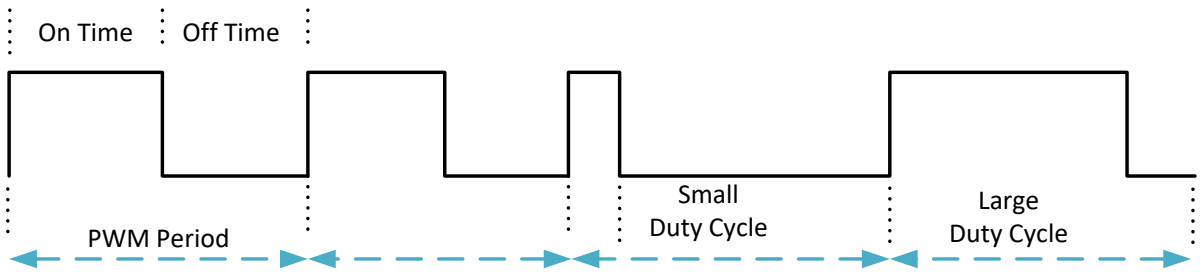
Real time control

In this week lab, we will use the Bucky board to perform “real time” control functions involving pulse width modulation (PWM). In particular, we will look at how PWM can be used to control the brightness of each color of the RGB LED. This allows controlling the exact color and intensity of a multicolor LED which you can usually only turn on or off as we did in the last lab.

In creating colors with paint we add different amounts of base colors to end up with the color we desire. For LEDs we have red, green and blue which are 3 base colors that can be used to create a wide spectrum of colors. When the R, G or B can only be 100% on or off there aren’t many different colors we can create. 8 or 2^3 is it. All off, R, G, B, RG, RB, GB and RGB. That’s it. We only get full brightness or off for each color. One particular aspect of the eye is it is kind of slow. Most people stop seeing individual images flashing in front of their face at about 30 images per second. After that is less like a flashing on and off set of pictures and starts looking like a continuous change or motion. This is why motion pictures or TV used 30 Hz for the NTSC standard for broadcasting motion pictures.

So what good does this do us. By using the Integrating or averaging nature of the eye you can have the eye take an LED that is on full some of the time and off some of the time and make it look like it is partially on all of time. If the period is shorter than 1/30 of a second it just looks like less light that is continuously on. It all about the integral of the on time compared to the off time. The shorter the pulse is the lower the amount of light the eye detects.

This is what is typically done. We pick a frequency or more correctly the period that we use to fit the on time versus off time waveform into. The PWM frequency or PWM period we can pick ourselves for this lab at least. Faster than 30Hz the on time compared to the total time of the period of the PWM frequency is varied to give different intensities of light. Let’s consider using 100 Hz to make calculating the percentage of on time versus the period of the 100Hz signal easy. So on time + off time is always 1/100 seconds. The period for 100Hz. Another definition is the Duty Cycle of the PWM waveform is the ON time / PWM_{PERIOD}.



Pretend that each blue PWM period is the same length.

7.1 Pulse width modulation

Pulse Width Modulation or PWM, is a method of using digital signals (0s and 1s) to represent continuous (or analog) signals using the principle of averaging. By exciting certain devices with a rapid stream of pulses that may take values of zeros and ones (each of which correspond to discrete energy levels), the device may be able fooled into behaving as if it was excited with their time- averaged value. While not all devices can be 'cheated' in this manner, in many cases, this allows the logical control of a digital device to control something like motors, lights, etc. as though there was a continuous signal driving them.

Terminology

PWM Period/PWM Frequency

Typical PWM patterns repeat themselves between their 'on' and 'off' states. The PWM period, (seconds), is the reciprocal of the repetition frequency (Hz), sometimes also called the refresh rate. The period of a signal is the interval of time a signal takes before repeating itself. The period and frequency of PWM signals are an important performance measure, since they represent how fast control functions may be varied in a variety of applications such as LCD screens, LEDs and motor control. One can imagine the projection of movies on the screen at the movie theater. Each frame of the movie is a static image, but displaying them at a fast enough speed (> 30 pictures a second), the eye processes them as a continuous image. If the repetition rate is slower than that, any movements appears jerky to our eyes. Many LED color displays and TV screens operate upon this principle, where multiple "discrete" images are blended together (integrated or averaged) to form a continuous image. We'll see more of this in lab, and play around with different periods of PWM signals and see its versatility.

Duty ratio

The second important concept regarding PWM is duty ratio of the signal, which represents the on-interval of a particular signal, normalized to the period of a PWM signal. The duty ratio will range from 100% (completely on) to 0% (completely off). This will dictate how long current flows through an LED to keep it on. If we extrapolate over multiple periods, we can establish that the signal is periodic and the LED is ON for its duty ratio percentage of time, on average. If the period is short enough, our eyes will not be able to recognize that it is both off and on for the times during the interval. Let's say that the duty ratio is 25%, our eyes would perceive as if the LED is emitting only 25% of the light compared to a fully on LED. In this manner, PWM may be used to develop continuously variable lighting, heating, motor speed, etc., using discrete levels.

7.2 In the lab

In this lab, we will be studying and verifying PWM operation using the microcontroller, measure PWM signal analog levels and timing using an oscilloscope.

In the standard PWM program you can the PWM frequency is fixed. You can control the duty cycle of each color of an RGB LED in real time or in our case while the program is running.

- The four push buttons are used as control inputs.
- Pressing the up pushbutton increases the duty cycle of the selected LED color by 10%. When it gets to 100% the duty cycle wraps around to 0% at the next press of the up button.
- Pressing the down pushbutton decreases the duty cycle of the selected LED color by 10%. When it gets to 0% the duty cycle wraps around to 100% at the next press of the down button.
- The right button changes which color will change with the up and down buttons. Each right button push cycles through the three colors. RED to GREEN to BLUE to RED again and so on.
- The left button changes which color will change with the up and down buttons. Each left button push cycles through the three colors. RED to BLUE to GREEN to RED again and so on.

The LCD screen tells which color is selected to change and what percentage the duty cycle is set to now. This happens after each button press.

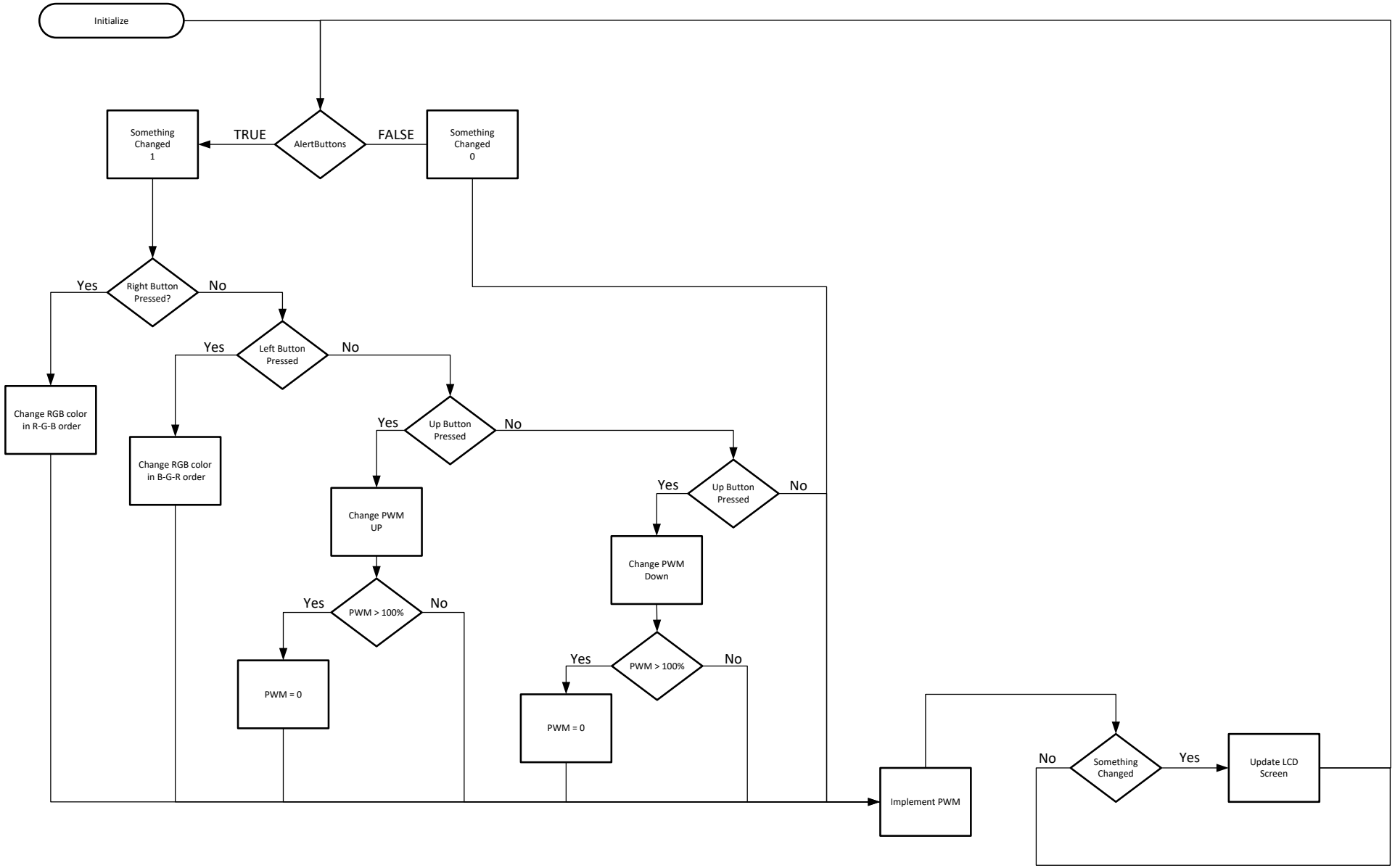
The PWM period has to be changed with a constant in the program, recompiled, reloaded and run.

7.2.1 Pulse width modulation

Figures 7.1provides a copy of the stock program for RGB_PWM.
Download PWM.zip and unzip to a folder of your choosing. It will create an RGB_PWM folder with everything needed to compile and run the program.
Start uVision 5.
Load Lab6a.uvprojx which is in the PWM-RGB folder of under the folder you created for this project.
Open main.c if it isn't already open.

This is where all the code is for today's lab. You will only need to make 1 change later for part of the lab.
Add your info to the top comment section.
Put your name into the ece210_lcd_add_msg statement that contains Your Name.
Compile the program now.
Load the program now.
What is the PWM frequency? _____

The flowchart for the program is shown next. This is followed by the code from main.c.


Initialize
AlertButtons
TRUE
FALSE
Something Changed 1
Something Changed 0
Right Button Pressed?
Yes
No
Left Button Pressed
Yes
No
Change RGB color in R-G-B order
Change RGB color in B-G-R order
Up Button Pressed
Yes
No
Change PWM UP
PWM > 100%
Yes
No
PWM = 0
Up Button Pressed
Yes
No
Change PWM Down
PWM > 100%
Yes
No
PWM = 0
Implement PWM
Something Changed
No
Yes
Update LCD Screen

```
/*******************************************************************************

 * main.c
 * Author: jkrachey@wisc.edu
 * modified by allie@wisc.edu
 ******************************************************************************/
#include "lab_buttons.h"

/*******************************************************************************
 * function declarations
 ******************************************************************************/
void rotate_led(uint8_t direction);
void change_pwm(uint8_t direction);
void implement_pwm(uint8_t milliseconds);

/*******************************************************************************
 * Global Variables
 ******************************************************************************/
#define LEFT                    0x00
#define RIGHT                   0x01
#define UP                      0x02
#define DOWN                    0x03
#define RED                     0x00
#define GREEN                   0x01
#define BLUE                    0x02
#define RED_ON                  0x02
#define GREEN_ON                0x08
#define BLUE_ON                 0x04
#define NUM_MS_TO_WAIT  1               // PWM period is = NUM_MS_TO_WAIT * PWM_MAX in
#define PWM_MAX                 10              // milliseconds. PWM_MAX=100 is 100ms PWM period.
#define PWM_MULT                100/PWM_MAX         // PWM_MAX sets the number of levels of PWM
#define PWM_START               PWM_MAX/2           // from 0 to 100%
#define PWM_FREQ                1/(0.001*PWM_MAX)
#define ALL_ON                  0x0f
#define OFF                     0x00

uint8_t led_to_change = RED;
uint8_t red_pwm = PWM_START;
uint8_t green_pwm = PWM_START;
uint8_t blue_pwm = PWM_START;

//*****************************************************************************
int
main(void)
{
  // declare and initialize variables used in main
  char display_level[20];

  uint8_t pwm_time = NUM_MS_TO_WAIT;
  uint8_t pwm_freq = PWM_FREQ;
  uint8_t something_changed = 0;

  ece210_initialize_board();
  ece210_lcd_add_msg("ECE210 PWM Lab", TERMINAL_ALIGN_LEFT, LCD_COLOR_BLUE);
  ece210_lcd_add_msg("Your name PWM Lab", TERMINAL_ALIGN_CENTER, LCD_COLOR_CYAN);
  sprintf(display_level,"PWM FREQ = %i Hz", pwm_freq);
```

```
ece210_lcd_add_msg(display_level, TERMINAL_ALIGN_LEFT, LCD_COLOR_MAGENTA);
ece210_lcd_add_msg("Up Increase PWM Level", TERMINAL_ALIGN_LEFT, LCD_COLOR_RED);
ece210_lcd_add_msg("Down Decrease PWM Level", TERMINAL_ALIGN_LEFT, LCD_COLOR_BLUE);
ece210_lcd_add_msg("Left Decrement LED Color", TERMINAL_ALIGN_LEFT, LCD_COLOR_YELLOW);
ece210_lcd_add_msg("Right Increment LED Color", TERMINAL_ALIGN_LEFT, LCD_COLOR_GREEN);
sprintf(display_level,"CHANGE RED %i%%", PWM_MULT*red_pwm);
ece210_lcd_add_msg(display_level, TERMINAL_ALIGN_LEFT, LCD_COLOR_RED);

while(1)
{

  if( AlertButtons)                         // check to see if any push button was pushed.
  {
    AlertButtons = false;
    if( btn_right_pressed())                // if RIGHT pushed then change which LED will
    {                                       // be changed next.
        rotate_led(RIGHT);                  // call the rotate_led routine.
        something_changed = 1;              // RED, GRREN, BLUE is the right order
    }

    if( btn_left_pressed())                 // if LEFT pushed then change which LED will
    {                                       // be changed next.
        rotate_led(LEFT);                   // call the rotate_led routine.
        something_changed = 1;              // BLUE, GRREN, RED is the left order.
    }

    if( btn_up_pressed())                   // if UP then increase the selected color by
    {                                       // one count. Each count is 1/PWM_MAX in percent
        change_pwm(UP);                     // duty cycle.
        something_changed = 1;              // duty cycle rolls from 100% to 0%
    }

    if( btn_down_pressed())             // if UP then decrease the selected color by
    {                                   // one count. Each count is 1/PWM_MAX in percent
        change_pwm(DOWN);               // duty cycle.
        something_changed = 1;          // duty cycle rolls from 0% to 100%
    }

  }  // end AlertButtons

  pwm_time = NUM_MS_TO_WAIT;        // pwm_time in case you want to change this
  implement_pwm(pwm_time);          // implement the desired duty cycles for each
                                                    // color. The red LEDs 604 - 607 are also run
                                    // with the same duty cycle as the red color of
                                    // the RGB LED.

   if (something_changed)           // display which color changed and the new duty
   {                                // cycle in percent.
        if (led_to_change == RED)   // If we changed RED
        {
                sprintf(display_level,"CHANGE RED %i%%", PWM_MULT*red_pwm);
                ece210_lcd_add_msg(display_level, TERMINAL_ALIGN_LEFT, LCD_COLOR_RED);
        }
        if (led_to_change == GREEN) // If we changed GREEN
        {
                sprintf(display_level,"CHANGE GREEN %i%%", PWM_MULT*green_pwm);
```

```
                ece210_lcd_add_msg(display_level, TERMINAL_ALIGN_LEFT, LCD_COLOR_GREEN);
        }
        if (led_to_change == BLUE)              // If we changed BLUE
        {
                sprintf(display_level,"CHANGE BLUE %i%%", PWM_MULT*blue_pwm);
                ece210_lcd_add_msg(display_level, TERMINAL_ALIGN_LEFT, LCD_COLOR_BLUE);
        }
        something_changed = 0;                  // reset something_changed

    }  // end something_changed

  }  // end while (1)
} // end main

/*********************************************************************
 *
 *      functions
 *
 *********************************************************************/
void rotate_led(uint8_t direction) // change the color of LED to change
{
                                // 0 to 1 to 2 back to 0 (RIGHT)
if (direction == RIGHT)                                         // or 2 to 1 to 0 back to 2 (LEFT)
        {
                led_to_change = (led_to_change + 1)%3;
        }
        else
                                // %3 is the modulo 3 function in C syntax.
        {
                led_to_change = (led_to_change - 1)%3;
        }
}  // end rotate_led

void change_pwm(uint8_t direction)      // increase or decrease the amount of on time
{
                                        // for the LEDs. Duty cycle is xxx_pwm/PWM_MAX
if(led_to_change == RED)                                // if RED is to be changed
        {
                if (direction == DOWN)                  // move duty cycle down
        {
        if (red_pwm == 0)                               // if already 0 then set to PWM_MAX
                        {
                                red_pwm = PWM_MAX;
                        }
                        else    // otherwise decrease red_pwm by 1
                        {
                                red_pwm = ( red_pwm - 1);
                        }
                }
                else            // UP  increase red_pwm by 1 and roll back to 0
                {               // move duty cycle up
                        if (red_pwm == PWM_MAX)     // if already PWM_MAX then to 0
                        {
                                red_pwm = 0;
                        }
                        else            // otherwise increase red_pwm by 1 using
```

```
                {
                        red_pwm = ( red_pwm + 1);
                }
        }
}

if(led_to_change == GREEN)
{
        if (direction == DOWN)
        {
                if (green_pwm == 0)
                {
                        green_pwm = PWM_MAX;
                }
                else
                {
                        green_pwm = ( green_pwm - 1);
                }
        }
        else
        {
                if (green_pwm == PWM_MAX)
                {
                        green_pwm = 0;
                }
                else
                {
                        green_pwm = ( green_pwm + 1);
                }
        }
}

if(led_to_change == BLUE)
{
        if (direction == DOWN)
        {
                if (blue_pwm == 0)
                {
                        blue_pwm = PWM_MAX;
                }
                else
                {
                        blue_pwm = ( blue_pwm - 1);
                }
        }
        else    // direction is UP
        {
                if (blue_pwm == PWM_MAX)
                {
                        blue_pwm = 0;
                }
                else
                {
                        blue_pwm = ( blue_pwm + 1);
                }
        }
```

```
        }

}       // end change_pwm

void implement_pwm(uint8_t milliseconds)
{
uint8_t         red_on = 0;
uint8_t         green_on = 0;
uint8_t         blue_on = 0;
uint8_t         leds_on= 0;
uint8_t  i = 0;

// ***************** Duty cycle display using the red_leds ******************************

if(led_to_change == RED)                                        // Prepare red_led duty cycle color RED
        {
                if (red_pwm > 0)
                {
                        ece210_red_leds_write(ALL_ON);          // Turn on the red LEDs if red pwm is not
0.
                }
        }
        if(led_to_change == GREEN)                              // Prepare red_led duty cycle color
GREEN
        {
                if (green_pwm > 0)
                {
                        ece210_red_leds_write(ALL_ON);          // Turn on the red LEDs if green pwm is
not 0.
                }
        }
        if(led_to_change == BLUE)                               // Prepare red_led duty cycle color
BLUE
        {
                if (blue_pwm > 0)
                {
                        ece210_red_leds_write(ALL_ON);          // Turn on the red LEDs if blue pwm is
not 0.
                }
        }

        ece210_tiva_rgb_write(OFF);                             // turn OFF the RGB LED

        for(i = 0; i < PWM_MAX; i++)                            // run loop for PWM_MAX times
        {
                if (red_pwm > i)                        // if the pwm desired value is >= i keep RED LED ON
                {                                       // this keeps the LED ON for red_pwm/PWM_MAX *100
percent of time
                        red_on = RED_ON;
                }
                else                                    // else turn the RED LED OFF
                {
                        red_on = OFF;
                        if(led_to_change == RED)                        // if RED is to be changed
                        {
```

```
                                        if (red_pwm == i)                       // turn off red LEDs if red_pwm
= i.
                                        {
                                                ece210_red_leds_write(OFF);     // pwm the red LEDs the same
as the RGB
                                        }
                                }
                        }

                        if (green_pwm > i)          // same as RED
                        {
                                green_on = GREEN_ON;
                        }
                        else
                        {
                                green_on = OFF;
                                if(led_to_change == GREEN)              // if GREEN is to be changed
                                {
                                        if (green_pwm == i)             // turn off red LEDs if
green_pwm = i.
                                        {
                                                ece210_red_leds_write(OFF);     // pwm the red LEDs the same
as the GREEN
                                        }
                                }

                        }

                        if (blue_pwm > i)           // same as RED
                        {
                                blue_on = BLUE_ON;
                        }
                        else
                                {
                                        blue_on = OFF;
                                        if(led_to_change == BLUE)               // if BLUE is to be changed
                                                if (blue_pwm == i)                      // turn off red
LEDs if blue_pwm = i.
                                        {
                                                ece210_red_leds_write(OFF);     // pwm the red LEDs the same
as the BLUE
                                        }
                                }

                                }

                        leds_on = red_on + green_on + blue_on; // prepare the RGB leds_on with new states

                        ece210_tiva_rgb_write(leds_on);                                 // Write the value out to
the RGB LED

                        ece210_wait_mSec( milliseconds);        // sets amount of time to wait for each partial
                }                                               // pwm period. The pwm period is PWM_MAX *
milliseconds
}       // end implement_pwmThat is a lot of code. Let's look at the different sections.
```

In the function declarations section at the top of the code there are 3 functions that were written to make the main loop easier to follow.

```
rotate_led( direction);
change_pwm( direction);
implement_pwm( milliseconds);
```

The functions shield you from the code while letting you know what the code does. The function names strongly suggest what the function does. We are introducing functions to you for 2 main reasons.

1. When something gets done over and over again in your program it is cleaner and easier to understand the flow of the program with functions getting called instead of having the code in the function pasted into the program every time the function is called.
2. We want to demonstrate how easy it is to put a function into your program by declaring it above and writing the code below. This lets you see the code if you want to use it elsewhere or want to understand what is going on in the code that performs the function.

The global variables section creates variables or constants that can be "seen" used by all of the code in main and in the functions. Another thing you might notice is the #defines are used like last time. In this case we use some of the #defined variable to create other #define constants.

1. One of the main reasons we create #defines is it defines one spot in the program that allows us to change a constant and this change will ripple through the program when it is compiled without having to go through and find everywhere it is used.
2. Another reason to use #defines is the compiler does the work of multiplying and dividing things "off line" which doesn't take any processor effort to calculate the numbers you use in the code later.

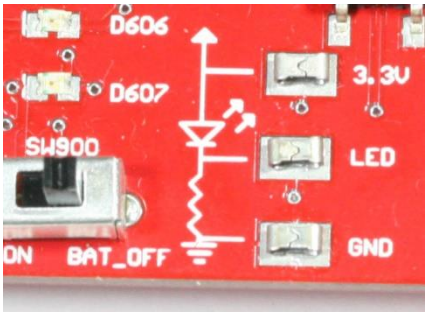
There are also 4 variables defined in the global section. These 4 get used by the functions. Having them global means they can be used by the functions without being part of the variables sent to the function between the (). This makes it easier to read the code as a beginner. Usually we don't use a lot of global variables this way.

Next we will look at the program. It is best to look at the flowchart first. This allows you to see what the program does without worrying about the details of the code that does it.

- After initialization is done we check to see if a button was pushed.
- If no button was pushed then implement the PWM as setup.
- If a button was pushed then see which one was pressed.
- Up and Down increase or decrease the Duty cycle of the PWM signal driving on color of the RGB LED.
- Right or Left sets which color will be changed with an Up and Down command.
- After the change implement the PWM.
- Display the new Color and PWM duty cycle.

7.3 Oscilloscope

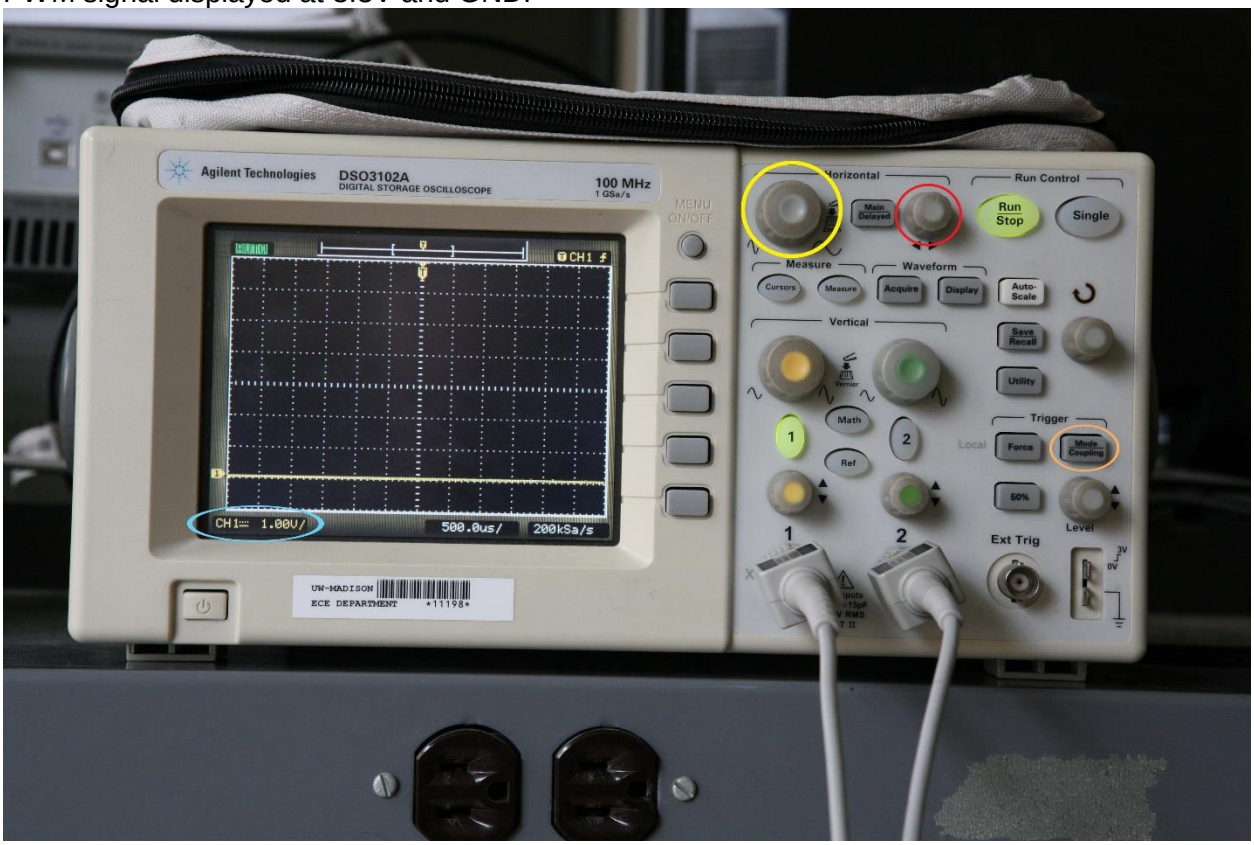
We are going to use the Oscilloscope to examine the PWM waveforms sent to the RED color of the RGB LED. We do this by sending the same signal to the RED LEDS D604-D607. The oscilloscope allows us to see what the PWM frequency or period is and what the duty cycle of the PWM signal is. Remember duty cycle is the ratio amount of time the LED is ON and the PWM period. Duty Cycle = (PWM ON time / PWM period) *100



Each measurement channel has a measurement probe, consisting of two connection points. One of them is a ground lead that is black in color, has an alligator clip terminal and is always connected to the ground of the circuit in which measurements are made. The other lead is the signal lead that has a hooked clip which is connected to signal pin to be measured. We will use the oscilloscope to observe the signal driving LED607 at the test points 3.3V and GND. The 3.3V test point has the PWM signal which goes from a logical 1 when the LED should be ON and a logical 0 when it is OFF.

If the duty cycle is anything other than 0% you can measure the waveform of the rectangle wave driving the LED. The PWM frequency or period tells one useful spec that you can use to set the Oscilloscope. The other useful spec is the voltages when the PWM signal is a 1 and 0.
The logical 0 voltage is ground or 0V and the logical 1 voltage is about 3V.
The code should be loaded and running. You should notice that the PWM frequency is listed on the LCD screen. For this first test it is set to 100Hz. Don't be surprised if it isn't exactly 100Hz when measured. That means the PWM period is 10mS.

Using the specs of the PWM signal we can setup the scope to give a good view of the duty cycle of the signal driving the LEDs. It is possible to see the duty cycle of the all color of the RGB LED. Whichever color has been selected to change the PWM value for is the color of the PWM signal displayed at 3.3V and GND.



- The period is 10mS so I suggest you set the horizontal time base to 2mS/Div which gives 20mS across the screen as there are 10 divisions across the screen. You adjust this by rotating the large knob (Yellow) just under the Horizontal label on the face of the scope.
- Press the Mode/Coupling button (Orange). Make sure the Channel for triggering is the one you have connected to 3.3V test point 1 or 2. Set the trigger mode to Auto.
- Make sure the channel number is illuminated for the channel you are using. Set the volts/division is set to 1.00V (Blue).
- Make sure the Run/Stop button is on. There should be couple of PWM periods of signal on the screen. If so push the 50% button in the Trigger section. Press Mode/Coupling again and select Normal Mode triggering. This should make the screen hold still. You know the scope is still measuring because it will blink TGD on the screen.
- If you rotate the Horizontal Position knob you can put a full period in the middle of the screen (Red).
- Time to measure the PWM period waveform. Use Cursors to measure the PWM period/frequency. Put the cursors in Manual Time mode and move the A cursor to the beginning of the period and move the B cursor to the end of the period.
- You could also use the Measure button. Select the correct channel number and choose Freq. or Period. There is also a Duty Cycle button. Pick the +Duty Cycle choice.

You need to acquire the PWM period or frequency and the trace of the Duty cycle you measured. Pick any duty cycle you want as long as it is non-zero.

We can now capture the scope image using the computer. Now, let us transfer these waveforms to the computer using the steps illustrated in the following procedure.

Acquiring the oscilloscope waveform data into a computer.
It is convenient to see the desired waveforms generated on the oscilloscope on the computer and save it for later references. This can be done by setting up a communication system between the oscilloscope and the computer. To establish this connection, we make use of the DSO3000 Series oscilloscope Scope Connect software. This software has already been installed in the lab computers. You will find a shortcut to this application on the computer desktop.
To view the waveforms and save it on the computer, follow the steps below:

1. Before setting up the oscilloscope-desktop connection, make sure you have connected the Oscilloscope probes of Channel 1 or Channel 2 to the Bucky board as described above and have the PWM duty cycle measurement on screen.
2. Now, make sure the oscilloscope is physically connected to the computer. This is done using a standard USB A/B cable.
3. Open the DSO3000 application. You will see three windows Waveform 0, Data 0 and Measurement 0. as shown in Figure 7.6
4. Click on the ‘connect to oscilloscope’ icon (see Figure 7.6), top left button. The oscilloscope should now be connected to the computer.
5. To see the waveforms generated on the oscilloscope, go to the Waveform 0 window and click on refresh. The waveform being displayed on the oscilloscope will now be seen on the computer screen.

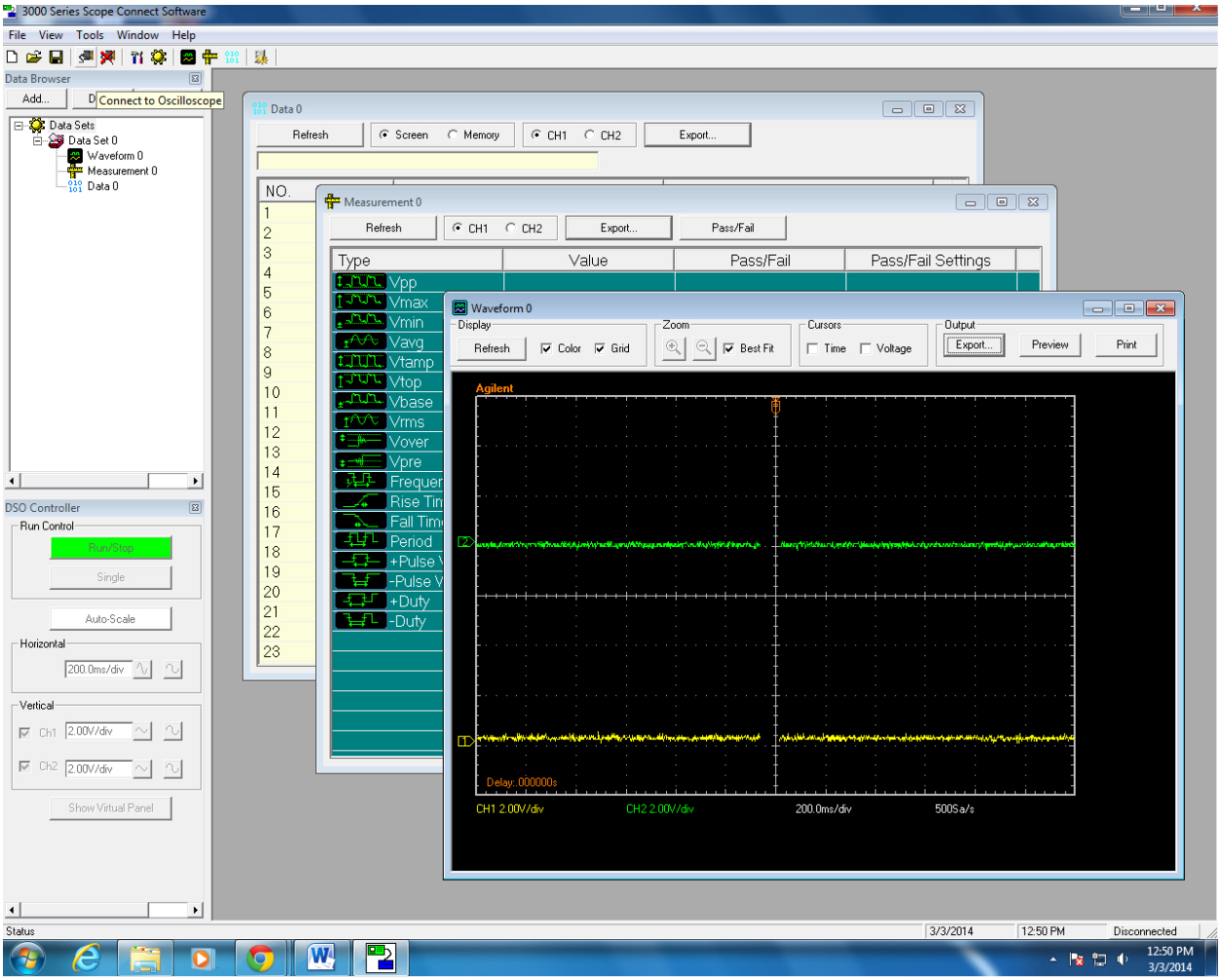


Figure 7.6 Scope Connect Software window.

6. To save this waveform, click on export and save it to the desired location. If two channels are being used, both the channel waveforms will be displayed. The file generated will be X.bmp file.
7. To view other parameters of the waveform such as V_{pp}, V_{max}, V_{avg}. etc, click on the Measurement 0 window and refresh. Here, you have to select the channel you are using. You can save this data by clicking on export and then saving it to the desired location. The file generated is an X.xls file.
8. To view the time series points of the wave, click on Data 0 window and refresh it. Here again, the channel has to be selected. The data can be saved by clicking on export and then saving it in the desired location. The file generated is an X.xls file.
9. Note: When the oscilloscope is connected to the computer, you will not be able to make any changes to waveform using the oscilloscope front panel. To make changes/vary settings in the oscilloscope, you need to disconnect the oscilloscope-computer connection with the icon for that.

Create 4 different colors by varying the duty cycles of each frequency. Don't use Red, Green, Blue, OFF or White. Measure the duty cycle for each color you chose.

Color 1 _____ R _____% G _____% B _____%

Color 2 _____ R _____% G _____% B _____
Color 3 _____ R _____% G _____% B _____
Color 4 _____ R _____% G _____% B _____

Now it is time to change the PWM frequency. We are using 100Hz or so now. No one should have been able to see flickering or blinking in the LEDs.
We are going to change the Frequency to 10Hz. Everyone should be able to see flickering or blinking at this frequency.
To change the PWM frequency to 10 Hz you need to change one line in the program. Find this spot in the main.c file.

```
#define NUM_MS_TO_WAIT    1              // PWM period is = NUM_MS_TO_WAIT * PWM_MAX in
#define PWM_MAX           10             // milliseconds. PWM_MAX=100 is 100ms PWM period.
#define PWM_MULT          100/PWM_MAX          // PWM_MAX sets the number of levels of PWM
#define PWM_START         PWM_MAX/2            // from 0 to 100%
#define PWM_FREQ          1/(0.001*PWM_MAX)
```

You need to change the number 10 after PWM_MAX to the number 100. That is it. This change will ripple through the program and change everything needed to cause the PWF frequency to change.

Measure the PWM frequency with the scope. PWM frequency = _____ Hz

Find the PWM frequency where you think it starts to blink with the scope by changing PWM_MAX starting at 10 and increasing the number. You can increase by any amount you want but don't go to fast or you will have to go back.

PWM frequency of seeing the blinking or flashing.

7.4 After the lab

How many distinct colors may be displayed on the RGB LED while using the stock program provided to you? Remember there are 11 levels, including 0, in the stock program.

How many distinct duty ratio values would you need to have for each of the R, G and B components to get a palette of 27,000 different colors?