13. Lecture
- Combinational Logic gates
  - Multiplexers / De-Multiplexers
  - Decoders / Encoders
  - Comparators
  - Parity Checkers / Generators
  - Binary Adders / Subtractors
  - Integer Multipliers

- Outputs, "at any time", are determined by the input combination
- Static CMOS logic
- Conventional static CMOS logic
- Switching Transistors / transmission gate
- Dynamic CMOS logic
- Domino logic

Timing Diagram
- Describe the functionality of a logic circuit across time
- Represented by a waveform
- For combinational logic, Output is a function of inputs

Timing Diagram Example

Combinational vs. Sequential circuits
- Outputs, "at any time", are determined by the input combination
- When input changed, output changed immediately
- Real circuits is imperfect and have "propagation delay"
- A combinational circuit
  - Performs logic operations that can be specified by a set of Boolean expressions
  - Can be built hierarchically

Output= f(In)
Output = f(In, previous In )

Timing Diagram of an AND Gate
Output change can occur "at any Time" for Combinational logic

Timing Diagram Example
Multiplexers (Mux)

- Functionality: Selection of a particular input
- Route 1 of N inputs (A) to the output F
- Require \( \log_2 N \) selection bits (S)
- En(able) bit can disable the route and set F to 0

\[
\begin{align*}
F &= \overline{S_3S_2A_0} + \overline{S_2S_1A_1} + S_0\overline{S_2}A_2 + S_1S_0A_3 \\
F &= \text{En} \cdot (\overline{S_3S_2}A_0 + \overline{S_2S_1}A_1 + S_0\overline{S_2}A_2 + S_1S_0A_3)
\end{align*}
\]
4-to-1 Mux w/ Enable Logic

\[ F = \text{EnS0}A_0 + \text{EnS0}A_1 + \text{EnS0}A_2 + \text{EnS0}A_3 \]

Reduce one Gate Delay by using 4-input AND gate for the 2nd level.

4-to-1 Mux using Transmission Gates

4-to-1 Mux using Transmission Gates

4-to-1 Mux using Transmission Gates

4-to-1 Mux using Transmission Gates

4-to-1 Mux using Transmission Gates
4-to-1 Mux using Transmission Gates with Enable (F=0 when En=0)

Demultiplexers (DeMux)

DeMux Operations

\[
\begin{array}{cccccc}
S_1 & S_0 & D_3 & D_2 & D_1 & D_0 \\
0 & 0 & 0 & 0 & 0 & A \\
0 & 1 & 0 & 0 & A & 0 \\
1 & 0 & A & 0 & 0 & 0 \\
1 & 1 & A & 0 & 0 & 0 \\
\end{array}
\]

\[
D_0 = S_1S_0A
\]

\[
D_1 = \overline{S_1}S_0A
\]

\[
D_2 = S_1\overline{S_0}A
\]

\[
D_3 = \overline{S_1}\overline{S_0}A
\]

DeMux Operations w/ Enable

\[
\begin{array}{cccccc}
S_1 & S_0 & D_1 & D_0 & D_2 & D_3 \\
0 & X & X & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & A \\
1 & 0 & 1 & 0 & A & 0 \\
1 & 1 & 0 & A & 0 & 0 \\
1 & 1 & 1 & A & 0 & 0 \\
\end{array}
\]

Decoders
1-to-2-Line Decoder

\[
\begin{array}{c|cc}
A & D1 & D0 \\
0 & 0 & 1 \\
1 & 1 & 0 \\
\end{array}
\]

\[
D_0 = \overline{A}
\]

\[
D_1 = A
\]

2-to-4-Line Decoder

\[
\begin{array}{c|ccccc}
A1 & A0 & D3 & D2 & D1 & D0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 \\
\end{array}
\]

\[
D_0 = A_1\overline{A}_0
\]

\[
D_1 = \overline{A}_1A_0
\]

\[
D_2 = A_1\overline{A}_0
\]

\[
D_3 = A_1A_0
\]

How about if no one should be enabled?

2-to-4-Line Decoder w/ Enable

\[
\begin{array}{c|ccccc}
En & A1 & A0 & D3 & D2 & D1 & D0 \\
0 & 0 & X & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]

\[
D_0 = En\overline{A}_1\overline{A}_0
\]

\[
D_1 = EnA_1\overline{A}_0
\]

\[
D_2 = En\overline{A}_1A_0
\]

\[
D_3 = EnA_1A_0
\]

3-to-8-Line Decoder

Truth Table

\[
\begin{array}{cccccccccccc}
A2 & A1 & A0 & D7 & D6 & D5 & D4 & D3 & D2 & D1 & D0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]
Another kind of decoder

BCD-to-7-Segment Decoder

Decode “2” and show 2

Decode “4” and show 4
### BCD-to-7-Seg. Decoder Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Since \( D_x = 1 \) only in one column at a time

\[ A_0 = D_1 + D_3 + D_5 + D_7 \]
\[ A_1 = D_2 + D_3 + D_6 + D_7 \]
\[ A_2 = D_4 + D_5 + D_6 + D_7 \]

---

### M-to-N-Line Encoder (\( M \leq 2^N \))

![M-to-N-Line Encoder Diagram](image)

### 8-to-3 Encoder

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Since \( D_x = 1 \) only in one column at a time

\[ A_0 = D_1 + D_3 + D_5 + D_7 \]
\[ A_1 = D_2 + D_3 + D_6 + D_7 \]
\[ A_2 = D_4 + D_5 + D_6 + D_7 \]

### Example 1 of an Encoder

![Example 1 of an Encoder](image)

**Example:** Wind direction encoder

- 0: North (N)
- 1: East (E)
- 2: South (S)
- 3: West (W)

### Adder

**Half Adder (1-bit)**

![Half Adder Diagram](image)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### Full Adder

<table>
<thead>
<tr>
<th>Cin</th>
<th>A</th>
<th>B</th>
<th>S (Sum)</th>
<th>Cout</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</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>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ S = \overline{A}B + \overline{B}A = A \oplus B \]

\[ C = AB \]

### 4-bit Ripple Adder using Full Adder

\[ S = \text{Cin} \oplus A \oplus B \]

\[ \text{Cout} = AB + \text{Cin}(A \oplus B) \]
Full Adder Propagation Delay

1st Stage Critical Path = 3 gate delays = \( D_{XOR} + D_{AND} + D_{OR} \).

2nd Stage Critical Path = 2 gate delays = \( D_{AND} + D\).

Since 1st Critical path > D,

Critical Path Delay ≈ 2(\(N-1\)) + 3 = (2N+1) Gate delays for 4-bit ripple adder (9 gate levels)

Issue of 4-bit Ripple Adder

Carry propagation is the main issue in an N-bit ripple adder.

A faster adder needs to address the serial propagation of the carry bit.

Let’s re-examine the equation for full adders.

Subtractor Design

\[ A - B = A + (-B) \]

Take 2’s complement of B

Perform addition of A and 2’s complement of B

Shift Register
Basic Shifting

- Shift directions
  - Left (multiply by 2)
  - Right (divide by 2)
    - Take floor value if the result is not an integer
    - Floor value of $X$ (or $|X|$) is the greatest integer number less than or equal to $X$, e.g.
      - $5/2 = 2$
      - $-3/2 = -2$

- Shift types
  - Logical (or unsigned)
  - Arithmetic (or signed)

Logical Shift

- Shift Left
  - MSB: Shifted out
  - LSB: Shifted in with a "0"
  - Examples:
    - $11001011 << 1 = 10010110$
    - $11001011 << 3 = 01011000$

- Shift right
  - MSB: Shifted in with a "0"
  - LSB: Shifted out
  - Examples: (Some ISA use triple ">>" for logical right shift)
    - $(11001011 >>> 1) = 01100101$
    - $(11001011 >>> 3) = 00011001$
    - $(11001011 >>> 5) = 00000000$

Logical (or unsigned) Right Shift
- $X$, $X$, $X$

Examples:
- $(11001011 >>> 1) = 01100101$
- $(11001011 >>> 3) = 00011001$
- $(11001011 >>> 5) = 00000000$

Logical Shift

4-bit Logical Shifter

![4-bit Logical Shifter Diagram]

$D_1 = S_1 A_1 + S_0 S_1 A_2$
$D_2 = S_1 A_1 + S_0 S_2 A_3$
$D_3 = S_1 A_1 + S_0 S_3 A_4$
$D_4 = S_1 A_1 + S_0 S_4 A_5$

4-bit Logical Shifter using 4-to-1 Mux

![4-bit Logical Shifter using 4-to-1 Mux Diagram]
4-bit Logical Shifter using 4-to-1 Mux

Sequential Logic Circuits

State machine example

Sequential Logic Circuits

Closed-Loop Logic for Storing Information

SR Latch
D Latch — Writing Data $D$

10T D Latch w/ Transmission Gates

10T D Latch w/ Transmission Gates

10T D Latch w/ Transmission Gates

D Latch Symbol

Latch is Transparent

- D Latch is called “transparent” or “level sensitive”
- Output follows input instantaneously

<table>
<thead>
<tr>
<th>$En$</th>
<th>$D$</th>
<th>$Q$</th>
<th>$\overline{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

NC: No Change
Transparency Property

- Transparent Latch
- Latch acts like a Wire

Problem of Transparency

- A momentary input change tunnels through the latch and the entire circuitry
- What problem this can cause?

Problem of Transparency

- Oscillating
- Unstable

Eliminating Transparency

- Separating the input and output, so they are independently controlled
- Only open one gate at a time to avoid tunneling

Behavior of Master-Slave Latches

- A Toggle Cell, will discuss more later
**Behavior of Master-Slave Latches**

- **Enable (En)**
- **D1 (input)**
- **Q1 = D2**
- **Q2**

**Flip-Flop (F/F)**

- **Input**
- **D1 Q1 D2 Q2**
- **Enable (or clock)**

**Negative Edge Triggered Flip-Flop**

- **Clock**
- **Input**
- **Q1 = D2**
- **Output**

**Positive Edge Triggered Flip-Flop**

- **Clock**
- **Input**
- **Q1 = D2**
- **Output**
**Flip Flops Symbols**

- Positive Edge Triggered D Flip Flop
- Negative Edge Triggered D Flip Flop

**Dual-phase Non-overlapped Clocks**

- In reality, enable control is not ideal
- Use dual phase clocks ( \( \phi_1 \) and \( \phi_2 \)) to replace Enable and its inversion

**Dual-Phase Non-overlapped Clocks**

- Input
- Output

**Registers**

- Input
- Output

**4-bit Register**

- Register is the most fundamental storage, e.g.
  - x86 ISA has 8 general purpose registers
  - MIPS ISA has 32 general purpose registers
- Each 1-bit Flip-flop is a single bit register
- Cascade 4 of 1-bit FFs = A 4-bit Register

**Read/Write Control a Register**

- Read: Retrieve data stored inside a flip-flop
- Write: Update with a new input data into a flip-flop
- Given \( \phi_1 \) and \( \phi_2 \) are continuous clock signals

**Dual-phase Non-overlapped Clocks**

- Input
- Output
Read/Write Control a Register

Another Read/Write Control of a Register

4-bit Register with Parallel Load

Logical Shift Register

Arithmetic Shift Register

Bidirectional Shift Register with Load (1-bit shown)
Design a Serial Adder (yet another adder)

Ex: 0111 (A) + 0011 (B)
(1) Clear SRs
(2) B = 0111 (4 clks)
(3) B=0111 A=0000
(4) B=1011 A=1000
(5) B=1101 A=1100
(6) B=0110 A=1110
(7) B=0011 A=0111
(8) B=0001 A=0011
(9) B=0000 A=1001
(10) B=0000 A=0100
(11) B=0000 A=1010

$A \leftarrow A+B$