

Intro to Digital Design

L2: More CL, Verilog Basics

Instructor: Naomi Alterman

Teaching Assistants:

Derek de Leuw

Isabel Froelich

Kevin Hernandez

Aarjav Jain

Packard Stephenson

Administrivia

- ❖ Lab demo time slots have been assigned on Canvas
 - Check the comment on the “Demo Time Slot” assignment
- ❖ Lab 1 this week – Basic Logic
 - **Report + code due on Wednesday @ 2:30p**
 - Check the lab report requirements closely
 - Show up to your demo time with your design synthesized and your simulations ready to view 🙌

Today

- ❖ **Logic Minimization**
- ❖ Verilog Basics
- ❖ Debugging, Simulations and Waveform Diagrams

Translating between Truth Tables and Boolean Equations

- ❖ Terms of equation come from rows of table
 - For 1, write variable name
 - For 0, write complement of variable
- ❖ Sum of Products (SoP)
 - From CSE311, “DNF” (disjunctive normal form)
 - Take truth table rows that output 1:
 - AND the inputs together, OR the rows together
- ❖ Product of Sums (PoS)
 - From CSE311, “CNF” (conjunctive normal form)
 - Take truth table rows that output 0:
 - OR the complemented inputs together, AND the rows together

$$\text{SoP: } C = \bar{A}B + \bar{B}A$$

$$\text{PoS: } C = (A + B) \cdot (\bar{A} + \bar{B})$$

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

Logic minimization

- ❖ Reduce complexity at gate level
 - Allows us to build smaller and faster hardware
 - Care about both # of gates, # of literals (gate inputs), # of gate levels, and types of logic gates

Logic minimization

- ❖ Reduce complexity at gate level
 - Allows us to build smaller and faster hardware
 - Care about both # of gates, # of literals (gate inputs), # of gate levels, and types of logic gates
- ❖ Faster hardware?
 - Fewer inputs implies faster gates in some technologies
 - Fan-ins (# of gate inputs) are limited in some technologies
 - Fewer levels of gates implies reduced signal propagation **delays**
 - # of gates (or gate packages) influences manufacturing costs
 - Simpler Boolean expressions → smaller transistor networks → smaller circuit delays → faster hardware

Are Logic Gates Created Equal?

❖ No!

2-Input Gate Type	# of CMOS transistors
NOT	2
AND	6
OR	6
NAND	4
NOR	4
XOR	8
XNOR	8

- ❖ Can recreate all other gates using only NAND or only NOR gates
 - Called “universal” gates
 - *e.g.*, $A \text{ NAND } A = \bar{A}$, $B \text{ NOR } B = \bar{B}$
 - DeMorgan’s Law helps us here!

DeMorgan's Law

X	Y	\bar{X}	\bar{Y}	NOR		NAND	
				$\overline{X + Y}$	$\bar{X} \cdot \bar{Y}$	$\overline{X \cdot Y}$	$\bar{X} + \bar{Y}$
0	0	1	1	1		1	
0	1	1	0	0		1	
1	0	0	1	0		1	
1	1	0	0	0		0	

❖ $\overline{X + Y} = \bar{X} \cdot \bar{Y}$

❖ $\overline{X \cdot Y} = \bar{X} + \bar{Y}$

❖ In Boolean Algebra, converts between AND-OR and OR-AND expressions

- $Z = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C}$

- $\bar{Z} = (A + B + \bar{C}) \cdot (A + \bar{B} + \bar{C}) \cdot (\bar{A} + B + \bar{C})$

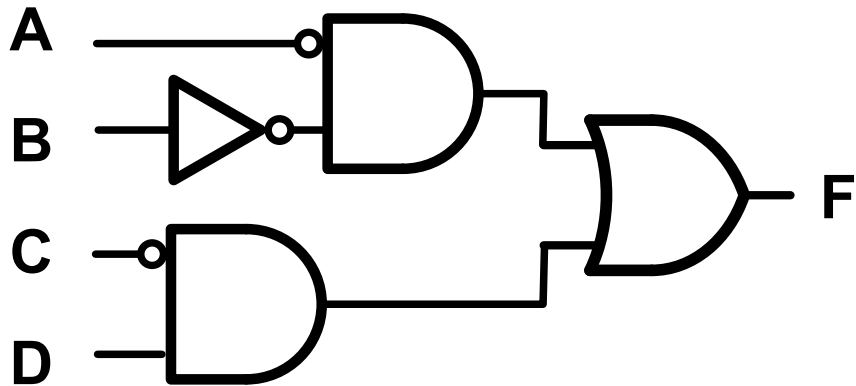
❖ At gate level, can convert from AND/OR to NAND/NOR gates

- “Flip” all input/output bubbles and “switch” gate



DeMorgan's Law Practice Problem

- ❖ Implement this circuit with only NOR gates:



Today

- ❖ Logic Minimization
- ❖ **Verilog Basics**
- ❖ Debugging, Simulations and Waveform Diagrams

Verilog

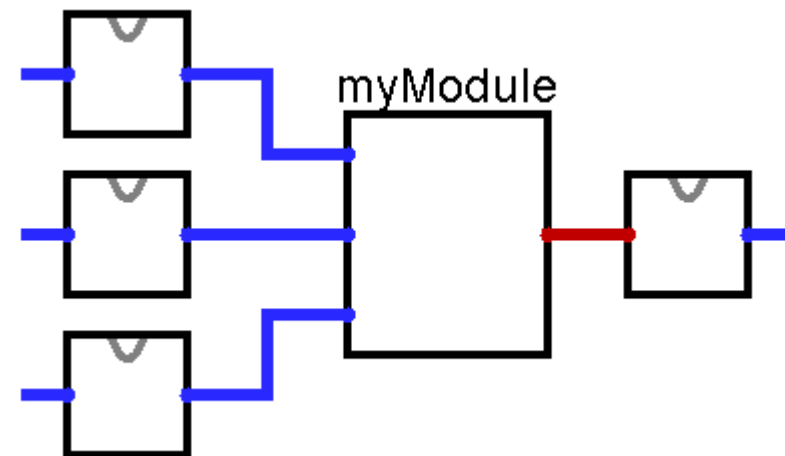
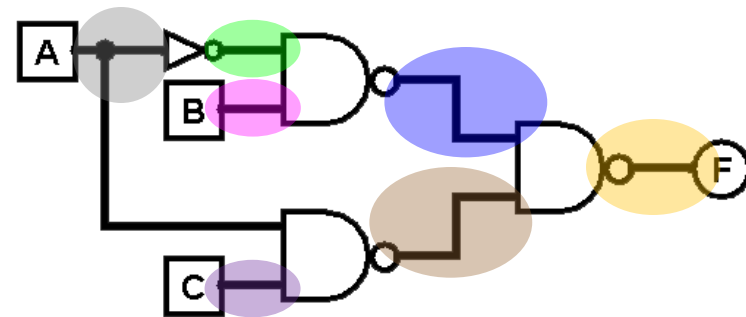
- ❖ A hardware description language (**HDL**)
 - Define circuit schematics using text editors
 - Simulate behavior before (wasting time) implementing
 - Find bugs early
- ❖ ***Syntax*** is like C/C++/Java, but **meaning** is *very* different
 - Borrows heavily from early concurrency-focused languages like *Modula*
 - VHDL (the other major HDL) borrows from Ada (per...the DoD??)
- ❖ 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
```

❖ It directly represents a physical circuit:



Verilog Primitives

- ❖ **Nets:** carry bits from one gate to another
 - SystemVerilog type: “**wire**”
 - Think of it *like* an immutable reference to mutable data in C++
- ❖ **Variables:** like a net, but the circuit setting the voltage can change
 - SystemVerilog type: “**reg**” or “**logic**”
 - NB: nothing to do with “registers” from Assembly (☹) !
 - “Variable” refers to the *source* of the data, not the data itself (“**driving** the voltage”)
 - Think of it *like* a pointer whose location is determined by a switch case in C++
- ❖ ...In this class, we’ll just use “**logic**” for everything

Verilog Primitives

❖ 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	$\sim a$
a AND b	$a \& b$
a OR b	$a b$
a NAND b	$\sim(a \& b)$
a NOR b	$\sim(a b)$
a XOR b	$a \wedge b$
a XNOR b	$\sim(a \wedge b)$

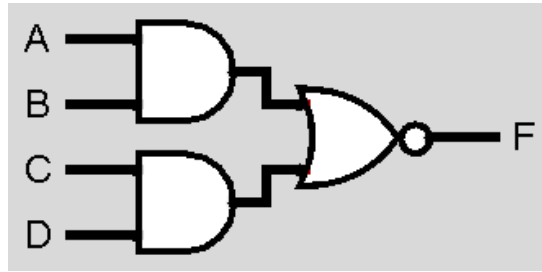
❖ Modules: “classes” in Verilog that define *blocks* of logic

- **Input:** Signals passed from outside to inside of block
- **Output:** Signals passed from inside to outside of block

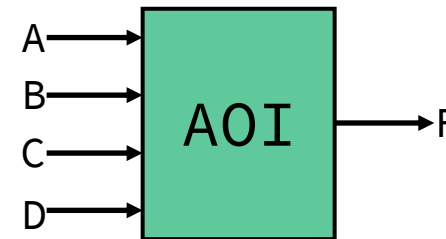
Verilog Execution

- ❖ 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
- ❖ In pure hardware, there is **no** notion of **reassignment**
 - Verilog variables represent physical wires. The value carried by the wire can change, but the wire's endpoints do not

Structural Verilog



Block Diagram:

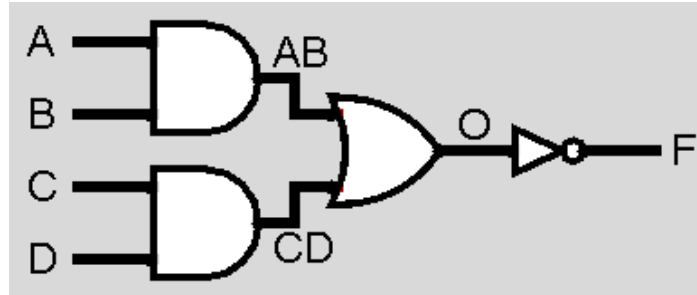


// Verilog code for AND-OR-INVERT gate

```
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 Verilog code

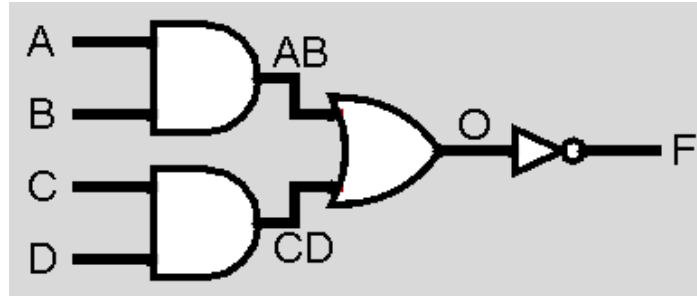
Verilog Wires



// Verilog code for AND-OR-INVERT gate

```
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
```

Verilog Gate Level



// Verilog code for AND-OR-INVERT gate

```
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
```

} was:

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

Verilog Hierarchy

// 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

```

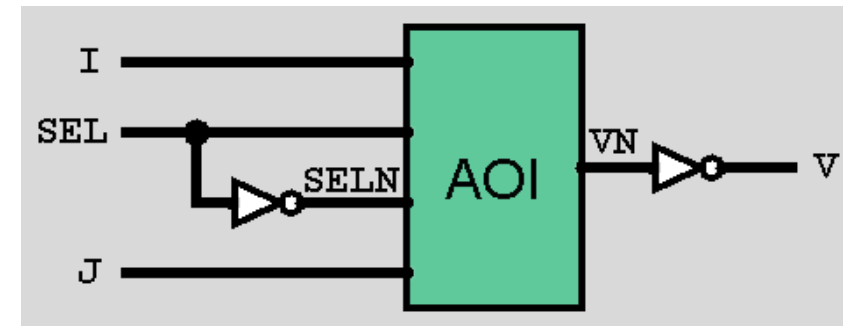
```

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

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

```

2-input MUX



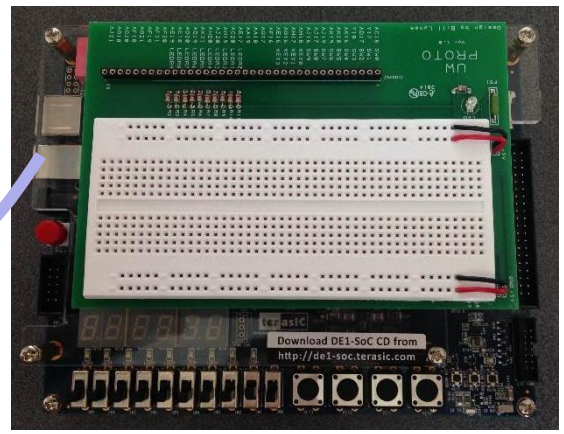
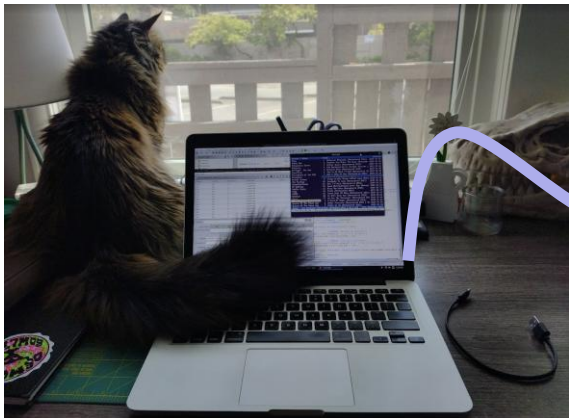
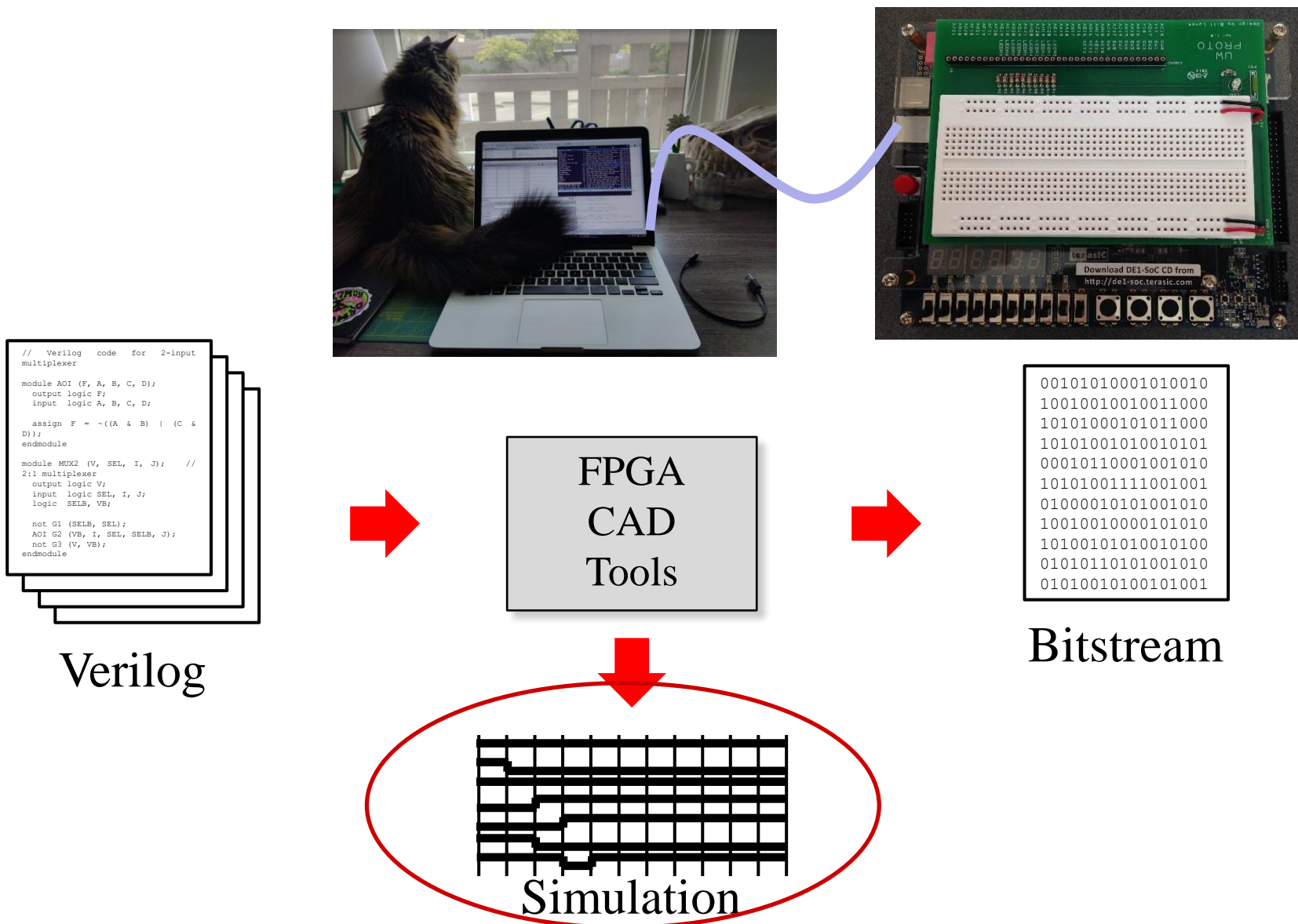
Miso Moment



Today

- ❖ Logic Minimization
- ❖ Verilog Basics
- ❖ **Debugging, Simulations and Waveform Diagrams**

Using an FPGA



Testbenches

- ❖ **ModelSim** is a digital logic simulator
 - Runs entirely on your computer and simulates the operation of your FPGA
 - Used for debugging the internals of buggy Verilog
 - No such thing as printf() in a physical circuit (at least not without microscopic tweezers)
- ❖ **Testbench** – a Verilog module that instantiates the circuit you're testing *and* provides a lil script to generate fake input signals
 - We need to mockup fake signals for *every* input to our module, *and* the timing of how they change.
 - Doesn't/can't get synthesized
 - **You need a testbench for every single module you write. Period.**

Verilog Testbenches

```
module MUX2_tb ();  
    logic SEL, I, J; // variables remember values  
    logic V;        // acts as net for reading output  
  
    initial // build stimulus (test vectors)  
    begin // start of "block" of code  
        SEL = 1; I = 0; J = 0; #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  
  
    MUX2 dut (.V, .SEL, .I, .J);  
  
endmodule // MUX2_tb
```

No ports

```
graph LR; TV[Test Vectors] --> MUX2[MUX2]; TV --> MUX2; TV --> MUX2; TV --> RA[Results Analysis];
```

"Device Under Test"

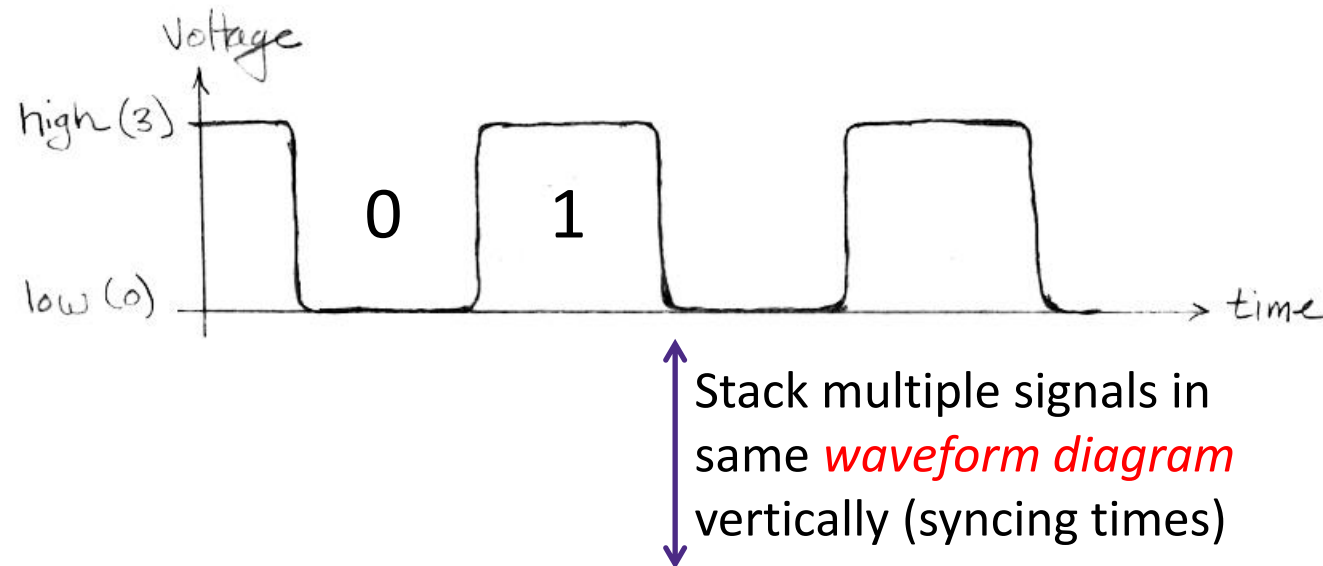
The screenshot displays a digital logic simulator interface with the following components:

- Instance Panel:** Shows a hierarchy starting with 'MUX2_bad_tb', containing 'dut' and 'std'.
- Objects Panel:** Lists signals: SEL, I, J, V, and i. A green circle indicates a measurement of 13 ps.
- Wave - Default Panel:**
 - Msgs Table:**

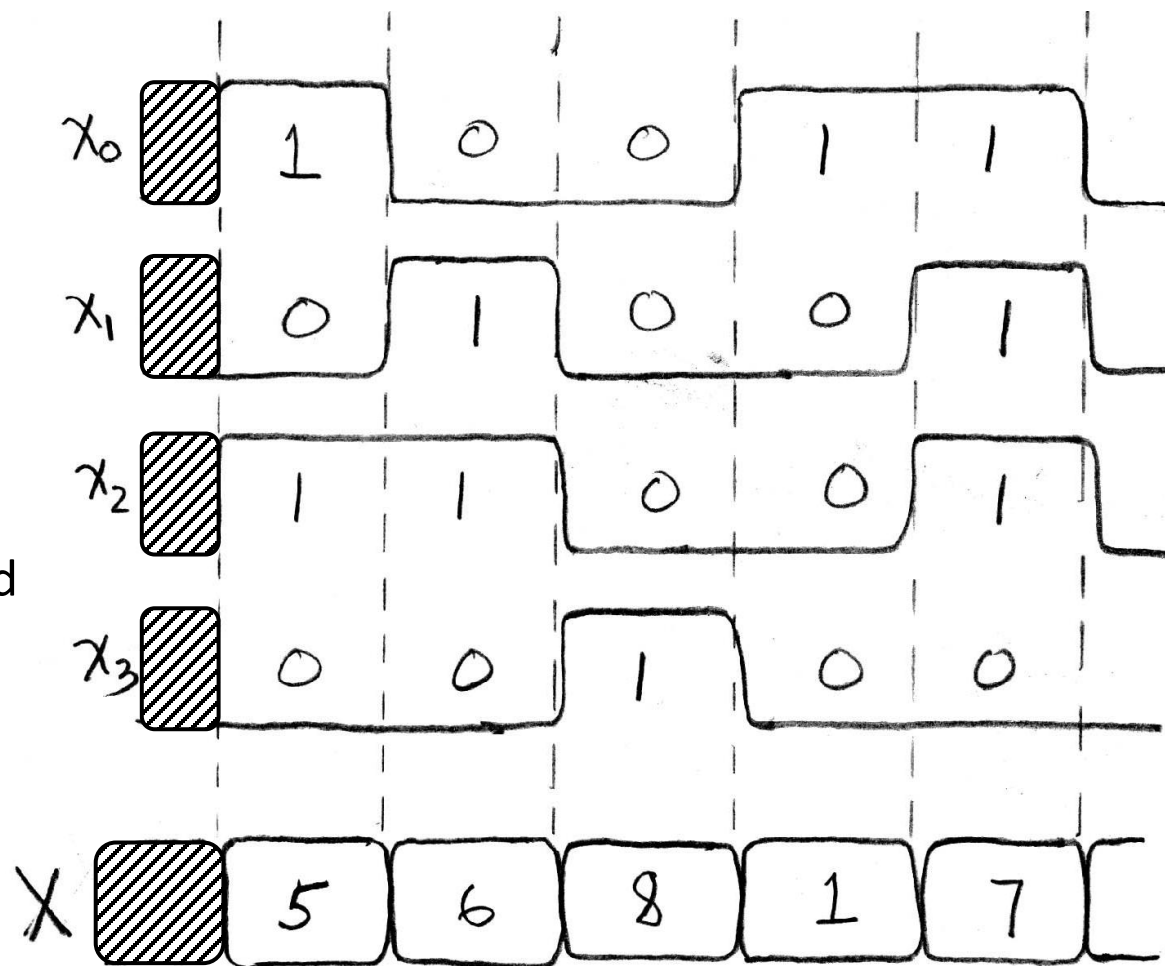
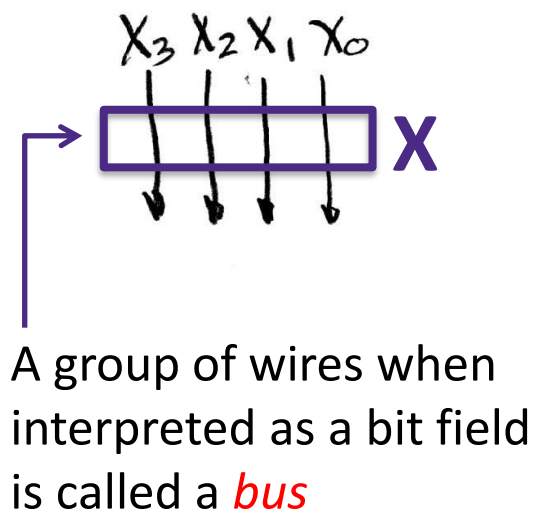
Signal	Value
SEL	0
I	0
J	1
V	0
 - Timing Diagram:** Shows waveforms for SEL, I, J, and V. A yellow vertical cursor is at 13 ps. A measurement box below the cursor shows '13 ps'.
 - Time Scale:** The horizontal axis is marked with 10 ps, 20 ps, 30 ps, 40 ps, and 50 ps.
 - Cursor Info:** Shows 'Now 80 ps' and 'ursor 1 13 ps'.
- Taskbar:** Shows 'Library', 'sim', 'Wave', and 'rundemo.do'.

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)



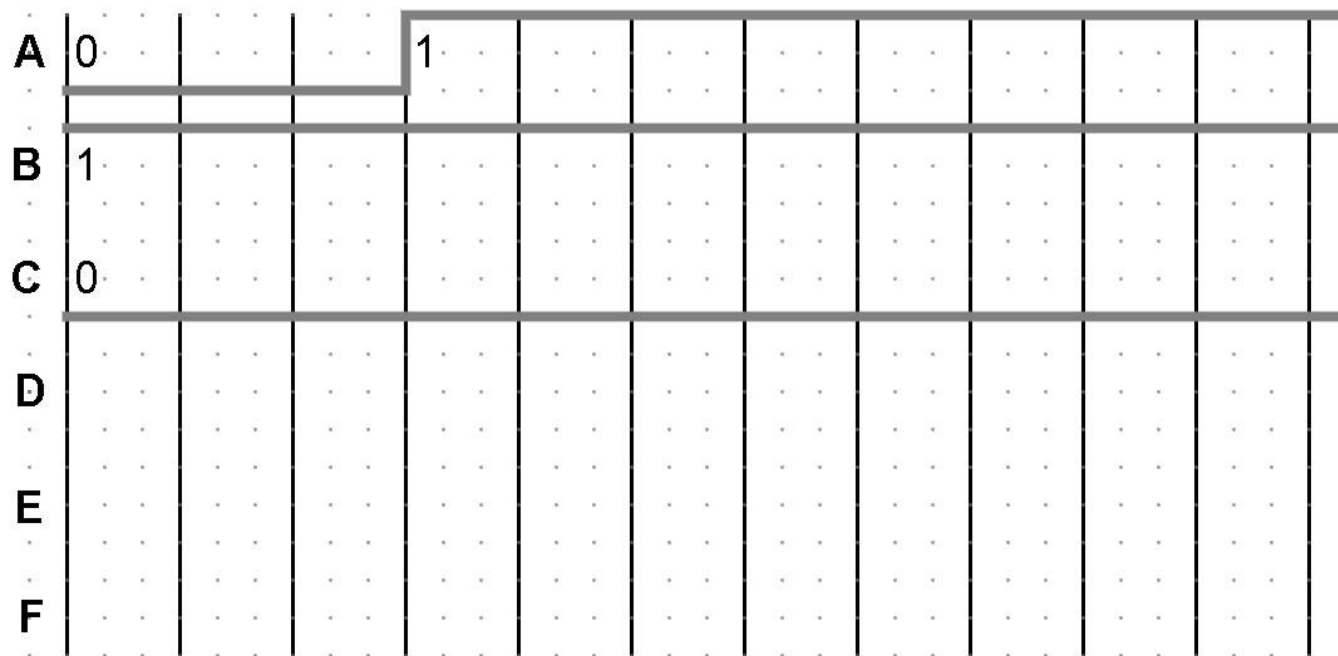
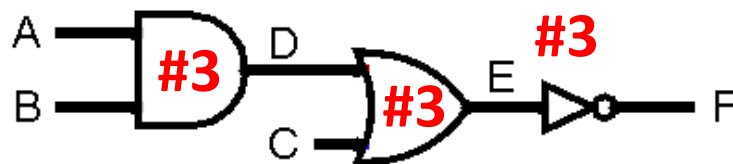
Signal Grouping



“undefined” (unknown) signal

Circuit Timing Behavior

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

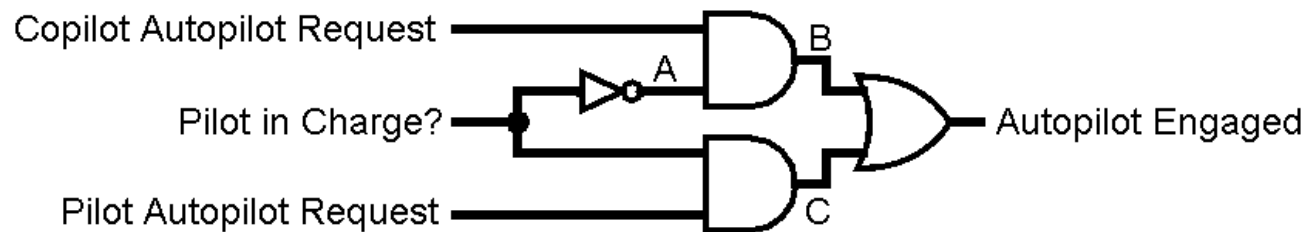


Timing Diagrams Matter



Who is in charge?

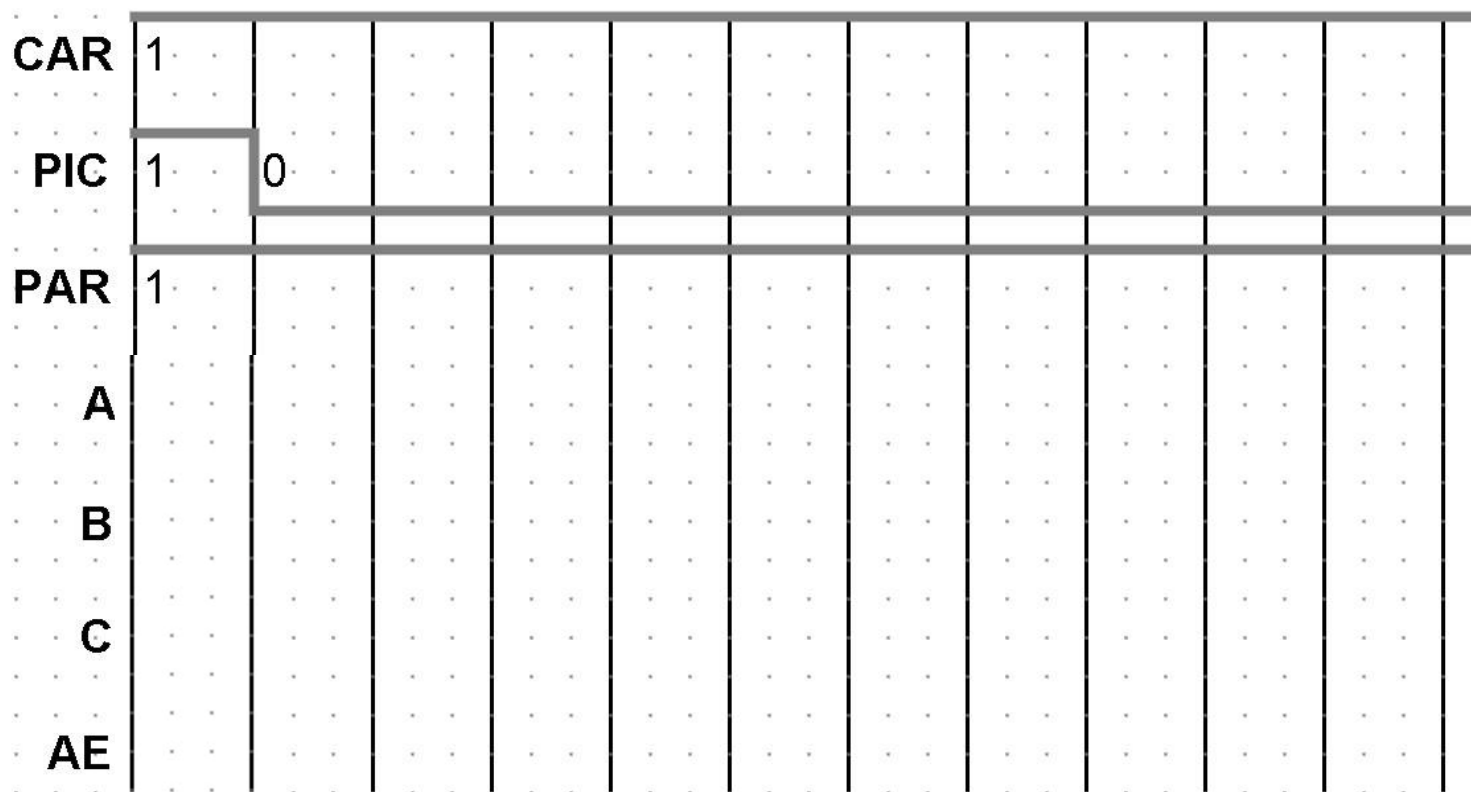
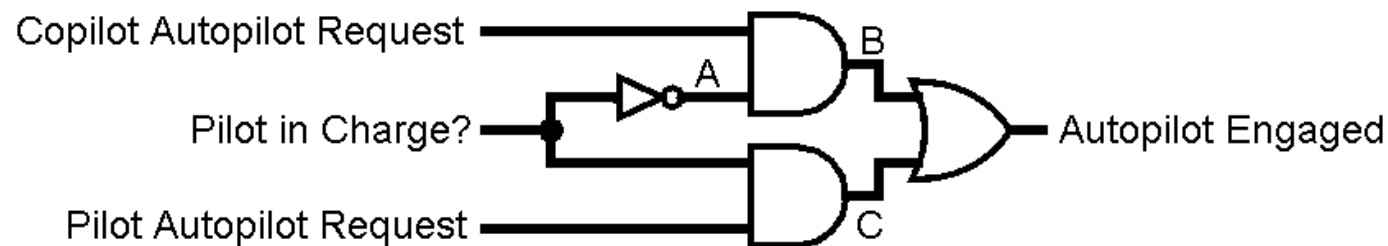
- Pilot?
- Copilot?
- Autopilot?



Circuit Timing: Hazards/Glitches

❖ Circuits can temporarily go to incorrect states!

- Assume 1 ns (3 tick) delay for all gates



Summary

- ❖ Verilog 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