**Tri-state gates**

- The third value
  - logic values: "0", "1"
  - don't care: "X" (must be 0 or 1 in real circuit!)
  - third value or state: "Z" — high impedance, infinite R, no connection

- Tri-state gates
  - additional input – output enable (OE)
  - output values are 0, 1, and Z
  - when OE is high, the gate functions normally
  - when OE is low, the gate is disconnected from wire at output
  - allows more than one gate to be connected to the same output wire
    - as long as only one has its output enabled at any one time (otherwise, sparks could fly)

```
<table>
<thead>
<tr>
<th>In</th>
<th>OE</th>
<th>Out</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>tri-state</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>buffer</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

**Tri-state and multiplexing**

- When using tri-state logic
  - (1) make sure never more than one "driver" for a wire at any one time (pulling high and low at the same time can severely damage circuits)
  - (2) make sure to only use value on wire when its being driven (using a floating value may cause failures)

- Using tri-state gates to implement an economical multiplexer

```
when Select is high
Input1 is connected to F

when Select is low
Input0 is connected to F

this is essentially a 2:1 mux
```
More complex counter example

- Complex counter
  - repeats 5 states in sequence
  - not a binary number representation

- Step 1: derive the state transition diagram
  - count sequence: 000, 010, 011, 101, 110

- Step 2: derive the state transition table from the state transition diagram

```
<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
<th>C+</th>
<th>B+</th>
<th>A+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 1 1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 0 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1 0 1</td>
<td>1 1 0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 0 0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

note the don't care conditions that arise from the unused state codes

More complex counter example (cont’d)

- Step 3: K-maps for next state functions

```
\[
\begin{array}{c|c|c|c|c}
A & B & C & C+ & B+ \\
0 & 0 & 0 & x & x \\
0 & 1 & x & 1 & x \\
1 & 0 & x & 1 & x \\
1 & 1 & x & 1 & x \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c}
A & B & C & A+ & C \\
0 & 0 & 0 & x & x \\
0 & 1 & x & 1 & x \\
1 & 0 & x & 1 & x \\
1 & 1 & x & 1 & x \\
\end{array}
\]

C+ := A

B+ := B' + A'C'

A+ := BC'
```
Comparison

- T FF: 5 gates, 10 literals, 15 wires
- RS FF: 3, 5, 12
- JK FF: 2, 4, 9
- D FF: 3, 5, 9

Summary:
- JK FF are most gate and literal efficient
- T FF are good mainly for straightforward counters
- D FF: Don't need the next state mapping stage. And **less wiring complexity**
- JK is **always** better than RS