State and Finite State Machines

Hakim Weatherspoon
CS 3410, Spring 2013
Computer Science
Cornell University

See P&H Appendix C.7. C.8, C.10, C.11
Stateful Components

Until now is combinatorial logic
- Output is computed when inputs are present
- System has no internal state
- Nothing computed in the present can depend on what happened in the past!

Need a way to record data
Need a way to build **stateful** circuits
Need a state-holding device

Finite State Machines
Goals for Today

State

• How do we store **one** bit?
• Attempts at storing (and changing) one bit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops
  – Master-Slave Flip-Flops
• Register: storing more than one bit, N-bits

Finite State Machines (FSM)

• How do we design logic circuits with state?
• Types of FSMs: Mealy and Moore Machines
• Examples: Serial Adder and a Digital Door Lock
Goal

How do we store store one bit?
First Attempt: Unstable Devices

A

B

C
Second Attempt: Bistable Devices

• Stable and unstable equilibria?

A Simple Device
Third Attempt: Set-Reset Latch

Can you store a value (with this circuit)?
Can you change its value?
Third Attempt: Set-Reset Latch

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Set-Reset (S-R) Latch
Stores a value Q and its complement
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch
Stores a value $Q$ and its complement

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>$\overline{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A | B | OR | NOR |
---|---|-----|-----|
0 | 0 | 0   | 1   |
0 | 1 | 1   | 0   |
1 | 0 | 1   | 0   |
1 | 1 | 1   | 0   |
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch
Stores a value Q and its complement

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>(\overline{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>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.
Next Goal

How do we avoid the forbidden state of S-R Latch?
Fourth Attempt: (Unclocked) D Latch

Fill in the truth table?

<table>
<thead>
<tr>
<th>D</th>
<th>Q</th>
<th>( \overline{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A | B | OR | NOR |
---|---|----|-----|
0 | 0 | 0  | 1   |
0 | 1 | 1  | 0   |
1 | 0 | 1  | 0   |
1 | 1 | 1  | 0   |
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding the forbidden state.
Next Goal

How do we coordinate state changes to a D Latch?
Clocks

Clock helps coordinate state changes

- Usually generated by an oscillating crystal
- Fixed period; frequency = 1/period

![Diagram of clock pulses with labels for clock high, falling edge, rising edge, clock period, and clock low.]
Can design circuits to change on the rising or falling edge

Trigger on rising edge = positive edge-triggered

Trigger on falling edge = negative edge-triggered

Inputs must be stable just before the triggering edge
Clock Disciplines

Level sensitive

- State changes when clock is high (or low)

Edge triggered

- State changes at clock edge

  - positive edge-triggered
  - negative edge-triggered
Fifth Attempt: D Latch with Clock

Fill in the truth table

<table>
<thead>
<tr>
<th>clk</th>
<th>D</th>
<th>Q</th>
<th>\overline{Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fifth Attempt: D Latch with Clock

Level Sensitive D Latch

Clock high:
set/reset (according to D)

Clock low:
keep state (ignore D)

<table>
<thead>
<tr>
<th>clk</th>
<th>D</th>
<th>Q</th>
<th>(\bar{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>(\bar{Q})</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>(\bar{Q})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Sixth Attempt: Edge-Triggered D Flip-Flop

D Flip-Flop
- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges

Activity#1: Fill in timing graph and values for X and Q

```
clk
D
X
Q
```
Takeaway

Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.
Next Goal

How do we store more than one bit, N bits?
Registers

Register
- D flip-flops in parallel
- shared clock
- extra clocked inputs: write_enable, reset, ...

D0
D1
D2
D3
clk

4-bit reg
clk
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.

An $N$-bit register stores $N$-bits. It is be created with $N$ D-Flip-Flops in parallel along with a shared clock.
An Example: What will this circuit do?

- Reset
- Run

32-bit reg

Clk

WE  R

+1

Decoder

...
Recap

We can now build interesting devices with sensors

• Using combinatorial logic

We can also store data values

• In state-holding elements
• Coupled with clocks
Make sure to go to your Lab Section *this week*

Design Doc for Lab1 due in one week, next Monday, Feb 4th

Completed Lab1 due in two weeks, Monday, Feb 11th

Work *alone*

Homework1 is out

Due in one week, next Wednesday, start early

Work *alone*

**But**, use your resources

- Lab Section, Piazza.com, Office Hours, Homework Help Session,
- Class notes, book, Sections, CSUGLab
Check online syllabus/schedule


Slides and Reading for lectures

Office Hours

Homework and Programming Assignments

Prelims (in evenings):

- Tuesday, February 26th
- Thursday, March 28th
- Thursday, April 25th

Schedule is subject to change
Collaboration, Late, Re-grading Policies

“Black Board” Collaboration Policy
• Can discuss approach together on a “black board”
• Leave and write up solution independently
• Do not copy solutions

Late Policy
• Each person has a total of four “slip days”
• Max of two slip days for any individual assignment
• Slip days deducted first for any late assignment, cannot selectively apply slip days
• For projects, slip days are deducted from all partners
• 20% deducted per day late after slip days are exhausted

Regrade policy
• Submit written request to lead TA, and lead TA will pick a different grader
• Submit another written request, lead TA will regrade directly
• Submit yet another written request for professor to regrade.
Goals for Today

State

• How do we store one bit?
• Attempts at storing (and changing) one bit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops
  – Master-Slave Flip-Flops
• Register: storing more than one bit, N-bits

Finite State Machines (FSM)

• How do we design logic circuits with state?
• Types of FSMs: Mealy and Moore Machines
• Examples: Serial Adder and a Digital Door Lock
Finite State Machines
Next Goal

How do we design logic circuits with state?
Finite State Machines

An electronic machine which has

- external inputs
- externally visible outputs
- internal state

Output and next state depend on

- inputs
- current state
Abstract Model of FSM

Machine is

\[ M = (S, I, O, \delta) \]

- **S**: Finite set of states
- **I**: Finite set of inputs
- **O**: Finite set of outputs
- **\( \delta \)**: State transition function

Next state depends on present input and present state
Automata Model

Finite State Machine

- inputs from external world
- outputs to external world
- internal state
- combinational logic
FSM Example

Input: up or down
Output: on or off
States: A, B, C, or D
Input: \(\text{\text{\text{\text{up or floor}}} = \text{down}}\)
Output: \(\text{\text{\text{\text{up or floor}}} = \text{off}}\)
States: \(\text{\text{\text{A, B, C, or D}}}\)
FSM Example

Legend

\[ i_0i_1i_2.../o_0o_1o_2... \]

\[ S_1S_0 \]

Input: 0=up or 1=down
Output: 1=on or 1=off
States: 00=A, 01=B, 10=C, or 11=D
Outputs and next state depend on both current state and input
Special Case: **Moore Machine**

Outputs depend only on current state
Moore Machine Example

Input: up or down
Output: on or off
States: A, B, C, or D
Activity #2: Create a Mealy FSM for a Serial Adder

Add two infinite input bit streams

- streams are sent with least-significant-bit (lsb) first
- How many states are needed to represent FSM?
- Draw and Fill in FSM diagram

Strategy:
(1) Draw a state diagram (e.g. Mealy Machine)
(2) Write output and next-state tables
(3) Encode states, inputs, and outputs as bits
(4) Determine logic equations for next state and outputs
Two states: S0 (no carry in), S1 (carry in)
Inputs: a and b
Output: z

• z is the sum of inputs a, b, and carry-in (one bit at a time)
• A carry-out is the next carry-in state.
• Arcs labeled with input bits a and b, and output z
Serial Adder: State Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>Current state</th>
<th>z</th>
<th>Next state</th>
</tr>
</thead>
</table>

(2) Write down all input and state combinations
Serial Adder: State Table

(3) Encode states, inputs, and outputs as bits
Serial Adder: Circuit

Next State | Current State | Output
------------|---------------|--------

(4) Determine logic equations for next state and outputs

Combinational Logic Equations
Example: Digital Door Lock

Digital Door Lock

Inputs:

• keycodes from keypad
• clock

Outputs:

• “unlock” signal
• display how many keys pressed so far
Door Lock: Inputs

Assumptions:

- signals are synchronized to clock
- Password is B-A-B

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Ø (no key)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>‘A’ pressed</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>‘B’ pressed</td>
</tr>
</tbody>
</table>
Door Lock: Outputs

Assumptions:
- High pulse on U unlocks door

Strategy:
1. Draw a state diagram (e.g. Moore Machine)
2. Write output and next-state tables
3. Encode states, inputs, and outputs as bits
4. Determine logic equations for next state and outputs
Door Lock: Simplified State Diagram

Idle

G1

“1”

G2

“2”

G3

“3”, U

B1

“1”

B2

“2”

B3

“3”

(1) Draw a state diagram (e.g. Moore Machine)
(1) Draw a state diagram (e.g. Moore Machine)
(2) Write output and next-state tables
Door Lock: Simplified State Diagram

(2) Write output and next-state tables
(2) Write output and next-state tables
Door Lock: Simplified State Diagram

(2) Write output and next-state tables

<table>
<thead>
<tr>
<th>Cur. State</th>
<th>Input</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Ø</td>
<td>Idle</td>
</tr>
<tr>
<td>Idle</td>
<td>“B”</td>
<td>G1</td>
</tr>
<tr>
<td>Idle</td>
<td>“A”</td>
<td>B1</td>
</tr>
<tr>
<td>G1</td>
<td>Ø</td>
<td>G1</td>
</tr>
<tr>
<td>G1</td>
<td>“A”</td>
<td>G2</td>
</tr>
<tr>
<td>G1</td>
<td>“B”</td>
<td>B2</td>
</tr>
<tr>
<td>G2</td>
<td>Ø</td>
<td>B2</td>
</tr>
<tr>
<td>G2</td>
<td>“B”</td>
<td>G3</td>
</tr>
<tr>
<td>G2</td>
<td>“A”</td>
<td>Idle</td>
</tr>
<tr>
<td>G3</td>
<td>any</td>
<td>Idle</td>
</tr>
<tr>
<td>B1</td>
<td>Ø</td>
<td>B1</td>
</tr>
<tr>
<td>B1</td>
<td>K</td>
<td>B2</td>
</tr>
<tr>
<td>B2</td>
<td>Ø</td>
<td>B2</td>
</tr>
<tr>
<td>B2</td>
<td>K</td>
<td>Idle</td>
</tr>
</tbody>
</table>
State Table Encoding

<table>
<thead>
<tr>
<th>D3</th>
<th>S2</th>
<th>S1</th>
<th>S0</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

State Table

<table>
<thead>
<tr>
<th>State</th>
<th>S2</th>
<th>S1</th>
<th>S0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>G1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>G2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>G3</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Input: K, A, B
Output: D0, D1, D2, D3
Next State: S'2, S'1, S'0

<table>
<thead>
<tr>
<th>K</th>
<th>A</th>
<th>B</th>
<th>S'2</th>
<th>S'1</th>
<th>S'0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<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>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Bits, and outputs as bits
Strategy:
(1) Draw a state diagram (e.g. Moore Machine)
(2) Write output and next-state tables
(3) Encode states, inputs, and outputs as bits
(4) Determine logic equations for next state and outputs
Door Lock: Implementation

Determine logic equations for next state and outputs

U = \overline{S_2}S_1S_0

D_0 = \overline{S_2}S_1S_0 + \overline{S_2}S_1S_0 + S_2\overline{S_1}S_0

D_1 = \overline{S_2}S_1S_0 + \overline{S_2}S_1S_0 + \overline{S_2}S_1S_0

(4) Determine logic equations for next state and outputs
(4) Determine logic equations for next state and outputs.

$$S_2' = S_2S_1S_0 K\overline{A}B + \overline{S}_2S_1S_0 K\overline{A}B + S_2S_1S_0 KAB + \overline{S}_2S_1S_0 K + S_2S_1S_0 KAB$$

$$D_0 = \overline{S}_2S_1S_0 + \overline{S}_2S_1S_0 + S_2S_1S_0$$

$$D_1 = \overline{S}_2S_1S_0 + \overline{S}_2S_1S_0 + \overline{S}_2S_1S_0$$
Summary

We can now build interesting devices with sensors

• Using combinational logic

We can also store data values

• Stateful circuit elements (D Flip Flops, Registers, ...)
• Clock to synchronize state changes
• State Machines or Ad-Hoc Circuits