Autoranging Flashlight

Damon Buxton, Electrical Engineering

March 22, 2019

University of Evansville
Table of Contents

I. Introduction

II. Background

III. Project Design
   A. Overview
   B. Distance Sensor
   C. Software design
   D. FET Drivers
   E. LED’s
   F. Power
   G. Case Design
   H. Standards and Constraints

IV. Results

Appendix A

References

List of Figures

1. Sensor output representation
2. FET driver circuit

3. Left: Cree XP-L. Right: Cree XP-L HI

4. Samsung INR18650-30Q 3000mAh Li-Ion Battery

5. Block Diagram

6. 3D rendering of case

Acknowledgements

I would like to thank my dad, Brad Buxton, for allowing me to use his 3D printer, and for helping me with the machining of the aluminum heatsinks.
I. Introduction

Flashlights are very useful tools, and flashlight technology has come a long way in a short amount of time. LED flashlights were only introduced about 20 years ago. Since then, hundreds of companies have created thousands upon thousands of different flashlight designs. Today, lights the size of your index finger can easily outperform lights the size of your forearm from 20 or 30 years ago. It is still the case though that most flashlights are very specific to one particular task. That is, if someone needs both a spot light and a floodlight, they must use two different lights. The autoranging flashlight combines the functionality of a spotlight and a floodlight.

II. Background:

Currently the only way to carry both a spotlight and a floodlight (with decent color rendering and a pleasant beam shape), is to carry two separate lights. The autoranging flashlight seamlessly combines the functionality of a spotlight and a floodlight. It automatically chooses the most appropriate proportion of each one, without the user having to even think about it. There is nothing else like it currently available on the market. It could be extremely useful for police, military, security personnel, hunters, campers, homeowners, etc.

Minimum requirements:

- The user interface will be intuitive, and feature variable brightness
Maximum brightness setting will be at least 1000 lumens

Useful “blinky” modes such as strobe and SOS

LEDs with a minimum color rendering index (CRI) of 80

III. Project Design

A. Overview

Distance measuring is accomplished through the use of a LiDAR laser distance sensor. This sensor measures a distance up to 40 meters, and outputs the distance as a PWM signal, with a pulse width ranging from 10 μs to 40 ms. This signal is then read into an ARM Nucleo STM32F446 microprocessor[9], which interprets the signal, and uses it to determine a ratio between two PWM outputs. These two PWM outputs are fed into two independent low-resistance FET drivers, which drive two high output LED’s. One LED has a dome, and is placed within a small reflector with a diffuse surface, creating a broad, floody beam. The other LED is dedomed, and is placed within a much larger reflector with a smooth surface, creating a very far-reaching spotlight beam. The ARM Nucleo board also controls the user interface, allowing the user to turn the light on/off, and access other features, such as changing the overall brightness, with simple button presses.

B. Distance Sensor
Distance measuring is done using the LIDAR-Lite V3 sensor made by Garmin[5]. Lidar, which stands for “Light Detection And Ranging”, works on the same principle as radar and sonar. Lidar measures distance by estimating the time delay from light emission to light detection. This sensor in particular uses a 905 nm infrared laser as it’s light source. The sensor outputs measurements as a PWM waveform. In this case, the PWM signal is not like a normal PWM signal one might expect. Measurements do not happen on their own; each measurement must be triggered by pulling the sensor’s “mode control pin” low. Pulling this line low for a very short amount of time prompts the sensor to take a measurement and return a high pulse in which every 10 µs = 1 cm. The low pulse is sent through a 1 kΩ resistor, while the received high pulse is not. Both send and receive signals are done over the same line. This results in a PWM signal that ranges from about 1.1 v to 3.3 v, whereas a normal PWM signal would range from 0 v to 3.3 v. This means that instead of passing the PWM signal through a low pass filter to get an analog value, which can be read into an ADC, we must read the PWM signal directly into the ARM Nucleo’s ADC, and use a function to actually measure the pulse width. As it turns out, it is actually easier to pull the mode control pin low indefinitely, which results in a variable frequency square wave. The pulse width of this square wave can be read into the ADC in the same manner that the PWM signal would be read. The program can start reading at any point in time on the square wave. If the program comes in during a high pulse, it does nothing for the remaining duration of the high pulse, and continues to do nothing during the entirety of the following low pulse. The program then increments the pulse width variable during the next high pulse. A pictorial description can be seen in Figure 1. Since voltages of 1.1 v and 3.3 v correspond to ADC values of 1365 and 4096 respectively, the function used to measure the pulse width looks something like this:
void TakeAMeasurement(void)
{
    PulseWidthVariable = 0;
    while(ADCvalue > 2700)
    {
        //do nothing
    }
    while(ADCvalue < 2700)
    {
        //do nothing
    }
    while(ADCvalue > 2700)
    {
        PulseWidthVariable++;
        Delay();
    }
}

Since the sensor’s output is somewhat erratic, an averaging function is used to smooth the output. It was found that averaging every 6 measurements yielded a smooth output while still maintaining an acceptably fast speed.
C. Software design

The ARM Nucleo takes the pulse width value and uses it to adjust the output PWM ratio between the two LED’s. This function looks something like this, where “brightness” is a number between 5 and 100, with 5 being the lowest brightness setting, and 100 being the highest:

```c
void output(void)
{
    outputvalueA = PulseWidthVariable*(brightness/100);
    outputvalueB = (4096-PulseWidthVariable)*(brightness/100);

    Spot_LED_PWM_Register = outputvalueA; //4096=full brightness
    Flood_LED_PWM_Register = outputvalueB; //4096=full brightness
}
```

The ARM Nucleo also handles the user interface, allowing the user to turn the light on/off, change the overall brightness level, activate strobe or SOS mode, and
activate “distance hold” mode. Distance hold mode allows the user to lock the current spot/flood ratio, effectively disabling the autoranging feature. The on/off is accomplished with a latching pushbutton, which inputs either a 1 or a 0 to the ARM Nucleo, corresponding to “on” or “off”, respectively. Doing it this way, instead of just cutting power to the whole system, allows the light to return to the same brightness level it was at when it was turned off. It also allows the back-lit pushbuttons to stay illuminated for a period of time after the light has been turned off.

The MODE button is a momentary pushbutton. The MODE button is used for adjusting brightness, activating strobe/SOS, and activating distance hold mode. When the user presses and holds the MODE button, the brightness level decreases until the user releases the MODE button, or until lowest brightness levels is reached. If the lowest brightness level is reached, the main and backlight LEDs will blink once to indicate the lowest brightness level has been reached, and the backlight LEDs will flash continuously until the user releases the button, or until strobe/SOS mode is activated. The user can then ramp the brightness back up the same way.

To activate strobe, the user simply ramps the brightness up to max, then continues to hold the MODE button for 2 seconds. After 2 seconds, strobe mode is activated. If the user releases the MODE button within 1.5 seconds of strobe mode being activated, strobe mode will turn off after the button is released. This momentary strobe function is to make the user interface more friendly, in case the user accidentally holds the MODE button for too long, but does not wish to enter into strobe mode. If the user holds the MODE button for longer than 1.5 seconds after strobe mode has been activated, strobe mode will stay active after the MODE button is released. To turn off strobe mode, the user can either do a single press of the MODE button, or switch the light off and back on again. The procedure for accessing SOS mode is the same as
accessing strobe mode, except the user must instead ramp the light down to the lowest brightness and continue holding the MODE button. For the light to stay in SOS mode after the MODE button has been released, the user must wait until the 4th SOS blink before releasing the MODE button.

To activate the distance hold feature, the user must simply do a single quick press of the MODE button. To deactivate the distance hold feature and go back to autoranging mode, the user must do another single quick press of the MODE button.

Li-ion batteries can be damaged if they are over-discharged. The light features a low voltage protection (LVP) mode to prevent this from happening. The 5 v regulator used for powering the ARM Nucleo board and distance sensor has an output pin that is pulled low when battery voltage drops below 3.2 v. This pin is wired to an input pin on the ARM Nucleo board. Whenever the battery voltage drops below 3.2 v, the brightness is decreased by 5% at a time until the battery voltage rises back above 3.2 v. The light still functions normally (except with decreased brightness) so long as the battery voltage hasn’t forced the brightness below 5%. Once the brightness is forced below 5%, LVP mode is activated. When LVP mode is activated the spotlight LED is turned off, and the flood LED is turned on at 5% brightness. This is done to keep the user from being stranded in the dark when the batteries die. The distance sensor and backlight LEDs turn off to decrease current draw. No adjustments can be made using the MODE button while LVP mode is active. If the user presses the mode button while the light is turned on and in LVP mode, the backlight LEDs will simply flash 5 times and turn off again. Since Li-ion batteries often recover relatively slowly from voltage sag, LVP mode is deactivated when the light is switched off. Often times if LVP mode is activated and the user switches the light off for a couple minutes, this
will give the batteries enough time to recover, allowing the user a few more minutes
on a lower brightness level before LVP mode is activated again.

Our eyes do not perceive brightness changes linearly. Our eyes actually perceive
brightness in a somewhat logarithmic fashion. To account for this, an exponential
correction factor must be used. After experimentation, the following equation was
found to give the best results.

\[
OutputValue = \frac{4096(200^{\frac{4096}{199}} - 1)}{199}
\]

The light also features a timed stepdown to protect against overheating and
unnecessary battery drain if the light is accidentally turned or left on. After 5 minutes
of being at a brightness of 75% or above, the light steps down to 25% brightness. The
user can immediately ramp the brightness back up if desired. After a stepdown, the
next press-and-hold of the MODE button will automatically result in a ramp-up.

D. FET Drivers

The two PWM signals outputted from the ARM Nucleo are not powerful enough
to drive the high-output Cree XP-L and XP-L HI LED’s, which can draw upwards of
3 amps a piece.\[^4\] To drive the LED’s, two independent FET drivers are used. The
MOSFETs used (NXP PSMN2R0-30YLDX) are low-resistance N-channel
MOSFETs.\[^7\] The driver design is shown in Figure 2.
E. LED’s

The two LEDs used are a Cree XP-L, and a Cree XP-L HI. Both LED’s have a color temperature of 5000k, and a Color Rendering Index (CRI) of 80.\textsuperscript{[4]} The XP-L is used with a 20mm orange peel reflector, which creates a diffuse and and floody beam. The XP-L HI is used with a 42mm smooth reflector, creating a very far-reaching spotlight beam. Due to the amount of heat these LED’s are capable of producing, they are mounted on copper-core MCPCB’s, which are attached to aluminum heatsinks within the enclosure.
The light is powered by four 18650 batteries, running in parallel. The parallel configuration allows the light to run on 1, 2, 3, or 4 batteries, which may be useful in emergency situations. When using 2, 3, or 4 batteries, using matched sets is very important for safe operation. While the LED’s can be powered directly from battery(+), the ARM Nucleo and LIDAR-Lite V3 sensor require a 5 v power supply. Since Li-ion battery voltage ranges from 3 v-4.2 v, a boost converter is used to supply the required 5 v. The boost converter chosen, the Adafruit PowerBoost 500[^6], also has a built in low-voltage protection feature, which pulls a pin low when battery voltage drops below 3.2 v. This pin is connected to and read by the ARM Nucleo.
Figure 4. Samsung INR18650-30Q 3000mAh Li-Ion Battery[8]

Figure 5. Block Diagram

G. Case Design
The case was designed using Autodesk Fusion 360 and 3D printed in PETG plastic. PETG plastic is both recyclable and biodegradable, so it is much better for the environment than most other plastics.[1] PETG also poses virtually no health risks to the user, because it is also one of the safest plastics. It is not known to leach any chemicals that are suspected of causing cancer or disrupting hormones.[2] If the light were to ever be mass-produced, injection molding would be a much more efficient process than 3D printing. However, a 3D printed case is perfectly acceptable and much more practical for use in this prototype. The case was designed to meet the following requirements:

- Easy and comfortable to hold
- Access to batteries is easy and does not require any tools
- Case is as compact as is reasonably possible
- Design is aesthetically pleasing

Figure 6. 3D rendering of case
Aside from ensuring that the user interface works as it is supposed to (i.e. user can turn light on/off, adjust brightness, access special modes, etc) three tests were conducted to ensure the light functions as it should. The first test was to stand 1 meter from a wall and, with the light turned on, slowly back away from the wall until the light fully transitioned from flood to throw. This showed that the light can transition between flood and throw smoothly. The next test was to stand in an open area, and alternate between shining the light at a target that is very far away, to a target that is very close. This test showed that the light can transition between flood and throw quickly. The third test was to qualitatively compare the brightness of the autoranging flashlight to two lights (one flood and one spot) whose brightness are known to be 1000 lumens. This test showed that the maximum output of the light is approximately 1000 lumens.

The light is very safe when used properly. Steps taken to improve safety include: low voltage protection mode, timed stepdown, non-toxic PETG plastic, and radiused edges on the case. On-board charging was considered, but it was decided that charging the batteries using a separate, dedicated charger was much safer. The biggest safety concern would be battery fire that could result if the user inserts one of the batteries backwards, causing a short circuit. Thermal fuses could easily be added to the production model, and would greatly reduce the possibility of this happening. This would require that double sided copper clad PCB be used for the battery contact boards instead of the solid copper plates that were used on the prototype. Alternatively, a physical non-conductive barrier could be used at the positive terminal,
but this would mean that only button-top 18650 batteries could be used; common flat-top 18650 batteries would not be compatible.

Due to the simple nature of the product, it will cause no considerable political impact.

IV. Results

The end product is an LED flashlight that uses a distance sensor to detect how far away the target object is, and adjust the beam profile accordingly. The user interface is intuitive, and features variable brightness. Maximum brightness setting is at least 1000 lumens. The LED’s have a minimum Color Rendering Index (CRI) of 80. the case is easy and comfortable to hold, and is as compact as is reasonably possible. Access to batteries is easy and does not require any tools. Lastly, the final product is aesthetically pleasing.
Appendix A

Program code:

//Damon Buxton
//Senior Project - Autoranging Flashlight

#include "stm32f446.h"
#include <math.h>

int flag;
int ADCvalue; //value read from ADC input
int brightness=100; //brightness, initialized to 100% when battery inserted
long PulseWidth; //measurement of the high pulse width
long AddedPulses = 0; //running total pulse width that have been added to compute the average
long PulseWidthAverage; //averaged pulse width
int TimesAdded = 0; //number of pulses that have been added to compute the average
int set = 0; //variable set to 1 during button press, set back to 0 when button released
int hasblinked = 0; //this variable = 1 if light has already blinked during button press-and-hold, signaling the end of brightness range has been reached
int updown = 0; //0 = next press-and-hold will ramp down. 1 = next press-and-hold will ramp up
int HoldDistance = 0; //if this variable = 1, the light will hold its current distance measurement until it is a 0 again
long StepdownCounter = 0; //this variable is for the time stepdown,
int LVPHasFlashed = 0; //this variable = 1 if the backlight LED's have flashed since the light has been turned on while in LVP mode
int LVPHasFlashedTheFirstTime = 0; //this variable = 1 if the backlight LED's and main LED's have flashed since the light first entered LVP mode. it is only reset if the batteries are removed, thus resetting the microcontroller
int LVPactive = 0; //this variable = 1 if the light is in LVP mode (PB6 is low)
int StrobeCounter = 0;
int StrobeStayOn = 0;
int StrobeStayOnCounter = 0;
int SOSCounter = 0;
int SOSStayOn = 0;
double outputvalueA; //variable for throw LED output
double outputvalueB; //variable for flood LED output
void delay(unsigned char x);          //delay function of x=1usec
void longdelay(unsigned char x);      //delay function of x=1msec
void superlongdelay(unsigned char x); //delay function of x=1sec
void readADC(void);                   //function that reads ADC value from sensor
void output(void);                    //function that outputs PWM value to LEDs
void ChangeBrightness(void);          //function that allows user to change overall brightness
void CheckForButtonPress(void);       //function to check is mode button is being pressed
void TIM3_IRQHandler(void);
void Strobe(void);
void SOS(void);
void MomentaryStrobe(void);
void MomentarySOS(void);

void delay(unsigned char x) // Creates a software delay of x microsec (delay(1)=approx 40usec)
{
    unsigned char i;
    int tmp;
    RCC_AHB1ENR |= 1;
    RCC_APB1ENR |= 1;
    NVICISER0 |= (1<<28);
    TIM2_DIER |= 1;
    TIM2_DIER |= (1 << 6);
    TIM2_CR1 |= (1 << 7);
    TIM2_PSC |= 0; //15
    TIM2_ARR = 160; //16000 = 1msec
    TIM2_CR1 |= 1;
    TIM2_EGR |= 1;
    tmp = 0;

    for(i=0;i<x;i++) // Loops the 1 usec delay x times
    {
        tmp= ~tmp;
        flag = 1;
        while(flag == 1);
        TIM2_CR1 |= 1;
    }
}
void longdelay(unsigned char x) // Creates a software delay of x msec
{
    unsigned char i;
    for(i=0;i<x;i++) // Loops the 1 msec delay x times
    {
        delay(100);
    }
}

void superlongdelay(unsigned char x) // Creates a software delay of x sec
{
    unsigned char i;
    for(i=0;i<x;i++) // Loops the 1 sec delay x times
    {
        longdelay(100);
    }
}

void TIM2_IRQHandler()
{
    flag = 0;
    TIM2_SR &= 0xFFFE;
}

void readADC(void)
{
    PulseWidth = 0;

    ADC1_CR2 |= 0x40000000;     // Bit 30 does software start of A/D conversion

    ADCvalue = ((int)ADC1_DR);    // y = 0-4096 (12 bit)

    while(ADCvalue > 2700)        //may come in during the middle of a high pulse, so wait until it
    passes (was 1800)
    {
        ADC1_CR2 |= 0x40000000;    // Bit 30 does software start of A/D conversion
        ADCvalue = ((int)ADC1_DR); // y = 0-4096 (12 bit)
    }

    while(ADCvalue < 2700)        //wait until the low pulse passes
ADC1_CR2 |= 0x40000000; // Bit 30 does software start of A/D conversion
ADCvalue = ((int)ADC1_DR); // y = 0-4096 (12 bit)

while(ADCvalue > 2700) //read the width of this high pulse
{
  ADC1_CR2 |= 0x40000000; // Bit 30 does software start of A/D conversion
  ADCvalue = ((int)ADC1_DR); // y = 0-4096 (12 bit)
  PulseWidth = PulseWidth+1;
  delay(1); //this delay will determine the resolution of the measurements
}

PulseWidth = PulseWidth*20; //multiplication factor for determining maximum desired measure distance. The larger the number, the shorter the distance before throw LED is maxed out and flood LED is 0.

if(PulseWidth > 4096) //keeps the pulse width below 4096 (it naturally can’t be below 0, so don’t need to worry about that)
{
  PulseWidth = 4096;
}

if(PulseWidth > 200) //ignore low readings (ambiguous readings, readings past the sensor’s range, and subsequently, readings below ~3in)
{
  AddedPulses = AddedPulses + PulseWidth;
  TimesAdded = TimesAdded + 1;
  if(TimesAdded == 6) //number of pulse width measurements to be averaged (low # = fast adjustment speed but unstable output, high # = slow adjustment speed but stable output
  {
    PulseWidthAverage = AddedPulses/TimesAdded; //compute average
    TimesAdded = 0;
    AddedPulses = 0;
  }
}

void output(void) //this functions outputs to the main LEDs
{
int CorrectionFactor = 200; //higher number = more exponential correction curve

if(PulseWidthAverage < 10)  //this sets the minimum possible brightness for ranging (not the user adjustable brightness) (cuts off to 0), because below a certain value, PWM begins to get flickery
{
    PulseWidthAverage = 0;
}

outputvalueA = PulseWidthAverage*brightness/100;  //factors the user adjustable brightness into the total brightness
outputvalueB = (4096-PulseWidthAverage)*brightness/100; //factors the user adjustable brightness into the total brightness, as well as sets flood LED opposite of throw LED

outputvalueA = 4096*(pow (CorrectionFactor, (outputvalueA / 4096)) - 1)/(CorrectionFactor-1); //correction factor for visual linearity
outputvalueB = 4096*(pow (CorrectionFactor, (outputvalueB / 4096)) - 1)/(CorrectionFactor-1);

outputvalueA = 1.1*outputvalueA - 50; //other range adjustments

if(brightness < 30)
{
    if(outputvalueB < 5)
    {
        outputvalueB = 5;
    }
}

if(outputvalueA > 4096)   //keeps outputvalueA at or below 4096
{
    outputvalueA = 4096;
}

if(outputvalueB > 4096)   //keeps outputvalueB at or below 4096
{
    outputvalueB = 4096;
}

//-------------------------------
// This is a test function to see if I'm getting full brightness at each end of the spectrum
if((outputvalueA > 4000) || (outputvalueB > 4000))
{
    // turn off backlight LEDs
    GFIOC_ODR &=(~1<<0);
    longdelay(50);
    // turn on backlight LEDs
    GFIOC_ODR |= 1<<0;
}

TIM3_CCR1 = outputvalueA; // 4096=full brightness  PC6 Throw
TIM3_CCR2 = outputvalueB; // 4096=full brightness  PC7 Flood

void ChangeBrightness(void)// this function allows the user to change brightness
{
    while(GPIOB_IDR & 1<<9)     // while mode button is being pressed
        {

            // Ramp Down
            if(updown == 0)           // if the light is to be ramped down this time
            {
                brightness=brightness-1; // decrease the brightness
                if(brightness <= 5)      // this sets the lowest brightness the user can set
                {
                    brightness = 5;
                    if(hasblinked == 0)   // this = 1 when if the main LEDs have flashed once, indicating
                                     // the user has reached the lowest brightness
                    {
                        brightness = 0;     // low brightness of flash
                        output();           // duration of flash
                        brightness = 5;
                        hasblinked = 1;     // don't do this IF statement again until the next press-and-hold
                    }
                }
            }

            // this blink the backlight LED's until the button is released when the user has reached
            // the highest/lowest brightness
            if(SOSCounter<=10)
            {
            
            // top level of the page
longdelay(100);      //duration of blink
GPIOC_ODR &= ~(1<<0); //turn off backlight LEDs
output();
longdelay(100);      //duration of blink
GPIOC_ODR |= (1<<0);  //turn on backlight LEDs
output();
SOSCounter++;
}

if(SOSCounter==10)
{
  while(GPIOB_IDR & 1<<9)     //while mode button is being pressed
  {
    MomentarySOS();
  }
}

longdelay(15);   //this delay (15 msec) time determines the ramp down speed
//readADC();      //does not read sensor during a brightness change, so flood/throw ratio remains the same until button is released
output();
}

//Ramp Up
if(updown == 1)     //if the light is to be ramped up this time
{
  brightness = brightness+1;
  if(brightness >= 100)   //highest brightness (100)
  {
    brightness = 100;
    if(hasblinked == 0)   //this = 1 when if the main LEDs have flashed once, indicating the user has reached the highest brightness
    {
      brightness = 0;      //low brightness of flash
      output();
      longdelay(25);       //duration of flash
      brightness = 100;
      hasblinked = 1;      //don't do this IF statement again until the next press-and-hold
if(StrobeCounter<=10)
{
  longdelay(100);  //duration of blink
  GPIOC_ODR &= ~(1<<0); //turn off backlight LEDs
  output();
  longdelay(100);  //duration of blink
  GPIOC_ODR |= (1<<0); //turn on backlight LEDs
  output();
  StrobeCounter++;
}
if(StrobeCounter==10)
{
  while(GPIOB_IDR & 1<<9)     //while mode button is being pressed
  {
    outputvalueA = 4096;
    outputvalueB = 4096;
    TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
    longdelay(70);      //duration of blink
    outputvalueA = 0;
    outputvalueB = 0;
    TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
    longdelay(70);      //duration of blink
  }
  if(StrobeStayOnCounter < 100)
  {
    StrobeStayOnCounter++;
  }
  if(StrobeStayOnCounter > 10)
  {
    StrobeStayOn = 1;
  }
}
longdelay(15);           //this delay (15 msec) time determines the ramp up speed

//readADC();             //does not read sensor during a brightness change, so flood/throw ratio
remains the same until button is released
output();

set = 1;                  //= 1 if the button has been pressed
}

hasblinked = 0;            //resets hasblinked, allowing main LEDs to flash once to indicate
highest/lowest brightness reached the next time the user does a press-and-hold
StrobeCounter = 0;
SOSCounter = 0;

if(StrobeStayOnCounter > 10)
{
    StrobeStayOn = 1;
}

while((StrobeStayOn == 1)&&(GPIOB_IDR & 1<<8))
{
    Strobe();
}

while((SOSStayOn == 1)&&(GPIOB_IDR & 1<<8))
{
    SOS();
}

if((updown == 0) && (set == 1)) //after the button press, if it was a ramp down, set to ramp up next time
{
    updown = 1;
    set = 0;
}
if((updown == 1)&&(set == 1)) //after the button press, if it was a ramp up, set to ramp down next time
{
    updown = 0;
    set = 0;
}
}

void CheckForButtonPress(void) //this function checks to see if the mode button is being pressed
{
    if(GPIOB_IDR & 1<9)          //if the mode button is being pressed
    {
        superlongdelay(3);       //this delay is for waiting to see if a short press or a press-and-hold
                              //has been entered
        if((GPIOB_IDR & 1<9)&&(StrobeStayOn == 0)&&(SOSStayOn == 0))         //if the button is still being
                              //pressed after the delay, indicating a press-and-hold
        {
            ChangeBrightness(); //Change Brightness
        }
    } else                         //if the button has been released during the delay, indicating a single
click
    {
        if((StrobeStayOn == 0)&&(SOSStayOn == 0))
        {
        }
        else                         //if HoldDistance was l, make it a 0
        {
            HoldDistance = 0;        //don't hold distance
        }
    }
}

if(StrobeStayOn == 1)
{
    StrobeStayOn = 0;
    StrobeStayOnCounter = 0;
void Strobe(void)
{
    outputvalueA = 4096;
    outputvalueB = 4096;
    TIM3_CCR1 = outputvalueA;  //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB;  //4096=full brightness  PC7 Flood
    longdelay(70);            //duration of blink
    CheckForButtonPress();
    outputvalueA = 0;
    outputvalueB = 0;
    TIM3_CCR1 = outputvalueA;  //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB;  //4096=full brightness  PC7 Flood
    longdelay(70);            //duration of blink
    CheckForButtonPress();
}

void SOS(void)
{
    int i;
    int j;
    for(i=0;i<3;i++)
    {
        outputvalueA = 4096;
        outputvalueB = 4096;
        TIM3_CCR1 = outputvalueA;  //4096=full brightness  PC6 Throw
        TIM3_CCR2 = outputvalueB;  //4096=full brightness  PC7 Flood
    }
for (j=0; j<2; j++)
{
    CheckForButtonPress();
    superlongdelay(1);
    if((SOSStayOn == 0) || ((GPIOB_IDR & 1<<8) == 0))
    {
        SOSStayOn = 0;
        break;
    }
}

outputvalueA = 0;
outputvalueB = 0;
TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood

for (j=0; j<10; j++)
{
    CheckForButtonPress();
    superlongdelay(1);
    if((SOSStayOn == 0) || ((GPIOB_IDR & 1<<8) == 0))
    {
        SOSStayOn = 0;
        break;
    }
}
for(i=0;i<3;i++)
{
    outputvalueA = 4096;
    outputvalueB = 4096;
    TIM3_CCR1 = outputvalueA; //4096=full brightness PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness PC7 Flood

    SOSStayOn = 1; //keep SOS on after mode button released

    for(j=0;j<6;j++)
    {
        CheckForButtonPress();
        superlongdelay(1);
        if((SOSStayOn == 0)||((GPIOB_IDR & 1<<8) == 0))
        {
            SOSStayOn = 0;
            break;
        }
    }

    outputvalueA = 0;
    outputvalueB = 0;
    TIM3_CCR1 = outputvalueA; //4096=full brightness PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness PC7 Flood

    for(j=0;j<3;j++)
    {
        CheckForButtonPress();
        superlongdelay(1);
        if((SOSStayOn == 0)||((GPIOB_IDR & 1<<8) == 0))
        {
            SOSStayOn = 0;
            break;
        }
    }
}
for (j=0; j<10; j++)
{
    superlongdelay(1);
    if ((SOSStayOn == 0) || ((GPIOB_IDR & 1<<8) == 0))
    {
        SOSStayOn = 0;
        break;
    }
}

for (i=0; i<3; i++)
{
    outputvalueA = 4096;
    outputvalueB = 4096;
    TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
    for (j=0; j<2; j++)
    {
        CheckForButtonPress();
        superlongdelay(1);
        if ((SOSStayOn == 0) || ((GPIOB_IDR & 1<<8) == 0))
        {
            SOSStayOn = 0;
            break;
        }
    }
    outputvalueA = 0;
    outputvalueB = 0;
    TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
    for (j=0; j<2; j++)
    {
        CheckForButtonPress();
        superlongdelay(1);
    }
}
if((SOSStayOn == 0) || ((GPIOB_IDR & 1<<8) == 0))
{
    SOSStayOn = 0;
    break;
}
}
}

for(j=0;j<15;j++)
{
    CheckForButtonPress();
    superlongdelay(1);
    if((SOSStayOn == 0) || ((GPIOB_IDR & 1<<8) == 0))
    {
        SOSStayOn = 0;
        break;
    }
}

void MomentarySOS(void)
{
    int i;
    for(i=0;i<3;i++)
    {
        if((GPIOB_IDR & 1<<9) == 0)
            {break;}
        outputvalueA = 4096;
        outputvalueB = 4096;
        TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
        TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
        superlongdelay(2);      //duration of blink
        if((GPIOB_IDR & 1<<9) == 0)
            {break;}
        outputvalueA = 0;
        outputvalueB = 0;
        TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
superlongdelay(2);       //duration of blink
if((GPIOB_IDR & 1<<9) == 0)
    {break;}
}

for(i=0;i<10;i++)
{
    superlongdelay(1);
    if((GPIOB_IDR & 1<<9) == 0)
        {break;}
}

for(i=0;i<3;i++)
{
    if((GPIOB_IDR & 1<<9) == 0)
        {break;}

    outputvalueA = 4096;
    outputvalueB = 4096;

    TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood

    SOSStayOn = 1;  //keep SOS on after mode button released
    superlongdelay(2);       //duration of blink
    if((GPIOB_IDR & 1<<9) == 0)
        {break;}

    superlongdelay(2);       //duration of blink
    if((GPIOB_IDR & 1<<9) == 0)
        {break;}

    outputvalueA = 0;
    outputvalueB = 0;

    TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
    TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
    superlongdelay(3);       //duration of blink
if((GPIOB_IDR & 1<<9) == 0)
{
  break;
}

for(i=0;i<10;i++)
{
  superlongdelay(1);
  if((GPIOB_IDR & 1<<9) == 0)
  {
    break;
  }
}

for(i=0;i<3;i++)
{
  if((GPIOB_IDR & 1<<9) == 0)
  {
    break;
  }

  outputvalueA = 4096;
  outputvalueB = 4096;
  TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
  TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
  superlongdelay(2); //duration of blink
  if((GPIOB_IDR & 1<<9) == 0)
  {
    break;
  }

  outputvalueA = 0;
  outputvalueB = 0;
  TIM3_CCR1 = outputvalueA; //4096=full brightness  PC6 Throw
  TIM3_CCR2 = outputvalueB; //4096=full brightness  PC7 Flood
  superlongdelay(2); //duration of blink
  if((GPIOB_IDR & 1<<9) == 0)
  {
    break;
  }
}

for(i=0;i<15;i++)
{
  superlongdelay(1);
  if((GPIOB_IDR & 1<<9) == 0)
  {
    break;
  }
}
int main()
{

    //ADC setup
    RCC_AHB1ENR |= 1;           //Bit 0 is GPIOA clock enable bit
    RCC_APB2ENR |= 0x100;       //Bit 8 is ADC 1 clock enable bit
    GPIOA_MODER |= 0xF00;       //PA4-PA5 are analog
    GPIOA_PUPDR &= 0xFFFFF0FF;  //Pins PA4 PA5 are no pull up and no pull down
    ADC1_CR2 |= 1;              //Bit 0 turn ADC on
    ADC1_CR2 |= 0x400;          //Bit 10 allows EOC to be set after conversion
    ADC1_SQR3 |= 0x30000;       //Bits 16 and 17 = 11 so clock divided by 8

    //PWM setup
    //Clock bits
    RCC_AHB1ENR |= 4;           //Bit 2 is GPIOC clock enable bit
    RCC_APB1ENR |= 2;           //Enable peripheral timer for timer 3 (bit 1)

    //I/O bits
    GPIOC_MODER |= 0x2A800;     //Bits 17 thru 10 = 10101010 for Alt Funct Mode on PC8 PC7 PC6 and PC5
    GPIOC_OSPEEDER |= 0x3FC00;  //Bits 17 thru 10 = 11111111 for high speed on PC8 PC7 PC6 and PC5
    GPIOC_AFRL = 0x22200000;   //Sets PC7 PC6 and PC5 to AF2 (TIM3...5)
    GPIOC_AFRH = 0x2;          //Sets PC8 to AF2 (TIM3...5)
    TIM3_CCMR1 |= 0x6060;       //Timer 3 ch1 and ch2 in PWM1 mode
    TIM3_CCMR1 |= 0x0C0C;       //Timer ch1 and ch2 Preload and fast enable
    TIM3_CR1 |= (1 << 7);       //Auto reload is buffered
    TIM3_PSC = 100;             //Don’t use prescaling 100
    TIM3_ARR = 500;             //16MHz/100)/3200 = 50Hz = 20ms 3200
    TIM3_CCR1 = 0;              //Duty cycle for ch1 starts at 0
    TIM3_CCR2 = 0;              //Duty cycle for ch2 starts at 0
    TIM3_CCER |= 1;             //Compare and capture output 1 enable (CC1E)
TIM3_CCER |= 0x10; //Compare and capture output 2 enable (bit 4 = 1) (CC2E)

TIM3_CCER |= 0x20; //reverse polarity of TIM3 Ch2
TIM3_EGR |= 1; //Enable event
TIM3_CR1 |= 1; //Enable Timer 3

//Pushbutton and LVP input pin setup
RCC_AHB1ENR |= 1<<1; //Bit 1 is GPIOB clock enable bit
GPIOB_MODER &= 0xFFF0CFFF; //set pin B6, B8 and B9 to input mode

//Output pins setup
GPIOC_MODER |= 0x55; //pins PC0, PC1, PC2, PC3 are output

int backlightwait = 0; //this variable is used to keep the backlight LEDs on for a time after the light has been switched off

while(1)
{
    if(GPIOB_IDR & 1<<8) //if latching on/off button is on (pin B8)
    {
        GPIOC_ODR |= 1<<1; //turn on sensor
    }

    if(LVPactive == 0) //if LVP is not active
    {
        backlightwait = 0; //reset backlight wait
        GPIOC_ODR |= 1<<0; //turn on backlight LEDs

        if(HoldDistance != 1) //if hold distance is not set, read the sensor
        {
            readADC();
        }

        if(brightness >= 75) //if brightness is at or above 75%, start counting towards stepdown
        {
            if(StepdownCounter < 60000)
            {
                StepdownCounter++;
            }
            delay(1);
        }
    }
}
if(brightness < 75)  //if brightness is below 75%, reset the counter for stepdown
    StepdownCounter = 0;

if(StepdownCounter > 38961) // 38961 after 5 min. THIS NUMBER IS FOR 5 MIN. IT WILL NEED TO BE
    // RECALIBRATED ONCE THE REST OF THE CODE HAS BEEN FINISHED! 7792 ~ 1 min
    brightness = 25;          //step the brightness down to 25% >>CHANGE THIS BACK TO 50
    updown = 1;               //this sets the mode button to ramp up whenever the user next
    //press-and-holds it
    output();
    CheckForButtonPress();

if(((GPIOB_IDR & 1<<6) == 0)&&(LVPactive != 1))   //if PB6 is pulled low by voltage regulator and LVP
    //isnt already active
    brightness = brightness-5;  
    output();
    longdelay(250);

    if(brightness < 5)                             //if the brightness has to be below 5% to
        //keep battery voltage above 3.2v
        LVPactive = 1;                                //LVP is active
    
}

if(LVPactive == 1)           //if LVP is active
{
    GPIOC_ODR &= ~(1<<1);      //turn off sensor
    brightness = 5;            //brightness = 5% (this is to keep the user from being left in the
dark if the batteries die)
    PulseWidthAverage = 0;     //flood LED on only
    output();
if(LVPHasFlashed == 0) //if the main and backlight LEDs have not already flashed since the light has been switched on
{
    int q;
    for(q=0;q<4;q++)        //flash main LEDs 3 times
    {
        brightness = 15;       //brightness of blink
        GPIOC_ODR |= 1<<0;     //turn on backlight LEDs
        output();
        longdelay(150);        //duration of high blink
        brightness = 0;        //dimness of blink
        GPIOC_ODR &= ~(1<<0);  //turn off backlight LEDs
        output();
        longdelay(150);        //duration of low blink
    }
    //GPIOC_ODR &= ~(1<<0);   //turn off backlight LEDs
    LVPHasFlashed = 1;      //don't flash the LEDs like this again until the user switches the light off and back on again
}

if(GPIOB_IDR & 1<<9)      //flash the backlight LEDs 5 times if LVP is active and the user presses the mode button while the light is on
{
    int q;
    for(q=0;q<5;q++)
    {
        GPIOC_ODR |= 1<<0;     //turn on backlight LEDs
        longdelay(150);        //duration of blink
        GPIOC_ODR &= ~(1<<0);  //turn off backlight LEDs
        longdelay(150);        //duration of blink
    }
}

else
{
    while((GPIOB_IDR & 1<<8) == 0) //while latching on/off button is off (pin B8)
    {

GPIOC_ODR &= ~(1<<1);     //turn off sensor
SOSStayOn = 0;            //turn off SOS
StrobeStayOn = 0;         //turn off Strobe

TIM3_CCR1 = 0;            //turn main LEDs off
TIM3_CCR2 = 0;            //turn main LEDs off

StepdownCounter = 0;      //reset the stepdown counter every time the light is switched off
LVPHasFlashed = 0;        //reset LVPHasFlashed every time the light is switched off

//LVPHasFlashedTheFirstTime = 0;
LVPactive = 0;            //reset LVP status to off

superlongdelay(1);       //this delay is for keeping the backlight LEDs on for set amount of
                         //time after the light has been switched off
                         //LVPHasFlashedTheFirstTime = 0;
                         //LVPactive = 0;
                         //this delay is for keeping the backlight LEDs on for set amount of
time after the light has been switched off
if(backlightwait < 500)   //this IF statement keeps backlightwait variable from looping back
around to 0
{
    backlightwait++;    
}

if((backlightwait == 30) && (GPIOC_ODR & 1<<0))   //keep the backlight LEDs on for 3 seconds
after the light has been switched off  300 = 30 sec
{ 
    int q;
    int t;

    for(q=0;q<100;q++)      //this fades out the backlight LED’s (psuedo PWM)
    {
        for(t=0;t<15;t++)
        {
            GPIOC_ODR &= ~(1<<0);//turn off backlight LEDs
            delay(q);
            GPIOC_ODR |= (1<<0); //turn on backlight LEDs
            delay(100-q);
            break;              //break out of this for loop
        
        }

    }

}

if(GPIOB_IDR & 1<<9) //if the user presses the mode button while the fade
is happening
{
    GPIOC_ODR |= (1<<0);//turn on backlight LEDs
    backlightwait = 0;  //reset backlightwait
    break;              //break out of this for loop
}

if(GPIOB_IDR & 1<<8) //if the user presses the on/off button during the fade


User interface:

On/Off

To switch the light on or off, the user simply presses the ON/OFF button.

Changing brightness level:

To change brightness level, the user must press and hold the MODE button. Brightness level decreases until user releases button, or until lowest brightness levels is reached. After releasing, another press-and-hold of the MODE button will cause brightness to increase. If lowest/highest brightness level is reached, main LEDs will blink once, indicating lowest/highest brightness level has been reached. Backlight LEDs will flash continuously until user releases MODE button, or until strobe/SOS mode is activated.

Accessing strobe and SOS modes:

To activate strobe, the user simply ramps brightness up to max, then continues to hold the MODE button for ~2 seconds. After 2 seconds, strobe mode is activated. If the user releases the MODE button within ~1.5 seconds of strobe mode being activated, strobe mode will turn off after the button is released. If the user holds the button for longer than ~1.5 seconds after strobe mode has been activated, strobe mode will stay active after the MODE button is released. To turn off strobe mode, the user can either do a single press of the MODE button, or switch the light off and back on again. The procedure for accessing SOS mode is the same.
as accessing strobe mode, except the user must instead ramp the light down to the minimum brightness and continue holding the MODE button. For the light to stay in SOS mode after the MODE button has been released, the user must wait until the 4th SOS blink before releasing the MODE button.

**Turning on/off “Distance Hold” mode:**

To turn on/off distance hold mode, the user must do a quick single press of the MODE button.

**Low voltage protection mode:**

Whenever the battery voltage drops below 3.2 v, brightness is decreased by 5% at a time until the battery voltage rises back above 3.2 v. Once brightness is forced below 5%, LVP mode is activated. When LVP mode is activated, the spotlight LED is turned off, and the flood LED is turned on at 5% brightness. No adjustments can be made using the MODE button while LVP mode is active. LVP mode is deactivated every time the light is switched off.

---

**References**


