State Transistional Logic

S-R Flip Flops

One way to describe how a didital system operates is to define a set of states. For example, a stoplight might have three states:

- Green
- Yellow
- Red

The heart of state-transistion logic is the flip-flop: a device which can have two states, on (1) and off (0). There are several types of flip flops. For the Micro810, the SR flip flop is what's normally used (it is part of the ladder logic software). The SR flip-flop has two inputs: set (S) and reset (R). These affect the output as follows:

S (set)	R (reset)	Q (output)	Operation
0	0	no change	idle
0	1	Q = 0	clear
1	0	Q = 1	set
1	1	?	not allowed

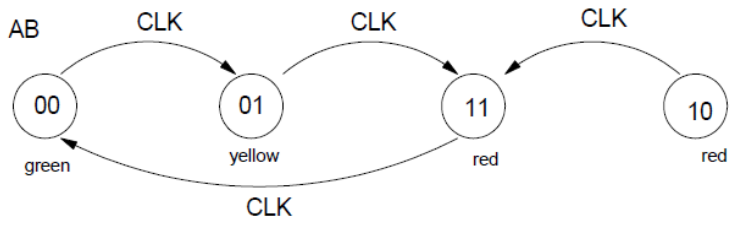
By defining the combinational logic for each input (S and R) correctly, you should be able to get the stoplight to go from one state to the next as defined above. To do this, you look at the present state of each flip-flop (A and B) and its next state. The input for S and R are as follows, where X means 'don't care'

Present State	Next State	S	R	Operation
0	0	0	X	don't set
	1	1	0	set
1	0	0	1	clear
	1	x	0	don't clear

State Transistion Logic on a PLC

With a PLC, you have access to AND, OR, and NOT functions. That is all you need to implement any logic function. The programs can be somewhat difficult to write or debug, however. For example, write a program to implement a stop-light with a manual input. Every time you press IN0, the light changes from green to yellow to red. This is also called a 3-element ring counter.

Step 1: Define the states. Since there are three states, use two flip flops: A and B. Assign the states and the state transistions as follows:



State Transition Diagram for a stoplight. (a.k.a. a ring counter)

Step 2: Determine the state transistion logic. Set up Karnough maps for set and clear.

$$S_A = \overline{A}B \qquad R_A = AB \qquad S_B = \overline{B} \qquad R_B = AB$$

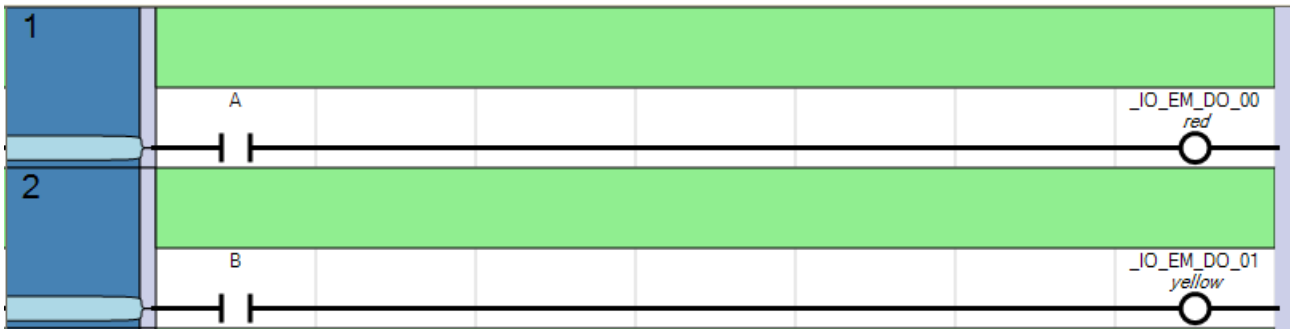
Step 3: Implement with ladder logic. Note that Code Composer executes the program line-by-line. This can create problems since if a variable is changed in rung 3, the new value is used in the latter rungs, not the old value. This makes the order in which you place the rungs inportant. To avaoid this, create two temp variables, zA and zB: the next value of A and B. At the end of the program, A and B are updated.

Also note that the outputs are defined at the top of the program. (see next page)

Notes:

i) The CLK input is a pulse input. If you make it a normal switch, the program will shift from state-to-state every cycle (10ms). By making CLK an edge device, CLK goes high for one progam cycle (10ms) each time you press the button.

ii) When writing this program, I would first ignore the actual requirements (red / yellow / green lights) and just make the outputs the flip-flop states as follows:



This allows you to step through the program, making sure it goes through the sequence of states

00 - 01 - 11 - repeat.

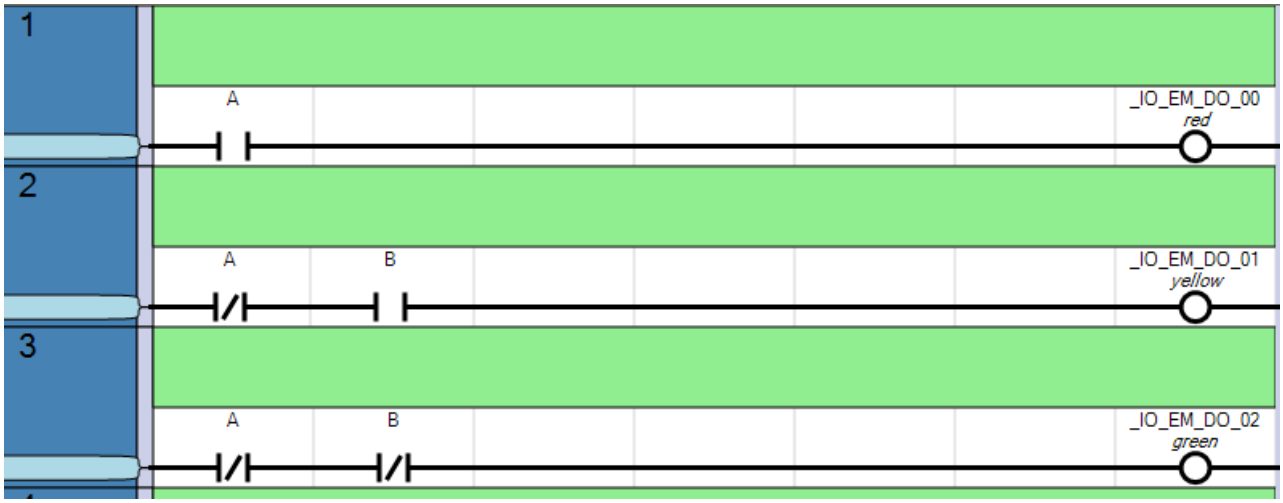
Once you're convinced the states are correct, you can come up with combinational logic for the lights:

$$Green = \overline{A}\,\overline{B}$$

$$Yellow = \overline{A}B$$

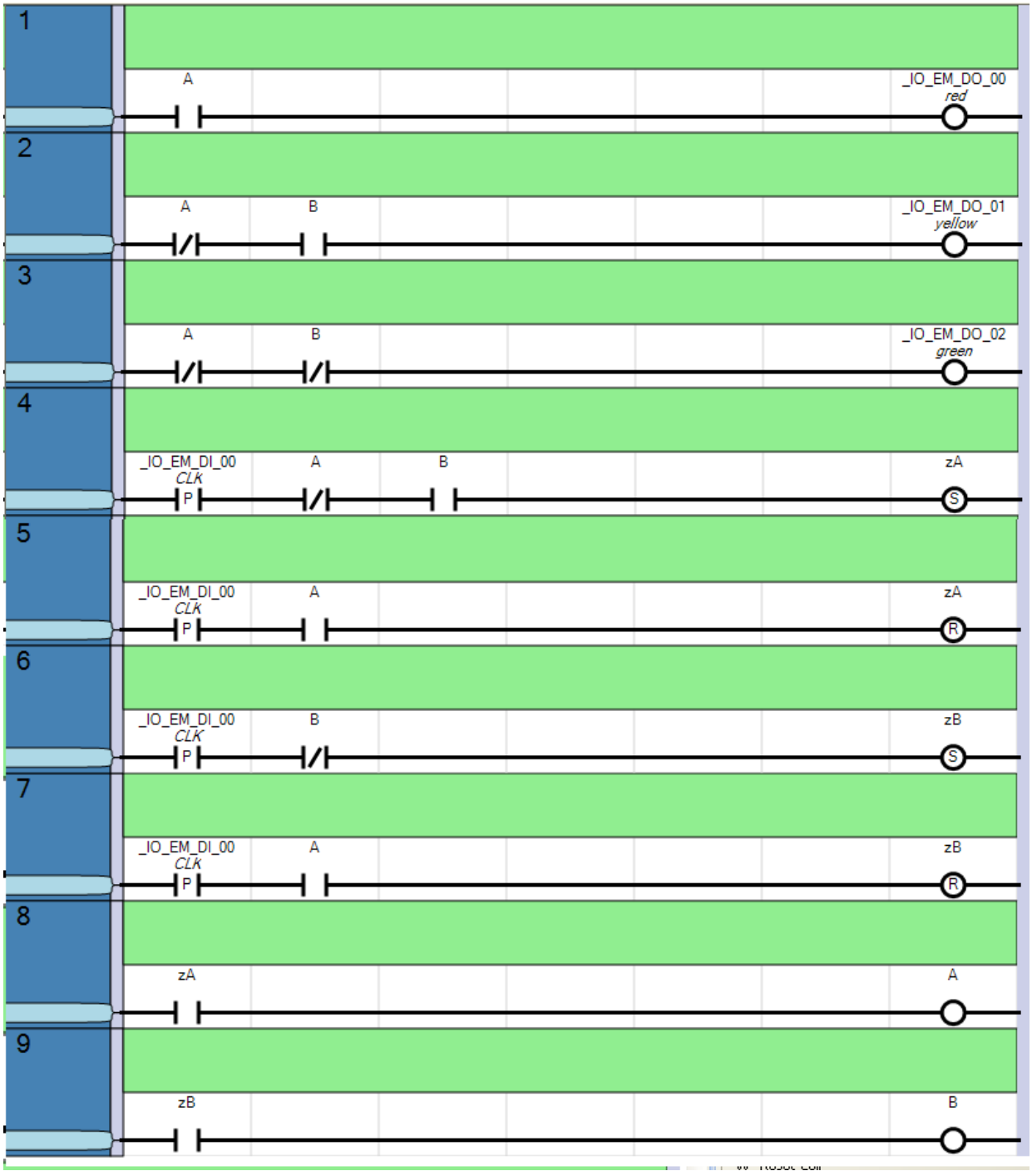
$$Red = AB + A\overline{B} = A$$

or, the first three rungs become



With this, you can step through the program (press button 0) and watch the stoplight go from green to yellow to red and repeat.

The entire program is on the following page:


1
A
_IO_EM_DO_00
red
2
A
B
_IO_EM_DO_01
yellow
3
A
B
_IO_EM_DO_02
green
4
_IO_EM_DI_00
CLK
P
A
B
zA
S
5
_IO_EM_DI_00
CLK
P
A
zA
R
6
_IO_EM_DI_00
CLK
P
B
zB
S
7
_IO_EM_DI_00
CLK
P
A
zB
R
8
zA
A
9
zB
B

Comments:

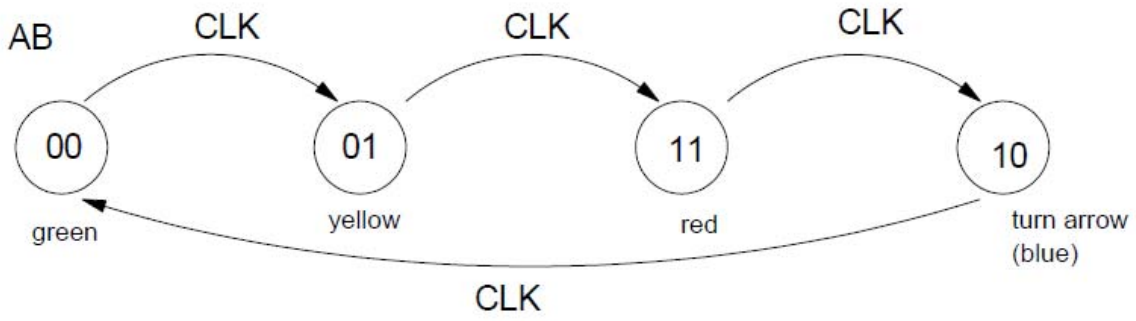
- You can implement any logic function using state-transistion logic. It helps if you first draw the state-transistion diagram first, however.
- To implement three states, you need two flip-flops. This gives four permutations (red - yellow - green - unused). You need to make sure that the unused state isn't a stuck-at state (meaning if it powers up in this state, you can get out.

State transition logic has several problems, however.

- First, it is difficult to understand without documentation.
- Second, it is hard to modify. If you wanted to add two more options (flashing red, flashing yellow) you have to redesign the entire program.

Example: Add a 4th state: left turn arror (indicated with a blue LED)

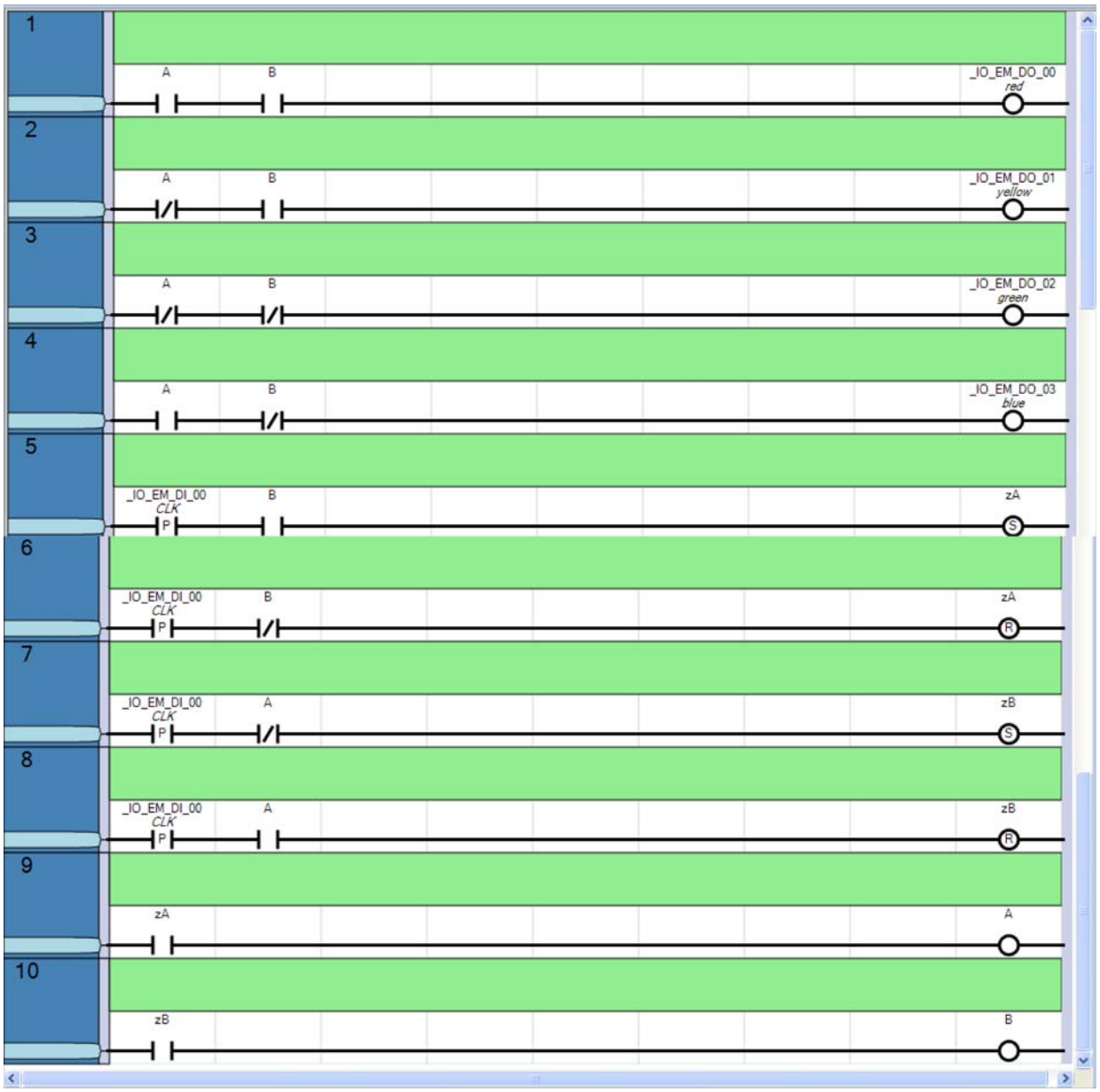
Step 1: Draw a state-transistion diagram:



Step 2: Define the SR logic using a Karnough map:

$S_A = B$ $R_A = A$ $S_B = \overline{A}$ $R_B = A$

The ladder logic program then looks like the following:



Note:

- Everything you learned in ECE 275: Digital Systems applies to ladder logic.
- If you want to change the operation, you need to do a complete redesign.
- If you want to add a 5th state, you need three state variables and a major redesign.