Multiplexers can be used to implement logic functions:

Multiplexer: 

\[ f = E \left( I_0 \bar{s}_1 s_0 + I_1 \bar{s}_2 s_1 s_0 + I_2 \bar{s}_2 s_1 s_0 + I_3 \bar{s}_2 s_1 s_0 \right) \]

Logic function: 

\[ f = \sum m(0, 2, 3, 5) = f(x, y, z) \] 3-variable function

How to implement \( f \) using a 8-to-1 MUX?

- \( x \rightarrow s_2 \)
- \( y \rightarrow s_1 \)
- \( z \rightarrow s_0 \)

Minterms:

\[
\begin{align*}
& m_0 \rightarrow I_0 \\
& m_1 \rightarrow I_1 \\
& \vdots \\
& m_3 \rightarrow I_2 \\
& m_5 \rightarrow I_6 \\
& m_7 \rightarrow I_7
\end{align*}
\]

Sometimes it is also possible to implement a 3-variable function using only a 4-to-1 MUX: requires some math simplification:

\[
f(x, y, z) = \bar{x} \bar{y} \bar{z} + \bar{x} y \bar{z} + \bar{x} y z + x \bar{y} z
\]

\[
= \bar{x} \bar{y} z + \bar{x} y (\bar{z} + z) + x \bar{y} z + x \bar{y} 0
\]

\[
= \bar{x} \bar{y} z + x \bar{y} z + \bar{x} y + \bar{x} \bar{y} z
\]
<table>
<thead>
<tr>
<th>m0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>m2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>m3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>m4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>m6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>m7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ f(x,y,z) = \sum m(0, 2, 3, 5) \]

This same function can be written as:

![4-to-1 MUX diagram]
Programmable Logic Devices (PLD's)

PLD's are combinations of AND-arrays & OR-arrays

PROM (Programmable Read Only Memory)  AND's are fixed & OR's programmable
PLA (Programmable Logic Array)     AND's & OR's are programmable
PAL (Programmable Array Logic)       AND's are programmable & OR's fixed

Programmable:  
field programmable (done by customer)
mask programmable (done by manufacturer)

One way to program your connections is to blow fuses of unwanted connections:

```
  a
  b
  c
  d
```

```
  a
  b
  c
  d
```

Fuses on a & b are blown = open
Connections for AND gate are 1's.
Open connections for OR gate are 0's.

Simplified Notations

```
  a
  b
  c
  d
```

```
  a
  b
  c
```

```
  a
  b
  c
```

```
  a
  b
  c
```

```
  a
  b
  c
```

```
  a
  b
  c
```

```
  a
  b
  c
```

```
  a
  b
  c
```
PROM: (AND's fixed & OR's programmable)

Decoders (AND gate versions)
(are minterm generator)

Inputs: use buffer (no change) / inverter (complemented)

\[
\begin{array}{c}
\text{AND} \\
\text{OR}
\end{array}
\]

For \( n \) 's, there are \( 2^n \) minterms

\( 2^n \) (n inputs) \( \times \) m PROM.

Programmable OR's.
PLA (AND's & OR's are programmable)

\[ n \times p \times m \] PLA

PAL = (AND's programmable & OR's fixed)
PLA for combinational logic design:

\[
\begin{align*}
    f_1(x, y, z) &= \Sigma m(0,1,3,4) \\
    f_2(x, y, z) &= \Sigma m(1,2,3,4,5)
\end{align*}
\]

Use a $3 \times 4 \times 2$ PLA

Discussion: 2 OR gates sufficient for the 2 outputs $f_1$ & $f_2$. 3 inputs sufficient for $x, y, z$.

4 AND gates are not sufficient for the 6 minterms $f_1$ & $f_2$ -> we need simplification using Karnaugh maps or Prime Implicant Table.

Karnaugh Maps:

<table>
<thead>
<tr>
<th>$xyz$</th>
<th>$f_1$</th>
<th>$f_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>011</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>110</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>111</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

6 groups of ones: 4, 2, 2, 1.
3 groups of $2$

Prime Implicants: $\overline{y}z, \overline{x}y, \overline{x}z$

Essential: $\overline{y}z \land \overline{x}z$

$f_1(x, y, z) = \overline{y}z + \overline{x}z$

4 distinctive products can now implement using the $3 \times 4 \times 2$ PLA.
3x4x2 PLA

Diagram of a 3x4x2 PLA with inputs x, y, z and outputs f1, f2.
Ch 6 Flip-Flops & Applications

Non-sequential Network (so far): output of a function depends on a set of inputs x, y, z, etc. at same time instant.

Sequential Network:
- output of a function also depends on previous time instants (inputs). Network has memory.
- Two types: synchronous (internal clock set update Q discrete time instants)
- asynchronous
- Memory will be provided by the flip-flop.

Bistable element: (central to all flip-flop circuits)

<table>
<thead>
<tr>
<th>State</th>
<th>$x = \overline{y} = Q$</th>
<th>$y = \overline{x} = Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Metastable state
(should be avoided)

Observation:
If $x = 0 \rightarrow Q = 1$ and will stay in this state until some voltage is introduced to "clear the setting" or to "clear memory" or "resetting"
SR Latch (Set/Reset Latch):

- NOR gate configuration

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q</th>
<th>\overline{Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0/1</td>
<td>1/0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Bistable Element:

For \( Q = 0 \), so \( Q = 1 \) (unpredictable)
Application of SR Latch:

**Contact bounce:**

\[ V_A = 0 \quad \text{and} \quad V_B = V \]

When we switch from B to A: \( V_B \) drops to 0 while \( V_A \) is still @ 0. Then \( V_A \) will race to V, but since the contacts bounce a few times before it stays, \( V_A \) oscillates between 0 & V a few times (switches) until it stays @ V. Same when we switch back to B.

This is a problem: e.g., key on a keyboard: when we press once, it may interpret as 2 or more pressings.

**Solution:** connect an SR latch after the switch:

- When not connected A or B will stay @ 0. In the figure \( V_A = 0 \) & \( V_B = V \) (S=0 & R=1).
- Switch from B to A \( \rightarrow (S=1 & R=0) \)
  \[ \rightarrow (Q=1; \bar{Q}=0) \]
- If switch bounce away from A: \( (S=0 & R=0) \) bistable output \( \rightarrow \) output state is not changed: \( (Q=1; \bar{Q}=0) \)
  \[ \rightarrow \text{bounce effect is eliminated} \]
SR Latch using two NAND's

\[ \begin{align*}
R & \quad S & \quad \bar{R} & \quad \bar{S} & \quad Q & \quad \bar{Q} \\
0 & \quad 0 & \quad 1 & \quad 1 & \quad 0 & \quad 1 & \text{Bistable element.}
\end{align*} \]

- \( S = 1 \): if \( \bar{Q} = 0 \) \( \rightarrow \) \( Q = 1 \) \& \( \bar{R} = 0 \) \( \rightarrow \) \( \bar{Q} = 1 \) \( \Rightarrow \) \( S = 1 \) \( \Rightarrow \) \( Q = 1 \) \& \( \bar{R} = 0 \) \( \rightarrow \) \( Q = 0 \) \& \( R = 0 \) \( \Rightarrow \) \( \bar{Q} = 1 \) \( \checkmark \)
- \( S = 0 \): if \( \bar{Q} = 1 \) \( \rightarrow \) \( Q = 1 \) \& \( R = 1 \) \( \rightarrow \) \( Q = 0 \) \& \( R = 1 \) \( \Rightarrow \) \( S = 0 \) \( \Rightarrow \) \( Q = 0 \) \& \( R = 1 \) \( \rightarrow \) \( Q = 1 \) \& \( \bar{R} = 1 \) \( \Rightarrow \) \( \bar{Q} = 0 \) \( \checkmark \)

\( x \quad y \quad \text{AND} \quad \text{NAND} \)
\|  \|  \|  \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 \\
1 & 1 & 1 & 0
Gated SR Latch: to add an enable/disable ability:

- 2 additional NANDs to a SR latch:

\[
\begin{array}{c}
S \\
| \hspace{1cm} | \\
A \\
| \hspace{1cm} | \\
B \\
| \hspace{1cm} | \\
Q \\
\end{array}
\]

As long as the Enable input is 0, the outputs of A & B are 1's.

- From the SR Truth Table, we get a bistable element.
  \[ Q & \bar{Q} \text{ are not changed.} \]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>S R C</td>
<td>Q \bar{Q}</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 \bar{Q}</td>
</tr>
</tbody>
</table>

Gated D Latch: eliminate the inputs leading to unpredictable outputs in the previous 3 latches.

- The state evolved from the gated SR latch.

\[
\begin{array}{c}
D \\
| \hspace{1cm} | \\
A \\
| \hspace{1cm} | \\
B \\
| \hspace{1cm} | \\
Q \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>D C</td>
<td>Q \bar{Q}</td>
</tr>
<tr>
<td>0 0</td>
<td>0 \bar{Q}</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1</td>
</tr>
</tbody>
</table>
Propagation delay in flip-flops:

SR latch:

\[ S \quad Q \quad \overline{Q} \quad R \]

Delay times: \( t_{PLH} \) (Low→High), \( t_{PHL} \)

Most of the time: omit finite slopes, all delay times are equal.
Master-Slave Flip Flops (pulse-triggered flip-flop)

SR flip-flops: when we reset, information affects the output right away (except for delays) → these flip-flops are called "transparent." In a Master-Slave flip-flop: a pulse at the Master's clock controls when the information is propagated to the outputs.

\[ S \rightarrow Q_m \rightarrow Q \rightarrow Q_s \rightarrow Q \]

C = Enable input:
When Master is enabled, Slave is disabled & vice versa.

Master enabled \( \Rightarrow \) Master disabled
Slave enabled \( \Rightarrow \) Slave disabled
M8S 5R Flip-Flop

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>C</th>
<th>Q</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Symbol for a M8S flip-flop.

Right edge of clock pulse is when the input information get propagated to the output.

Other type: M8S JK flip-flop

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>C</th>
<th>Q</th>
<th>Q</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Q</td>
<td>Q</td>
</tr>
</tbody>
</table>
M8.5 D Flip-Flop:

M8.5 T Flip-Flop:

additional inverter b/w J & K in the M8.5 SR FF

J & K in the M8.5 JK FF are connected → T:

\[ \text{Diagram of T Flip-Flop} \]