Review Problem 4

- Simplify the following Boolean Equation

$$AB + AC + \overline{A}B$$

$= AB + \bar{A}B + AC$ (COMM.)

$= (A+\bar{A})B + AC$ (DIST.)

$= B + AC$ (UNITY)

Review Problem 5

- Simplify the following Boolean Equation, starting with DeMorgan's Law

$$\overline{F} = A\overline{B} + AC$$

$$F = \overline{A\overline{B} + AC}$$

$$= \overline{(A\overline{B})}\,\overline{(AC)}$$

$$= (\overline{A} + B)(\overline{A} + \overline{C})$$

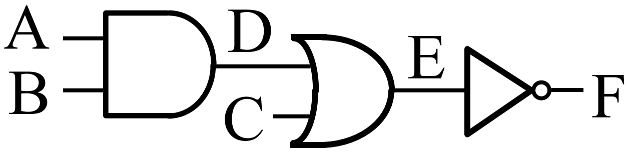
$$= \overline{A}\,\overline{A} + \overline{A}\,\overline{C} + \overline{A}B + B\overline{C}$$

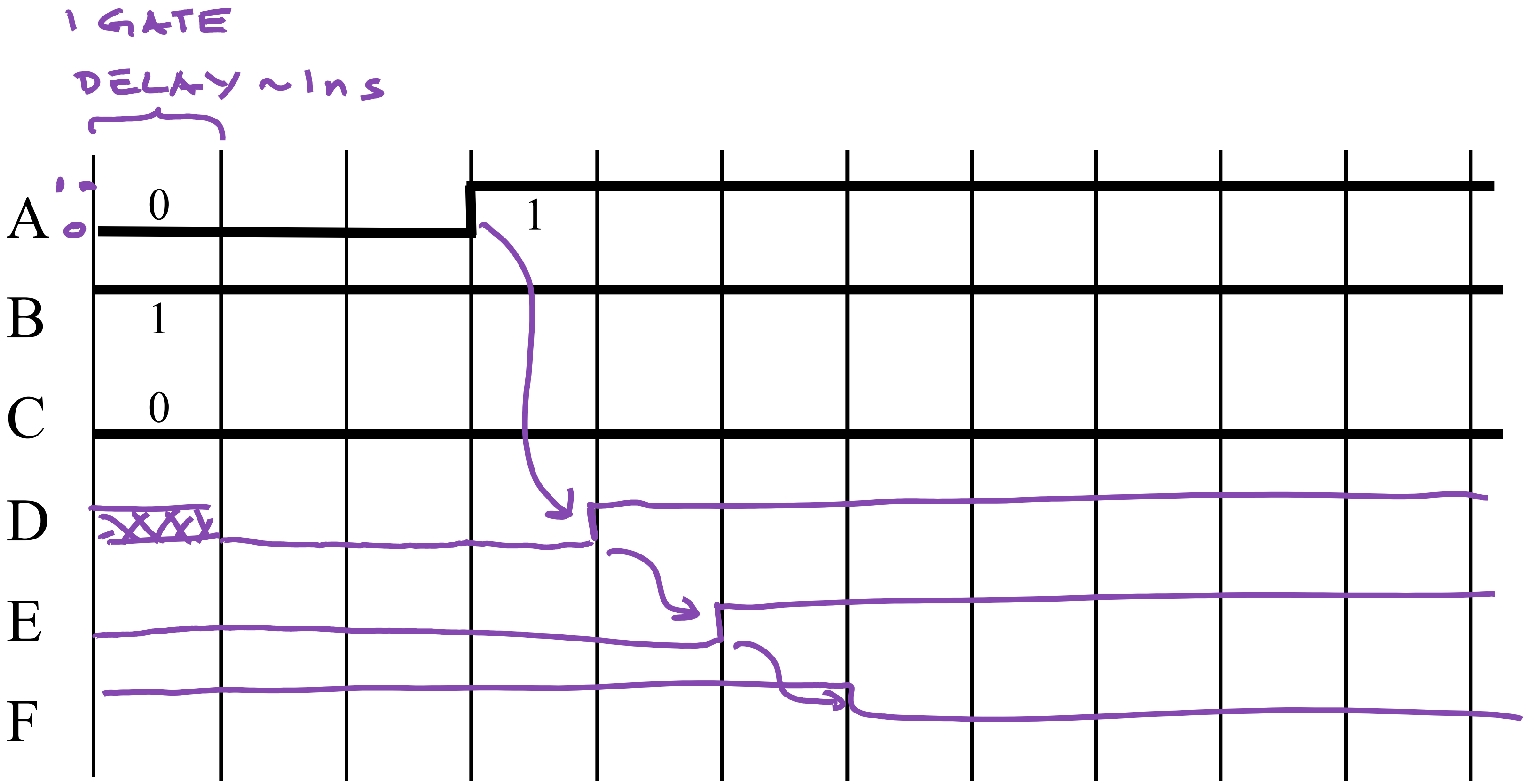
$$= \overline{A}(1 + \overline{C} + B) + B\overline{C}$$

$$= \overline{A} + B\overline{C}$$

Circuit Timing Behavior

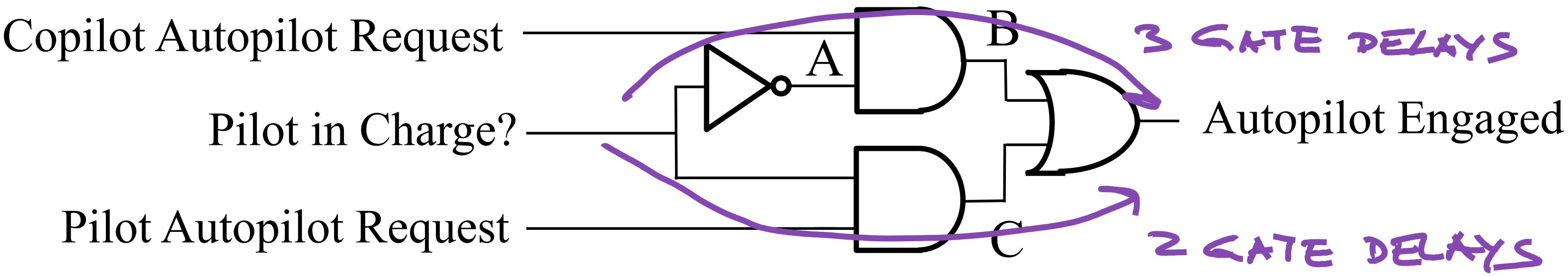
- Simple model: gates react after fixed delay

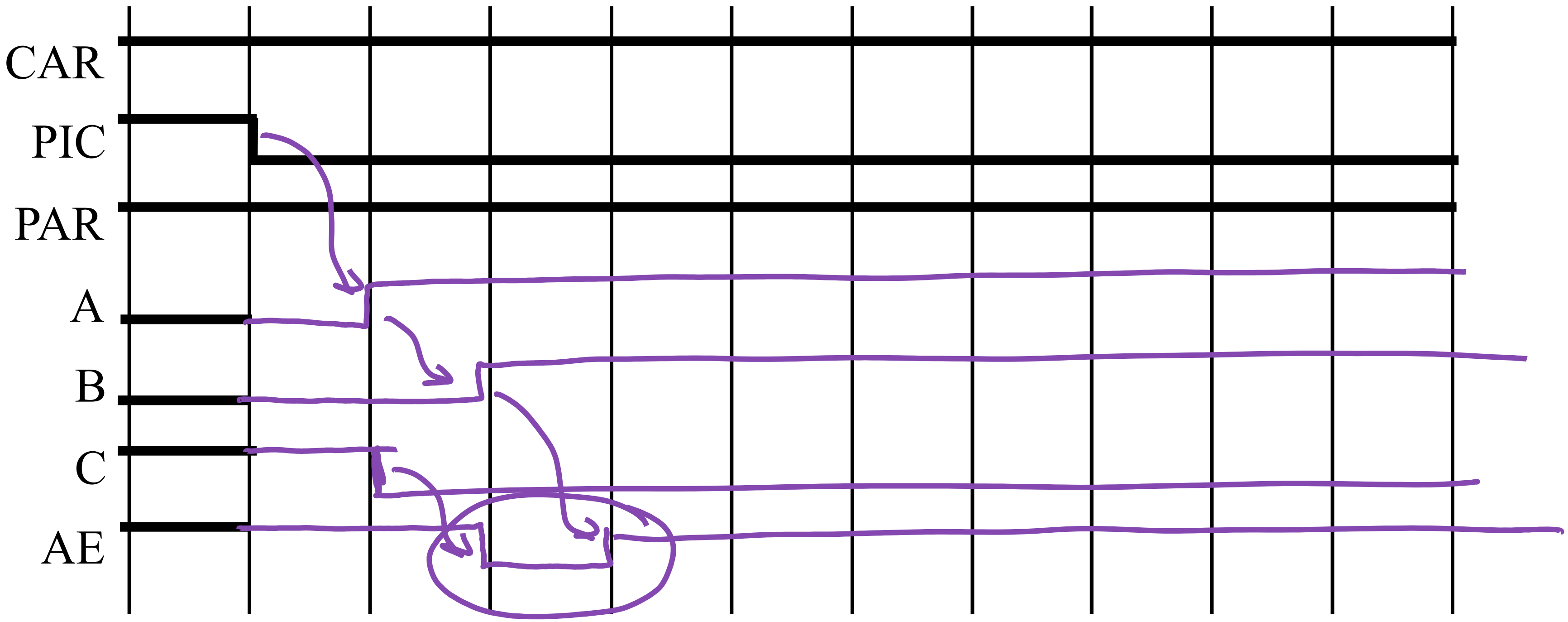




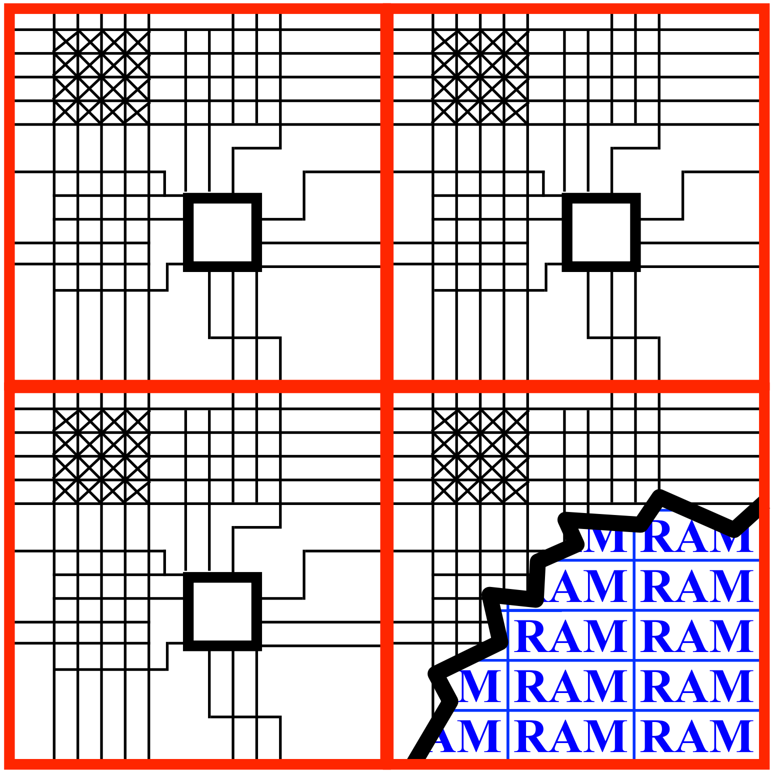
Hazards/Glitches

- Circuit can temporarily go to incorrect states





Field Programmable Gate Arrays (FPGAs)



Logic cells imbedded in a general routing structure

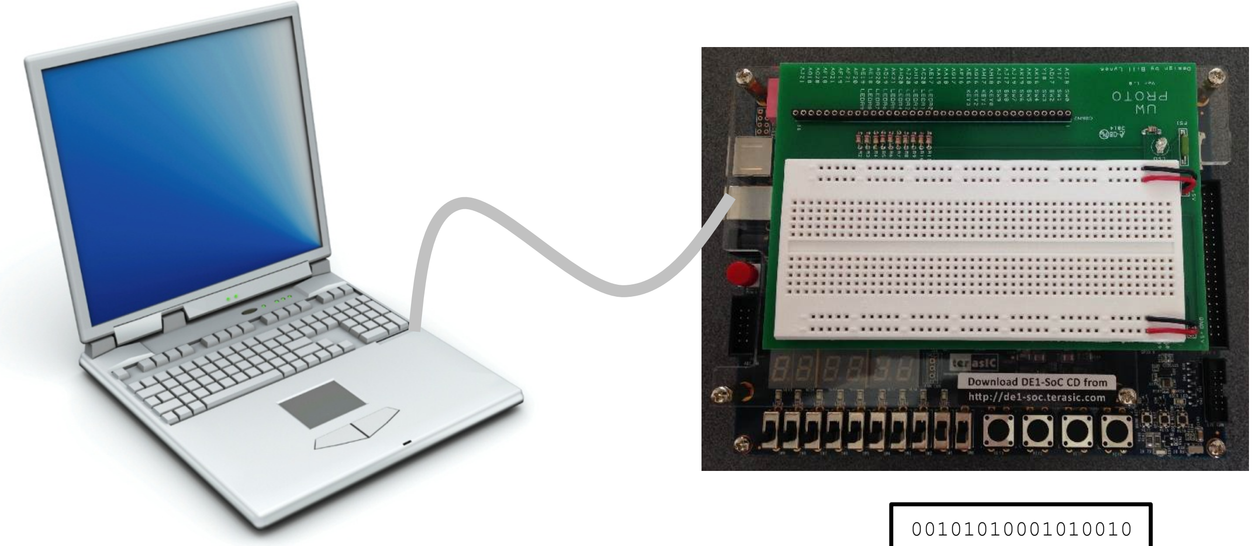
□

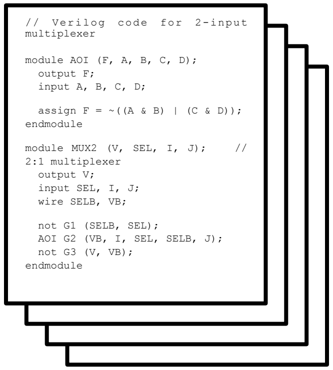
Logic cells usually contain:

- **6-input Boolean function calculator**
- **Flip-flop (1-bit memory)**

All features electronically (re)programmable

Using an FPGA

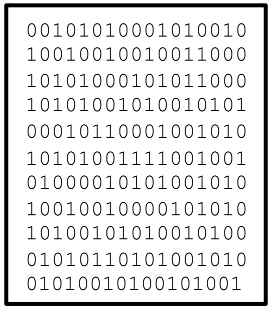




```
// Verilog code for 2-input
multiplexer

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

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

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

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

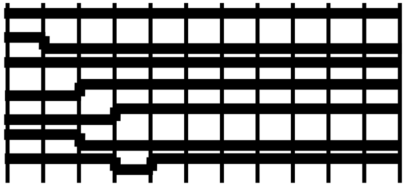
Verilog

FPGA
CAD
Tools



```
0010101000101001О
```

Bitstream



Simulation

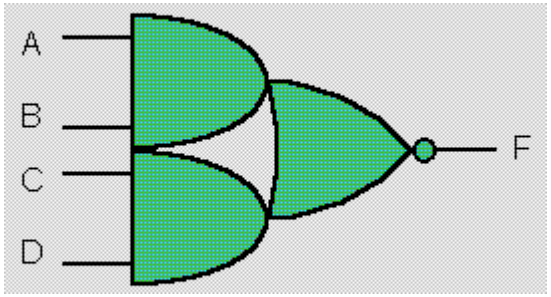
Verilog

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

- Similar to C/C++/Java
  - VHDL similar to ADA

- Modern version is “System Verilog”
  - Superset of previous; cleaner and more efficient

Structural Verilog



```
// Verilog code for AND-OR-INVERT gate
```

FUNCTION

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

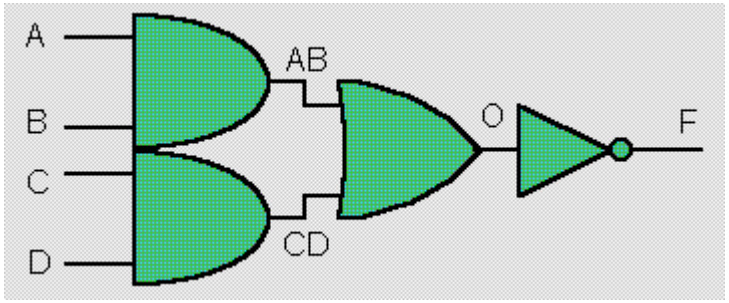
CONTINUOUS ASSIGNMENT: F CHANGES WHEN INPUTS CHANGE

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

NOT AND OR

```
// end of Verilog code
```

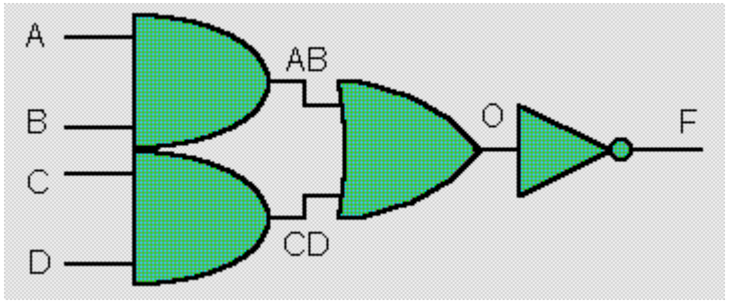
Verilog Wires/Variables



```
// Verilog code for AND-OR-INVERT gate

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

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

LOCAL VARIABLE IN MODULE; ACTUAL WIRE

Verilog Gate Level



```
// Verilog code for AND-OR-INVERT gate

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

  and a1(AB, A, B);
  and a2(CD, C, D);
  or o1(O, AB, CD);
  not n1(F, O);
endmodule
```

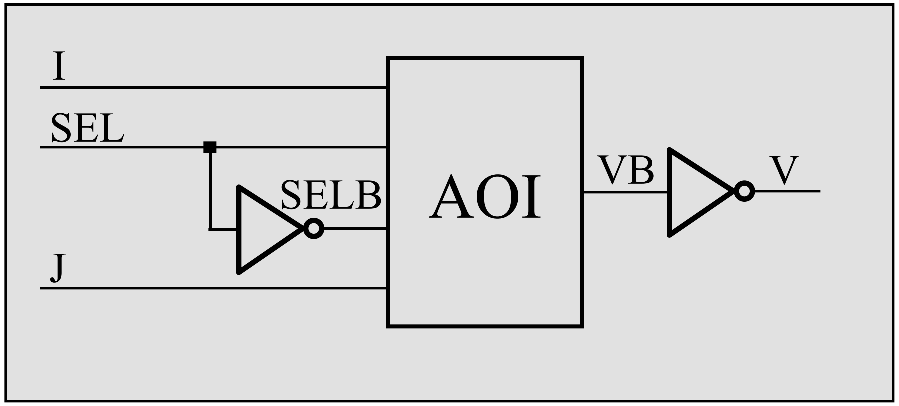
OTHER KEYWORDS:

NAND, NOR, XOR, XNOR, BUF

Verilog Hierarchy

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

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

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

2-input Mux



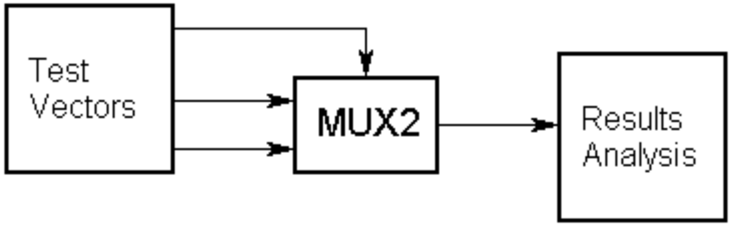
```
module MUX2 (V, SEL, I, J);   // 2:1 multiplexer
  output V;
  input SEL, I, J;
  wire SELB, VB;

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

ORDER DOESN'T MATTER IN THIS NOTATION

G2(VB, I, SEL, SELB, J)

Verilog Testbenches

reg: SIMULATION REGISTER - KEEPS TRACK OF A SIGNAL VALUE (USE "reg" IN "INITIAL" AND "ALWAYS" BLOCK, ELSE WIRE)



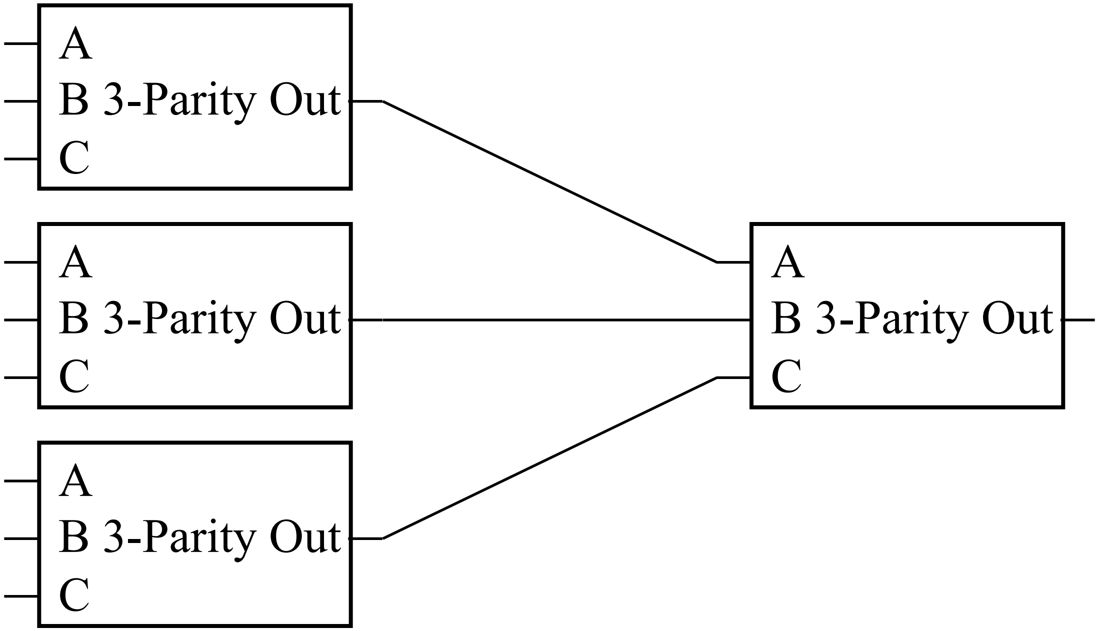
```
module MUX2TEST;   // No ports!
   reg SEL, I, J;   //  Remembers value -
reg
  wire V;

  initial   // Stimulus
  begin
    SEL = 1; I = 0; J = 0;
    #10 I = 1;
    #10 SEL = 0;
    #10 J = 1;
  end

  MUX2 M (.V, .SEL, .I, .J);

  initial   // Response
       $monitor($time, , SEL, I, J, , V);
```

,V(V)

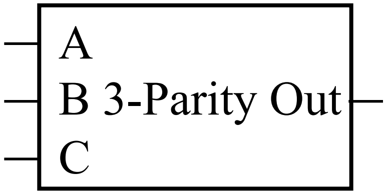
```
TIME SEL I J  V
  0  100  0
 10  110  1
 20  010  0
 30  011  1
```

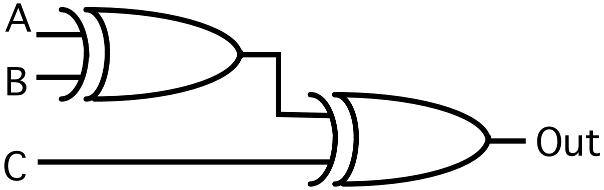
Debugging Complex Circuits

- Complex circuits require careful debugging
  - Rip up and retry?
- Ex. Debug a 9-input odd parity circuit
  - True if an odd number of inputs are true



Debugging Complex Circuits (cont.)





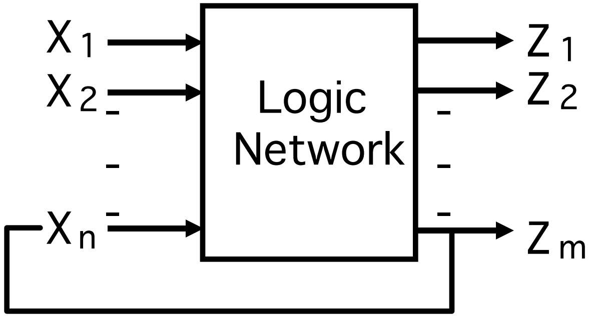
DON'

Debugging Approach

- Test all behaviors.
  - All combinations of inputs for small circuits, subcircuits.
- Identify any incorrect behaviors.
- Examine inputs and outputs to find earliest place where value is wrong.
  - Typically, trace backwards from bad outputs, forward from inputs.
  - Look at values at intermediate points in circuit.
- DO NOT RIP UP, DEBUG!

Combinational vs. Sequential Logic

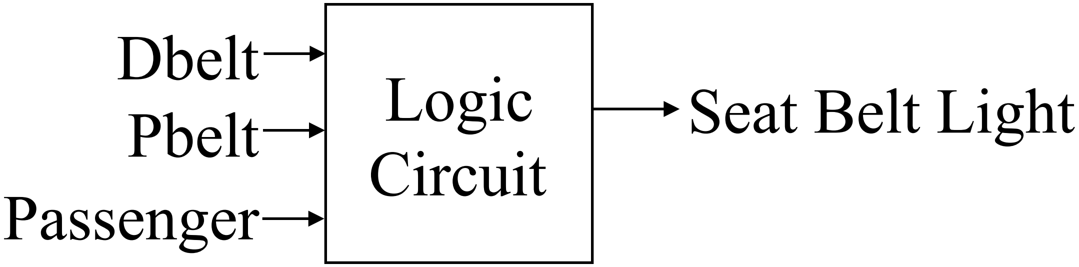
- Readings: 5-5.4.4



Network implemented from logic gates. The presence of feedback distinguishes between ***sequential*** and ***combinational*** networks.

Combinational logic

no feedback among inputs and outputs
outputs are a pure function of the inputs
e.g., seat belt light:
(Dbelt, Pbelt, Passenger) mapped into (Light)


Dbelt
Pbelt
Passenger
Logic Circuit
Seat Belt Light