Formal Report - Pulse Monitor

Shiori Kuboki

37612124

Abstract

In this lab, a pulse monitor using infrared LED and a phototransistor was made. An amplifier and comparator were also added to transmit information to the MSP430. My original expectation was to plot the pulse and display the heart rate using Python, but decided to change this to use the 7-segment LED on the breadboard to display the heart rate instead, due to limitations from the USB3 port problem. As a result of this project, I was successfully able to build a working pulse monitor that display heart rate. In conclusion, I feel that this project was a success but also has much room for further improvement.

Introduction

The motivation for this project was to work on something that required a good understanding of both coding in C and building circuit components. My goal was to aim for a project that could enhance my knowledge in these two areas while making use of my previous knowledge in my two majors, physics and computer science. Looking at past projects, he pulse monitor caught my eye since medical physics and computer science in one of my interests. This also sounded like it would also fulfill my original purpose of being able to code and build, thus was chosen as my project.

Theory

Out of the many ways to measure pulse, the method I used was to measure using an infrared LED and a phototransistor. By using these, the pulse is measured by detecting the amount of infrared light that passes through your finger; if a 'pulse' of blood passes through your finger, it creates a brief moment of opacity, which is detected using the phototransistor.

Comparing the amount of infrared light your finger block and the amount of infrared light your pulse blocks is significantly different, and we see clearly that the latter is much smaller. Due to this, the measurements are very subtle and thus hard to analyze. In order to overcome this, we build an amplifier after the LED and phototransistor. The amplifier will amplify our measurements to make analysis easier. A high-pass filer was also added to filter out any unnecessary noise.

The next part of this project is the comparator. The MSP430 is not the best at detecting waveform voltages but is better at just reading 'high' or 'low' voltages. The comparator will aid in this process by changing our pulse waveform into a square wave based on whether the voltage is past or below a certain voltage.

Finally, the resulting square wave is transmitted to the MSP430 for further analysis such as calculating heart rate.

<u>Apparatus</u>

<u>Infrared LED & Phototransistor</u>

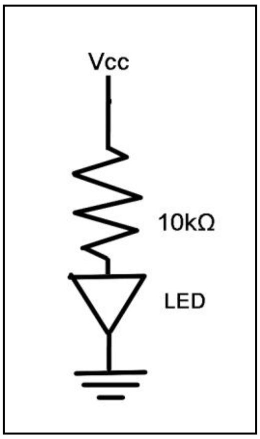

Diagram 1
Infrared LED
Circuit

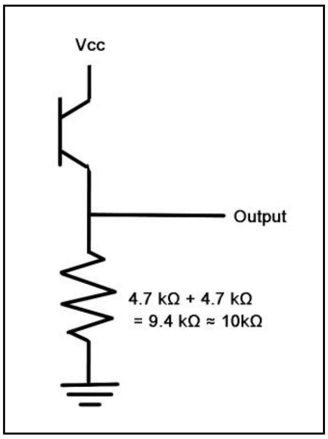

Diagram 2
Phototransistor
Circuit

First, I built the circuit shown in Diagram 1 for the infrared LED (HSDL4220) and the phototransistor circuit shown in Diagram 2 for the phototransistor (QSD124). The LED and the phototransistor are both powered by 5V from the board. A 10kΩ resistor was used in the LED circuit, setting the current in the circuit to be $(5-1.5)/(10x10^3)$, where 1.5V is the forward voltage for the LED. The 10kΩ resistor was chosen for the phototransistor because the resistor is acting as a method to flow current and higher resistance makes the phototransistor more sensitive to light, thus the output will have better signal. The implementation of the two circuit diagrams can be seen in Diagram 3 (next page). Because there were not available 10kΩ resistors, two 4.7kΩ resistors were connected in series to create a resistance of 9.4kΩ, which is approximately 10kΩ. The finger is to be placed in between the two heads of the infrared LED and the phototransistor.

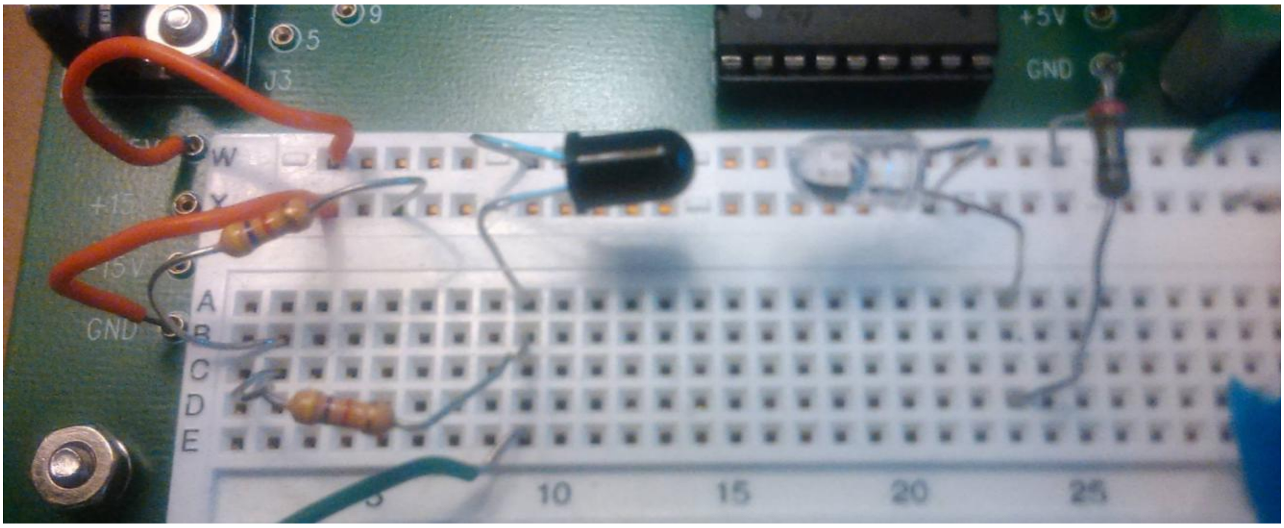

Diagram 3
Infrared LED & Phototransistor

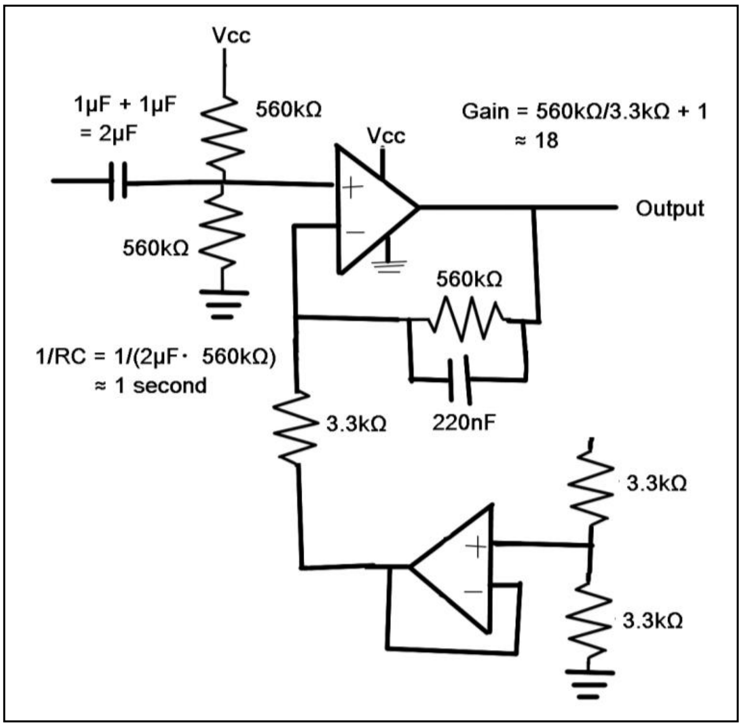

Diagram 4
Amplifier Circuit with Values

<u>Amplifier</u>

Next, an amplifier was required to amplify our measurements from the LED and the phototransistor. The circuit diagram for this can be seen in Diagram 4 and Diagram 5 (next page).

The amplifier was made using mainly the op-amp (LM124). As seen in Diagram 5, we first put a high-pass filter in order to filter through only long wavelengths such that we are able to get rid of noise.

The particular wavelength can be calculated from the RC constant as follows:

$RC = 2\mu F \cdot 560k\Omega$

$= 1.120$ seconds $\approx$ 1 second

Because there were no available $2\mu F$ capacitors, two $1\mu F$ capacitors were connected in parallel.

The next part of the circuit is a non-inverting amplifier made using op-amps.

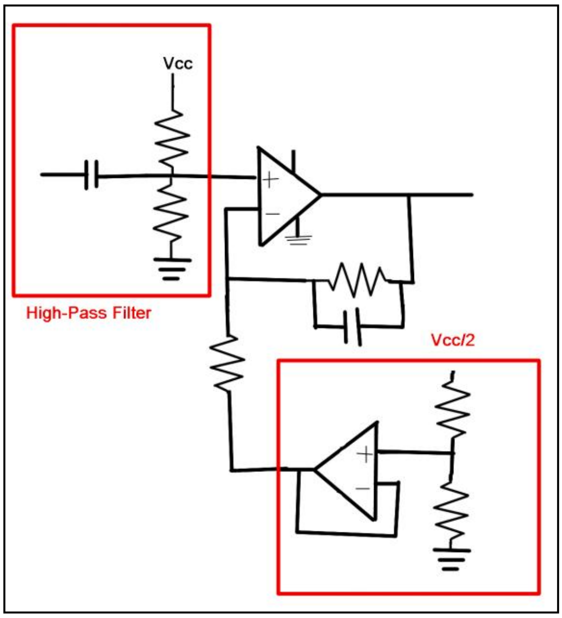

Diagram 5
Amplifier Circuit

The gain is calculated as follows:

$V_{out} = V_{in}(1 + R_f/R_{in})$

$= V_{in}(1 + 560k\Omega/3.3k\Omega)$

$\approx V_{in}*170$

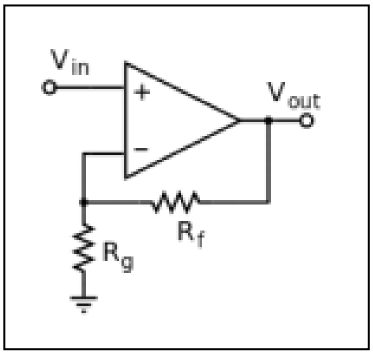

Diagram 6
Non-inverting Amplifier

Thus, the gain is approximately 170. A typical non-inverting amplifier is in the structure shown in Diagram 6. For my project, we need to create a virtual ground instead of ground (see Diagram 6) because I want to be able to use +5V voltage and ground to power the op-amp instead of ±15V. In order to do so, we create a $V_{cc}/2$ voltage source instead using another op-amp (see Diagram 5).

The actual implementation of the amplifier and high-pass filter is shown on

Diagram 7. Now, I have my measurements from the LED and phototransistor amplified by approximately 170.

The final component of the circuit is the inverting comparator. Because the MSP430 is better at just reading high/low values rather than changing voltages, a comparator was added to change the pulse waveform into a square wave. The comparator takes a voltage and compares the voltage on the waveform to that voltage to see if it is higher or lower; according to this, the square wave output is high or low. The circuit diagram of the comparator is shown in Diagram 8. As shown, a variable resistor is connected, which controls the voltage the waveform is compared to. The 560k Ω in the circuit sets the amount of hysteresis. The internal structure of the comparator (LM311) and the actual implementation of the circuit is shown in Diagram 9 and Diagram 10 (next page).

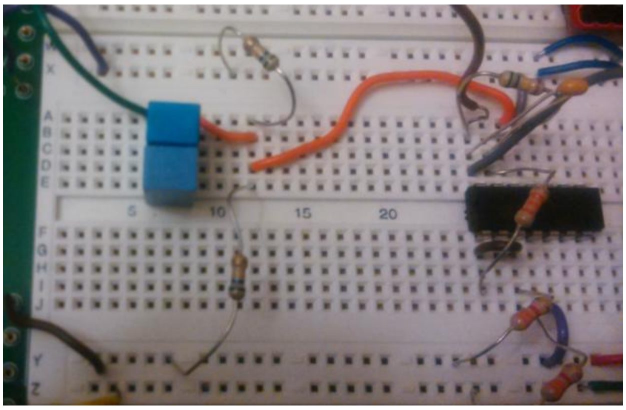

Diagram 7
High-pass Filter and Amplifier

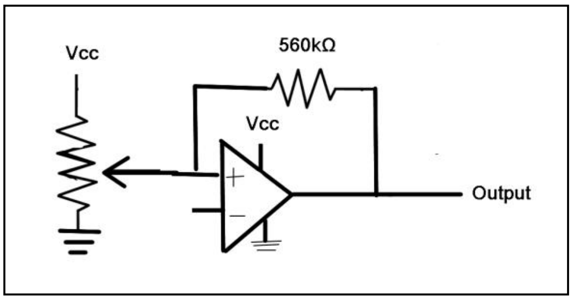

Diagram 8
Comparator Circuit

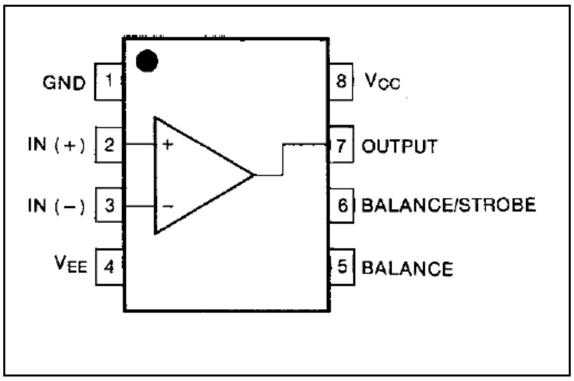

Diagram 9
Comparator

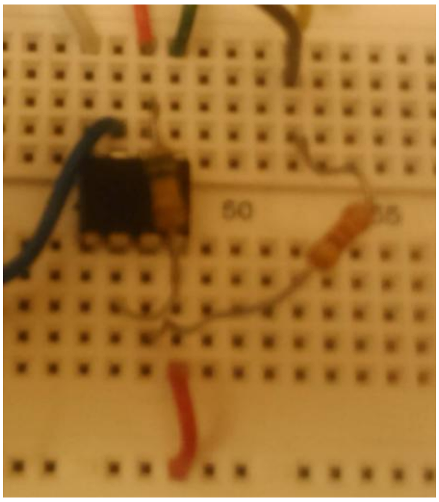

Diagram 10
Comparator

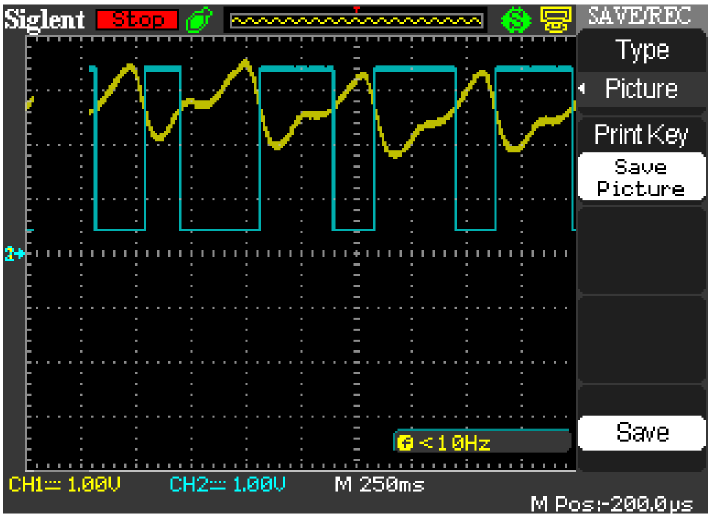

Diagram 11
Waveform after Comparator

The resulting waveform after the comparator is shown in Diagram 11. The yellow waveform is the pulse (after being amplified) and the blue square wave is the resulting waveform after the comparator. We can see that the pulse waveform has been translated into a square wave nicely.

Shown in Diagram 12 and Diagram 13 (next page) is the appearance of the entire circuit. Note that the value shown on the 7-segment display on Diagram 13 is not

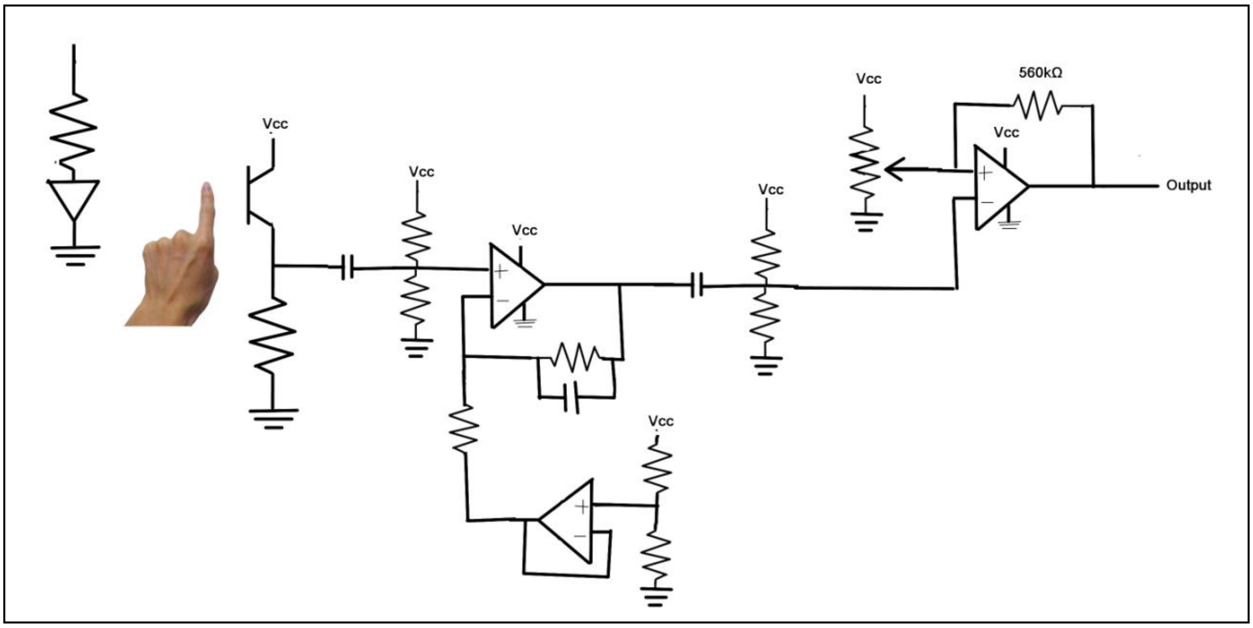

Diagram 12
Entire Circuit

accurate because the finger is not placed there. Also, another high-pass filter with the same RC constant as the one before was added between the amplifier and the comparator to filter out any new noise that may have been created after the amplifier. The general flow of the circuit is as follows:

1. The infrared LED goes through the finger.
2. The phototransistor catches the signal.
3. The high-pass filter filters out noise.
4. The amplifier amplifies the signal.
5. Another high-pass filter filters out noise.
6. The comparator changes the signal to a square wave.

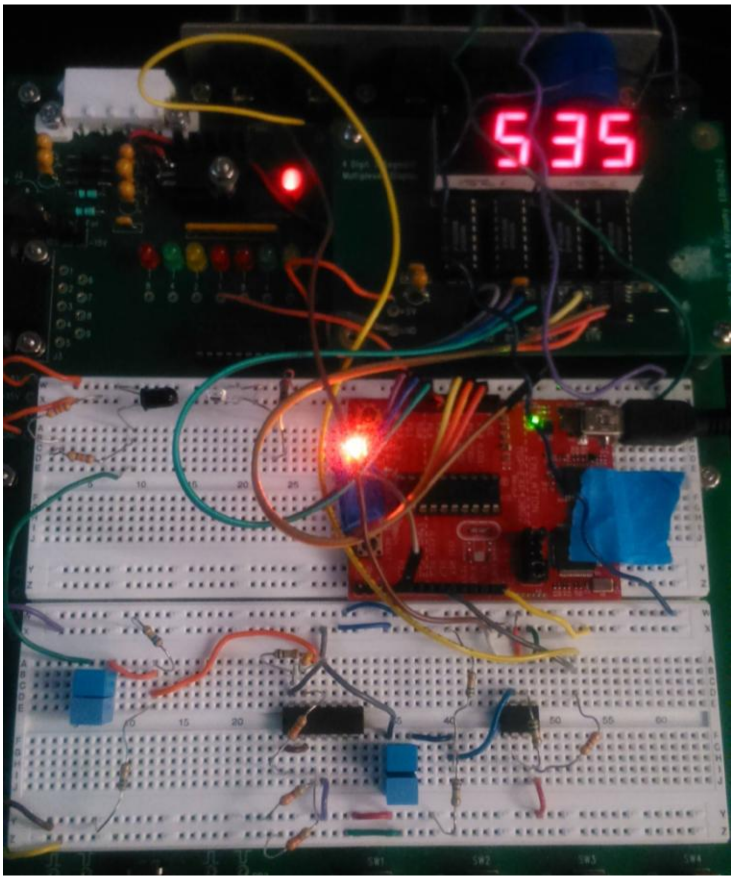

Diagram 13
Entire Circuit

MSP430

The final main component of this circuit is the MSP430 and its code made in Code Composer Studio in C. After the comparator generated a square wave output, this result was transmitted to the MSP430 from P1.7. Then, the input from P1.7 was analyzed to generate the pulse calculations. The flow chart on Diagram 14 shows the general flow of the main function in the code. In short, the code loops forever, executing calculations whenever the heart beats. The loop obtains the current value and calculates the frequency based on the period of the square wave, calculated from the current value and the previous value. In order to avoid overflow, long is used to initialize most of the parameters. In addition, an

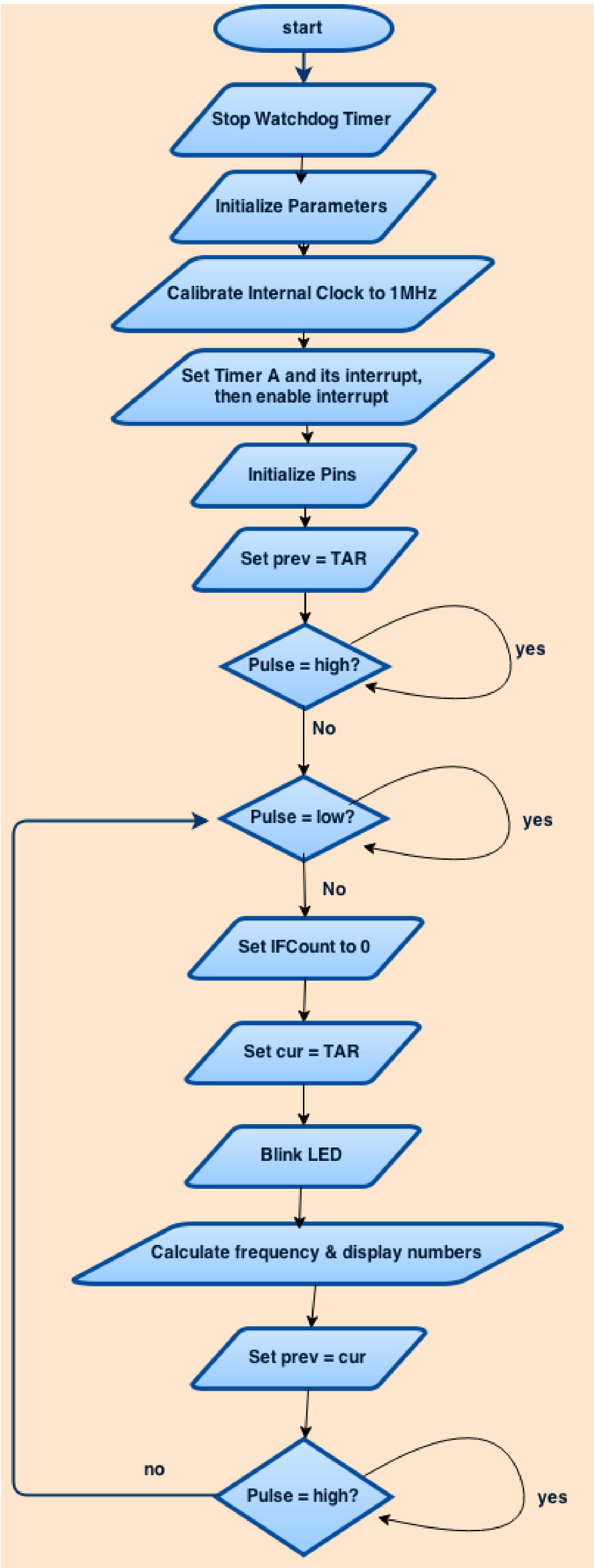

Diagram 14
Main Function Flow Chart

interrupt flag is set for timer A such that whenever there is an overflow, we count the number of times it overflowed, and add it to our current value. Also in the loop, I blink the LED whenever there is a pulse and I also update the 7-segment display to show the heart rate. The implementation of this main function, the helper function displayNumbers, and the interrupt handler can be seen in detail in the appendix later on.

Results

The result of this project is shown on Diagram 15, which shows the 7-segment LED displaying the correct heart rate and the external LED blinking for every hear beat when the finger is placed, as expected. The pulse shown on the oscilloscope is the same as shown on Diagram 11 (page 7). The results were not verified by an actual pulse calculator but was verified by comparing the heart rates to the periods shown on the oscilloscope and by directly comparing my own pulse to the blinking of the LED. However, the 7-segment LED only displays the heart rate every second time; every other time, the display showed the

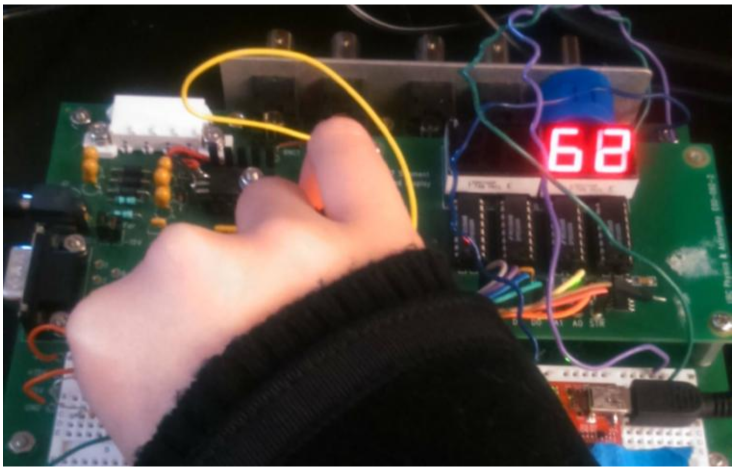

Diagram 15
Pulse Monitor

value 0. I believe that this is due to a bug in the code because of the way that prev and cur are calculated and stored; probably, prev and cur are not stored correctly such that they must 'reset' their values once (displaying 0) before a new calculation.

Discussion

As talked about in the results, the 7-segment LED did not work completely as I wanted it to, so this is definitely something that I want to work on a bit more. Furthermore, the heart rate was calculated this time basing on 1 period of the heart rate, but I would like to base it off the average of several periods to make the heart rate more accurate. Also, I think that it would've been really cool if I could've used a beeper to make it beep whenever my heart rate was too low (about to die), but this could also not be done because of the problem with calculating the heart rate. Another part I could improve on is the sensitivity of the LED and phototransistor; the project only gives a nice waveform if the LED does not pass through the nail of your finger, but it would be much more practical to be able to create something that could monitor the heart beat regardless. Plotting the results on Python would've also been nice, but because of the USB3 problem with my laptop, I concluded that it would be too difficult to be able to debug just on the lab computers, so I had to give that up.

The most difficult part about this project was probably the overflows; I could not figure out why my calculations were not outputting expected values since I did not except to have to count the overflows. This was solved by Dr. Michal's help. Furthermore, implementing the interrupt handler was also difficult because it was difficult to find that my code doing 'IFCount++' was not supposed to be placed inside my switch statements but outside of them but still keeping the switch statements (which seemed to do nothing after taking 'IFCount++' out) in order to clear the interrupt flag. Another difficulty was debugging on Code Composer Studio. For some reason, CCS does not let us place breakpoints at certain lines, which made stepping through the code difficult. In addition, the printf function was not supported because it is too big for CCS to manage, so I could not print the values to check what they were (although I found out later that I could hover over values to check them).

On the other hand, what I found very simple was to display numbers on the 7-segment LED. Not only was this done in a previous lab, but I also was familiar with it from my computer science courses, so this was very easy. Also, being able to look at waveforms and check voltages using the oscilloscope was very helpful when debugging.

Conclusions

In conclusion, I think that it is safe to say that in this experiment I was successfully able to make a working pulse monitor. Although it still has bugs (regarding the '0' for the 7-segment display) and many areas to improve on, the apparatus is able to show my pulse and the heart rate, which was my main purpose of the project.

References

"High-Performance T-1 3/4 (5mm) TS AlGaAs Infrared (875nm) Lamp." http://www.suzushoweb.com/pdf_file/1500000000015.pdf

"LM124N Datasheet(PDF) - Texas Instruments." ALLDATASHEET.COM, http://www.alldatasheet.com/datasheet-pdf/pdf/176978/TI/LM124N.html

"LM311 Single Comparator." https://www.fairchildsemi.com/datasheets/LM/LM311.pdf

"QSD122, QSD123, QSD124 Plastic Silicon Infrared Phototransistor." http://www.farnell.com/datasheets/1717172.pdf

Appendix

Main Function

```
#include <msp430.h>

/*
 * main.c
 */


volatile unsigned int IFCount = 0;       /* Counts interrupt for overflow */


void displayNumbers(unsigned int num, unsigned int digit);

 int main(void) {


    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer

    unsigned long cur = 0;              /* Current value */
    unsigned long prev = 0;             /* Previous value */
    unsigned long freq = 0;
    unsigned int ifreq;
    /* next three lines to use internal calibrated 1MHz clock: */
    BCSCTL1 = CALBC1_1MHZ;       // Set range
    DCOCTL = CALDCO_1MHZ;
    BCSCTL2 &= ~(DIVS_3);

    TACTL = TAIE + TASSEL_2 + MC_2;// set SMCLK as source, divide by 8, continuous mode

    __enable_interrupt();
    // *NOTE TO SELF*
    // OUT  -   P1.0=0x01 (LED), P1.5=0x20 (External LED), P1.6=0x40 (LED)
    //          P1.1=0x02 (STR), P1.2=0x04 (A0), P1.3=0x08 (A1)
    //          P2.0=0x01 (D0), P2.1=0x02 (D1), P2.2=0x04 (D2), P2.3=0x08 (D3)
    // IN   -   P1.7=0x80 (output)
    P1DIR = 0x7F;                // Set output direction except for P1.7
    P1REN = 0x80;                // Enable resistor on P1.7
    P1OUT |= 0x80;               // Set direction as pullup for P1.7
    P1OUT |= 0x41;               // Turn on 2 LEDs
    P1OUT &= ~0x20;              // Turn external LED off
    P2DIR = 0x0F;                // Set output direction for P2.0, P2.1, P2.2, P2.3


    prev = TAR;
    // wait until low
    while (P1IN & 0x80);
```

↓ continues onto next page

```

    for(;;) {

        // wait until high
        while (!(P1IN & 0x80));
        IFCount = 0;
        // count current period
        cur = TAR;

        // Toggle LED to show heart beat
        P1OUT ^= 0x20;           // Toggle P1.5
        __delay_cycles(10000);   // blink for this long
        P1OUT ^= 0x20;           // Toggle P1.5


        // calculate frequency
        // 65536 = 2^16
        // 4294967296 = 2^32
        // Count for the overflow
        cur = (cur - prev) + (65536*IFCount);
        //freq = (prev/3) + (pprev/3) + (cur/3);

        // bpm = 60,000,000 micro sec/ min. / freq

        freq = cur/1000;
        ifreq = 60000/freq;

        displayNumbers((ifreq%10000)/1000, 3);
        displayNumbers((ifreq%1000)/100, 2);
        displayNumbers((ifreq%100)/10, 1);
        displayNumbers((ifreq%10)/1, 0);

        //pprev = prev;
        prev = cur;

        // wait until low
        while (P1IN & 0x80);

    }

}
```

displayNumbers

```c
void displayNumbers(unsigned int num, unsigned int digit) {

    // Set all displays to low
    P1OUT &= ~0x3E;
    P2OUT &= ~0xFF;
    // Set STR (P1.1) to high
    P1OUT |= 0x02;

    if (digit==1)
        P1OUT |= 0x04;              // Set A0=1, A1=0
    else if (digit==2)
        P1OUT |= 0x08;              // Set A0=0, A1=1
    else if (digit==3)
        P1OUT |= 0x0C;              // Set A0=1, A1=1
    else
        ;                           // Do nothing

    P2OUT |= num;

    P1OUT &= ~0x02;                 // Set STR to low
    P1OUT |= 0x02;                  // Set STR to high


}
```

Interrupt Handler

```c
// Timer 1 interrupt service routine
#pragma vector=TIMER0_A1_VECTOR
__interrupt void Timer_A(void) {

    IFCount++;

    switch( TAIV ) {
    case  2:
        break;                          // CCR1 not used
    case  4:
        break;                          // CCR2 not used
    case 10:
        break;
    }
}
```