2DP4: Microprocessor Systems Lab 4

Instructor: Dr. Doyle

Vaibhav Ariyur – L04 – ariyurv- 400012535

Contents

Introduction and Background

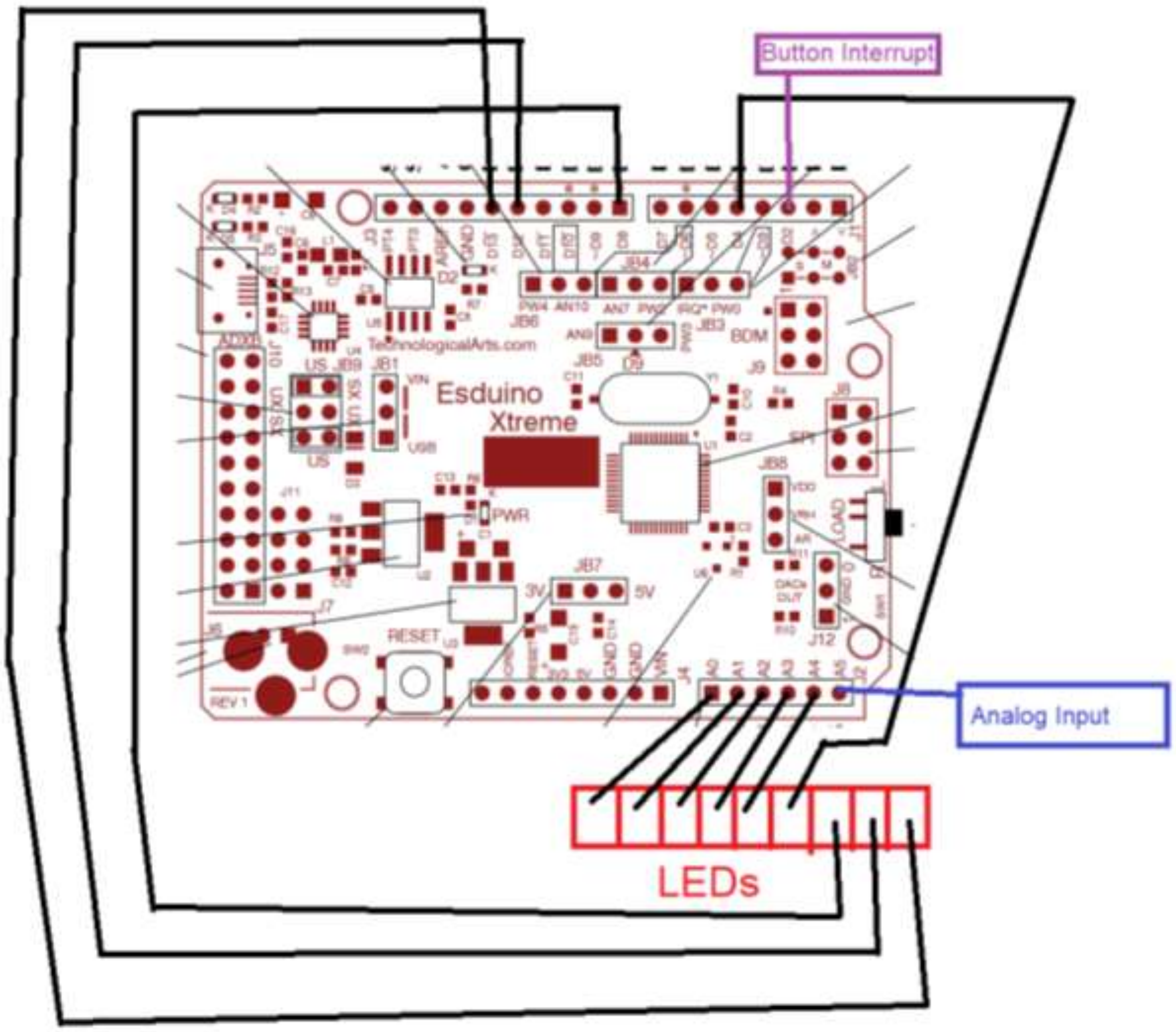
An ADC is a device which takes in analog input and convert it to digital values. This is done by taking in some measurement in real-life (i.e. distance, temperature, and speed), converting it to a voltage within a certain range (I.e., 0-5V). Then, the temperature within that range is converted into a binary value. The higher the bits of binary values there are, the greater the resolution. A 16 bit range will be much more accurate of measurements than a 4 bit range.

Electric thermometers use a thermistor as a peripheral component to measure temperature. A thermistor changes it's resistance as temperature changes. This change in resistance causes different voltages to arise. Then, the microcontroller kicks in and converts the voltage to a binary value, and then uses a formula to convert that bit value back to temperature for the user to read. [1]

In this project, a transducer converts pulse strength to a voltage. The purpose of this project is to measure a user's current heart rate in BPM. Since the heart beats cyclically, an algorithm will measure the length of one period in order to extrapolate BPM.

Design Methodology

Pin Mappings



Quantifying Signal Properties

Since ATDCTL1 was set to convert analog values to 10 bits of data,

Transducer

The pulse monitor converted pulse strength to a voltage. The result of converting pulse, which occurs periodically, to a voltage, resulted in a cyclical wave form. For this project, an input sine wave was used in place of the pulse monitor.

Precondition/Amplifier/Buffer

No precondition, amplifier, or buffer was used. The inputted sine wave took the form of an ideal signal input.

ADC

To achieve analog to digital conversion, VRH was set to 3V via JB7 and VLH was 0V. Since the signal would have a 10-bit value, the following formula was used to convert raw bit values to voltages:

$$Vin = rawDecimalValue * \left(\frac{3}{1024}\right)$$

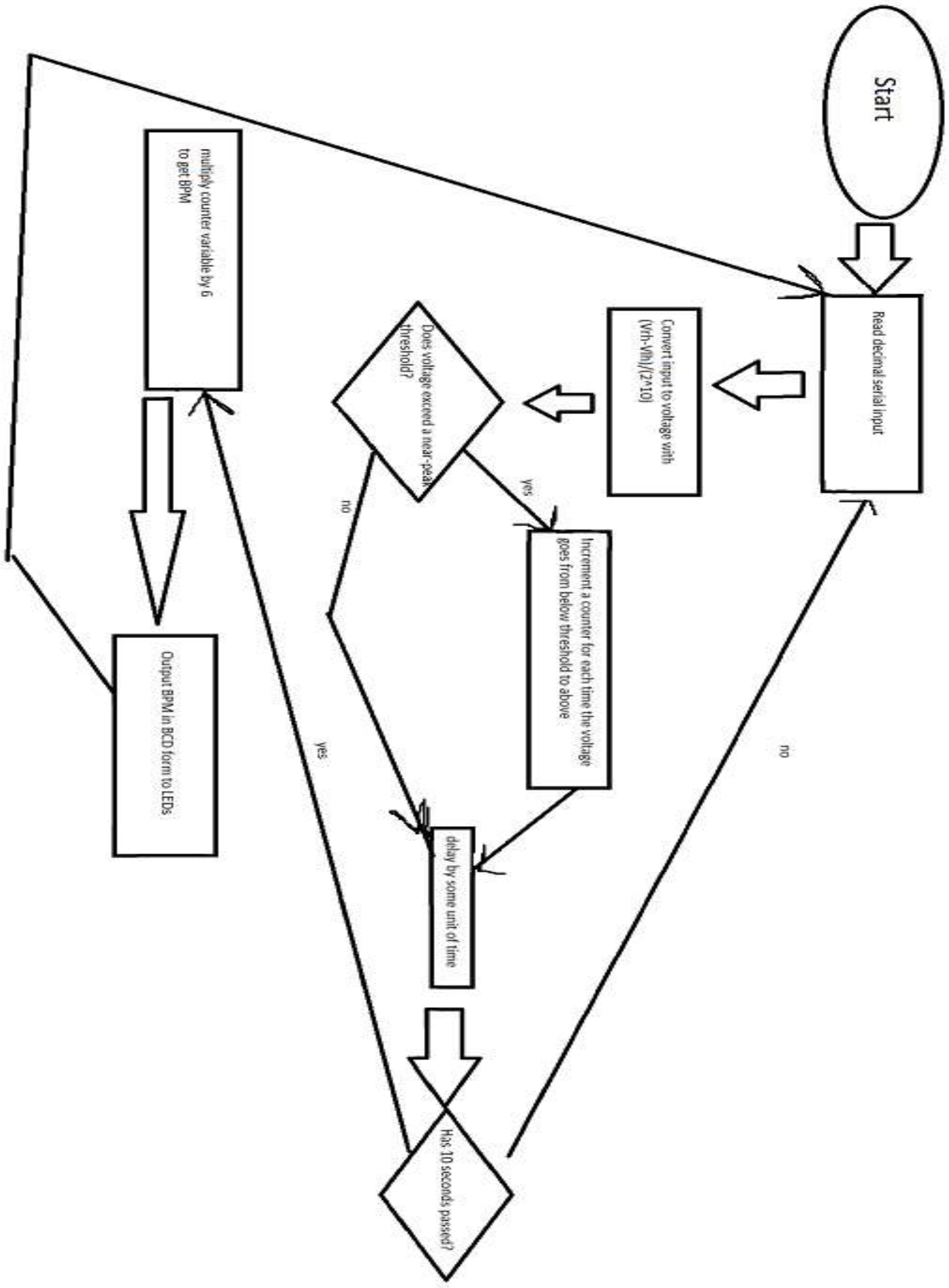
3V was used as VRH instead of 5V in order for the system to be more sensitive to signal changes. If 5V could be used instead of 3V in the scenario where the analog signal would be assigned a 12-bit value.

BPM LED Display (BCD)

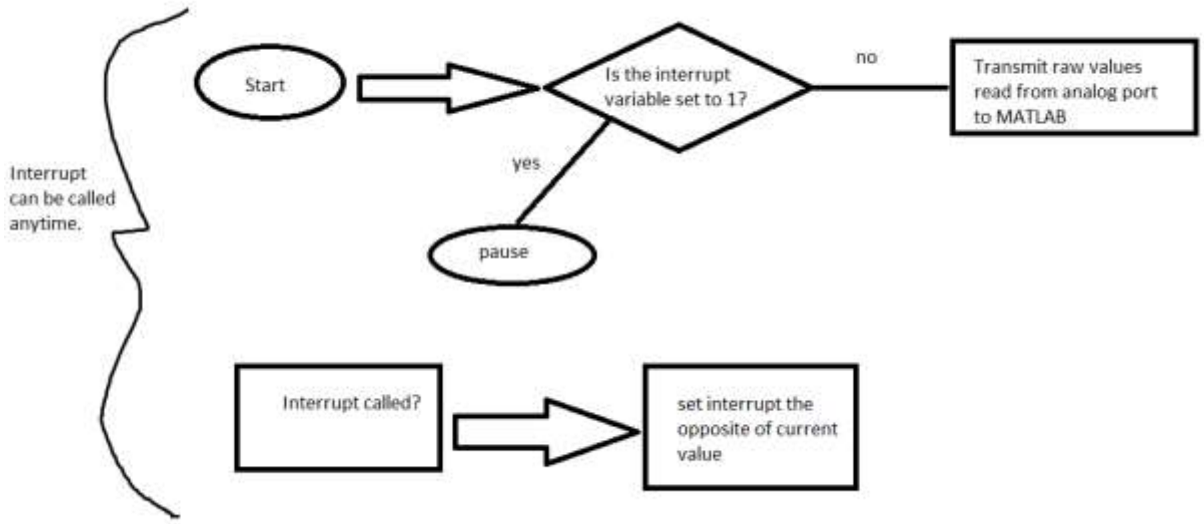
As shown in the pin mapping:

- PT1AD register was important for the display of the 6 most significant digits
- PT0AD register was important for the display of the 2nd and 3rd least significant digit
- Port J was important for the display of the least significant digit

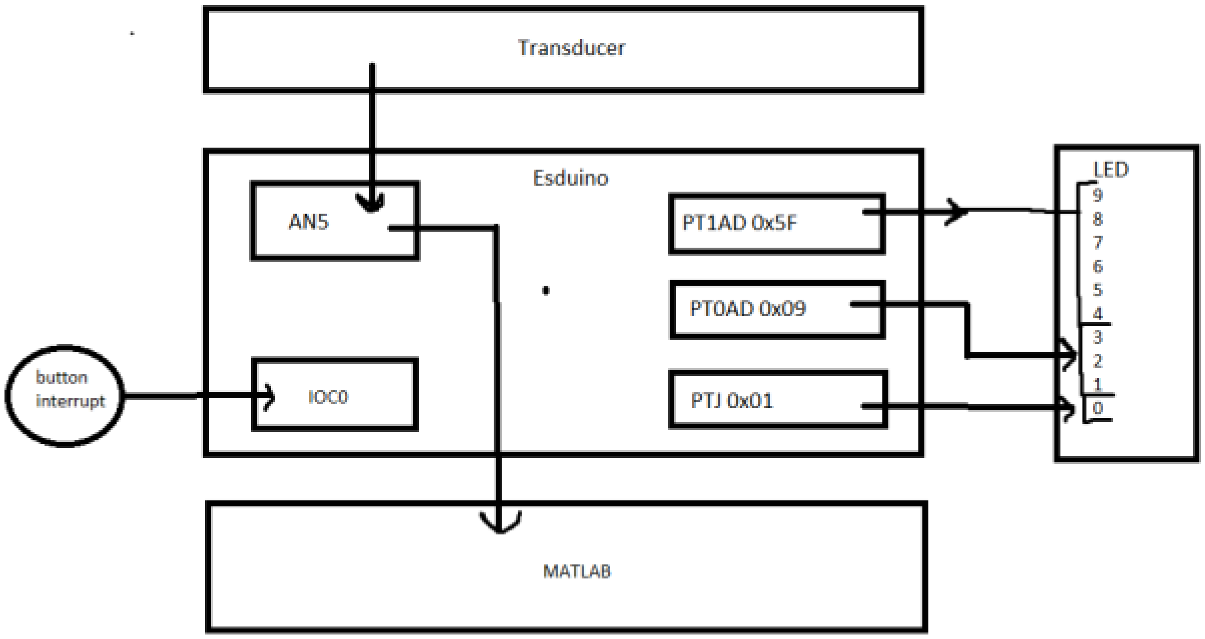
Data Processing


Start
Read decimal serial input
Convert input to voltage with (Vrh-Vrl)/(2^10)
Does voltage exceed a near-peak threshold?
yes
no
Increment a counter for each time the voltage goes from below threshold to above
delay by some unit of time
Has 10 seconds passed?
no
yes
multiply counter variable by 6 to get BPM
Output BPM in BCD form to LEDs

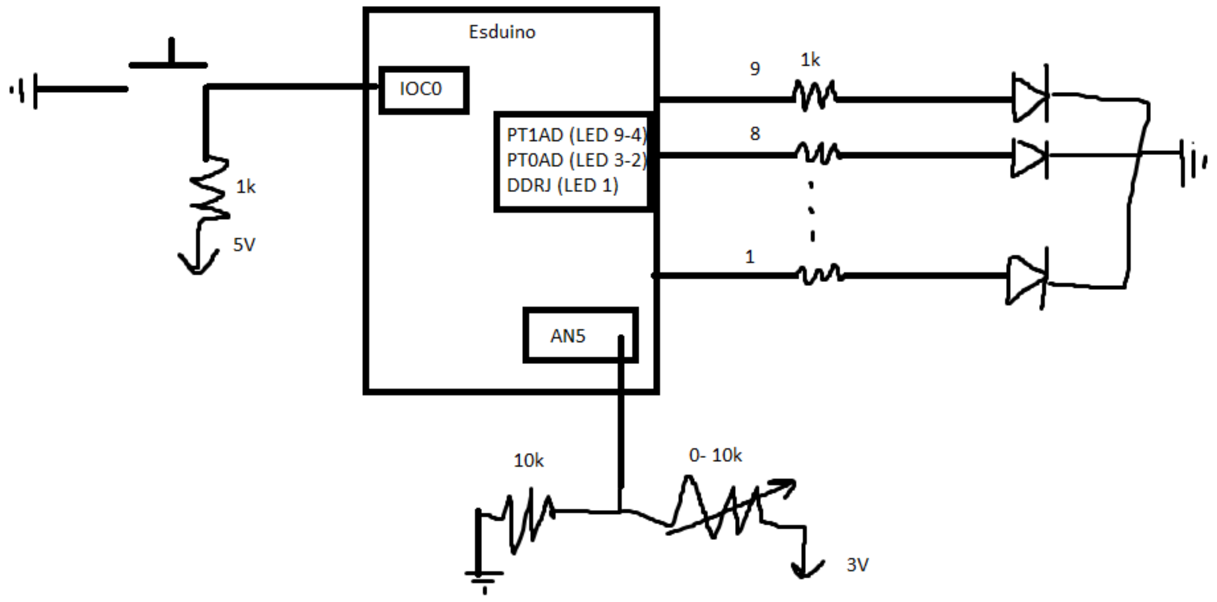
Control/Communication


Interrupt can be called anytime.
Start
Is the interrupt variable set to 1?
no
Transmit raw values read from analog port to MATLAB
yes
pause
Interrupt called?
set interrupt the opposite of current value

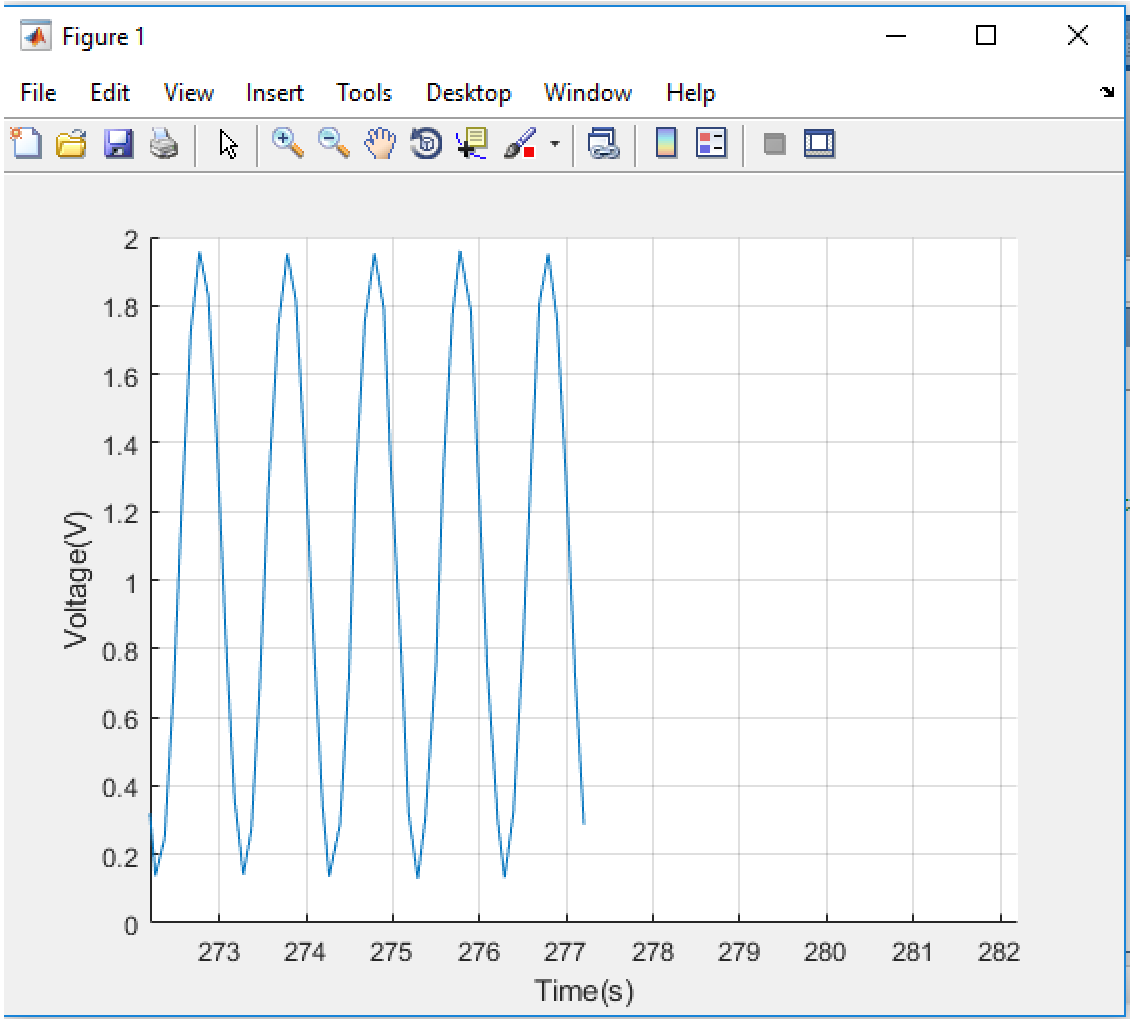
Block Diagram



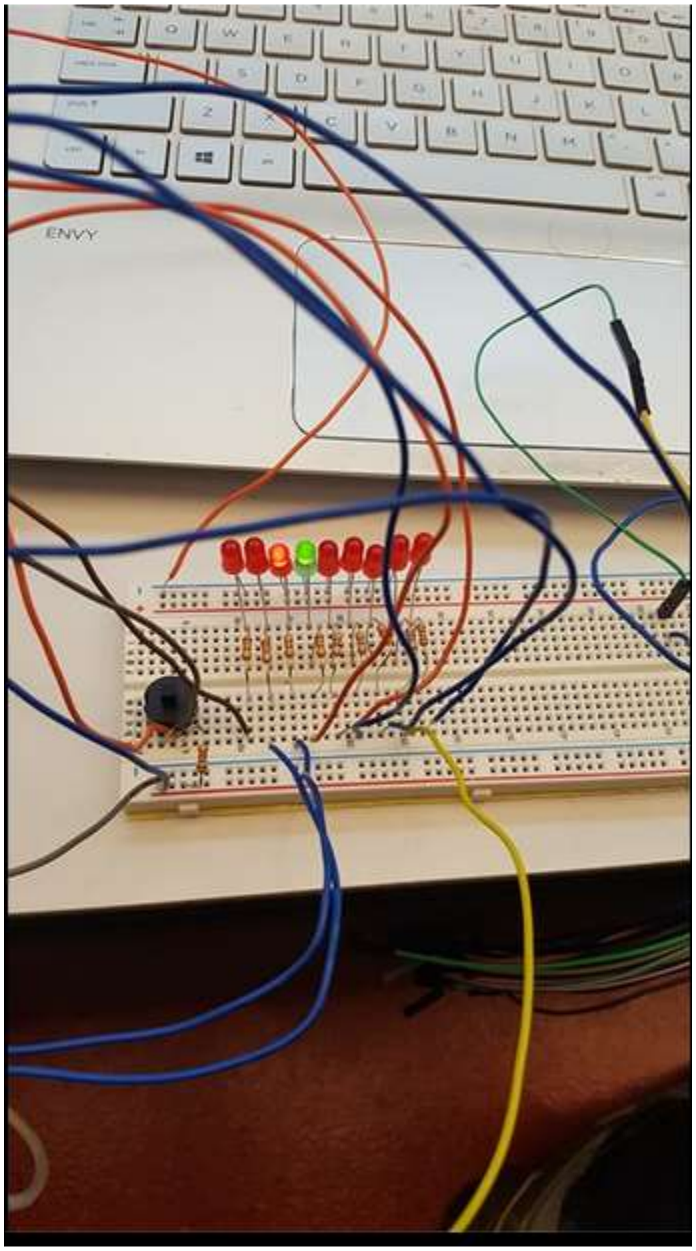
Full System Circuit Schematic



Results



1 Hz wave input display on MATLAB



BCD display of 1 Hz wave input

Discussion

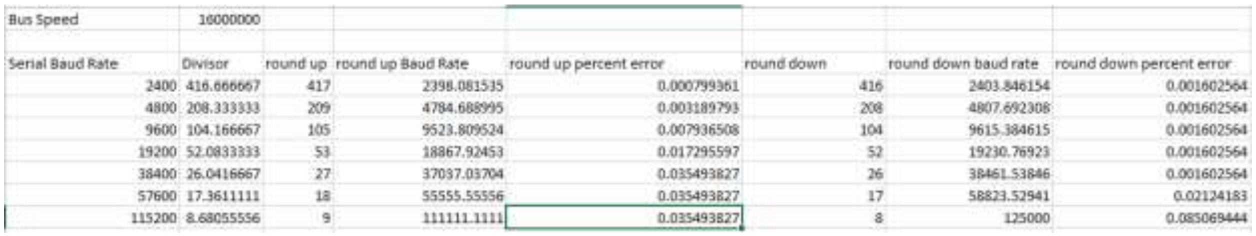
What is the maximum quantization error?

Since my voltage range is 0 to 3.3V with 1024 bit resolution, the maximum quantization error would be:

$$\frac{3.3-0}{1024} = 3.3 * 10\text{^}(-3)$$

This means that a maximum difference of 3.3mV would be missed out by the ADC.

Based upon the assigned bus speed, what was the maximum standard serial communication rate that can be implemented? How can this be verified?



Bus Speed	16000000						
Serial Baud Rate	Divisor	round up	round up Baud Rate	round up percent error	round down	round down baud rate	round down percent error
2400	416.666667	417	2398.081535	0.000799361	416	2403.846154	0.001602564
4800	208.333333	209	4784.688995	0.003189793	208	4807.692308	0.001602564
9600	104.166667	105	9523.809524	0.007936508	104	9615.384615	0.001602564
19200	52.0833333	53	18867.92453	0.017295597	52	19230.76923	0.001602564
38400	26.0416667	27	37037.03704	0.035493827	26	38461.53846	0.001602564
57600	17.3611111	18	55555.55556	0.035493827	17	58823.52941	0.02124183
115200	8.68055556	9	111111.1111	0.035493827	8	125000	0.085069444

The maximum standard serial communication rate implemented is 115200 bits per second. This can be verified by the following formulas:

$$IdealBaudDivisor = \frac{busSpeed}{16 * IdealBaudRate}$$

$$CalculatedBaudRate = \frac{busSpeed}{16 * roundedBaudDivisor}$$

$$\%error = abs(\frac{IdealBaudRate - CalculatedBaudRate}{IdealBaudRate})$$

First, for the baud rates 2400, 4800, 9600, 19200, 38400, 57600, and 115200, the ideal baud divisor must be calculated. However, the baud divisor must be an integer, so a Calculated Baud Rate must be undergone with the rounded-up and rounded-down versions of each divisor. Finally, the %error must be calculated, and the highest baud rate with a %error<6% will be the maximum baud rate. In this case, it was 115200 bits per second.

Reviewing the entire system, what was the primary limitation on speed?

The primary limitation on speed would be how the Delay1ms is used, and the decision to use a Delay1ms instead of Delay1us (microsecond) or even Delay1ns (nanosecond). Bus speed is not a primary limitation on speed because no matter what bus speed is used, it can all be modified to match 1ms or 1us with the TCNT register (i.e. instead of TCNT+16000, do TCNT+14000).

To test what the primary limitation on speed is, one should play around with various bus speeds, baud rates, and millisecond delays ceteris-paribus. Then, one should see how quickly values from the analog input are samples on MATLAB.

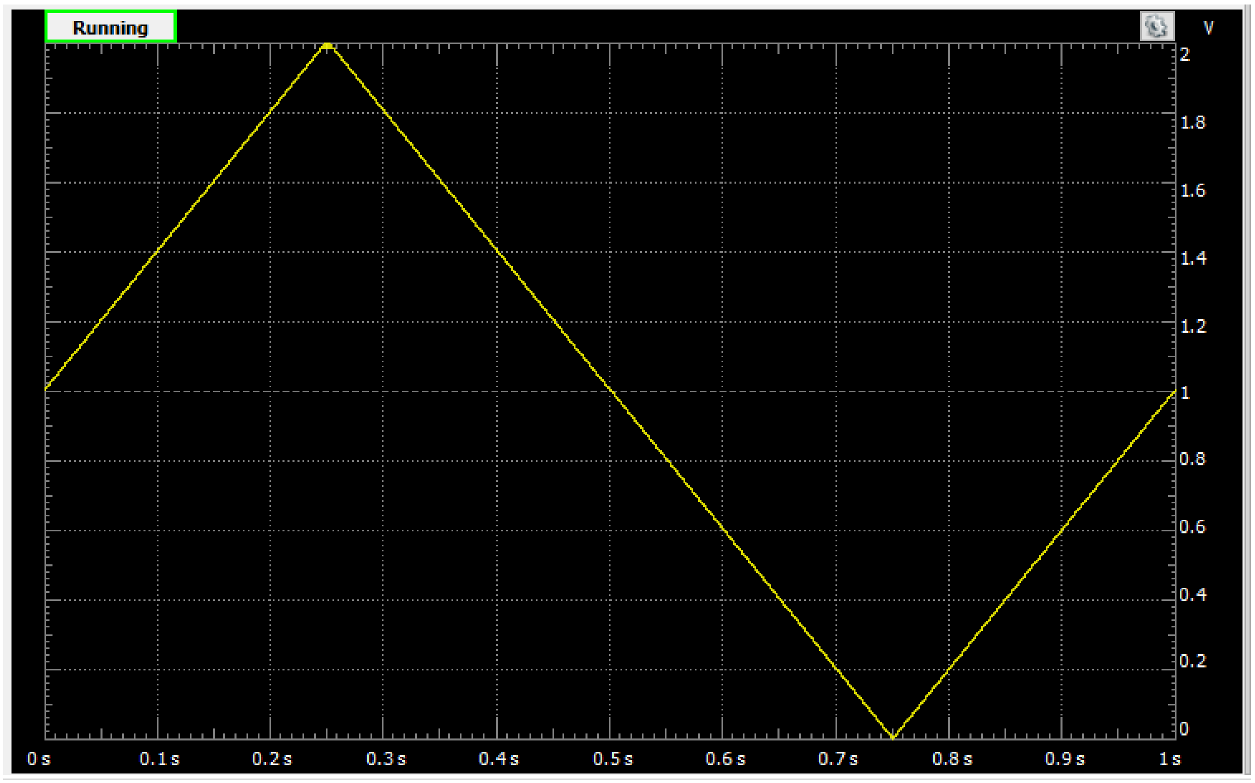
Based upon the Nyquist Rate, what is the maximum frequency of the analog signal you can effectively reproduce? What happens when your input exceeds this frequency?

Since the expected input is anywhere between 1-2Hz (60-120 bpm), the maximum frequency to effectively reproduce all signals within this range is 10Hz. However, if I were to sample the 120 bpm to the maximum sampling rate, it would be 20Hz. If my input were to exceed this frequency, nothing too noticeable would happen until the sampling frequency become 0.2 times the input. By then, the plot would miss a lot of significant inputs and the curve would look overly simplified, and may even look like a singular straight line.

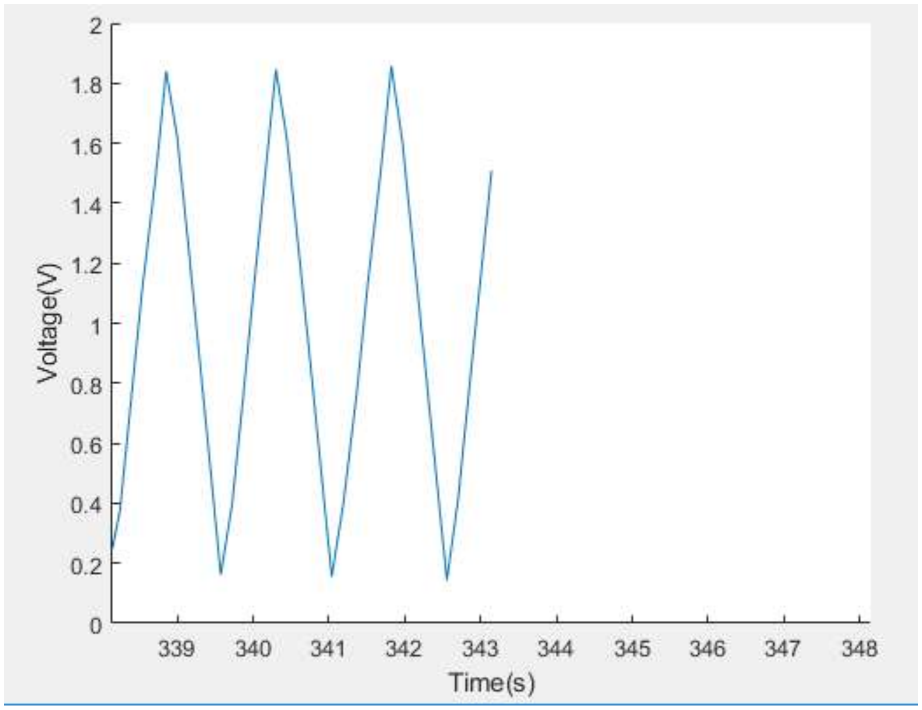
Are sharp signals accurately reproduced?

Sawtooth

Input

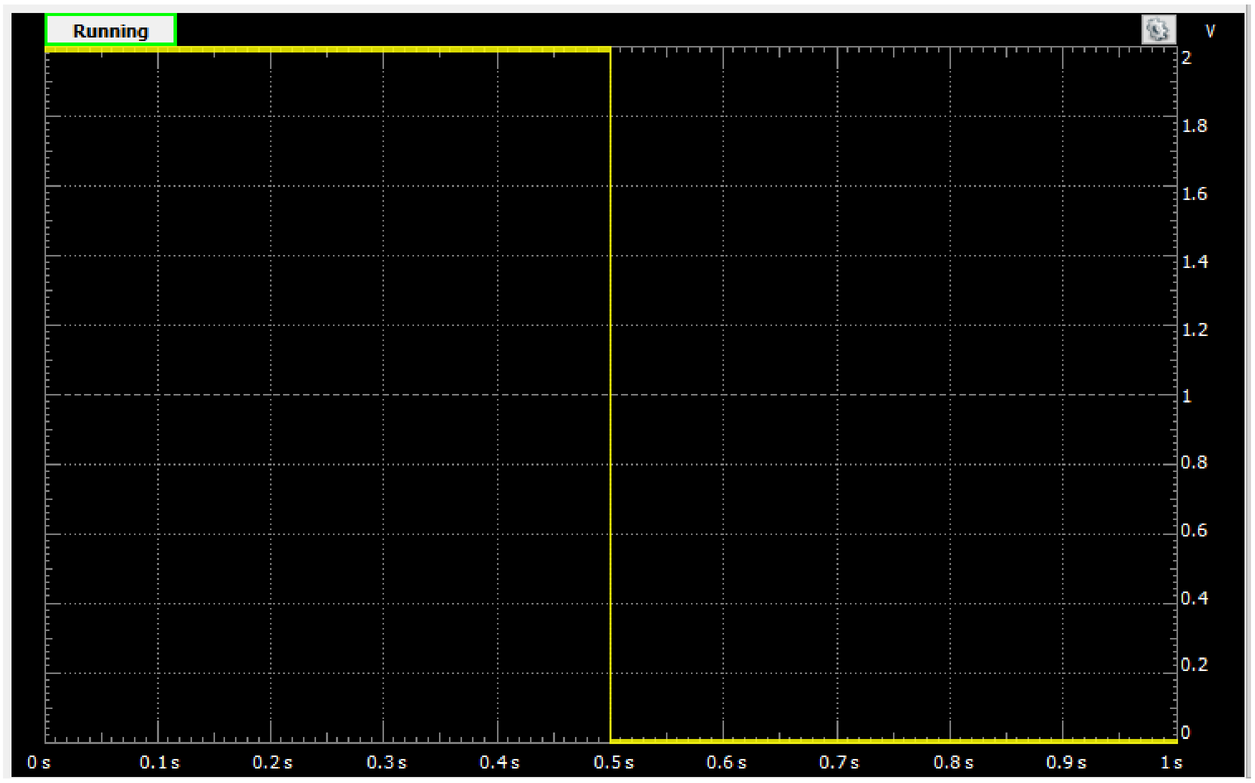


MATLAB display

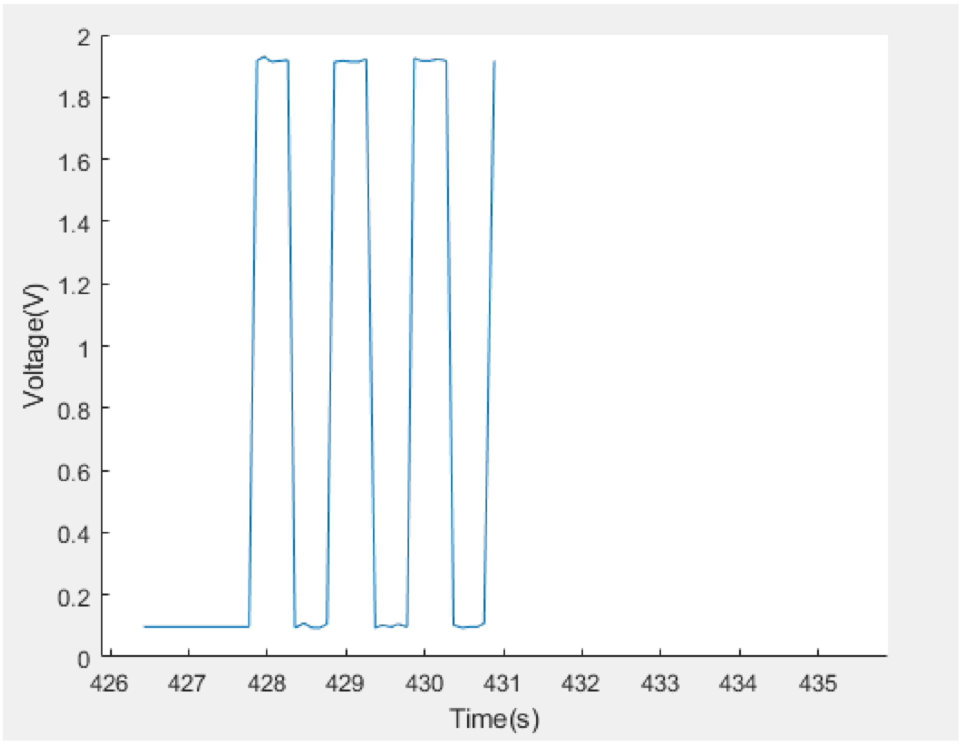


Rectangular

Input



MATLAB



Conclusion

To conclude, the transducer, analog-to-digital conversion, BPM BCD display, and graphical display are all functional. First the clock rate was set. In order to set the clock rate, the SYNR and REFDIV registers had to be altered so that 1 cycle took 62.5ns (16MHz). Then, the baud rate was set to the highest amount which was 115200 bits per second. In order to find the bit rate, an up-rounded and down-rounded baud divisor was calculated for all possible bit rates up to 115200. Then, a percent error was calculated, and the highest rate with less than 6% error was chosen. Then, raw values read from the transducer from the AN5 port would be transmitted to MATLAB. ADC would be done in MATLAB by subtracting Vdd=3V by Vground, and dividing it by the 10 bit resolution (2^{10}). This voltage value would be graphically displayed with a sampling rate of 10Hz, which is 10 times the lowest anticipated value (60 bpm). Finally, the frequency was calculated by determining how many times a threshold value was passed in 10 seconds, and was displayed on 9 LEDs in BCD form.

Appendix

Esduino Code

```
/*Include*/
  #include <hidef.h>
  #include "derivative.h"
  #include "SCI.h" //Serial Communication header file


/*Prototypes*/
  void setClk(void);
  void delay1ms(unsigned int multiple);
  void OutCRLF(void);

int i =0;
short val=0;
int flipflop=0;
int counter=0;
int tenSecCounter=0;
int BPM;
int tens;
int ones;
short button;



void main(void) {


  setClk();

  SCI_Init(115200);
```

```
ATDCTL1=0x25;
ATDCTL3=0x88;
ATDCTL4=0x00;
ATDCTL5=0x25;
ATDDIEN=0x0020;

  /* put your own code here */
//Set Environment



//Set Ports
DDRJ = 0x01;      //set all port J as output
DDR1AD=0x5F;
DDR0AD=0x09;


TSCR1 = 0x90;

TSCR2 = 0x00;


TIOS = 0xFE;

PERT = 0x01;

TCTL3 = 0x00;
TCTL4 = 0x02;

TIE = 0x01;


  EnableInterrupts;

for(;;){
  while(button==0x01){
  }
  val=ATDDR0;
  delay1ms(100);
  SCI_OutUDec(val); OutCRLF();
  if(val>600){
    if(flipflop==0){
      flipflop=1;
      counter++;
  }
  }else {
     flipflop=0;
    }
  tenSecCounter++;
  if(tenSecCounter==100){
      BPM=counter*6;
      tenSecCounter=0;
      counter=0;
      flipflop=0;
  }
  tens=(BPM/10)%10;
  ones=BPM%10;
```

```
    if(BPM>100){
      PT1AD_PT1AD0=1;

    } else{
      PT1AD_PT1AD0=0;
    }
    //Ten's digit
    if(tens>=8){
      PT1AD_PT1AD1=1;
    }else{
      PT1AD_PT1AD1=0;
    }
    if(tens%8>=4){
      PT1AD_PT1AD2=1;
    }else{
      PT1AD_PT1AD2=0;
    }
    if(tens%4>=2){
      PT1AD_PT1AD3=1;
    }else{
      PT1AD_PT1AD3=0;
    }
    if(tens%2>=1){
      PT1AD_PT1AD4=1;
    }else{
      PT1AD_PT1AD4=0;
    }

    //One's digit
    if(ones>=8){
      PT1AD_PT1AD6=1;
    }else{
      PT1AD_PT1AD6=0;
    }
    if(ones%8>=4){
      PT0AD_PT0AD0=1;
    }else{
      PT0AD_PT0AD0=0;
    }
    if(ones%4>=2){
      PT0AD_PT0AD3=1;
    }else{
      PT0AD_PT0AD3=0;
    }
    if(ones%2>=1){
      PTJ=0x01;
    }else{
      PTJ=0x00;
    }
  }
}

/////////////////////////////////////////////////////
void delay1ms(unsigned int k){
  int ix;
//enable timer and fast flag clear
     TSCR1 = 0x90;
```

```
// disable timer interrupt, select prescaler to 1
     TSCR2 = 0x00;

// enable OC0(not necessary)
     TIOS |= 0x02;

     TC1 = TCNT + 16000;

     for(ix= 0; ix<k; ix++){
         while(!(TFLG1_C1F));

         TC1+= 16000;
     }

     TIOS &= ~0x01;
}

void setClk(void){

  //CPMUPROT = 0;                //Protection of clock configuration is
disabled
  //CPMUCLKS = 1;

  CPMUSYNR=0x07;     //set PLL multiplier (0x42 = 01 000010)
  CPMUREFDIV = 0x03;//set PLL divider (0x80 = 10 000000)
  CPMUPOSTDIV=0x00;
  CPMUCLKS=0x80;           // set Post Divider
  CPMUOSC = 0x80;                // Enable external oscillator
  while (!(CPMUFLG &0x08)) {}  // wait for it
  //CPMUPLL = CPMUCLKS;            // Engage PLL to system
}

void OutCRLF(void){
  SCI_OutChar(CR);
  SCI_OutChar(LF);
  //PTJ ^= 0x20;          // toggle LED D2
}

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

  button ^= 0x01;      //Toggles Button variable
  delay1ms(100);

  temp = TC0;       //allows another TIC interrupt
}
```

Matlab Code

```
delete(instrfindall);
s = serial('com6'); %initialize serial communication
set(s,'BaudRate',56700); %speed at which we transmit (bits per second)
s.Terminator = 'CR'; % sets the terminator to CR/ new line
fopen(s);
```

```
grid on; %adds grid lines
j = 0;
data = 0;
t = 0;
tic ; % starts a stopwatch timer to measure performance

while (1) %loop
    j = j + 1;
    tline = fgetl(s) %reads serial data and store it in an array
    data(j) = (str2double(tline))*(3/1024);
    t(j) = toc; %update clock, toc reads the elapsed time from the stopwatch
timer
    if j > 1
        xlim([t(j)-5 t(j)+5]); %update the x-axis
        line([t(j-1) t(j)],[data(j-1) data(j)]); %plot the data
        xlabel('Time(s)'); %label the axis
        ylabel('Voltage(V)');
        hold on;
        drawnow %disp(data(j)); %originally was i
    end
end
fclose(s); %close the ports used for serial communication
delete(s);
clear s;
```

Works Cited

[1]H. Garden, H. Garden and H. Appliances, "How Thermometers Work", *HowStuffWorks*, 2018. [Online]. Available: https://home.howstuffworks.com/therm3.htm. [Accessed: 07- Apr- 2018].