Exercise 2: RGB LED

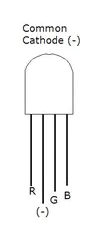
# Objective

This exercise has two parts.

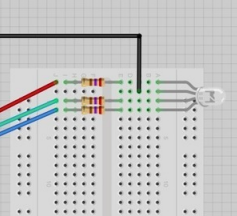
1. Build a circuit a push button and RGB LED on a breadboard.
2. Write ladder logic to control the circuit
   1. Step through an 8-color sequence with two control modes
   2. mode 1: push button changes color
   3. mode 2: colors change cyclically with a timer
3. Connect to an HMI.
   1. The HMI is provided. You will need to import it into SCADABr and start it running.

# Build the circuit

1. Insert the 3 color LED into the breadboard. The RGB LED has a common cathode pin. It is the longest pin on the device. Connect this pin to ground on the Arduino.



1. Connect the R, G, and B pins to a 1K Ohm resistor and connect the other end of the resistor to pins 7, 8, and 12 respectively.



1. Connect a button to pin 2 of the Arduino. Refer to exercise one for the appropriate button circuit.
2. Connect power and ground rails of the breadboard to the 5V and GND pins of the Arduino.

# Write Ladder Logic

1. Using the OpenPLC Editor create a new project.
2. Create the following variables.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Class | Type | Location |
| count | Local | INT | %MW8 |
| color0 | Local | INT | %MW0 |
| color1 | Local | INT | %MW1 |
| color2 | Local | INT | %MW2 |
| color3 | Local | INT | %MW3 |
| color4 | Local | INT | %MW4 |
| color5 | Local | INT | %MW5 |
| color6 | Local | INT | %MW6 |
| color7 | Local | INT | %MW7 |
| red\_anode | Local | BOOL | %QX100.0 |
| green\_anode | Local | BOOL | %QX100.1 |
| blue\_anode | Local | BOOL | %QX100.2 |
| button | Local | BOOL | %IX100.0 |
| mode | Local | BOOL | %QX0.0 |
| count\_en | Local | BOOL |  |
| auto\_count\_en | Local | BOOl |  |

The 8 color variables (*color0-color7*) are connected to registers. These registers can be set by the HMI (SCADABR) via MODBUS. This will allow the user to control what color is output for each count value.

*count* is also a register, but this variable will be controlled by ladder logic. The %MW8 is a read/write type and in this case is used to allow the HMI to read the count and display it. An attacker might choose to write to this variable!

*red\_anode*, *green\_anode*, and *blue\_anode* are controlled by the ladder logic. These values will drive the Arduino pins connected to the RGB anodes on the RGB LED.

*button* is an input from the Arduino. When the button is pressed this value will be 1. When the button is released this value will be 0.

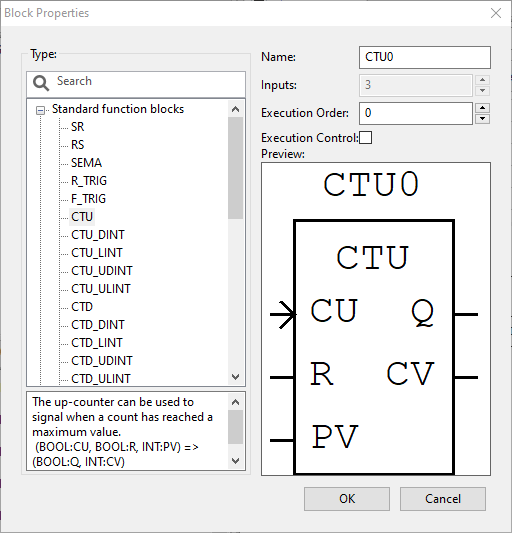
*mode* is an input from the SCADABR HMI. %QX0.0 is a single bit read/write coil. *mode* is not connected to the breadboard. Instead we will control it from the HMI.

*count\_en* is an internal ladder logic variable that is used to advance a counter controlling the count variable. Since it is internal it is not connected to a location.

*auto\_count\_en* is another internal ladder logic variable that is used to advance a counter controlling the count variable. Since it is internal it is not connected to a location.

1. Use a CTU to count repeatedly from 0-7.

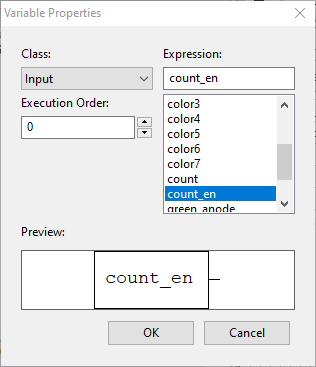
The CTU is a up-counter.



CU is the enable pin of the CTU block. When CU is high the counter will advance. The counter actually advances once per 0->1 transition of CU. So, if you leave CU set to high it will not continue to count. Connect CU to the count\_en variable. This is an internal variable we will configure in step 6. To connect a variable to a pin first instance a VAR (drag it on to the ladder edit console).



Configure the VAR properties to select count\_en for Expression field and set the Class to Input. Next connect the VAR instance to the CU pin of the CTU object.

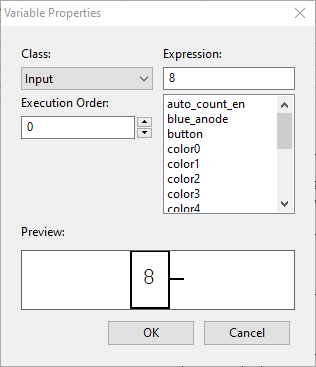


CV is the current value. We will connect this pin to the count variable we create above. The CV pin must be connected to a variable of type INT (there are other CTU variants for other variable types). The Q pin is high CV == PV. We will use PV to set the maximum value we want our counter to achieve before resetting to 0.

Connect PV to a constant with value 8. To create a constant instance a VAR (drag it on to the ladder edit console).



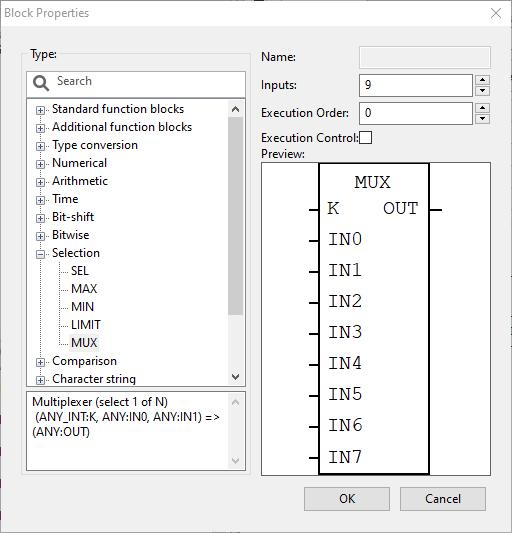
Set the Class to Input and the Expression to 8.



Connect the CTU Q pin to the R pin. This will assert the R pin when PV==CV and reset the CTU to 0. Note: The CTU

1. Use a MUX to select the color to display. The count output from the counter above is the select input (K) of the MUX. The 8 color vectors are connected to IN0-IN7 respectively.

Instance a MUX object. The MUX object is found under the Library (right side of screen) in the “Selection” group of objects.



Notice the types for K, IN0-IN7, and OUT. K is of type ANY\_INT. We don’t set the type but connect variables of the correct type. Our CTU is producing an INT so for K we are good. The type for IN0-IN7 and OUT is listed as ANY. These must all be the same type to avoid errors. We will use INT for all.

Set the number of inputs to be 9. This is 8 (the number of possible selections: IN0-IN7) plus 1 (the selector input, K).

Connect the K input to the count variable from the CTU object.

Connect IN0-IN7 to color0-color7 respectively.

The MUX acts like a switch (C/C++) statement.

switch (K) {

case 0: OUT = IN0; break;

case 1: OUT = IN1; break;

case 2: OUT = IN2; break;

case 3: OUT = IN3; break;

case 4: OUT = IN4; break;

case 5: OUT = IN5; break;

case 6: OUT = IN6; break;

case 7: OUT = IN7; break;

}

1. Use Divide-by-2 and Modulo-by-2 to select the bits of the color vector and connect these to red, green, and blue anodes.

The output from our MUX object is a 16-bit INT. We want to select the least significant 3 bits of the MUX output from above and connect them to red\_anode (bit 2), green\_anode (bit 1), and blue\_anode (bit 0).

There are 2 ways to do this (1) write a custom structured text function block or (2) use the DIV and MOD objects and compute the values we need. We haven’t learned structured text, so, use option 2.

* blue\_anode = MUX.OUT bit 0 = MUX.OUT % 2

*Note: The result of this is a type INT. Use an INT\_TO\_BOOL object to convert before connecting to the blue\_anode coil.*

* green\_anode = MUX.OUT bit 1 = (MUX.OUT / 2) % 2

*Note: The result of this is a type INT. Use an INT\_TO\_BOOL object to convert before connecting to the green\_anode coil.*

* green\_anode = MUX.OUT bit 2 = ((MUX.OUT / 2) / 2) % 2

*Note: The result of this is a type INT. Use an INT\_TO\_BOOL object to convert before connecting to the red\_anode coil.*

The output of the bit selection logic should connect to coils connected to *blue\_anode, green\_anode¸ and red\_anode respectively.*

The 3 coils should connect to a right power rail.

1. Create logic to drive the count\_en input of the CTU object. This logic will need to determine if the circuit in auto or manual mode and act accordingly.

Use a SEL object to select between the button input and the auto\_count\_en signal based upon the value of the mode input.

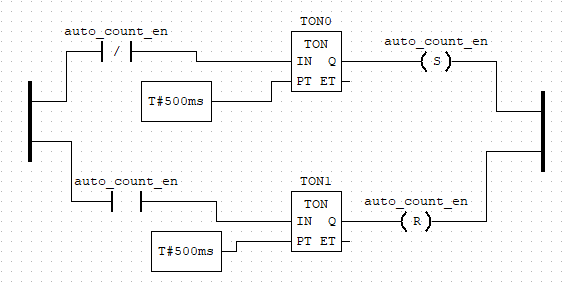
Connect the mode signal to the G input of the SEL object.

Connect the button variable to the IN0 input of the SEL object.

Connect the auto\_count\_en variable to the IN1 input of the SEL object.

1. Create a clock waveform circuit driving auto\_count\_en with a 50/50 duty cycle and 1 Hz frequency.

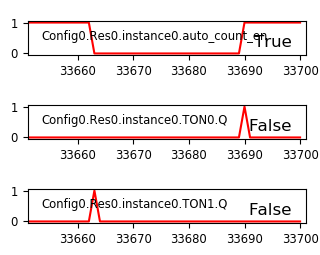
In auto mode we would like the CTU object to automatically advance. This will cause the color to change each second.



The circuit above uses an on-delay timer (TON) and set and reset coils to create the desired clock waveform.

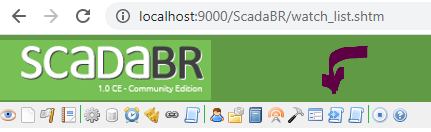
The IN input of a TON is an enable signal. When the TON is enabled (IN transitions from 0 -> 1) the timer waits the amount of time specified on the PT input and then turns on the Q output. When the TON is disabled (IN transitions from 1 -> 0) the Q output immediately turns off. The ET output shows the elapsed time of the timer. ET is the time since the last 0 -> 1 transition on IN.

The circuit above uses set and reset coils. These coils have memory and behave like latches. When a S (set) coil is set to 1 its associated variable stays set until reset. When a R (reset) coil is triggered (a 0->1 transition on its input) its associated variable stays reset (value 0) until set. The figure below demonstrates this behavior. The top waveform is auto\_count\_en. The next two waveforms are TON0.Q and TON1.Q respectively. The short pulse on TON0.Q causes auto\_count\_en to go high and stay high. The short pulse on TON1.Q causes auto\_count\_en to go low and stay low.

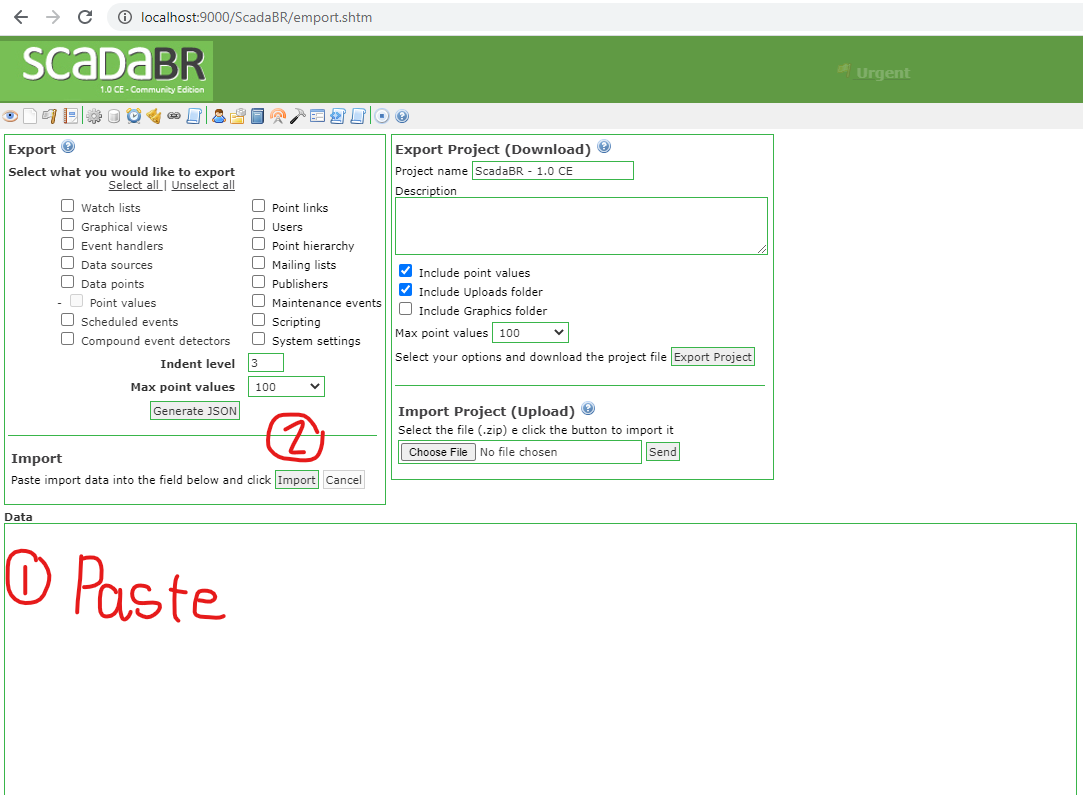


# Connect the HMI

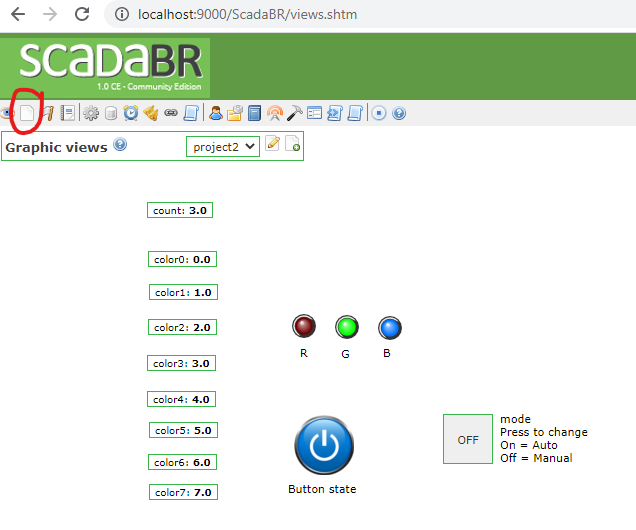
1. We will not develop a new HMI for this exercise. Instead we will import an existing HMI.
2. The Exercise 2 folder on Piazza includes an exercise2-hmi.json file. Download this file to your PC.
3. Navigate to SCADABr.
   1. Turn on the SCADABr VM.
   2. Login with your account.
   3. Start the Apache service by running the following command.  
      *sudo /opt/tomcat6/apache-tomcat6-6.0.53/bin/bootstrap.jar*
   4. Navigate to SCADABr in your host’s browser.  
      <http://localhost:9000/ScadaBR>
   5. Login to SCADABr (user: admin, password: admin)
4. Import the exercise2-hmi.json file.
   1. Click Import/Export on the tool bar. 



* 1. Manually paste contents of the exercise2-hmi.json file into the data field.



* 1. Press the Import button marked above.
  2. Press the Graphical views button. 



* 1. You need to update the IP address of the host (your PLC) in the data sources screen.
     1. Go to data sources screen.
     2. On the left side look for Host IP. It is set to 10.0.2.5. Change this to your PLC’s IP address.  
          
        To get your PLC’s IP address.
        1. Log onto the PLC
        2. Type the command ***ip a***
        3. Record the IP address
  2. You may need to stop and restart the Data sources.
     1. Go to the Data sources screen. Press .
     2. Click the status icon to disable. Press .
     3. Click the status icon to enable. Press .

# Post Exercise Report

Answer the following questions.

1. This exercise uses division operations to extract bits from a vector. What might happen if the denominator of the division operation became zero? Hint: Exception handling.
2. Assume causing the LED to flash RED is a signal to an operator that the system was in an alarm state, how might an attacker cause the RGB LED to flash RED by manipulating the color0-color7 variables and changing the mode.
3. If you power off the SCADABr HMI VM while the PLC program is running does the PLC program and LED circuit continue to operate? Which if any functionality is lost?