Overview

- FSM model to capture sequential behavior
  - FSM: describe timing behavior of sequential circuit.
  - States, and transitions among states.
- FSM as a Controller
  - Controller-Datapath architecture.
  - Controller design process.
  - Convert FSM to a circuit.
- Convert a circuit to FSM.
- Miscellaneous
  - Common mistakes when capturing FSMs.
  - Metastability
  - Glitching

Finite-State Machines (FSMs) and Controllers

- Want sequential circuit with particular behavior over time
- Example: Laser timer
  - Pushing button causes x=1 for exactly 3 clock cycles
  - Precisely-timed laser pulse
  - b=1 gets stored in first D flip-flop
  - Then 2nd flip-flop on next cycle, then 3rd flip-flop on next cycle
  - OR the three flip-flop outputs, so x should be 1 for three cycles

Bad job — what if button pressed a second time during those 3 cycles?

Need a Better Way to Design Sequential Circuits

- Also bad because of ad hoc design process
  - How create other sequential circuits?
- Need
  - A way to capture desired sequential behavior
  - A way to convert such behavior to a sequential circuit

Step 1: Capture behavior
- Create a truth table or equations, whichever is most natural for the given problem, to describe the desired behavior of each output of the combinational logic.

2A: Create equations
- This substep is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.

2B: Implement as a gate-based circuit
- For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

Step 2: Convert to circuit
- Like we had for designing combinational circuits

Capturing Sequential Circuit Behavior as FSM

- Finite-State Machine (FSM)
  - Describes desired behavior of sequential circuit
  - Akin to Boolean equations for combinational behavior
- List states, and transitions among states
  - Example: Toggle x every clock cycle
  - Two states: "Lo" (x=0), and "Hi" (x=1)
  - Transition from Lo to Hi, or Hi to Lo, on rising clock edge (clk)
  - Arrow points to initial state (when circuit first starts)

FSM Example: Three Cycles High System

- Want 0, 1, 1, 0, 1, 1, ...
  - For one clock cycle each
- Capture as FSM
  - Four states: 0, first 1, second 1, third 1
  - Transition on rising clock edge to next state
Three-Cycles High System with Button Input

- Four states
- Wait in “Off” while b is 0 (b*clk^)
- When b is 1 (b*clk^), transition to On1
  - Sets x=1
  - Next two clock edges, transition to On2, then On3
- So x=1 for three cycles after button pressed

FSM Simplification: Rising Clock Edges Implicit

- Every edge ANDed with rising clock edge
- What if we wanted a transition without a rising edge
- We don’t consider such asynchronous FSMs – less common, and advanced topic
- Only consider synchronous FSMs – rising edge on every transition

FSM Definition

- FSM consists of
  - Set of states
  - Ex: {Off, On1, On2, On3}
  - Set of inputs, set of outputs
  - Ex: Inputs: {b}, Outputs: {x}
  - Initial state
  - Ex: “Off”
  - Set of transitions
  - Each with condition
  - Describes next states
  - Ex: Has 5 transitions
  - Set of actions
  - Sets outputs in each state
  - Ex: x=0, x=1, x=1, and x=1

FSM Example: Secure Car Key

- Many new car keys include tiny computer chip
  - When key turned, car’s computer (under engine hood) requests identifier from key
  - Key transmits identifier
  - Else, computer doesn’t start car
- FSM
  - Wait until computer requests ID (a=1)
  - Transmit ID (in this case, 1 1 0 1)

FSM Example: Secure Car Key (cont.)

- Nice feature of FSM
  - Can evaluate output behavior for different input sequences
  - Timing diagrams show states and output values for different input waveforms
- Ex: Earlier Flight-Attendant Call Button
  - Previously built using SR latch, then D flip-flop
  - Capture desired bit storage behavior using FSM instead
    - Clear and precise description of desired behavior
    - We’ll later convert to a circuit
How To Capture Desired Behavior as FSM

- **List states**
  - Give meaningful names, show initial state
  - Optionally add some transitions if they help
- **Create transitions**
  - For each state, define all possible transitions leaving that state.
- **Refine the FSM**
  - Execute the FSM mentally and make any needed improvements.

FSM Capture Example: Code Detector

- **Capture as FSM**
  - **List states**
  - Some transitions included
  - Start, Red, Green, Blue
  - Inputs: s, r, g, b
  - Outputs: u, a

Controller Design

- Converting FSM to sequential circuit
  - Circuit called controller
  - Standard controller architecture
    - State register stores encoding of current state
  - Combinational logic computes outputs and next state from inputs and current state
  - Rising clock edge takes controller to next state

FSM Capture Example: Code Detector

- **Capture as FSM**
  - **Create transitions**
  - Repeat for remaining states
  - **Refine FSM**
  - Mentally execute
  - Works for normal sequence
  - Check unusual cases
  - All colored buttons pressed
    - Door open!
    - Change conditions: other buttons NOT pressed also

Controller for laser timer FSM

- General form

3.4 Laser timer FSM

- Inputs: I, O
- Controller
  - Combinational logic
  - State register
- FSM inputs
  - I
- FSM outputs
  - O
  - State register
Controller Design Process

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 1: Capture the FSM</td>
<td>Create an FSM that describes the desired behavior of the controller.</td>
</tr>
</tbody>
</table>

2A: Set up architecture
2B: Encode the states
2C: Fill in the truth table
2D: Implement combinational logic

- Use state register of appropriate width and combinational logic. The logic’s inputs are the state register bits and the FSM inputs, outputs are next state bits and the FSM outputs.
- Assign unique binary number (encoding) to each state. Usually use fewest bits, assign encoding to each state by counting up in binary.
- Translate FSM to truth table for combinational logic such that the logic will generate the outputs and next state signals for the given FSM. Ordering the inputs with state bits first makes the correspondence between the table and the FSM clear.
- Implement the combinational logic using any method.

Controller Design: Laser Timer Example

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

Controller Design: Laser Timer Example (cont)

- Step 2C: Fill in the truth table

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Controller Design: Laser Timer Example (cont)

- Step 2D: Implement combinational logic

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Controller Design: Laser Timer Example (cont)

- Step 2D: Implement combinational logic (cont)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Understanding the Controller’s Behavior
Controller Example: Button Press Synchronizer

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

Controller Example: Sequence Generator

- Want generate sequence 0001, 0011, 1100, 1000, (repeat)

Step 1: FSM (get from table)

\[
\begin{align*}
&\text{Inputs: } \text{bi} \quad \text{Outputs: } \text{bo} \\
&s1 = 0, s0 = 0 \Rightarrow \text{bo} = 0 \\
&s1 = 0, s0 = 1 \Rightarrow \text{bo} = 1 \\
&s1 = 1, s0 = 0 \Rightarrow \text{bo} = 1 \\
&s1 = 1, s0 = 1 \Rightarrow \text{bo} = 0
\end{align*}
\]

Step 2A: Set up architecture

- \( \text{bo} = \text{s1' s0' bi} + \text{s1' s0 bi} + \text{s1 s0' bi} \)

Step 2B: Encode states

- \( \text{bo} = \text{s1' s0' bi} + \text{s1' s0 bi} + \text{s1 s0' bi} \)

Step 2C: Fill in truth table

- \( \text{bi} = 0 \rightarrow \text{bo} = 1 \text{ for } \text{s1} = 1, \text{s0} = 0 \)

Step 2D: Implement combinational logic

- \( \text{bo} = s1 s0' bi + s1 s0 bi + s0' s1' bi \)

Controller Example: Secure Car Key

- (from earlier example)

Controller Example: Button Press Synchronizer (cont)

- FSM inputs: bi; FSM outputs: bo

Reverse Engin. the D-flip-flop Flight Attten. Call Button

- 2D: Circuit to eqns

- 2B: (Un)encode

- 2A: Set up arch - already done

Work backwards

Converting a Circuit to FSM (Reverse Engineering)

- Step 1: FSM (get from table)

- Step 2D: Circuit to eqns

- What does this circuit do?

- Step 2C: Fill in truth table

- Step 2A: Set up architecture

- Step 2B: Encode states

- Controller Example: Enable D-flip flop

- Step 1: Fill (get from table)

- Step 2: Implement combinational logic

- Controller Example: Enable D-flip flop (cont)

- Controller Example: Enable D-flip flop (cont)

- Step 1: Capture FSM

- Step 2A: Set up architecture

- Step 2B: Encode states

- Step 2C: Fill in truth table

- Step 2D: Implement combinational logic
Common Mistakes when Capturing FSMs

- Non-exclusive transitions
- Incomplete transitions

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
  - One condition true: OR of all conditions of transitions leaving a state) should equal 1
- Example

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)

Simplifying Notations

- FSMs
  - Assume unassigned output implicitly assigned 0
- Sequential circuits
  - Assume unconnected clock inputs connected to same external clock

Mathematical Formalisms

- Two formalisms to capture behavior thus far
  - Boolean equations for combinational circuit design
  - FSMs for sequential circuit design
- Not necessary
  - But tremendously beneficial
    - Structured methodology
    - Correct circuits
    - Automated design, automated verification, many more advantages

More on Flip-Flops and Controllers

- Non-ideal flip-flop behavior
  - Can’t change flip-flop input too close to clock edge
  - Setup time: time D must be held stable before edge
    - Else, stable value not present at internal latch
  - Hold time: time D must be held stable after edge
    - Else, new value doesn’t have time to loop around and stabilize in internal latch

Setup time validation

Leads to oscillation!
Metastability

- Violating setup/hold time can lead to bad situation
  - Metastable state: Any flip-flop state other than stable 1 or 0
    - Eventually settles to either, but we don’t know which
    - For internal circuits, we can make sure to observe setup time
    - But what if input is 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

Example of Reducing Metastability Probability

- Recall earlier secure car key controller

Flip-Flop Set and Reset Inputs

- Some flip-flops have additional reset/set inputs
  - Synchronous
    - Synch. reset: Clears Q to 0 on next clock edge
    - Synch. set: Sets Q to 1 on next clock edge
    - Have priority over D input
  - Asynchronous
    - Asynch. reset: Clear Q to 0, independently of clock
    - Asynch. set: set Q to 1, indep. of clock

Initial State of a Controller

- All our FSMs had initial state
  - But our sequential circuits did not
  - Can accomplish using flip-flops with reset/set inputs
    - Shown circuit initializes flip-flops to 01
  - Designer must ensure reset-controller input is 1 during power up of circuit
    - By electronic circuit design

Glitching

- Glitch: Temporary values on outputs that appear soon after input changes, before stable new output values
- Designer must determine whether glitching outputs may pose a problem
  - If so, may consider adding flip-flops to outputs
    - Delays output by one clock cycle, but may be OK
    - Called registered output
- Laser timer controller with flip-flop to prevent glitches on x from unintentionally turning on laser
Glitching

- Alternative registered output approach, avoid 1 cycle delay:
  - Add extra state register bit for each output
  - Connect output directly to its bit
  - No logic between state register flip-flop and output, hence no glitches

But uses more flip-flops, plus more logic to compute next state

Product Profile: Pacemaker

Summary

- FSM model to capture sequential behavior
  - FSM: describe timing behavior of sequential circuit.
  - States, and transitions among states.
- FSM as a Controller
  - Controller-Datapath architecture.
  - Controller design process.
  - Convert FSM to a circuit.
  - Convert a circuit to FSM.
- Miscellaneous
  - Common mistakes when capturing FSMs.
  - Metastability
  - Glitching