Controller Design

- Five step controller design process

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Capture the FSM</td>
</tr>
<tr>
<td>2.</td>
<td>Create the architecture</td>
</tr>
<tr>
<td>3.</td>
<td>Encode the states</td>
</tr>
<tr>
<td>4.</td>
<td>Create state table</td>
</tr>
<tr>
<td>5.</td>
<td>Implement combinational logic</td>
</tr>
</tbody>
</table>

Controller Design: Laser Timer Example

- Step 1: Capture the FSM
  - Already done
- Step 2: Create architecture
  - 2-bit state register (for 4 states)
  - Input b, output x
  - Next state signals n1, n0
- Step 3: Encode the states
  - Any encoding with each state unique will work

Inputs: b; Outputs: x

Controller Design: Laser Timer Example (cont)

- Step 4: Create state table

<table>
<thead>
<tr>
<th>States</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>0 0</td>
<td>0 0</td>
</tr>
<tr>
<td>On1</td>
<td>0 1</td>
<td>0 1</td>
</tr>
<tr>
<td>On2</td>
<td>1 0</td>
<td>1 0</td>
</tr>
<tr>
<td>On3</td>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>

Controller Design: Laser Timer Example (cont)

- Step 5: Implement combinational logic

\[
\begin{align*}
    n1 &= s1's0b' + s1's0b + s1s0'b' + s1s0'b \\
    n1 &= s1's0 + s1s0' \\
    n0 &= s1's0'b + s1s0'b' + s1s0'b \\
    n0 &= s1's0'b + s1s0'b' + s1s0'b
\end{align*}
\]
Controller Design: Laser Timer Example (cont)

- Step 5: Implement combinational logic (cont)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>

- Want simple sequential circuit that converts button press to single cycle duration, regardless of length of time that button actually pressed.
  - We assumed such an ideal button press signal in earlier example, like the button in the laser timer controller.

Controller Example: Button Press Synchronizer

- Want generate sequence 0001, 0011, 1100, 1000, (repeat)
  - Each value for one clock cycle
  - Common, e.g., to create pattern in 4 lights, or control magnets of a "stepper motor"

- Step 2: Create architecture

- Step 3: Encode states

- Step 5: Create combinational circuit

Controller Example: Sequence Generator

- From earlier example

Controller Example: Secure Car Key

- Step 1: FSM
- Step 2: Create architecture
- Step 3: Encode states
- Step 4: State table
- Step 5: Create combinational circuit
Common Pitfalls Regarding Transition Properties

- Only one condition should be true
  - For all transitions leaving a state
  - Else, which one?
- One condition must be true
  - For all transitions leaving a state
  - Else, where go?

Verifying Correct Transition Properties

- Can verify using Boolean algebra
  - Only one condition true: AND of each condition pair (for transitions leaving a state) should equal 0
    \[ a \land \overline{a} = 0 \]
    \[ \overline{a} \land a = 0 \]
  - One condition true: OR of all conditions of transitions leaving a state) should equal 1
    \[ a \lor \overline{a} = 1 \]
    \[ \overline{a} \lor a = 1 \]

Example

\[ a \land \overline{a} = 0 \]
\[ \overline{a} \land a = 0 \]
\[ a \lor \overline{a} = 1 \]
\[ \overline{a} \lor a = 1 \]

Q: For shown transitions, prove whether:
- Only one condition true (AND of each pair is always 0)
- One condition true (OR of all transitions is always 1)

Evidence that Pitfall is Common

- Recall code detector FSM
  - We “fixed” a problem with the transition conditions
  - Do the transitions obey the two required transition properties?
    - Consider transitions of state Start, and the “only one true” property
      \[ a \land \overline{a} = 0 \]
      \[ \overline{a} \land a = 0 \]
      \[ a \lor \overline{a} = 1 \]
      \[ \overline{a} \lor a = 1 \]

More on Flip-Flops and Controllers

- Other flip-flop types
  - SR flip-flop: like SR latch, but edge triggered
  - JK flip-flop: like SR (S+J, R+K)
    \[ T=0: S=J, R=K \]
  - T flip-flop: JK with inputs tied together
    \[ \text{Toggles on every rising clock edge} \]
- Previously utilized to minimize logic outside flip-flop:
  - Today, minimizing logic to such extent is not as important
  - D flip-flops are thus by far the most common

Non-Ideal Flip-Flop Behavior

- Can’t change flip-flop input too close to clock edge
  - Setup time: time that D must be stable before edge
    \[ \text{Setup time violation} \]
  - Hold time: time that D must be held stable after edge
    \[ \text{Hold time} \]

Setup time violation

Setup time violation leads to oscillation!
**Metastability**

- Violating setup/hold time can lead to bad situation known as metastable state
  - Metastable state: Any flip-flop state other than stable 1 or 0
  - Eventually settles to one or other, but we don’t know which
  - For internal circuits, we can make sure to observe setup time
  - But what if input comes from external (asynchronous) source, e.g., button press?
- Partial solution
  - Insert synchronizer flip-flop for asynchronous input
  - Special flip-flop with very small setup/hold time
  - Doesn’t completely prevent metastability

**Flip-Flop Set and Reset Inputs**

- Some flip-flops have additional inputs
  - Synchronous reset: clears Q to 0 on next clock edge
  - Synchronous set: sets Q to 1 on next clock edge
  - Asynchronous reset: clear Q to 0 immediately (not dependent on clock edge)
  - Example timing diagram shown
  - Asynchronous set: set Q to 1 immediately

**Active Low Inputs**

- We’ve assumed input action occur when input is 1
  - Some inputs are instead active when input is 0 -- “active low”
  - Shown with inversion bubble
  - So to reset the shown flip-flop, set R=0. Else, keep R=1.

**Initial State of a Controller**

- All our FSMs had initial state
  - But our sequential circuit designs did not
  - Can accomplish using flip-flops with reset/set inputs
  - Shown circuit initializes flip-flops to 01 during power up of circuit
    - By electronic circuit design

**Design Challenge**

- Design Challenge
  - Draw a state diagram for a finite state machine that has an input x and an output y. Whenever x changes from 0 to 1, y should be 1 for four clock cycles and then return to 0 (even if x is still 1). Using the five-step sequential logic design process, implement the controller using a state register and logic gates.

Due Next Lecture (as announced in class)