Date: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Section: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Names: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ and \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
| LABORATORY CHECK OFFS | |
| **Section 1 Preparing the RGB LED circuit and testing it**  Pressing switch turns on correct LED color (Red/Green)  **Section 2 Viewing serial data on the oscilloscope**  Oscilloscope traces of serial data and clock when LED color is set to green.  Measure the total time required to clock out all the data (from when clock first goes low to when it stays high =\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ microseconds  **Section 3 Rebuilding the BLINK\_GR state machine using the RGB led**  Demo Blink\_G, Blink\_R, Blink\_GR, BlinkRate  **Section 4 Measuring code execution time using a software timer**  Record the execution time for greenOn() using digitalWrite = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Section 5 Measuring code execution time when PORT command is used**  Record the execution time for greenOn() using PORT = \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  **Section 6** **Improving the state machine to avoid redundantly running code**  Serial Monitor prints new state names BLINK\_G\_ON and BLINK\_G\_OFF and the execution time is only printed once. | \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_  \_\_\_\_ |
| Points | \_\_\_\_ |

PRELAB:

Solder the header pins on the RGB LED and wire it up on the protoboard as described in section 1.

Learning Outcomes for this lab:

At the end of this lab:

1. You will be able to create a bitmask using the bit shift command <<.
2. You will be able to use a bitmask to select one bit at a time from a 32-bit integer.
3. You will be able to convert a 32-bit integer into a serial data stream and clock the data out using a clock signal.
4. You will be able to set the color and brightness of an RGB led using serial data.
5. You will be able to measure the execution time for a section of code using the microseconds() command.
6. You will be able to make a state machine run more efficiently adding states so that most actions are performed only during the entry to and exit from the state.

What is this lab about ?

Microcontrollers interface to a variety of external sensors and devices. In the last lab, you interfaced with an analog sensor (e.g. QTR reflectance sensor), a digital device (e.g a red LED), and a digital input (e.g. switch). In this lab you will learn how interface the microcontroller to a device that requires serial data to control it. The external device for this lab is a tri-color RGB LED which has an internal chip that receives serial data allowing the microcontroller to individually set the brightness of each LED color.

You will create a serial data signal and a clock signal and use these to set the color and brightness of an RGB LED by sending data to to the RGB LED’s internal shift register.

You will also learn how to create sounds using a speaker and the tone() library.

The lab will also be a review of state machine structure:

* 1. Input scan and signal conditioning
  2. State entry housekeeping
  3. State business
  4. State exit housekeeping
  5. State transition

How to Succeed With This Lab:

When in doubt, print it out. In other words, when you are troubleshooting use the Serial.println() command to check each step of the code. Also, use the oscilloscope as appropriate to see what the clock and serial data pins are doing. Refer back to the previous lab handout for sample code of the state machine. Utilize the Arduino.cc website for reference information on Arduino pin definitions and language.

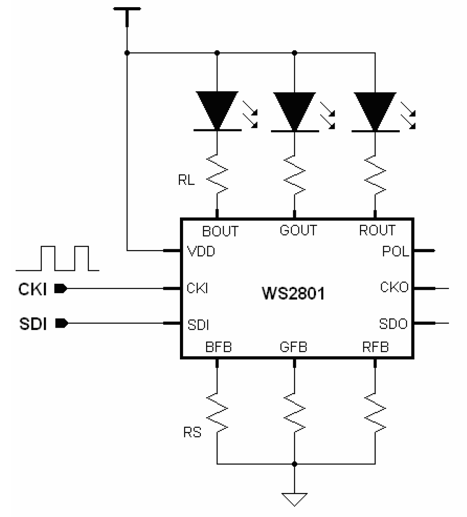
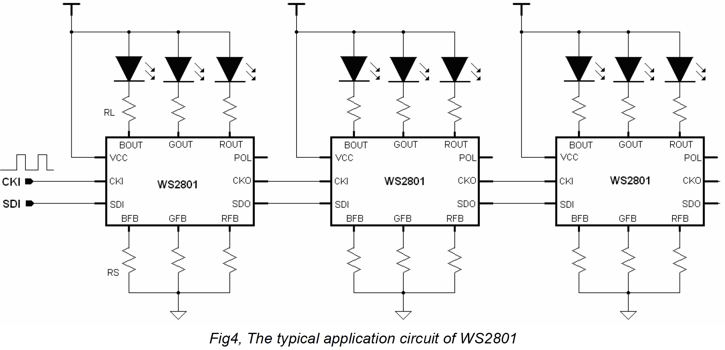
The number one problem students face doing this lab:

Understanding bit shifting operations and bit masking operations.

**SECTION 1 – Preparing the RGB LED circuit.**

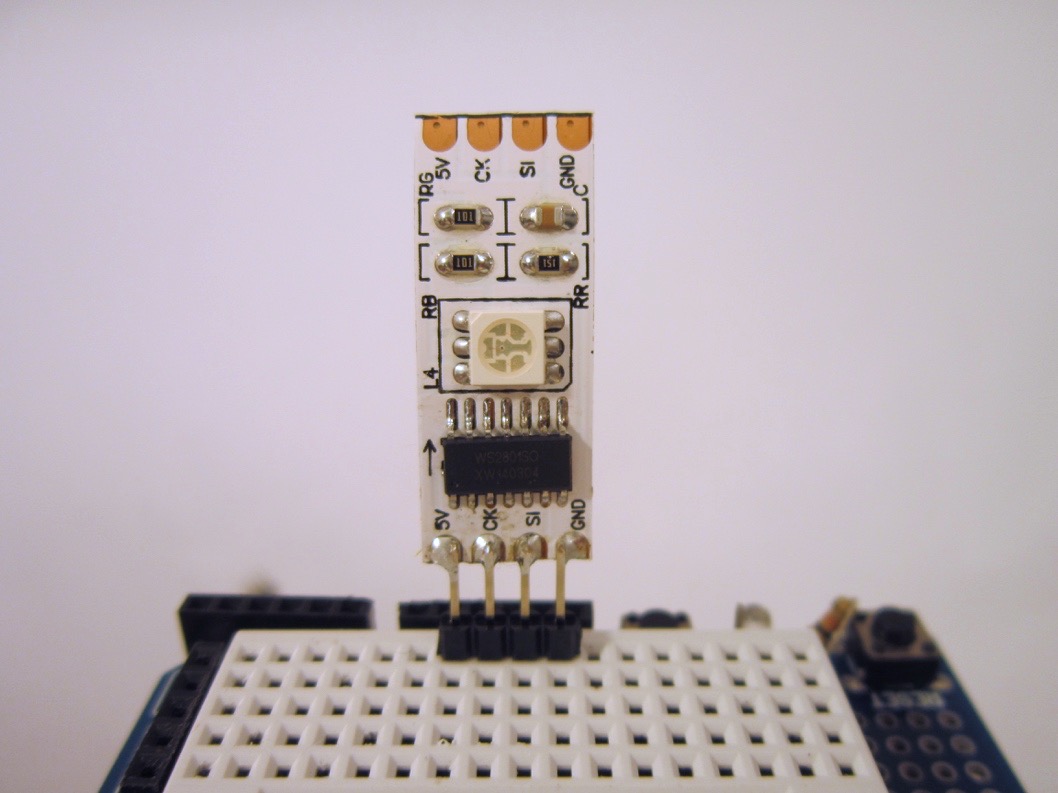
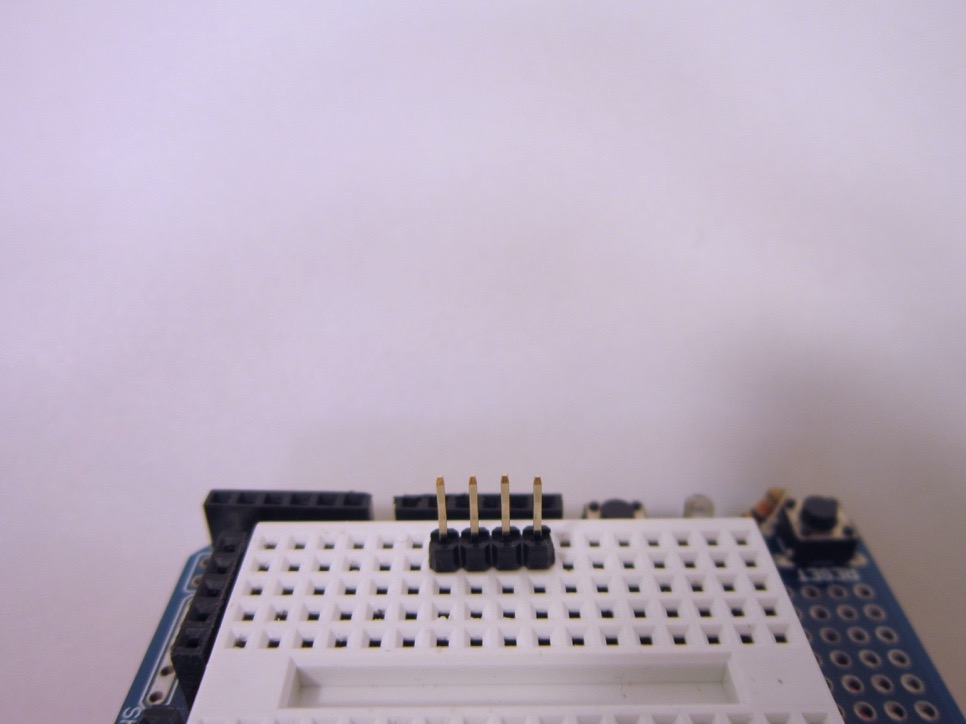
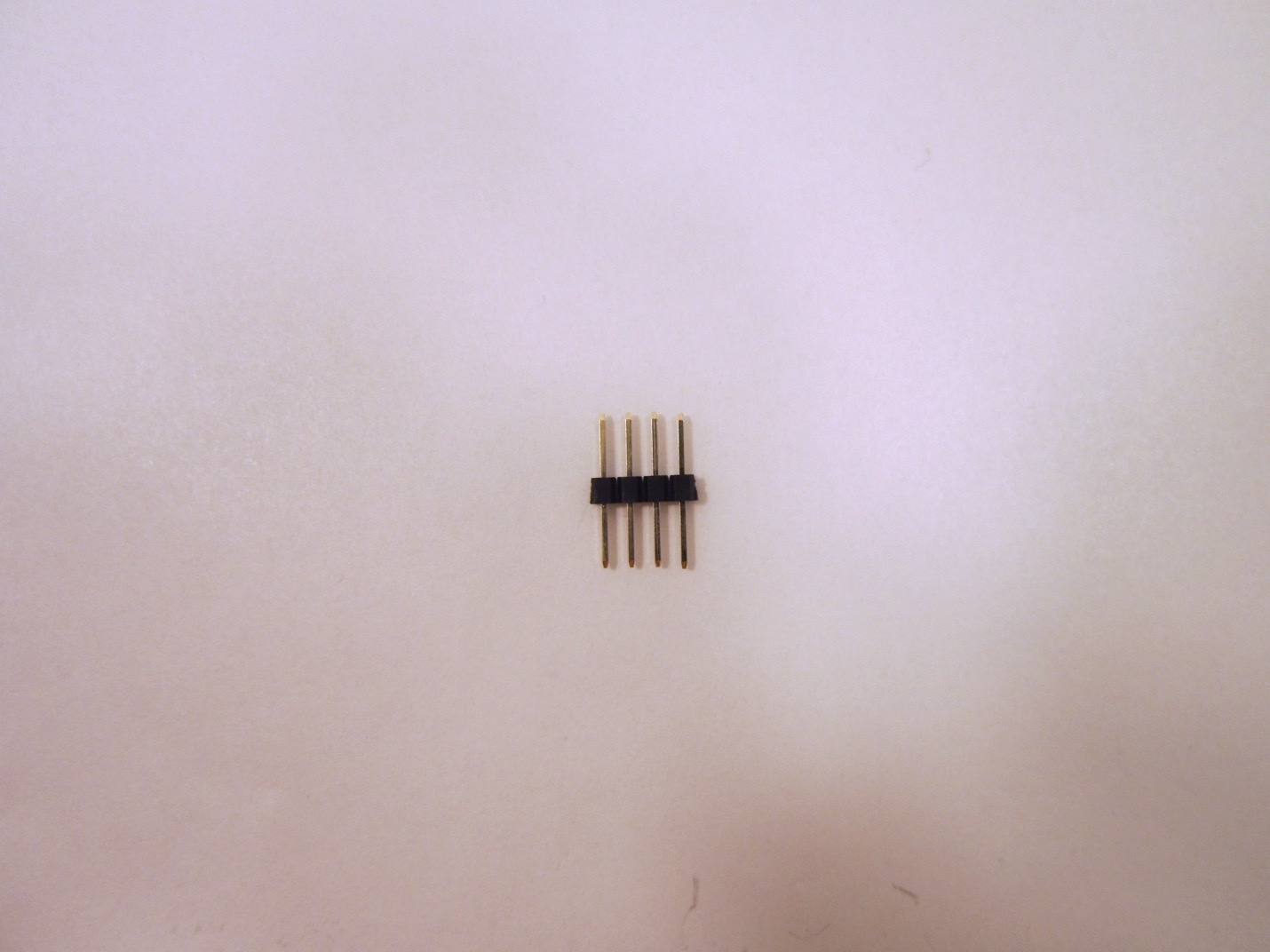
The RGB LED is driven by a constant current driver chip (Ws2801) that receives serial data and stores it in an internal shift register.

*(\*\* Note that the green and blue colors may be swapped from what is shown below. This is due to manufacturer changes. This is ok, you can just swap the green and blue values in code. This will make sense as you run the code. \*\*)*

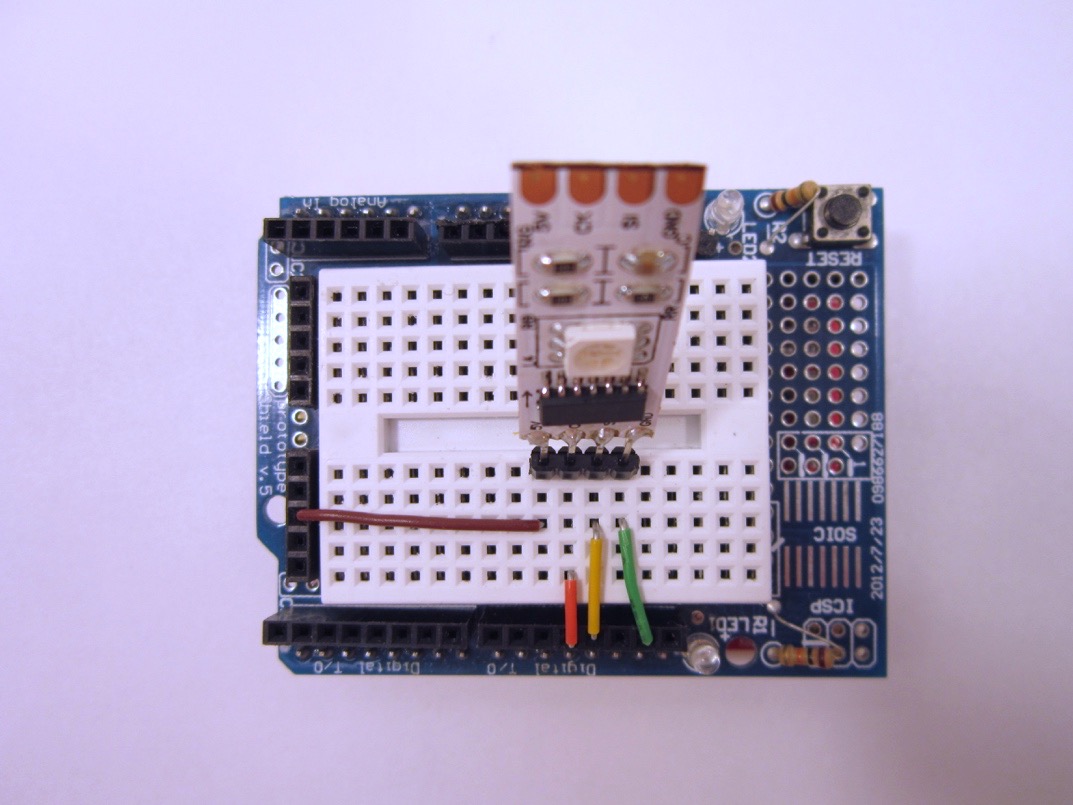
 

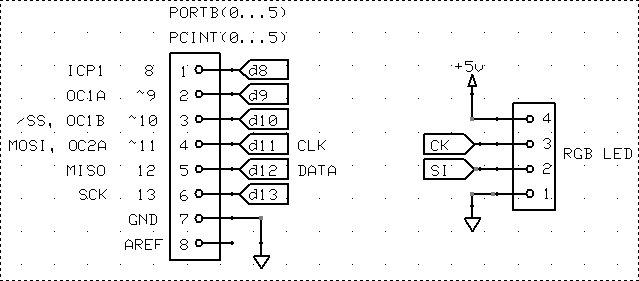
**Procedure:**

1. The RGB LED strip is shown in the picture below. The LED strip is designed to be connected in series so that one serial data line can be used to control hundreds of LEDs and set the color and brightness in each LED to a different value.
   1. The black chip (ws2801) is constant current driver for three LEDs. The Note that there is an arrow near the black chip package. Serial data is fed into the SI (Serial In) pin and each data bit is clocked in, one at a time, on the rising edge of the CK (Clock) pin. The serial data into the chip is buffered and sent out to the next LED in the strip.
   2. Three LEDs (RGB) are housed in the white package.
   3. At the top of the picture are solder connection points for passing 5V, ground, serial data and clock signals to the next LED strip.
   4. We will be using just one section of LED strip, but if want to you can connect many LEDs in series.

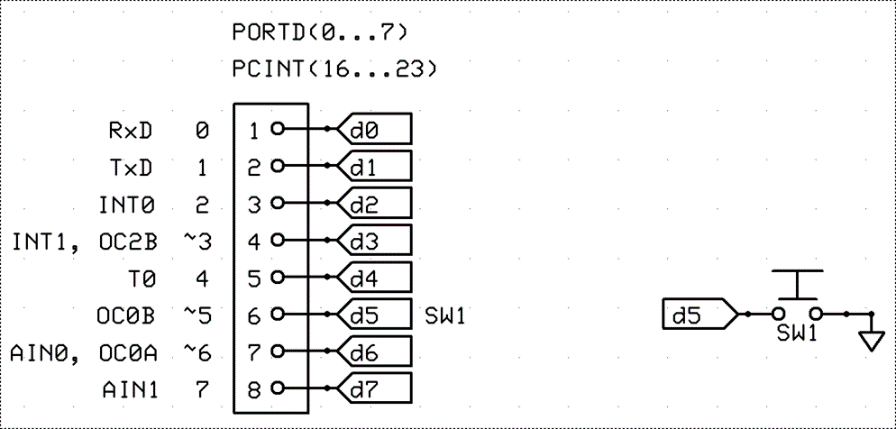


1. Break off a section the header strip that is four pins long. Use the head strip with long pins on each side. These pins will be soldered to the LED strip as shown. Put the header pins into your breadboard to make it easer to solder the LED strip. To solder the LED strip, you may find it helpful to have another person hold the strip while you solder. Start by getting just one pin soldered so that the strip is held in place. Then move on to soldering the other pins.
2. Wire the RGB LED as shown in the figure and in the schematic below. The LED needs 5V and ground. Connect the LED serial data pin SI to Arduino pin 12 and connect the led clock pin CK to Arduino pin 11.





1. Next, add in a switch to ground on digital pin 5 as shown in the schematic below.



1. Copy the sample code from below and verify that the LED lights up red.

// Lab3\_minLEDredSWpress.ino

#define LED\_DATA\_PIN 12

#define LED\_CLOCK\_PIN 11

#define SW1\_PIN 5

boolean isSwPressed=false;

void setup() {

Serial.begin(9600);

Serial.println("Lab3\_minLEDredSWpress.ino");

pinMode(SW1\_PIN, INPUT\_PULLUP);

pinMode(LED\_DATA\_PIN, OUTPUT);

pinMode(LED\_CLOCK\_PIN, OUTPUT);

digitalWrite(LED\_DATA\_PIN, LOW);

digitalWrite(LED\_CLOCK\_PIN, LOW);

} //setup

// =======================================================================

void loop() {

isSwPressed = !digitalRead(SW1\_PIN);

if (isSwPressed) {

display\_color\_on\_RGB\_led(0xFF0000); //red

//display\_color\_on\_RGB\_led(0x00FF00); //green

//display\_color\_on\_RGB\_led(0x0000FF); //blue

}

else {

display\_color\_on\_RGB\_led(0x000000); // all leds off

}

}//loop()

// =======================================================================

void display\_color\_on\_RGB\_led(unsigned long color) {

unsigned long bitmask=0UL; // UL unsigned long literal (forces compiler to use long data type)

unsigned long masked\_color\_result=0UL;

digitalWrite(LED\_CLOCK\_PIN,LOW); //start with clock low.

for(int i=23; i>=0; i--) { // clock out one data bit at a time, starting with the MSB first

bitmask= (1UL<<i); // build bit mask. Note must use "1UL" unsigned long literal, not "1"

masked\_color\_result = color & bitmask; // reveals just one bit of color at time

boolean data\_bit=!(masked\_color\_result==0); // this is the bit of data to be clocked out.

digitalWrite(LED\_DATA\_PIN,data\_bit);

digitalWrite(LED\_CLOCK\_PIN,HIGH);

digitalWrite(LED\_CLOCK\_PIN,LOW);

}

digitalWrite(LED\_CLOCK\_PIN,HIGH);

delay(1); // after writing data to LED driver, must hold clock line

// high for 1 ms to latch color data in led shift register.

}//display\_color\_on\_RGB\_led()

**Description of the Code**

In setup(), the led data and clock pins are configured as outputs and the switch pin is configured as an input. In the main loop, the switch is read and if it is pressed, serial data is sent to the RGB led driver (WS2801 chip) to turn the red LED on at full brightness (255 decimal = 0xFF hex). If the switch is released, all three LEDs (RGB) are given brightness commands of zero (0x00).

The function display\_color\_on\_RGB\_led() controls the LED brightness. It accepts a color variable that is 32 bits long. The lowest 8 bits of the color variable represent the brightness of the blue LED. The brightness can be set from 0 to 255 (full on). The next higher set of 8 bits of the color variable represent the brightness of the green LED. The next higher set of 8 bits of the color variable represent the brightness of the red LED. Finally, the uppermost 8 bits of the number are not used but are included because the microcontroller internally represents number as either 16 bit integers or 32 bit long integers.

The color variable is constructed as a 24-bit number that is stored into a 32-bit long integer datatype as shown below.

Generic 32 bit integer

MSB LSB

00000000 00000000 00000000 00000000

Color value example (all LEDs at zero brightness):

MSB LSB

(ignored) red green blue

00000000 00000000 00000000 00000000 (brightness value for each color (8 bits each))

0 0 0 0 (brightness in decimal format)

0x00 0x00 0x00 0x00 (brightness in hex format)

Color value example (red LED on at full brightness):

MSB LSB

(ignored) red green blue

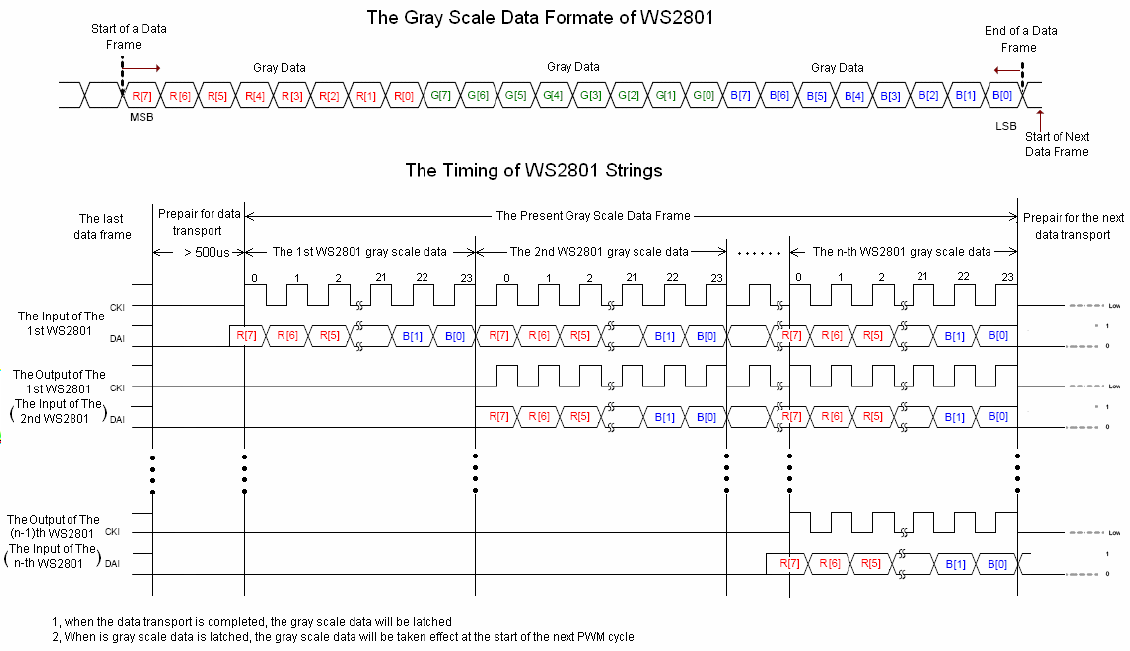
00000000 11111111 00000000 00000000 (brightness value for each color (8 bits each))

0 255 0 0 (brightness in decimal format)

0x00 0xFF 0x00 0x00 (brightness in hex format)

total color equals red => 0x00FF0000 (32 bits) or just 0xFF0000 (24 bits)

Only the lower 24 bits of the color variable control the brightness but the upper 8 bits are needed to fit the brightness data into a 32-bit integer value that the computer can store.



The function display\_color\_on\_RGB\_led() takes the 32-bit color variable and formats it as serial data so that the LED driver chip can receive it. First, the upper 8 bits of the 32-bit color variable are ignored and not sent to the LED driver chip. The remaining 24 bits are then sent out, one bit at time to the LED driver chip. The most significant bit is sent first. This will be the MSB of the red LED brightness value. To access this bit, a bitmask is created that pulls out just the 24th bit of the color variable. When the for loop starts, i=23 and the bit mask selects the 24th bit (MSB)

bitmask= (1UL<<i);

This color bit is set on the data pin and then clocked out by taking the clock line from LOW to HIGH to LOW.

digitalWrite(LED\_DATA\_PIN,data\_bit);

digitalWrite(LED\_CLOCK\_PIN,HIGH);

digitalWrite(LED\_CLOCK\_PIN,LOW);

The creation of the bitmask is done inside a for loop so that each individual bit in the color variable is selected one at time and clocked out to the RGB LED’s internal shift register. The for loop starts with the most significant bit of the color variable (bit 24) and works it way down the the least significant bit.

for(int i=23; i>=0; i--) {

bitmask= (1UL<<i);

The bit mask is created by taking a value of 1 and shifting it over to the bit location of interest. The first time through the for loop, the bit mask is set to a 1 shifted over 23 times which aligns it on the 24th bit of the color variable (the MSB). Note that the bit mask is shift over not just a “1” but a “1UL” . This tells the compiler to use a 32 bit unsigned long literal value for the 1. The reason for this is that a decimal number like -1,1,2,123456, etc. without any suffix will get the smallest type it will fit, starting with int, long. With out the “1UL” the bitmask variable will default to a 16 bit integer and will not work as desired for masking a 32-bit integer. Debugging this type of problem can be very challenging. When working with bitmasks, be very careful to specific the datatype of every variable, even a “1”.

After each bit is masked and clocked out, the for-loop ends. The final step is to set the clock signal high for at least 500 microseconds. Holding the clock line high tells the WS2801 driver that the data it has received should be latched in. At this point the RBG colors will be updated and displayed.

digitalWrite(LED\_CLOCK\_PIN,HIGH);

delay(1); // after writing data to LED driver, must hold clock line

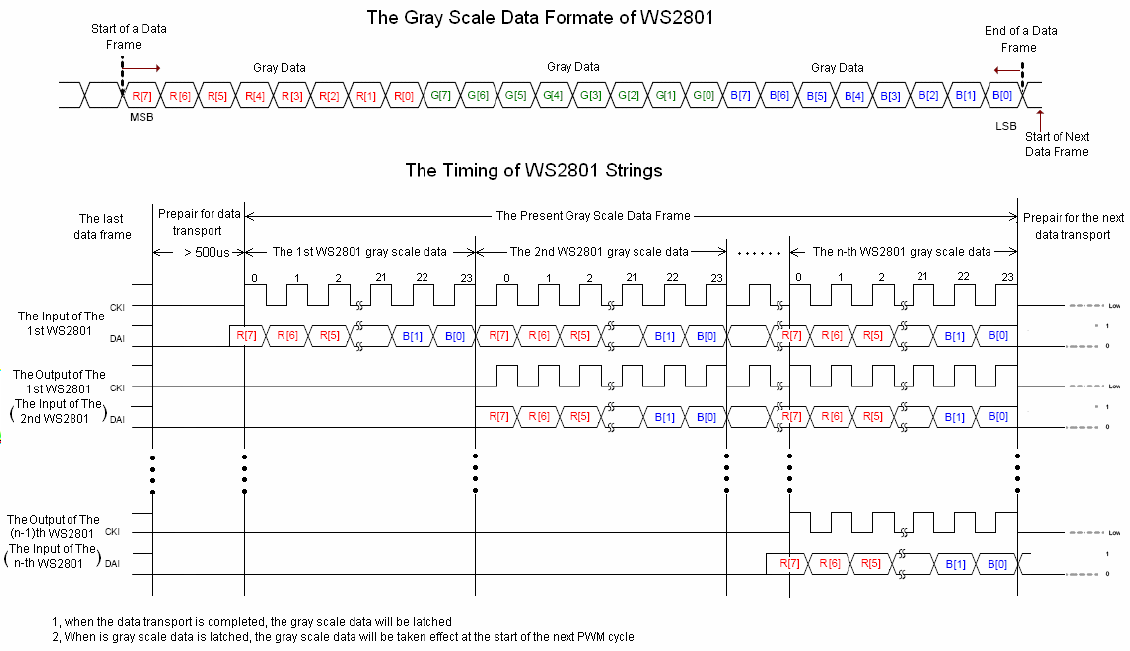
// high for 1 ms to latch color data in led shift register.

1. Signoff - Demonstrate to your lab instructor that the red LED comes on when you press the button. Now change the code so that the green LED comes on and demonstrate this to your instructor.

**SECTION 2 – Viewing serial data on the oscilloscope**

**Procedure:**

1. Connect two channels on the oscilloscope to display the clock signal and the serial data signal. Capture serial data and clock traces when the RGB color is set to green.
2. Show your oscilloscope traces to your instructor for a signoff.
3. Just for your interest, a timing diagram is shown below for the case where three WS2801 driver chips are connected in series (three different RGB LEDs would be used). This diagram shows how the output serial data from the first chip is sent to the second chip as serial input data.



**SECTION 3 – Rebuilding the BLINK\_GR state machine using the RGB led**

In this section, you will replace the discrete red and green LED’s from lab 2’s blink\_GR state machine with the new RGB LED. You will then demonstrate the RGB statemachine and all its capabilities. Changing the code over will be easier because the statemachine structure uses functions such as redOn(), greenOn() and allOff(). You will insert new code in these functions but the rest of the state machine structure will remain the same.

**Procedure:**

1. Re read the code in section 1 so that you understand what the main functions are and how to use them.
2. Copy your code from section 8 of lab 2 (full state machine with Blink\_R and Blink\_GR) and paste it into a new file.
3. In this new file, remove all code references to LED1 and LED2 (i.e. remove the pin number definitions, pinmode statements and digital writes.) Keep the QTR pins, etc.
4. Add pin definitions for the LED data and clock pins and configure these pins as outputs in the setup() section.
5. Include the function display\_color\_on\_RGB\_led() in your new file.
6. Modify the redOn(), greenOn() and allOff() functions to use the new RGB LED. In each one, you will need to first create a color and then display that color on the RGB led.
7. Upload your code and verify it works. Demonstrate the state machine to your instructor.

**SECTION 4 – Measuring code execution time using a software timer**

In this section, you will measure how long it takes to execute a particular section of code. Previously, you have used an oscilloscope to do this for the individual digitalWrite() and PORT commands. It also possible to use a sfotware timer built into the code to measure execution time. The Arduino code has a default timer for milliseconds, which is used by the delay function. The Arduino code also has a timer with microsecond resolution that can be read using the command microseconds().

**Procedure:**

1. Modify just the greenOn() function to include a measurement of execution time. The code declares two variables to store the time in microseconds, one for when the timer is is first read and then one for when the timer is read after the function display\_color\_on\_RGB\_led() executes. The elapsed time is the end time value minus the start time value.
2. Insert the code pieces bolded in the code snippet in the text box below.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

void greenOn(void) {

// digitalWrite(LED1\_PIN, LOW);

// digitalWrite(LED2\_PIN, HIGH);

**unsigned long start\_time\_microseconds,end\_time\_microseconds;**

**start\_time\_microseconds=micros();**

display\_color\_on\_RGB\_led(0x00FF00);

**end\_time\_microseconds=micros();**

**Serial.print("Displaying the green color took ");**

**Serial.print(end\_time\_microseconds-start\_time\_microseconds);**

**Serial.println(" microseconds ");**

}

1. Run the code and observe the Serial Monitor’s output when the code is in the Blink\_G state and the green LED is on.
2. Record the time value on your signoff sheet.

**SECTION 5 – Measuring code execution time when PORT command is used**

From the last lab, you know that digitalWrite() commands are slower than PORT commands. In this section you will measure the difference in execution time for the display\_color\_on\_RGB\_led() function when it uses PORT commands instead of digitalWrite() commands.

**Procedure:**

1. Modify only the display\_color\_on\_RGB\_led() function so that it uses PORT commands instead of digitalWrite() commands. Comment out the digitalWrite() command so that you still have a reference for what the PORT commands are trying to accomplish.

**Tip:**  The PORT values for each Arduino pin can be determined from the Pin Mapping diagram given in the Lab #1 handout. The schematics in section 1 also show the PORT labels for the Arduino headers.

1. Measure the execution time of the greenOn() function again and record the value on your signoff sheet.

**Observation:**  You might notice some strange behavior when the green LED is on. It may be staying on much longer than expected and longer than the red LED. Notice also that the Serial.println() inside display\_color\_on\_RGB\_led() seems to keep printing the entire time the green LED is on. Why is this? Why is the led being turned on repeatedly even when the green LED is already on? This is not an efficient way for the code to operate. When turning on the green LED only meant doing a digitalWrite() that took 2.5 microseconds, this was not a concern but now that turning on the green LED by writing serial data takes milliseconds, this becomes a problem.

**SECTION 6 – Improving the state machine to avoid redundantly running code**

The code for the BLINK\_G state is shown in the text box below. Comments have been added to clarify what each section of code is doing. Notice that in the main state business section, the stateTimer is incremented and then if condition if (stateTimer < 250) is checked. If true, the green LED is turned on by calling greenOn(). This means that whenever the green LED is on, serial data to turn on the LED is repeatedly written and re-written to it. This is not necessary and consumes many clock cycles of microcontroller instructions.

case BLINK\_G:

**// state entry housekeeping (done once on entry)**

if (isNewState) {

stateTimer = 0;

Serial.println("BLINK\_G");

}

**//state business (done everytime through the loop)**

stateTimer++;

if (stateTimer < 250) greenOn();

else allOff();

if (stateTimer >= 1000) stateTimer = 0;

**// state exit condition and exit housekeeping (done once on exit)**

if (isSwJustReleased) {

allOff();

state = BLINK\_R;

}

break;

A better way to implement the state machine is to split the BLINK\_G state into two different states, BLINK\_G\_ON and BLINK\_G\_OFF and eliminate the BLINK\_G state.

The new BLINK\_G\_ON state is configured so that on entry, the green LED is turned on once. Then, during the state business, the LED does not need to be touched again saving processor time. The state business of BLINK\_G\_ON state is to increment the stateTimer. There are two exit conditions from the BLINK\_G\_ON state. First, if the stateTimer exceeds 250 milliseconds, then exit and go to the BLINKG\_G\_OFF state. Second, if the switch is just released, exit to the BLINK\_R state. Note that you do not need to fix the BLINK\_R state right now.

The new BLINK\_G\_OFF state is configured so that on entry, the green LED is turned off just one time. Then, during the state business, the LED does not need to be touched again saving processor time. The state business of BLINK\_G\_OFF state is to increment the stateTimer. There are two exit conditions from the BLINK\_G\_OFF state. First, if the stateTimer exceeds 1000 milliseconds, then exit and go to the BLINK\_G\_ON state. Second, if the switch is just released, exit to the BLINK\_R state.

**Procedure:**

1. Modify the state machine in your code to eliminate the BLINK\_G state and replace it by two new states, BLINK\_G\_ON and BLINK\_G\_OFF.
2. You will need to change the enum definition to have these two new states. The old version was : enum {LED\_OFF, BLINK\_G, BLINK\_R, BLINK\_GR, BLINK\_RATE}; The new version is?
3. Create copy the old BLINK\_G state and modify it as described above.
4. Make any other modifications you need to get the state machine to run. At this point you only need to fix the blink green mode. The other modes e.g. BLINK\_GR won’t work correctly yet.
5. Run your new code and observe that in the serial monitor the new state names are printed and the execution time is only printed once. Demonstrate the code to your lab instructor for a sign off.