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

- \((x+y)’ = x’y’\)
- \((xy)’ = x’+y’\)
- 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

\[ \begin{align*}
\text{Swap} & : & & & \text{OR} & \rightarrow & \text{AND} \\
\text{Original} & : & & & & & \\
\text{Push} & : & & & & & \\
\end{align*} \]
Bubble Pushing Example
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

PoS
K-maps

• K-Maps: visual & systematic Boolean simplification

• 2 important Boolean identities:
  • (1+A)=1
  • (A+Ā)=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

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

Fa21  
EECS 151/251A Discussion 4
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>

<table>
<thead>
<tr>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fa21 EECS 151/251A Discussion 4
4-input K-map example

\[ F(A,B) = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}C\overline{D} + \overline{A}B\overline{C}\overline{D} + \overline{A}B\overline{C}D + AB\overline{C}\overline{D} + ABCD + ABCD + ABCD + ABCD \]
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>Output function</td>
<td></td>
<td></td>
</tr>
<tr>
<td># states</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output synchronous</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Output delay</td>
<td></td>
<td></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
Moore Vending Machine

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


Mealy Vending Machine

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

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
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

```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
```

```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) NS = S10;
            if (N == 1'b1) NS = S5;
        end
        ... 
        S25: 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
```
DeMorgan’s Law: Bubble Pushing

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

• \((x+y)' = x'y'\)
• \((xy)' = x'+y'\)
• 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
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

PoS
K-maps

• K-Maps: visual & systematic Boolean simplification
• 2 important Boolean identities:
  • (1+A)=1
  • (A+Ā)=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

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

Diagram:

00 01 11 10

0

1
Simplification – Karnaugh maps (POS)

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

00 01 11 10

0 1
4-input K-map example

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