State and Finite State Machines

Anne Bracy
CS 3410
Computer Science
Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.

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

Combinational logic

• Output computed directly from inputs
• System has no internal state
• Nothing depends on the past!

Need:

• to record data
• to build stateful circuits
• a state-holding device

Enter: Sequential Logic & Finite State Machines
Goals for Today

State

• Storing 1 bit
  – Bistable Circuit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops

• Storing N bits:
  – Registers
  – Memory

Finite State Machines (FSM)

• Mealy and Moore Machines
• Serial Adder Example
Round 1: Bistable Circuit

• Stable and unstable equilibria?

In stable state, $\bar{A} = B$

How do we change the state?
Round 2: Set-Reset (SR) 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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>

For reference:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Round 2: Set-Reset (SR) 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>Q</td>
<td>( \overline{Q} )</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</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>0</td>
</tr>
</tbody>
</table>

What happens when \( S, R \) changes from 1,1 to 0,0?
\( Q, \overline{Q} \) become unstable, oscillate (0,0 → 1,1 → 0,0)

If \( Q \) is 1, will stay 1
If \( Q \) is 0, will stay 0

For reference:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
</tr>
</tbody>
</table>

For reference:

Hotel California
What’s wrong with using the SR Latch to store 1 bit?

A. Q is undefined when S=0 and R=0  
   (That’s why this is called the forbidden state.)
B. Q oscillates between 0 and 1 when the inputs transition from S=1 and R=1 \( \rightarrow \) S=0 and R=0
C. The SR Latch is problematic b/c it has two outputs to store a single bit.
D. There is nothing wrong with the SR Latch!
Round 3: D Latch (1)

- Inverter prevents SR Latch from entering 1,1 state
- C = enables change

C = 1, D Latch *transparent*: set/reset (according to D)
C = 0, D Latch *opaque*: keep state (ignore D)

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</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>
Round 3: D Latch (1)

- Inverter prevents SR Latch from entering 1,1 state
- C = enables change

C = 1, D Latch *transparent*: set/reset (according to D)
C = 0, D Latch *opaque*: keep state (ignore D)

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>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>
Aside: Clocks
Clock helps coordinate state changes

- Fixed period
- Frequency = 1/period
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
Clock Methodology

- Negative edge, synchronous

Edge-Triggered → signals must be stable near falling edge
“near” = before and after

\[ t_{\text{setup}} \quad t_{\text{hold}} \]
Round 3: D Latch (2)

- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- clk = enables change

<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>0</td>
<td>1</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>

```
clk
---
D
---
Q
---
```
Round 3: D Latch (2)

- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- clk = enables change

<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>Q</td>
<td>\overline{Q}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>\overline{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>
Round 4: D Flip-Flop

- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges
**Round 4: D Flip-Flop**

Clock = 1:  
L1 *transparent*  
L2 *opaque*

Thus, on edge of the clock (when *CLK* falls from 1→0)  
(D passes through to Q)

Clock = 0:  
L1 *opaque*  
L2 *transparent*  

Thus, on edge of the clock (when *CLK* falls from 1→0)  
(D passes through to Q)
DFF Activity: Fill in the timing graph

Clock = 1
D passes through L1 to X

Clock = 0
X passes through L2 to Q
Round 4: D Flip-Flop

- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges
Goals for Today

State

• Storing 1 bit
  – Bistable Circuit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops

• Storing N bits:
  – Registers
  – Memory

Finite State Machines (FSM)

• Mealy and Moore Machines
• Serial Adder Example
Registers

- D flip-flops in parallel
- shared clock
- Additional (optional) inputs: writeEnable, reset, ...

![Diagram of 4-bit register with D inputs and shared clock]
Register File

- N read/write registers
- Indexed by register number

Register File

**Dual-Read-Port**

**Single-Write-Port**

32 x 32

Register File
Writing to the Register File (1)

Register File
- N read/write registers
- Indexed by register number

addi r5, r0, 10

How to write to one register in the register file?
- Need a decoder
## Activity: 3-to-8 decoder truth table & circuit

<table>
<thead>
<tr>
<th>i2</th>
<th>i1</th>
<th>i0</th>
<th>o0</th>
<th>o1</th>
<th>o2</th>
<th>o3</th>
<th>o4</th>
<th>o5</th>
<th>o6</th>
<th>o7</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>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<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>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
<td>0</td>
<td>1</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>
<td>1</td>
<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>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

3-to-8 decoder

3

$R_W$

101
Activity: 3-to-8 decoder truth table & circuit

<table>
<thead>
<tr>
<th>i2</th>
<th>i1</th>
<th>i0</th>
<th>o0</th>
<th>o1</th>
<th>o2</th>
<th>o3</th>
<th>o4</th>
<th>o5</th>
<th>o6</th>
<th>o7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3-to-8 decoder

R_W

o0

o5
Writing to the Register File (2)

Register File

- N read/write registers
- Indexed by register number

addi r5, r0, 10

How to write to one register in the register file?

- Need a decoder
Reading from the Register File

Register File

• N read/write registers
• Indexed by register number

How to read from two registers?
• Need a multiplexor
Register File

- N read/write registers
- Indexed by register number

Implementation:
- D flip flops to store bits
- Decoder for each write port
- Mux for each read port
Register File

- N read/write registers
- Indexed by register number

Implementation:
- D flip flops to store bits
- Decoder for each write port
- Mux for each read port
Tradeoffs

Register File tradeoffs

+ Very fast (a few gate delays for both read and write)
+ Adding extra ports is straightforward

– Doesn’t scale

e.g. 32Mb register file with 32 bit registers

Need 32x 1M-to-1 multiplexor and 32x 20-to-1M decoder

How many logic gates/transistors?
Takeaways

• 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.

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

• A D Flip-Flip 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 plus a shared clock.
Memory

• Storage Cells + bus
• Inputs: Address, Data (for writes)
• Outputs: Data (for reads)
• Also need R/W signal (not shown)

- N address bits $\rightarrow 2^N$ words total
- M data bits $\rightarrow$ each word M bits
Memory

- Storage Cells + bus
- Decoder selects a word line
- R/W selector determines access type
- Word line is then coupled to the data lines
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
E.g. How do we design a 4 x 2 Memory Module?

(i.e. 4 word lines that are each 2 bits wide)?
iClicker Question

What’s your familiarity with memory (SRAM, DRAM)?

A. I’ve never heard of any of this.
B. I’ve heard the words SRAM and DRAM, but I have no idea what they are.
C. I know that DRAM means main memory.
D. I know the difference between SRAM and DRAM and where they are used in a computer system.
Typical SRAM Cell

Each cell stores one bit, and requires 4 – 8 transistors (6 is typical)
SRAM Summary

SRAM

• A few transistors (~6) per cell
• Used for working memory (caches)

• But for even higher density...
Dynamic RAM: DRAM

Dynamic-RAM (DRAM)

- Data values require constant refresh

Each cell stores one bit, and requires 1 transistor.
Dynamic RAM: DRAM

Dynamic-RAM (DRAM)

- Data values require constant refresh

Each cell stores one bit, and requires 1 transistors
DRAM vs. SRAM

Single transistor vs. many gates
• Denser, cheaper ($30/1GB vs. $30/2MB)
• But more complicated, and has analog sensing

Also needs refresh
• Read and write back...
• ...every few milliseconds
• Organized in 2D grid, so can do rows at a time
• Chip can do refresh internally

Hence... slower and energy inefficient
Memory

Register File tradeoffs

+ Very fast (a few gate delays for both read and write)
+ Adding extra ports is straightforward
  – Expensive, doesn’t scale
  – Volatile

Volatile Memory alternatives: SRAM, DRAM, ...

  – Slower
  + Cheaper, and scales well
  – Volatile

Non-Volatile Memory (NV-RAM): Flash, EEPROM, ...

  + Scales well
  – Limited lifetime; degrades after 100000 to 1M writes
Goals for Today

State

• Storing 1 bit
  – Bistable Circuit
  – Set-Reset Latch
  – D Latch
  – D Flip-Flops

• Storing N bits:
  – Registers
  – Memory

Finite State Machines (FSM)

• Mealy and Moore Machines
• Serial Adder Example
Finite State Machines

An electronic machine which has

• external inputs
• externally visible outputs
• internal state

Output and next state depend on

• inputs
• current state
Automata Model

Finite State Machine

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

Diagram:

- Registers
- Current State
- Input
- Comb. Logic
- Output
- Next State
Input: 1 or 0
Output: 1 or 0
States: A or B

What input pattern is the FSM “looking for”?

(Mealy Machine)
Mealy Machine

General Case: Mealy Machine

Outputs and next state depend on both current state and input
Special Case: **Moore Machine**

Outputs depend only on current state

Legend:
- **state out**
- **start out**

```
input

start out

state out
```

```
A

1

0

B

1

0

C
```

```
0

1
```

```
1
```
Activity: Build a Logic Circuit 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

...10110

...01111

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
FSM: State Diagram – start here

___ states:
Inputs: ??? and ???
Output: ???

•  

...10110

...01111

...00101
FSM: State Diagram

___ states:
Inputs: ??? and ???
Output: ???
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.
-
(2) Write down all input and state combinations
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
(2) Write down all input and state combinations
(3) Encode states, inputs, and outputs as bits
Serial Adder: State Assignment

(3) Encode states, inputs, and outputs as bits

Two states, so 1-bit is sufficient
(same flip-flop will encode the state)
**FSM: State Diagram**

<table>
<thead>
<tr>
<th>??</th>
<th>??</th>
<th>Current state</th>
<th>?</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

Combinational Logic Equations

\[ z = \overline{a} b s + a b s + \overline{a} b s + a b \]
\[ s' = a b s + a b s + \overline{a} b s + a b \]
Sequential Logic Circuits

Next State $s'$

Current State $s$

Input $a$ $b$

Output $z$

Next State $s'$

$\begin{align*}
z &= \overline{ab}s + \overline{ab}s + \overline{abs} + \overline{abs} + \overline{abs} + \overline{abs} \\
s' &= ab\overline{s} + a\overline{bs} + a\overline{bs} + \overline{abs} + \overline{abs}
\end{align*}$

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
Summary

We can now store data values

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