Data Acquisition and Display using a BPM System

Derek Chiu
Chiud5@mcmaster.ca
400062312

Table of Contents

***Abstract* – In this paper we look at the acquisition, processing and display of a continuous analog signal using an esduinoXtreme microcontroller. The purpose of this experiment is to create a system that can graph a waveform and measure its beats per minute. This paper will go over how the esduino was configured and extra hardware and software used for extended functionality. The system was tested using waves with a frequency of 1-2 Hz to replicate a human heartbeat and was able to accurately display the BPM of our test waves in addition to recreating them in Matlab.**

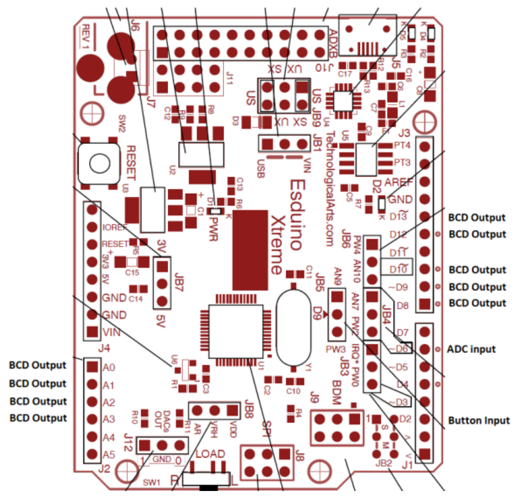
I. Introduction & Background

Technology is becoming more and more ingrained in our daily life, with computers and embedded systems finding their way into more and more products including automobiles, home appliances and clothing. One reason computers have found themselves in more and more products is their ability to acquire data from the world around them using sensors. These sensors are used to take a desired analog stimulus be it light, sound, colour, temperature or anything else and quantify them into a digital signal that the computer or in our case, microcontroller can understand. In this experiment, we forgo using a sensor instead emulating the signals we would get from the sensor using waveforms and directly feeding them into our microcontroller, the esduinoXtreme, which will be referred for the rest of this report as the esduino.

At its core, this embedded system focuses on the use of an analog to digital converter to create data we can process and turn into meaningful output. Our analog to digital converter or ADC, as it will be referred to for the remainder of this article will be used to approximate the voltage of a given waveform at certain periods of time with this data used to graphically reproduce the input signal in Matlab and display the BPM of our waveform using LEDs. The waveform reproduction and BPM output will be controlled by a button used to tell the esduino when to accept and process data and when cease activity.

II. Design Methodology

a. Final Pin Assignment Map



1 Pin Assignment Map

In this diagram we have the pin assignment map showing the pins used for ADC input, button input and BCD output.

b. Quantifying Signal Properties

The signals we will be receiving will come directly from a function generator. In general, there are 2 main components to any waveform we will receive with our microcontroller.

The first component we need to concern ourselves with is the amplitude of our wave. For our given microcontroller, the amplitude of the wave cannot exceed 5V peak-to-peak, with its peak being no more than 5V due to the limits presented by the microcontroller. The amplitude of the wave is measured using the ADC that is already installed on the esduino.

The second component we have to consider is the frequency of our wave. In our application, we will be simulating a heartbeat with a BPM of 60 to 120 BPM. This translates to a frequency of 1-2 Hz. In order to properly measure and reproduce the wave, we will need to sample at a rate faster than the fastest possible frequency we will need to measure which in this case, is 2 Hz. To accurately measure the wave, we will measure it at 10 points in a given period, giving us a sampling frequency of 20Hz.

c. Transducer

In our case, we do not have a transducer as we are feeding our esduino a signal generated by a function generator. However, the signal generated by the function generator is supposed to imitate the signal we would get from a heartbeat sensor, which is a transducer.

d. Precondition/Amplification/Buffer

As stated previously, we are using a signal generated by a function generator, so no other circuitry is required to modify our input signal for use with our microcontroller.

e. ADC

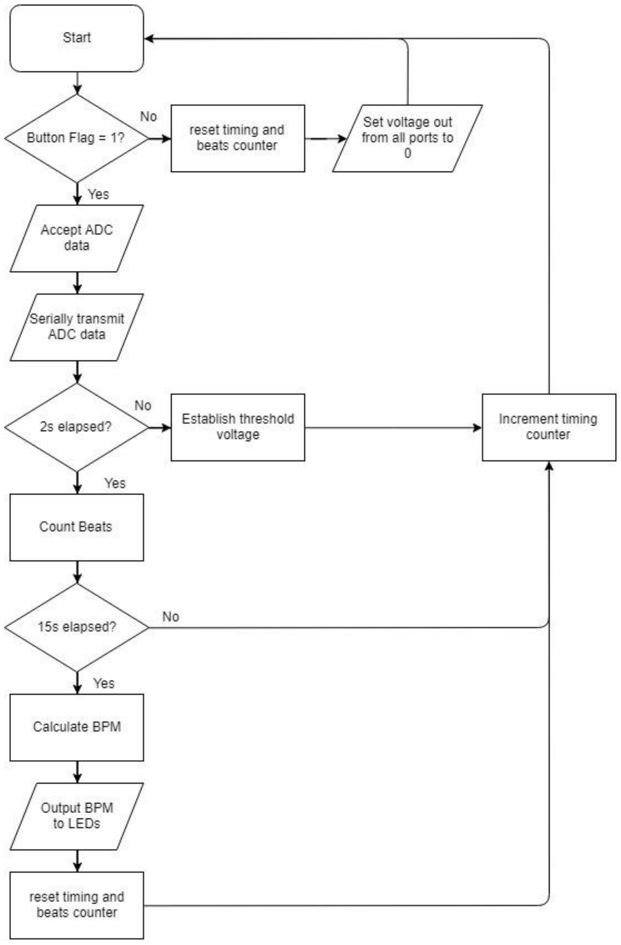
As we briefly went over previously, the ADC or analog to digital converter is used to take our analog signal in this case, a waveform, and turn it into a digital value our esduino can process. The esduino comes with an onboard ADC that uses the method of successive approximation to convert analog input into a digital value. In our application, we are using analog channel 7 as our ADC channel, with a resolution of 10 bits.

f. BPM LED Display

To display our BPM, we chose to use an array of 7 LEDs to display the BCD or binary-coded decimal value of our BPM. The LEDs are each connected to a different pin and represent a different bit. To represent our ones place, we use the pins A0-3, with A0 representing the LSB and A3 representing the MSB. Similarly, the tens place is represented using pins D8-10 along with D12, with A8 representing the LSB and D12 representing the MSB. If the BPM exceeds 100, this will be represented using the pin D13. We do not require any other LEDs to represent the hundreds place, as we are not measuring any BPM greater than 120. On the microcontroller, pins A0-A3, D8-10 and D12 are controlled by port AD, with the pin D13 controlled by port J.

g. Data Processing

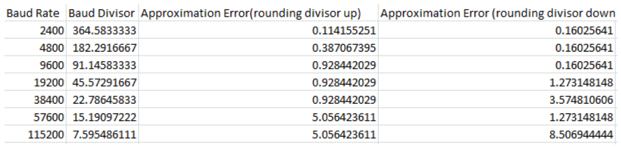
Our algorithm for processing uses a threshold voltage in order to measure the BPM of our given waveform. Our program begins by accepting values from the ADC for 2 seconds, finding the highest and lowest values processed and averages the two to determine our threshold voltage. After the threshold voltage is determined, we will raise a flag each time our input voltage exceeds the threshold voltage, lowering the flag and incrementing our beat count each the first time the input voltage is below the threshold voltage after a flag has been raised. After 15 seconds, the number of beats we have counted is multiplied by 4 to determine the BPM of our waveform and is output to the BCD LED Display. The BPM is recalculated every 15 seconds, with the BCD LED Display changed accordingly.


Start
Button Flag = 1?
No
reset timing and beats counter
Set voltage out from all ports to 0
Yes
Accept ADC data
Serially transmit ADC data
2s elapsed?
No
Establish threshold voltage
Increment timing counter
Yes
Count Beats
15s elapsed?
No
Yes
Calculate BPM
Output BPM to LEDs
reset timing and beats counter

2 Algorithm Flowchart of the Main Algorithm

h. Control/Communication

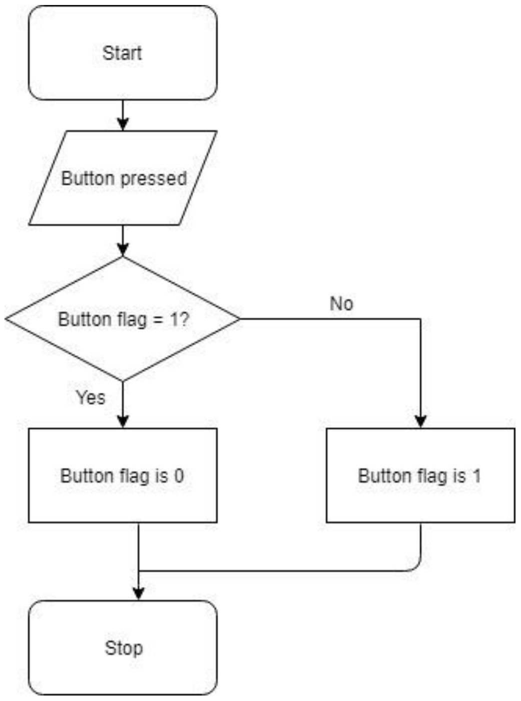
Our esduino transfers data to the computer through serial output. Our chosen baud rate is 57600, as this was the highest baud rate we could achieve with an acceptable level of loss.



Baud Rate	Baud Divisor	Approximation Error(rounding divisor up)	Approximation Error (rounding divisor down
2400	364.5833333	0.114155251	0.16025641
4800	182.2916667	0.387067395	0.16025641
9600	91.14583333	0.928442029	0.16025641
19200	45.57291667	0.928442029	1.273148148
38400	22.78645833	0.928442029	3.574810606
57600	15.19097222	5.056423611	1.273148148
115200	7.595486111	5.056423611	8.506944444

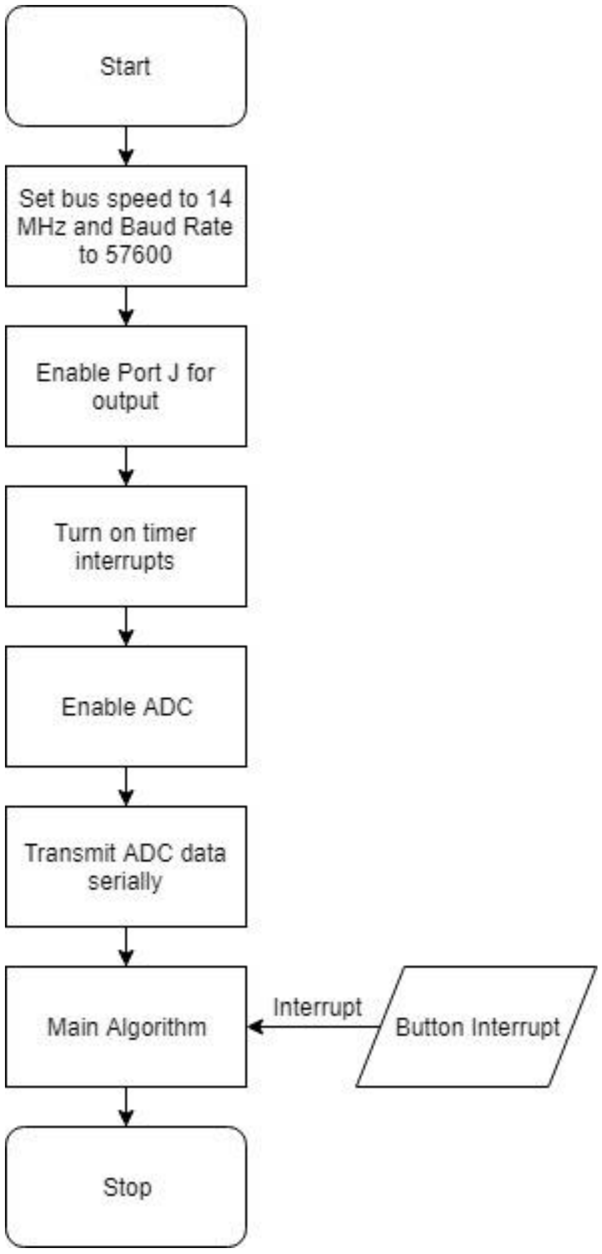
3 Baud Divisors and Approximation Error for our Bus Speed (14Mhz)

The esduino is controlled using an external button. The first time the button is pressed, the esduino will begin processing and sending data it receives from the ADC. A second press of the button will stop all data processing as well as the sending and receiving of ADC data. In addition, the BPM LED Display will turn off immediately. Pressing the button again will restart data flow and processing.



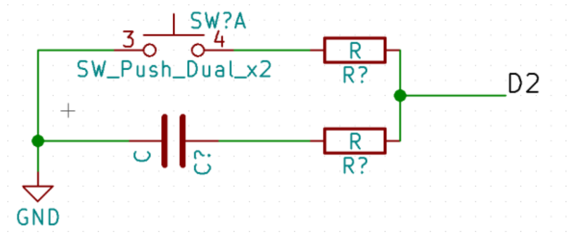
4 Algorithm Flowchart for whenever the button is pressed

i Full System Block Diagram

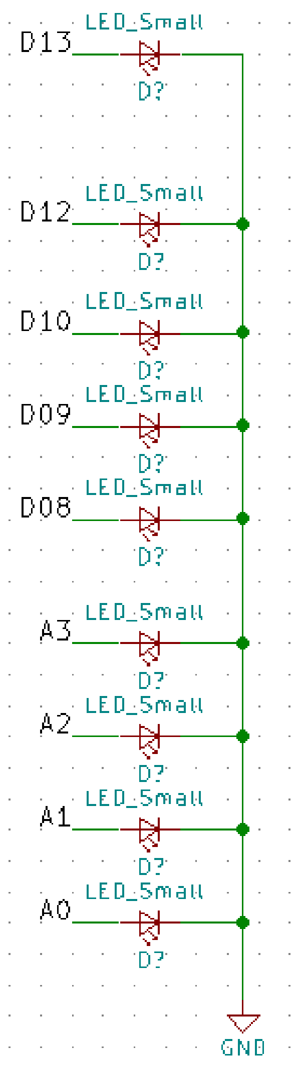


5 Full System Block Diagram

j Full System Circuit Diagram



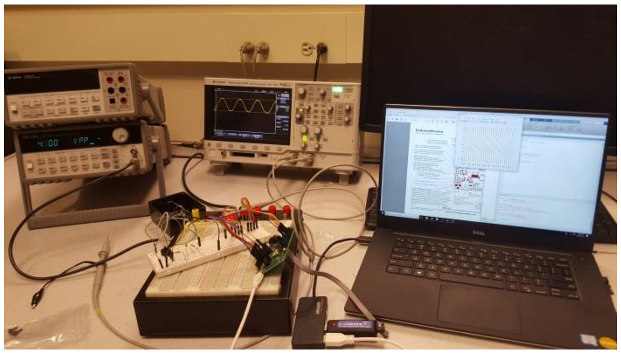
6 Circuit Diagram for Button Connected to pin D2, debounced using a capacitor

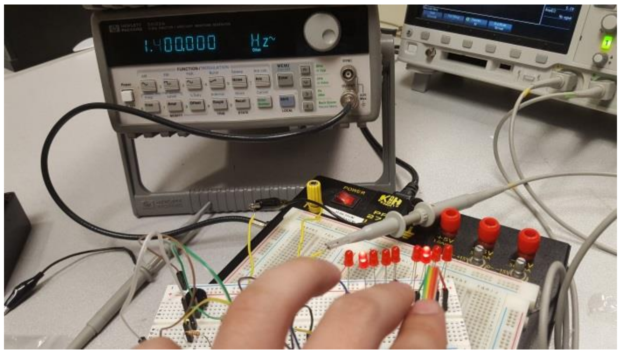


7 BPM LED Display with connected pins

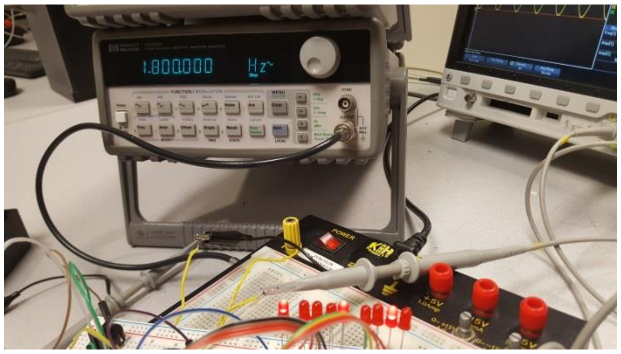
III. Results

Our system worked as intended, with the ability to accurately reproduce input waveforms up to 2 Hz in matlab, as well as the ability to measure BPM within a margin of error ±2 Beats every 15 seconds. The button worked to stop the flow and processing of data. Below are some images of testing, note that for the matlab plots, the x-axis is the number of samples taken and the y-axis is the voltage received.

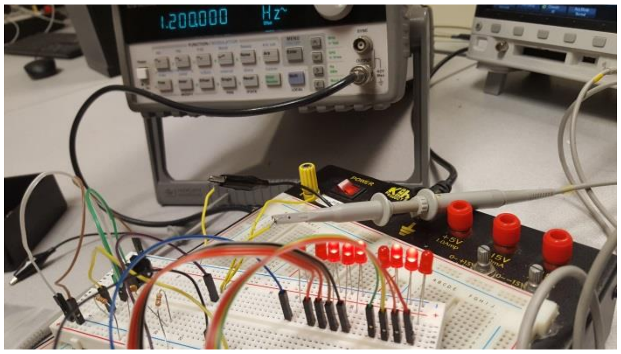

8 An Image of the completed setup, with a function generator feeding a waveform to the ADC



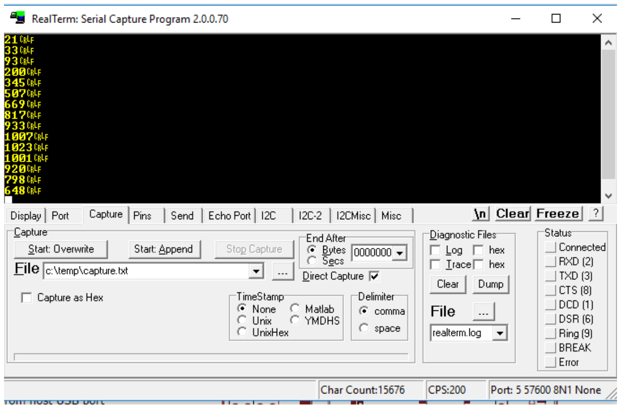
9 Measured BPM for a wave with a frequency of 1.4 Hz (84 BPM)



10 Measured BPM for a waveform with a frequency of 1.8 Hz (108 BPM)



11 Measured BPM for a waveform of 1.2 Hz (72 BPM)

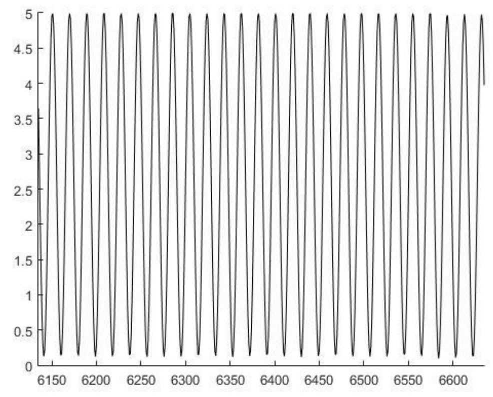

RealTerm: Serial Capture Program 2.0.0.70

```
21CRLF
33CRLF
93CRLF
200CRLF
345CRLF
507CRLF
669CRLF
817CRLF
933CRLF
1007CRLF
1023CRLF
1001CRLF
920CRLF
798CRLF
648CRLF
```

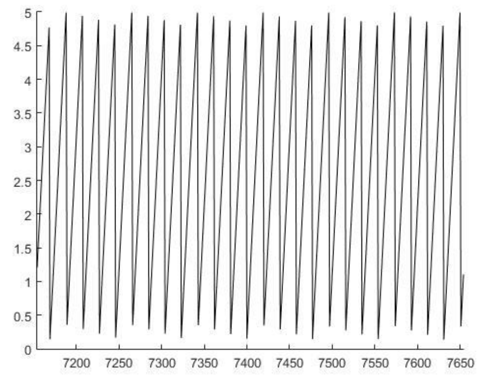
Capture — File: c:\temp\capture.txt; End After: (●) Bytes 0000000; Direct Capture [✓]; Capture as Hex [ ]; TimeStamp: (●) None; Delimiter: (●) comma; Diagnostic File: realterm.log

Char Count:15676 | CPS:200 | Port: 5 57600 8N1 None

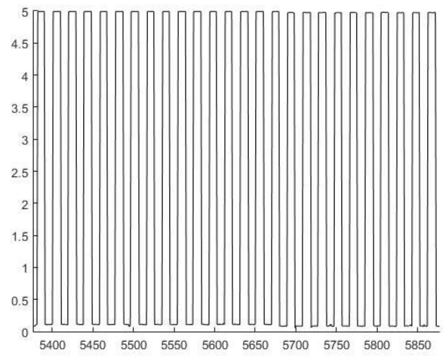
12 Serial Communication of ADC values from the Esduino to the Computer



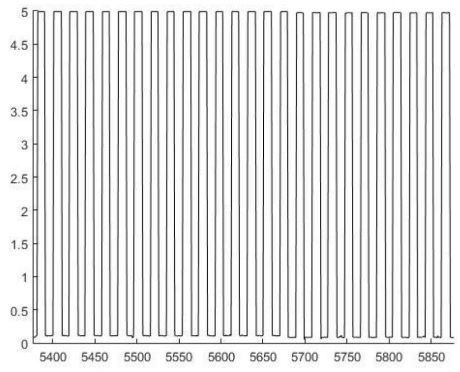
13 Recreation of a sine wave with 4.9VPP Frequency of 1Hz and offset of 2.5V Note: X-axis indicates number of samples taken, not time elapsed



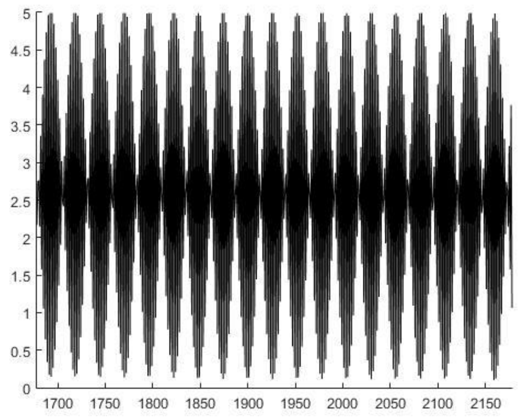
14 Recreation of a sawtooth waveform with 4.9VPP Frequency of 1Hz and offset of 2.5V Note: X-axis indicates number of samples taken, not time elapsed



15 Recreation of a square waveform with 4.9VPP Frequency of 1Hz and offset of 2.5V Note: X-axis indicates number of samples taken, not time elapsed



16 Recreation of a square wave with 4.9VPP Frequency of 1Hz and offset of 2.5V Note: X-axis indicates number of samples taken, not time elapsed



17 Recreation of a sine wave with 4.9VPP Frequency of 10Hz and offset of 2.5V Note: X-axis indicates number of samples taken, not time elapsed

IV. Discussion

1. Calculate your maximum quantization error

Because our ADC has a resolution of 10 bits and our microcontroller cannot accept a value greater than 5V, the maximum quantization error is $5/2^{10}$, or 4.8828125e-3V.

2. Based upon your assigned bus speed, what is the fastest standard serial rate you can implement. How can you verify?

The fastest standard serial rate I can implement is 57600. Any faster of a standard serial rate will result in too large of an error such that the difference between the baud rate of the esduino and the baud rate of the computer will be too different. This can be verified by assigning higher and higher baud rates to the esduino until it can no longer be processed by a computer when on the same theoretical baud rate.

3. Reviewing the entire system, what is your primary limitation on speed? How did you test this?

The primary limitation on speed is my algorithm for measuring BPM. This was tested by running my code with and without the algorithm to measure BPM, and it was found to de-synchronise with the waveform faster when the algorithm to measure BPM was present.

4. Based upon the Nyquist Rate, what is the maximum frequency of the analog signal you can effectively reproduce? What happens when your input signal exceeds this frequency?

Based upon the Nyquist Rate, the maximum frequency I can effectively reproduce is 10 Hz while still hoping to see the shape of the waveform. Any higher of a frequency, and the shape will become harder to make out.

5. Are input signals with sharp transitions (e.g., square, sawtooth, etc.) accurately reproduced? Justify your answer.

Input signals with sharp transitions are accurately reproduced, as you can see above in III. They can be accurately reproduced because the ADC will still detect their abrupt changes in voltage, which will be correctly modeled in matlab.

V. Conclusion

Overall, we were able to design an embedded system that can take accept and graphically reproduce a waveform with a maximum frequency of 2Hz, and a maximum voltage of 5V. During the construction of our embedded system, we ran into issues accepting signals from our pulse sensor, with the voltage changes being too low for our esduino to detect. This resulted in the decision to use a function generator to imitate a pulse.

VI. User Manual

The pin diagram and circuits for our embedded circuit were provided previously in this report. Please note that when using LEDs, to use LEDs rated for 5V. Otherwise, be sure to place all LEDs in series with a resistor. When arranging the LEDs, arrange them from right to left with the rightmost LED as the LSB and the leftmost LED as the MSB. When plotting the waveform in matlab, make sure to check the com port the esduino is attached to using device manager and change the com port in the matlab code to the com port indicated in the device manager. After the com port has been correctly changed, make sure to begin serial communication on the esduino before you begin plotting the waveform, to make sure that the matlab will be able to communicate serially. When feeding a waveform into the embedded system, the waveform should not have a maximum voltage of more than 5V, and a frequency of more than 2Hz, since the waveform will not be able to be modeled properly and the esduino can be damaged by a signal greater than 5V.

VII. Appendix

```
/*
 *Derek Chiu
 *400062312
 *Chiud5
 *2DP4 Final Project
 */
#include <hidef.h>      /* common
defines and macros */
#include "derivative.h"      /*
derivative-specific definitions */
#include "SCI.h"


void OutCRLF(void){
  SCI_OutChar(CR);
  SCI_OutChar(LF);
}

void setClk(void);
void delay1ms(unsigned int multiple);

unsigned short button = 0;
unsigned short val = 0;
unsigned short max = 0;
unsigned short min = 0;
unsigned short threshold = 0;
unsigned short flag = 0;
unsigned short postTotal = 0;
unsigned int beat = 0;
unsigned int count = 0;
unsigned int bpm = 0;
char lut[10] =
{0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x
07,0x08,0x09};

void main(void) {
  setClk();  //set clock speed to
14MHz
  SCI_Init(57600); //set baudrate to
57600


//Setting up ports for output
  DDRJ = 0x01;          // Set pin D13
(on board LED) to an output
  PTJ = 0x00;           // Make sure
the LED isn't on to begin with, since
that would be confusing


//Timing for Interrupts
/*
 * The next six assignment statements
configure the Timer Input Capture
 */
  TSCR1 = 0x90;    //Timer System
Control Register 1
                    // TSCR1[7] = TEN:
Timer Enable (0-disable, 1-enable)
                    // TSCR1[6] =
TSWAI:  Timer runs during WAI (0-
enable, 1-disable)
                    // TSCR1[5] =
TSFRZ:  Timer runs during WAI (0-
enable, 1-disable)
                    // TSCR1[4] =
TFFCA:  Timer Fast Flag Clear All (0-
normal 1-read/write clears interrupt
flags)
                    // TSCR1[3] = PRT:
Precision Timer (0-legacy, 1-
precision)
                    // TSCR1[2:0] not
used

  TSCR2 = 0x00;    //Timer System
Control Register 2
                    // TSCR2[7] = TOI:
Timer Overflow Interrupt Enable (0-
inhibited, 1-hardware irq when TOF=1)
                    // TSCR2[6:3] not
used
                    // TSCR2[2:0] =
Timer Prescaler Select: See Table22-12
of MC9S12G Family Reference Manual
r1.25 (set for bus/1)


  TIOS = 0xFE;     //Timer Input
Capture or Output capture
                    //set TIC[0] and
input (similar to DDR)
  PERT = 0x01;     //Enable Pull-Up
resistor on TIC[0]

  TCTL3 = 0x00;    //TCTL3 & TCTL4
configure which edge(s) to capture
  TCTL4 = 0x02;    //Configured for
falling edge on TIC[0]

/*
```

```
 * The next one assignment statement
configures the Timer Interrupt Enable
 */

  TIE = 0x01;      //Timer Interrupt
Enable

/*
 * The next one assignment statement
configures the ESDX to catch Interrupt
Requests
 */

    EnableInterrupts; //CodeWarrior's
method of enabling interrupts


    ATDCTL1 = 0x27;   // Set
resolution to 10 bits
    ATDCTL3 = 0x88;     // right
justified, one sample per sequence
    ATDCTL4 = 0x06;     // prescaler =
6; ATD clock = 14MHz / (2 * (6 + 1))
== 1.0MHz
  ATDCTL5 = 0x27;       // continuous
conversion on channel 7

   //Main Algorithm begins

  for(;;) {

    if(button == 1){
      val=ATDDR0;
      SCI_OutUDec(val);
      OutCRLF();
      //counts for 2 secons initally
to establish a threshold voltage
      if(count == 0){
        max = val;
        min = val;
      } else if(count < 40){
        if(val > max){
          max = val;
        }
        if(val < min){
          min = val;
        }
      } else if(count == 40){
        threshold = (max+min)/2;
        max = val;
      } else { //counts the number of
beats in 15 seconds
        min = max;
        max = val;
        if(flag == 0 && (min+max)/2 >
(threshold)){
          flag = 1;
        }else if (flag ==1 &&
(min+max)/2 < (threshold)){
          flag = 0;
          beat = beat +1;
        }
      }
      //calculates and outputs BPM
every 15 seconds
      if(count == 341){
      //change ports to ad for bcd
output
        DDR0AD = 0x0F;
        DDR1AD = 0x0F;
        bpm = beat*4;
        if(bpm > 100){
          PTJ = 0x01;
          bpm = bpm-100;
        }else{
          PTJ = 0x00;
        }
        PT0AD = lut[bpm/10];
        bpm = bpm%10;
        PT1AD = lut[bpm];
        count = 41;
        beat = 0;
      //change back to adc
        ATDCTL1 = 0x27;   // Set
resolution to 10 bits
        ATDCTL3 = 0x88;     // right
justified, one sample per sequence
        ATDCTL4 = 0x06;     //
prescaler = 6; ATD clock = 14MHz / (2
* (6 + 1)) == 1.0MHz
        ATDCTL5 = 0x27;     //
continuous conversion on channel 7

      }
      delay1ms(50);
      count = count + 1;
    }else{
     count = 0;
     beat = 0;
     PTJ = 0x00;
     PT0AD = 0x00;
     PT1AD = 0x00;
    }
  } /* loop forever */
  /* please make sure that you never
leave main */
}

/*
 * This is the Interrupt Service
Routine for TIC channel 0 (Code
Warrior has predefined the name for
you as "Vtimch0"
 */
```

```
interrupt  VectorNumber_Vtimch0 void
ISR_Vtimch0(void)
{
  unsigned int temp;

  if(button == 0){
      button = 1;
    } else {
      button = 0;
    }


  temp = TC0;        //Refer back to
TFFCA, we enabled FastFlagClear, thus
by reading the Timer Capture input we
automatically clear the flag, allowing
another TIC interrupt
}

void delay1ms(unsigned int multiple){
  int ix;/* enable timer and fast
timer flag clear */
  TC2 = TCNT + 14000;
  for(ix = 0; ix < multiple; ix++) {
    while(!(TFLG1_C2F)); {
      TC2 += 14000;
    }
  }
}

void setClk(void){ //Set E-Clock speed
to 14 MHz

 CPMUREFDIV = 0x00;
 CPMUSYNR = 0x5A;
 CPMUPOSTDIV = 0x01;
 CPMUCLKS = 0x80;
 CPMUOSC = 0x00;

 while(!(CPMUFLG & 0x08));

}
```