Chapter 5 – Sequential Circuits

Part 1 – Storage Elements and Sequential Circuit Analysis

HOANG Trang

Overview

- Part 1 - Storage Elements and Analysis
  - Introduction to sequential circuits
  - Types of sequential circuits
  - Storage elements
    - Latches
    - Flip-flops
  - Sequential circuit analysis
    - State tables
    - State diagrams
    - Equivalent states
    - Moore and Mealy Models

- Part 2 - Sequential Circuit Design

- Part 3 – State Machine Design
Introduction to Sequential Circuits

- A Sequential circuit contains:
  - Storage elements: Latches or Flip-Flops
  - Combinational Logic:
    - Implements a multiple-output switching function
    - **Inputs** are signals from the outside.
    - **Outputs** are signals to the outside.
    - Other inputs, **State** or **Present State**, are signals from storage elements.
    - The remaining outputs, **Next State** are inputs to storage elements.
Introduction to Sequential Circuits

- Combinatorial Logic
  - Next state function
    \[ \text{Next State} = f(\text{Inputs, State}) \]
  - Output function (Mealy)
    \[ \text{Outputs} = g(\text{Inputs, State}) \]
  - Output function (Moore)
    \[ \text{Outputs} = h(\text{State}) \]

- Output function type depends on specification and affects the design significantly
Types of Sequential Circuits

Depends on the times at which:

- storage elements observe their inputs, and
- storage elements change their state

- **Synchronous**
  - Behavior defined from knowledge of its signals at discrete instances of time
  - Storage elements observe inputs and can change state only in relation to a timing signal (clock pulses from a clock)

- **Asynchronous**
Types of Sequential Circuits

Synchronous (clock)
Discrete Event Simulation=> to understand sequential circuit

- In order to understand the time behavior of a sequential circuit we use discrete event simulation.

- Rules:
  - Gates modeled by an ideal (instantaneous) function and a fixed gate delay
  - Any change in input values is evaluated to see if it causes a change in output value
  - Changes in output values are scheduled for the fixed gate delay after the input change
  - At the time for a scheduled output change, the output value is changed along with any inputs it drives

Simulated NAND Gate

- Example: A 2-Input NAND gate with a 0.5 ns delay:

Assume A and B have been 1 for a long time
- At time t=0, A changes to a 0 at t= 0.8 ns, back to 1.

<table>
<thead>
<tr>
<th>t (ns)</th>
<th>A</th>
<th>B</th>
<th>F(I)</th>
<th>F</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>-∞</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>A=B=1 for a long time</td>
</tr>
<tr>
<td>0</td>
<td>1⇒0</td>
<td>1</td>
<td>1⇐0</td>
<td>0</td>
<td>F(I) changes to 1</td>
</tr>
<tr>
<td>0.5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1⇐0</td>
<td>F changes to 1 after a 0.5 ns delay</td>
</tr>
<tr>
<td>0.8</td>
<td>1⇐0</td>
<td>1</td>
<td>1⇒0</td>
<td>1</td>
<td>F(Instantaneous) changes to 0</td>
</tr>
<tr>
<td>0.13</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1⇒0</td>
<td>F changes to 0 after a 0.5 ns delay</td>
</tr>
</tbody>
</table>
Gate Delay Models

- Suppose gates with delay $n$ ns are represented for $n = 0.2$ ns, $n = 0.4$ ns, $n = 0.5$ ns, respectively:
Circuit Delay Model

- Consider a simple 2-input multiplexer:
- With function:
  - \( Y = A \) for \( S = 1 \)
  - \( Y = B \) for \( S = 0 \)

- “Glitch” is due to delay of inverter
Storing State

What if A connected to Y?

- With function:
  - $Y = B$ for $S = 1$, and $Y(t)$ dependent on $Y(t-0.9)$ for $S = 0$

- The simple combinational circuit has now become a sequential circuit because its output is a function of a time sequence of input signals!

Y is stored value in shaded area
Storing State (Continued)

- Simulation example as input signals change with time. Changes occur every 100 ns, so that the tenths of ns delays are negligible.

<table>
<thead>
<tr>
<th>Time</th>
<th>B</th>
<th>S</th>
<th>Y</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Y “remembers” 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Y = B when S = 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td>Now Y “remembers” B = 1 for S = 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td>No change in Y when B changes</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Y = B when S = 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Y “remembers” B = 0 for S = 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No change in Y when B changes</td>
</tr>
</tbody>
</table>

- Y represent the **state** of the circuit, not just an output.
Another example: create Sequential circuit from combinational circuit (nice QUIZ)

- Design a circuit: inputs S and R that are never simultaneously 1, and output:
  \[ f = 1 \text{ if } S \text{ was 1 more recently than } R \]
  \[ 0 \text{ else} \]
Another example: create Sequential circuit from combinational circuit

Solution? K-Map => how many variables?
Tip: 3 variables.
Unstable: in Storing State (Continued)

- Suppose we place an inverter in the “feedback path.”

- The following behavior results:

- The circuit is said to be **unstable**.

- For $S = 0$, the circuit has become what is called an **oscillator**. Can be used as crude clock.

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>Y</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>$Y = B$ when $S = 1$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Now $Y$ “remembers” $B$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Y$, 1.1 ns later</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>$Y$, 1.1 ns later</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>$Y$, 1.1 ns later</td>
</tr>
</tbody>
</table>
**Unstable:** other example

NOR with feedback

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1 -&gt; 0 -&gt; 1 -&gt; 0 (unstable, oscillator)</td>
</tr>
</tbody>
</table>
S – R Latch (NOR)

- Cross-coupling two NOR gates gives the S – R Latch:

- Which has the time sequence behavior:

<table>
<thead>
<tr>
<th>Time</th>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>(\overline{Q})</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>Stored state unknown</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>“Set” Q to 1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Now Q “remembers” 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>“Reset” Q to 0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Now Q “remembers” 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Both go low</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>Unstable!</td>
</tr>
</tbody>
</table>

Clocked S - R Latch

- Adding two NAND gates to the basic \( S - R \) NAND latch gives the clocked \( S - R \) latch:

- Has a time sequence behavior similar to the basic \( S - R \) latch except that the \( S \) and \( R \) inputs are only observed when the line \( C \) is high.

- \( C \) means “control” or “clock”.

**Clocked S - R Latch (continued)**

- The Clocked S-R Latch can be described by a table:

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q(t)</th>
<th>S R</th>
<th>Q(t+1)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Clear Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Set Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>???</td>
<td>Indeterminate</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>No change</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Clear Q</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Set Q</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>???</td>
<td>Indeterminate</td>
<td></td>
</tr>
</tbody>
</table>

- The table describes what happens after the clock [at time \((t+1)\)] based on:
  - current inputs \((S, R)\) and
  - current state \(Q(t)\).
D Latch

- Adding an inverter to the S-R Latch, gives the D Latch:
- Note: there are no “indeterminate” states!

<table>
<thead>
<tr>
<th>Q</th>
<th>D</th>
<th>Q(t+1)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No change</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Set Q</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Clear Q</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>No Change</td>
</tr>
</tbody>
</table>

The graphic symbol for a D Latch is:
Flip-Flops

- The latch timing problem
- Master-slave flip-flop
- Edge-triggered flip-flop
- Standard symbols for storage elements
- Direct inputs to flip-flops
The Latch Timing Problem

- In a sequential circuit, paths may exist through combinational logic:
  - From one storage element to another
  - From a storage element back to the same storage element
- The combinational logic between a latch output and a latch input may be as simple as an interconnect
- For a clocked D-latch, the output Q depends on the input D whenever the clock input C has value 1
The Latch Timing Problem (continued)

- Consider the following circuit:

  ![Latch Circuit Diagram]

- Suppose that initially $Y = 0$.

  Clock

  ![Y Loop Through Connection]

- As long as $C = 1$, the value of $Y$ continues to change!

- The changes are based on the delay present on the loop through the connection from $Y$ back to $Y$.

- This behavior is clearly unacceptable.

- Desired behavior: $Y$ changes only once per clock pulse
The Latch Timing Problem (continued)

- A solution to the latch timing problem is to **break** the closed path from Y to Y within the storage element.
- The commonly-used, path-breaking solutions replace the clocked D-latch with: 2 solutions:
  - a master-slave flip-flop
  - an edge-triggered flip-flop
S-R Master-Slave Flip-Flop

- Consists of two clocked S-R latches in series with the clock on the second latch inverted
- The input is observed by the first latch with $C = 1$
- The output is changed by the second latch with $C = 0$
- The path from input to output is broken by the difference in clocking values ($C = 1$ and $C = 0$).
- The behavior demonstrated by the example with $D$ driven by $Y$ given previously is prevented since the clock must change from 1 to 0 before a change in $Y$ based on $D$ can occur.
Flip-Flop Problem

- The change in the flip-flop output is delayed by the pulse width which makes the circuit slower or
- S and/or R are permitted to change while C = 1
  - Suppose Q = 0 and S goes to 1 and then back to 0 with R remaining at 0
    - The master latch sets to 1
    - A 1 is transferred to the slave
  - Suppose Q = 0 and S goes to 1 and back to 0 and R goes to 1 and back to 0
    - The master latch sets and then resets
    - A 0 is transferred to the slave
  - This behavior is called *Is catching*
Flip-Flop Solution => edge-triggered

- Use edge-triggering instead of master-slave
- An *edge-triggered* flip-flop ignores the pulse while it is at a constant level and triggers only during a transition of the clock signal
- Edge-triggered flip-flops can be built directly at the electronic circuit level, or
- A *master-slave* D flip-flop which also exhibits *edge-triggered behavior* can be used.
Edge-Triggered D Flip-Flop

- The edge-triggered D flip-flop is the same as the master-slave D flip-flop.

- It can be formed by:
  - Replacing the first clocked S-R latch with a clocked D latch or
  - Adding a D input and inverter to a master-slave S-R flip-flop.

- The delay of the S-R master-slave flip-flop can be avoided since the 1s-catching behavior is not present with D replacing S and R inputs.

- The change of the D flip-flop output is associated with the negative edge at the end of the pulse.

- It is called a negative-edge triggered flip-flop.
Positive-Edge Triggered D Flip-Flop

- Formed by adding inverter to clock input
  - \( Q \) changes to the value on \( D \) applied at the positive clock edge within timing constraints to be specified
  - Our choice as the standard flip-flop for most sequential circuits

Reference

Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides
© 2008 Pearson Education, Inc.
Review FF

Master-Slave FF

Positive-edge-triggered FF

Negative-edge-triggered FF

HOANG Trang

Reference

Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides
© 2008 Pearson Education, Inc.
Review FF (continued): quiz

Assume \( Q = 0 \) initially.

\[ \text{CLOCK} \]

\[ J \]

\[ K \]

pos-edge triggered \( Q \)

neg-edge triggered \( Q \)

master-slip \( Q \)
Standard Symbols for Storage Elements

- **Master-Slave:**
  - Postponed output indicators

- **Edge-Triggered:**
  - Dynamic indicator

(a) Latches

(b) Master-Slave Flip-Flops

(c) Edge-Triggered Flip-Flops
Sequential Circuit Analysis

- **General Model**
  - **Current State** at time (t) is stored in an array of flip-flops.
  - **Next State at time (t+1)** is a Boolean function of State and Inputs.
  - **Outputs at time (t)** are a Boolean function of State (t) and (sometimes) Inputs (t).

**Reference**

HOANG Trang

Logic and Computer Design Fundamentals, 4e
PowerPoint® Slides
© 2008 Pearson Education, Inc.
Example 1

- **Input:** \( x(t) \)
- **Output:** \( y(t) \)
- **State:** \( (A(t), B(t)) \)

**What is the **Output** Function?**

**What is the **Next State** Function?**
Example 1 (continued)

- Boolean equations for the functions:

\[ A(t+1) = A(t)x(t) + B(t) \]

\[ B(t+1) = A(t)x(t) \]

\[ y(t) = x(t)(B(t) + A(t)) \]
State Table

- **State table** – table with the following four sections:
  - *Present State* – the values of the state variables for each allowed state.
  - *Input* – the input combinations allowed.
  - *Next-state* – the value of the state at time \((t+1)\) based on the present state and the input.
  - *Output* – the value of the output as a function of the present state and (sometimes) the input.

- From the viewpoint of a truth table:
  - the **inputs**: Input, Present State
  - the **outputs**: Output, Next State
Example 1: State Table (from slide 35)

- By using the next state and output equations => State Table

\[
A(t+1) = A(t)x(t) + B(t)x(t)
\]

\[
B(t+1) = \bar{A}(t)x(t)
\]

\[
y(t) = x(t)(B(t) + A(t))
\]

<table>
<thead>
<tr>
<th>Present State</th>
<th>Input</th>
<th>Next State</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(t) B(t) x(t)</td>
<td>A(t+1) B(t+1) y(t)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 1: Alternate State Table

- 2-dimensional table that matches well to a K-map. Present state rows and input columns in Gray code order.
  - \( A(t+1) = A(t)x(t) + B(t)x(t) \)
  - \( B(t+1) = \overline{A(t)}x(t) \)
  - \( y(t) = \overline{x(t)}(B(t) + A(t)) \)

<table>
<thead>
<tr>
<th>Present State ( A(t) B(t) )</th>
<th>Next State ( x(t)=0 )</th>
<th>Next State ( x(t)=1 )</th>
<th>Output ( x(t)=0 )</th>
<th>Output ( x(t)=1 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 0 \ 0 )</td>
<td>( A(t+1)B(t+1) )</td>
<td>( A(t+1)B(t+1) )</td>
<td>( y(t) )</td>
<td>( y(t) )</td>
</tr>
<tr>
<td>( 0 \ 1 )</td>
<td>( 0 \ 0 )</td>
<td>( 1 \ 1 )</td>
<td>( 1 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( 1 \ 0 )</td>
<td>( 0 \ 0 )</td>
<td>( 1 \ 0 )</td>
<td>( 1 )</td>
<td>( 0 )</td>
</tr>
<tr>
<td>( 1 \ 1 )</td>
<td>( 0 \ 0 )</td>
<td>( 1 \ 0 )</td>
<td>( 1 )</td>
<td>( 0 )</td>
</tr>
</tbody>
</table>
State Diagrams

- The sequential circuit function can be represented in graphical form as a state diagram with the following components:
  - A circle with the state name in it for each state
  - A directed arc from the Present State to the Next State for each state transition
  - A label on each directed arc with the Input values which causes the state transition, and
  - A label:
    - On each circle with the output value produced, or
    - On each directed arc with the output value produced.
State Diagrams

- Label form:
  - On circle with output included:
    - state/output
    - Moore type output depends only on state
  - On directed arc with the output included:
    - input/output
    - Mealy type output depends on state and input
Example 1: State Diagram

- Which type?
- Diagram gets confusing for large circuits
- For small circuits, usually easier to understand than the state table
Equivalent State Definitions

- Two states are *equivalent*:
  + each possible input sequence $\Leftrightarrow$ an identical output sequence.
  + next states for each input symbol are the same or equivalent.
Equivalent State Example

- For states S3 and S2,
  - the output for input 0 is 1 and input 1 is 0, and
  - the next state for input 0 is S0 and for input 1 is S2.
  - By the alternative definition, states S3 and S2 are equivalent.
Equivalent State Example

- Replacing S3 and S2 by a single state gives state diagram:

- Examining the new diagram, states S1 and S2 are equivalent since
  - their outputs for input 0 is 1 and input 1 is 0, and
  - their next state for input 0 is S0 and for input 1 is S2,

- Replacing S1 and S2 by a single state gives state diagram:
Moore and Mealy Models

- Sequential Circuits or Sequential Machines are also called *Finite State Machines* (FSMs). Two formal models exist:
  - **Moore Model**
    - Named after E.F. Moore
    - Outputs are a function **ONLY of states**
    - Usually specified on the states.
  - **Mealy Model**
    - Named after G. Mealy
    - Outputs are a function of **inputs AND states**
    - Usually specified on the state transition arcs.
Moore and Mealy Example Diagrams

- Mealy Model State Diagram maps inputs and state to outputs.

- Moore Model State Diagram maps states to outputs.
Moore and Mealy Example Tables

- **Moore Model state table maps state to outputs**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State x=0</th>
<th>Next State x=1</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Mealy Model state table maps inputs and state to outputs**

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State x=0</th>
<th>Next State x=1</th>
<th>Output x=0</th>
<th>Output x=1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Mixed Moore and Mealy Outputs

- In real designs, some outputs may be Moore type and other outputs may be Mealy type.
- Example: Figure 5-17(a) can be modified to illustrate this
  - State 00: Moore
  - States 01, 10, and 11: Mealy
- Simplifies output specification
Conclusion: part 1: analysis???

+ Output function
+ Next-state func
+ How to present?
  . State table (Moore, Mealy)
  . State diagram (useful for small circuit)
+ Equivalent state (time: method to find out: 20-30 hours)