Administrativia

• Homework 3 is out – due Monday 9/27, 12:00am
• Homework 4 out this week.
Agenda

• DeMorgan’s Law
  • Bubble pushing

• Karnaugh maps
  • POS
  • SOP

• Finite state machines
DeMorgan’s Law: Bubble Pushing

• $(x+y)' = x'y'$
• $(xy)' = x'+y'$
• Bubble = inversion (NOT)
DeMorgan’s Law: Bubble Pushing

• \((x+y)' = x'y' \subseteq\)
• \((xy)' = x'+y' \subseteq\)
• Bubble = inversion (NOT)

• For a single gate:
  • Swap AND for OR & vice versa
  • Backward pushing: add bubbles to input
  • Forward pushing: add bubbles to output
Bubble Pushing Example

\[ Y = \overline{(A+D)} \cdot C \cdot D \]

\[ Y = \overline{A} + B + C + D \]
SoP & PoS

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

SoP

\[ \text{out} = \overline{A} \overline{B} C + \overline{A} B \overline{C} \ldots \]

PoS

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

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

\[ (\overline{A} + B + C) \]
K-maps

• K-Maps: visual & systematic Boolean simplification

• 2 important Boolean identities:
  • \((1+A)=1\)
  • \((A+\bar{A})=1\)

• Leverages **gray coding** to organize neighboring minterms
  • Adjacent minterms only differ by a single bit!

• Key to solving: form groups of 1’s by multiples of 2
  • As large & as few as possible
  • Overlapping is OK, wrap boundary where possible
  • Write AND expression for each group
  • Make new SoP expression
K-map example

\( F(A,B) = \overline{A}B + \overline{A} \)

\( 1 + B = 1, \ A + \overline{A} = 1 \)

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

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

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

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

\( = A + B \)

\( = A + B \)
Simplification – Karnaugh maps (SOP)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ f = a' \overline{b} + a \overline{c} \]

\[ = a' (b' + c) \]

Gray coded
Simplification – Karnaugh maps (POS)

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ f = (a') (c' + b') \]
4-input K-map example

F(A,B) = \overline{A}B\overline{C}D + \overline{A}BCD + 
\overline{A}BCD + \overline{AB}\overline{CD} + 
A\overline{B}CD + ABC\overline{D} + 
AB\overline{CD} + ABCD + 
ABCD

\[ F(A,B) = \overline{C}D + \overline{A}BCD + \overline{AB}C + AB\overline{CD} + 
CD + A\overline{B}D \]
Finite state machines
FSM review

• Sequential circuit where output depends on present and past inputs

• Has finite number of states, and can only be in one state at a time

• Combinational logic used to calculate next state and output

• Represented by state transition diagram
## Moore vs. Mealy FSM

<table>
<thead>
<tr>
<th></th>
<th>Moore</th>
<th>Mealy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Output function</strong></td>
<td>based only on present state</td>
<td>based on both present state and input</td>
</tr>
<tr>
<td><strong># states</strong></td>
<td>usually more</td>
<td>fewer</td>
</tr>
<tr>
<td><strong>Output synchronous</strong></td>
<td>synchronous</td>
<td>asynchronous (can glitch w/ inputs)</td>
</tr>
<tr>
<td><strong>Output delay</strong></td>
<td>delayed by one clock cycles</td>
<td>immediately available w/ input</td>
</tr>
</tbody>
</table>
Example - Vending Machine FSM

• Dispenses a soda if it receives at least 25 cents
  • Doesn’t return change or rollover to next purchase

• Customer can insert three different coins:
  • Quarter – 25¢ – Q
  • Dime – 10¢ – D
  • Nickel – 5¢ – N

module vending_machine (inputs clk, rst, input Q, D, N 
output dispense)
Moore Vending Machine

module vending_machine(
  input clk, rst,
  input Q, D, N,
  output dispense
);

\[ Q = \text{Q high only} \]
\[ D = \text{D high only} \]
\[ N = \text{N high only} \]
\[ x = \overline{Q} \cdot D \cdot N \]
Mealy Vending Machine

module vending_machine(
    input clk, rst,
    input Q, D, N,
    output dispense
);

Fa21  EECS 151/251A Discussion 4  18
Verilog Implementation

• Two main sections:
  • State transition (sequential)
  • State/output logic (combinational)

module vending_machine()

// inputs, outputs, clk, rst

// define state bits

// define state names as local params

// state transitions

// next state and output logic
endmodule
Setup and state transitions

module vending_machine(
    input clk, rst,
    input Q, D, N,
    output dispense
);

reg [2:0] NS, CS;

localparam S0 = 3’d0,
    S5 = 3’d1,
    S10 = 3’d2,
    S15 = 3’d3,
    S20 = 3’d4,
    S25 = 3’d5;

always @(posedge clk) begin
    if (rst) CS <= S0;
    else CS <= NS;
end

...
Moore vs. Mealy combinational logic

**Moore Logic**

```verilog
always @(*) begin
    NS = CS;
    case (CS)
        S0: begin
            if (Q == 1'b1) NS = S25;
            if (D == 1'b1) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        S5: begin
            if (Q == 1'b1) NS = S25;
            if (D == 1'b1) NS = S15;
            if (N == 1'b1) NS = S10;
        end
        ... 
        S25: begin
            if (Q == 1'b1) NS = S25;
            if (D == 1'b1) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        default: NS = S0;
    endcase

    assign dispense = (CS == S25);
endmodule
```

**Mealy Logic**

```verilog
reg dispense;
always @(*) begin
    NS = CS;
    dispense = 1'b0;
    case (CS)
        S0: begin
            if (Q == 1'b1) begin
                NS = S0;
                dispense = 1'b1;
            end
            if (D == 1'b1) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        ... 
        S15: begin
            if (Q == 1'b1) begin
                NS = S0;
                dispense = 1'b1;
            end
            if (D == 1'b1) begin
                NS = S0;
                dispense = 1'b1;
            end
            if (N == 1'b1) NS = S10;
        end
        ... 
        default: NS = S0;
    endcase
end
endmodule
```