Clocked Circuits
Additional Slides

Vishal Saxena
ECE, Boise State University

Nov 30, 2010
D Latch

- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D
D Latch

- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D
D Latch

- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D
D Latch

- When CLK = 1, latch is transparent
  - D flows through to Q like a buffer
- When CLK = 0, the latch is opaque
  - Q holds its old value independent of D
D Latch Design

- Multiplexer chooses D or old Q

![D Latch Circuit Diagram]
D Latch Operation

CLK = 1

D → Q
Q

CLK = 0

D → Q
Q

Vishal Saxena | CMOS Inverter
D Flip-flop

- When CLK rises, D is copied to Q
  - At all other times, Q holds its value
- a.k.a. positive edge-triggered flip-flop, master-slave flip-flop
D Flip-flop

- When CLK rises, D is copied to Q
  - At all other times, Q holds its value
- a.k.a. positive edge-triggered flip-flop, master-slave flip-flop
D Flip-flop

- When CLK rises, D is copied to Q
  - At all other times, Q holds its value
- a.k.a. positive edge-triggered flip-flop, master-slave flip-flop

![D Flip-flop Diagram]
D Flip-flop Design

- Built from master and slave D latches
D Flip-flop Operation

CLK = 1

CLK = 0

Vishal Saxena | CMOS Inverter
Latches and Flops  Sequencing  Sequencing Methods

DFF Example 1

\[ t_{\text{setup}} = t_{T_1} + t_{I_1} \]
\[ t_{\text{hold}} = -(t_{T_1} + t_{I_1}) \]
\[ t_{\text{pcq}} = t_{T_3} + t_{I_3} \]
DFF Example 1

- $t_{setup} = t_{T_1} + t_{I_1}$
- $t_{hold} = -(t_{T_1} + t_{I_1})$
- $t_{pcq} = t_{T_3} + t_{I_3}$
DFF Example 1

- \( t_{\text{setup}} = tT_1 + tI_1 \)
- \( t_{\text{hold}} = -(tT_1 + tI_1) \)
- \( t_{\text{pcq}} = tT_3 + tI_3 \)
DFF Example 1

- $t_{\text{setup}} = t_{T_1} + t_{I_1}$
- $t_{\text{hold}} = -(t_{T_1} + t_{I_1})$
- $t_{\text{pcq}} = t_{T_3} + t_{I_3}$
DFF Example 1

- \( t_{setup} = tT_1 + tI_1 \)
- \( t_{hold} = -(tT_1 + tI_1) \)
- \( t_{pcq} = tT_3 + tI_3 \)
DFF Example 2

- $t_{\text{setup}} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4}$
- $t_{\text{hold}} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3})$ \(\text{(Note that the}$ \overline{\text{CLK}} \text{ and}$ CLK \text{ have an overlap equal to an inverter delay (}$ t_{I_0} \text{)})\)
- Need to hold input \(D\), otherwise the master will latch new $D$ value.
- $t_{\text{PCQ}} = t_{T_3} + t_{I_6}$
DFF Example 2

\[ t_{\text{setup}} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4} \]

\[ t_{\text{hold}} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3}) \]

(Note that the \( \overline{CLK} \) and \( CLK \) have an overlap equal to an inverter delay \( t_{I_0} \))

- Need to hold input (D), otherwise the master will latch new D value.

\[ t_{\text{pcq}} = t_{T_3} + t_{I_6} \]
DFF Example 2

\[ t_{\text{setup}} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4} \]

\[ t_{\text{hold}} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3}) \]

(Note that the \( \overline{CLK} \) and \( CLK \) have an overlap equal to an inverter delay \( t_{I_0} \)).

- Need to hold input (D), otherwise the master will latch new D value.

\[ t_{\text{pcq}} = t_{T_3} + t_{I_6} \]
DFF Example 2

\[ t_{\text{setup}} = t_{I_1} + t_{T_1} + t_{I_3} + t_{I_4} \]

\[ t_{\text{hold}} = t_{I_0} - (t_{I_1} + t_{T_1} + t_{I_3}) \]

(Note that the \( \overline{CLK} \) and \( CLK \) have an overlap equal to an inverter delay \( t_{I_0} \))

- Need to hold input (D), otherwise the master will latch new D value.

\[ t_{\text{pcq}} = t_{T_3} + t_{I_6} \]
DFF Example 2

- \( t_{setup} = t_I + t_T + t_{I3} + t_{I4} \)
- \( t_{hold} = t_{I0} - (t_I + t_T + t_{I3}) \) (Note that the \( \overline{CLK} \) and \( CLK \) have an overlap equal to an inverter delay (\( t_{I0} \))
  - Need to hold input (D), otherwise the master will latch new D value.
- \( t_{pcq} = t_T + t_{I6} \)
Set / Reset

- Set forces output high when enabled
- Flip-flop with asynchronous set and reset
- Set forces output high when enabled
- Flip-flop with asynchronous set and reset
A dynamic latch

- sometimes called clocked CMOS ($C^2MOS$) latch
- second design is inferior as the input noise passed into the output when the latch is opaque
A dynamic latch
- sometimes called clocked CMOS ($C^2MOS$) latch
- second design is inferior as the input noise passed into the output when the latch is opaque

(a) 
(b) Bad
A dynamic latch

- sometimes called clocked CMOS ($C^2$MOS) latch
- second design is inferior as the input noise passed into the output when the latch is opaque
- **Combinational logic**
  - output depends on current inputs

- **Sequential logic**
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline
Sequencing

- **Combinational logic**
  - output depends on current inputs

- **Sequential logic**
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline

Finite State Machine

Pipeline
- **Combinational logic**
  - output depends on current inputs

- **Sequential logic**
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline

---

**Finite State Machine**

![Finite State Machine Diagram](image)

**Pipeline**

![Pipeline Diagram](image)
- **Combinational logic**
  - output depends on current inputs

- **Sequential logic**
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline

![Diagram of CL clk in out and Pipeline Finite State Machine](image-url)
Sequencing

- Combinational logic
  - output depends on current inputs

- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline

Finite State Machine

Pipeline
Sequencing

- **Combinational logic**
  - output depends on current inputs

- **Sequential logic**
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline

![Finite State Machine](image1)

![Pipeline](image2)
Sequencing

- Combinational logic
  - output depends on current inputs

- Sequential logic
  - output depends on current and previous inputs
  - Requires separating previous, current, future
  - Called state or tokens
  - Ex: FSM, pipeline
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary

- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses

- This is called wave pipelining in circuits
- In most circuits, dispersion is high
  - Delay fast tokens so they don’t catch slow ones.
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary.

Ex: fiber-optic cable

- Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses

This is called wave pipelining in circuits.

In most circuits, dispersion is high

- Delay fast tokens so they don’t catch slow ones.
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary

- Ex: fiber-optic cable
  - Light pulses (tokens) are sent down cable
  - Next pulse sent before first reaches end of cable
  - No need for hardware to separate pulses
  - But dispersion sets min time between pulses

- This is called wave pipelining in circuits

- In most circuits, dispersion is high
  - Delay fast tokens so they don’t catch slow ones.
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary

Ex: fiber-optic cable

- Light pulses (tokens) are sent down cable
- Next pulse sent before first reaches end of cable
- No need for hardware to separate pulses
- But dispersion sets min time between pulses

This is called wave pipelining in circuits

In most circuits, dispersion is high

- Delay fast tokens so they don’t catch slow ones
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary

Ex: fiber-optic cable

- Light pulses (tokens) are sent down cable
- Next pulse sent before first reaches end of cable
- No need for hardware to separate pulses
- But dispersion sets min time between pulses

This is called wave pipelining in circuits

In most circuits, dispersion is high

- Delay fast tokens so they don’t catch slow ones.
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary.

Ex: fiber-optic cable
- Light pulses (tokens) are sent down cable
- Next pulse sent before first reaches end of cable
- No need for hardware to separate pulses
- But dispersion sets min time between pulses

This is called wave pipelining in circuits.
- In most circuits, dispersion is high
  - Delay fast tokens so they don’t catch slow ones.
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary

Ex: fiber-optic cable

- Light pulses (tokens) are sent down cable
- Next pulse sent before first reaches end of cable
- No need for hardware to separate pulses
- But dispersion sets min time between pulses

This is called wave pipelining in circuits

In most circuits, dispersion is high

- Delay fast tokens so they don’t catch slow ones.
If tokens moved through pipeline at constant speed, no sequencing elements would be necessary

Ex: fiber-optic cable
- Light pulses (tokens) are sent down cable
- Next pulse sent before first reaches end of cable
- No need for hardware to separate pulses
- But dispersion sets min time between pulses

This is called wave pipelining in circuits

In most circuits, dispersion is high
- Delay fast tokens so they don’t catch slow ones.
Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.

- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence
- Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead
- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence
Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.

Inevitably adds some delay to the slow tokens

Makes circuit slower than just the logic delay

- Called sequencing overhead

Some people call this clocking overhead

- But it applies to asynchronous circuits too
- Inevitable side effect of maintaining sequence
Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.

Inevitably adds some delay to the slow tokens

Makes circuit slower than just the logic delay

- Called sequencing overhead

Some people call this clocking overhead

- But it applies to asynchronous circuits too
- Inevitable side effect of maintaining sequence
Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.

- Inevitably adds some delay to the slow tokens
- Makes circuit slower than just the logic delay
  - Called sequencing overhead

- Some people call this clocking overhead
  - But it applies to asynchronous circuits too
  - Inevitable side effect of maintaining sequence
Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.

Inevitably adds some delay to the slow tokens

Makes circuit slower than just the logic delay

Called sequencing overhead

Some people call this clocking overhead

But it applies to asynchronous circuits too

Inevitable side effect of maintaining sequence
Use flip-flops to delay fast tokens so they move through exactly one stage each cycle.
Inevitably adds some delay to the slow tokens
Makes circuit slower than just the logic delay
Called sequencing overhead
Some people call this clocking overhead
But it applies to asynchronous circuits too
Inevitable side effect of maintaining sequence
Sequencing Elements

- **Latch**: Level sensitive
  - a.k.a. transparent latch, D latch

- **Flip-flop**: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register
Sequencing Elements

- **Latch**: Level sensitive
  - a.k.a. transparent latch, D latch
- **Flip-flop**: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register
Latch: Level sensitive
  - a.k.a. transparent latch, D latch

Flip-flop: edge triggered
  - a.k.a. master-slave flip-flop, D flip-flop, D register
Latch: Level sensitive
- a.k.a. transparent latch, D latch

Flip-flop: edge triggered
- a.k.a. master-slave flip-flop, D flip-flop, D register
Timing Diagrams

- Transparent
- Opaque
- Edge-trigger
Timing Diagrams

- Transparent
- Opaque
- Edge-trigger

Timing diagrams for Latches and Flips:
Latches and Flops Sequencing Sequencing Methods

Timing Diagrams

- Transparent
- Opaque
- Edge-trigger

Vishal Saxena | CMOS Inverter
Timing Diagrams

- Transparent
- Opaque
- Edge-trigger
Timing Diagrams

- Transparent
- Opaque
- Edge-trigger
- **Flip-flops**
- **2-phase latches**
- **Pulsed Latches**

![Diagram showing flip-flops and combinational logic](image)

- **$T_c$**
- **clk**

---

Vishal Saxena | CMOS Inverter
- Flip-flops
- 2-phase latches
- Pulsed Latches
- Flip-flops
- 2-phase latches
- Pulsed Latches
- **Flip-flops**
- **2-phase latches**
- **Pulsed Latches**

The image illustrates the timing diagrams for flip-flops, 2-phase transparent latches, and pulsed latches, showing the propagation of signals through combinational logic blocks in each half-cycle.
### Timing Diagrams

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_{pd}$</td>
<td>Logic Prop. Delay</td>
</tr>
<tr>
<td>$t_{cd}$</td>
<td>Logic Cont. Delay</td>
</tr>
<tr>
<td>$t_{pcq}$</td>
<td>Latch/Flop Clk-&gt;Q Prop. Delay</td>
</tr>
<tr>
<td>$t_{ccq}$</td>
<td>Latch/Flop Clk-&gt;Q Cont. Delay</td>
</tr>
<tr>
<td>$t_{pdq}$</td>
<td>Latch D-&gt;Q Prop. Delay</td>
</tr>
<tr>
<td>$t_{cdq}$</td>
<td>Latch D-&gt;Q Cont. Delay</td>
</tr>
<tr>
<td>$t_{setup}$</td>
<td>Latch/Flop Setup Time</td>
</tr>
<tr>
<td>$t_{hold}$</td>
<td>Latch/Flop Hold Time</td>
</tr>
</tbody>
</table>

**Table:** Contamination and Propagation Delays

![Timing Diagrams](image)
- $t_{pd} \leq T_c - (\text{?})$
- $t_{pd} \leq T_c - (t_{\text{setup}} + t_{\text{pcq}})$
- $(t_{\text{setup}} + t_{\text{pcq}})$ is the sequencing overhead
- setup check
Max-Delay: Flip-Flops

- $t_{pd} \leq T_c - (?)$
- $t_{pd} \leq T_c - (t_{setup} + t_{pcq})$
- $(t_{setup} + t_{pcq})$ is the sequencing overhead
- setup check
Max-Delay: Flip-Flops

- \( t_{pd} \leq T_c - (?) \)
- \( t_{pd} \leq T_c - (t_{setup} + t_{pcq}) \)
- \((t_{setup} + t_{pcq})\) is the sequencing overhead
- setup check

\[ \]
\( t_{pd} \leq T_c - (?) \)
\( t_{pd} \leq T_c - (t_{setup} + t_{pcq}) \)
\( (t_{setup} + t_{pcq}) \) is the sequencing overhead
setup check
- $t_{cd} \geq \text{?}
- t_{cd} \geq t_{\text{hold}} - t_{\text{ccq}}$
- hold check
- $t_{cd} \geq ?$
- $t_{cd} \geq t_{hold} - t_{ccq}$
- hold check
- $t_{cd} \geq \text{?}$
- $t_{cd} \geq t_{\text{hold}} - t_{\text{ccq}}$
- hold check

Min-Delay: Flip-Flops