CSE140: Components and Design Techniques for Digital Systems

Sequential Circuit Introduction
Latches and Flip-Flops

Tajana Simunic Rosing
Where we are now….

• What we covered last time:
  – ALUs and other combinational circuits

• What we’ll do next:
  – Sequential circuits and memory

• Upcoming deadlines:
  – ZyBook today: Sec 4.1-2; Thursday: 4.3-6
  – HW#4 assigned, due next Tuesday
  – Quiz #4 on Thursday!

• Excellent results on the Midterm #2
  – Coming up in 10 days – generally harder than Midterm#1
  – Cumulative
What is a sequential circuit?

A circuit whose output depends on current inputs and past outputs

A circuit with memory

\[ y_i = f_i(S^t, X) \]
\[ s_{i+1} = g_i(S^t, X) \]
Why do we need circuits with ‘memory’?

- Circuits with memory can be used to store data
- Systems have circuits that run a sequence of tasks

**Memory Hierarchy**

- Hard disk
- Main Memory
- Cache
- Registers

Sources: TSR, Katz, Boriello & Vahid
Simplest memory element

"data"  "load"  "remember"  "stored value"
Flight attendant call button

- Flight attendant call button
  - Press call: light turns on
    - **Stays on** after button released
  - Press cancel: light turns off
  - Logic circuit to implement this?

- SR latch implementation
  - Call=1 : sets Q to 1 and keeps it at 1
  - Cancel=1 : resets Q to 0

![Diagram of SR latch implementation]

1. **Call button pressed** – light turns on
2. **Call button released** – light stays on
3. **Cancel button pressed** – light turns off

Sources: TSR, Katz, Boriello & Vahid
SR Latch Analysis

- $S = 1, R = 0$: 
  
  then $Q = 1$ and $\overline{Q} = 0$

- $S = 0, R = 1$: 
  
  then $Q = 1$ and $\overline{Q} = 0$
SR Latch Analysis

– $S = 0, R = 0$:
  then $Q = Q_{prev}$
  – Memory!

– $S = 1, R = 1$:
  then $Q = 0, \bar{Q} = 0$
  – Invalid State
    $\bar{Q} \neq \text{NOT } Q$
What if a kid presses both call and cancel & then releases them?

- If $S=1$ and $R=1$ at the same time and then released, $Q =$?
  - Can also occur also due to different delays of different paths
  - $Q$ may oscillate and eventually settle to 1 or 0 due to diff. path delay

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>R</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td>hold</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td>not allowed</td>
</tr>
</tbody>
</table>
SR Latch Symbol

- SR stands for Set/Reset Latch
  - Stores one bit of state ($Q$)
- Control what value is being stored with $S$, $R$ inputs
  - **Set:** Make the output 1
    
    ($S = 1$, $R = 0$, $Q = 1$)
  - **Reset:** Make the output 0
    
    ($S = 0$, $R = 1$, $Q = 0$)
  - **Hold:** Keep data stored
    
    ($S = 0$, $R = 0$, $Q = Q_{\text{previous}}$)

Sources: TSR, Katz, Boriello & Vahid
SR Latch Characteristic Equation

To analyze, break the feedback path

\[
\begin{array}{ccc|c}
S & R & Q(t) & Q(t+\Delta) \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 \\
1 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 \\
1 & 1 & 0 & X \\
1 & 1 & 1 & X \\
\end{array}
\]

- **hold**
- **reset**
- **set**
- **not allowed**

characteristic equation

\[
Q(t+\Delta) = S + R' Q(t)
\]

Sources: TSR, Katz, Boriello & Vahid
Avoiding $S=R=1$ Part 1: Level-Sensitive SR Latch

- Add input “C”
  - Change C to 1 only after S and R are stable
  - C is usually a clock (CLK)
**Clocks**

- **Clock** -- Pulsing signal for enabling latches; ticks like a clock
- **Synchronous** circuit: sequential circuit with a clock

- **Clock period**: time between pulse starts
  - Above signal: period = 20 ns
- **Clock cycle**: one such time interval
  - Above signal shows 3.5 clock cycles
- **Clock duty cycle**: time clock is high
  - 50% in this case
- **Clock frequency**: $1$/period
  - Above: freq = $1/20\text{ns} = 50\text{MHz}$
Clock question

The clock shown in the waveform below has:

A. Clock period of 4ns with 250MHz frequency  
B. Clock duty cycle 75%  
C. Clock period of 1ns with 1GHz frequency  
D. A. & B.  
E. None of the above

Sources: TSR, Katz, Boriello & Vahid
Avoiding S=R=1 Part 2: Level-Sensitive D Latch

- SR latch requires careful design so SR=11 never occurs
- D latch helps by inserting the inverter between S & R inputs
  - Inserted inverter ensures R is always the opposite of S when C=1
D Latch Truth Table

<table>
<thead>
<tr>
<th>CLK</th>
<th>D</th>
<th>̅D</th>
<th>S</th>
<th>R</th>
<th>Q prev</th>
<th>̅Q prev</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>̅X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
D Latch Summary

- Two inputs: $CLK$, $D$
  - $CLK$: controls when the output changes
  - $D$ (the data input): controls what the output changes to

- Function
  - When $CLK = 1$,
    $D$ passes through to $Q$ (*transparent*)
  - When $CLK = 0$,
    $Q$ holds its previous value (*opaque*)

- (Mostly) avoids invalid case $Q = Q'$
Level-Sensitive D Latches

Assume that data in all latches is initially 0. Input Y=1 and Clk transitions from 0->1. When Clk=0 again, the stored values in latches are:

A. Q1=1, Q2=0, Q3=0, Q4=0 for both clock A & B
B. Q1=1, Q2=1, Q3=1, Q4=1 for clock A
   Q1=1, Q2=0, Q3=0, Q4=0 for clock B
C. Q1=1, Q2=1, Q3=1, Q4=1 for both clocks
D. More information is needed to determine the answer
E. None of the above
• **Flip-flop**: Bit storage that stores on the clock edge, not level
• Master-slave design: master loads when Clk=0, then slave when Clk=1
D Flip-Flop: Characteristic Equation

Characteristic Equation

\[ Q(t+1) = D(t) \]
Rising vs. Falling Edge D Flip-Flop

The triangle means clock input, edge triggered

Symbol for rising-edge triggered D flip-flop

Symbol for falling-edge triggered D flip-flop

Internal design: Just invert servant clock rather than master

rising edges

falling edges

Sources: TSR, Katz, Boriello & Vahid
Enabled D-FFs

• **Inputs:** $CLK$, $D$, $EN$
  – The enable input ($EN$) controls when new data ($D$) is stored

• **Function**
  – $EN = 1$: $D$ passes through to $Q$ on the clock edge
  – $EN = 0$: the flip-flop retains its previous state
Additional D-FF Features

• Reset (set state to 0) – R
  – synchronous: \( D_{\text{new}} = R' \cdot D_{\text{old}} \) (when next clock edge arrives)
  – asynchronous: doesn't wait for clock
• Preset or set (set state to 1) – S (or sometimes P)
  – synchronous: \( D_{\text{new}} = D_{\text{old}} + S \) (when next clock edge arrives)
  – asynchronous: doesn't wait for clock
• Both reset and preset
  – \( D_{\text{new}} = R' \cdot D_{\text{old}} + S \) (set-dominant)
  – \( D_{\text{new}} = R' \cdot D_{\text{old}} + R'S \) (reset-dominant)
• Selective input capability (input enable or load) – LD or EN
  – multiplexor at input: \( D_{\text{new}} = LD' \cdot Q + LD \cdot D_{\text{old}} \)
  – load may or may not override reset/set (usually R/S have priority)
• Complementary outputs – Q and Q'
D Flip-Flops

Assume that the data in all D-FFs is initially 0. Input Y=1. When Clk goes from 0->1, the stored values in D-FFs are:

A. Q1=1, Q2=0, Q3=0, Q4=0 for both clock A & B
B. Q1=1, Q2=1, Q3=1, Q4=1 for clock A
   Q1=1, Q2=0, Q3=0, Q4=0 for clock B
C. Q1=1, Q2=1, Q3=1, Q4=1 for both clocks
D. More information is needed to determine the answer
E. None of the above
**Bit Storage Overview**

- **SR latch**: S=1 sets Q to 1, R=1 resets Q to 0. Problem: SR=11 yield undefined Q.

- **Level-sensitive SR latch**: S and R only have effect when C=1. We can design outside circuit so SR=11 never happens when C=1. Problem: avoiding SR=11 can be a burden.

- **D latch**: SR can’t be 11 if D is stable before and while C=1, and will be 11 for only a brief glitch even if D changes while C=1. Problem: C=1 too long propagates new values through too many latches: too short may not enable a store.

- **D flip-flop**: Only loads D value present at rising clock edge, so values can’t propagate to other flip-flops during same clock cycle. Tradeoff: uses more gates internally than D latch, and requires more external gates than SR – but gate count is less of an issue today.

Sources: TSR, Katz, Boriello & Vahid
Comparison of latches and flip-flops

- **positive edge-triggered flip-flop**

- **level-sensitive latch**

Sources: TSR, Katz, Boriello & Vahid
Registers and Counters
Building blocks with FFs: Basic Register
Shift register

- Holds & shifts samples of input
Pattern Recognizer

- Combinational function of input samples
Design of a Universal Shift Register

### Nth cell

<table>
<thead>
<tr>
<th>clear</th>
<th>s0</th>
<th>s1</th>
<th>new value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>output</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>output value of FF to left (shift right)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>output value of FF to right (shift left)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>input</td>
</tr>
</tbody>
</table>

### Diagram

- **Input** to Nth cell
- **Output** from Nth cell
- **Left_in** to Nth cell
- **Left_out** from Nth cell
- **Clear** to Nth cell
- **Right_in** to Nth cell
- **Right_out** from Nth cell
- **Clock** to Nth cell
- **Q[N-1]** to Nth cell
- **Q[N+1]** to Nth cell
- **Input[N]** to Nth cell
- **Output** from Nth cell
- **Q** to Nth cell
- **Clock** to Nth cell
- **Clear** to Nth cell
- **S0 and S1** to control mux

### Control mux

- **S0** and **S1** to control mux
- **0**, **1**, **2**, **3** to control mux

### Purpose

- The diagram illustrates the design of a universal shift register, showing how data is shifted left or right, cleared, and how input values are handled through the control signals. The table provides the new values for different control signals, allowing for the implementation of various shift register functions.
Counters

• Sequences through a fixed set of patterns
General Counters

- Default operation: count up
- QA-QD counter output
- A-D parallel load data
- LOAD enables data load
- RCO ripple carry out
- CLR clears data
- EN counter enable
Finite State Machines
Circuit Specifications

- **Combinational Logic**
  - Truth tables, Boolean equations, logic diagrams (no feedback)

- **Sequential Networks**: State Diagram (Memory)
  - State and Excitation Tables
  - Characteristic Expression
  - Logic Diagram (FFs and feedback loops)

RTL: Register-Transfer Level Description
Finite State Machines: Two Bit Counter Example

Symbol/ Circuit

State Diagram

Current state | Next State
---|---
S₀ | S₁
S₁ | S₂
S₂ | S₃
S₃ | S₀

State Table

<table>
<thead>
<tr>
<th>Q₁(t)</th>
<th>Q₀(t)</th>
<th>Q₁(t+1)</th>
<th>Q₀(t+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Which is the most likely circuit realization of the two bit counter?

**State Table**

<table>
<thead>
<tr>
<th>$Q_1(t)$</th>
<th>$Q_0(t)$</th>
<th>$Q_1(t+1)$</th>
<th>$Q_0(t+1)$</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**A.** Combinational circuit

**B.** Circuit with 2 flip flops

**C.** Circuit with one flip flop

Sources: TSR, Katz, Boriello & Vahid
Two Bit Counter Circuit

State Table

<table>
<thead>
<tr>
<th>$Q_1(t)$</th>
<th>$Q_0(t)$</th>
<th>$Q_1(t+1)$</th>
<th>$Q_0(t+1)$</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>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

We store the current state using D-flip flops so that:

- Inputs to the combinational circuit don’t change while the next output is computed
- The transition to the next state only occurs at the rising edge of the clock

$$D_0(t) = Q_0(t)'$$
$$D_1(t) = Q_0(t) Q_1(t)' + Q_0(t)' Q_1(t)$$
FSM Definition

- FSM consists of
  - Set of states
  - Set of inputs, set of outputs
  - Initial state
  - Set of transitions
    - Only one can be true at a time

- FSM representations:
  - State diagram
  - State table
Watch for transition properties!
(note that more transitions need to be added)

FSM Example

Is this FSM fully defined?
A. Yes
B. No

Sources: TSR, Katz, Boriello & Vahid
FSM Controller Design Process with a Three Bit Counter Example

1. State Diagram
2. State Table
3. State Assignments
4. Excitation Table
   (present state, inputs; next state, outputs)
5. Circuit

State Table with Assigned State Patterns

<table>
<thead>
<tr>
<th>C3</th>
<th>C2</th>
<th>C1</th>
<th>N3</th>
<th>N2</th>
<th>N1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</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>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Mealy and Moore Machines

Mealy Machine: \( y_i(t) = f_i(X(t), S(t)) \)
Moore Machine: \( y_i(t) = f_i(S(t)) \)

\[ s_i(t+1) = g_i(X(t), S(t)) \]
This Counter Design Is:

A. Moore machine  
B. Mealy machine  
C. None of the above
Mars rover has a binary input x. When it receives the input sequence \( x(t-2, t) = 001 \) from its life detection sensors, it means that the it has detected life on Mars 🌟 and the output \( y(t) = 1 \), otherwise \( y(t) = 0 \) (no life on Mars 🌙).

This pattern recognizer should have
A. One state because it has one output
B. One state because it has one input
C. Two states because the input can be 0 or 1
D. More than two states because ….
E. None of the above
Which of the following diagrams is a correct Mealy solution for the 001 pattern recognizer on the Mars rover?

A. 1/0

B. 1/0

C. Both A and B are correct

D. None of the above
Mars Life Recognizer FFs

Pattern Recognizer ‘001’

What does state table need to show to design controls of C1?
A. (current input x(t), current state S(t) vs. next state, S(t+1))
B. (current input, current state vs. current output y(t))
C. (current input, current state vs. current output, next state)
D. None of the above

Sources: TSR, Katz, Boriello & Vahid
State Diagram => State Table with State Assignment

<table>
<thead>
<tr>
<th>S(t) \ x</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1,0</td>
<td>S0,0</td>
</tr>
<tr>
<td>S1</td>
<td>S2,0</td>
<td>S0,0</td>
</tr>
<tr>
<td>S2</td>
<td>S2,0</td>
<td>S0,1</td>
</tr>
</tbody>
</table>

State Assignment
S0: 00
S1: 01
S2: 10

<table>
<thead>
<tr>
<th>S(t) \ x</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01,0</td>
<td>00,0</td>
</tr>
<tr>
<td>01</td>
<td>10,0</td>
<td>00,0</td>
</tr>
<tr>
<td>10</td>
<td>10,0</td>
<td>00,1</td>
</tr>
</tbody>
</table>

Q_{1(t+1)}Q_{0(t+1)}, y
State Diagram $\Rightarrow$ State Table $\Rightarrow$ Excitation Table $\Rightarrow$ Circuit

<table>
<thead>
<tr>
<th>$Q_1(t)$ $Q_0(t)$ $x$</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01,0</td>
<td>00,0</td>
</tr>
<tr>
<td>01</td>
<td>10,0</td>
<td>00,0</td>
</tr>
<tr>
<td>10</td>
<td>10,0</td>
<td>00,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>id</th>
<th>$Q_1 Q_0 x$</th>
<th>$D_1$</th>
<th>$D_0$</th>
<th>$y$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello & Vahid
State Diagram => State Table => Excitation Table => Circuit

<table>
<thead>
<tr>
<th>id</th>
<th>Q₁Q₀x</th>
<th>D₁</th>
<th>D₀</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

\[ D₁(t) = x'Q₀ + x'Q₁ \]
\[ D₀(t) = Q'_1Q'_0 x' \]
\[ y = Q₁x \]
State Diagram => State Table => Excitation Table => Circuit

\[ D_1(t) = x'Q_0 + x'Q_1 \]
\[ D_0(t) = Q_1'Q_0'x' \]
\[ y = Q_1x \]
Which of the following diagrams is a correct Moore solution to the ‘001’ pattern recognizer?

A. 

B. 

C. Both A and B are correct

D. None of the above

Sources: TSR, Katz, Boriello & Vahid
Moore Mars Life Recognizer: FF Input Specs

Pattern Recognizer ‘001’

What does state table need to show to design controls of C2?
A. (current input $x(t)$, current state $S(t)$ vs. next state, $S(t+1)$)
B. (current input, current state vs. current output $y(t)$)
C. (current state vs. current output $y(t)$ and next state)
D. (current state vs. current output $y(t)$)
E. None of the above
Moore Mars Life Recognizer: State Table

<table>
<thead>
<tr>
<th>S(t) \ x</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S1,0</td>
<td>S0,0</td>
</tr>
<tr>
<td>S1</td>
<td>S2,0</td>
<td>S0,0</td>
</tr>
<tr>
<td>S2</td>
<td>S2,0</td>
<td>S3,0</td>
</tr>
<tr>
<td>S3</td>
<td>S1,1</td>
<td>S0,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q_1Q_0 \ x</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01,0</td>
<td>00,0</td>
</tr>
<tr>
<td>01</td>
<td>10,0</td>
<td>00,0</td>
</tr>
<tr>
<td>10</td>
<td>10,0</td>
<td>11,0</td>
</tr>
<tr>
<td>11</td>
<td>01,1</td>
<td>00,1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Q_1Q_0x</th>
<th>D_1</th>
<th>D_0</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
# Mars Life Recognizer: Combinational Circuit Design

<table>
<thead>
<tr>
<th>id</th>
<th>Q₁Q₀x</th>
<th>D₁</th>
<th>D₀</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

### D₁(t):

<table>
<thead>
<tr>
<th>x(t)</th>
<th>Q₀</th>
<th>Q₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

### D₀(t):

<table>
<thead>
<tr>
<th>x(t)</th>
<th>Q₀</th>
<th>Q₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

### y(t):

<table>
<thead>
<tr>
<th>x(t)</th>
<th>Q₀</th>
<th>Q₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello & Vahid
Mars Life Recognizer Circuit Implementation

State Diagram => State Table => Excitation Table => Circuit
Timing Constraints in Sequential Designs
Timing Constraints in Sequential Circuit Designs

- Our seemingly logically correct design can go wrong – signals don’t travel in zero time 😞
- We next look at timing constraints for combinational and sequential logic.
I. Min delay of a gate, also called contamination delay: $t_{cd}$
   Minimum time from when an input changes until the output *starts* to change

II. Max delay of a gate, also called propagation delay: $t_{pd}$
   Maximum time from when an input changes until the output *is* guaranteed to reach its final value (i.e., stop changing)
Combinational Logic: Output Timing Constraints

Which path in the above circuit determines the contamination delay of the circuit (assuming the delay of all the gates is the same)?

A. Blue path
B. Red path
C. Both
D. Neither
Which path in the above circuit determines the propagation delay of the circuit (assuming the delay of all the gates is the same)?

A. Blue path
B. Red path
C. Both
D. Neither
**D-FF Input Constraints: Setup and Hold Times**

I. Setup time: $t_{\text{setup}}$
   
   Time *before* the clock edge that data must be stable (i.e. not change)

II. Hold time: $t_{\text{hold}}$
   
   Time *after* the clock edge that data must be stable

Aperture time: $t_a$

Time around clock edge that data must be stable ($t_a = t_{\text{setup}} + t_{\text{hold}}$)
Output Timing Constraints

I. Min delay of FF, also called contamination delay or min CLK to Q delay: $t_{ccq}$
   Time after clock edge that $Q$ might be unstable (i.e., starts changing)

II. Max delay of FF, also called propagation delay or maximum CLK to Q delay: $t_{pcq}$
    Time after clock edge that the output $Q$ is guaranteed to be stable (i.e. stops changing)

Sources: TSR, Katz, Boriello & Vahid
The timing of which of the following signals can cause a setup-time violation?

A. The input signal D(t)
B. The output signal Q(t)
C. Both of the above
D. None of the above
Causes of Timing Issues in Sequential Circuits

- Input to a FF comes from the output of another FF through a combinational circuit
- The FF and combinational circuit have a min & max delay

Which of the following violations occurs if max delay of R1 is zero & max delay of the combinational circuit is equal to the clock period?
A. Hold time violation for R2
B. Setup violation for R2
C. Hold time violation for R1
D. Setup violation for R1
E. None of the above

Sources: TSR, Katz, Boriello & Vahid
Setup Time Constraint

- Input to a FF comes from the output of another FF through a combinational circuit.
- The FF and combinational circuit have a min & max delay.

Setup time constraint:

\[ T_c \geq t_{\text{setup}} + \text{max delay(FF)} + \text{max delay(combinational)} \]

\[ T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}} \]
Causes of Timing Issues in Sequential Circuits

- Input to a FF comes from the output of another FF through a combinational circuit
- The FF and combinational circuit have a min & max delay

Which of the violations would occur if the min delay of R1 was zero and the combinational circuit was just a wire?

A. Hold time violation for R2
B. Setup violation for R2
C. Hold time violation for R1
D. Setup violation for R1
E. None of the above
Hold Time Constraint

- Input to a FF comes from the output of another FF through a combinational circuit
- The FF and combinational circuit have a min & max delay

Hold time constraint:

\[ t_{\text{hold}} < \text{min delay(FF)} + \text{min delay(combinational)} \]

\[ t_{\text{hold}} < t_{\text{ccq}} + t_{\text{cd}} \]
Once a flip flop has been built, its timing characteristics stay fixed: $t_{\text{setup}}$, $t_{\text{hold}}$, $t_{\text{ccq}}$, $t_{\text{pcq}}$.

What about the clock? Does the clock edge arrive at the same time to all the D-FFs on the chip?
Clock Skew

- The clock doesn’t arrive at all registers at the same time
- **Skew**: difference between the two clock edges
- Perform the **worst case analysis**

Sources: TSR, Katz, Boriello & Vahid
In the worst case, CLK2 is earlier than CLK1.

- $t_{pcq}$ is max delay through FF, $t_{pd}$ is max delay through logic.

\[ T_c \geq t_{pcq} + t_{pd} + t_{\text{setup}} + t_{\text{skew}} \]

\[ t_{pd} \leq T_c - (t_{pcq} + t_{\text{setup}} + t_{\text{skew}}) \]
In the worst case, CLK2 is later than CLK1

$t_{ccq}$ is min delay through FF, $t_{cd}$ is min delay through logic

$t_{ccq} + t_{cd} > t_{hold} + t_{skew}$

$t_{cd} > t_{hold} + t_{skew} - t_{ccq}$
Timing Analysis Example

Timing Characteristics

\[ t_{ccq} = 30 \text{ ps} \]
\[ t_{pcq} = 50 \text{ ps} \]
\[ t_{setup} = 60 \text{ ps} \]
\[ t_{hold} = 70 \text{ ps} \]

\[ t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps} \]
\[ t_{cd} = 25 \text{ ps} \]

Setup time constraint:

\[ T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps} \]

\[ f_c = 1/T_c = 4.65 \text{ GHz} \]

Hold time constraint:

\[ t_{ccq} + t_{cd} > t_{hold} ? \]
\[ (30 + 25) \text{ ps} > 70 \text{ ps} ? \text{ No!} \]
Timing Analysis Example

Add buffers to the short paths:

\[ t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps} \]
\[ t_{cd} = 2 \times 25 \text{ ps} = 50 \text{ ps} \]

**Setup time constraint:**
\[ T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps} \]
\[ f_c = 1/T_c = 4.65 \text{ GHz} \]

**Hold time constraint:**
\[ t_{ccq} + t_{cd} > t_{hold} \]
\[ (30 + 50) \text{ ps} > 70 \text{ ps} \] **Yes!**

Timing Characteristics

- \( t_{ccq} = 30 \text{ ps} \)
- \( t_{pcq} = 50 \text{ ps} \)
- \( t_{setup} = 60 \text{ ps} \)
- \( t_{hold} = 70 \text{ ps} \)
- \( t_{pd} = 35 \text{ ps} \)
- \( t_{cd} = 25 \text{ ps} \)

Sources: TSR, Katz, Boriello & Vahid
Sequential Circuit Design Summary

- SRAM memory, SR Latch, D Latch, D-FF
- Design procedure for FSMs
  1. Capture FSM
  2. Create state table
  3. Assign the states
  4. Excitation table
  5. Implement the combinational logic
- Mealy vs. Moore FSM
- Non-ideal properties of FFs
  - Setup/hold time constraints
  - Maximum operating frequency
  - Clock skew
MORE FSM EXAMPLES
15 cents for candy! Watch out – no change!

- **Moore machine**
  - outputs associated with state

- **Mealy machine**
  - outputs associated with transitions
Example: Moore implementation

- Encode states and map to logic

<table>
<thead>
<tr>
<th>Present State</th>
<th>Inputs</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1 Q0</td>
<td>D N</td>
<td>D1 D0</td>
<td>Open</td>
</tr>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- -</td>
<td>-</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td>1 1</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- -</td>
<td>-</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- -</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Mealy implementation

<table>
<thead>
<tr>
<th>present state Q1 Q0</th>
<th>inputs D N</th>
<th>next state D1 D0</th>
<th>output open</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- - -</td>
<td>-</td>
</tr>
<tr>
<td>0 1</td>
<td>0 0</td>
<td>0 1 0</td>
<td>0</td>
</tr>
<tr>
<td>0 1</td>
<td>0 1</td>
<td>1 0 0</td>
<td>0</td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td>1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- - -</td>
<td>-</td>
</tr>
<tr>
<td>1 0</td>
<td>0 0</td>
<td>1 0 0</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>1 1 1</td>
<td>1</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
<td>- - -</td>
<td>1</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello & Vahid
FSM design: Multiple input counter

- Given FSM of a multiple input counter, design the circuit implementing its functionality.

<table>
<thead>
<tr>
<th>present state</th>
<th>next state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>S0 S1 S2 S3</td>
<td>1</td>
</tr>
<tr>
<td>S1</td>
<td>S0 S3 S1 S3</td>
<td>0</td>
</tr>
<tr>
<td>S3</td>
<td>S1 S0 S0 S3</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>S1 S3 S2 S0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>State</th>
<th>00</th>
<th>01</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>00</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>01</td>
<td>00</td>
<td>10</td>
</tr>
</tbody>
</table>
Multiple input counter: Logic for D-FF

- Derive logic equations for inputs of D-FF

<table>
<thead>
<tr>
<th>Input</th>
<th>00</th>
<th>01</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>11</td>
<td>00</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>01</td>
<td>00</td>
<td>10</td>
</tr>
</tbody>
</table>

Sources: TSR, Katz, Boriello & Vahid