Example: vending machine

- Release item after 15 cents are deposited
- Single coin slot for dimes, nickels
- No change
Example: vending machine (cont’d)

- Suitable abstract representation
  - tabulate typical input sequences:
    - 3 nickels
    - nickel, dime
    - dime, nickel
    - two dimes
  - draw state diagram:
    - inputs: N, D, reset
    - output: open chute
  - assumptions:
    - assume N and D asserted for one cycle
    - each state has a self loop for N = D = 0 (no coin)
Example: vending machine (cont’d)

- Minimize number of states - reuse states whenever possible

### Symbolic State Table

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Output Open</th>
</tr>
</thead>
<tbody>
<tr>
<td>0¢</td>
<td>0 0</td>
<td>0¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5¢</td>
<td>0 0</td>
<td>5¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10¢</td>
<td>0 0</td>
<td>10¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>15¢</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15¢ (open)</td>
<td>– –</td>
<td>15¢</td>
<td>1</td>
</tr>
</tbody>
</table>

### Diagram
Example: vending machine (cont’d)

- Uniquely encode states

<table>
<thead>
<tr>
<th>present state</th>
<th>inputs</th>
<th>next state</th>
<th>output</th>
<th>open</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>D N</td>
<td>D1 D0</td>
<td>open</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>0 1 0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>1 0 0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>– – –</td>
<td>– –</td>
<td>–</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>1 0 0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>1 1 0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>– – –</td>
<td>– –</td>
<td>–</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0 0</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>1 1 0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>1 1 0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>– – –</td>
<td>– –</td>
<td>–</td>
</tr>
<tr>
<td>1 1</td>
<td>– –</td>
<td>– – –</td>
<td>1 1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Moore implementation

- Mapping to logic

D1 = Q1 + D + Q0 N
D0 = Q0' N + Q0 N' + Q1 N + Q1 D
OPEN = Q1 Q0
Example: vending machine (cont’d)

- One-hot encoding

<table>
<thead>
<tr>
<th>present state</th>
<th>inputs</th>
<th>next state output</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q3 Q2 Q1 Q0</td>
<td>D N</td>
<td>D3 D2 D1 D0 open</td>
<td></td>
</tr>
<tr>
<td>0 0 0 1</td>
<td>0 0</td>
<td>0 0 0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>- -</td>
<td>- - - - -</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>0 0</td>
<td>0 0 1 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>- -</td>
<td>- - - - -</td>
<td></td>
</tr>
<tr>
<td>0 1 0 0</td>
<td>0 0</td>
<td>0 1 0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0 0 0 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>- -</td>
<td>- - - - -</td>
<td></td>
</tr>
<tr>
<td>1 0 0 0</td>
<td>- -</td>
<td>- - - - -</td>
<td></td>
</tr>
</tbody>
</table>

D0 = Q0 D’ N’
D1 = Q0 N + Q1 D’ N’
D2 = Q0 D + Q1 N + Q2 D’ N’
D3 = Q1 D + Q2 D + Q2 N + Q3
OPEN = Q3
Equivalent Mealy and Moore state diagrams

- **Moore machine**
  - outputs associated with state

- **Mealy machine**
  - outputs associated with transitions

```
Moore machine
  - outputs associated with state

Mealy machine
  - outputs associated with transitions
```

```
0¢ [0]  N' D' + Reset
  N [N']
  D [D]
  N+D [N']

10¢ [0]  N' D'
  N [N']
  D [D]
  N+D [N']

15¢ [1]  Reset'

(N' D' + Reset)/0
```

```
0¢  N' D'/0
  N/0
  D/1
  N+D/1

10¢  N' D'/0
  N/0
  D/1
  N+D/1

15¢  Reset'/1

Reset/0
```

Spring 2005  CSE370 - guest lecture
Example: Mealy implementation

<table>
<thead>
<tr>
<th>present state</th>
<th>inputs</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>D N</td>
<td>D1 D0 open</td>
<td></td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>1 1 1</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>0 1</td>
<td>1 1 1</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
D0 = Q0'N + Q0N' + Q1N + Q1D
\]

\[
D1 = Q1 + D + Q0N
\]

\[
OPEN = Q1Q0 + Q1N + Q1D + Q0D
\]
Example: Mealy implementation

\[
\begin{align*}
D_0 &= Q_0'N + Q_0N' + Q_1N + Q_1D \\
D_1 &= Q_1 + D + Q_0N \\
OPEN &= Q_1Q_0 + Q_1N + Q_1D + Q_0D
\end{align*}
\]

make sure OPEN is 0 when reset  
– by adding AND gate
Vending machine: Moore to synch. Mealy

- OPEN = Q1Q0 creates a combinational delay after Q1 and Q0 change in Moore implementation
- This can be corrected by retiming, i.e., move flip-flops and logic through each other to improve delay
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
- Implementation now looks like a synchronous Mealy machine
  - it is common for programmable devices to have FF at end of logic
Vending machine: Mealy to synch. Mealy

- OPEN.d = Q1Q0 + Q1N + Q1D + Q0D
- OPEN.d = (Q1 + D + Q0N)(Q0'N + Q0N' + Q1N + Q1D)
  = Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D
Hardware Description Languages and Sequential Logic

- **Flip-flops**
  - representation of clocks - timing of state changes
  - asynchronous vs. synchronous

- **FSMs**
  - structural view (FFs separate from combinational logic)
  - behavioral view (synthesis of sequencers – not in this course)

- **Data-paths** = data computation (e.g., ALUs, comparators) + registers
  - use of arithmetic/logical operators
  - control of storage elements
Example: reduce-1-string-by-1

- Remove one 1 from every string of 1s on the input
Verilog FSM - Reduce 1s example

- **Moore machine**

  module reduce (clk, reset, in, out);
  input clk, reset, in;
  output out;

  parameter zero = 2'b00;
  parameter one1 = 2'b01;
  parameter two1s = 2'b10;

  reg out;
  reg [2:1] state;  // state variables
  reg [2:1] next_state;

  always @(posedge clk)
    if (reset) state = zero;
    else       state = next_state;

  state assignment (easy to change, if in one place)
always @ (in or state)
  case (state)
    zero:
      // last input was a zero
      begin
        if (in) next_state = one;
        else    next_state = zero;
      end
    one:
      // we've seen one 1
      begin
        if (in) next_state = two1s;
        else    next_state = zero;
      end
    two1s:
      // we've seen at least 2 ones
      begin
        if (in) next_state = two1s;
        else    next_state = zero;
      end
  endcase
endmodule

always @(in)
  begin
    if (in)
      begin
        if (in) next_state = one;
        else    next_state = zero;
      end
    end
Mealy Verilog FSM

module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;
    reg out;
    reg state; // state variables
    reg next_state;

    always @(posedge clk)
        if (reset) state = zero;
        else state = next_state;

    always @(in or state)
        case (state)
            zero: // last input was a zero
                begin
                    out = 0;
                    if (in) next_state = one;
                    else next_state = zero;
                end
            one: // we've seen one 1
                begin
                    if (in) begin
                        next_state = one; out = 1;
                    end else begin
                        next_state = zero; out = 0;
                    end
                endcase
        endmodule
Synchronous Mealy Machine

module reduce (clk, reset, in, out);
    input clk, reset, in;
    output out;
    reg out;
    reg state; // state variables

    always @(posedge clk)
        if (reset) state = zero;
        else
            case (state)
                zero: // last input was a zero
                    begin
                        out = 0;
                        if (in) state = one;
                        else    state = zero;
                    end
                one: // we've seen one 1
                    if (in) begin
                        state = one; out = 1;
                    end else begin
                        state = zero; out = 0;
                    end
            endcase
endmodule
State assignment

- Choose bit vectors to assign to each “symbolic” state
  - with n state bits for m states there are \( \frac{2^n!}{(2^n - m)!} \)
    \[ \log n \leq m \leq 2^n \]
  - \( 2^n \) codes possible for 1st state, \( 2^n - 1 \) for 2nd, \( 2^n - 2 \) for 3rd, ...
  - huge number even for small values of n and m
    - intractable for state machines of any size
    - heuristics are necessary for practical solutions
  - optimize some metric for the combinational logic
    - size (amount of logic and number of FFs)
    - speed (depth of logic and fanout)
    - dependencies (decomposition)
State assignment strategies

- **Possible strategies**
  - sequential – just number states as they appear in the state table
  - random – pick random codes
  - one-hot – use as many state bits as there are states (bit=1 → state)
  - output – use outputs to help encode states
  - heuristic – rules of thumb that seem to work in most cases

- **No guarantee of optimality – another intractable problem**
Current state assignment approaches

- For tight encodings using close to the minimum number of state bits
  - best of 10 random seems to be adequate (averages as well as heuristics)
  - heuristic approaches are not even close to optimality
  - used in custom chip design

- One-hot encoding
  - easy for small state machines
  - generates small equations with easy to estimate complexity
  - common in FPGAs and other programmable logic

- Output-based encoding
  - ad hoc - no tools
  - most common approach taken by human designers
  - yields very small circuits for most FSMs
  - popular in PLDs
State machines and PLDs

- Moore and synchronous Mealy most common
- Output-directed state assignment
  - All outputs already have FFs and are fed back in as input to logic array
  - Use these as part of the state register
  - Add only as many extra states bits as needed to make all state codes unique
Implementation using PALs

- Programmable logic building block for sequential logic
  - macro-cell: FF + logic
    - D-FF
    - two-level logic capability like PAL (e.g., 8 product terms)
### 22V10 PAL

- Combinational logic elements (SoP)
- Sequential logic elements (D-FFs)
- Up to 10 outputs
- Up to 10 FFs
- Up to 22 inputs
22V10 PAL Macro Cell

- Sequential logic element + output/input selection
Vending machine example (Moore PLD mapping)

\[ \begin{align*}
D0 &= \text{reset}'(Q0'N + Q0N' + Q1N + Q1D) \\
D1 &= \text{reset}'(Q1 + D + Q0N) \\
\text{OPEN} &= Q1Q0
\end{align*} \]
Vending machine (synch. Mealy PLD mapping)

\[ \text{OPEN} = \text{reset}'(Q1Q0N' + Q1N + Q1D + Q0'ND + Q0N'D) \]