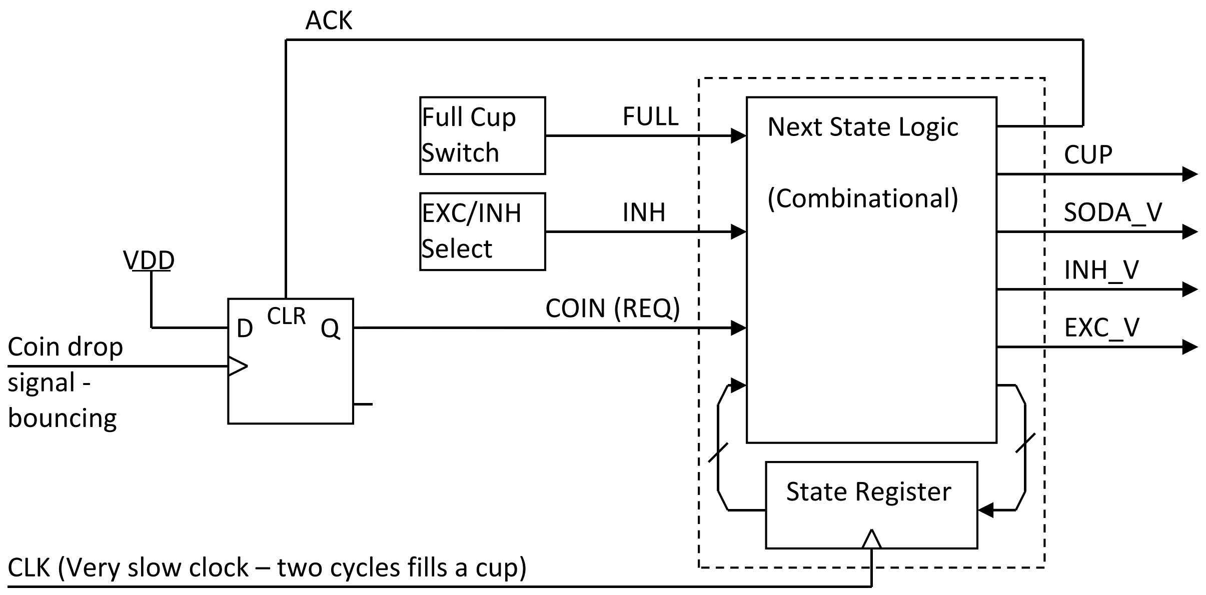
Handout on the Design of a Diabolical Coke ™ Machine

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.

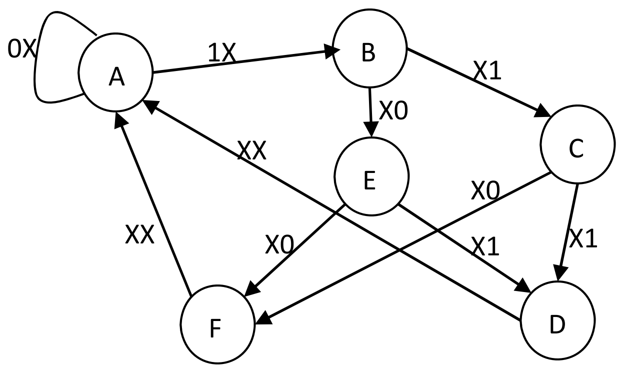
State Table:

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

Transition Table:

COIN	INH	Pres. State	Q[2:0]		Next State	D[2:0]
0	X	A	000		A	000
1	X	A	000		B	001
X	0	B	001		E	100
X	1	B	001		C	010
X	0	C	010		F	101
X	1	C	010		D	011
X	0	E	100		F	101
X	1	E	100		D	011
X	X	D	011		A	000
X	X	F	101		A	000

COIN	INH	Pres. State	Q[3:0]		Next State	D[3:0]
0	X	A	0000		A	0000
1	X	A	0000		B	1000
X	0	B	1000		E	0101
X	1	B	1000		C	0001
X	0	C	0001		F	0111
X	1	C	0001		D	0011
X	0	E	0101		F	0111
X	1	E	0101		D	0011
X	X	D	0011		A	0000
X	X	F	0111		A	0000



XX = COIN, INH

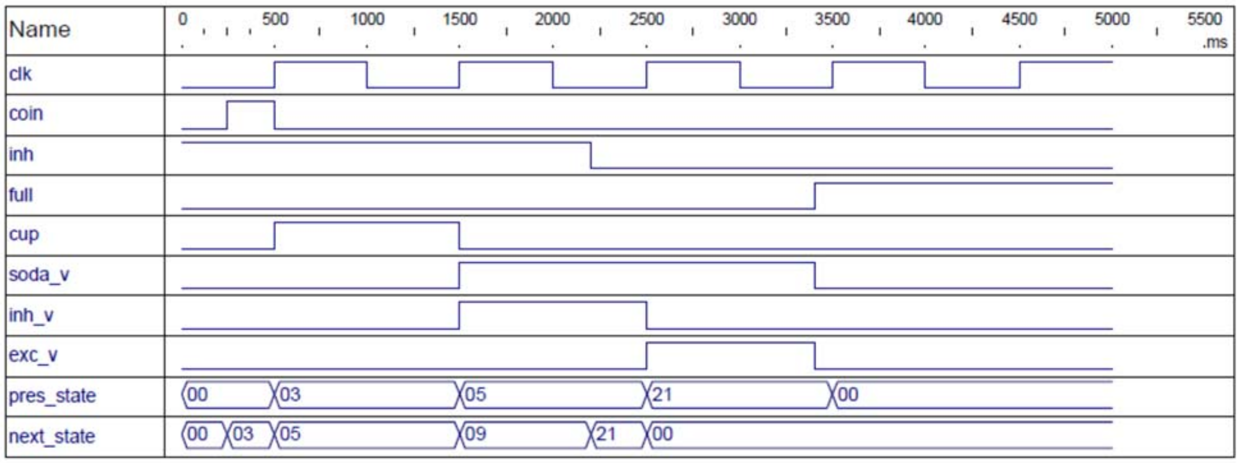
State-bit Assignment Styles:

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.

Output	**Style # 1**	**Style # 2**	**Style # 3**
CUP	$\overline{Q2}\cdot\overline{Q1}\cdot Q0$	$Q3$	QB
INH_V	$Q1\cdot\overline{FULL}$	$\overline{Q2}\cdot\overline{Q0}\cdot\overline{FULL}$	$(QC+QD)\cdot\overline{FULL}$
EXC_V	$Q2\cdot\overline{FULL}$	$\overline{Q2}\cdot\overline{Q0}\cdot\overline{FULL}$	$(QF+QE)\cdot\overline{FULL}$
SODA_V	$(Q2+Q1)\cdot\overline{FULL}$	$Q0\cdot\overline{FULL}$	$QA\cdot\overline{QB}\cdot\overline{FULL}$

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



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 inh_v = ~pres_state[2] & 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'b000011,
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
```