# Intro to Digital Design

## SystemVerilog Basics

**Instructor:** Chris Thachuk

**Teaching Assistants:**

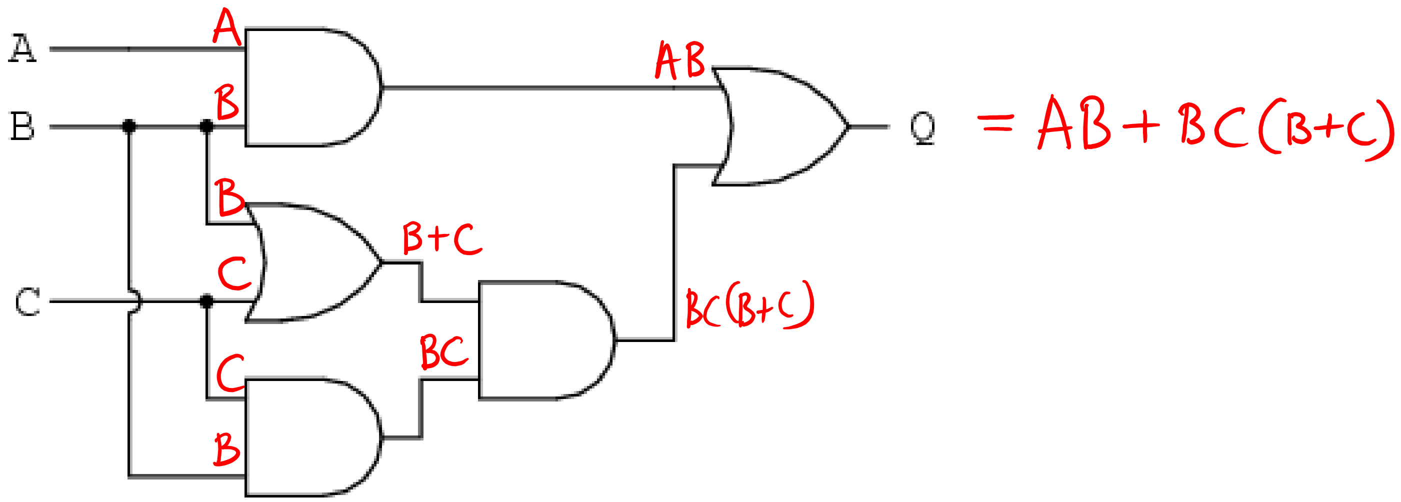
| | |
|---|---|
| Eujean Lee | Nandini Talukdar |
| Stephanie Osorio-Tristan | Wen Li |

# Relevant Course Information

- Lab demo slots will be assigned on Canvas (*tomorrow*)
- Lab 1 & 2 – Basic Logic and Verilog (*due 10/16 @ 2:30pm*)
  - Digit(s) recognizer using switches and LED
  - For full credit, find minimal logic
  - Check the lab report requirements closely
- If you haven't done so yet, pick up a white lab kit + Okiocam ASAP from CSE 003 when TAs are present (labs + office hours)
- New sections Fridays @ 3 pm on Zoom
  - Materials and recording available afterward

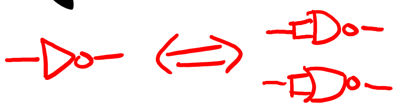
# Practice Question:

- Write out the Boolean Algebra expression for Q for the following circuit. No simplification necessary.

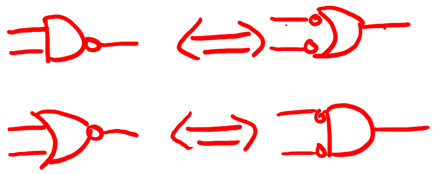


$= AB + BC(B+C)$

# Practice Question:



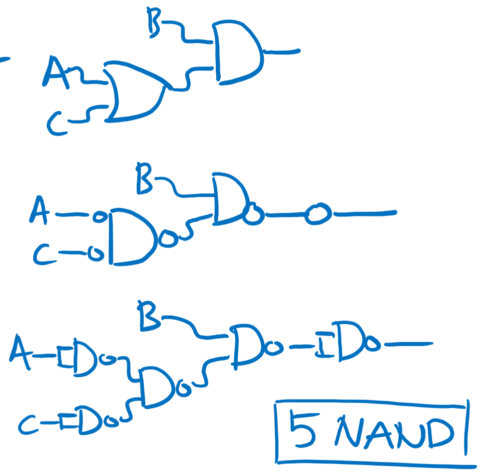
DeMorgan's:



- Implement the Boolean expression B(A + C) with the fewest number of a single universal gate. What does your solution look like?

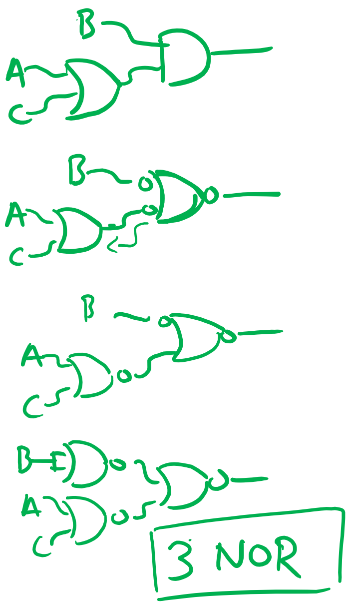
(NOR/NAND)

NAND:



5 NAND

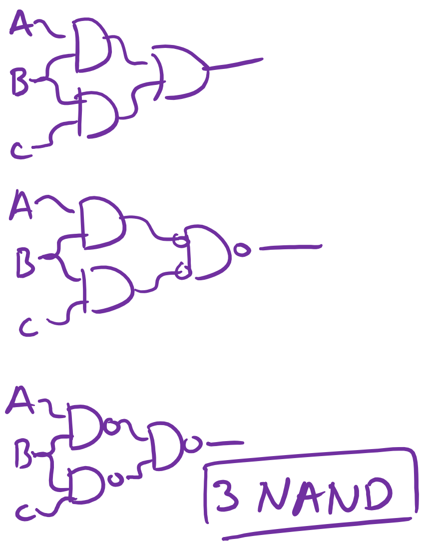
NOR:



3 NOR

NAND: (alternative)

AB + BC



3 NAND

# Lecture Outline

- **Thinking About Hardware**
- Verilog Basics
- Waveform Diagrams
- Debugging in Verilog

# Verilog

- Programming language for *describing hardware*
  - Simulate behavior before (wasting time) implementing
  - Find bugs early
  - Enable tools to automatically create implementation

- *Syntax* is similar to C/C++/Java, but behavior is very different
  - VHDL (the other major HDL) is more similar to ADA

- Modern version is **SystemVerilog**
  - Superset of previous; cleaner and more efficient

# Verilog: Hardware Descriptive Language

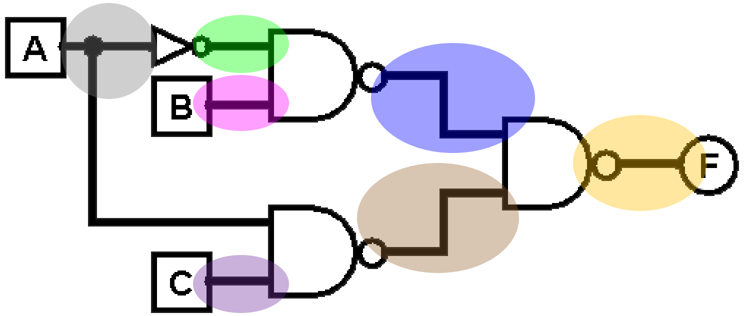
- Although it looks like code:

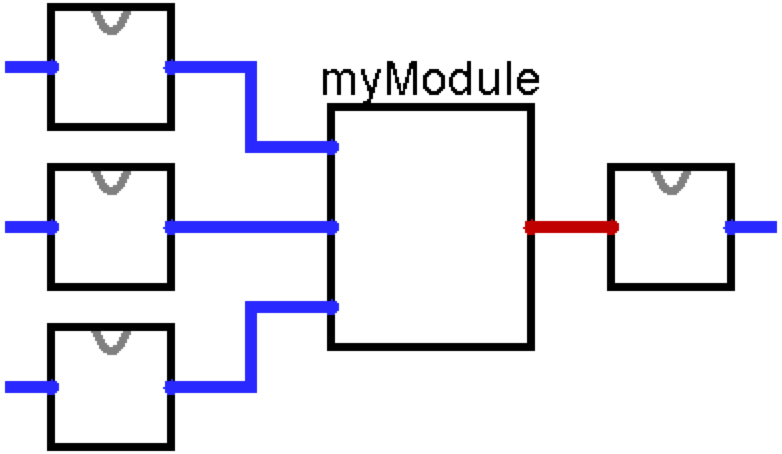
```
module myModule (F, A, B, C);
   output logic F;
   input  logic A, B, C;
   logic AN, AB, AC;

   nand gate1(AB,AN, B);
   nand gate2(AC, A, C);
   nand gate3( F,AB,AC);
   not   not1(AN, A);
endmodule
```

↔ equivalent

- Keep the hardware in mind:





# Verilog Primitives

- **Nets** (`wire`): transmit value of connected source
  - Problematic if connected to two different voltage sources
  - Can connect to many places (always possible to "split" wire)
- **Variables** (`reg`): variable voltage sources
  - Can "drive" by assigning arbitrary values at any given time
  - SystemVerilog: variable $\boxed{\texttt{logic}}$ can be used as a net, too

☆ we will primarily use this

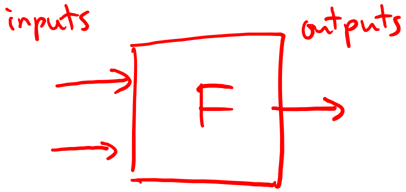
- Logic Values
  - **0** = zero, low, FALSE
  - **1** = one, high, TRUE
  - **X** = unknown, uninitialized, contention (conflict)
  - **Z** = floating (disconnected), high impedance

# Verilog Primitives

- **Gates**:

| Gate | Verilog Syntax |
|---|---|
| NOT a | `~a` |
| a AND b | `a & b` |
| a OR b | `a \| b` |
| a NAND b | `~(a & b)` |
| a NOR b | `~(a \| b)` |
| a XOR b | `a ^ b` |
| a XNOR b | `~(a ^ b)` |

- **Modules**: "classes" in Verilog that define *blocks*
  - Input: Signals passed from outside to inside of block
  - Output: Signals passed from inside to outside of block



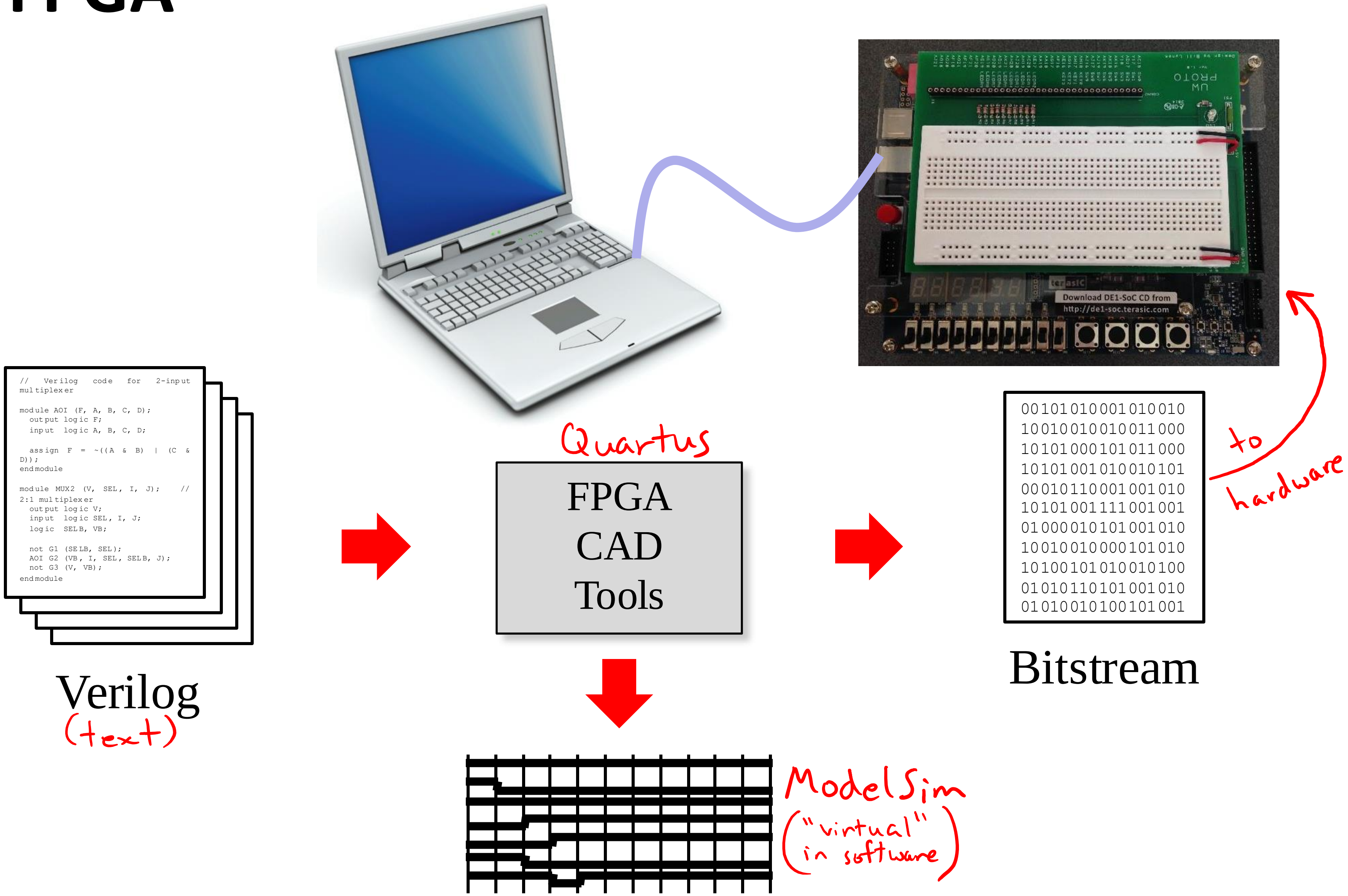
# Verilog Execution

- Physical wires transmit voltages (electrons) near-instantaneously
  - Wires by themselves have no notion of sequential execution
- Gates and modules are constantly performing computations
  - Can be hard to keep track of!
- In pure hardware, there is no notion of initialization
  - A wire that is not driven by a voltage will naturally pick up a voltage from the environment

# Lecture Outline

- Thinking About Hardware
- **Verilog Basics**
- Waveform Diagrams
- Debugging in Verilog

# Using an FPGA



```
//   Verilog   code   for   2-input
multiplexer

module AOI (F, A, B, C, D);
  output logic F;
  input  logic A, B, C, D;

  assign  F  =  ~((A  &  B)  |  (C  &
D));
endmodule

module MUX2 (V, SEL, I, J);    //
2:1 multiplexer
  output logic V;
  input  logic SEL, I, J;
  logic  SELB, VB;

  not G1 (SELB, SEL);
  AOI G2 (VB, I, SEL, SELB, J);
  not G3 (V, VB);
endmodule
```

Verilog
(text)

Quartus

FPGA
CAD
Tools

```
001010100010100010
100100100100111000
101010001010011000
101010010100010101
000101100010010010
101010011110010001
010000101010010010
100100100001010010
101001010100010100
010101101010010010
010100101001010001
```

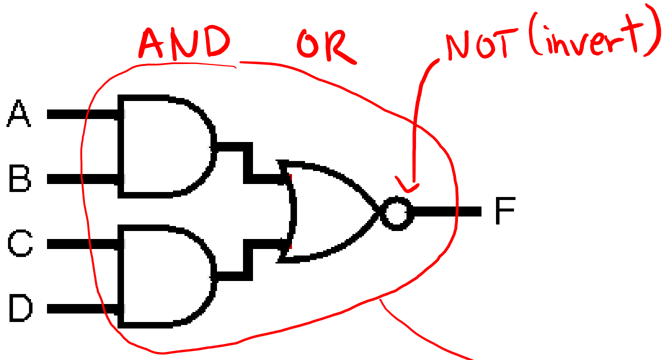
Bitstream

to hardware

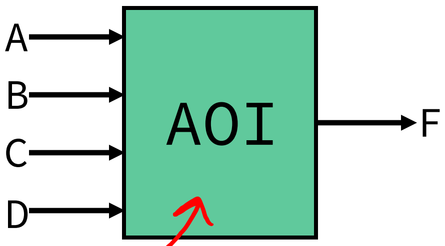
Simulation

ModelSim
("virtual" in software)

# Structural Verilog



Block Diagram: (details hidden)



```
// SystemVerilog code for AND-OR-INVERT circuit

module AOI (F, A, B, C, D);
   output logic F;
   input  logic A, B, C, D;

   assign F = ~((A & B) | (C & D));
endmodule

// end of SystemVerilog code
```

module name

ports (connections to block)

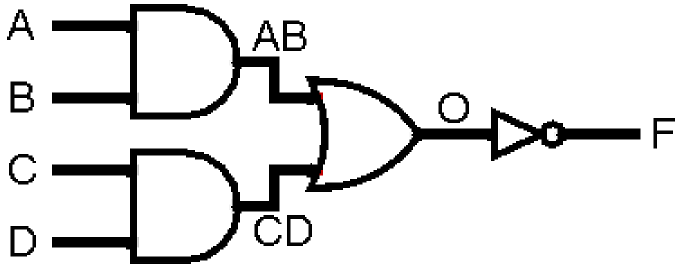
port type

OR

NOT

AND

# Verilog Wires



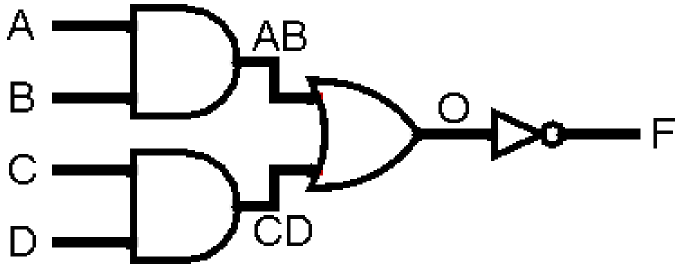
```
// SystemVerilog code for AND-OR-INVERT circuit

module AOI (F, A, B, C, D);
   output logic F;
   input  logic A, B, C, D;
   logic  AB, CD, O;  // now necessary

   assign AB = A & B;
   assign CD = C & D;
   assign O = AB | CD;
   assign F = ~O;
endmodule
```

identical in hardware,
just more explicit in Verilog code

# Verilog Gate Level



```
// SystemVerilog code for AND-OR-INVERT circuit

module AOI (F, A, B, C, D);
   output logic F;
   input  logic A, B, C, D;
   logic  AB, CD, O;   // now necessary
   and a1(AB, A, B);
   and a2(CD, C, D);
   or  o1(O, AB, CD);
   not n1(F, O);
endmodule
```

module type

instance name

port connections

was:

```
assign AB = A & B;
assign CD = C & D;
assign O = AB | CD;
assign F = ~O;
```

# Verilog Hierarchy

```
// SystemVerilog code for AND-OR-INVERT circuit
module AOI (F, A, B, C, D);
   output logic F;
   input  logic A, B, C, D;

   assign F = ~((A & B)|(C & D));
endmodule
```

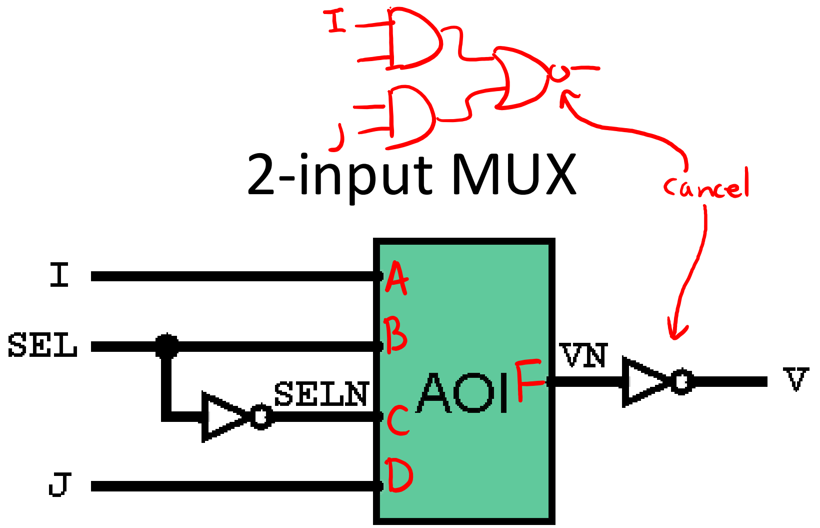
user-defined

```
// 2:1 multiplexer built on top of AOI module
module MUX2 (V, SEL, I, J);
   output logic V;
   input  logic SEL, I, J;
   logic  SELN, VN;

   not G1 (SELN, SEL);
   AOI G2 (.F(VN), .A(I), .B(SEL), .C(SELN),
.D(J));
   not G3 (V, VN);
endmodule
```

primitive (built-in)

explicit connection to port



if SEL==0, V=J
if SEL==1, V=I

# Technology

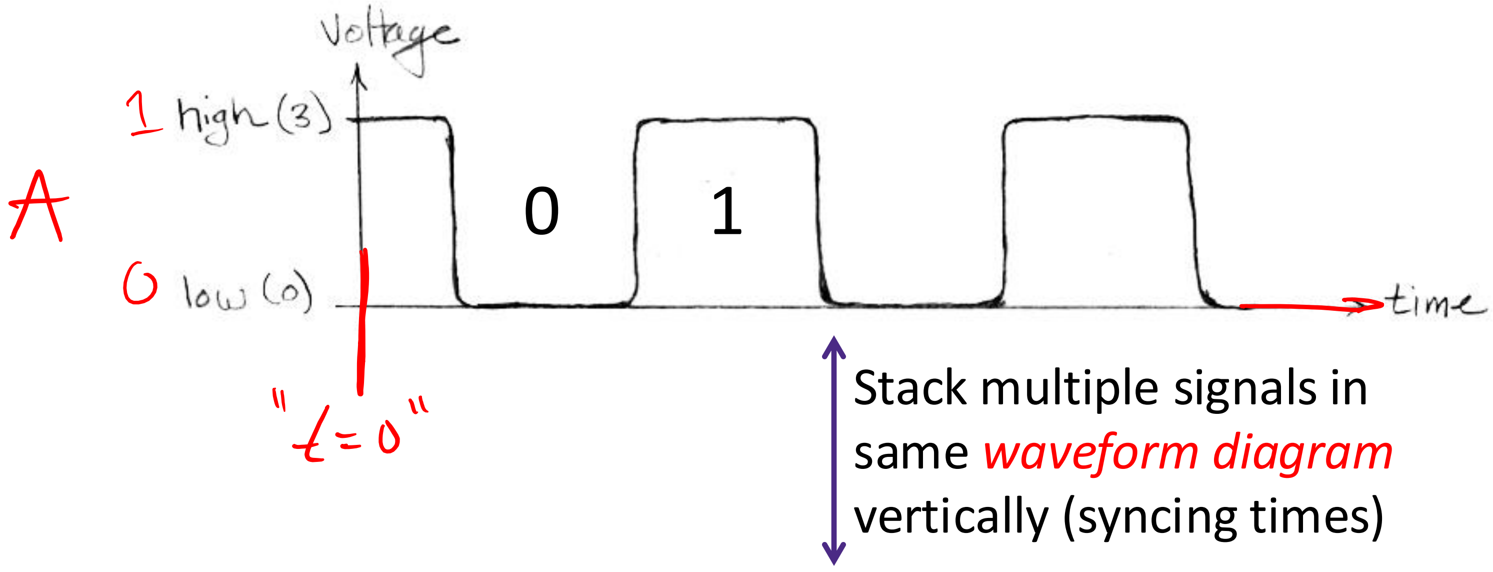
# Break

# Lecture Outline

- Thinking About Hardware
- Verilog Basics
- **Waveform Diagrams**
- Debugging in Verilog

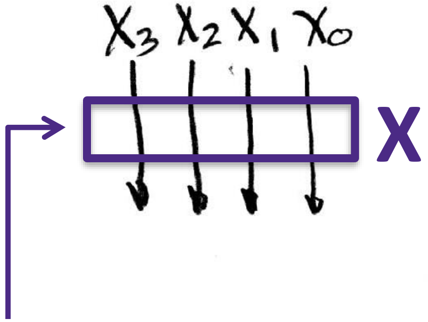
# Signals and Waveforms

- Signals transmitted over wires continuously
  - Transmission is effectively instantaneous (a wire can only contain one value at any given time)
  - In digital system, a wire holds either a 0 (low voltage) or 1 (high voltage)

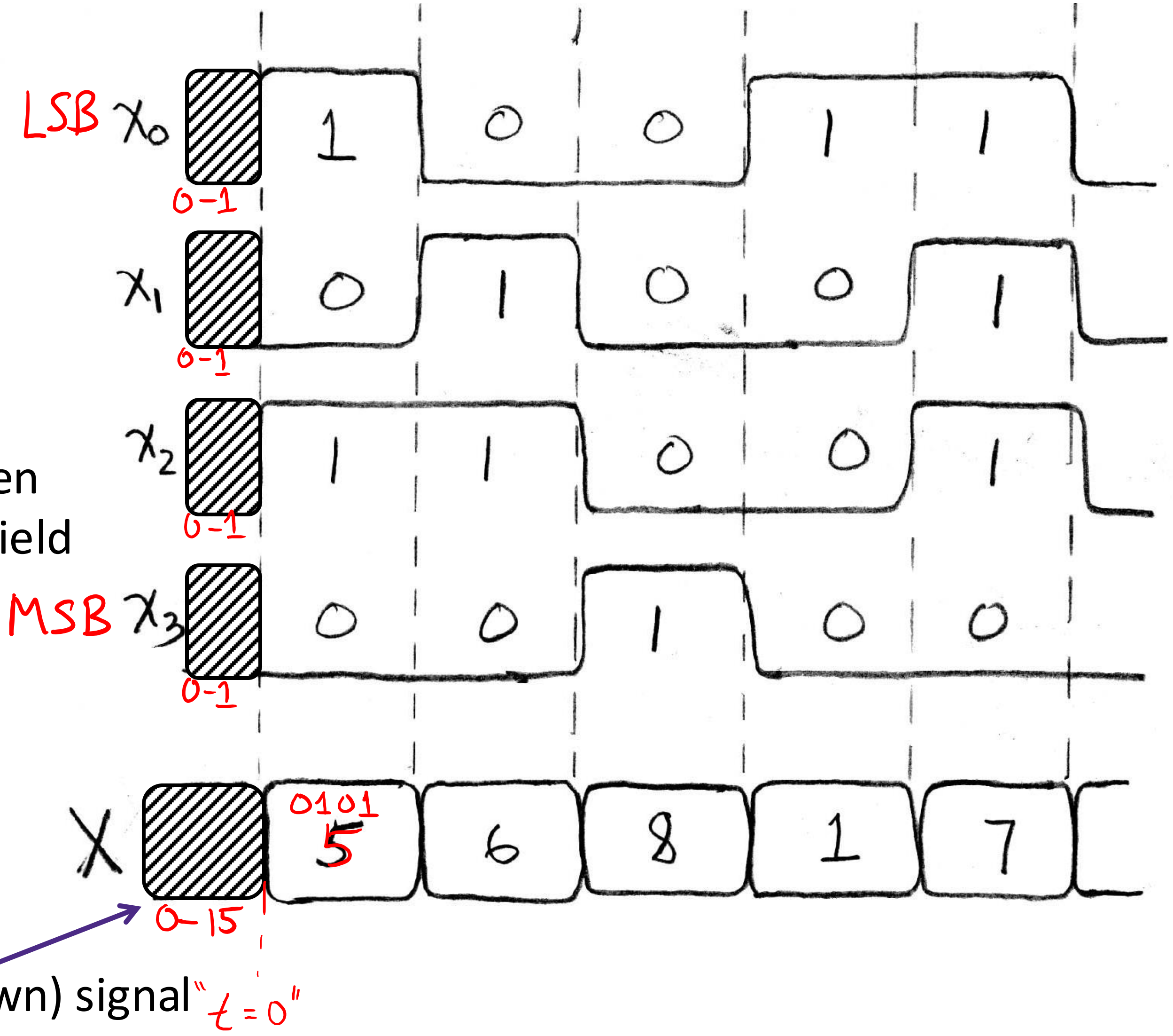


Stack multiple signals in same *waveform diagram* vertically (syncing times)

# Signal Grouping



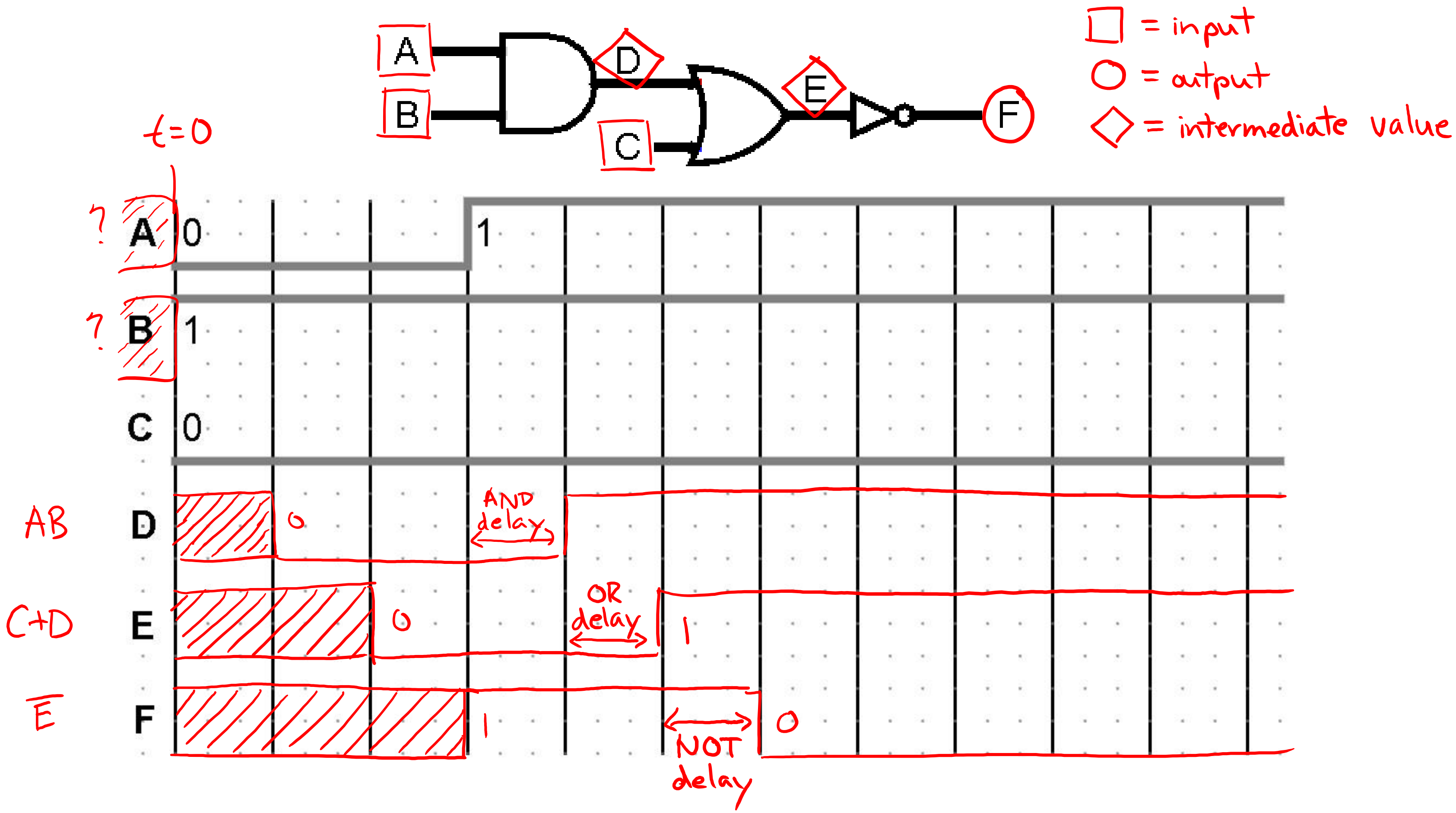
A group of wires when interpreted as a bit field is called a *bus*



“undefined” (unknown) signal

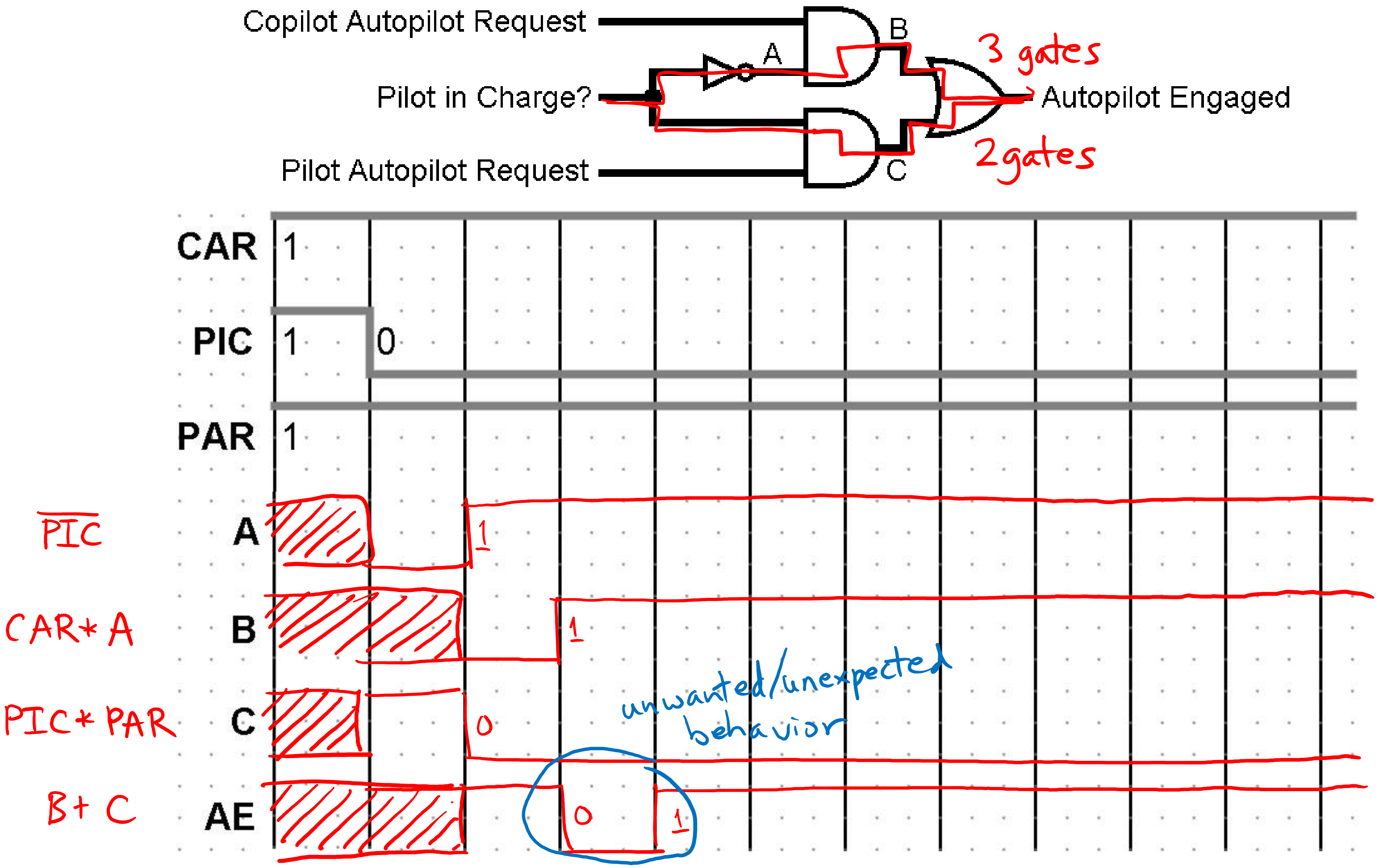
# Circuit Timing Behavior

- Simple Model: Gates "react" after fixed delay
  - Example: Assume delay of all gates is 1 ns (= 3 ticks)

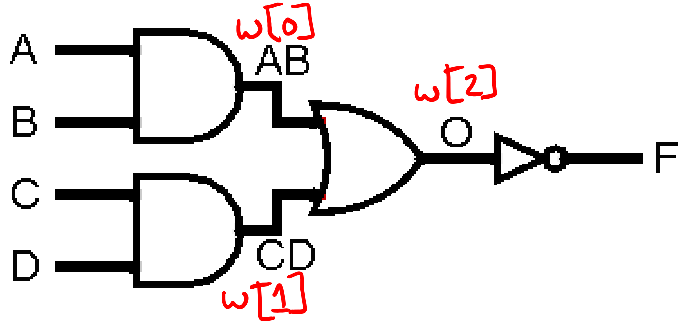


# Circuit Timing: Hazards/Glitches

- Circuits can temporarily go to incorrect states!
  - Assume 1 ns delay (3 ticks) for all gates



# Verilog Buses



```
// SystemVerilog code for AND-OR-INVERT circuit

module AOI (F, A, B, C, D);
   output logic F;
   input  logic A, B, C, D;
   logic  [2:0] w;  // necessary

   assign w[0] = A & B;
   assign w[1] = C & D;
   assign w[2] = w[0] | w[1];
   assign F = ~w[2];
endmodule
```

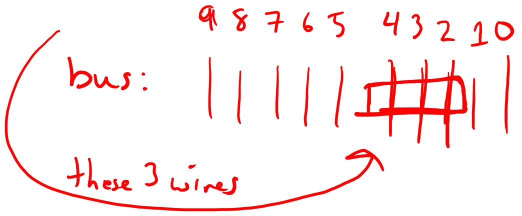
declare bus width

individual wire of bus

Just for illustration – this is bad coding style!

# Verilog Signal Manipulation

- Bus definition: `[n-1:0]` is an n-bit bus
  - Good practice to follow bit numbering notation
  - Access individual bit/wire using “array” syntax (*e.g.*, `bus[1]`)
  - Can access sub-bus using similar notation (*e.g.*, `bus[4:2]`)



- Multi-bit constants: `n'b#…#`
  - `n` is width, `b` is radix specifier (`b` for binary), `#`s are digits of number
  - decimal binary hex
    *e.g.*, `4'd12`, `4'b1100`, `4'hC` ← these are all equivalent
- Concatenation: `{sig, …, sig}`
  - Ordering matters; result will have combined widths of all signals
- Replication operator: `{n{m}}`
  - repeats value `m`, `n` times

# Practice Question

width 5

```
logic [4:0] apple;
```

width 4

```
logic [3:0] pear;
```

width 10

```
logic [9:0] orange;
assign apple = 5'd20;
```

index: 4 3 2 1 0

→ 5'b10100

```
assign pear = {1'b0, apple[2:1], apple[4]};
```

{0 , 10 , 1}

- What's the value of `pear`?

  4'b0101 = $\boxed{5}$ (across 4 wires)

- If we want orange to be the *sign-extended* version of apple, what is the appropriate Verilog statement?

MSB

0000 sign extension → 0000 0000

1000 from 4 to 8 bits 1111 1000

```
assign orange = { {5{apple [4]}} , apple};
```

# Lecture Outline

- Thinking About Hardware
- Verilog Basics
- Waveform Diagrams
- **Debugging in Verilog**

# Test Benches

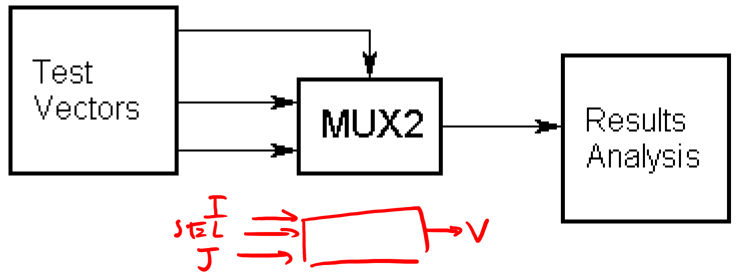
- *Needed for simulation only!*
  - Software constraint to mimic hardware

- ModelSim runs entirely on your computer
  - Tries to simulate your FPGA environment without actually using hardware – no physical signals available
  - Must create fake inputs for FPGA's physical connections
    - *e.g.*, `LEDR`, `HEX`, `KEY`, `SW`, `CLOCK_50`
  - Unnecessary when code is loaded onto FPGA

- Need to define both input signal combinations as well as their *timing*

# Verilog Test Benches

(simulation only!)



defines simulation environment & timing

self-contained environment so no ports

variables for inputs

read output

```
module MUX2_tb ();
  logic SEL, I, J;   // simulated inputs
  logic V;           // net for reading output

  // instance of module we want to test ("device under test")
  MUX2 dut (.V(V), .SEL(SEL), .I(I), .J(J));

  initial   // build stimulus (test vectors)
  begin     // start of "block" of code
    {SEL, I, J} = 3'b100; #10; // t=0:  S=1, I=0, J=0 -> V=0
    I = 1;                #10; // t=10: S=1, I=1, J=0 -> V=1
    SEL = 0;              #10; // t=20: S=0, I=1, J=0 -> V=0
    J = 1;                #10; // t=30: S=0, I=1, J=1 -> V=1
  end       // end of "block" of code

endmodule  // MUX2_tb
```

"{"

"}"

time delay

# Better Verilog Test Bench

```
module MUX2_tb ();
  logic SEL, I, J;  // simulated inputs
  logic V;          // net for reading output

  // instance of module we want to test ("device under
test")
  MUX2 dut (.V(V), .SEL(SEL), .I(I), .J(J));

  int i;
  initial  // build stimulus (test vectors)
  begin    // start of "block" of code
    for(i = 0; i < 8; i = i ~~+ 1) begin~~
      {SEL, I, J} = i;  #10;
    end
  end      // end of "block" of code

endmodule  // MUX2_tb
```

< runs through SEL, I, J with delay between each one

| SEL, | I, | J |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |
| 1 | 1 | 1 |

# Debugging Circuits

- Complex circuits require careful debugging
  - Test as you go; don’t wait until the end (system test)
  - *Every* module should have a test bench (unit test)

1) Test all behaviors
   - All combinations of inputs for small circuits, subcircuits
2) Identify any incorrect behaviors
3) Examine inputs & outputs to find earliest place where value is wrong
   - Typically trace backwards from bad outputs, forwards from inputs
   - Look at values at intermediate points in circuit

# Hardware Debugging

- Simulation (ModelSim) is used to debug logic *design* and should be done thoroughly before touching FPGA
  - Unfortunately, ModelSim is not a perfect simulator

- If interfacing with other circuitry (*e.g.*, breadboard), will also need to debug circuitry layout there
  - Similar process, but with power sources (inputs) and voltmeters (probe the wires)
  - Often just a poor electrical connection somewhere

- Sometimes things simply fail
  - All electrical components fail eventually, whether you caused it to or not

# Summary

- SystemVerilog is a hardware description language (HDL) used to program your FPGA
  - Programmatic syntax used to describe the connections between gates and registers

- Waveform diagrams used to track intermediate signals as information propagates through CL

- Hardware debugging is a critical skill
  - Similar to debugging software, but using different tools