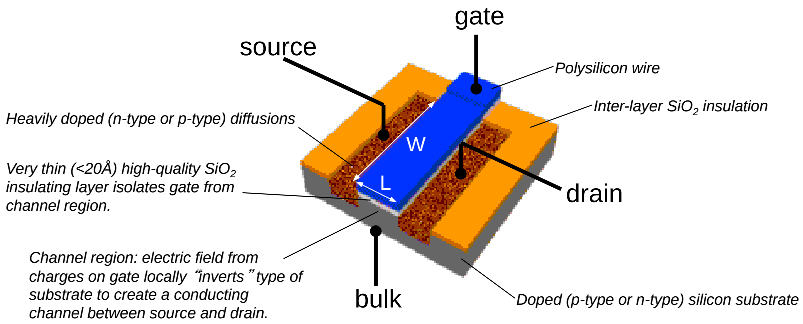
Wishes Granted: CMOS



V_{OUT} eventually reaches V_{DD}

V_{OUT} eventually reaches GND

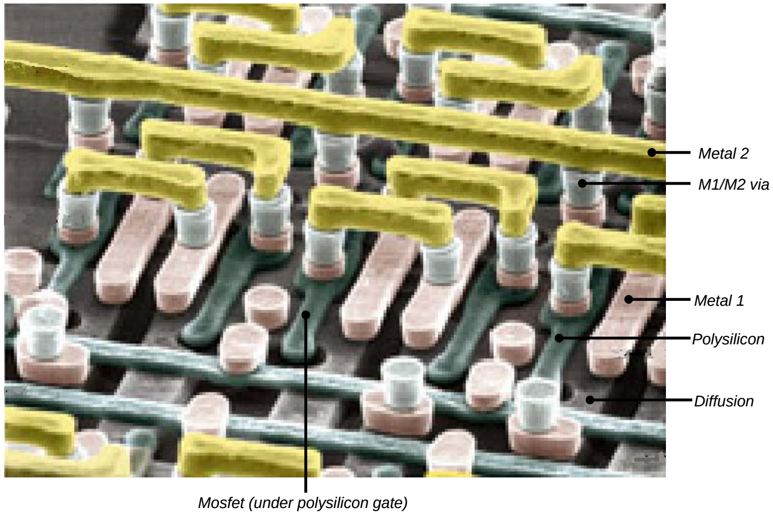
MOSFETS: Gain & Non-linearity


gate
source
Polysilicon wire
Inter-layer SiO_2 insulation
Heavily doped (n-type or p-type) diffusions
W
L
drain
Very thin (<20Å) high-quality SiO_2 insulating layer isolates gate from channel region.
Channel region: electric field from charges on gate locally "inverts" type of substrate to create a conducting channel between source and drain.
bulk
Doped (p-type or n-type) silicon substrate

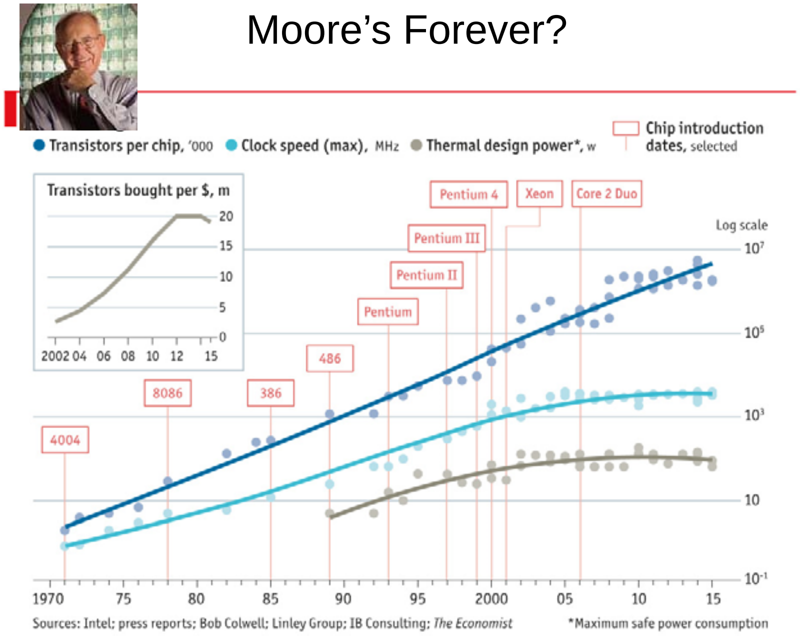
MOSFETs (metal-oxide-semiconductor field-effect transistors) are four-terminal voltage-controlled switches. Current flows between the diffusion terminals if the voltage on the gate terminal is large enough to create a conducting "channel", otherwise the mosfet is off and the diffusion terminals are not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

Digital Integrated Circuits

IBM photomicrograph (SiO_2 has been removed!)



Moore's Forever?

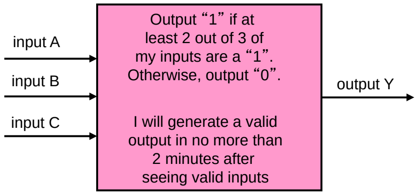


Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; *The Economist*

*Maximum safe power consumption

Economist March 12-18, 2016

Functional Specifications



Output "1" if at least 2 out of 3 of my inputs are a "1". Otherwise, output "0".

I will generate a valid output in no more than 2 minutes after seeing valid inputs

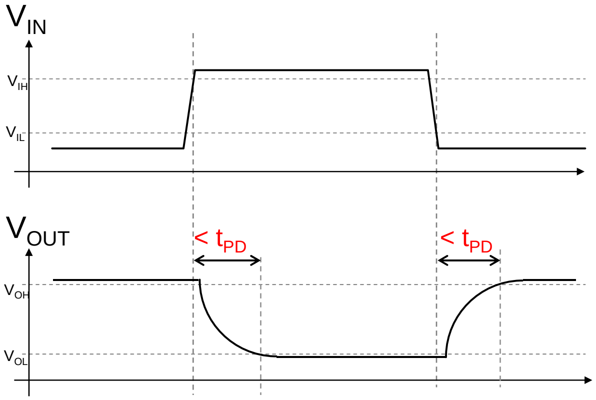
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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 <u>upper bound</u> on the delay from valid inputs to valid outputs (aka "$t_{PD,MAX}$")

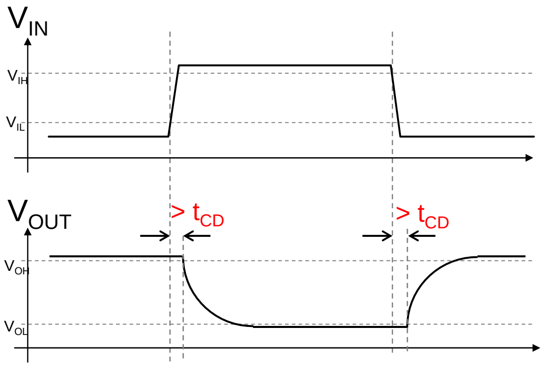


Design goal:
minimize propagation delay

Contamination Delay

an optional, additional timing spec

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

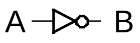


Do we really need t_{CD}?

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

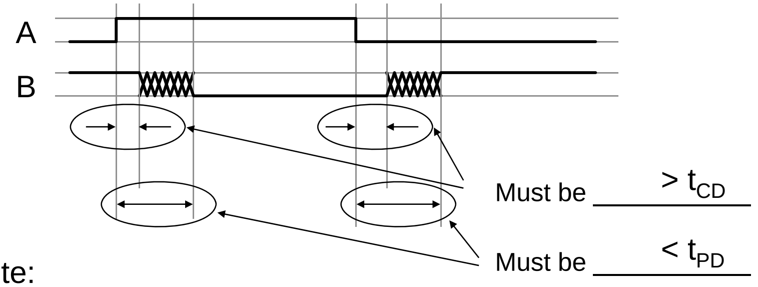
If t_{CD} is not specified, safe to assume it' s 0.

The Combinational Contract



A	B
0	1
1	0

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



Must be $> t_{CD}$

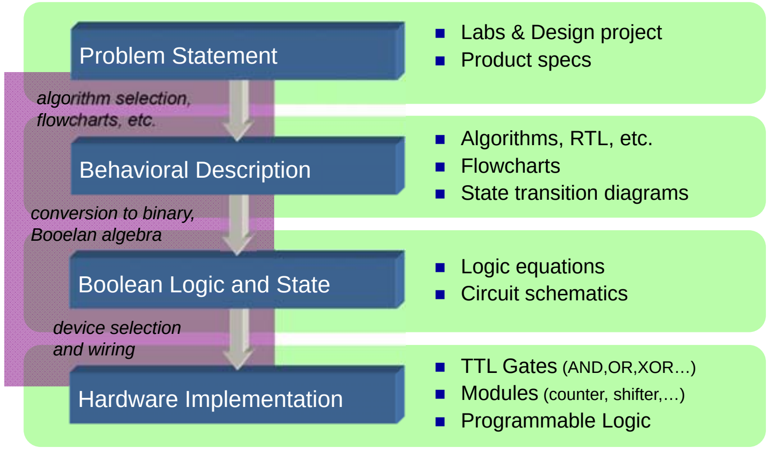
Must be $< t_{PD}$

Note:

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

Building Digital Systems

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



Problem Statement
- Labs & Design project
- Product specs

algorithm selection, flowcharts, etc.

Behavioral Description
- Algorithms, RTL, etc.
- Flowcharts
- State transition diagrams

conversion to binary, Booelan algebra

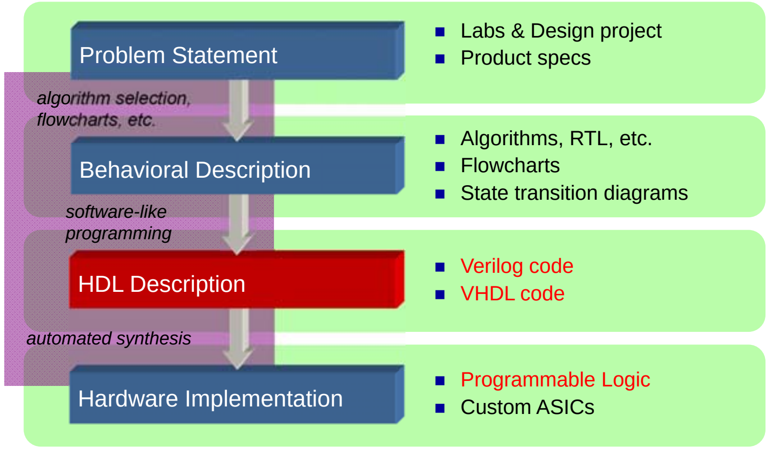
Boolean Logic and State
- Logic equations
- Circuit schematics

device selection and wiring

Hardware Implementation
- TTL Gates (AND,OR,XOR...)
- Modules (counter, shifter,...)
- Programmable Logic

Building Digital Systems with HDLs

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



Problem Statement
- Labs & Design project
- Product specs

algorithm selection, flowcharts, etc.

Behavioral Description
- Algorithms, RTL, etc.
- Flowcharts
- State transition diagrams

software-like programming

HDL Description
- Verilog code
- VHDL code

automated synthesis

Hardware Implementation
- Programmable Logic
- Custom ASICs

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

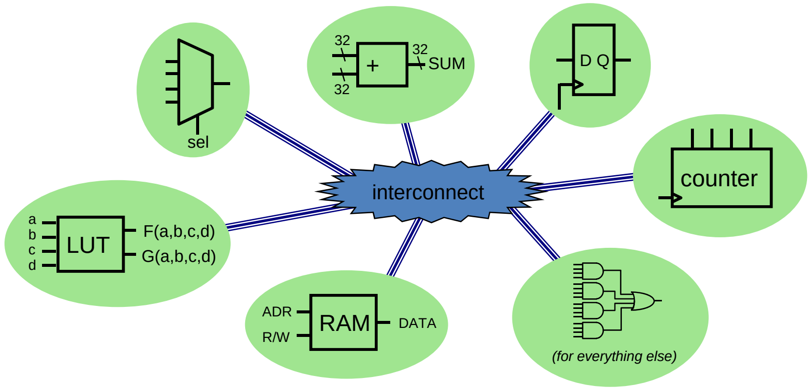
Hardware structures can be modeled effectively in either VHDL and Verilog. Verilog is similar to c and a bit easier to learn.

Verilog HDL

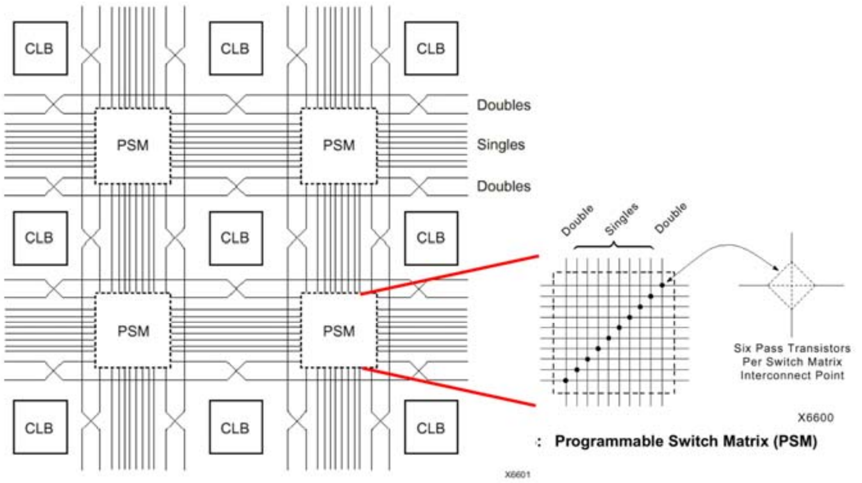
- Misconceptions
  - The coding style or clarity does not matter as long as it works
  - Two different Verilog encodings that simulate the same way will synthesize to the same set of gates
  - Synthesis just can’t be as good as a design done by humans
    - Shades of assembly language versus a higher 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 CLB (configurable logic blocks) and used to build logic blocks.



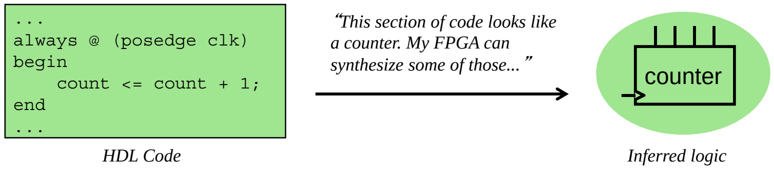
Xilinx FPGA Interconnect Details



Why six transistors?

Synthesis and Mapping for FPGAs

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



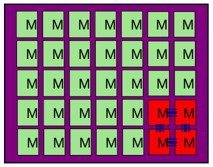
```
...
always @ (posedge clk)
begin
   count <= count + 1;
end
...
```

“This section of code looks like a counter. My FPGA can synthesize some of those...”

HDL Code

Inferred logic

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



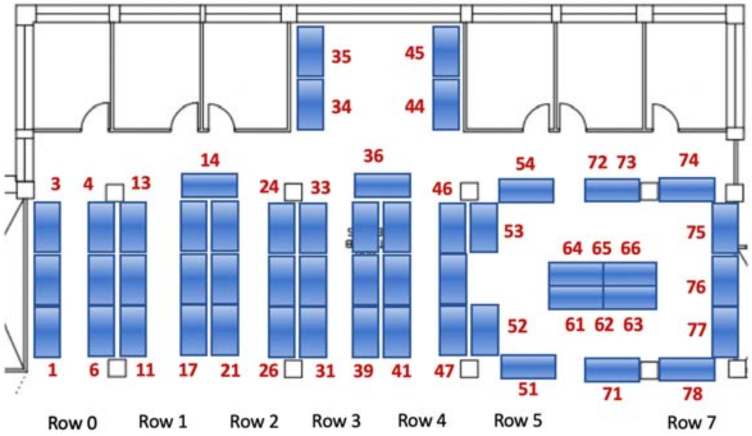
“This design only uses 10% of the FPGA. Let’s use the CLB in one corner to minimize the distance between blocks.”

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 ($<t_{PD}$)

Lab Checkoff

- Lab check off will be in two groups: Thu or Fri.
- Thu group indicated by @G1 or @G2 in grade sheet



Lecture 1 Part 2

Quick SystemVerilog and Vivado Overview for Lab 1

6.111 Thru the Ages

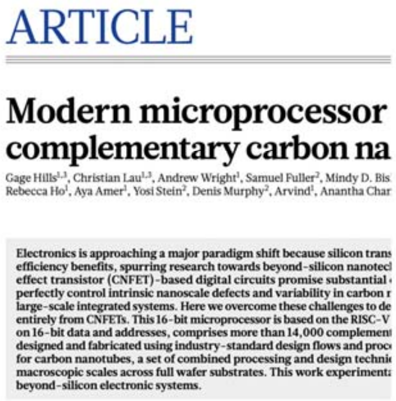
Lab kit 1990 aka “digital death”

Labkit 2005

Nexys 4 - 2016

PYNQ - 2019***

Things Change….

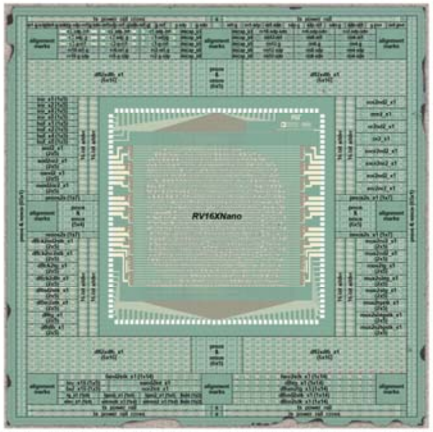


ARTICLE

Modern microprocessor complementary carbon na

Gage Hills[1,3], Christian Lau[1,3], Andrew Wright[1], Samuel Fuller[2], Mindy D. Bis
Rebecca Ho[1], Aya Amer[1], Yosi Stein[2], Denis Murphy[2], Arvind[1], Anantha Char

Electronics is approaching a major paradigm shift because silicon trans efficiency benefits, spurring research towards beyond-silicon nanotecl effect transistor (CNFET)-based digital circuits promise substantial perfectly control intrinsic nanoscale defects and variability in carbon r large-scale integrated systems. Here we overcome these challenges to de entirely from CNFETs. This 16-bit microprocessor is based on the RISC-V on 16-bit data and addresses, comprises more than 14,000 complement designed and fabricated using industry-standard design flows and proc for carbon nanotubes, a set of combined processing and design techni macroscopic scales across full wafer substrates. This work experiment beyond-silicon electronic systems.



Extended Data Fig. 2 | Microscopy image of a full fabricated RV16X-NANO die. The processor core is in the middle of the die, with test circuitry surrounding the perimeter (when the RV16X-NANO is diced for packaging, these test structures are removed). The test structures include test structures for monitoring fabrication, as well as for measuring and characterizing all of the 63 standard cells in our standard cell library.

https://www.nature.com/articles/s41586-019-1493-8

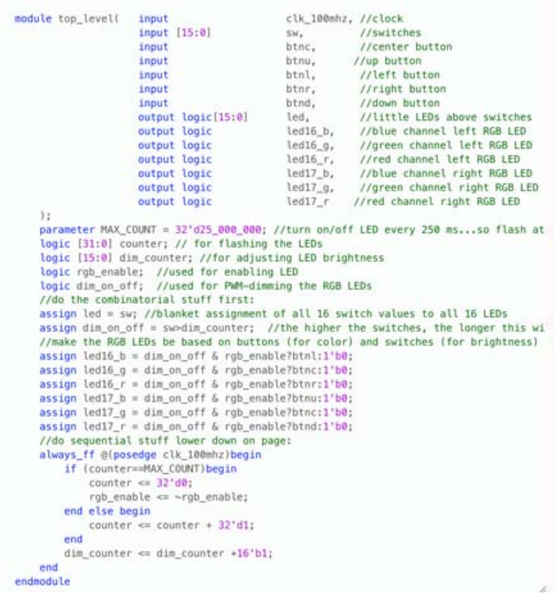
Vivado Demo

https://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/vivado_quickstart/quickstart.html

6.111
Simple Demo for Vivado Setup
Fall 2019

What Controls This?

- SystemVerilog interpreted with Vivado



```
module top_level(   input                   clk_100mhz, //clock
                    input [15:0]            sw,         //switches
                    input                   btnc,       //center button
                    input                   btnu,      //up button
                    input                   btnl,       //left button
                    input                   btnr,       //right button
                    input                   btnd,       //down button
                    output logic[15:0]      led,        //little LEDs above switches
                    output logic            led16_b,    //blue channel left RGB LED
                    output logic            led16_g,    //green channel left RGB LED
                    output logic            led16_r,    //red channel left RGB LED
                    output logic            led17_b,    //blue channel right RGB LED
                    output logic            led17_g,    //green channel right RGB LED
                    output logic            led17_r    //red channel right RGB LED
    );
    parameter MAX_COUNT = 32'd25_000_000; //turn on/off LED every 250 ms...so flash at
    logic [31:0] counter; // for flashing the LEDs
    logic [15:0] dim_counter; //for adjusting LED brightness
    logic rgb_enable;  //used for enabling LED
    logic dim_on_off;  //used for PWM-dimming the RGB LEDs
    //do the combinatorial stuff first:
    assign led = sw; //blanket assignment of all 16 switch values to all 16 LEDs
    assign dim_on_off = sw>dim_counter;  //the higher the switches, the longer this wi
    //make the RGB LEDs be based on buttons (for color) and switches (for brightness)
    assign led16_b = dim_on_off & rgb_enable?btnl:1'b0;
    assign led16_g = dim_on_off & rgb_enable?btnc:1'b0;
    assign led16_r = dim_on_off & rgb_enable?btnr:1'b0;
    assign led17_b = dim_on_off & rgb_enable?btnu:1'b0;
    assign led17_g = dim_on_off & rgb_enable?btnc:1'b0;
    assign led17_r = dim_on_off & rgb_enable?btnd:1'b0;
    //do sequential stuff lower down on page:
    always_ff @(posedge clk_100mhz)begin
        if (counter==MAX_COUNT)begin
            counter <= 32'd0;
            rgb_enable <= ~rgb_enable;
        end else begin
            counter <= counter + 32'd1;
        end
        dim_counter <= dim_counter +16'b1;
    end
endmodule
```

"Verilog" or "SystemVerilog"?

- SystemVerilog is a superset of Verilog (just like C++ is a superset of C)
- We will learn/use SystemVerilog (.sv files), **however** ~90% of what we do is just Verilog!
- We're transitioning to this for 2019 since this is how the field is. It also opens a lot more doors for self-study:
  - SystemVerilog is pretty nice so maybe you'll arrive organically at needing some of its behaviors in the project

Online Verilog/SystemVerilog Resources

- There are books, but you can figure out most of it from:
  - Doing the labs (motivation to have to figure it out)
  - Formal-ish online resources (best free tutorials):
    - http://www.asic-world.com/verilog/veritut.html
    - http://www.asic-world.com/systemverilog/tutorial.html
  - Informal (StackOverflow)

How to Think About Designing Logic

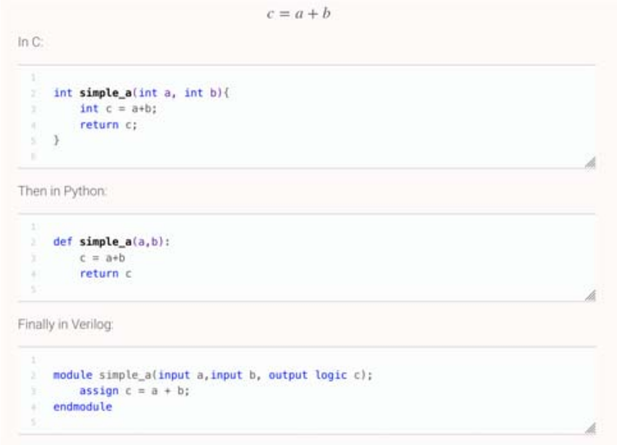
- A piece of software/program should be thought of as a set of **instructions** that are executed roughly sequentially
- You describe what to **do** with software

How to Think About Designing Logic

- A design in a Hardware Description Language (HDL) describes the wiring and placement of components. Think of it is specifying a **blueprint** for what you want to build
- You describe what to **build** with HDL
- *Everything you describe exists in parallel (sort of)*

Three Languages, Similar Functionality

- Verilog allows us to design hardware in ways similar to how we write programs:



$c = a + b$

In C:

```c
int simple_a(int a, int b){
    int c = a+b;
    return c;
}
```

Then in Python:

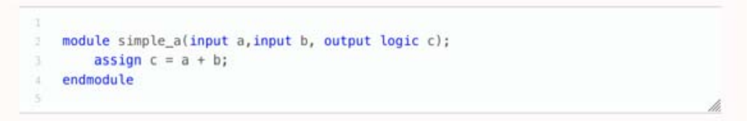
```python
def simple_a(a,b):
    c = a+b
    return c
```

Finally in Verilog:

```verilog
module simple_a(input a,input b, output logic c);
    assign c = a + b;
endmodule
```

Verilog is Very Low Level

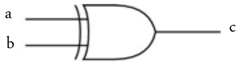
The Verilog implementation is only one-bit wide by default



```verilog
module simple_a(input a,input b, output logic c);
    assign c = a + b;
endmodule
```

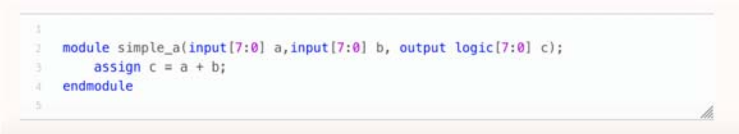
- a is one bit
- b is one bit
- c is one bit

Will most likely get synthesized to or the equivalent:



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

If we wanted an 8 bit adder:



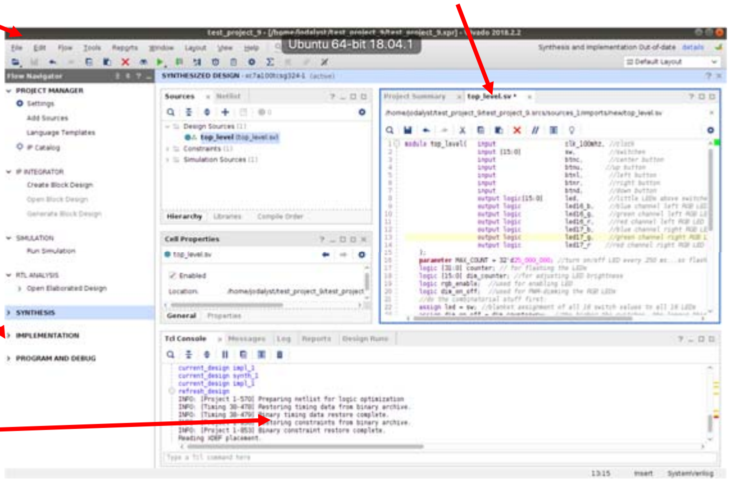
```verilog
module simple_a(input[7:0] a,input[7:0] b, output logic[7:0] c);
    assign c = a + b;
endmodule
```

- We'll be using Xilinx FPGAs in this class, and in order to do that, we'll need to use their software development environment **Vivado**
- Large piece of software that takes care of everything for us
- Lab 1 is designed to get you working with it quickly, but it is going to take a few labs to get to really know it.

Vivado QuickStart

- Video shown at beginning
- We have a fully-working project for you to build here: https://web.mit.edu/6.111/volume2/www/f2019/handouts/labs/vivado_quickstart/quickstart.html
- We recommend building it before starting Lab 1 since it has some setup stuff in it.

Verilog Gets Turned into a Circuit



Vivado

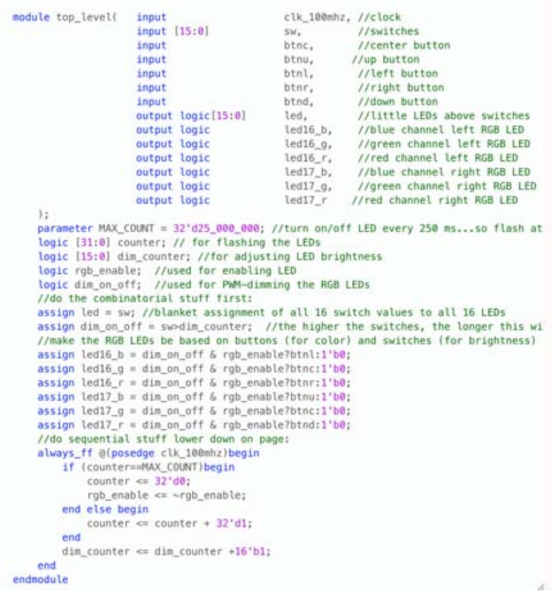
SystemVerilog file

Build Processes

Console

Verilog:

- Start with this:

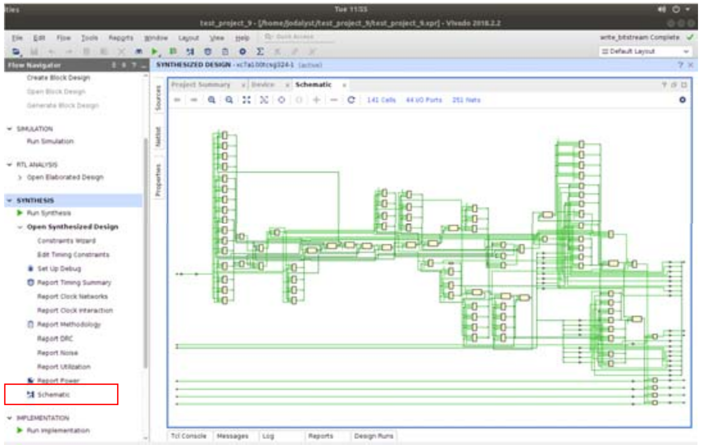


```
module top_level(   input                   clk_100mhz, //clock
                    input [15:0]            sw,         //switches
                    input                   btnc,       //center button
                    input                   btnu,      //up button
                    input                   btnl,       //left button
                    input                   btnr,       //right button
                    input                   btnd,       //down button
                    output logic[15:0]      led,        //little LEDs above switches
                    output logic            led16_b,    //blue channel left RGB LED
                    output logic            led16_g,    //green channel left RGB LED
                    output logic            led16_r,    //red channel left RGB LED
                    output logic            led17_b,    //blue channel right RGB LED
                    output logic            led17_g,    //green channel right RGB LED
                    output logic            led17_r    //red channel right RGB LED
    );
    parameter MAX_COUNT = 32'd25_000_000; //turn on/off LED every 250 ms...so flash at
    logic [31:0] counter; // for flashing the LEDs
    logic [15:0] dim_counter; //for adjusting LED brightness
    logic rgb_enable;  //used for enabling LED
    logic dim_on_off;  //used for PWM-dimming the RGB LEDs
    //do the combinatorial stuff first:
    assign led = sw; //blanket assignment of all 16 switch values to all 16 LEDs
    assign dim_on_off = sw>dim_counter;  //the higher the switches, the longer this wi
    //make the RGB LEDs be based on buttons (for color) and switches (for brightness)
    assign led16_b = dim_on_off & rgb_enable?btnl:1'b0;
    assign led16_g = dim_on_off & rgb_enable?btnc:1'b0;
    assign led16_r = dim_on_off & rgb_enable?btnr:1'b0;
    assign led17_b = dim_on_off & rgb_enable?btnu:1'b0;
    assign led17_g = dim_on_off & rgb_enable?btnc:1'b0;
    assign led17_r = dim_on_off & rgb_enable?btnd:1'b0;
    //do sequential stuff lower down on page:
    always_ff @(posedge clk_100mhz)begin
        if (counter==MAX_COUNT)begin
            counter <= 32'd0;
            rgb_enable <= ~rgb_enable;
        end else begin
            counter <= counter + 32'd1;
        end
        dim_counter <= dim_counter +16'b1;
    end
endmodule
```

This Circuit then Gets Designed

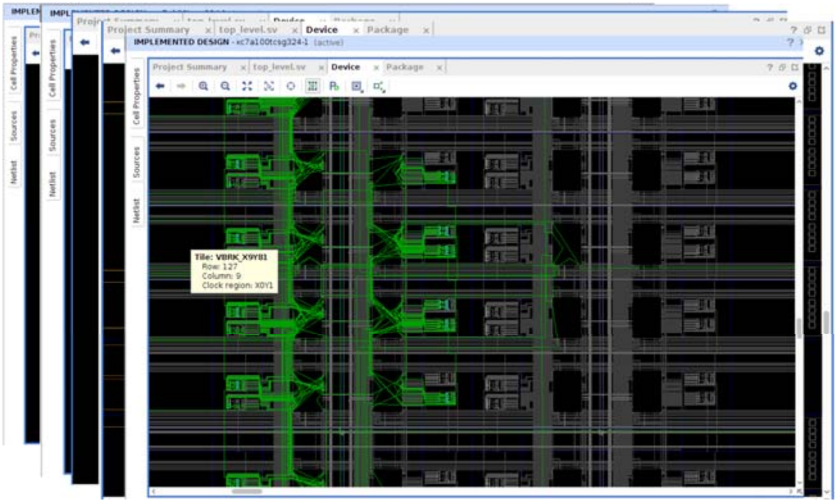
Under Synthesis go to: Schematic

Verilog is interpreted into a circuit



Then it gets placed (Implemented)

With Implementation Open go to: Windows>Device



Then it gets loaded onto board

6.111
Simple Demo for Vivado Setup
Fall 2019

For Lab 1

- The circuits you'll be building will be purely combinational, meaning they will be ***stateless***
  - Current outputs based ONLY on current inputs
  - No clocks…your designs won't need to include the concept of time
  - This shouldn't be too bad and should let us focus more on syntax and Vivado rather than the intricacies of other stuff.

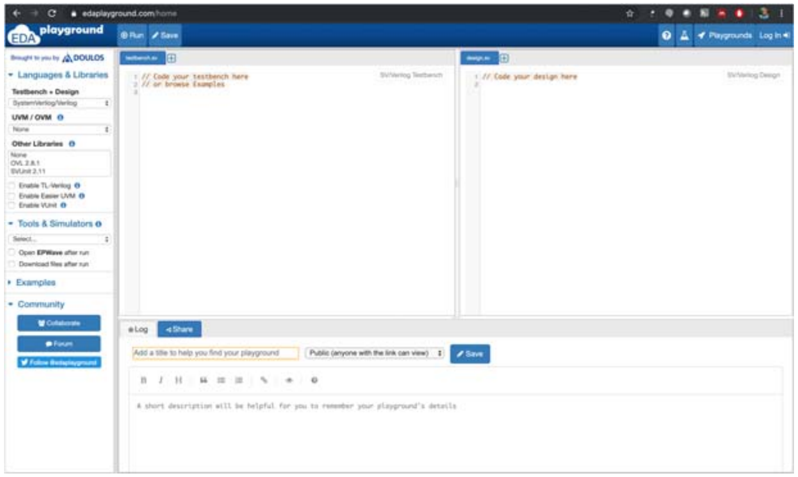
Practicing Verilog

- Install Vivado (takes up lots of space and requires Windows or Linux)

- Or….

Online Verilog/System Verilog

edaplayground.com/home



```
// Code your testbench here
// or browse Examples
```

```
// Code your design here
```

Testbench + Design: SystemVerilog/Verilog

UVM / OVM: None

Other Libraries: None, OVL 2.8.1, SVUnit 2.11

□ Enable TL-Verilog
□ Enable Easier UVM
□ Enable VUnit

Tools & Simulators: Select...

□ Open EPWave after run
□ Download files after run

Add a title to help you find your playground

Public (anyone with the link can view)

A short description will be helpful for you to remember your playground's details

Icarus Verilog

- Works well, but is only Verilog (not SystemVerilog) so a subset of commands we use such as always_ff always_comb will not be supported
- http://iverilog.icarus.com/

Recommendation

- Start Lab 1 as soon as possible (due next week)
- Lab 1 and Pset 1 will mutually support one another (Pset 1 is designed to help you do Lab 1)
- Lab 2 is longer, and will come out tomorrow/this weekend early in case you want to get started on it.