Sequential Circuits
Circuits with state

Silvina Hanono Wachman
Computer Science & Artificial Intelligence Lab
M.I.T.
Combinational circuits

Such circuits have no cycles (feedback) or state elements
Simple circuits with feedback, i.e., a cycle

- Circuits with cycles can hold state
- Generally behavior is difficult to analyze and requires paying attention to propagation delays

But how do we change its state?

This circuit can hold a 0 or 1

This circuit will oscillate between 0 and 1
D Latch: a famous circuit that can hold state

if C=0, the value of D passes to Q
if C=1, the value of Q holds

Let $Q^{t-1}$ represent the value previously held in DL; $Q^t$ represents the current value.
Building circuits with D latches

- If two latches are driven by the same C signal, they pass signals at the same time and hold signals at the same time.
- The composed latches look just like a single D latch (assuming signals aren’t changing too fast)
Building circuits with D latches
continued

- If latches are driven by inverted C signals, one is always holding, and one is always passing
- How does this circuit behave?
  - When $C = 0$, $Q$ holds its old value, but $Q^{\text{int}}$ follows the input $D$
  - When $C = 1$, $Q^{\text{int}}$ holds its old value, but $Q$ follows $Q^{\text{int}}$
  - $Q$ doesn’t change when $C = 0$ or $C = 1$, but it changes its value when $C$ transitions from 0 to 1 (a *rising-edge* of $C$)
Edge-Triggered D Flip flop
A basic storage element

Suppose C changes periodically (called a Clock signal)

Data is sampled at the rising edge of the clock and must be stable at that time
Clarification: D Flip-Flop Timing

- Flip-flop input D should not change around the rising edge of the clock to avoid metastability.
- Formally, D should be a stable and valid digital value:
  - For at least $t_{\text{SETUP}}$ before the rising edge of the clock.
  - For at least $t_{\text{HOLD}}$ after the rising edge of the clock.
- Flip-flop propagation delay $t_{\text{PD}}$ is measured from rising edge of the clock (CLK→Q).
Meeting the Setup-Time Constraint

- To meet FF2’s setup time,

\[ t_{CLK} \geq t_{PD,FF1} + t_{PD,CL} + t_{SETUP,FF2} \]

- The slowest register-to-register path in the system determines the clock \( \rightarrow \) limited amount of combinational logic between registers
Meeting the Hold-Time Constraint

- Propagation delay ($t_{PD}$): Upper bound on time from valid inputs to valid outputs
  - But upper bounds don’t help us analyze hold time...
- Contamination delay ($t_{CD}$): Lower bound on time from invalid inputs to invalid outputs
- To meet FF2’s hold-time constraint,

$$t_{CD,FF1} + t_{CD,CL} \geq t_{HOLD,FF2}$$

- Tools may need to add logic to fast paths to meet $t_{HOLD}$
D Flip-flop with Write Enable
The building block of Sequential Circuits

Data is captured only if EN is on

<table>
<thead>
<tr>
<th>EN</th>
<th>D</th>
<th>( Q^t )</th>
<th>( Q^{t+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

No need to show the clock explicitly
Clocked Sequential Circuits

- In this class we will deal with only clocked sequential circuits
- We will also assume that all flip flops are connected to the same clock
- To avoid clutter, the clock input will be implicit and not shown in diagrams
- Clock inputs are not needed in BSV descriptions unless we design multi-clock circuits
Registers

Register: A group of flip-flops with a common clock and enable

Register file: A group of registers with a common clock, a shared set of input and output ports
### An example

**Modulo-4 counter**

<table>
<thead>
<tr>
<th>Prev State</th>
<th>NextState inc = 0</th>
<th>NextState inc = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1q0</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>00</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>01</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>00</td>
</tr>
</tbody>
</table>

### Finite State Machine (FSM) representation

\[
\begin{align*}
q_0^{t+1} &= \neg\text{inc} \cdot q_0^t + \text{inc} \cdot \neg q_0^t \\
    &= \text{inc} \oplus q_0^t \\
q_1^{t+1} &= \neg\text{inc} \cdot q_1^t + \text{inc} \cdot \neg q_1^t \cdot q_0^t + \text{inc} \cdot q_1^t \cdot \neg q_0^t \\
    &= \neg\text{inc} \cdot q_1^t + \text{inc} \cdot (q_1^t \oplus q_0^t)
\end{align*}
\]
Circuit for the modulo counter using D flipflops

\[ q_0^{t+1} = \neg \text{inc} \cdot q_0^t + \text{inc} \cdot \neg q_0^t \]
\[ q_1^{t+1} = \neg \text{inc} \cdot q_1^t + \text{inc} \cdot (q_1^t \oplus q_0^t) \]

- We can use two D flip flops with enables to store \( q_0 \) and \( q_1 \)
- Notice, the state of the flip flop changes only when \( \text{inc} \) is 1. Thus, we can simplify the equations as shown

\[ \begin{align*}
q_0^{t+1} &= \neg q_0^t \\
q_1^{t+1} &= q_1^t \oplus q_0^t
\end{align*} \]
A method for computing GCD

Euclid’s algorithm for computing the Greatest Common Divisor (GCD):

<table>
<thead>
<tr>
<th>a: 15</th>
<th>b: 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

def gcd(a, b):
    if a == 0: return b  # stop
    elif a >= b: return gcd(a-b, b)  # subtract
    else: return gcd(b, a)  # swap

answer
Sequential Circuit for GCD

- Suppose we have two registers to hold the values a and b and combinational circuits to compute \((a \geq b), (a - b)\) and \((a \neq 0)\).
- We can define the next state computation as:

  \[
  \begin{align*}
  p &= (a^t \geq b^t) \\
  a^{t+1} &= p \cdot (a^t - b^t) + \neg p \cdot b^t \\
  b^{t+1} &= p \cdot b^t + \neg p \cdot a^t
  \end{align*}
  \]

  \[(a^t \neq 0)\]

- At every cycle \(t\), registers a and b should be enabled if \((a^t \neq 0)\)

```python
def gcd(a, b):
    if a == 0: return b  # stop
    elif a >= b: return gcd(a-b, b)  # subtract
    else: return gcd(b, a)  # swap
```
Generating the GCD circuit

- Registers with enable
- Connect the enable signals
- The data value for registers depends on $p$; introduce muxes
- Connect $p$
- Connect the next state values for $a$ and $b$

$p = (a^t >= b^t)$
$a^{t+1} = p \cdot (a^t - b^t) + \neg p \cdot b^t$
$b^{t+1} = p \cdot b^t + \neg p \cdot a^t$

\[\text{(a}^t \neq 0)\]

Notice
- $a$ and $b$ will be updated only if $a \neq 0$
- The circuit is incomplete – we need to define the start condition
- Much room for optimization: $b$ mux can be eliminated!
Finite State Machines (FSM) and Sequential Circuits

- FSMs are a mathematical object like the Boolean Algebra
  - A computer (in fact any digital hardware) is an FSM
- Synchronous Sequential Circuits is a method to implement FSMs in hardware
Large digital circuits, e.g., computers

- Large circuits need to be described as a collection of cooperating FSMs, i.e., as a collection of interconnected modules each of which represents a sequential circuit
- State diagrams and next-state tables are not suitable for describing such circuits

Next lecture – sequential machines as objects in an object-oriented language
Take-home problems

- Eliminate the mux for register b
- Introduce muxes to inject the initial values of a and b to start the GCD