Handout on the Design of a Diabolical Coke™ Machine

In-class Example of a Finite State Machine

System Block Diagram:

Comments:

The system outputs drive a mechanical cup drop mechanism and three solenoid valves to dispense carbonated water and two flavors. Shutoff is insured by both timing (fill is to last two and only two clock cycles) and weight of the cup (FULL signal from a spring-loaded switch). The signal for the customer having deposited sufficient coins is from a mechanical switch that closes only transiently with multiple bounces. The REQ/ACK subsystem implemented in a REQ/ACK scheme is needed to capture the deposit because the clock is not sufficiently fast to do so.

State Table:

<table>
<thead>
<tr>
<th>State</th>
<th>State Name</th>
<th>Function</th>
<th>Style 1 (Binary)</th>
<th>Style 2 (ad hoc)</th>
<th>Style 3 (one-hot)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>waiting</td>
<td>Wait for money</td>
<td>000</td>
<td>0000</td>
<td>000000</td>
</tr>
<tr>
<td>B</td>
<td>release</td>
<td>Drop a cup</td>
<td>001</td>
<td>1000</td>
<td>000011</td>
</tr>
<tr>
<td>C</td>
<td>soda_inh1</td>
<td>Dispense soda with inh 1 cycle</td>
<td>010</td>
<td>0001</td>
<td>000101</td>
</tr>
<tr>
<td>D</td>
<td>soda_inh2</td>
<td>Dispense soda with inh 2 cycle</td>
<td>011</td>
<td>0011</td>
<td>001001</td>
</tr>
<tr>
<td>E</td>
<td>soda_exc1</td>
<td>Dispense soda with exc 1 cycle</td>
<td>100</td>
<td>0101</td>
<td>010001</td>
</tr>
<tr>
<td>F</td>
<td>soda_exc2</td>
<td>Dispense soda with exc 2 cycle</td>
<td>101</td>
<td>0111</td>
<td>100001</td>
</tr>
</tbody>
</table>
### Transition Table:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>A</td>
<td>000</td>
<td>A</td>
<td>000</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>A</td>
<td>000</td>
<td>B</td>
<td>001</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>B</td>
<td>001</td>
<td>E</td>
<td>100</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>B</td>
<td>001</td>
<td>C</td>
<td>010</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>C</td>
<td>010</td>
<td>F</td>
<td>101</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>C</td>
<td>010</td>
<td>D</td>
<td>011</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>E</td>
<td>100</td>
<td>F</td>
<td>101</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>E</td>
<td>100</td>
<td>D</td>
<td>011</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>D</td>
<td>011</td>
<td>A</td>
<td>000</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>F</td>
<td>101</td>
<td>A</td>
<td>000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>A</td>
<td>0000</td>
<td>A</td>
<td>0000</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>A</td>
<td>0000</td>
<td>B</td>
<td>1000</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>B</td>
<td>1000</td>
<td>E</td>
<td>0101</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>B</td>
<td>1000</td>
<td>C</td>
<td>0001</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>C</td>
<td>0001</td>
<td>F</td>
<td>0111</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>C</td>
<td>0001</td>
<td>D</td>
<td>0011</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>E</td>
<td>0101</td>
<td>F</td>
<td>0111</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>E</td>
<td>0101</td>
<td>D</td>
<td>0011</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>D</td>
<td>0011</td>
<td>A</td>
<td>0000</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>F</td>
<td>0111</td>
<td>A</td>
<td>0000</td>
</tr>
</tbody>
</table>

### State-bit Assignment Styles:

The motivation for the second and third choices of state encoding was the possibility of simplifying the logic for next state and output decoding, particularly in the case of style #2 the criterion was output decoding. Here is the output decoding logic as it works out in terms of the state variables Qn, Qn-1,...,Q0.

<table>
<thead>
<tr>
<th>Output</th>
<th>Style # 1</th>
<th>Style # 2</th>
<th>Style # 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUP</td>
<td>$Q^2 \cdot \overline{Q} \cdot Q^0$</td>
<td>$Q^3$</td>
<td>$QB$</td>
</tr>
<tr>
<td>INH_V</td>
<td>$Q \cdot FULL$</td>
<td>$Q^2 \cdot \overline{Q} \cdot FULL$</td>
<td>$(QC + QD) \cdot \overline{FULL}$</td>
</tr>
<tr>
<td>EXC_V</td>
<td>$Q^2 \cdot FULL$</td>
<td>$Q^2 \cdot \overline{Q} \cdot FULL$</td>
<td>$(QF + QE) \cdot \overline{FULL}$</td>
</tr>
<tr>
<td>SODA_V</td>
<td>$(Q^2 + Q^1) \cdot FULL$</td>
<td>$Q^0 \cdot FULL$</td>
<td>$QA \cdot QB \cdot FULL$</td>
</tr>
</tbody>
</table>
The gain in output logic is minimal in this example. In a more serious design one would face the problem of deciding which assignments made the best system and the criteria for that are the system speed and its cost or equivalently the silicon area or number of gates. The approach would be to use static timing analysis to see if any one choice would reduce logic delays enough to speed the system up. That would be optimized against the size of the system. With luck, a design exists that minimizes both time and dollars.

**Verilog Functional Timing Diagram Simulating a Soda Sale with Mix of Half with INH and Half with EXC**

<table>
<thead>
<tr>
<th>Name</th>
<th>0</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
<th>3000</th>
<th>3500</th>
<th>4000</th>
<th>4500</th>
<th>5000</th>
<th>5500</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inh</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>soda_v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inh_v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>exc_v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pres_state</td>
<td>00</td>
<td>03</td>
<td>05</td>
<td>21</td>
<td>09</td>
<td>21</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>next_state</td>
<td>00</td>
<td>03</td>
<td>05</td>
<td>09</td>
<td>21</td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Verilog Files: All Bit-assignment Styles

//----------------------------------------------------------------------------------
// Title : coke_style1
// Design : coke_tm
// Company : Brown University
// : School of Engineering
//
// Description : Example of Verilog FSM coding - Coke (TM) machine
//
//----------------------------------------------------------------------------------

\timescale 1 ns / 1 ps

module coke_style1 (input full, inh, coin, clk, output reg cup, inh_v, exc_v, soda_v);

reg [2:0] pres_state, next_state; // State variables

// An "initial" block affects only simulation - you cannot count on it for operation
initial pres_state = 3'b000;

// State bit assignments
parameter [2:0] waiting = 3'b000,
cup_drop = 3'b001,
dispense_i1 = 3'b010,
dispense_i2 = 3'b011,
dispense_e1 = 3'b100,
dispense_e2 = 3'b101;

// State register
always @(posedge clk) pres_state <= next_state;

// Next state logic
always @(pres_state, coin, inh) begin
  case (pres_state)
    waiting: if (coin == 1'b1) next_state = cup_drop;
    else next_state = waiting;
    cup_drop: if (inh == 1) next_state = dispense_i1;
    else next_state = dispense_e1;
    dispense_i1: if (inh == 1) next_state = dispense_i2;
    else next_state = dispense_e2;
    dispense_e1: if (inh == 1) next_state = dispense_i2;
    else next_state = dispense_e2;
    dispense_i2: next_state = waiting;
    dispense_e2: next_state = waiting;
    default next_state = waiting;
  endcase
end

// Output logic - note that outputs are reg type even though not ff outputs always @ (pres_state, full) begin
  if (pres_state == cup_drop) cup = 1;
  else cup = 0;
  if (pres_state == dispense_i1 | dispense_i2) begin
    inh_v = ~full;
    soda_v = ~full;
  end
  if (pres_state == dispense_e1 | dispense_e2) begin
    exc_v = ~full;
    soda_v = ~full;
  end
end
endmodule

//-- -------------------------------------------------------
//--
//-- Title : coke_style2
//-- Design : coke_tm
//-- Company : Brown University
//-- : School of Engineering
//--
//-- Description : Example of Verilog FSM coding - Coke (TM) machine
//--
//-- -------------------------------------------------------

`timescale 1 ns / 1 ps

//-- {module {coke_style2}} - ad hoc state bit assignments for simplified output logic
module coke_style2 (input full ,inh ,coin ,clk , output cup ,inh_v ,exc_v ,soda_v);

reg [3:0] pres_state, next_state;

// An "initial" block affects only simulation - you cannot count on it for operation
initial pres_state = 4'b0000;

// State bit assignments - ad hoc for simple outputs
parameter [3:0] waiting = 4'b0000,
  cup_drop = 4'b1000,
  dispense_i1 = 4'b0001,
  dispense_i2 = 4'b0011,
  dispense_e1 = 4'b0101,
  dispense_e2 = 4'b0111;

// State register
always @ (posedge clk) pres_state <= next_state;
Next state logic
always @ (pres_state, coin, inh) begin
    case (pres_state)
        waiting: if (coin == 1'b1) next_state = cup_drop;
                 else next_state = waiting;
        cup_drop: if (inh == 1) next_state = dispense_i1;
                  else next_state = dispense_e1;
        dispense_i1: if (inh == 1) next_state = dispense_i2;
                     else next_state = dispense_e2;
        dispense_e1: if (inh == 1) next_state = dispense_i2;
                     else next_state = dispense_e2;
        dispense_i2: next_state = waiting;
        dispense_e2: next_state = waiting;
        default next_state = waiting;
    endcase
end

Output - expose direct boolean equations to show style effect

assign cup = pres_state[3];
assign soda_v = pres_state[0] & ~full;
assign exc_v = pres_state[2] & ~full;
endmodule

-------------------------------
Title    : coke_style3
Design   : coke_tm
Company   : Brown University
           : School of Engineering
//
// Description : Example of Verilog FSM coding - Coke (TM) machine
//
-------------------------------
`timescale 1 ns / 1 ps

//{module {coke_style3}} - One-hot design for simplified state decoding
module coke_style3 (input full ,inh ,coin ,clk , output cup ,inh_v ,exc_v ,soda_v );

reg [5:0] pres_state, next_state;
// An "initial" block affects only simulation - you cannot count on it for operation
initial pres_state = 6'b000000;

// State bit assignments - One-hot
parameter [5:0] waiting = 6'b000000,
cup_drop = 6'b0000011,
dispense_i1 = 6'b000101,
dispense_i2 = 6'b001001,
dispense_e1 = 6'b010001,
dispense_e2 = 6'b100001;

// State register
always @ (posedge clk) pres_state <= next_state;

// Next state logic
always @ (pres_state, coin, inh) begin
  case (pres_state)
    waiting: if (coin == 1'b1) next_state = cup_drop;
    else next_state = waiting;
    cup_drop: if (inh == 1) next_state = dispense_i1;
    else next_state = dispense_e1;
    dispense_i1: if (inh == 1) next_state = dispense_i2;
    else next_state = dispense_e2;
    dispense_e1: if (inh == 1) next_state = dispense_i2;
    else next_state = dispense_e2;
    dispense_i2: next_state = waiting;
    dispense_e2: next_state = waiting;
    default next_state = waiting;
  endcase
end

// Output - expose direct Boolean equations to show effect of style
assign cup = pres_state[1];
assign soda_v = ~pres_state[1] & pres_state[0] & ~full;
assign inh_v = (pres_state[2] | pres_state[3]) & ~full;
assign exc_v = (pres_state[4] | pres_state[5]) & ~full;
endmodule