# EE 404 - Controls Lab #2:
# Implementing State-Transition Logic on a PLC

**Objective:**

Assuming that speed is not of essence, PLC's can be used to implement state transition logic. The advantage of using a PLC over using hardware is that programming is much easier and more versatile than building hardware. This lab attempts to demonstrate that state-transition logic can be implemented on a PLC and that such implementations are relatively easy to program.

In order to implement switching logic, flip flops are required. Two basic flip-flops are the T, used in step 1, and D, used in step 2. Once a way to implement a flip flop is found, these can be used to implement state-transition logic, as described in step 4.

**Procedure:**

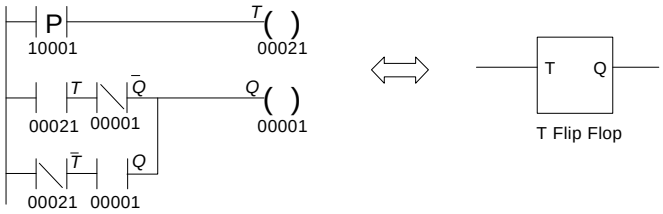
**PLC Implementation of a T Flip Flop.**

One way to implement a T-flip flop uses asynchronous logic. The output (Q) should remain unchanged if T=0, and flip if T=1. The equation for this is

$$Q = Q \cdot \bar{T} + \bar{Q} \cdot T \qquad (1)$$

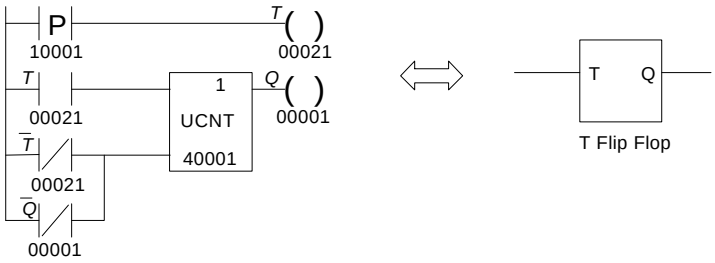
This circuit can then be implemented in ladder logic using the following program. Here, a rising edge on switch #1 is detected in the first rung. This signal then toggles Q according to (1).

**Step #1:** Program in the following circuit and verify that it does operate as a T-flip-flop.




A second way to build a T-flip flop is to use an up-counter that counts to 1. If Q=0 initially, the counter is enabled through the 4th rung. When T pulses, the counter increments, setting Q=1. When T pulses again, the counter is cleared.

**Step #2:** Program in the following circuit and verify that it behaves as a T-flip flop.



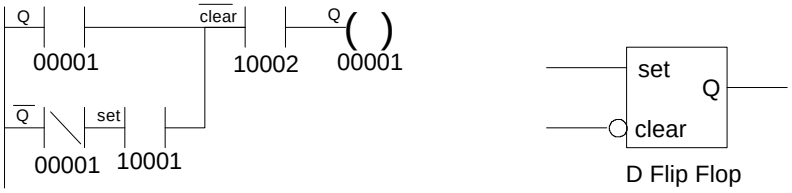

**Set-Clear Flip Flops (D Flip Flops)**

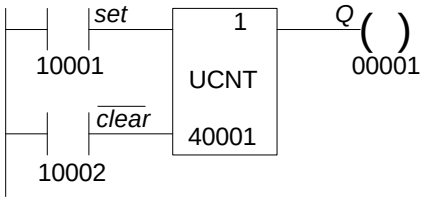
A second type of flip flop is a D-type. Here, Q=1 when set (D) goes high. Q=0 on clear. Two different circuits which implement this function follow: The first uses asynchronous logic with

$$Q = Q \cdot \overline{clear} + \overline{Q} \cdot D \cdot \overline{clear}. \qquad (2)$$

The second uses a counter.

**Step #3:** Program in each of the following two circuits. Verify that the output is set when SW1 (D) is high and is cleared when SW2 (clear) is high.



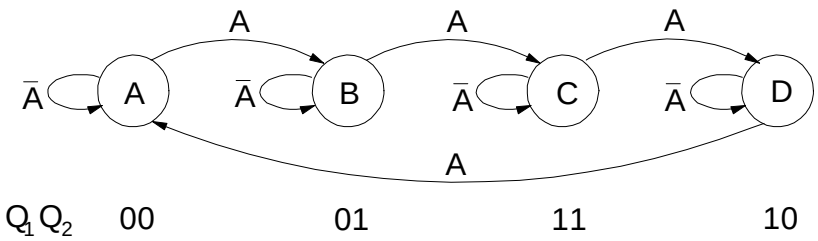




| set | $\overline{clear}$ | Q |
|---|---|---|
| x | 0 | 0 |
| 0 | 1 | Q |
| 1 | 1 | 1 |

**Implementing Switching Logic on a PLC**

**Problem:** Design a circuit which counts how many times SW1 goes high. On the 4th count, start counting from zero again.

**Solution:** One solution is to use state-transition logic. Each state defines one set of events. In this case, whether the switch has been set 0, 1, 2, or 3 times. When SW-1 (a) pulses, you are to go to the next state. This can be expressed symbolically as follows:




**Implementation Using D-Type Flip-Flops**

Using flip-flops, each state is assigned a unique bit pattern. Since 4 states exist, two bits are required, $Q_1Q_2$, with the value of $Q_1Q_2$ corresponding to each state being arbitrarily defined in the above figure.

Once the state-transition diagram is defined, the inputs to the flip flops as a function of the input and the state is determined. Assuming that D-type flip flops are being used, the Karnough maps for Set and Clear are as follows.

Set1

| A \ Q1Q2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | 0 | X | X |
| 1 | 0 | 1 | X | X |

$\overline{\text{Clear1}}$

| A \ Q1Q2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | X | 1 | 1 |
| 1 | X | 1 | 1 | 0 |

Set2

| A \ Q1Q2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | 0 | X | X | 0 |
| 1 | 1 | X | 0 | 0 |

$\overline{\text{Clear2}}$

| A \ Q1Q2 | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 0 | X | 1 | 1 | X |
| 1 | 1 | 1 | 0 | X |

For example, if you are in state 00 (A) and the input is 0, you want to remain in state 00. This means that Set should be zero and Clear can be anything. This is denoted as Set=0 and Clear=X in the upper left corner of each map.

The minimal covering maps which define each input are denoted by the circles on the Karnough maps and result in the equations

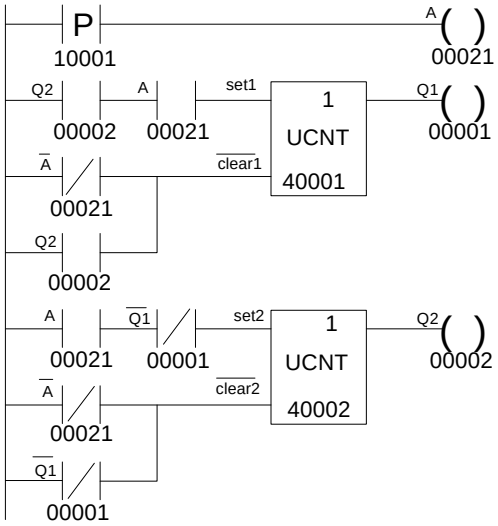
$$Set1 = A \cdot Q_2 \tag{3}$$

$$\overline{Clear1} = Q_2 + \bar{A}$$

$$Set2 = A \cdot \overline{Q_1}$$

$$\overline{Clear2} = \bar{A} + \overline{Q_2}$$

A ladder-logic program to implement these equations along with D-flip flops follows:




**Step #6:** Program in the above circuit and verify that $Q_1Q_2$ pass through the states $00 \rightarrow 01 \rightarrow 11 \rightarrow 10 \rightarrow 00 \rightarrow$ .... each time SW1 pulses high.

**Implementation Using Asynchronous Logic**

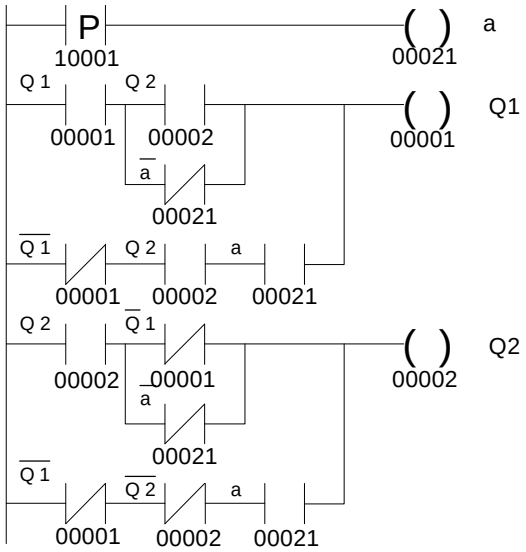
Instead of using state-transition logic, asynchronous logic could also be used. Here, the states, Q1 and Q2, should be

$$Q_1 = Q_1 \cdot (Q_2 + \bar{a}) + \overline{Q_1} \cdot (Q_2 \cdot a) \tag{4}$$

$$Q_2 = Q_2 \cdot \left(\overline{Q_1} + \bar{a}\right) + \overline{Q_2} \cdot \left(\overline{Q_1} \cdot a\right)$$

(Set Q1 if you're in state 11 or 1x and **a** is low. Also set $Q_1$ if you're in state 01 and **a** is high.)

**Step #7:** Implement the following program on a PLC. Verify that the states transition from $00 \rightarrow 01 \rightarrow 11 \rightarrow 01 \rightarrow 00 \rightarrow$ ... each time SW1 goes high.




**Write-up:**

1) Give the equivalent circuit for each of the ladder-logic programs using AND-OR-NOT gates.

2) Comment on the advantages and disadvantages of implementing switching logic with PLC's vs. discrete-components.

3) Comment on some of the advantages and disadvantages of using flip-flops instead of asynchronous logic to implement switching logic. Which method would you prefer using if writing a program? Which would you prefer if modifying someone else's program? Why?

4) Design a J-K flip-flop using latter logic. The logic diagram is

| J | K | Q |
|---|---|---|
| 0 | 0 | Q |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | Q |