Review Problem 4

- Simplify the following Boolean Equation

\[ AB + AC + \overline{AB} \]

\[ = AB + \overline{A}B + AC \quad (\text{comm.}) \]
\[ = (A+\overline{A})B + AC \quad (\text{dist.}) \]
\[ = B + AC \quad (\text{unity}) \]
Review Problem 5

Simplify the following Boolean Equation, starting with DeMorgan’s Law

\[ \bar{F} = AB + AC \]

\[ F = \overline{\overline{A\overline{B} + A\overline{C}}} \]

\[ = (A\overline{B})(A\overline{C}) \]

\[ = (A + B)(A + \overline{C}) \]

\[ = \overline{A}A + \overline{A}\overline{C} + A\overline{B} + B\overline{C} \]

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

\[ = \overline{A} + 2\overline{C} \]
Circuit Timing Behavior

- Simple model: gates react after fixed delay

![Circuit Diagram]

- Gate 1: Delay ~1 ns
Hazards/Glitches

- Circuit can temporarily go to incorrect states

Diagram:
- Copilot Autopilot Request
- Pilot in Charge?
- Pilot Autopilot Request
- Autopilot Engaged
- 3 gate delays
- 2 gate delays

Time lines for CAR, PIC, PAR, A, B, C, AE
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

FPGA CAD Tools

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
// Verilog code for AND-OR-INVERT gate

module AOI (F, A, B, C, D);
    output F;
    input A, B, C, D;
    assign F = ~(A & B) | (C & D);
endmodule

// 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
// 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
// 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 (.F(VB), .A(I), .B(SEL), .C(SELB), .D(J));
  not G3 (V, VB);
endmodule
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; .V(V)
    end

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

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

\[
\text{A} \\
\text{B 3-Parity Out} \\
\text{C}
\]

\[
\begin{array}{c}
\text{A} \\
\text{B} \\
\text{C} \\
\text{Out}
\end{array}
\]

\textbf{DON'T}
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)

```
   X_1  ![Logic Network]  Z_1
   X_2  ![Logic Network]  Z_2
   X_n  ![Logic Network]  Z_m
```

```
Dbelt  ![Logic Circuit]  Seat Belt Light
Pbelt
Passenger
```