Chapter 3

CPE100: Digital Logic Design I

Section 1004: Dr. Morris
Sequential Logic Design
Chapter 3 :: Topics

- Introduction
- Latches and Flip-Flops
- Synchronous Logic Design
- Finite State Machines
- Timing of Sequential Logic
- Parallelism
• Previously, Combinational Logic design had outputs only depend on current value of inputs
• Outputs of sequential logic depend on current and prior input values – it has memory.
• Some definitions:
  • State: all the information about a circuit necessary to explain its future behavior
  • Latches and flip-flops: state elements that store one bit of state
  • Synchronous sequential circuits: combinational logic followed by a bank of flip-flops
Sequential Circuits

- Give sequence to events (i.e. a notion of time)
- Have memory (short-term)
- Use feedback from output to input to store information
  - Need to “remember” past output
State Elements

- The state of a circuit influences its future behavior
- State elements store state
  - Bistable circuit
  - SR Latch
  - D Latch
  - D Flip-flop
Bistable Circuit

- Fundamental building block of other state elements
- Two outputs: $Q$, $\bar{Q}$ (state)
- No inputs

Redrawn circuit to emphasize symmetry
Bistable Circuit Analysis

- Consider the two possible cases:
  - \( Q = 0 \):
    - then \( Q = 1, \bar{Q} = 0 \) (consistent)
Consider the two possible cases:

- $Q = 0$: then $\bar{Q} = 1$, $Q = 0$ (consistent)
- $Q = 1$: then $\bar{Q} = 0$, $Q = 1$ (consistent)

Stores 1 bit of state in the state variable, $Q$ (or $\bar{Q}$)

But there are **no inputs to control the state**
SR (Set/Reset) Latch

- SR Latch
  - S – set Q=1
  - R – reset Q=0
SR (Set/Reset) Latch

• SR Latch

• Consider the four possible cases:
  • $S = 1, \, R = 0$
  • $S = 0, \, R = 1$
  • $S = 0, \, R = 0$
  • $S = 1, \, R = 1$
SR Latch Analysis

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

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

- **$S = 1, R = 0$:**
  \[ \text{then } Q = 1 \text{ and } \bar{Q} = 0 \]

- **$S = 0, R = 1$:**
  \[ \text{then } Q = 0 \text{ and } \bar{Q} = 1 \]
SR Latch Analysis

- **$S = 1, R = 0$:**
  
  then $Q = 1$ and $\bar{Q} = 0$
  
  *Set the output*

- **$S = 0, R = 1$:**
  
  then $Q = 0$ and $\bar{Q} = 1$
  
  *Reset the output*
SR Latch Analysis

- $S = 0, R = 0$: then $Q = Q_{prev}$

\[ \text{\begin{tikzpicture}[scale=0.8]
    \node (N1) at (0,0) [circle,draw] {$Q$};
    \node (N2) at (-1,0) [circle,draw] {$Q$};
    \node (R) at (0,-1) [draw] {$R$};
    \node (S) at (-1,-1) [draw] {$S$};
    \draw (N1) -- (R);
    \draw (N2) -- (S);
    \node (N3) at (2,0) [circle,draw] {$Q$};
    \node (N4) at (1,0) [circle,draw] {$Q$};
    \node (R) at (2,-1) [draw] {$R$};
    \node (S) at (1,-1) [draw] {$S$};
    \draw (N3) -- (R);
    \draw (N4) -- (S);
\end{tikzpicture}} \]
SR Latch Analysis

- \( S = 0, R = 0: \)
  
  then \( Q = Q_{\text{prev}} \)

\[ \begin{align*}
  Q_{\text{prev}} &= 0 \\
  Q_{\text{prev}} &= 1
\end{align*} \]
SR Latch Analysis

- **$S = 0, R = 0$:**
  then $Q = Q_{prev}$

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

- **$S = 0, R = 0$:**
  
  then $Q = Q_{\text{prev}}$

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

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

- **$S = 1, R = 1$:**
  then $Q = 0, \bar{Q} = 0$
  Invalid State
  $\bar{Q} \neq \text{NOT } Q$
**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
    \[ S = 1, \quad R = 0, \quad Q = 1 \]
  - **Reset:** Make the output 0
    \[ S = 0, \quad R = 1, \quad Q = 0 \]
D Latch

- Two inputs: \( CLK, D \)
  - \( CLK \): controls \textit{when} the output changes
  - \( D \) (the data input): controls \textit{what} the output changes to

- Function
  - When \( CLK = 1 \),
    \( D \) passes through to \( Q \) (\textit{transparent})
  - When \( CLK = 0 \),
    \( Q \) holds its previous value (\textit{opaque})

- Avoids invalid case when \( Q \neq \text{NOT } \overline{Q} \)
D Latch Internal Circuit

\[ \begin{array}{c|c|c|c|c|c|c} 
CLK & D & \overline{D} & S & R & Q & \overline{Q} \\
0 & X & & & & & \\
1 & 0 & 1 & 0 & 1 & & \\
1 & 1 & 1 & 1 & 1 & & 
\end{array} \]
D Latch Internal Circuit

![D Latch Internal Circuit Diagram]

<table>
<thead>
<tr>
<th>CLK</th>
<th>D</th>
<th>$\overline{D}$</th>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>$\overline{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>$\overline{X}$</td>
<td>0</td>
<td>0</td>
<td>Q_{prev}</td>
<td>$\overline{Q}_{prev}$</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 Flip-Flop

- **Inputs:** $CLK, D$
- **Function**
  - Samples $D$ on rising edge of $CLK$
    - When $CLK$ rises from 0 to 1, $D$ passes through to $Q$
    - Otherwise, $Q$ holds its previous value
  - $Q$ changes only on rising edge of $CLK$
- **Called** *edge-triggered*
- **Activated on the clock edge**
• Two back-to-back latches (L1 and L2) controlled by complementary clocks
• When $CLK = 0$
  • L1 is transparent
  • L2 is opaque
    – $D$ passes through to N1
Two back-to-back latches (L1 and L2) controlled by complementary clocks

- When $CLK = 0$
  - L1 is transparent
  - L2 is opaque
  - $D$ passes through to N1

- When $CLK = 1$
  - L2 is transparent
  - L1 is opaque
  - N1 passes through to $Q$
D Flip-Flop Internal Circuit

• Two back-to-back latches (L1 and L2) controlled by complementary clocks

• When $CLK = 0$
  – L1 is transparent
  – L2 is opaque
  – $D$ passes through to N1

• When $CLK = 1$
  – L2 is transparent
  – L1 is opaque
  – N1 passes through to $Q$

• Thus, on the edge of the clock (when $CLK$ rises from 0 $\rightarrow$ 1)
  – $D$ passes through to $Q$
D Latch vs. D Flip-Flop

CLK
D Q
Q

CLK
D Q
Q

D Latch vs. D Flip-Flop

CLK
D
Q (latch)
Q (flop)
D Latch vs. D Flip-Flop

CLK
D
Q
Q

D
Q
Q

CLK
D
Q
Q

Q (latch)
Q (flop)
SR Latch
S = 1, R = 0: Q = 1
S = 0, R = 1: Q = 0

D Latch
CLK = 1: Q = D
CLK = 0: Q = Q_{prev}

D Flip-flop
CLK = 0 → 1: Q = D
Otherwise: Q = Q_{prev}
Registers

![Registers diagram](image-url)
Enabled Flip-Flops

- **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

![Symbol for an enabled flip-flop]
Enabled Flip-Flops

• **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

---

Inputs: $CLK, D, EN$

Function:

- $EN = 1$: $D$ passes through to $Q$ on the clock edge
- $EN = 0$: the flip-flop retains its previous state

**Internal Circuit**

**Symbol**
Resettable Flip-Flops

- **Inputs:** $CLK, D, Reset$
- **Function:**
  - $Reset = 1$: $Q$ is forced to 0
  - $Reset = 0$: flip-flop behaves as ordinary D flip-flop

**Symbols**

- $D$ and $Q$
- $r$ as reset input

Resettable Flip-Flops

- Two types:
  - Synchronous: resets at the clock edge only
  - Asynchronous: resets immediately when $Reset = 1$

- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop

- Synchronously resettable flip-flop?
Resettable Flip-Flops

- Two types:
  - **Synchronous**: resets at the clock edge only
  - **Asynchronous**: resets immediately when $Reset = 1$

- Asynchronously resettable flip-flop requires changing the internal circuitry of the flip-flop

- Synchronously resettable flip-flop?

\[
\begin{align*}
&D \quad \text{Reset} \\
\text{Internal} & \quad \text{Circuit} \\
&\downarrow \quad \text{CLK} \\
&D \quad Q \quad Q
\end{align*}
\]
Settable Flip-Flops

Inputs: $CLK$, $D$, $Set$

Function:
- $Set = 1$: $Q$ is set to 1
- $Set = 0$: the flip-flop behaves as ordinary D flip-flop

Symbols

\[\begin{array}{c}
\text{D} \\
\text{Q} \\
\text{Set}
\end{array}\]
• Registers inserted between combinational logic
• Registers contain state of the system
• State changes at clock edge: system synchronized to the clock
• **Rules** of synchronous sequential circuit composition:
  • Every circuit element is either a register or a combinational circuit
  • At least one circuit element is a register
  • All registers receive the same clock signal
  • Every cyclic path contains at least one register
• **Rules** of synchronous sequential circuit composition:
  • Every circuit element is either a register or a combinational circuit
  • At least one circuit element is a register
  • All registers receive the same clock signal
  • Every cyclic path contains at least one register

• **Two common synchronous sequential circuits**
  • Finite State Machines (FSMs)
  • Pipelines
Finite State Machine (FSM)

• Consists of:
  – State register
    • Stores current state
    • Loads next state at clock edge
  – Combinational logic
    • Computes the next state
    • Computes the outputs
Finite State Machines (FSMs)

- Next state determined by current state and inputs
- Two types of finite state machines differ in output logic:
  - **Moore FSM**: outputs depend only on current state
  - **Mealy FSM**: outputs depend on current state and inputs

Moore FSM

Mealy FSM
FSM Design Procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
• **Traffic light controller**
  – Traffic sensors: $T_A$, $T_B$ (TRUE when there’s traffic)
  – Lights: $L_A$, $L_B$
1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
• Inputs: $CLK$, $Reset$, $T_A$, $T_B$
• Outputs: $L_A$, $L_B$

Note: multiple bits for output
FSM Design Procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
• Moore FSM: outputs labeled in each state
• States: Circles
• Transitions: Arcs

Reset

S0

$L_A$: green
$L_B$: red

Academic Ave.  Bravado Blvd.
Dining Hall  Labs
Dorms  Fields
Dining Hall  Labs
Dorms  Fields
• **Moore FSM**: outputs labeled in each state
• **States**: Circles
• **Transitions**: Arrows

FSM State Transition Diagram

S0
- $L_A$: green
- $L_B$: red

S1
- $L_A$: yellow
- $L_B$: red

S2
- $L_A$: red
- $L_B$: green

S3
- $L_A$: red
- $L_B$: yellow

Reset

T_A

T_B
FSM Design Procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
### FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$T_A$</td>
<td>$T_B$</td>
</tr>
<tr>
<td>S0</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- **S0**: $T_A$: green, $T_B$: red
- **S1**: $T_A$: yellow, $T_B$: red
- **S2**: $T_A$: red, $T_B$: green
- **S3**: $T_A$: red, $T_B$: yellow

[Diagram showing state transitions]
## FSM State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>S1</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>S3</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>S0</td>
</tr>
<tr>
<td>S0</td>
<td>1</td>
<td>S0</td>
</tr>
<tr>
<td>S1</td>
<td>X</td>
<td>S2</td>
</tr>
<tr>
<td>S2</td>
<td>X</td>
<td>S3</td>
</tr>
<tr>
<td>S3</td>
<td>X</td>
<td>S0</td>
</tr>
</tbody>
</table>

- **TA**: green
- **TB**: red
- **TA**: yellow
- **TB**: red
- **TA**: red
- **TB**: green
FSM Design Procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
### FSM Encoded State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( S_0 )</td>
<td>( T_A )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

#### State Encoding

<table>
<thead>
<tr>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>00</td>
</tr>
<tr>
<td>S1</td>
<td>01</td>
</tr>
<tr>
<td>S2</td>
<td>10</td>
</tr>
<tr>
<td>S3</td>
<td>11</td>
</tr>
</tbody>
</table>

Two bits required for 4 states
### FSM Encoded State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Inputs</th>
<th>Next State</th>
<th>State</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$T_A$</td>
<td>$T_B$</td>
<td>$S'_1$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

Two bits required for 4 states

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S_1} \overline{S_0} T_A + S_1 \overline{S_0} T_B$$

State Encoding:

- S0: 00
- S1: 01
- S2: 10
- S3: 11

**Legend:**
- LA: green
- LB: red
- LA: yellow
- LB: red
- LA: red
- LB: yellow

**Diagram:**

- Reset
- $T_A$
- $T_B$
- S0
- S1
- S2
- S3
FSM Design Procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
## FSM Output Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Outputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>$S_0$</td>
<td>$L_{A1}$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

### Output Encoding

<table>
<thead>
<tr>
<th>Output</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>00</td>
</tr>
<tr>
<td>yellow</td>
<td>01</td>
</tr>
<tr>
<td>red</td>
<td>10</td>
</tr>
</tbody>
</table>

Two bits required for 3 outputs
### FSM Output Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Outputs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_1 )</td>
<td>( L_{A1} )</td>
<td>( L_{A0} )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Output Encoding

<table>
<thead>
<tr>
<th>Output</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>green</td>
<td>00</td>
</tr>
<tr>
<td>yellow</td>
<td>01</td>
</tr>
<tr>
<td>red</td>
<td>10</td>
</tr>
</tbody>
</table>

Two bits required for 3 outputs

\[
\begin{align*}
L_{A1} &= S_1 \\
L_{A0} &= \overline{S_1} S_0 \\
L_{B1} &= S_1 \\
L_{B0} &= S_1 S_0
\end{align*}
\]
FSM Design Procedure

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
### FSM Schematic: State Register

- **State Register**

- **Equations:**
  
  \[
  S'_1 = S_1 \oplus S_0 \\
  S'_0 = S_1 S_0 T_A + S_1 S_0 T_B \\
  L_{A1} = S_1 \\
  L_{A0} = \overline{S_1 S_0} \\
  L_{B1} = \overline{S_1} \\
  L_{B0} = S_1 S_0
  \]
FSM Schematic: Next State Logic

$S'_1 = S_1 \oplus S_0$

$S'_0 = \overline{S_1 S_0 T_A} + S_1 S_0 T_B$
FSM Schematic: Output Logic

\[
L_{A1} = S_1 \\
L_{A0} = S_1S_0 \\
L_{B1} = \overline{S_1} \\
L_{B0} = S_1S_0
\]
FSM Timing Diagram

CLK

Reset

TA

TB

S'_1:0

S_1:0

L_A1:0

L_B1:0

Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5 Cycle 6 Cycle 7 Cycle 8 Cycle 9 Cycle 10

Green (00) Red (10) Yellow (01)

S0 (00) S1 (01) S2 (10) S3 (11) S0 (00) S1 (01)

Yellow (01) Red (10) Green (00)

Green (00) Red (10) Yellow (01)

0 5 10 15 20 25 30 35 40 45

t (sec)
• **Binary** encoding:
  – i.e., for four states, 00, 01, 10, 11

• **One-hot** encoding
  – One state bit per state
  – Only one state bit HIGH at once
  – i.e., for 4 states, 0001, 0010, 0100, 1000
  – Requires more flip-flops
  – Often next state and output logic is simpler
1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
1. Design a circuit to detect 3 or more 1’s in a row in a bit stream

2. Vending machine: Release an item after receiving 15 cents
   • Single coin slot but tells if you put in dime or nickel
   • No change given

1. Identify inputs and outputs
2. Sketch state transition diagram
3. Write state transition table
4. Select state encodings
5. Rewrite state transition table with state encodings
6. Write output table
7. Write Boolean equations for next state and output logic
8. Sketch the circuit schematic
Timing

- Flip-flop samples $D$ at clock edge
- $D$ must be stable when sampled
  - Similar to a photograph, $D$ must be stable around clock edge
  - Moving right before or after shutter click results in blurry photo
- If not, metastability can occur
Input Timing Constraints

- **Setup time**: $t_{\text{setup}} = \text{time before clock edge data must be stable (i.e. not changing)}$
- **Hold time**: $t_{\text{hold}} = \text{time after clock edge data must be stable}$
- **Aperture time**: $t_a = \text{time around clock edge data must be stable} \ (t_a = t_{\text{setup}} + t_{\text{hold}})$

![Diagram](image)
Output Timing Constraints

- **Propagation delay**: \( t_{pcq} \) = time after clock edge that the output \( Q \) is guaranteed to be stable (i.e., to stop changing)

- **Contamination delay**: \( t_{ccq} \) = time after clock edge that \( Q \) might be unstable (i.e., start changing)
Dynamic Discipline

- Synchronous sequential circuit inputs must be stable during aperture (setup and hold) time around clock edge
  - Specifically, inputs must be stable:
    - at least $t_{\text{setup}}$ before the clock edge
    - at least until $t_{\text{hold}}$ after the clock edge

- Previously, static discipline:
  - With logically valid inputs, every circuit element must produce logically valid outputs
• The delay between registers has a minimum and maximum delay, dependent on the delays of the circuit elements.
Setup Time Constraint

- Depends on the maximum delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least $t_{\text{setup}}$ before clock edge

$$T_c \geq \ldots$$
Setup Time Constraint

- Depends on the maximum delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least $t_{\text{setup}}$ before clock edge

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

\[ t_{pd} \leq \]
Setup Time Constraint

- Depends on the maximum delay from register R1 through combinational logic to R2
- The input to register R2 must be stable at least $t_{\text{setup}}$ before clock edge

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

\[
t_{\text{pd}} \leq T_c - (t_{\text{pcq}} + t_{\text{setup}})
\]

$(t_{\text{pcq}} + t_{\text{setup}})$: sequencing overhead
Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2
- The input to register R1 must be stable for at least $t_{\text{hold}}$ after the clock edge
Hold Time Constraint

- Depends on the **minimum** delay from register R1 through the combinational logic to R2.
- The input to register R1 must be stable for at least $t_{\text{hold}}$ after the clock edge.

\[
t_{\text{hold}} < t_{ccq} + t_{cd}
\]

\[
t_{cd} >
\]
Hold Time Constraint

- Depends on the minimum delay from register R1 through the combinational logic to R2.
- The input to register R2 must be stable for at least $t_{\text{hold}}$ after the clock edge.

$t_{\text{hold}} < t_{\text{ccq}} + t_{\text{cd}}$

$t_{\text{cd}} > t_{\text{hold}} - t_{\text{ccq}}$
Timing Analysis

Timing Characteristics

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

Setup time constraint:

- $T_c \geq t_{pcq} + t_{pd} + t_{setup}$

Hold time constraint:

- $t_{ccq} + t_{cd} > t_{hold}$?

$t_{pd} =$

$t_{cd} =$

Setup time constraint:

$T_c \geq t_{pcq} + t_{pd} + t_{setup}$

Hold time constraint:

$t_{ccq} + t_{cd} > t_{hold}$?
### Timing Analysis

#### Timing Characteristics

- $t_{ccq} = 30$ ps
- $t_{pcq} = 50$ ps
- $t_{setup} = 60$ ps
- $t_{hold} = 70$ ps
- $t_{pd}$ per gate = 35 ps
- $t_{cd}$ per gate = 25 ps

#### Timing Analysis

- $t_{pd} = 3 \times 35$ ps = 105 ps
- $t_{cd} = 25$ ps

**Setup time constraint:**

$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

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

$$f_c = \frac{1}{T_c} = 4.65 \text{ GHz}$$

**Hold time constraint:**

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

$$ (30 + 25) \text{ ps} > 70 \text{ ps} \ ? \ \text{No!}$$
Timing Characteristics

\[
\begin{align*}
t_{ccq} & = 30 \text{ ps} \\
t_{pcq} & = 50 \text{ ps} \\
t_{\text{setup}} & = 60 \text{ ps} \\
t_{\text{hold}} & = 70 \text{ ps} \\
t_{pd} & = 35 \text{ ps} \\
t_{cd} & = 25 \text{ ps}
\end{align*}
\]

Setup time constraint:
\[
T_c \geq f_c
\]

Hold time constraint:
\[
t_{ccq} + t_{cd} > t_{\text{hold}} ?
\]
Timing Analysis

Add buffers to the short paths:

Timing Characteristics

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

Setup time constraint:
\[
T_c \geq t_{pcq} + t_{pd} + t_{setup}
\]
\[
T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}
\]

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

\( t_{pd} = 3 \times 35 \text{ ps} = 105 \text{ ps} \)

\( t_{cd} = 2 \times 25 \text{ ps} = 50 \text{ ps} \)

\( f_c = 1/T_c = 4.65 \text{ GHz} \)
Parallelism

- Two types of parallelism:
  - Spatial parallelism
    - duplicate hardware performs multiple tasks at once
  - Temporal parallelism
    - task is broken into multiple stages
    - also called pipelining
    - for example, an assembly line
Parallelism Definitions

- **Token**: Group of inputs processed to produce group of outputs
- **Latency**: Time for one token to pass from start to end
- **Throughput**: Number of tokens produced per unit time

Parallelism increases throughput
Parallelism Example

- Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
- 5 minutes to roll cookies
- 15 minutes to bake
- What is the latency and throughput without parallelism?
Parallelism Example

- Ben Bitdiddle bakes cookies to celebrate traffic light controller installation
- 5 minutes to roll cookies
- 15 minutes to bake
- What is the latency and throughput without parallelism?

\[
\text{Latency} = 5 + 15 = 20 \text{ minutes} = \frac{1}{3} \text{ hour}
\]
\[
\text{Throughput} = 1 \text{ tray/} \frac{1}{3} \text{ hour} = 3 \text{ trays/hour}
\]
• What is the latency and throughput if Ben uses parallelism?
  
  – **Spatial parallelism:** Ben asks Allysia P. Hacker to help, using her own oven
  
  – **Temporal parallelism:**
    
    • two stages: rolling and baking
    • He uses two trays
    • While first batch is baking, he rolls the second batch, etc.
Spatial Parallelism

Latency = ?
Throughput = ?
Latency = 5 + 15 = 20 minutes = \( \frac{1}{3} \) hour

Throughput = 2 trays/ \( \frac{1}{3} \) hour = 6 trays/hour
Latency = ?
Throughput = ?
Temporal Parallelism

Latency = 5 + 15 = 20 minutes = 1/3 hour

⇒ true latency = 30 minutes (with 10 minutes idle)

Throughput = 1 trays/ 1/4 hour = 4 trays/hour

Using both spatial and temporal techniques, the throughput would be 8 trays/hour
• Circuit with no parallelism
• Registers have:
  • Propagation clock-to-Q = 0.3 ns
  • Setup time = 0.2 ns
Parallelism Example

- Circuit with no parallelism
- Find critical path
- Compute minimum cycle time, latency, and throughput

![Diagram of sequential logic design with delay times](image-url)
Parallelism Example

• Circuit with parallelism: 2 stages
• Find critical path
• Compute minimum cycle time, latency, and throughput

![Diagram of a circuit with parallelism: 2 stages]
Parallelism Example

- Circuit with parallelism: 3 stages
- Find critical path
- Compute minimum cycle time, latency, and throughput

Sequential Logic Design