Lab 08: The Voting Machine

Arturo Salinas-Aguayo
CSE 2301: Principles and Practice of Digital Logic Design
Dr. Mohammad Khan, Section 003L-1248
Electrical and Computer Engineering Department

College of Engineering, University of Connecticut
Coded in LaTeX

Theory

What is a Multiplexor?

A Multiplexor allows the designer to choose from several possible inputs based on the value of a *select* signal. The output is selected by the select signal. For example, for a 4:1 MUX has four data inputs and one output.

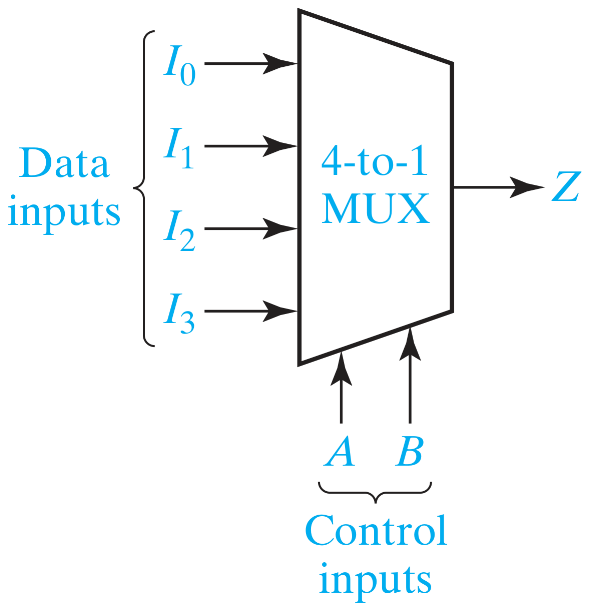


Figure 1: 4:1 MUX

Essentially, these IC's act as switches which select one of the inputs that is connected to them. In general,
n control inputs can be used to select one of 2^n data inputs.

$$Z = \sum_{k=0}^{2^n-1} m_k \cdot I_k$$

Where m_k is the k-th minterm of the select signals and I_k is the respective data input.

The select lines can also be tied to a clock counter sort of signal in order to cycle through the inputs. This is useful in analog circuits where a waveform can be "compressed" into many cycles and decoded such that input 1 has a certain frequency and amplitude and input 2 has another. This can introduce a lot of noise however and makes it difficult to troubleshoot.

Why just half of a 74153?

The lab has us design a circuit to implement two MUX's as a lookup table. In this orientation, because each input MUX has two select states, and we only utilize one output per MUX, we only need one of the two MUX's in the 74153.

Functionally, we are essentially turning each pair of MUX's into a 4 bit selector.

Since we are using Shannon's expansion to convert the inputs to a function that depends on the last input, each three person voting machine has select lines assigned to two people, W and X in this case, and Y is used to be the input to the mux as needed. By doing this, the inputs are limited to $1, 0, Y$ or $\overline{Y}$ and the output is a function of Y.

Discussion

Example 1 A Three Person Voting Machine

1. **Derive the Truth Table from the Inputs:**
   Given the inputs, $W(S_1)$ $X(S_0)$ and Y,a standard truth table is developed, with the output, F being the majority vote outcome. W and X were denoted S_1 and S_0 forward thinking to the implementation.

$W(S_1)$	$X(S_0)$	Y	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 1: Voting Machine Logic

2. **Intuit the Design**
   With the 4-1 MUX, we only have two select lines, so the classic implementation where each select line "selects" a specific output, the complexity must be reduced from three to two. Grouping the first two inputs S_1 and S_0 and making a fifth column which designates our four inputs D_0 D_1 D_2 and D_3 allows this to be done simply:

$W(S_1)$	$X(S_0)$	Y	F	D_i
0	0	0	0	$\mathrm{D}_0 = \mathbf{0}$
0	0	1	0	$\mathrm{D}_0 = \mathbf{0}$
0	1	0	0	$\mathrm{D}_1 = \mathbf{Y}$
0	1	1	1	$\mathrm{D}_1 = \mathbf{Y}$
1	0	0	0	$\mathrm{D}_2 = \mathbf{Y}$
1	0	1	1	$\mathrm{D}_2 = \mathbf{Y}$
1	1	0	1	$\mathrm{D}_3 = \mathbf{1}$
1	1	1	1	$\mathrm{D}_3 = \mathbf{1}$

Table 2: Simplified Voting Machine Logic

3. **Outcome**
   This table shows that the inputs to the MUX can now be mapped to either the third voter, Y, Logic High, or Logic Low. This is exactly what is needed for hardware implementation.

Example 2 A 12-Person Voting Machine?

A almost identical approach was given to the expansion from three to 12 possible voters, but with an 8-1 MUX added at the final output.

1. **The 74151 MUX**
   The 74151 MUX has *three* select lines which can be mapped to our inputs. Essentially, the problem is an expansion of the first in learning how to get four seperate input logic inputs to a single output that represents a majority vote.

2. **The Select Lines**
   In order to do this, the MUX from Part 1 is used as an input, signifying the vote of a block of three voters. If each block of three voters is an input, a logic table indicating the results of these MUX's can be made and implemented

3. **The Simplification**
   The same simplification must occur. There are four inputs to be selected, with only three select lines, so one of the inputs becomes the logic.

$D(S_2)$	$C(S_1)$	$B(S_0)$	A	F	D_i
0	0	0	0	0	$D_0 = \mathbf{0}$
0	0	0	1	0	$D_0 = \mathbf{0}$
0	0	1	0	0	$D_1 = \mathbf{0}$
0	0	1	1	0	$D_1 = \mathbf{0}$
0	1	0	0	0	$D_2 = \mathbf{0}$
0	1	0	1	0	$D_2 = \mathbf{0}$
0	1	1	0	0	$D_3 = \mathbf{A}$
0	1	1	1	1	$D_3 = \mathbf{A}$
1	0	0	0	0	$D_4 = \mathbf{0}$
1	0	0	1	0	$D_4 = \mathbf{0}$
1	0	1	0	0	$D_5 = \mathbf{A}$
1	0	1	1	1	$D_5 = \mathbf{A}$
1	1	0	0	0	$D_6 = \mathbf{A}$
1	1	0	1	1	$D_6 = \mathbf{A}$
1	1	1	0	1	$D_7 = \mathbf{1}$
1	1	1	1	1	$D_7 = \mathbf{1}$

Table 3: 12 Person Logic Table with MUX Inputs

4. **The Outcome**
   An reduction of what our data inputs to the 8:1 MUX is developed and easily implementable in hardware with the three varying inputs 0 or Logic Low, A, and 1 or Logic High. The select lines input the logical outcome from the first three voting machines.

Practice Questions

Example 3 The Difference between Mealy and Moore Machines
In sequential logic, there are two ways to illustrate the forms of how a circuit operates called *Finite State Machines*. These forms show the *next state logic* and the *output logic*.

In general, a FSM contains 2^k *finite* output states such that there are M inputs, N outputs, and k bits of state.

In a **Moore Machine**, the outputs depend exclusively on the *current state* of the machine. This means that the output remains consistent as long as the system stays in a particular state, regardless of changes in the inputs. Consequently:

- **Predictable Outputs**: Because the outputs are tied solely to the state, they are stable and do not change unexpectedly due to momentary fluctuations in the inputs.
- **Design Simplicity**: A Moore Machine is often simpler to design because the output logic only needs to account for the state, not the inputs.

In contrast, a **Mealy Machine**'s outputs are determined by a combination of the *current state* and the *current input*. This makes the output sensitive to changes in the input, allowing for faster responses:

- **Responsive Outputs**: Since outputs are based on both the state and input, a Mealy Machine can react immediately to changes in inputs, making it more dynamic.
- **Complexity**: The dual dependency on both inputs and state can make a Mealy Machine more complex to design and predict. However, it can also lead to fewer states being required, as the inputs directly influence the outputs.

Example 4 J-K Flip Flop Made from D Flip-Flop

The *J-K Flip-Flop* is an extended *S-R Flip-Flop* which attempts to solve the erroneous state in which both State and Reset are Enabled.

This introduces a new "Toggle" state in which on each clock pulse, the output Q and $\overline{Q}$ flip values.

The *D Flip-Flop* or *Delay Flip-Flop* are widely used to form shift or storage registers. This only has one data input D, a clock CLK, and the outputs Q and $\overline{Q}$. But first, some terms to describe these *things*.

- Transparent - When Data, D flows to Output Q.
- Opaque - When Data, D is blocked from flowing and Q retains its old value.
- Master(Leader) - When two back to back flip-flops or latches are controlled by complimentary clocks and the output of the Master(Leader) Q flows into the input of the Slave(Follower) D
- Slave(Follower) - The second latch or flip-flop in the complimentary clock chain. This follows what the Master does.

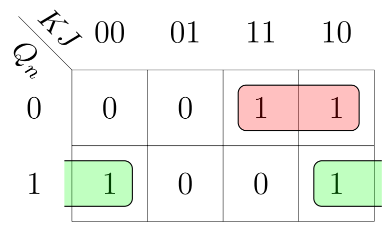
The D Flip Flop is just two D latches tied together which are simply SR Latches that are clocked. More information on the distinction between these can be found in the next example.

To convert from one to the other, the the table describing Q_n, our current state, and Q_{n+1}, the next state is populated logically. This will form the inputs to the D Flip-Flop. Recall that the J and K inputs correspond to Set and Reset from the SR Latch.

Q_n	J	K	Q_{n+1}	D_i
0	0	0	0	0
0	0	1	0	0
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Table 4: Mapping J-K Logic to D Logic

I use a Karnaugh Map to simplify this mapping:



Therefore,

$$D = Q_n\overline{K} + \overline{Q_n}J$$

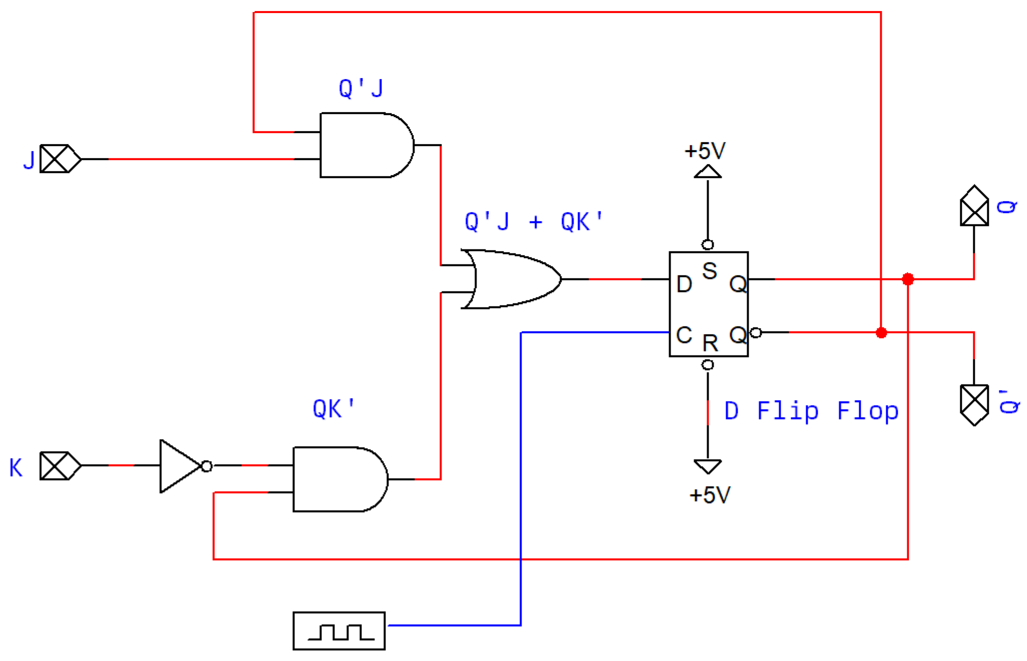


Figure 2: The J-K Flip-Flop Implemented with D Flip-Flop

Example 5 Latch and Flip-Flops?
The main difference between any latch and any flip-flop is that the flip-flop is *edge-triggered*.

Edge-Triggered means that when a clock's rising or falling edge is felt on the circuitry, depending on whether it is positive or negative edge-triggered, triggers the data to flow and the Flip-Flop to be transparent.

In general when not specified, it is assumed that a *latch* and *flip-flop* refers to a D flip-flop or a D latch.

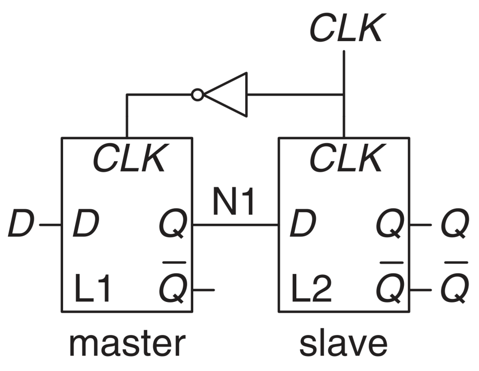


Figure 3: A D Flip-Flop made from D Latches

1. The Master is enabled during the clock's LOW phase.
2. The Slave is enabled during the clock's HIGH phase.

Latches are Level Triggered, not Edge-Triggered.

The complimentary clock pulse is tied to a slave and a master latch, which handle the pulse on different clock *levels*. This can lead to buggy output behavior, so the Flip-Flop can precicely capture data at the clock edge.