L20 Finite State Machine

We have learned how to build a memory unit and how to use a clock to control the memory unit. Today, we will see how Mem + Clock can help build a sequential logic.

1. Robot Design Example:

A Robot to move on \( A \rightarrow B \). When it reaches the end, it reverse the direction and move backward, until we stop it.

Questions to think about:

How does the robot know whether to move forward or backward?

The robot needs to know its "current state", that is whether it is moving forward or backward. Thus, the next action depends on both "current state" and "inputs". How to solve this type of problem?

2. Finite State Machine (FSM)

\[ \text{Inputs} \rightarrow \begin{cases} \text{Forward} \\ \text{Backward} \end{cases} \rightarrow \text{Outputs} \]

This is called "Finite State Machine"
1) Elements of FSM:
   - Input
   - Current state
   - Next state
   - Output

2) Daily Example:
   - Input: Temperature of the room
   - State: Door opened/Door closed
   - Output: Actions (open the door, close the door)

3. How to use FSM?
   - Back to our Robot example.

   Step 1: Understand the function we want to achieve.
   1) Moving along the line
   2) Reverse when hits the end
   3) Stop when the stop button is hit

   Step 2: Define FSM.
   - Input:
     - Line sensor (L): 0: no line, 1: line
     - Button (B): 0: not pressed, 1: pressed
   - States:
     - $S_0$: Forward
     - $S_1$: Backward
     - $S_2$: Stop
Step 3: Represent the function by "state table".

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs L B</th>
<th>Next State</th>
<th>Outputs M S</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>0 0</td>
<td>$S_2$</td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>$S_1$</td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
<tr>
<td>$S_2$</td>
<td>0 0</td>
<td>$S_1$</td>
<td>0 0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>$S_2$</td>
<td>1 0</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
<tr>
<td>$S_3$</td>
<td>0 0</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
<tr>
<td></td>
<td>1 0</td>
<td>$S_1$</td>
<td>x 1</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
<td>$S_3$</td>
<td>x 1</td>
</tr>
</tbody>
</table>

This can be represented more clearly by the "state transition diagram" (STG).

**STG Conventions:**
- $\circ$ : State
- input / output : edges represent transition with associated inputs / outputs.
Then, how can we implement the transitions?

If we regard the "state table" as a "truth table," it builds the connection between "Current State" and "Next State.""Inputs" and "outputs." 

```
<table>
<thead>
<tr>
<th>Current States</th>
<th>Combinational Logic</th>
<th>Next States</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inputs</td>
<td></td>
<td>outputs</td>
</tr>
</tbody>
</table>
```

How many bits are required for representing states? 3 states ⇒ 2 bits.

Current State (CS₀, CS₁) ⇒ Next State (NS₀, NS₁)

Step 4: Determine the combinational logic.

Function ⇒ Truth Table ⇒ K-Map ⇒ SOP ⇒ Circuit.
<table>
<thead>
<tr>
<th>Current State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_0$</td>
<td>$00$</td>
<td>$00$</td>
<td>$01$</td>
</tr>
<tr>
<td>$S_1$</td>
<td>$01$</td>
<td>$00$</td>
<td>$10$</td>
</tr>
<tr>
<td>$S_2$</td>
<td>$01$</td>
<td>$01$</td>
<td>$10$</td>
</tr>
<tr>
<td>$S_3$</td>
<td>$10$</td>
<td>$00$</td>
<td>$10$</td>
</tr>
<tr>
<td>$S_4$</td>
<td>$10$</td>
<td>$01$</td>
<td>$10$</td>
</tr>
<tr>
<td>$S_5$</td>
<td>$11$</td>
<td>$10$</td>
<td>$10$</td>
</tr>
<tr>
<td>$S_6$</td>
<td>$11$</td>
<td>$11$</td>
<td>$10$</td>
</tr>
</tbody>
</table>

Truth Table: [from STG]

Step 4.1

```

```
Step 4.2: Truth Table $\Rightarrow$ k-Map.

We have 4 inputs and 4 outputs for the concerned combinational logic. So, we need the k-Map for 4 outputs. $N_{S_0}, N_{S_1}, M, S$ as on slide 12.

Step 4.3: k-Map $\Rightarrow$ SOP

Step 5: Put memory and the combinational logic together to get the sequential logic.

Note that $(C_{S_0}, C_{S_1})$ and $(N_{S_0}, N_{S_1})$ are within the sequential logic.
Remarks for the Project:

1. How many states do we need?

2. Any difference between the above example and the project?
   1) Is there transition from $S_1$ to $S_3$?
   2) Are the inputs that trigger $S_1 \Rightarrow S_2$ the same as those that trigger $S_2 \Rightarrow S_3$?

3. Can we simplify our design?
   1) From 2.1), can we simplify the use of the D-FF?
   2) From 2.2), can we simplify the clock?