

# Experiments for the

# Lab9500

## Experiment 221

### - Traffic Light State Machine -

**Introduction:** Earlier, we investigated a special kind of state machine called a counter. For the most part, the counting or state sequence is unchanging. A state-machine with no variation in the state sequence can also be called a sequencer. A simple traffic light is a good example of a sequencer. A conventional traffic light controls traffic in two orthogonal directions. Let us assume that the direction of one set of lights controls traffic on a main street and we will hereafter call that direction *main*. The other set of lights controls traffic on a side street less traveled than the main street and we will call that direction *side*. For a simple sequencer traffic light, the only difference in the light pattern is that in the main direction the time of the green light is longer than that for the side direction.

For a long time, simple traffic lights had a modulo four sequence. 1) Green in the main direction, red in the side 2) Amber in the main direction, red in the side. 3) Green in the side direction, red in the main. 4) Amber in the side direction, red in the main. At this point the sequence would repeat. Modern traffic lights alter this sequence by adding a state after the amber ones in which the light in the direction presently amber goes red, but the light in the other direction remains red. The purpose of this, of course, is to provide a “grace” period so that if a car is trying to “beat the amber” and misses, continuing through the intersection when its direction is red, the other direction has not yet turned green. In theory this should provide some additional safety. However, since everyone is aware of the overlapping reds, those trying to beat the light have started to depend on the overlapping red grace period.

Whether it is still effective or not, the typical sequence is now modulo six, with an overlapping red state after an amber state. If we denote the lights in the main direction with capital R,A and G for Red, Amber and Green, and corresponding lower case letters for the side direction, the six states are Gr, Ar, Rr, Rg, Ra, and RR. There is nothing magic about the starting state. It is arbitrary.

We already know how to make a state machine by using a set of flip-flops to represent the various states. We can make a present state/next state table, and from the table and Karnaugh maps, derive the equations for the state machine. We have already used this technique to implement counters. With ABEL we have another technique we can use.

ABEL has state machine capabilities that makes it possible to make a state machine without making the present/next state transition table and using Karnaugh maps to derive the D (or T) inputs to the flip-flops that make up the state machine. The starting point is not that different.

for the state machine approach. We need to start with a state diagram with circles representing the state, and arrowed lines connecting state with the lines labeled with the condition that causes that state change. In the declaration section of the program, we define a state register and some states:

This ABEL state defines the state machine variables.

$$\text{SREG} = [\text{Q2}, \text{Q1}, \text{Q0}];$$

The binary state values can be assigned randomly, but we could also ascribe some meaning to the state flip-flops. Let bit Q2 be true for the main direction, let Q1 be true for an amber light, and let Q0 be true for a green light..

Then we could name the state 001 Rg. The leading zero per the above relation indicates a side amber or green light, and the trailing 1 indicates it is a green light. If the leading bit, Q2, is a one that would indicate green for the main direction. Accordingly, those two states would be.

$$\begin{aligned} \text{Rg} &= [0,0,1]; \text{ "For example} \\ \text{Gr} &= [1,0,1]; \end{aligned}$$

Note, SREG is not a keyword. We could just as easily said instead TLIGHT.

ABEL lets us follow the state register declaration with symbolic state definitions, such as Rg and Gr. Note, that these are the same as  $\neg\text{Q2}*\text{Q1}*\text{Q0}$  and  $\text{Q2}*\text{Q1}*\text{Q0}$  respectively. We can immediately see how much more understandable the symbolic definitions are. Note, also that the bit patterns of the state counter Q2 .. Q1 are arbitrary. We ascribed a property to each of the three bit positions, but this wasn't really necessary. If this was a design to be made with discrete parts, it makes a big difference if some signal is one product term or two or more! In the CPLD, however, there are minimally five product terms per macrocell, so we don't have to go out of our way to make the state codes efficient.

In the state machine structure in ABEL, a series of flip-flop nodes are used to represent the various states which are given names. The assignment of names to binary flip-flop combinations is strictly arbitrary. Even if the state machine is a sequencer, there is no requirement to assign states in a linear fashion. Indeed, there may be some benefit from assigning a non-linear sequence such as the one just described.. In the simple mod six traffic light (which we will step manually with a pushbutton), the current state determines what the next state will be. Later when we design a self-advancing light, the light will include a counter/timer to determine the duration of a state. So the present state not only determines the next state, but also the count to be loaded into the timer/counter which determines the duration of the next state. By numbering the states carefully, a minimal number of product terms will be required. For example, the amber time occurs after either one of the green times. Suppose the two green states (CBA) are 100 for main green and 101 for side green. What they have in common is  $\text{C}*/\text{B}$ . So this single product term could be used to select the preload of the timer to be for the amber time.

In practice, the “fat” of the 9500 series is probably sufficient enough so that even if you do not go out of your way to choose optimum state definitions, you’ll never notice the difference and the “fitter” will manage to implement the design. Still, it is good design practice to use state definitions that reduce the number of product terms required.

A simple traffic light then could be little more than a mod 6 counter with the sequence of states assigned the names as above. Later, however, we will propose that our not-so-simple traffic light (eventually) have two additional modes. One will be called Night Mode. In Night Mode, the amber light in the main direction and the red light in the side direction *flash* at a typical rate of 2Hz. In most locales, a flashing amber indicates caution, but no stopping. A flashing red means stop, look, then go.

Night Mode usually occurs in the wee hours of the night and morning when traffic is very minimal. During daytime or evening low traffic times, it may be desirable to have a mode which I will call Trip Mode. In this mode, the light remains indefinitely in the main green state until a car approaches from the side and trips a road switch. Here are some considerations for trip mode. When in trip mode, the main green should stay green for its regular duration. That is, if the main light has just turned green, a car tripping the switch should not truncate that state and cause the light to turn amber. (Design details of added modes will be discussed in a later experiment.

If we used a straight-forward mod 6 counter for the simple traffic light state machine, then when we added additional states in the future, we would have to design the state machine from scratch. However, if we use the ABEL *state machine* construct, then we need only add to the state transition definitions to add additional states. A complete redesign will not be necessary.

## **Experimentation**

For the suggested experiment, assume that a manual pushbutton provides the clocks necessary for state changes. That is, the operator will become the “timer”.

Use the state machine structure of ABEL and design a simple modulo 6 traffic light state machine. The pushbutton switch S1 (S11) should be used to advance the state machine.