Embedded Final Project: Smart Streetlight Ver 2.0

Dylan Dancel, David Russo
Rowan University

December 23, 2018

Contact Information

Dylan Dancel: danceld9@students.rowan.edu
David Russo: russod1@students.rowan.edu

1 Design Overview

Smart Streetlight is a program automated by the MSP430F5529 development board that functions as a light controlling system. The design senses road vibrations and calculates an incoming car's speed based on the vibration information. In addition to the monitoring of vehicle speed, Smart Streetlight also measures the light intensity to determine if it is daytime or nighttime. The tuning for these light levels can be calibrated depending on the desired threshold for which the light considers night.

1.1 Design Features

- Power Efficient
- Ability to calibrate light intensity sensing

1.2 Featured Applications

- Power-efficient Automated Street Lights
- Automated Hall-Lights

1.3 Design Resources

- GitHub
- MSP430 Family User Guide
- MSP430F5529 Datasheet
- ADXL250 Datasheet
- OPA234 Datasheet
- OrCad Capture
- LM660CN Datasheet

1.4 Block Diagram

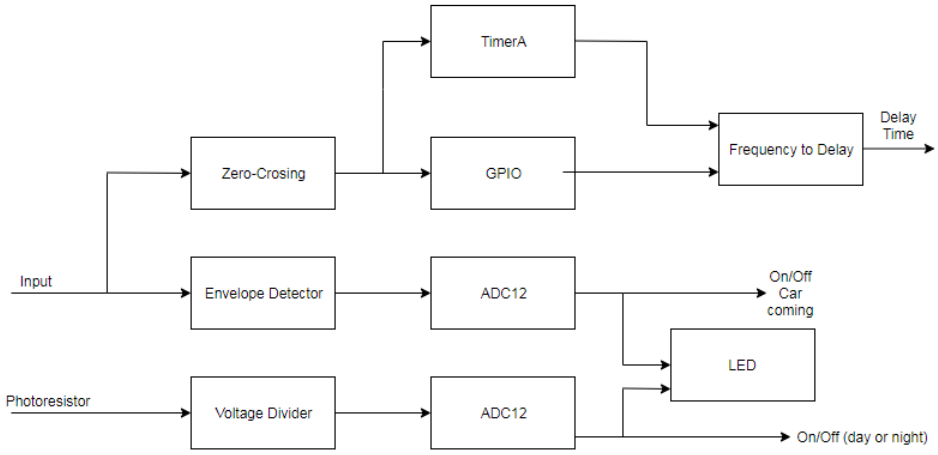

Figure 1: Block Diagram of Smart Streetlight

1.5 Board Image

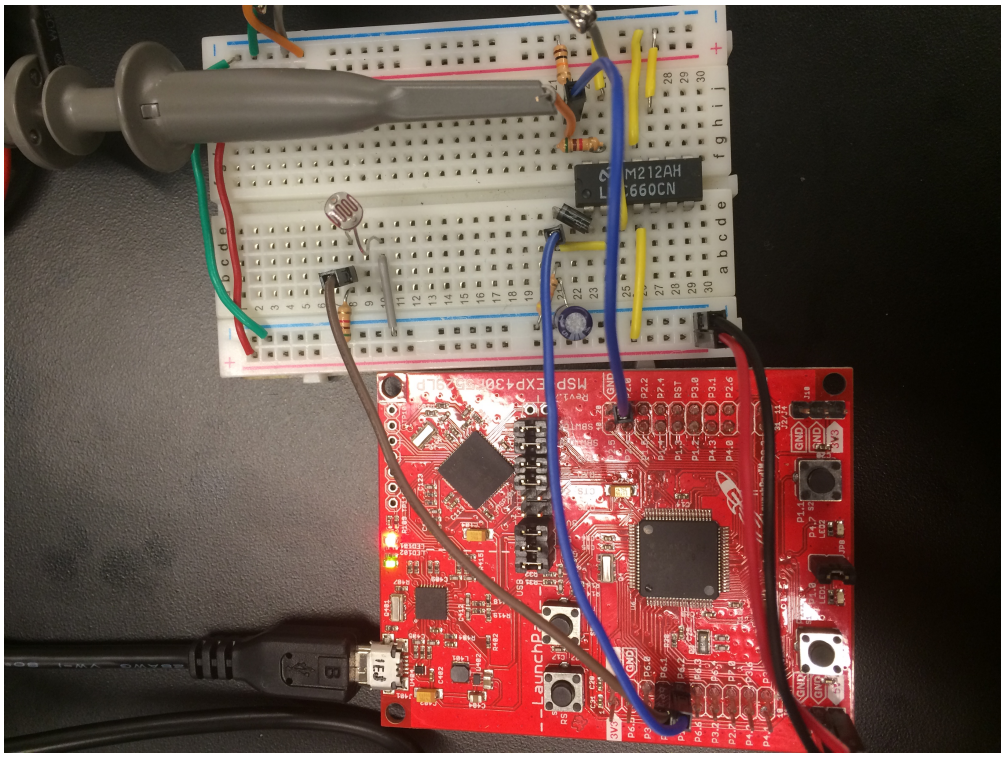

Figure 2: Circuitry for envelope detecting, zero-crossing, and light sensing

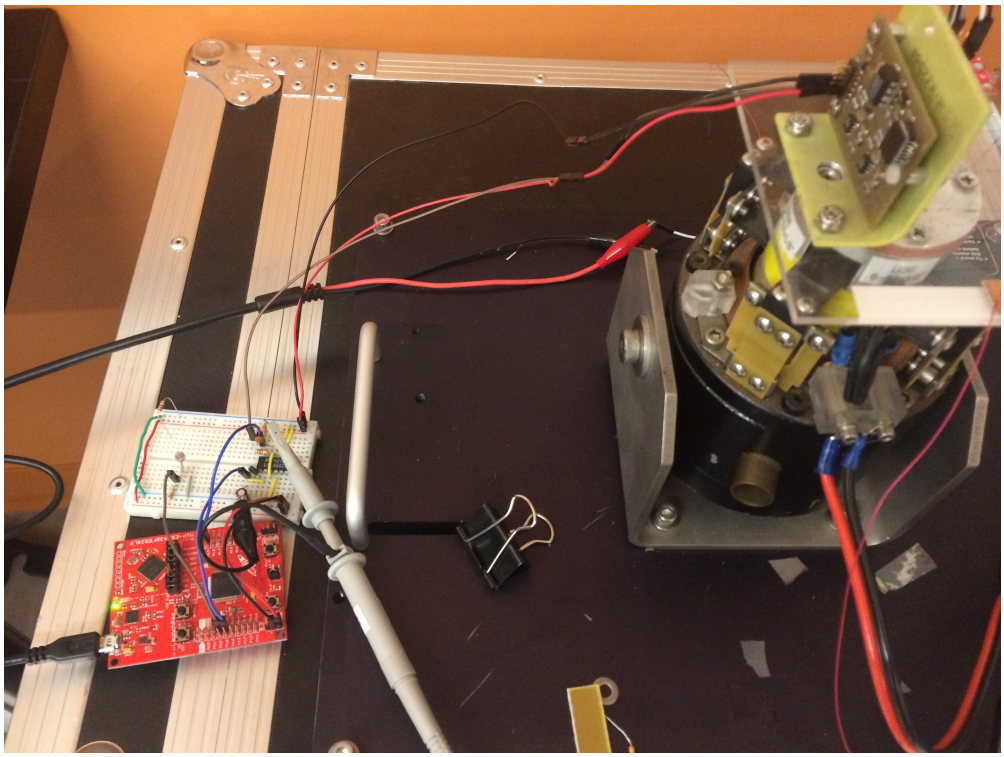

Figure 3: Circuitry for envelope detecting, zero-crossing, and light sensing connected to the accelerometer and vibration table

2 Key System Specifications

Parameter	Specifications	Details
Baud Rate	9600	Baud Rate to communicate with the MSP430F5529
Timer A0	16-bit	16-bit timer that has 5 capture compare registers
SMCLK	1 MHz	1MHz clock that is active in low power mode 0 and 1
ADC12	12-bit	Quantizes analog signal to binary values
Photoresistor	15kΩ	Resistance changes as the intensity of light changes

Table 1: Specifications

3 System Description

The desired outcome is to be able to analyze a frequency to determine a car's speed. Depending on the car's speed, it will determine a calculated delay time before a street-

light will turn on. This streetlight will not turn on if the light level of the environment is too bright and a light sensor will be used to accomplish this design choice.

3.1 Highlighted Devices

- MSP430F5529 Microprocessor
- Labworks ET-126 Vibration Table
- ADXL250AQC Accelerometer
- 2 MΩ Photoresistor

3.2 MSP430F5529

The MSP430F5529 is a Texas Instrument microprocessor with a Timer A0 that contains five capture compare registers, six GPIO configurable 8-bit ports, and a 12-bit analog-to-digital converter (ADC12). Bit 0 of Port 2 (P2.0) and Timer A0 handled all information needed to determine the car's speed, which is used in time delay calculations. ADC12 sampled all the necessary information for determining the light intensity of the environment and if a car is present, which is used to determine whether or not to turn on the street light.

4 SYSTEM DESIGN THEORY

4.1 Design Requirement 1

The first design requirement of Smart Streetlight is to allow the MSP430F5529 to take in readings through a zero-crossing circuit. This zero-crossing circuit detects how fast the car is physically moving and a timer is set to capture the amount of time between information. The conversion for the time delay is handled in the software. The setup of the zero-crossing circuit uses an operation amplifier (op-amp) as a comparator. As shown in Figure 8, the op-amp circuit outputs 3.3 V if the sinusoidal input is positive, and 0 V if negative. GPIO pin P2.0 receives the uni-polar square-wave output of the op-amp circuit. Values read from this pin are used to configure a Timer A0 module which will count the time between positive edges, and then count how long to wait before turning on the light. A time delay is made by setting the the timer A0 capture compare register 0 (TA0CCR0) to the calculated delay time based on the speed of the vehicle. Once the value in the Timer A0 register equals the value of TA0CCR0, the street light illuminates and the timer is reset to stop mode to conserve energy. To simplify the early models of Smart Streetlight, it is assumed that the sensing module is 200 meters away from the streetlight.

4.2 Design Requirement 2

The second design requirement is for Smart Streetlight to sense whether it is daytime or nighttime. If the system senses enough light to be considered daytime, the whole control system will shut off. This is done through a photo resistor in a voltage divider circuit, as shown in 9. As the light increases, the resistance of the photoresistor decreases. This phenomenon is leveraged in our design by connecting light intensity to voltage output. Output voltage measured by the ADC is inversely proportional to the resistance. The voltage across the photo resistor is then fed into the ADC12 input channel separate from the input channel used in the zero crossing circuit. This input is compared to turn-on and turn-off thresholds. The system turn-on and turn-off thresholds are different values so the system does not oscillate between on and off if it is dusk or dawn.

4.3 Design Requirement 3

The third design requirement is for Smart Streetlight to be sensitive to the envelope of the vibration waves. Since there is expected to be ongoing ambient vibrations, the system must distinguish between ambient vibrations and vehicle-induced road vibrations. The difference will be their amplitudes - bigger amplitude signals are considered vehicles, while smaller amplitudes would be ambience. Like with the photoresistor circuit, turn-on and turn-off voltages are set so the system will only operate the amplitude rises above the turn-on threshold. To obtain this result, the sinusoid is rectified with a super-diode op-amp configuration to make the signal positive, and smoothed with a capacitor to ground in parallel with the output. The placement of the capacitor makes the signal look like 6. This way, while a car is still present, the voltage will stay above the turn-off threshold.

5 Getting Started Software/Firmware/How to use the device

To power the device, a USB-to-micro-USB must connect the MSP430F5529 loaded with the code to a USB-A type a power supply. All other connections use MSP430F5529 pins.

5.1 Device Specific Information

The positive rail on the breadboard must be connected to the 5 V pin on the MSP430F5529 and the negative rail to ground on the MSP. P6.1 must be connected to the positive leg of the photoresistor as it connects to input channel 1 of ADC12. P6.2 must be connected to the cathode of the rectifier diode as it connects to input channel 2 of ADC12. P2.0 must be connected to positive end of the 10kΩ resistor in zero-crossing circuit. This 3.3 V signal will input through GPIO. Lastly, P2.6 must be connected to the anode of the off-board LED as it represents the ultimate goal of this project: strategically turning on and off the street light.

6 Test Setup

The program requires an extensive laboratory setup. Table 2 shows all the equipment needed for the setup. Briefly, The function generator outputs a signal to the power amplifier, which drives the vibration table to vibrate at a specific frequency and amplitude. The accelerometer sits on the vibration table and outputs a voltage proportional to the acceleration it experiences. This voltage is the source for the zero-crossing and envelope detection circuitry.

Manufacturer	Model	Description	Notes
Furman	PL-8	Power Conditioner	Serves as a power strip
QSC	RMX1450	Power Amplifier	drives vibe table
Agilent	33210A	Arb. waveform Generator	Sends signal into amp
Labworks Inc.	ET-126	Vibration Table	Simulates road vibrations

Table 2: Instrument Table

6.1 Test Data

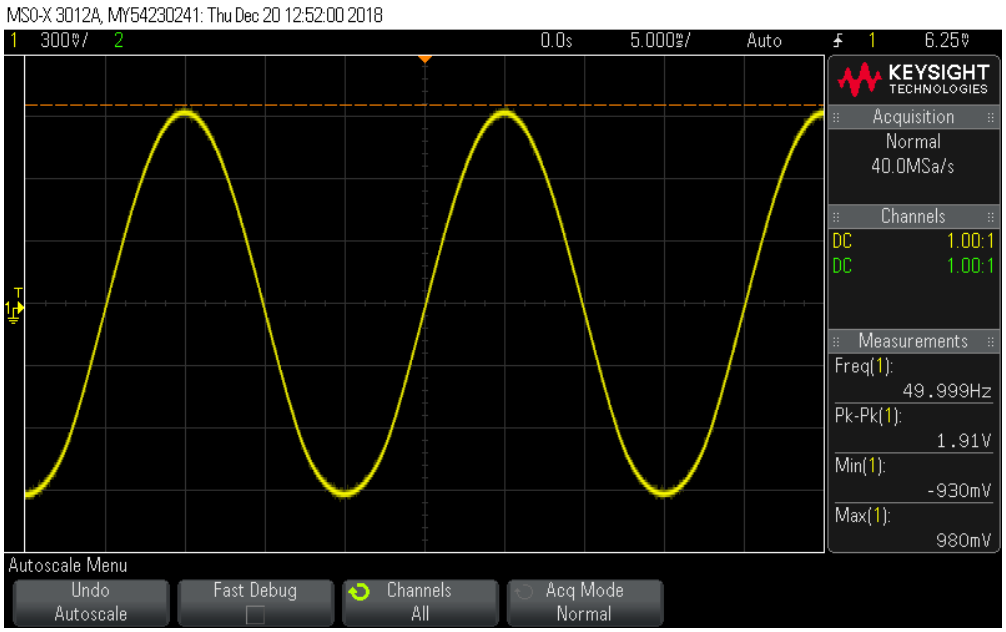

Figure 4: Input Signal from vibration table

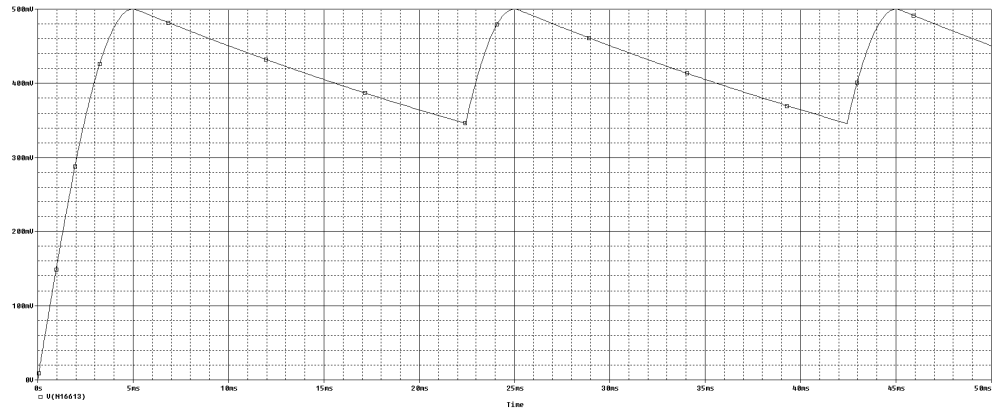

Figure 5: Simulation of envelope detector circuit.

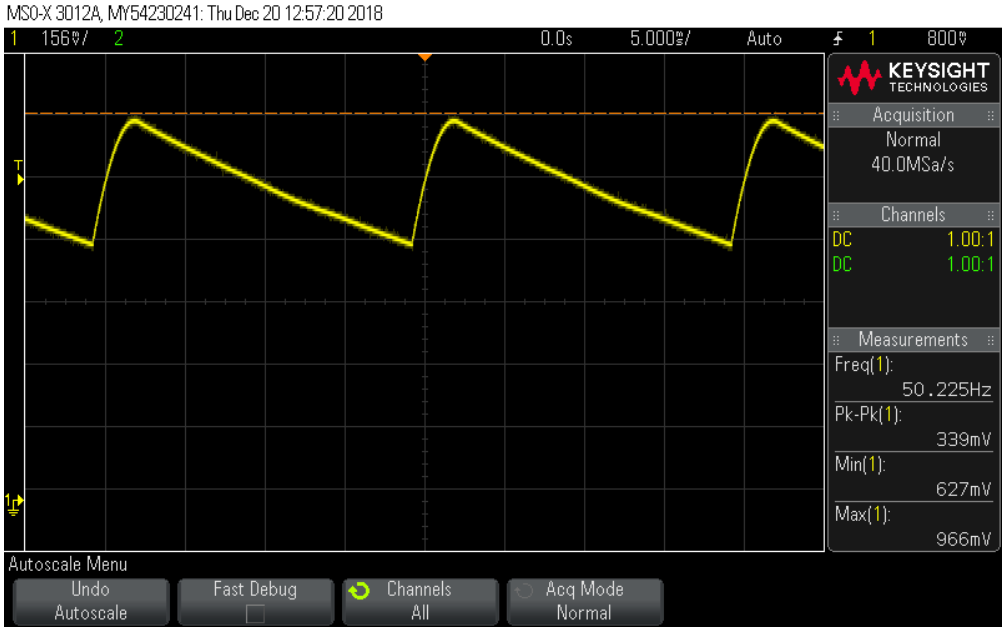

Figure 6: Oscilloscope reading envelope detector

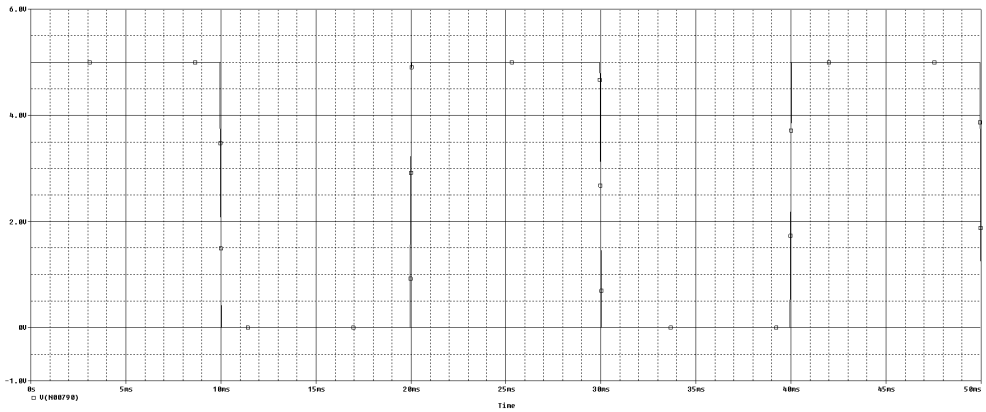

Figure 7: Zero Crossing Simulation

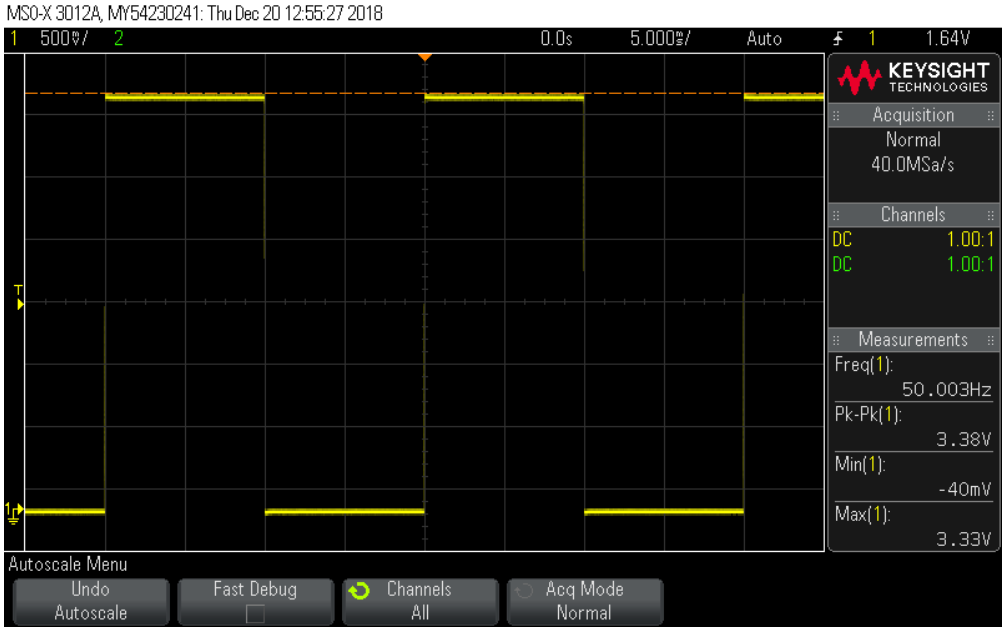

Figure 8: Oscilloscope Readings Zero Crossing

7 Conclusion and Future Works

The simple smart streetlight overall is a working project. It was able to receive data using the zero-crossing and envelope detector successfully to control an LED based on a given input signal. Future work's include the implementation of Wi-Fi modules instead of having a single MSP430 processor at each streetlight. A singular MSP430

would sense data from a fixed point away from the streetlights with it communicating over Wi-Fi to these wireless modules that are now at the streetlights. This would be limited to the amount of input channels to the MSP430 that is used but will prove to be more economical. Additionally, to make the product more economical the devices would be also powered by the vibrations of by road vibrations generated by piezo-electrics. Overall, the design can be improved and prove to be successful in terms of energy efficiency.

8 Design Files

8.1 Schematics

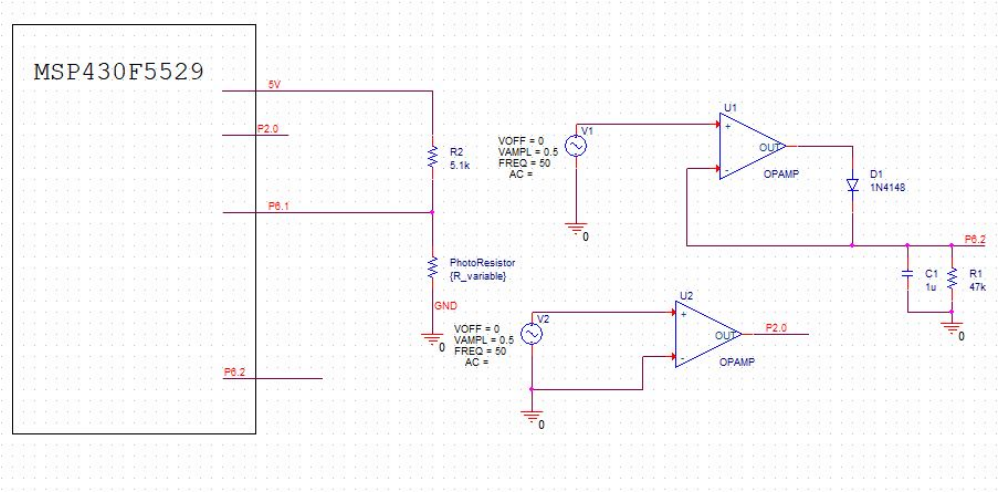

Figure 9: Voltage Divider circuit with photo resistor and envelope detector

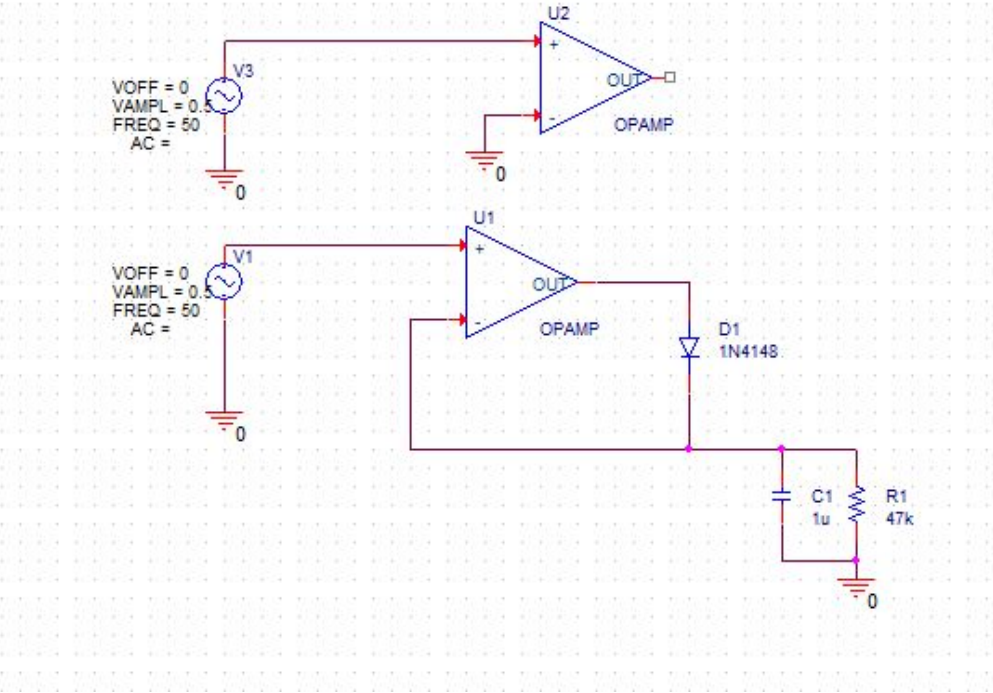

Figure 10: Envelope detector circuit

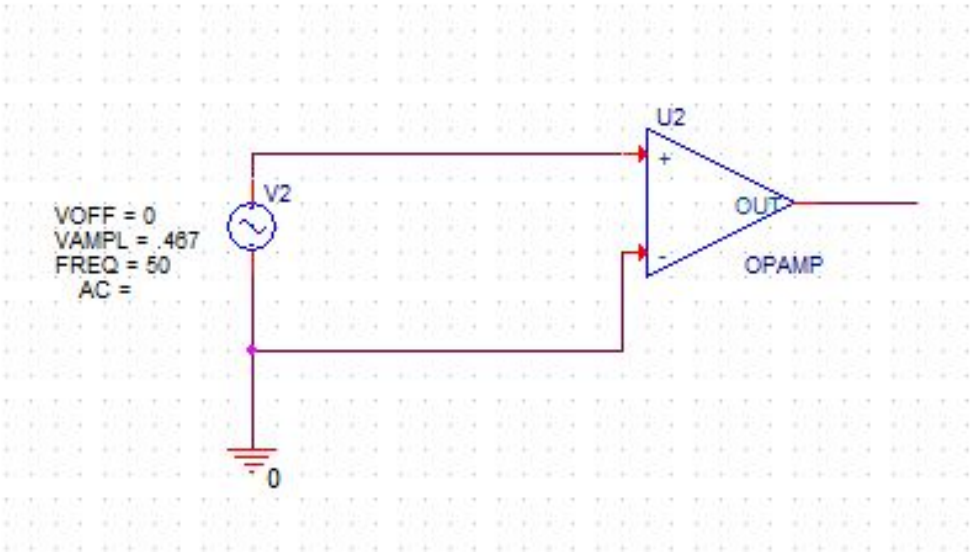

Figure 11: Zero Crossing Circuit

9 Bill of Materials

- MSP430F5529 x1
- LM660CN x1
- Photoresistor 15kΩ x1
- 5.1kΩ Resistor x1
- 47kΩ Resistor x1
- 1μF Electrolytic Capacitor
- Green LED x1
- Diode 1N4148 x1

10 Code

```c
#include <msp430.h>
#include <math.h>
/**
 * main.c
 *
 * Authors: Dylan Dancel, David Russo
 * Created: 12/12/2018
 * Last Edited: 12/20/2018
 *
 * Smart Street Light
 * ------------------
 *
 * This program uses the MSP430F5529 to simulate a highway street light that only turns on at night when cars are coming.
 * An accelerometer uses the road vibrations to sense sense when a car is coming and how fast it is going.
 * A photoresistor senses whether it is daytime or nighttime. During the day, the system will shut off.
 * The program is based on a model where the street light is 200 meters down the road from the sensor module.
 * The sensor module will delay turning on the street light so it turns on 5 seconds before the car reaches the light.
 *
 * Assumptions:
 * 1. The frequency of the road vibration is proportional to the speed of the vehicle: velocity = k * frequency
 * 2. Cars are moving at a constant velocity
 * 3. Expected frequency range on the road is 10-100 Hz
 *
 */

/*-------------------FUNCTIONS--------------------*/

void ADC12setup();                              // configures the ADC12 peripheral
void UARTsetup();                               // configures UART to interface with Realterm
void portSetup();                               // configures the ports to read the photoresistor
void zeroCrossingSetup();                       // configures Port and timer for zero-cross detection
```

```
void periodToDelayConversion(double x);             // converts period of the input signal to light turn-on delay
void initiateTimerDelay(short del);                 // converts the delay in seconds to CCR0 value

/*---------------GLOBAL VARIABLES---------------*/

unsigned int q;                                     // used in ADC12ISR to store the value of ADC12MEM0 (light info)
unsigned int q2;                                    // used in ADC12ISR to store the value of ADC12MEM1 (envelope info)
int timerState = 0;                                 // used in Port_2 interrupt to track the state when measuring frequency
double period;                                      // used in Port_2 interrupt to store the value of TA0R
const double k = 0.5;                               // constant relating car velocity to vibration frequency. We chose k = 0.5
short delay;                                        // used in periodToDelayConversion and represents delay time [s] to turn on light
_Bool isCarPresent = 0;                             // tracks when a car is present. 1 when car is present, and 0 otherwise
_Bool isDark;                                       // tracks when it is dark. 1 when it is dark, and 0 otherwise
const _Bool turnOnLight = 1;                        // Sends over UART to another board to dictate when to turn on and off Lights

/*-----------------MAIN FUNCTION-----------------*/

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                       // stop watchdog timer

    ADC12setup();                                   // configure ADC12 for light & vibration info
    UARTsetup();                                    // configure UART to communicate
    portSetup();                                    // configure ports to read photoresistor
    zeroCrossingSetup();                            // configures P2.0 to receive square signal, P2.6 to drive an LED, and TA0 to count
    __bis_SR_register(GIE);                         // if LPM0 used, ADC12_ISR will force exit
    while (1)                                       // infinite loop
    {
        ADC12CTL0 |= ADC12SC;                       // start conversion, enable conversion
    }
}

/*-----------------SETUP FUNCTIONS----------------*/

// ADC12 Setup
void ADC12setup()
{
    ADC12CTL0 = ADC12ON + ADC12SHT0_15 + ADC12MSC;  // ADC12 Control reg 0 on, sampling hold time 64 ADC12clk cycles
    ADC12CTL1 = ADC12SHP + ADC12CONSEQ_3;           // ADC12 Sampling and hold pulse set to sample input signal
                                                    // single-channel, single conversion
    ADC12MCTL0 = ADC12INCH_1;                       // ADC12 Mem0 control reg, input channel 0,
    ADC12MCTL1 = ADC12INCH_2 + ADC12EOS;            // ADC12 Mem1 control reg, input channel 1
    ADC12CTL0 |= ADC12ENC;                          // ADC12 enable conversion
    ADC12CTL0 |= ADC12SC;                           // Start sample and conversion
    ADC12IE = 0x02;                                 // interrupt enabled
}
```

```
// Port Setup to Read LDR
void portSetup()
{
    P6SEL = BIT1;                                   // Configures to input to ADC12
    P6DIR &= ~BIT1;                                 // Configures to input to ADC12

    // Green Test LED on board
    P4SEL &= ~BIT7;                                 // P4.7 set as GPIO
    P4DIR |= BIT7;                                  // P4.7 Set as an output
    P4OUT |= BIT7;                                  // P4.7 preset to ON
}

// UART Setup to interface with Realterm
void UARTsetup()
{
    P3SEL |= BIT3 + BIT4;                           // P3.3,4 = USCI_A0 TXD/RXD
    P4SEL |= BIT4 + BIT5;                           // P4.4,4.5 =USCI_A1 TXD/RXD

    // pin-to-pin UART setup
    UCA0CTL1 |= UCSWRST;                            // **Put state machine in reset**
    UCA0CTL1 |= UCSSEL_2;                           // SMCLK
    UCA0BR0 = 104;                                  // 1MHz, 9600 Baud Rate
    UCA0BR1 = 0;                                    // 1MHz, 9600 Baud Rate
    UCA0MCTL |= UCBRF_0 + UCBRS_1;                  // Modulation UCBRSx=1, UCBRFx=0
    UCA0CTL1 &= ~UCSWRST;                           // **Initialize USCI state machine**
    UCA0IE |= UCRXIE;                               // Enable USCI_A0 RX interrupt

    // USB-to-pin UART setup - we might need to use 5400 baud rate for ACLK (BR0 = 6, BR1 = 0)?
    UCA1CTL1 |= UCSWRST;                            // **Put state machine in reset**
    UCA1CTL1 |= UCSSEL_2;                           // SMCLK
    UCA1BR0 = 104;                                  // 1MHz, 9600 Baud Rate
    UCA1BR1 = 0;                                    // 1MHz 9600 Baud Rate
    UCA1MCTL |= UCBRF_0 + UCBRS_1;                  // Modulation UCBRSx=1, UCBRFx=0
    UCA1CTL1 &= ~UCSWRST;                           // **Initialize USCI state machine**
    UCA1IE |= UCRXIE;                               // Enable USCI_A1 RX interrupt
}

void zeroCrossingSetup()
{
    // P2.0 receives information from the zero crossing circuit
    P2SEL &= ~BIT0;                                 // P2.0 set as GPIO
    P2DIR &= ~BIT0;                                 // P2.0 set as an input
    P2IE |= BIT0;                                   // enables interrupt on P2.0
    P2IES |= BIT0;                                  // sets interrupt on positive edge
    P2IFG &= ~BIT0;                                 // clears the interrupt flag
```

```
    // Timer
    TA0CCTL0 = CCIE;                            // capture compare enabled
    TA0CTL = TASSEL_2 | MC_0 | ID_1;            // SMCLK, Continuous mode, Internal Divide by 2

    // Test Red LED to represent street light
    P1SEL &= ~BIT0;                             // P1.0 set to GPIO
    P1DIR |= BIT0;                              // P1.0 set to Output
    P1OUT &= ~BIT0;                             // P1.0 initialized to 0

    // off Board LED
    P2SEL &= ~BIT6;                             // P2.6 set to GPIO
    P2DIR |= BIT6;                              // P2.6 set to output
    P2OUT &= ~BIT6;                             // P2.6 initialized to 0
}

/*---------------INTERRUPT VECTORS--------------*/

// ADC12 Interrupt Service Routine
#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR(void)
{
    switch(__even_in_range(ADC12IV,34))
    {
        case  0: break;                         // Vector  0:  No interrupt
        case  2: break;                         // Vector  2:  ADC overflow
        case  4: break;                         // Vector  4:  ADC timing overflow
        case  6: break;                         // Vector  6:  ADC12IFG0
        case  8:
            q = ADC12MEM0;                      // write value in MEM0 to unsigned int q
            q2 = ADC12MEM1;                     // write value in MEM1 to unsigned int q2
                if (q >= 4095)                  // if ADC12MEM0 >= certain value, its night time
                {
                    P4OUT |= BIT7;              // Turn on P4.7 (green on-board LED)
                    P3OUT |= BIT1;              // Turn on P3.1 (GPIO)
                    isDark = 1;                 // Set flag to signal it is dark out
                }
                else if (q <= 3334)             // else if ADC12MEM0 <= certain value its daytime
                {
                    P4OUT &= ~BIT7;             // Turn off P4.7;
                    P3OUT &= ~BIT1;             // Turn off P3.1;
                    isDark = 0;                 // signify it is light out
                }
                if (q2 > 573)                   // turn on threshold: car is present
                    isCarPresent = 1;           // Car is present
                else if (q2 < 163)              // turn off threshold: car is not present
                    isCarPresent = 0;           // Car is not present
            break;
        case 10: break;                         // Vector 10:  ADC12IFG2
```

```
        case 12: break;                         // Vector 12:  ADC12IFG3
        case 14: break;                         // Vector 14:  ADC12IFG4
        case 16: break;                         // Vector 16:  ADC12IFG5
        case 18: break;                         // Vector 18:  ADC12IFG6
        case 20: break;                         // Vector 20:  ADC12IFG7
        case 22: break;                         // Vector 22:  ADC12IFG8
        case 24: break;                         // Vector 24:  ADC12IFG9
        case 26: break;                         // Vector 26:  ADC12IFG10
        case 28: break;                         // Vector 28:  ADC12IFG11
        case 30: break;                         // Vector 30:  ADC12IFG12
        case 32: break;                         // Vector 32:  ADC12IFG13
        case 34: break;                         // Vector 34:  ADC12IFG14
        default: break;
    }
}

// USB-to-pin UART Interrupt Service Routine
#pragma vector=USCI_A1_VECTOR
__interrupt void USCI_A1_interrupt(void)
{
    switch (__even_in_range(UCA1IV, 4))
    {
        case 0: break;                          // Vector 0 - no interrupt
        case 2:                                 // Vector 2 - RXIFG
            while (!(UCA1IFG & UCTXIFG));       // USCI_A1 TX buffer ready?
                delay = UCA1RXBUF;              // assigns target to the value from Realterm
            break;
        case 4: break;                          // Vector 4 - TXIFG
        default: break;
    }
}

// Pin-to-pin UART Interrupt Service Routine
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_interrupt(void)
{
    switch (__even_in_range(UCA0IV, 4))
    {
        case 0: break;                          // Vector 0 - no interrupt
        case 2:                                 // Vector 2 - RXIFG
            while (!(UCA0IFG & UCTXIFG));       // USCI_A1 TX buffer ready?
                delay = UCA0RXBUF;              // assigns delay to the value from Realterm
            break;
        case 4: break;                          // Vector 4 - TXIFG
        default: break:
```

```
    }
}

// Zero-crossing Circuitry PORT Interrupt Service Routine
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
    // Store value of TA0R in a variable
    // reset timer
    // if voltage is above turn on threshold, do calculations to change TA0R into delay turn-on time

    switch (timerState)
    {
        case 0:                                 // no car sensed OR it is daytime
            P1OUT &= ~BIT0;                     // turn off red test LED
            P2OUT &= ~BIT6;                     // turn off OFF-board LED
            if (isCarPresent && isDark)         // if a car is present at night time, go to state 1
                timerState++;                   // go to state 1
            // if below threshold or it is light, stay in state 0
            break;
        case 1:                                 // timer has not started yet
            TA0R = 0;                           // reset TA0R
            TA0CTL = TASSEL_2 | MC_2 | ID_1;    // SMCLK, Continuous mode, Internal Divider = 2
            if (!isCarPresent || !isDark)       // if below above the "daylight" threshold or no car is present
                timerState = 0;                 // then go back to state 0
            else
                timerState++;                   // otherwise, go to state 2
            break;
        case 2:                                 // port interrupt has triggered a second time
            period = TA0R;                      // period equals the value in TA0R
            TA0R = 0;                           // reset TA0R
            if (!isCarPresent || !isDark)       // if above the "daylight" threshold or no car is present
                timerState = 0;                 // then go back to state 0
            else
                timerState++;                   // otherwise, go to state 3
            break;
        case 3:                                 // port interrupt has triggered a third time
            period += TA0R;                     // in the process of taking a 3 measurement average...
            TA0R = 0;                           // reset TA0R
            if (!isCarPresent || !isDark)       // if above the "daylight" threshold or no car is present
                timerState = 0;                 // then go back to state 0
            else
                timerState++;                   // otherwise, go to state 4
            break;
        case 4:                                 // port interrupt has triggered a fouth time
            period += TA0R;                     // in the process of taking a 3 measurement average...
            period /= 3;                        // period = 3 measurement average
            periodToDelayConversion(period);    // calculates the delay to turn on the light in seconds
```

```
            if (delay == 0)                         // if the calculated delay light turn-on time is 0
            {
                P1OUT |= BIT0;                      // turn on red on-board LED immediately (TEST)
                P2OUT |= BIT6;                      // turn on OFF-board LED immediately
                UCA0TXBUF = turnOnLight;            // send a signal that says turn on the light
            }
            else
                initiateTimerDelay(delay);          // convert from delay [s]
            if (!isCarPresent || !isDark)           // if above the "daylight" threshold or no car is present
                timerState = 0;                     // then go back to state 0
            else
                timerState++;                       // otherwise, go to state 5
            break;
        case 5:                                     // wait
            if (!isCarPresent || !isDark)           // if above the "daylight" threshold or no car is present
                timerState = 0;                     // then go back to state 0
            break;
        default:
            if (!isCarPresent || !isDark)           // if below threshold, go to state 0
                timerState = 0;                     // then go back to state 0
            break;
    }
    P2IFG &= ~BIT0;                                 // clear P2.0 interrupt flag
}

#pragma vector=TIMER0_A0_VECTOR                     // interrupt service routine for Timer 0 CCR1
__interrupt void TIMER0_A0(void)
{
    if (timerState == 5)                            // if in the "wait" phase
    {
        // Turn on the LED P1.0
        P1OUT |= BIT0;                              // turn on ON-Board LED (Test)
        P2OUT |= BIT6;                              // turn on OFF-Board LED
        UCA0TXBUF |= turnOnLight;                   // Transmit a 1 through UART pin-to-pin
        // wait (for now) 10 seconds
        // Turn off the LED
        TA0R = 0;                                   // reset TA0R
        TA0CCR0 = 40959;                            // equivalent to 10 seconds
        timerState++;                               // go to state 6
    }
    else if (timerState == 6)
    {
        P1OUT &= ~BIT0;                             // turn off ON-board LED after 10 seconds (test)
        P2OUT &= ~BIT6;                             // turn off OFF-board LED after 10 seconds
        UCA0TXBUF = !turnOnLight;                   // Transmit a 0 through UART pin-to-pin
        TA0CTL = TASSEL_2 | MC_0 | ID_1;            // change Timer A0 to stop mode (MC_0)
        timerState = 0;                             // go back to state 0: no car or daylight
    }
}
```

```
/*---------------INTERNAL FUNCTIONS---------------*/

void periodToDelayConversion(double x)
{
    double frequency, velocity = 0;
    double d;
    // inputs the average of the ADC voltage
    x = (2*period)/1000000;                         // converts from [us] to [s]
    frequency = 1/x;                                // converts from [s] to [Hz]
    velocity = k*frequency;                         // converts from [Hz] to [m/s]
    d = 200/velocity - 5;                           // converts from [m/s] to [s]
    if (d <= 0.0)                                   // if vehicle moves faster than 40 [m/s]
        delay = 0;                                  // lower boundary conditions [s]
    else if (d >= 15.0)                             // if vehicle moves slower than 10 [m/s]
        delay = 15;                                 // upper boundary condition [s]
    else                                            // otherwise, in the normal condition
        delay = round(d);                           // round delay into a 16-bit short
}

void initiateTimerDelay(short del)
{
    // Turn on LED P1.0 as the street light after Timer A0 Delay
    TA0CTL = TASSEL_1 | MC_1 | ID_3;                // ACLK, Up-mode, Internal divider = 8
    TA0CCR0 = (4096*delay)-1;                       // converts time delay [s] to CCR0 value
}
```