Finite State Machines
Register Transfer Level Abstraction

Finite State Machine

Control path

Data path

Hardware blocks
sequential
group sequential blocks

Clk
The Finite State Machine Abstraction

➤ Changes state according to different inputs. In each state there is a unique output. (Moore machine). The outputs may also depend directly on inputs (Mealy machine).

➤ FSM are sequential systems (use flip-flops to make transitions). In each state combinational circuits produce the outputs from inputs.

➤ Only use synchronous state machines in this course

➤ Support material

Finite State Machine Structure

- State machines consist of a state memory which in implementation is a bank of n flip-flops. For n flip-flops there are $2^n$ possible states representable. **WHY?**
- The flip-flops are in parallel and undergo their transitions (change state) on the active edge of a common clock - synchronous state machine.
- The next state is determined by the next state logic, $F$ which is a function of the inputs and the current state.
- The output logic, $G$ determines the output as a function of the current state and the (possibly) the inputs.
- Thus we see that a synchronous FSM consists of two combinational bits (the input and output logic blocks) and one sequential bit (the state memory block).
Describing FSMs Mathematically

Equations. Moore machine:

\[ s(t + 1) = F(s(t), x(t)) \quad \text{– state equation} \]
\[ z(t) = G(s(t)) \quad \text{– output equation} \]

Equations. Mealy machine:

\[ s(t + 1) = F(s(t), x(t)) \quad \text{– state equation} \]
\[ z(t) = G(s(t), x(t)) \quad \text{– output equation} \]
\[ x(t) \text{ - input, } s(t) \text{ - state, } z(t) \text{ - output} \]
State Machine Schematically: Moore Machine

State-machine structure (Moore)

- Inputs
- Next-state Logic $F$
- Excitation
- State Memory
- Current state
- Output Logic $G$
- Outputs
- Clock signal
- Typically edge-triggered D flip-flops
- Output depends on state only
State Machine Schematically: Mealy Machine

State-machine structure (Mealy)

- Next-state Logic $F$
- State Memory
- Output Logic $G$
- Outputs
- Outputs depend on state and input
- Clock signal
- Typically edge-triggered D flip-flops
- Inputs

ENGN3213: Digital Systems and Microprocessors L#12-13
Representations of Finite State Machines
State Diagram Representation

The state is a description of a system characterised by a given set of inputs

- States are symbolised by a bubble in the form of a circle or a box.
- Loops or branches between the state bubbles represent state transitions with the input condition for the transition written on them.
- For a Mealy machine the outputs depend on both the input and the state. Thus they are written underneath the inputs on the loops.
- The outputs of a Moore machine can be written inside the state bubbles.
State Diagrams

Moore Machine

Mealy Machine
Pedestrian Traffic Controller (Moore machine)

- **s0**: G, HALT
- **s1**: Y, HALT
- **s2**: R, WALK
- **s3**: R, HALT

Transitions:
- **W** from s0 to s1
- **W** from s1 to s2
- **RESET** from s2 to s3
- **RESET** from s3 to s0
A very useful representation of the FSM is the **State Transition Table** representation (also called a **Next State Table**)

- The left-most column gives the present state of the system.
- The second column gives the **inputs**.
- The third column gives the **next state** to which a transition occurs given the current state and the stated set of inputs.
- The remaining columns are the **outputs**.
### Pedestrian Traffic Controller State Transition Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0 if $\overline{W}$ else s1 if $W$</td>
<td>G, HALT</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>Y, HALT</td>
</tr>
<tr>
<td>s2</td>
<td>s3</td>
<td>R, WALK</td>
</tr>
<tr>
<td>s3</td>
<td>s0</td>
<td>R, HALT</td>
</tr>
</tbody>
</table>
### FSM State Transition Table

<table>
<thead>
<tr>
<th>state</th>
<th>F[2:0]</th>
<th>Next state</th>
<th>IREn</th>
<th>PCEn</th>
<th>AccEn</th>
<th>M[1:0]</th>
<th>Xsel</th>
<th>Ysel</th>
<th>Asel</th>
<th>Ren</th>
<th>Wen</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>xxx</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>000</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>xxx</td>
<td>1</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>01</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>11</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>N</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Z</td>
<td>00</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>xx</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Notes:**
- N and Z are the Negative and Zero state of the Accumulator, respectively. (used to reduce the size of the table, as drawn)
- If a value is not going to be latched it doesn’t matter what it is! (e.g. ALU output for STO)
- STP operates by remaining in its evaluation state.
State Machine VERILOG Representation (not unique)

➤ The transition between states is a sequential operation (in the sense of a sequential circuit)
➤ In a FSM the actions that produce outputs from the inputs are always combinational
➤ Follow Cummings:
  ➤ State transitions model with **sequential ALWAYS block**
  ➤ Next state logic model with **combinational ALWAYS block**
  ➤ Output logic model with **combinational ALWAYS block**
Two State Moore Machine in Verilog

module FSM(clk,...)

parameter S0 = 1'b0;
parameter S1 = 1'b1;

always @(posedge clk) state <= Snext;       //STATE MEMORY -> SEQUENTIAL

always @(*) begin
    case(state)
        S0:
            if(something) Snext = S1;
            else Snext = S0;
        S1:
            if(something else) Snext = S0;
            else Snext = S1;
        default:
            Snext = S0;
    endcase
end

always @(state)                   //OUTPUT LOGIC
    case(state)
        S0:
            Output = f(state);
        S1:
            Output = g(state);
        default:
            Output = h(state);
    endcase
endmodule
Two State Mealy Machine in Verilog

module FSM(clk, ....)

parameter S0 = 1'b0;
parameter S1 = 1'b1;

always @(posedge clk) state <= Snext;  // STATE MEMORY -> SEQUENTIAL

always @(*) begin
  // NEXT STATE LOGIC
  case(state)
    S0:
      if(something) Snext = S1;
      else Snext = S0;
    S1:
      if(something_else) Snext = S0;
      else Snext = S1;
    default:
      Snext = S0;
  endcase
  end

always @(state or inputs)  // OUTPUT LOGIC
  case(state)
    S0:
      Output = f(inputs, state);
    S1:
      Output = g(inputs, state);
    default:
      Output = h(inputs, state);
  endcase
endmodule
State Machine Electronic Representation

Combinational

Next State
Logic

F(inputs, State)

Sequential

D Q

D Q

D Q

Combinational

Output
Logic

G(state, input?)

Inputs

Outputs
Excitation and Characteristic equations

The State transition equation consists of an excitation equation and a characteristic equation.

The Excitation equation specifies the input applied to a flip-flop device as a function of current state and inputs.

The Characteristic equation specifies the next state of a flip-flop device as a function of its excitation. That is, it is the state transition equation of flip-flop itself.

- D flip-flop: $Q^* = D$
- J-K flip-flop: $Q^* = JQ + KQ$
- T flip-flop with enable: $Q^* = TQ + TQ$
State Machine Equations

Combinational

Sequential

Next State Logic

F(inputs, State)

Outputs

Inputs

Excitation Equation

Characteristic Equation

Combinational

Output Logic

G(state, input?)

Outputs
Observation about FSMs and VERILOG

```verilog
module top_module(clk, RESET, other_inputs, Q1, ..., QN, other_outputs);
    excitation_logic el(input1, input2, ..., Qs, Q*s);
    D-flip-flop dff1(clk, Q1*, Q1, RESET);
    ...
    D-flip-flop dffN(clk, QN*, QN, RESET);
endmodule

module excitation_logic(input1, input2, ..., Qs, Q*s);
    assign Q1* = F(Q1, ..., QN);
    ...
    assign QN* = F(Q1, ..., QN);
endmodule

module D-flip-flop(clk, Q*, Q, RESET);
    always @ (posedge clk) begin
        if (RESET == 0) Q <= Q*;
        if (RESET == 1) Q <= 0;
    end
endmodule
```
Examples of FSMs: Alarm Circuit (Moore machine)
Examples of FSMs: Alarm Circuit State Diagram

\[
\begin{array}{c}
\text{S0 Armed} \\
\text{Tr}
\end{array}
\quad
\begin{array}{c}
\text{S1 Sound} \\
\text{Rst}
\end{array}
\]

\[
\begin{array}{c}
\text{Tr} \\
\text{Rst}
\end{array}
\]
Examples of FSMs: Alarm Circuit (Moore machine)

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0 trip=0</td>
<td>s1 trip=1</td>
<td>0 Silent</td>
</tr>
<tr>
<td>s1</td>
<td>s0 reset=1</td>
<td>s1 reset = 0</td>
<td>1 Sound</td>
</tr>
</tbody>
</table>
FSM Analysis Steps

Starting from a FSM Boolean logic diagram (i.e. electronic circuit):

1. Determine the state transition and output functions F and G.
   a) Find the excitation equation.
   b) Using the characteristic equation, determine the corresponding
      next-state values (trivial with D Flip-Flop)
   c) Find the output equations

2. Construct state/output table: For each state/input combination, determine
   the next state and output values

3. Draw the state diagram
Design and Synthesis of Finite State Machines
Finite State Machine Design and Synthesis

- Design is the creative part, like writing a program
- The design process is actually the opposite process of the state machine analysis.
- Synthesis is like turning the crank, like the Xilinx compiler does
- Recall the analysis steps:
  1. Determine the state transition and output functions
  2. Construct the next-state and output tables
  3. Draw the state diagram
Finite-State Machine Design Steps

1. Determine the inputs / outputs. Determine the states and give them mnemonic names
2. Draw up a state diagram and a next state table.
3. Render the inputs, outputs and states in binary format (VERILOG parameter).
4. Draw an excitation table
5. Draw an Output table
6. Use K-maps to obtain produce minimal next state and output combinational logic.
7. Use the standard VERILOG formulation to simulate your design and check for correct operation. Revise as appropriate.
8. Check for potential practical problems (e.g. non-ideal effects).
Design Examples

1. The alarm controller (Minimal FSM, Straight Karnaugh map)
2. Pedestrian Traffic Controller (Minimal FSM, Straight Karnaugh map)
3. Vending Machine Controller (One-hot)
4. Keyboard (Verilog)
Alarm Controller (Minimal FSM, Straight Karnaugh map)

Problem Statement:

1. The alarm should sound when the beam is broken.

2. The alarm should reset from the “sounding” to the “armed” (silent) state when a reset button is pushed (level triggering) regardless of the status of the beam.
Alarm Controller State Diagram

State Diagram:

- **S0 Armed**
  - Transition labeled $\bar{T_r}$
  - Transition labeled $R_{st}$

- **S1 Sound**
  - Transition labeled $T_r$
  - Transition labeled $R_{st}$
# Alarm Controller: Excitation Table

- A-armed,
- S-sound
- T-trip level and
- R-reset level.

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Next</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T'R'</td>
<td>A</td>
<td>Silent</td>
</tr>
<tr>
<td>A</td>
<td>T'R</td>
<td>A</td>
<td>Silent</td>
</tr>
<tr>
<td>A</td>
<td>TR'</td>
<td>S</td>
<td>Silent</td>
</tr>
<tr>
<td>A</td>
<td>TR</td>
<td>A</td>
<td>Silent</td>
</tr>
<tr>
<td>S</td>
<td>T'R'</td>
<td>S</td>
<td>Sound</td>
</tr>
<tr>
<td>S</td>
<td>T'R</td>
<td>A</td>
<td>Sound</td>
</tr>
<tr>
<td>S</td>
<td>TR'</td>
<td>S</td>
<td>Sound</td>
</tr>
<tr>
<td>S</td>
<td>TR</td>
<td>A</td>
<td>Sound</td>
</tr>
</tbody>
</table>
### Alarm Controller: Excitation Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T'R'$</td>
<td>$T'R$</td>
<td>$TR$</td>
<td>$TR'$</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>A</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>S</td>
<td>S</td>
<td>A</td>
<td>A</td>
<td>S</td>
</tr>
</tbody>
</table>

Now we need to code the symbols (states/output) into binary numbers (note the use of the Gray code)

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>States</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Alarm Controller Excitation Table

The entries in the excitation table are both the next states in response to the inputs, but also the input states to the input logic that produces the input excitation.

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/ TR</td>
<td>00</td>
<td>01</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

*Diagram of input logic and D Flip Flop with inputs T and R, and output Q*
## Alarm Controller Excitation Truth Table

<table>
<thead>
<tr>
<th>T</th>
<th>R</th>
<th>S</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Alarm Controller: Input Logic

Do Karnaugh maps to work out the Boolean input logic.

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

\[ Q = T \cdot \overline{R} + T \cdot \overline{S} + \overline{R} \cdot S \]  \hfill (1)
Alarm Controller Circuit
**Alarm Controller: Comments**

- Is the input $T \cdot \overline{R}$ really necessary?
- Remember to synchronise the level sensitive inputs $T$ and $R$ to the clock.
- Are there any glitches produced according to the truth table?
Pedestrian Traffic Controller (Minimal FSM, Straight Karnaugh map)

Pedestrian Traffic Controller

• **Problem Statements**: Design a traffic light controller which has two input buttons: ‘walk’ by pedestrian and ‘reset’, and five output lights: ‘walk/halt’ for people crossing, ‘red/yellow/green’ for cars. If the walk button pressed, the green light should change to yellow and red, then halt to walk light.
s0
G, HALT

W

s1
Y, HALT

W

s2
R, WALK

W

s3
R, HALT

RESET
Traffic Controller: State Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0 if $\bar{W}$ else s1 if $W$</td>
<td>G, HALT</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>Y, HALT</td>
</tr>
<tr>
<td>s2</td>
<td>s3</td>
<td>R, WALK</td>
</tr>
<tr>
<td>s3</td>
<td>s0</td>
<td>R, HALT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$W=0$</td>
<td>$W=1$</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>
Pedx Controller: Excitation Table

Code the symbols (states/output) into binary numbers (use Gray code order)

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next State $W = 0$</th>
<th>Next State $W = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>00</td>
<td>00</td>
</tr>
</tbody>
</table>

One Karnaugh map for each next state...

<table>
<thead>
<tr>
<th>Q1,Q2</th>
<th>Next State $W = 0$</th>
<th>State $W = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q1,Q2</th>
<th>Next State $W = 0$</th>
<th>State $W = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Thus the excitation logic becomes $Q_1 \land Q_2$ and $\overline{Q_2} \cdot (Q_1 + W)$. 

**Pedx Controller: Excitation**
Traffic Controller: State Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0 if $\overline{w}$ else s1 if $W$</td>
<td>G, HALT</td>
</tr>
<tr>
<td>s1</td>
<td>s2</td>
<td>Y, HALT</td>
</tr>
<tr>
<td>s2</td>
<td>s3</td>
<td>R, WALK</td>
</tr>
<tr>
<td>s3</td>
<td>s0</td>
<td>R, HALT</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next state</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>W=0 W=1</td>
<td>G</td>
</tr>
<tr>
<td>00</td>
<td>00 01</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>10 10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>11 11</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>00 00</td>
<td>0</td>
</tr>
</tbody>
</table>
Traffic Controller: Output Equations

\[ R = A \]
\[ Y = \overline{A}.B \]
\[ G = \overline{A}.B \]
\[ HALT = \overline{A} + B = \overline{WALK} \]
\[ WALK = A.B \]
Pedx Controller: D Flip-Flop Circuit Diagram
**Vending Machine Controller (One hot)**

**Problem Statement** Design a state machine controller for a vending machine to satisfy the following,

- Sell one item worth 75c
- Return change and deliver the item when the coins inserted into the machine exceed the sum of 75c
- Return all coins upon request without releasing the item
- Input variables $COIN$ indicating if a coin is deposited, $RETURN$ if the return change button is pushed and $SUM$ which is the output of the coin tallying device internal to the vending machine. $SUM$ consists of three inputs indicating whether the current coin entry is $<, =, >$ the price of the product.
Vending Machine Controller State Diagram

Diagram showing the states:

- **s0**: Initial state.
- **s1**: Sum less than 75.
- **s2**: Coin returned.
- **s3**: Sum greater than 75.
- **s4**: Return change.
- **s5**: Return all coins.

Transitions:
- From s0 to s2 on coin.return
- From s2 to s4 on coin.return
- From s4 to s5 on change_available
- From s3 to s5 on change_available

States:
- s0: Start state.
- s1: Sum less than 75.
- s2: Coin returned.
- s3: Sum greater than 75.
- s4: Return change.
- s5: Return all coins.
## Vending Machine Controller Next State Table

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
<th>release candy</th>
<th>return all coins</th>
<th>return change</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td>s0 if coin=0 and return=0</td>
<td>s1 if coin=1 and return = 0</td>
<td>s5 if coin = 0 and return = 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>s0 if sum&lt;75</td>
<td>s2 if sum=75</td>
<td>s3 if sum&gt;75</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>s0</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s3</td>
<td>s4 if change available = 1</td>
<td>s5 if change available = 0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s4</td>
<td>s2</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s5</td>
<td>s0</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>input</td>
<td>abbreviation</td>
<td>output</td>
<td>abbreviation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>---------------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>coin</td>
<td><em>CN</em></td>
<td>release-candy</td>
<td><em>RC</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>return</td>
<td><em>RTN</em></td>
<td>return-change</td>
<td><em>RCH</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum&lt;75</td>
<td><em>SM0</em></td>
<td>return-all-coins</td>
<td><em>RAC</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum=75</td>
<td><em>SM1</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum&gt;75</td>
<td><em>SM2</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>change-available</td>
<td><em>CA</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Present state</td>
<td>next state</td>
<td>next state</td>
<td>next state</td>
<td>release change</td>
<td>return all coins</td>
<td>return change</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>---------------------</td>
<td>----------------</td>
<td>------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>s0</td>
<td>s0 if coin=0 and return=0</td>
<td>s1 if coin=1 and return = 0</td>
<td>s5 if coin = 0 and return = 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s1</td>
<td>s0 if sum&lt;75</td>
<td>s2 if sum=75</td>
<td>s3 if sum&gt;75</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>s4 if change available = 1</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s3</td>
<td>-</td>
<td>s5 if change available = 0</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>s4</td>
<td>s2</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>s5</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABCDEF</td>
<td>000001 if CN = 0 and RTN = 0</td>
<td>000010 if CN = 1 and RTN = 0</td>
<td>100000 if CN = 0 and RTN = 1</td>
</tr>
<tr>
<td>000001</td>
<td>000001</td>
<td>000100</td>
<td>001000</td>
</tr>
<tr>
<td>000010</td>
<td>000001 if SM0 = 1</td>
<td>001000 if SM1 = 1</td>
<td>001000 if SM2 = 1</td>
</tr>
<tr>
<td>000100</td>
<td>000001</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>001000</td>
<td>010000 if CA = 1</td>
<td>100000 if CA = 0</td>
<td>-</td>
</tr>
<tr>
<td>010000</td>
<td>000100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>100000</td>
<td>000001</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Vending Machine Controller Equations

With D flip-flops with RESET, activation of the RESET signal will force the state machine into state $s_0 = 000001$.

\[
D_A = F \cdot \overline{CN} \cdot RTN + C \cdot \overline{CA}
\]
\[
D_B = C \cdot \overline{CA}
\]
\[
D_C = E \cdot SM2
\]
\[
D_D = E \cdot SM1 + B
\]
\[
D_E = F \cdot CN \cdot \overline{RTN}
\]
\[
D_F = F \cdot \overline{CN} \cdot \overline{RTN} + E \cdot \overline{SM0} + D + A
\]

Note simplification afforded by **one hot coding**.
Problem Statement: Design a FSM controller to interpret the scan codes of an AT keyboard.

➤ Consider only the numerical key pad

➤ Do not send any signals to the keyboard

➤ If a non-character key is pressed generate a NOKEY signal
Keyboards send SCAN CODES
AT Keyboard controller Verilog

- The keyboard sends a scan code representing a character when a key is pressed. E.g. \(1Ch\) is sent for the symbol \(A\). (note that this is not upper case \(A\), just the key \(A\))

- When the key is released the scan code \(F0\) is sent followed by the scan code again. e.g. \(1CF0C1\).

- Some keys are extended keys requiring two scan codes. These are prepended by \(EO\), followed by \(EOF0\) and resent with \(EO\) prepended. e.g. when the enter key (\(5A\)) is pressed, \((EO5A) EOFO(EO5A)\).

Note that the keyboard does not process key combinations like \(SHIFT^X\) or \(^C\). It just sends scan codes. When you press the Caps Lock for example, the keyboard will send the scan code for the cap locks. It is then up to your keyboard BIOS to send a code to the keyboard to turn on the Caps lock LED and to indicate to the computer that keys are now entering in uppercase.
Keyboard Controller State Diagram

- Si
- S0
- S1

- Valid S.C. / KeyID
- F0 / NOKEY
- E0 / NOKEY
- Valid extended S.C. / KeyID
SCAN CODES - (use \texttt{parameter})

```verbatim
parameter ZERO_SC = 8'h70;
parameter ONE_SC = 8'h69;
parameter TWO_SC = 8'h72;
parameter THREE_SC = 8'h7a;
parameter FOUR_SC = 8'h6b;
parameter FIVE_SC = 8'h73;
parameter SIX_SC = 8'h74;
parameter SEVEN_SC = 8'h6c;
parameter EIGHT_SC = 8'h75;
parameter NINE_SC = 8'h7d;
parameter ENTER_SC = 8'h5a;//extended
parameter CHS_SC = 8'h7d;//extended
parameter CLX_SC = 8'h7a;//extended
parameter CLR_SC = 8'h77;
parameter PLUS_SC = 8'h79;
parameter MINUS_SC = 8'h7b;
parameter TIMES_SC = 8'h7c;
parameter DIV_SC = 8'h4a;//extended
parameter DP_SC = 8'h71;
parameter E0_SC = 8'hE0;
parameter F0_SC = 8'hF0;
```

59 ENGN3213: Digital Systems and Microprocessors L#12-13
States and Outputs - (use parameter)

parameter si = 4'b0000;
parameter s0 = 4'b0001;
parameter s1 = 4'b0010;

parameter ZERO = 5'h00;
parameter ONE = 5'h01;
parameter TWO = 5'h02;
parameter THREE = 5'h03;
parameter FOUR = 5'h04;
parameter FIVE = 5'h05;
parameter SIX = 5'h06;
parameter SEVEN = 5'h07;
parameter EIGHT = 5'h08;
parameter NINE = 5'h09;
parameter ENTER = 5'h0A;
parameter CHS = 5'h1A;
parameter CLX = 5'h0B;
parameter CLR = 5'h1B;
parameter PLUS = 5'h0C;
parameter MINUS = 5'h1C;
parameter TIMES = 5'h0D;
parameter DIV = 5'h1D;
parameter DP = 5'h0E;
parameter NOKEY = 5'h1E;
always @(posedge sys_clk) begin
  state <= nextstate;
end
case(state)
  begin
    case(keyStroke)
      begin
        case(keyStroke)
          FO_SC:      nextstate = s0;
          F0_SC:      nextstate = s1;
          default:    nextstate = si;
        endcase
      end
    endcase
  end

begin
  case(keyStroke)
    begin
      case(keyStroke)
        ZERO_SC:     nextstate = si;
        ONE_SC:     nextstate = si;
        TWO_SC:     nextstate = si;
        THREE_SC:   nextstate = si;
        FOUR_SC:    nextstate = si;
        FIVE_SC:    nextstate = si;
        SIX_SC:     nextstate = si;
        SEVEN_SC:   nextstate = si;
        EIGHT_SC:   nextstate = si;
        NINE_SC:    nextstate = si;
        CLR_SC:     nextstate = si;
        PLUS_SC:    nextstate = si;
        MINUS_SC:   nextstate = si;
        TIMBS_SC:   nextstate = si;
        DP_SC:      nextstate = si;
        F0_SC:      nextstate = s0;
        E0_SC:      nextstate = si;
        default:    nextstate = si;
      endcase
    end
  endcase
end
end
always @(*) begin // output logic
    if(done & pc) begin
        case(state)
            /* */
            s1: /* */
                $;
            /* */
            s2: /* */
               begin
                   case(keyStroke)
                   zero_sc: keyID = zero;
                   one_sc: keyID = one;
                   two_sc: keyID = two;
                   three_sc: keyID = three;
                   four_sc: keyID = four;
                   five_sc: keyID = five;
                   six_sc: keyID = six;
                   seven_sc: keyID = seven;
                   eight_sc: keyID = eight;
                   nine_sc: keyID = nine;
                   clc_sc: keyID = clc;
                   plus_sc: keyID = plus;
                   minus_sc: keyID = minus;
                   times_sc: keyID = times;
                   dp_sc: keyID = dp;
                   fo_sc: keyID = nkey;
                   default: keyID = nkey;
               endcase
            end
          endcase
        end
    end
end

always @(*) begin // output logic
    if(done & pc) begin
        case(state)
            /* */
            s1: /* */
                $;
            /* */
            s2: /* */
               begin
                   case(keyStroke)
                   enter_sc: keyID = enter;
                   chs_sc: keyID = chs;
                   clc_sc: keyID = clc;
                   div_sc: keyID = div;
                   default: keyID = nkey;
               endcase
            end
          endcase
        end
    end
end