First, We Will Complete
Sequential Logic Design
We Covered A Lot of Sequential Logic

- **Circuits that can store information**
  - Cross-coupled inverter
  - R-S Latch
  - Gated D Latch
  - D Flip-Flop
  - Register
  - Memory

- **Sequential logic circuits**
  - State & Clock
  - Asynchronous vs. Synchronous

- **Finite State Machines (FSM)**
  - How to design FSMs
Recall: Sequential Circuits

- Circuits that produce output depending on current and past input values – circuits with memory
Recall: Sequential Logic Circuits

**Combinational**
Only depends on current inputs

**Sequential**
Opens depending on past inputs

https://www.easykeys.com/228_ESP_Combination_Lock.aspx
https://www.fosmon.com/product/tsa-approved-lock-4-dial-combo
Recall: State Diagram of Our Sequential Lock

- Completely describes the operation of the sequential lock

Recall: Finite State Machines (FSMs) Consist of:

- **Five elements:**
  1. A **finite** number of **states**
     - *State*: snapshot of all relevant elements of the system at the time of the snapshot
  2. A **finite** number of external **inputs**
  3. A **finite** number of external **outputs**
  4. An explicit **specification of all state transitions**
     - How to get from one state to another
  5. An explicit **specification of what determines each external output value**

**FSM:** A discrete-time model of a stateful system
Recall: Finite State Machines (FSMs)

- Each FSM consists of three separate parts:
  - next state logic
  - state register
  - output logic

At the beginning of the clock cycle, next state is latched into the state register.
Recall: Finite State Machines (FSMs) Consist of:

- **Sequential Circuits**
  - State register(s)
    - Store the current state and
    - Provide the next state at the clock edge

- **Combinational Circuits**
  - Next state logic
    - Determines what the next state will be
  - Output logic
    - Generates the outputs
Recall: State Register Implementation

- How can we implement a **state register**? Two properties:
  1. We need to store data at the **beginning** of every clock cycle
  2. The data must be **available** during the **entire clock cycle**

![Diagram showing clock transitions and register input/output behavior](image-url)
Currently, we cannot simply wire a clock to WE of a latch. When the clock is high, Q will not take on D's value AND when the clock is low, the latch will propagate D to Q.

Recall: The Problem with Latches: Transparency

Recall the Gated D Latch

How can we change the latch, so that

1) D (input) is **observable** at Q (output) only at the **beginning of next** clock cycle?

2) Q is **available for the full clock cycle**
Recall: The D Flip-Flop

- 1) state change on clock edge, 2) data available for full cycle

- When the clock is low, 1\textsuperscript{st} latch propagates D to the input of the 2\textsuperscript{nd} (Q unchanged)
- Only when the clock is high, 2\textsuperscript{nd} latch latches D (Q stores D)
  - At the rising edge of clock (clock going from 0->1), Q gets assigned D
Recall: The D Flip-Flop

1) state change on clock edge, 2) data available for full cycle

- At the rising edge of clock (clock going from 0->1), \( Q \) gets assigned \( D \)
- At all other times, \( Q \) is unchanged
Recall: The D Flip-Flop

- 1) state change on clock edge, 2) data available for full cycle

At the rising edge of clock (clock going from 0->1), Q gets assigned D
- At all other times, Q is unchanged

We can use D Flip-Flops to implement the state register
Recall: Rising-Clock-Edge Triggered Flip-Flop

- **Two inputs**: CLK, D

- **Function**
  - The flip-flop “samples” D on the rising edge of CLK (positive edge)
  - When CLK rises from 0 to 1, D passes through to Q
  - Otherwise, Q holds its previous value
  - Q changes only on the rising edge of CLK

- A flip-flop is called an **edge-triggered state element** because it captures data on the clock edge
  - A latch is a **level-triggered** state element
D Flip-Flop Based Register

- Multiple parallel D flip-flops, each of which storing 1 bit

This register stores 4 bits

This line represents 4 wires

Condensed
A 4-Bit D-Flip-Flop-Based Register (Internally)

Finite State Machines (FSMs)

- Next state is determined by the current state and the inputs
- Two types of finite state machines differ in the **output logic**:  
  - **Moore FSM**: outputs depend only on the current state
Finite State Machines (FSMs)

- Next state is determined by the current state and the inputs
- Two types of finite state machines differ in the **output logic**:
  - **Moore FSM**: outputs depend only on the current state
  - **Mealy FSM**: outputs depend on the current state and the inputs
Finite State Machine Example

- “Smart” traffic light controller
  - **2 inputs:**
    - Traffic sensors: $T_A$, $T_B$ (TRUE when there’s traffic)
  - **2 outputs:**
    - Lights: $L_A$, $L_B$ (Red, Yellow, Green)
  - State can change every 5 seconds
    - Except if green and traffic, stay green

From H&H Section 3.4.1
Finite State Machine Black Box

- **Inputs:** CLK, Reset, T_A, T_B
- **Outputs:** L_A, L_B

![Traffic Light Controller Diagram]

CLK

T_A

T_B

Traffic Light Controller

L_A

L_B

Reset
Moore FSM: outputs labeled in each state
- States: Circles
- Transitions: Arcs
**Finite State Machine Transition Diagram**

- **Moore FSM:** outputs labeled in each state
  - **States:** Circles
  - **Transitions:** Arcs

In the diagram:
- **States:** S0, S1
- **Transitions:** TA
- **Signals:** LA: green, LB: red
  - From S0 to S1: LA: yellow, LB: red
- **Locations:**
  - Academic Ave.
  - Bravado Blvd.
  - Labs
  - Dorms
  - Fields

**Reset** arrow indicates the start of the sequence.
Finite State Machine Transition Diagram

- **Moore FSM**: outputs labeled in each state
  - **States**: Circles
  - **Transitions**: Arcs
Finite State Machine Transition Diagram

- **Moore FSM**: outputs labeled in each state
  - **States**: Circles
  - **Transitions**: Arcs

---

**Diagram Details**:

- **States**: Circles labeled with states and transitions.
- **Transitions**: Arrows between states indicating transitions.
- **Labels**: Each state is labeled with specific transitions and outputs.
- **Output Colors**: States S0, S1, S2, S3 are labeled with output colors:
  - S0: $L_A$: green, $L_B$: red
  - S1: $L_A$: yellow, $L_B$: red
  - S2: $L_A$: red, $L_B$: green
  - S3: $L_A$: red, $L_B$: yellow
- **Reset**: An arrow labeled 'Reset' points back to state S0.

---

**Location Labels**:

- **Academic Ave.**
- **Bravado Blvd.**
- **Dining Hall**
- **Fields**
- **Labs**
- **Dorms**
Finite State Machine Transition Diagram

- **Moore FSM:** outputs labeled in each state
  - **States:** Circles
  - **Transitions:** Arcs
Finite State Machine:
State Transition Table
### FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- **S0**: $L_A$: green, $L_B$: red
- **S1**: $L_A$: yellow, $L_B$: red
- **S2**: $L_A$: red, $L_B$: green
- **S3**: $L_A$: red, $L_B$: yellow

**Transitions:**
- $T_A$: Reset
- $T_B$: Input A

**LEDs:**
- **LA**: green
- **LB**: red
- **LA**: yellow
- **LB**: red
- **LA**: red
- **LB**: yellow
- **LA**: red
- **LB**: green
FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>$T_A$</td>
<td>$T_B$</td>
</tr>
<tr>
<td>S0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
### FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>
FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>0</td>
<td>$S_0'$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>0</td>
<td>$S_0'$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>1</td>
<td>$S_1$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>1</td>
<td>$S_0'$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$S_1$</th>
<th>$S_0$</th>
<th>$T_A$</th>
<th>$T_B$</th>
<th>$S_1'$</th>
<th>$S_0'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

State Encoding

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>
**FSM State Transition Table**

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$T_A$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

**State Encoding**

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>

$S'_1 = \_?
### FSM State Transition Table

#### Current State | Inputs | Next State
---|---|---
$S_1$ | $S_0$ | $T_A$ | $T_B$ | $S'_1$ | $S'_0$
0 | 0 | 0 | X | 0 | 1
0 | 0 | 1 | X | 0 | 0
0 | 1 | X | X | 1 | 0
1 | 0 | X | 0 | 1 | 1
1 | 0 | X | 1 | 1 | 0
1 | 1 | X | X | 0 | 0

### State Encoding

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>

### State Equation

$$S'_1 = (\bar{S}_1 \cdot S_0) + (S_1 \cdot \bar{S}_0 \cdot \bar{T}_B) + (S_1 \cdot \bar{S}_0 \cdot T_B)$$
FSM State Transition Table

Current State | Inputs | Next State
---|---|---
$S_1$ | $S_0$ | $T_A$ | $T_B$ | $S'_1$ | $S'_0$
0 | 0 | 0 | 1 | 0 | 1
0 | 0 | 1 | 1 | 0 | 0
0 | 1 | 1 | 1 | 1 | 0
1 | 0 | 0 | 0 | 1 | 1
1 | 0 | 1 | 1 | 1 | 0
1 | 1 | 0 | 0 | 0 | 0

State Encoding

- S0: 00
- S1: 01
- S2: 10
- S3: 11

\[ S'_1 = (\overline{S}_1 \cdot S_0) + (S_1 \cdot \overline{S}_0 \cdot \overline{T}_B) + (S_1 \cdot \overline{S}_0 \cdot T_B) \]

\[ S'_0 = ? \]
FSM State Transition Table

\[
S'_{1} = (\overline{S_{1}} \cdot S_{0}) + (S_{1} \cdot \overline{S_{0}} \cdot \overline{T_{B}}) + (S_{1} \cdot S_{0} \cdot T_{B})
\]

\[
S'_{0} = (\overline{S_{1}} \cdot \overline{S_{0}} \cdot \overline{T_{A}}) + (S_{1} \cdot \overline{S_{0}} \cdot \overline{T_{B}})
\]

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S(_1)</td>
<td>S(_0)</td>
<td>T(_A)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>
### FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$T_A$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

**Simplified**

$S'_1 = S_1 \ XOR S_0$

$S'_0 = (S_1 \cdot S_0 \cdot \overline{T_A}) + (S_1 \cdot S_0 \cdot \overline{T_B})$

**State Encoding**

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>
Finite State Machine:
Output Table
FSM Output Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( S_0 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Reset

\[ S_0 \]
- \( L_A \): green
- \( L_B \): red

\[ S_1 \]
- \( L_A \): yellow
- \( L_B \): red

\[ S_2 \]
- \( L_A \): red
- \( L_B \): green

\[ S_3 \]
- \( L_A \): red
- \( L_B \): yellow
FSM Output Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>00</td>
</tr>
<tr>
<td>yellow</td>
<td>01</td>
</tr>
<tr>
<td>red</td>
<td>10</td>
</tr>
</tbody>
</table>
FSM Output Table

Current State | Outputs
--- | ---
| $S_1$ | $S_0$ | $L_{A1}$ | $L_{A0}$ | $L_{B1}$ | $L_{B0}$
| 0 | 0 | 0 | 0 | 1 | 0
| 0 | 1 | 0 | 1 | 1 | 0
| 1 | 0 | 1 | 0 | 0 | 0
| 1 | 1 | 1 | 0 | 0 | 1

Output Encoding

<table>
<thead>
<tr>
<th>Output</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>00</td>
</tr>
<tr>
<td>yellow</td>
<td>01</td>
</tr>
<tr>
<td>red</td>
<td>10</td>
</tr>
</tbody>
</table>

$L_{A1} = S_1$
FSM Output Table

Current State | Outputs
---|---
| | $L_{A1}$ | $L_{A0}$ | $L_{B1}$ | $L_{B0}$
0 | 0 | 0 | 0 | 1 | 0
0 | 1 | 0 | 1 | 1 | 0
1 | 0 | 1 | 0 | 0 | 0
1 | 1 | 1 | 0 | 0 | 1

Output | Encoding
---|---
green | 00
yellow | 01
red | 10
FSM Output Table

Current State | Outputs
---|---
\(S_1\) | \(S_0\) | \(L_{A1}\) | \(L_{A0}\) | \(L_{B1}\) | \(L_{B0}\)
0 | 0 | 0 | 0 | 1 | 0
0 | 1 | 0 | 1 | 1 | 0
1 | 0 | 1 | 0 | 0 | 0
1 | 1 | 1 | 0 | 0 | 1

\[L_{A1} = S_1\]
\[L_{A0} = \overline{S_1} \cdot S_0\]
\[L_{B1} = \overline{S_1}\]

Output | Encoding
---|---
green | 00
yellow | 01
red | 10
## FSM Output Table

### Current State

<table>
<thead>
<tr>
<th></th>
<th>S₁</th>
<th>S₀</th>
<th>Lₐ₁</th>
<th>Lₐ₀</th>
<th>L₉₁</th>
<th>L₉₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### Outputs

<table>
<thead>
<tr>
<th></th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>00</td>
</tr>
<tr>
<td>yellow</td>
<td>01</td>
</tr>
<tr>
<td>red</td>
<td>10</td>
</tr>
</tbody>
</table>

\[
L_{A1} = S_1 \\
L_{A0} = S_1 \cdot S_0 \\
L_{B1} = \overline{S_1} \\
L_{B0} = S_1 \cdot S_0
\]
Finite State Machine: Schematic
FSM Schematic: State Register
FSM Schematic: State Register

CLK

S'_1
S'_0

S_1
S_0

r

Reset
state register
FSM Schematic: Next State Logic

\[ S'_1 = S_1 \text{xor} S_0 \]

\[ S'_0 = (\overline{S}_1 \cdot \overline{S}_0 \cdot \overline{T}_A) + (S_1 \cdot \overline{S}_0 \cdot \overline{T}_B) \]
FSM Schematic: Output Logic

\[
\begin{align*}
L_{A1} &= S_1 \\
L_{A0} &= \overline{S_1} \cdot S_0 \\
L_{B1} &= \overline{S_1} \\
L_{B0} &= S_1 \cdot S_0
\end{align*}
\]
FSM Timing Diagram

S0
L_A: yellow
L_B: red

S1
L_A: yellow
L_B: red

S2
L_A: red
L_B: green

S3
L_A: red
L_B: yellow

Cycle 1
CLK
Reset
T_A
T_B
S'_{1:0}
S_{1:0}
L_{A1:0}
L_{B1:0}

T_A
T_B

0 5

Green (00)
Red (10)
Yellow (01)
FSM Timing Diagram

- Cycle 1: S0 (00), L_A: yellow, L_B: red
- Cycle 2: S0 (00), L_A: yellow, L_B: red
- Cycle 3: S1 (10), L_A: red, L_B: green
- Cycle 4: S2 (01), L_A: red, L_B: yellow
- Cycle 5: S3 (11), L_A: red, L_B: yellow

- CLK, Reset, T_A, T_B

- S’_1:0, S_1:0, L_A1:0, L_B1:0

- Time: 0, 5, 10 seconds
FSM Timing Diagram

CLK
Reset
T_A
T_B
S'_1:0
S_1:0
L_{A1:0}
L_{B1:0}

Cycle 1  Cycle 2  Cycle 3  Cycle 4

S_0
L_A: yellow
L_B: red

S_1
L_A: yellow
L_B: red

S_2
L_A: red
L_B: green

S_3
L_A: red
L_B: yellow

0  5  10  15
FSM Timing Diagram

CLK
Reset
TA
TB
S'1:0
S1:0
LA1:0
LB1:0
Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5

CLK
Reset
TA
TB
S'1:0
S1:0
LA1:0
LB1:0

S0
L_A: yellow
L_B: red

S1
L_A: yellow
L_B: red

S2
L_A: red
L_B: green

S3
L_A: red
L_B: yellow

Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5

0 5 10 15 20

Green (00)
Red (10)
Yellow (01)
FSM Timing Diagram
FSM Timing Diagram
This is from H&H Section 3.4.1
FSM Timing Diagram

See H&H Chapter 3.4
Finite State Machine:
State Encoding
How do we encode the state bits?

- Three common state binary encodings with different tradeoffs
  1. Fully Encoded
  2. 1-Hot Encoded
  3. Output Encoded

Let’s see an example Swiss traffic light with 4 states

- Green, Yellow, Red, Yellow+Red
1. Binary Encoding (Full Encoding):
   - Use the minimum possible number of bits
     - Use $\log_2(\text{num\_states})$ bits to represent the states
   - Example state encodings: 00, 01, 10, 11
   - Minimizes # flip-flops, but not necessarily output logic or next state logic

2. One-Hot Encoding:
   - Each bit encodes a different state
     - Uses $\text{num\_states}$ bits to represent the states
     - Exactly 1 bit is “hot” for a given state
   - Example state encodings: 0001, 0010, 0100, 1000
   - Simplest design process – very automatable
   - Maximizes # flip-flops, minimizes next state logic
3. **Output Encoding:**

- Outputs are **directly accessible** in the state encoding.

- For example, since we have **3 outputs** (light color), encode state with **3 bits**, where each bit represents a color.

  - **Example states:** 001, 010, 100, 110
    - Bit\(_0\) encodes **green** light output,
    - Bit\(_1\) encodes **yellow** light output
    - Bit\(_2\) encodes **red** light output

- **Minimizes** output logic.

- Only works for Moore Machines (output function of state).
3. Output Encoding:
   - Outputs are directly accessible in the state encoding
   - For example, since we have 3 outputs (light color), encode state with 3 bits, where each bit represents a color
   - Example states: 001, 010, 100, 110
     - Bit 0 encodes green light output
     - Bit 1 encodes yellow light output
     - Bit 2 encodes red light output
   - Minimizes output logic
   - Only works for Moore Machines (output depends only on state)

The designer must carefully choose an encoding scheme to optimize the design under given constraints

- Minimizes output logic
- Only works for Moore Machines (output depends only on state)
Moore vs. Mealy Machines
Recall: Moore vs. Mealy FSMs

- Next state is determined by the current state and the inputs
- Two types of FSMs differ in the **output logic**:
  - **Moore FSM**: outputs depend only on the current state
  - **Mealy FSM**: outputs depend on the current state and the inputs
Moore vs. Mealy FSM Examples

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1’s and 0’s on it.
- The snail smiles whenever the last four digits it has crawled over are 1101.
- Design Moore and Mealy FSMs of the snail’s brain.

Moore FSM

```
inputs M next state logic k next state CLK k state output logic N outputs
```
Moore vs. Mealy FSM Examples

- Alyssa P. Hacker has a snail that crawls down a paper tape with 1’s and 0’s on it.
- The snail smiles whenever the last four digits it has crawled over are 1101.
- Design Moore and Mealy FSMs of the snail’s brain.
State Transition Diagrams

Moore FSM

Mealy FSM

What are the tradeoffs?
FSM Design Procedure

- **Determine** all possible states of your machine

- **Develop** a *state transition diagram*
  - Generally this is done from a textual description
  - You need to 1) determine the **inputs** and **outputs** for each **state** and 2) figure out how to get from one state to another

- **Approach**
  - Start by defining the **reset state** and what happens from it – this is typically an easy point to start from
  - Then continue to add **transitions** and **states**
  - Picking **good state names** is very important
  - Building an FSM is **like** programming (but it *is not* programming!)
    - An FSM has a sequential “control-flow” like a program with conditionals and goto’s
    - The if-then-else construct is controlled by one or more inputs
    - The outputs are controlled by the state or the inputs
  - In hardware, we typically have many concurrent FSMs

**FSM: A discrete-time model** of a stateful system
What is to Come: LC-3 Processor

Figure 4.3 The LC-3 as an example of the von Neumann model
What is to Come: LC-3 Datapath
Backup Slides:
Different Flip-Flop Types
Enabled Flip-Flops

- **Inputs**: CLK, D, EN
  - The enable input (EN) controls when new data (D) is stored
- **Function**:
  - **EN = 1**: D passes through to Q on the clock edge
  - **EN = 0**: the flip-flop retains its previous state
Resettable Flip-Flop

- **Inputs:** CLK, D, Reset
  - The Reset is used to set the output to 0.

- **Function:**
  - \textit{Reset} = \textbf{1}: Q is forced to 0
  - \textit{Reset} = \textbf{0}: the flip-flop behaves like an ordinary D flip-flop

Symbols

\[ D \quad Q \quad \text{Reset} \]

\[ r \]
Resettable Flip-Flops

- Two types:
  - **Synchronous**: resets at the clock edge only
  - **Asynchronous**: resets immediately when Reset = 1

- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop (see Exercise 3.10)

- Synchronously resettable flip-flop?
Settable Flip-Flop

- **Inputs:** CLK, D, Set
- **Function:**
  - Set = 1: Q is set to 1
  - Set = 0: the flip-flop behaves like an ordinary D flip-flop