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.

Mealy machine  Moore machine
The control path of an RTL system is based on a finite state machine. An FSM is quite abstract, but for this discussion one can think of an FSM as a system that possesses a finite number of states. In each state the output of the system is fully deterministic. When a new set of inputs is applied, the system undergoes a transition to a different state with different outputs. The **bubbles** in the above figures represent the states and the arrow loops represent the transitions.

Notice that the inputs are written on the loops in the case of the Moore machine. Generally the outputs can be written inside the state bubbles of the Moore machine.

In the case of the Mealy machine both the inputs and the outputs are written on the loops. (WHY?)

**In this course we will only consider synchronous FSMs.** Synchronous FSMs are FSMs in which all transitions occur on the active edge of the clock. Moreover, even external inputs to the FSM, though naturally asynchronous because they cannot know about the clock, will be rendered synchronous through an appropriate circuit.

Apart from the above **state diagrams** one can also represent a FSM with a **state transition table** as follows...

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

State transition table
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 tick 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)
State Machine Electronic Block Diagram

- **Combinational**
  - Next State Logic
  - F(inputs, State)

- **Sequential**
  - D Q

- **Combinational**
  - Output Logic
  - G(state, input?)

**Inputs**

**Outputs**
State Machine Models: 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 Models: Mealy Machine

State-machine structure (Mealy)

inputs

Next-state Logic

F

excitation

State Memory

clock input

current state

Output Logic

G

output depends on state and input

typically edge-triggered D flip-flops

dock signal

outputs

ENGN3213: Digital Systems and Microprocessors L#12-13
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:**

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

\( x(t) \) - input, \( s(t) \) - state, \( z(t) \) - output
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.
  - S-R latch: \( Q^* = S + RQ \)
  - D flip-flop: \( Q^* = D \)
  - J-K flip-flop: \( Q^* = JQ + KQ \)
  - T flip-flop with enable: \( Q^* = TQ + TQ \)
State Machine Equations

- **Outputs Inputs**
- **Combinational**
- **Sequential**
- **Combinational**

- **Next State Logic**
  - $F(\text{inputs, State})$

- **Output Logic**
  - $G(\text{state, input?})$

- **Inputs**
- **Excitation Equation**
- **Characteristic Equation**
A Remark About the Flip-Flops

- In as much as most state machines nowadays are designed using PLDs, CPLDs, FPGAs or ASICs, positive edge triggered D flip-flops are most commonly used to design FSMs.
- This is what our VERILOG model describes.
- Sometimes however more efficient (less CMOS gate) designs can be made with negative edge triggered D flip-flops or J-K flip-flops.
Representations of Finite State Machines
State Diagrams

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.
Revision?: Pedestrian Traffic Controller (Moore machine)

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

Transition:
- W from s0
- W from s1
- RESET from s3
Another representation of the FSM is the **next state table** (also sometimes called **next state/output table**).

- The left-most column gives the state of the system.
- The next columns give the state to which a transition must occur for the stated set of **input** signals.
- The final column are the **outputs**.
Examples of FSMs: Alarm Circuit (Moore machine)
Examples of FSMs: Alarm Circuit State Diagram

- State Diagram:
  - S0: Armed
  - S1: Sound

- Transitions:
  - Tr: From S0 to S1
  - Rst: From S1 to S0
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>


Examples of FSMs: Coke Vending Machine

- **s0**: initial state
- **s1**: coin input
- **s2**: release candy
- **s3**: return change
- **s4**: return all coins
- **s5**: return all coins

Transitions:
- **coin.return** from s0 to s5
- **sum<75** from s1 to s2
- **sum=75** from s2 to s4
- **sum>75** from s3 to s2

Conditions:
- **coin.return**
- **sum<75**
- **sum=75**
- **change_available**
- **return_change**
Examples of FSMs: Candy Vending Machine

<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</td>
<td>s1 if coin=1</td>
<td>s5 if coin = 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>and return=0</td>
<td>and return = 0</td>
<td>and return = 1</td>
<td></td>
<td></td>
<td></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>s2</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>-</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>
</tbody>
</table>
FSM Analysis Steps

Starting from a FSM logic diagram:

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
State Machines: Example (Mealy Machine)
State Machines: Example: Step 1a) Excitation Equations

\[
D_0 = Q_0 \cdot EN' + Q_0' \cdot EN
\]
\[
D_1 = Q_1 \cdot EN' + Q_1' \cdot Q_0 \cdot EN + Q_1 \cdot Q_0' \cdot EN
\]
State Machines: Example: Step 1bc) Transition/output Equations

- Excitation equations
  \[ D_0 = Q_0 \cdot EN' + Q_0' \cdot EN \]
  \[ D_1 = Q_1 \cdot EN' + Q_1' \cdot Q_0 \cdot EN + Q_1 \cdot Q_0' \cdot EN \]

- Characteristic equations of D F/F
  \[ Q_0^* = D_0 \]
  \[ Q_1^* = D_1 \]

- Next state equations by substituting excitation equations into characteristic equations
  \[ Q_0^* = Q_0 \cdot EN' + Q_0' \cdot EN \]
  \[ Q_1^* = Q_1 \cdot EN' + Q_1' \cdot Q_0 \cdot EN + Q_1 \cdot Q_0' \cdot EN \]

- Output equation
  \[ MAX = Q_1 \cdot Q_0 \cdot EN \]
State Machines: Example: Step 2) State/output Tables

- From the results

\[ Q_0^* = Q_0 \cdot EN' + Q_0' \cdot EN \]
\[ Q_1^* = Q_1 \cdot EN' + Q_1' \cdot Q_0 \cdot EN + Q_1 \cdot Q_0' \cdot EN \]
\[ \text{MAX} = Q_1 \cdot Q_0 \cdot EN \]

<table>
<thead>
<tr>
<th>EN</th>
<th>S</th>
<th>S*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>A</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EN</th>
<th>S</th>
<th>S*</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

transition
table

state table

state/output
table
State Machines: Example: Step 3) State Diagram

- So, the system sequences through $A \rightarrow B \rightarrow C \rightarrow D \rightarrow A \ldots$ whenever $EN$ is 1 and holds its value with $EN=0$.
- This is an up-counter with an enable input. The output is 1 whenever it rolls over ($D \rightarrow A$).
State Machines: Example 2

Excitation Equation:

\[
\begin{align*}
T1 &= Y \\
T2 &= \overline{X} \cdot Y \cdot Q1
\end{align*}
\]
State Machines: Example 2 (CTD)

- Excitation Equations:
  \[ T1 = Y \]
  \[ T2 = \overline{X}.Y.Q1 \]

- Char. eq. for T-ff:
  \[ Q1^* = T1.\overline{Q1} + \overline{T1}.Q1 \]
  \[ Q2^* = T2.Q2 + \overline{T2}.Q2 \]

- Thus the transition eq:
  \[ Q1^* = Y.\overline{Q1} + \overline{Y}.Q1 \]
  \[ Q2^* = \overline{X}.Y.Q1.\overline{Q2} + \overline{X}.Y.Q1.Q2 \]
  \[ = \overline{X}.Y.Q1.\overline{Q2} + (X + \overline{Y} + \overline{Q1}).Q2 \]

<table>
<thead>
<tr>
<th>Q1</th>
<th>Q2</th>
<th>XY</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>10,1</td>
<td>11,0</td>
</tr>
<tr>
<td>10</td>
<td>10,1</td>
<td>01,1</td>
</tr>
<tr>
<td>11</td>
<td>11,0</td>
<td>00,0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>E(\bar{h})</th>
<th>Q1(\bar{h})</th>
<th>Q2(\bar{h})</th>
</tr>
</thead>
<tbody>
<tr>
<td>S, Z</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
State Machines: Example 2 (CTD)

- State diagram

<table>
<thead>
<tr>
<th>S</th>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

XY/Z = 00/1

A

B

C

D

00/1

01/1

01/0

00/0

10/0

11/0
The arc can be labelled with a transition expression: a transition is taken for inputs for which the expression is 1.

This is useful in Moore machine where output is not dependent on the inputs.
Verilog Description of an FSM

- Verilog can be used to describe higher levels of abstraction than circuits.
- 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 Wakerly Tables 7-58 and 7-59:
  - State transitions model with **sequential ALWAYS** block
  - Next state logic model with **combinational ALWAYS** block
  - Output logic model with **combinational ALWAYS** block
Design and Synthesis of Finite State Machines
Last Lecture

- We saw the definition of the Mealy and Moore state machines.
- We defined the next state (or output) table and the state diagram.
- We defined the following terms to describe state machines: excitation and characteristic equations, next state logic, state memory and output logic.
- We looked at a counter example where the circuit was given and we drew up the next state table and the state diagram.
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 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.

4. Draw an excitation table - a truth table showing the inputs and current state binary values as inputs and the desired next state binary values as the outputs.

5. Use K-maps to obtain produce minimal next state and output combinational logic.

6. Use the standard VERILOG formulation to simulate your design and check for correct operation. Revise as appropriate.

7. Check for potential practical problems (e.g. non-ideal effects).
Design Examples

1. The alarm controller
2. Up/Down/Stop Counter
3. Pedestrian Traffic Controller
4. Vending Machine Controller

.... Practice makes perfect!
Alarm Controller

Problem Statement:

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

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

Based on the state diagram, the state/output or next state table can be constructed.

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

<table>
<thead>
<tr>
<th>Present state</th>
<th>$T'R'$</th>
<th>$T'R$</th>
<th>$TR$</th>
<th>$TR'$</th>
</tr>
</thead>
<tbody>
<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>
## Alarm Controller: State/Output Table (CTD)

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

Consider the values within the next state table to be not only the ensuing 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 alarm controller and excitation logic](image-url)
## 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>0</td>
<td>1</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>
Do Karnaugh maps to work out the Boolean input logic.

$$Q = T \cdot \overline{R} + T \cdot \overline{S} + \overline{R} \cdot S$$  \hspace{1cm} (1)
Alarm Controller Circuit

\[ \text{Diagram of the alarm controller circuit with gates and a DFF.} \]

---

45 ENGN3213: Digital Systems and Microprocessors L#12-13
Alarm Controller: Comments

➤ Is the input $T \cdot \overline{R}$ really necessary?

➤ Are there any glitches produced according to the truth table?

➤ Use the techniques of lecture 11 to synchronise the level sensitive inputs $T$ and $R$ to the clock.
Up/Down/Stop Counter

1. Design a synchronous up/down/stop counter that
   ➤ has two input buttons (‘mode’ and ‘stop’).
   ➤ If the ‘mode’ button is 0, it counts upwards. If the ‘mode’ input is 1 it counts downwards.
   ➤ If ‘stop’ is pressed, it holds the current state regardless of input mode.
   ➤ Output indicates the even states (second and fourth).

2. For example, if the states are binary number then for every input, it will sequence through:

   up (M=0)
   0, 1, 2, 3, 4, 0, 1, 2, 1, 0, 4, 3, 2, 1, 1, 1, 1, etc

   Stop (S=1)

   down (M=1)
Up/Down/Stop Counter State Diagram

State Diagram

- State A: odd
  - Transition: 'up' to State B: even
  - Transition: 'down' to State D: even
  - Transition: 'stop'

- State B: even
  - Transition: 'up' to State A: odd
  - Transition: 'down' to State C: odd
  - Transition: 'stop'

- State C: odd
  - Transition: 'down' to State D: even
  - Transition: 'stop'

- State D: even
  - Transition: 'up' to State C: odd
  - Transition: 'stop'
Up/Down/Stop Counter: State/Output Table

Based on the state diagram, the state/output table can be constructed:

<table>
<thead>
<tr>
<th>Present state</th>
<th>next state</th>
<th>next state</th>
<th>next state</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B Up</td>
<td>D Down</td>
<td>A stop</td>
<td>Odd</td>
</tr>
<tr>
<td>B</td>
<td>C Up</td>
<td>A Down</td>
<td>B stop</td>
<td>Even</td>
</tr>
<tr>
<td>C</td>
<td>D Up</td>
<td>B Down</td>
<td>C stop</td>
<td>Odd</td>
</tr>
<tr>
<td>D</td>
<td>A Up</td>
<td>C Down</td>
<td>D stop</td>
<td>Even</td>
</tr>
</tbody>
</table>

Now we need to code the symbols (states/output) into binary numbers.
The methods of coding into numbers are not unique.
- Binary coding, Gray coding, etc.

### State/Output Table (CTD)

<table>
<thead>
<tr>
<th>State</th>
<th>Symbol</th>
<th>(Q_1Q_0)</th>
<th>(Q_1Q_0)</th>
<th>(Q_3Q_2Q_1Q_0)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
<td>00</td>
<td>0001</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>01</td>
<td>01</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>10</td>
<td>0100</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td>11</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

- Minimum number of flip-flops + glitch free
- Minimum number of flip-flops
- Output logic can be simple (also called One-hot method)
Up/Down/Stop Counter: State/Output Table (CTD)

- Lets use the Gray code
- For the up/down inputs, we can assign one bit M (mode) M=0 for the up input, and M=1 for the down input.
- For the stop input we still need one more bit (S=1 for stop).
- The output (y) will be 1 for the second and fourth states.

<table>
<thead>
<tr>
<th>state</th>
<th>$Q_1Q_0$</th>
<th>input</th>
<th>SM</th>
<th>output</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>00</td>
<td>up</td>
<td>00</td>
<td>odd</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>01</td>
<td>down</td>
<td>01</td>
<td>even</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>11</td>
<td>stop</td>
<td>1X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The next-state table becomes

Now we need to decide the flip-flop devices, and thus find out the inputs for the flip-flop.

<table>
<thead>
<tr>
<th>Present state</th>
<th>Next State (Q^{*1}Q^{*0})</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>up</td>
<td>down</td>
</tr>
<tr>
<td>Q_1Q_0</td>
<td>S=0,M=0</td>
<td>S=0,M=1</td>
</tr>
<tr>
<td>00</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>00</td>
<td>11</td>
</tr>
</tbody>
</table>
Up/Down/Stop Counter (CTD)

- From the next-state table, we need to find out the required flip-flop inputs (or excitation table).
- If we use the D flip-flop, the FF inputs are directly the next-state values, but for JK FF, it is a bit more complex.

<table>
<thead>
<tr>
<th>D</th>
<th>( Q_{t+} )</th>
<th>( Q_t \rightarrow Q_{t+} )</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0 → 0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1 → 1</td>
<td>1</td>
</tr>
</tbody>
</table>

D F/F

<table>
<thead>
<tr>
<th>J</th>
<th>K</th>
<th>( Q_{t+} )</th>
<th>( Q_t \rightarrow Q_{t+} )</th>
<th>J</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( Q_t )</td>
<td>0 → 0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0 → 1</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1 → 0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>( \overline{Q_t} )</td>
<td>1 → 1</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

JK F/F
Up/Down/Stop Counter: State/Output Table (CTD)
Up/Down/Stop Counter: (CTD)

<table>
<thead>
<tr>
<th>SM</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>01</td>
<td>11</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>11</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>00</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

A*B*

Transition table

Next states

\[ Q_t \rightarrow Q_{t+} \]

\[ D \]

| 0 \rightarrow 0 | 0 |
| 0 \rightarrow 1 | 1 |
| 1 \rightarrow 0 | 0 |
| 1 \rightarrow 1 | 1 |

D flip-flop table

Excitation table

Inputs to D flip-flops

ENGN3213 by Dr. Jac Kim
Up/Down/Stop Counter: (CTD)

\[ D_1 = \overline{S}M\overline{Q}_0 + \overline{S}MQ_0 + SQ_1 = \overline{S}(M \oplus Q_0) + SQ_1 \]

\[ D_0 = \overline{S}M\overline{Q}_1 + \overline{S}MQ_1 + SQ_0 = \overline{S}(M \oplus Q_1) + SQ_0 \]

\[ y = Q_1 \oplus Q_0 \]
Up/Down/Stop Counter: State/Output Table (CTD)

\[
\begin{align*}
D_1 &= \overline{S} (M \oplus Q_0) + SQ_1 \\
D_0 &= \overline{S} (M \oplus Q_1) + SQ_0
\end{align*}
\]
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/hold’ 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.
# 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 ( 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: State/Output 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>( W = 0 )</th>
<th>( 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>

Use Karnaugh maps...

<table>
<thead>
<tr>
<th>Q1,Q2</th>
<th>( W = 0 )</th>
<th>( W = 1 )</th>
<th>Q1,Q2</th>
<th>( W = 0 )</th>
<th>( W = 1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>0</td>
<td>00</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>1</td>
<td>01</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>10</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Thus the excitation logic becomes $Q_1 \times Q_2$ and $\overline{Q_2} \cdot (Q_1 + W)$. 
## 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 $w$</td>
<td>G, HALT</td>
</tr>
<tr>
<td></td>
<td>else s1 if $w$</td>
<td></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>

### State Transition 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>----------------</td>
<td>-------------</td>
<td>---------</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>
Traffic Controller: Output Equations

\[
\begin{align*}
R &= A \\
Y &= \overline{A} \cdot B \\
G &= \overline{A} \cdot \overline{B} \\
HALT &= \overline{A} + B = \overline{WALK} \\
WALK &= A \cdot \overline{B}
\end{align*}
\]
Pedx Controller: D Flip-Flop Circuit Diagram
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
Vending Machine Controller State Diagram

s0

- coin.return
- sum<75

s1

- coin.return
- sum>75

s2

- release_candy
- sum=75

s3

- change_available
- sum>75

s4

- return_change
- change_available

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>$CN$</td>
<td>release-candy</td>
<td>$RC$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>return</td>
<td>$RTN$</td>
<td>return-change</td>
<td>$RCH$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum&lt;75</td>
<td>$SM0$</td>
<td>return-all-coins</td>
<td>$RAC$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum=75</td>
<td>$SM1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sum&gt;75</td>
<td>$SM2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>change-available</td>
<td>$CA$</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 candy</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>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>
</tbody>
</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>ABCDEF</td>
<td>-------------</td>
<td>------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>000001</td>
<td>00001 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>
<td></td>
</tr>
<tr>
<td>000010</td>
<td>00001 if $SM0 = 1$</td>
<td>001000 if $SM1 = 1$</td>
<td>001000 if $SM2 = 1$</td>
<td></td>
</tr>
<tr>
<td>000100</td>
<td>000001</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>001000</td>
<td>010000 if $CA = 1$</td>
<td>100000 if $CA = 0$</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>010000</td>
<td>000100</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>100000</td>
<td>000001</td>
<td>-</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 \( s_0 = 000001 \).

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

Note simplification afforded by one hot coding.
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                     //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)                       //OUTPUT LOGIC
  case(state)
    S0:
      Output = f(state);
    S1:
      Output = g(state);
    default:
      Output = h(state);
  endcase
end
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
end
endmodule