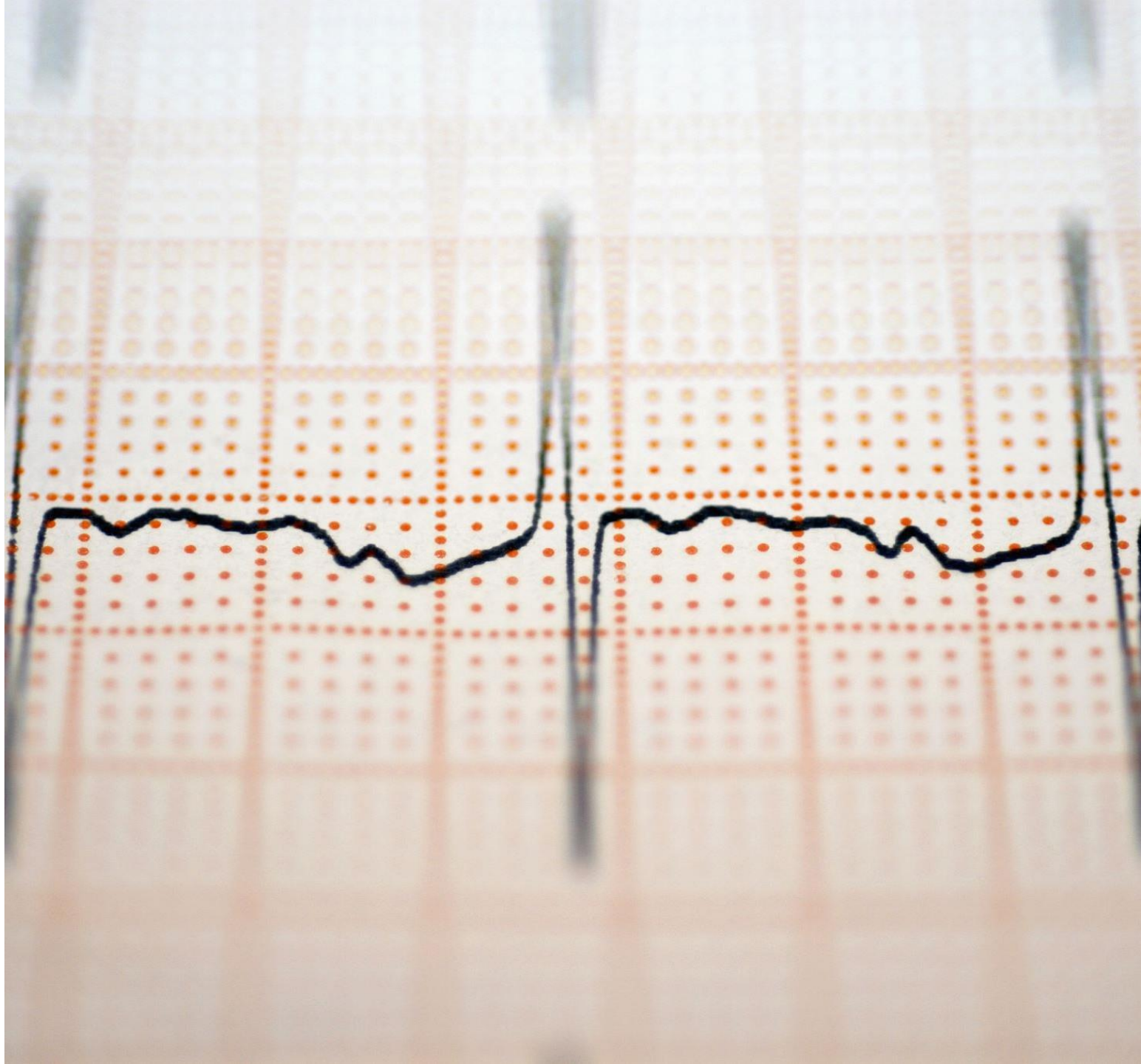
Cardiac Monitoring System

Heart Pulse Sensor

Prepared by:
Gerard Aingello Cruz, Damian Flores-Menjivar, Jackie Martinez, and Chanel Williams



Introduction (Project Objectives)

Students showcase understanding of course through applying a microcontroller with a real-life application

Design and implement a heart pulse monitoring system using an Arduino microcontroller, LCD screen and a heart pulse sensor.

Equipment

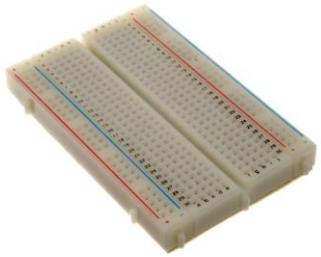
Arduino ~~UNO~~ Mega

Potentiometer

LCD Screen

Push Button

Pulse Sensor



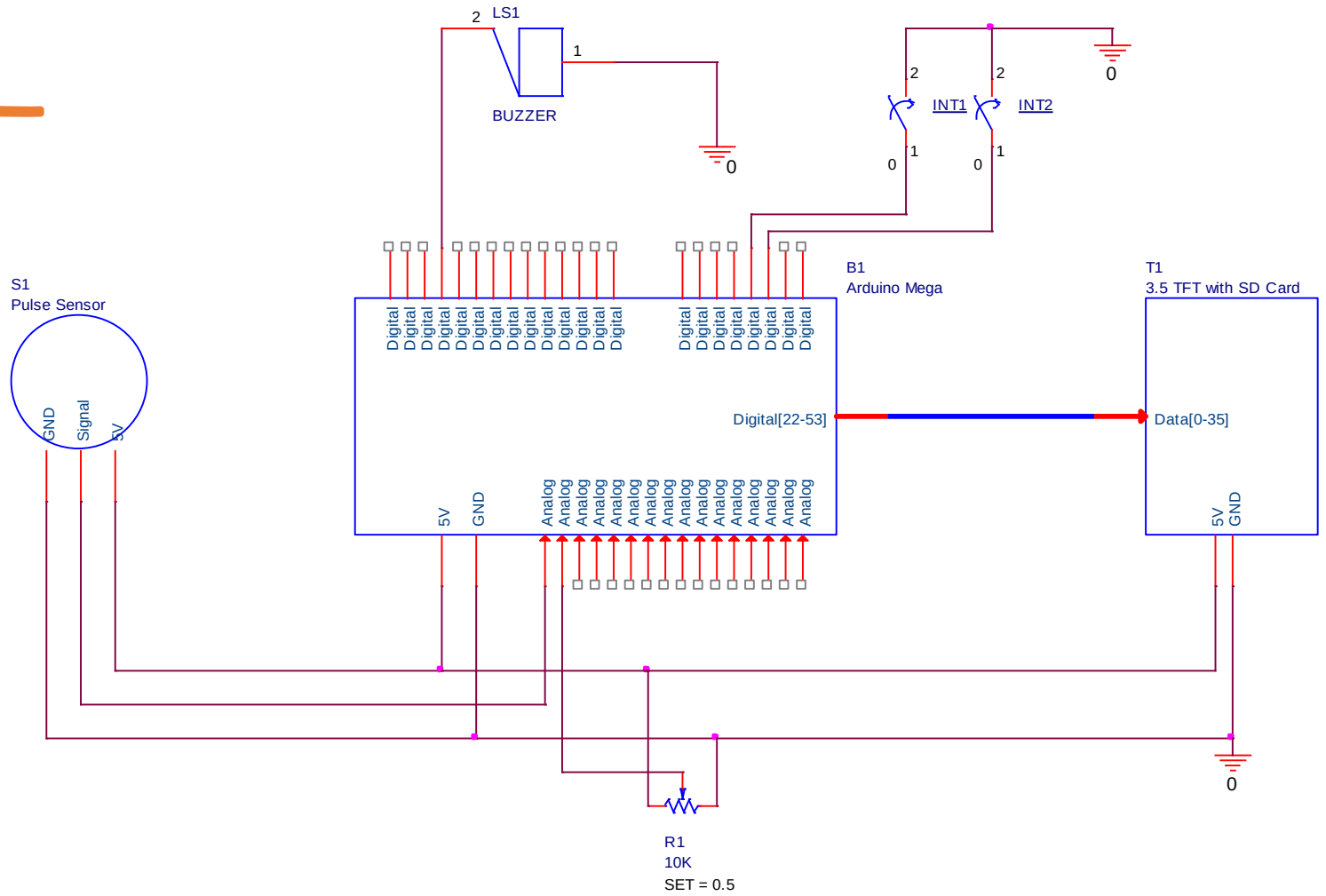
Breadboard

Wires

Resistors

Buzzer

Wiring the Circuit



Code

```
#define USE_ARDUINO_INTERRUPTS false
#include <PulseSensorPlayground.h>

const int OUTPUT_TYPE = SERIAL_PLOTTER;

/*
  Pinout:
    PULSE_INPUT = Analog Input. Connected to the pulse sensor
     purple (signal) wire.
    PULSE_BLINK = digital Output. Connected to an LED (and 1K series resistor)
     that will flash on each detected pulse.
    PULSE_FADE = digital Output. PWM pin onnected to an LED (and 1K series resistor)
     that will smoothly fade with each pulse.
     NOTE: PULSE_FADE must be a pin that supports PWM. Do not use
     pin 9 or 10, because those pins' PWM interferes with the sample timer.
    THRESHOLD should be set higher than the PulseSensor signal idles
     at when there is nothing touching it. The expected idle value
     should be 512, which is 1/2 of the ADC range. To check the idle value
     open a serial monitor and make note of the PulseSensor signal values
     with nothing touching the sensor. THRESHOLD should be a value higher
     than the range of idle noise by 25 to 50 or so. When the library
     is finding heartbeats, the value is adjusted based on the pulse signal
     waveform. THRESHOLD sets the default when there is no pulse present.
     Adjust as neccesary.
*/
const int PULSE_INPUT = A0;
const int PULSE_BLINK = LED_BUILTIN;
const int PULSE_FADE = 5;
const int THRESHOLD = 550;   // Adjust this number to avoid noise when idle

/*
  samplesUntilReport = the number of samples remaining to read
  until we want to report a sample over the serial connection.

  We want to report a sample value over the serial port
  only once every 20 milliseconds (10 samples) to avoid
  doing Serial output faster than the Arduino can send.
*/
byte samplesUntilReport;
const byte SAMPLES_PER_SERIAL_SAMPLE = 10;

/*
  All the PulseSensor Playground functions.
*/

PulseSensorPlayground pulseSensor;
```

Code

```
#include <TFT_HX8357.h> // Hardware-specific library

// Invoke library
TFT_HX8357 tft = TFT_HX8357();

int calc1, calc2, value;

#define BLACK   TFT_BLACK
#define WHITE   TFT_WHITE
#define RED     TFT_RED
#define YELLOW  TFT_YELLOW
#define GREEN   TFT_GREEN  //redefinitions if using different displays

int HSIZE; //horizontal and vertical size of display
int VSIZE;

int color;
int bgcolor;
volatile int BPM;

volatile int rotation;
volatile int oldRotation;
volatile bool mode;
volatile int size;
volatile bool takingMeasurement;
volatile int x, sX; //volatile since these will be updated in ISR
```

Code

```
int y, sY; //for drawing waveform

int speed = 2;  //value for clearing lines(larger takes longer, so
                //larger is "slower", lower than 5 is most accurate)
String test = "BPM:";

#define pot A1
#define buzzerPin 3

int sample;

int results[5] = {};

unsigned int count = 0;
unsigned int averageBPM, currentBPM;
unsigned long elapsedTime;
unsigned long startTime;
bool start = false; //more global variables

// int hbY[6] = {160, 130, 240, 80, 180, 160};

volatile unsigned long last_interrupt_time1 = 0;  //volatile since will
volatile unsigned long last_interrupt_time2 = 0;  //be updated in ISR

#include <SD.h>

File storage; //setting up SD card file
```

Code

```
void setup() {
  // Configure the PulseSensor manager.
  pulseSensor.analogInput(PULSE_INPUT);
  pulseSensor.blinkOnPulse(PULSE_BLINK);
  pulseSensor.fadeOnPulse(PULSE_FADE);

  pulseSensor.setSerial(Serial);
  pulseSensor.setOutputType(OUTPUT_TYPE);
  pulseSensor.setThreshold(THRESHOLD);

  // Skip the first SAMPLES_PER_SERIAL_SAMPLE in the loop().
  samplesUntilReport = SAMPLES_PER_SERIAL_SAMPLE;

  // Now that everything is ready, start reading the PulseSensor signal.
  if (!pulseSensor.begin()) {
    /*
       PulseSensor initialization failed,
       likely because our Arduino platform interrupts
       aren't supported yet.

       If your Sketch hangs here, try changing USE_PS_INTERRUPT to false.
    */
    for(;;) {
      // Flash the led to show things didn't work.
      digitalWrite(PULSE_BLINK, LOW);
      delay(50);
      digitalWrite(PULSE_BLINK, HIGH);
      delay(50);
    }
  }
```

Code

```
  SD.begin(53); //SD card CS is on pin 53
  storage = SD.open("storage.txt", FILE_READ); //opens file with previous data in read mode
  for(int i = 0; i < 5; i++) {
    results[i] = storage.readStringUntil('\n').toInt(); //writes contents of file to array
  }
  storage.close();
  pinMode(18, INPUT_PULLUP);                                //button is wired to ground and to pin
  attachInterrupt(digitalPinToInterrupt(18), ISR1, FALLING);  //no need for physical pull up resistor
  pinMode(19, INPUT_PULLUP);                                //since pin is normally logic high
  attachInterrupt(digitalPinToInterrupt(19), ISR2, FALLING);  //interrupt will trigger on falling edge
  rotation = 3;
  mode = 0;
  tft.begin(); //enables writing to display
  tft.setRotation(rotation); //sets rotation to flipped portrait
  HSIZE = tft.width() - 1;
  VSIZE = tft.height() - 1;
  color = GREEN;        //sets main color to green
  bgcolor = BLACK;      //bg color to black
  size = 0;             //forces updateSize to happen
  mode = true;          //starts off in main mode
  updateSize();
  tft.fillScreen(bgcolor);          //fills with black
  tft.setTextColor(color, bgcolor); //text color white
  tft.setTextSize(size);
  tft.setCursor(0,0); //start text at top of screen
  tft.print(test);    //writes "BPM:"
}
```

Code

```
void loop() {

  if(mode) { //checks mode
    mainMode();
  }
  else {
    altMode();
  }
}

void checkHeartbeat() {
  if (pulseSensor.sawNewSample()) {
  /*
    Every so often, send the latest Sample.
    We don't print every sample, because our baud rate
    won't support that much I/O.
  */
    drawTFT(); //draws heartbeat signal from sensor
    if (--samplesUntilReport == (byte) 0) {
      samplesUntilReport = SAMPLES_PER_SERIAL_SAMPLE; //the library polls the sensor less frequently

      //pulseSensor.outputSample();

      /*
        At about the beginning of every heartbeat,
        report the heart rate and inter-beat-interval.
      */
      if (pulseSensor.sawStartOfBeat()) {
        count++; //used for second mode (just here to group code together)
        tone(buzzerPin, 820); //buzzes buzzer at 820Hz
        //drawHeartbeat(); //self-explanatory
        BPM = pulseSensor.getBeatsPerMinute();
        updateBPM(); //self explanatory
        updateColor(); //self explanatory
        noTone(buzzerPin); //turns off buzzer (above calculations provide delay)
      }
    }
  }
}

void mainMode() {
  checkHeartbeat(); //main drawing function
}
```

Code

```
void altMode() { //for taking measurements that will be saved (30s)
  if (takingMeasurement) {
    count = 0; //resets counts of beats
    tft.fillScreen(BLACK);
    refreshDisplay(); //clears display
    while(!start)
    {
      if (pulseSensor.sawNewSample()) {
        if (--samplesUntilReport == (byte) 0) {
            samplesUntilReport = SAMPLES_PER_SERIAL_SAMPLE;
            if (pulseSensor.sawStartOfBeat()) { //checks for start of beat, then starts measurement
              start = true;
            }
        }
      }
    }
    refreshDisplay(); //to make sure
    size = 10;
    updateSize(); //force update size and display
    startTime = millis();
    while (elapsedTime <= 30000) {
      checkHeartbeat(); //uses main function to display heartbeat and calculate BPM
      elapsedTime = millis() - startTime;
    }
    noTone(buzzerPin);          //turns off buzzer just in case
    tft.fillScreen(BLACK);      //clears screen
    currentBPM = BPM;           //gets last bpm recorded
    averageBPM = 2*count;       //since 30 second interval, times 2 gets average BPM
    tft.setCursor(0,0);         //print results
    tft.println("Results:");
    tft.println("Last = ");
    tft.println(currentBPM);
    tft.println("Average = ");
    tft.println(averageBPM);
    count = 0;                  //resets beat count
    for (int i = 4; i > 0; i--) {
    results[i] = results[i - 1];  //moves down one, discarding oldest
    }
    results[0] = currentBPM;      //and writing newest to top
    storage = SD.open("storage.txt", FILE_WRITE | O_TRUNC); //write array of 5
    for(int i = 0; i < 5; i++) {                            //results to SD card
      storage.println(results[i]);                          //always starts at top
    }
    storage.close(); //close file
    delay(2000); //stay on results display for 2 sec
    tft.fillScreen(BLACK);
    size = 10; //force update
    elapsedTime = 0;
    start = false; //clears start of measurement "flag"
    takingMeasurement = false; //clears measurement "flag"
  }
  else {
    updateSize(); //checks pot for changes
                  //just displays last 5 results otherwise
  }
}
```

Code

```
void drawTFT() {
  sample = pulseSensor.getLatestSample(); //self explanatory
  sX = x; //sets first point of new line
  sY = y;
  x++; //increases x coordinate
  value = map(sample, 0, 1023, calc2, 0); //set output of sensor to drawing limits of tft
  y = value + calc1; //account for the fact that coordinates go top to bottom instead of bottom top
  tft.drawLine(sX, sY, x, y, color); //draws line from previous value to new value
  for(int i = 1; i <= speed; i++) {
    tft.drawFastVLine((x+i), calc1, calc2 + 1, bgcolor); //clears lines ahead
  }
  if(x >= HSIZE) { //checks if the wave has reached the end of the screen
    updateSize(); //checks size
    tft.drawFastVLine(0, calc1, calc2 + 1, bgcolor);  //clears first two vertical lines
    tft.drawFastVLine(1, calc1, calc2 + 1, bgcolor);  //when heartrate reaches end of screen
    x=0;
  }
}
```

Code

```
void updateBPM() {
  tft.setCursor(test.length()*(5+1)*size, 0); //moves cursor past "BPM:" to
  tft.print(BPM);                             //avoid overwriting it
  tft.print("  ");                            //clears third digit if value before had three digits
}

void updateColor() {
  int newColor;
  if (BPM >= 120) {                 //self-explanatory color setting
    newColor = RED;
  }
  if ((BPM < 120) && (BPM >= 90)) {
    newColor = YELLOW;
  }
  if (BPM < 90) {
    newColor = GREEN;
  }
  if(newColor != color) { //set color if new and reprint BPM in new color
    color = newColor;
    tft.setTextColor(color, bgcolor);
    tft.setTextSize(size);
    tft.setCursor(0,0);
    tft.print(test); //prints "BPM:"
    updateBPM();
  }

}
```

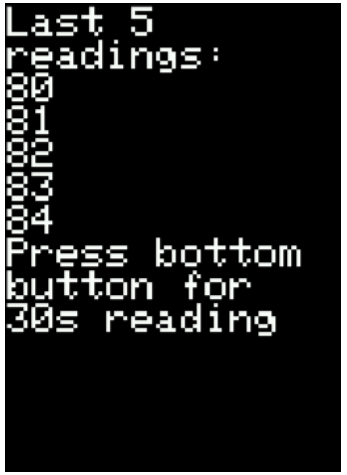
Code

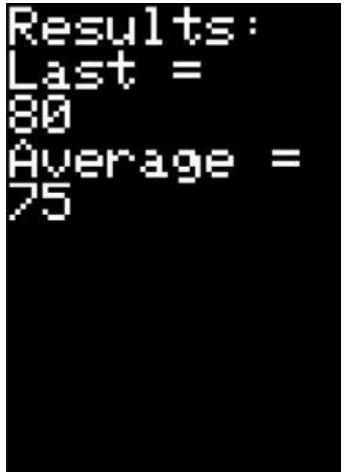
```
void updateSize() {
  int newSize = map(analogRead(pot), 10, 1024, 1, 5) + 1; //reads pot, maps it from sizes 2 to 5
  if (size != newSize) { //update if the size is new
    size = newSize;
    tft.setTextSize(size);
    tft.setCursor(0,0);
    if(mode || takingMeasurement) { //checks if it is in main mode or currently taking measurement
      tft.print(test + "       "); //extra spaces are to clear space after if going to smaller size
      color = BLACK;
      updateColor();
      calc1 = (size * 7); //calculates upper limit of drawable screen
      calc2 = (VSIZE - calc1); //lower limit
    }
    else {                      //otherwise it is currently displaying previous measurements
      tft.fillScreen(BLACK);
      tft.setTextColor(WHITE);
      tft.println("Last 5");
      tft.println("readings:");
      for(int i = 0; i < 5; i++) {
        tft.println(results[i]);
      }
      tft.println("Press bottom");
      tft.println("button for");
      tft.println("30s reading");

    }
  }

}

void updateRotation() {
  tft.setRotation(rotation);
  VSIZE = tft.height() - 1; // -1 to account for top left being (0,0)
  HSIZE = tft.width() - 1;
  x = 0;
  sX = 0;
  tft.fillScreen(BLACK); //clear screen
  refreshDisplay();
}

void refreshDisplay() {
  tft.setCursor(0,0);
  tft.print(test + "       ");  //extra spaces are to clear space
  updateColor();                //after if going to smaller size
  updateBPM();
  calc1 = (size * 7); //calculates upper limit of drawable screen
                      //(7 is how tall a character of size 1 is)
  calc2 = (VSIZE - calc1); //lower limit
}
```

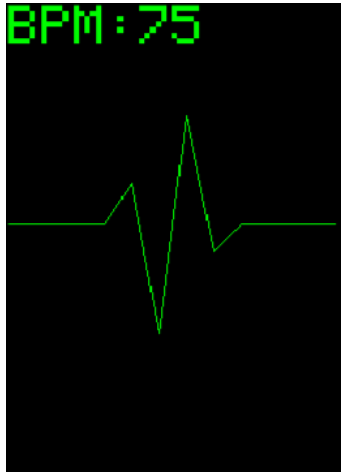
Code

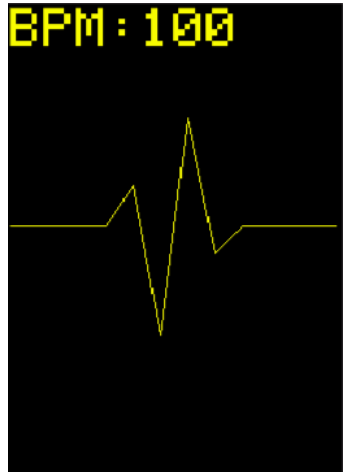
```
void ISR1() {
  unsigned long interrupt_time1 = millis();
  // If interrupts come faster than 200ms, assume it's a bounce and ignore
  if (interrupt_time1 - last_interrupt_time1 > 200)
  {
    if(mode || takingMeasurement) { //will rotate screen in main and measurement mode
      oldRotation = rotation; //keep old rotation to compare in other functions
      rotation++; //increment rotation
      if (rotation >= 4) //3 is max, wrap around
      {
        rotation = 0;
      }
      updateRotation();
    }
    else {
      if(!takingMeasurement && !mode) { //checks if in alternate mode, but in results screen
        BPM = 0;
        takingMeasurement = true; //turns on measurement mode
      }
    }
  }
  last_interrupt_time1 = interrupt_time1;
}

void ISR2() {
  unsigned long interrupt_time2 = millis();
  // If interrupts come faster than 200ms, assume it's a bounce and ignore
  if (interrupt_time2 - last_interrupt_time2 > 200)
  {
    if(!takingMeasurement) { //if taking measurement, button does nothing
      size = 0;
      noTone(buzzerPin); //turns off buzzer just in case
      mode = !mode; //switches mode
      if(mode)
      {
        x = 0;
        sX = 0; //resets cursor
        tft.fillScreen(BLACK); //resets display
      }
      else {
        tft.fillScreen(BLACK); //resets display
      }
    }
  }
  last_interrupt_time2 = interrupt_time2;
}
```

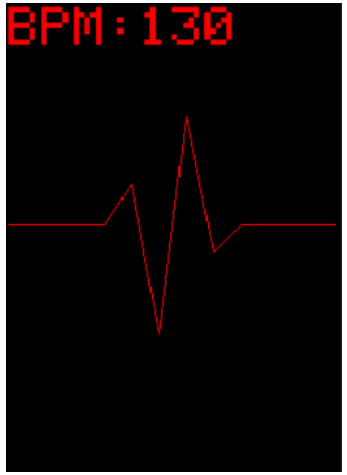
User Interface





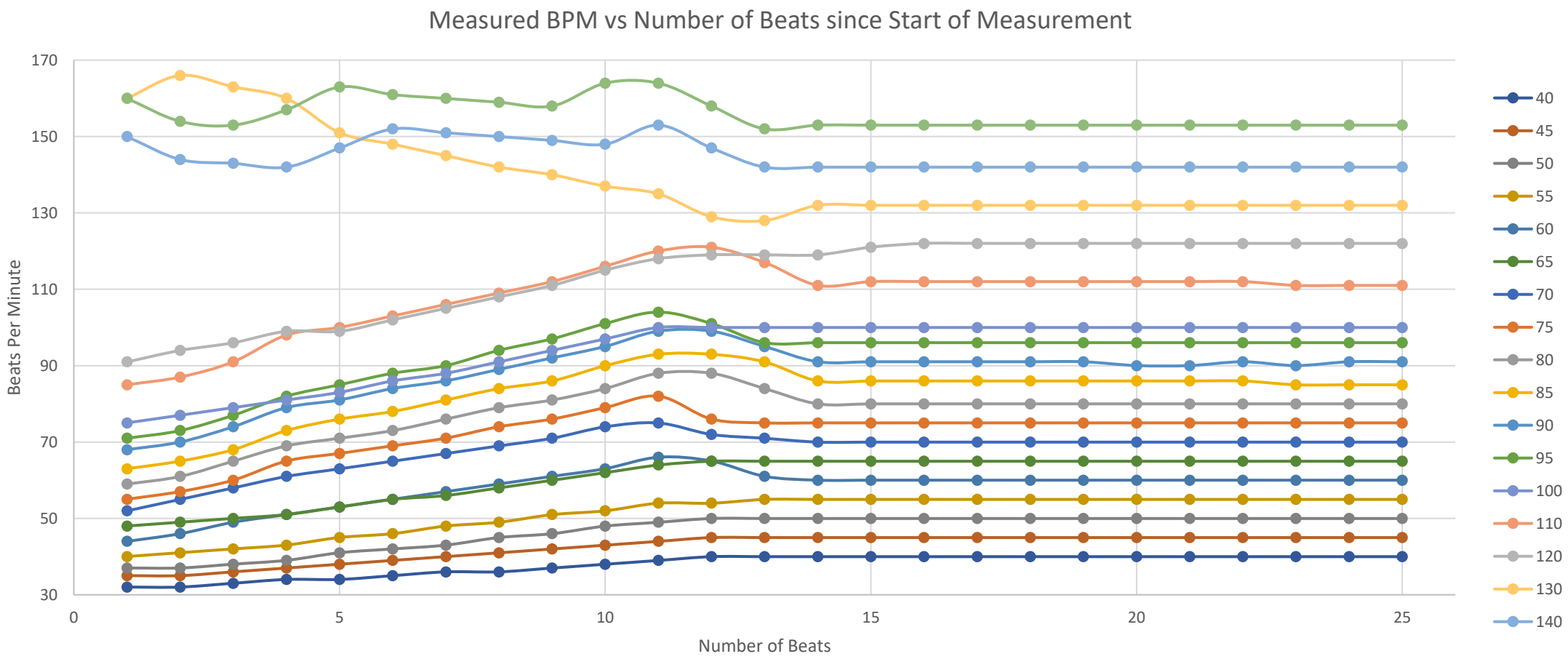




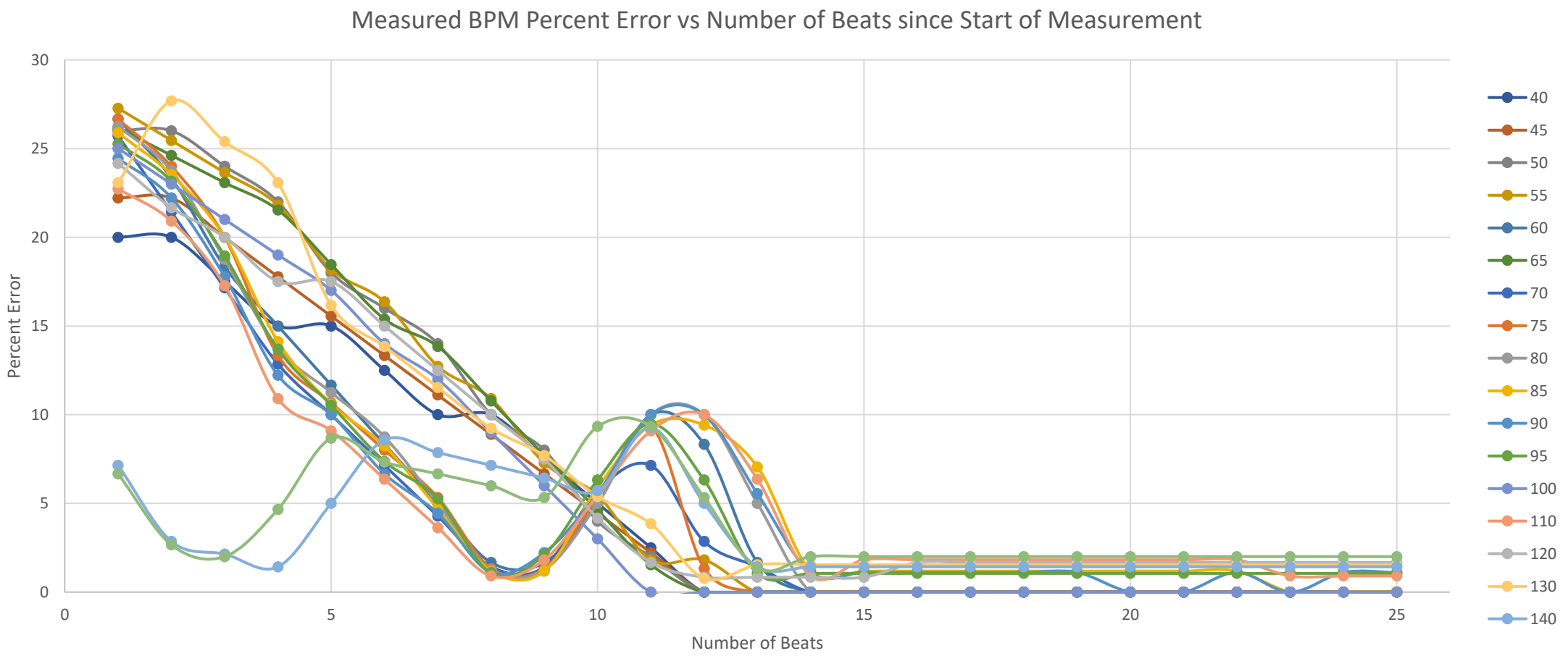


Experimental Results

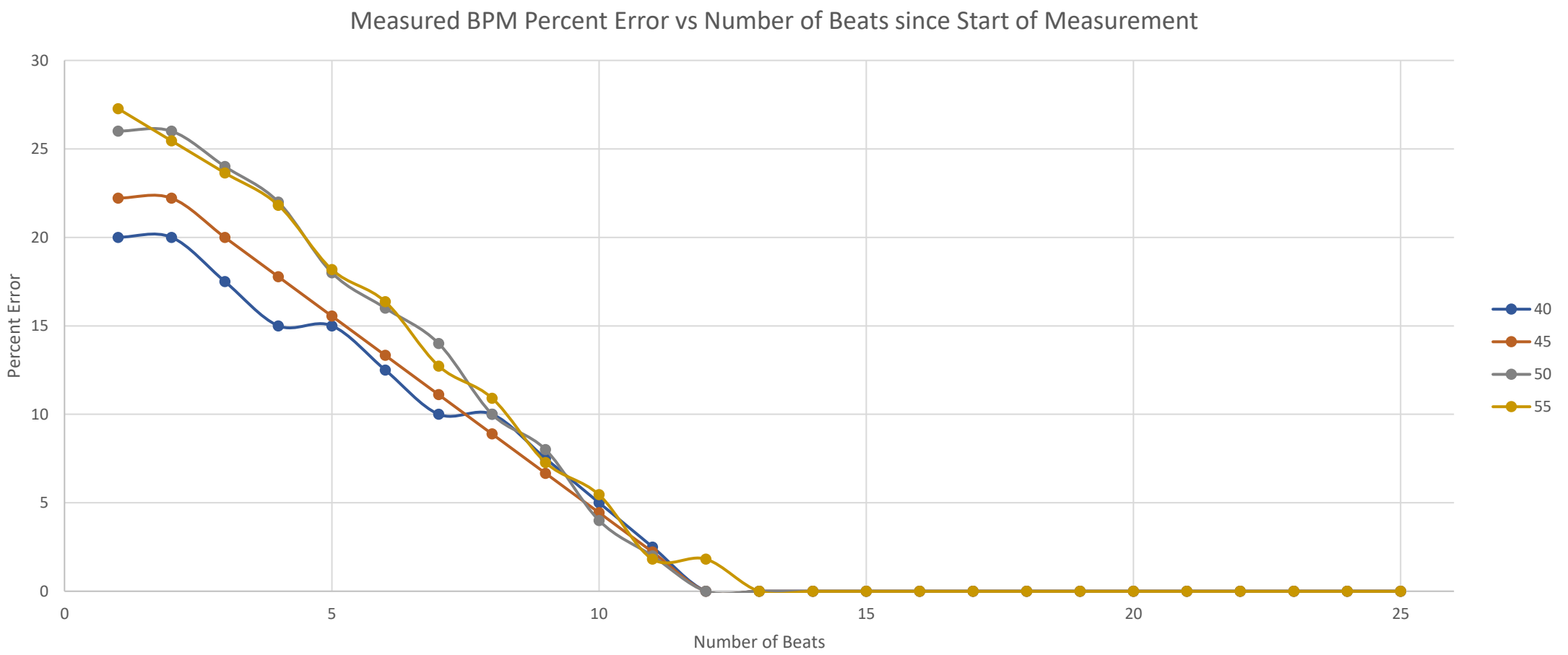
Reference Waveform



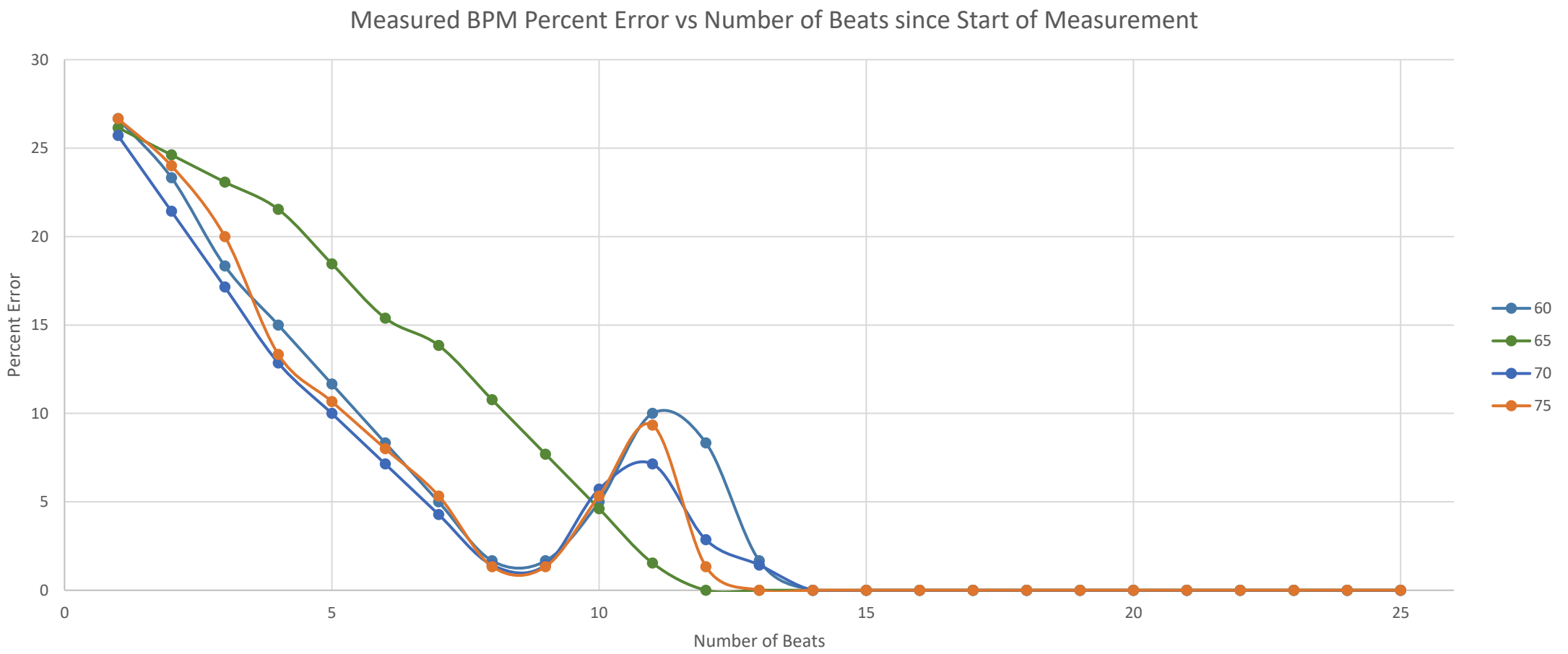
Reference Waveform



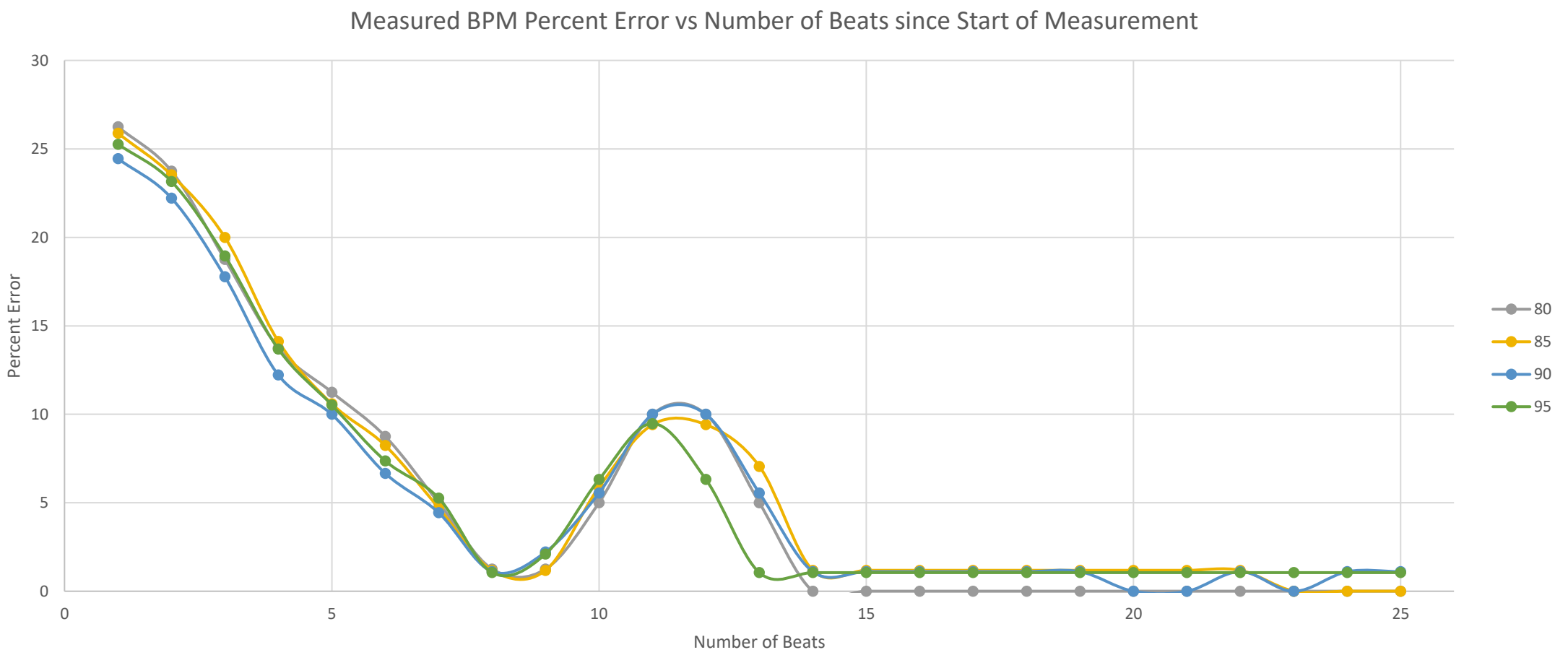
Reference Waveform



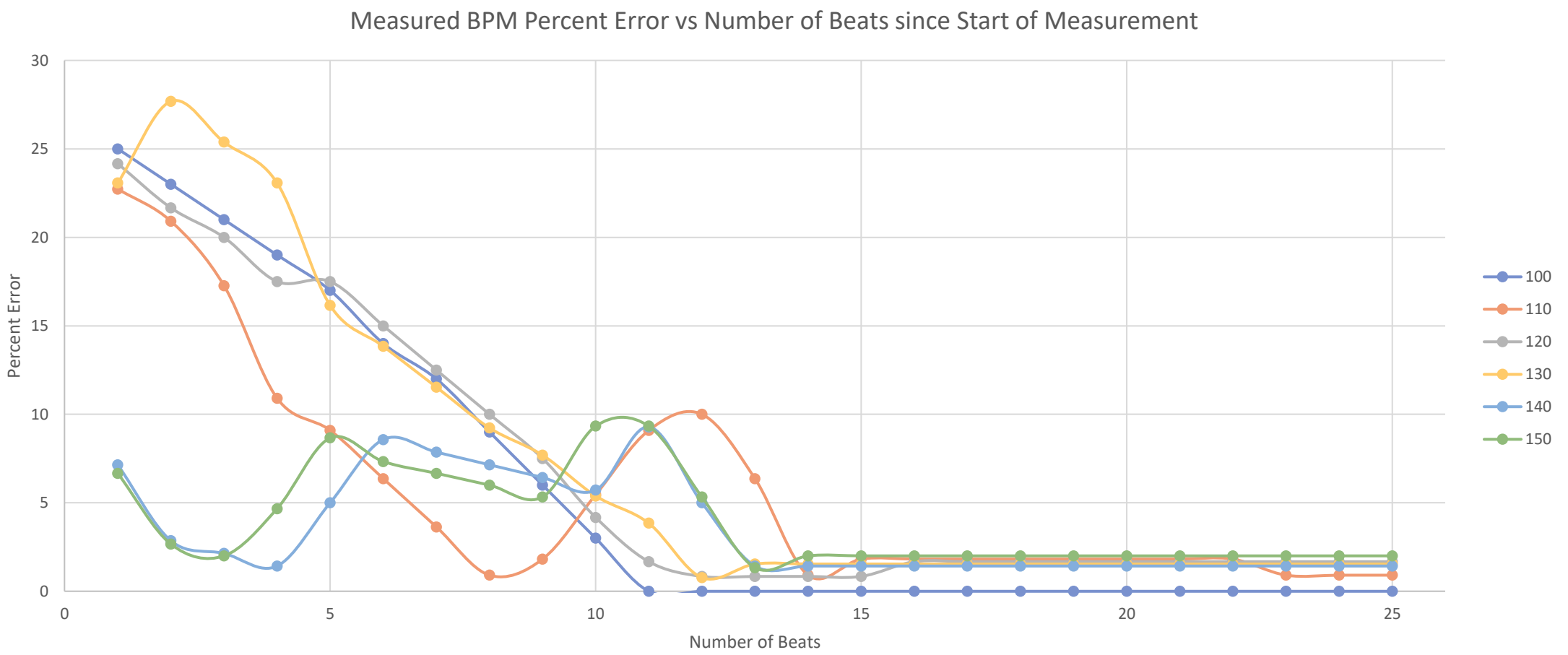
Reference Waveform



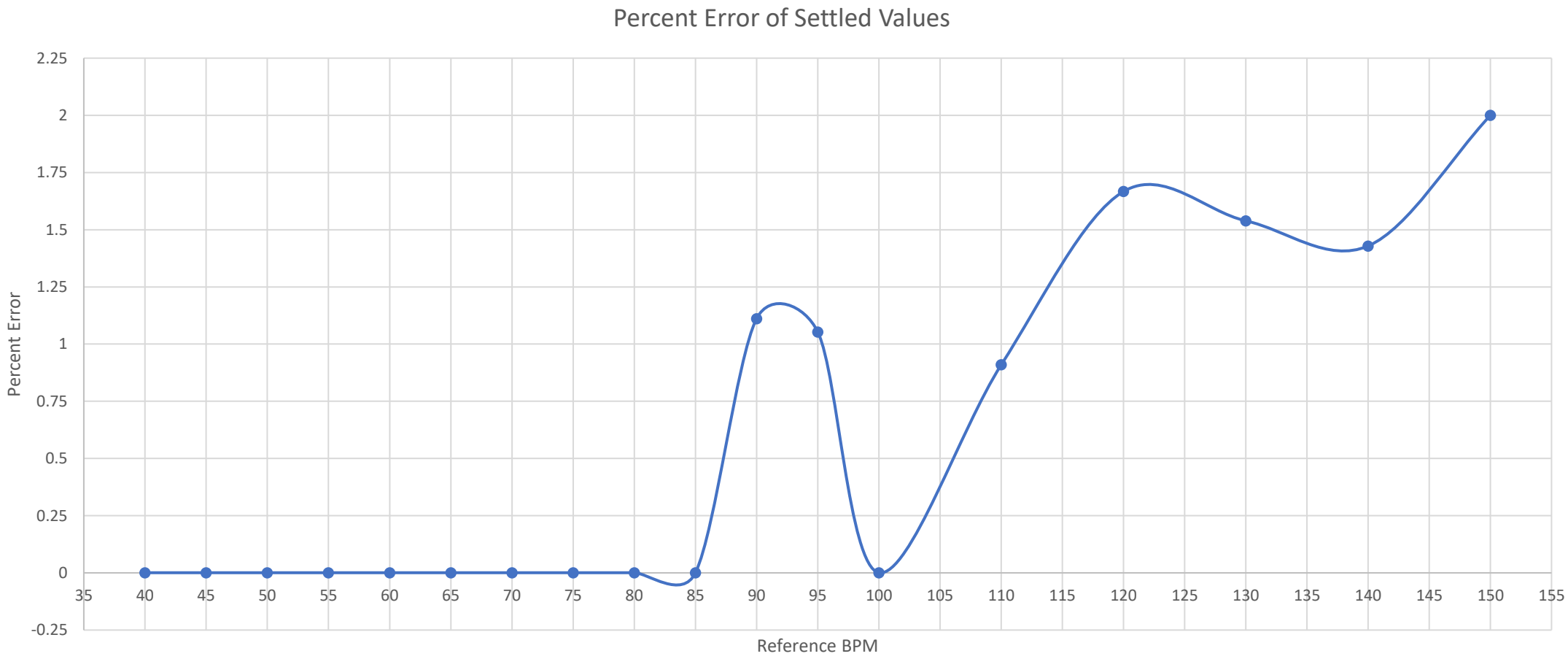
Reference Waveform



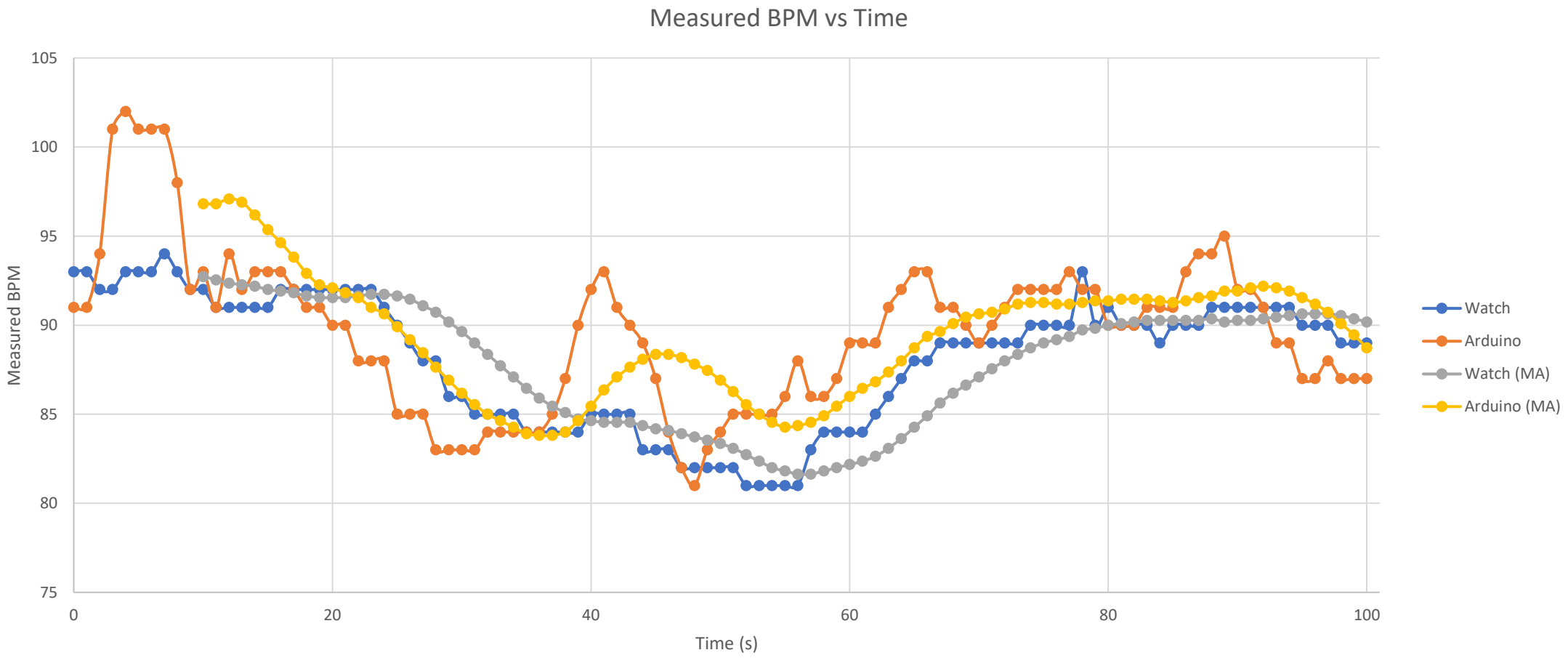
Reference Waveform



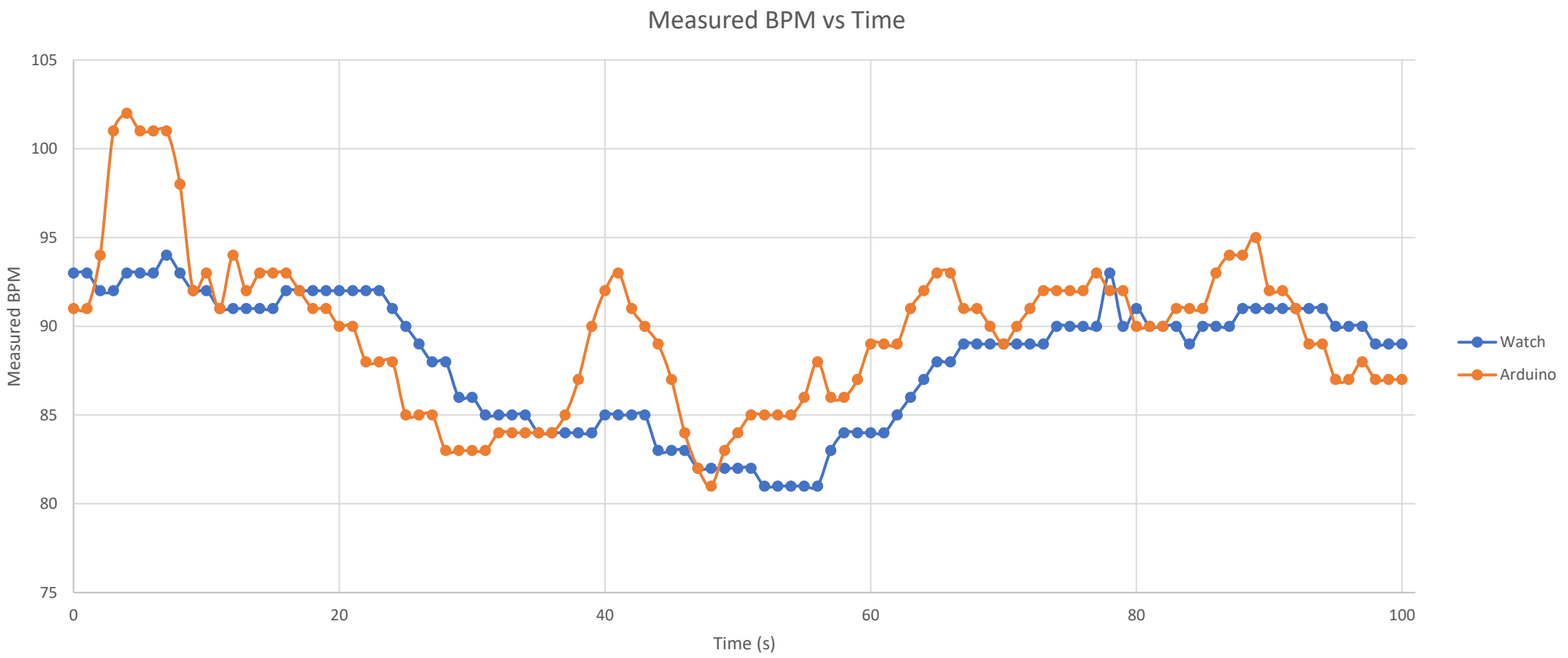
Reference Waveform



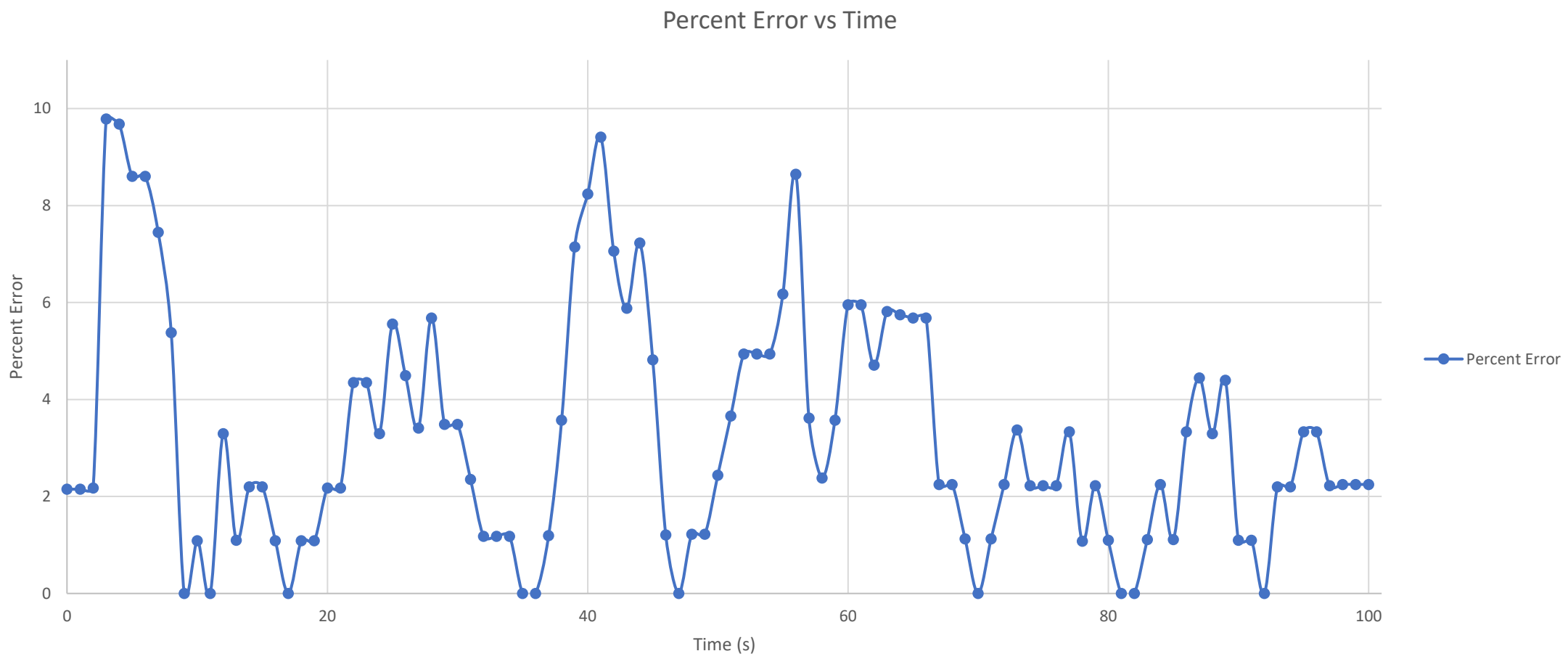
Heartbeat (Resting)



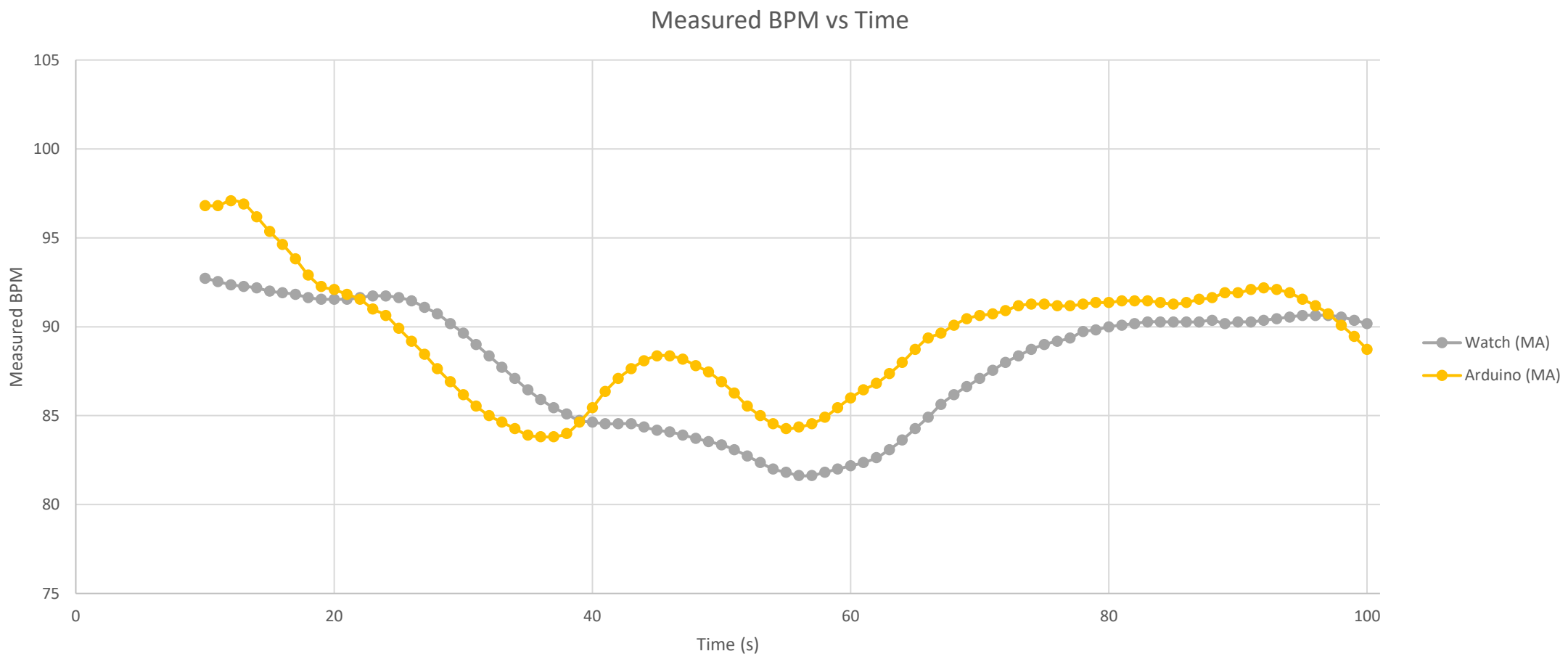
Heartbeat (Resting)



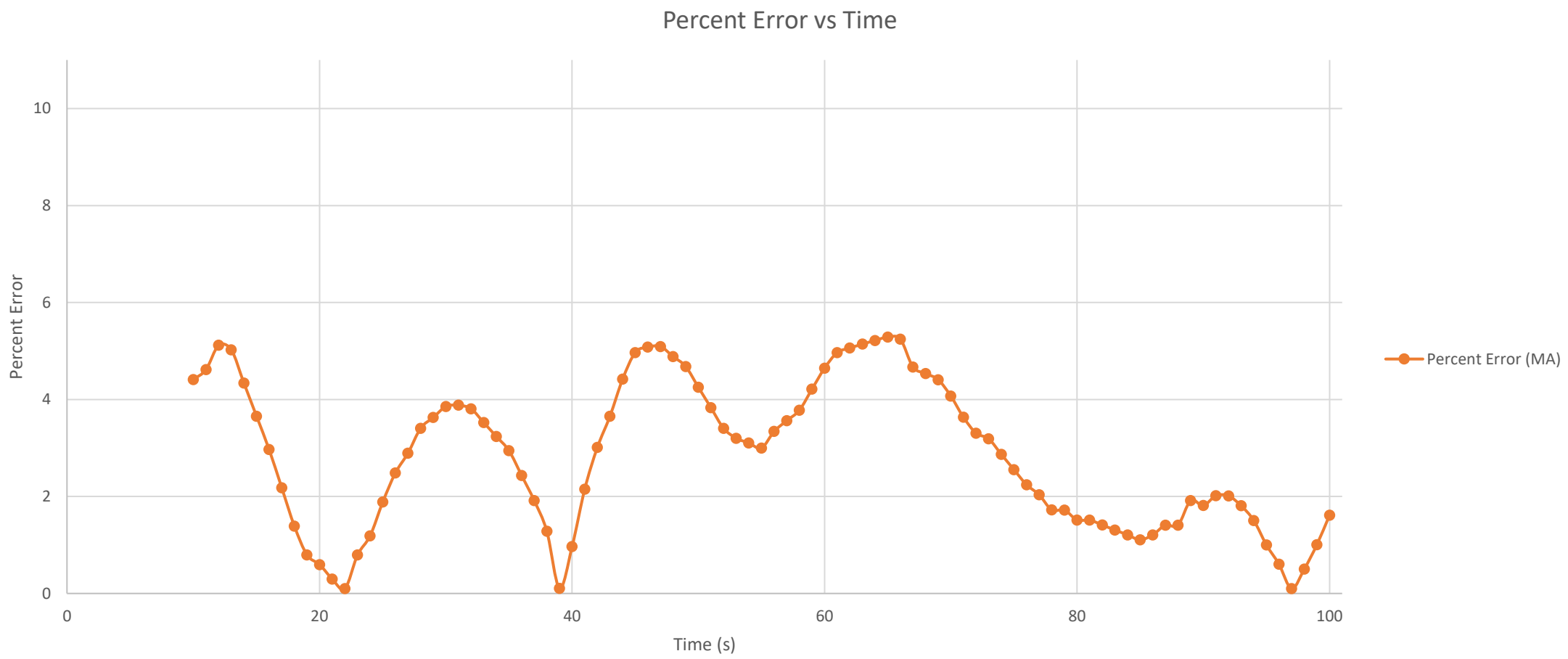
Heartbeat (Resting)



Heartbeat (Resting) (Moving Average)



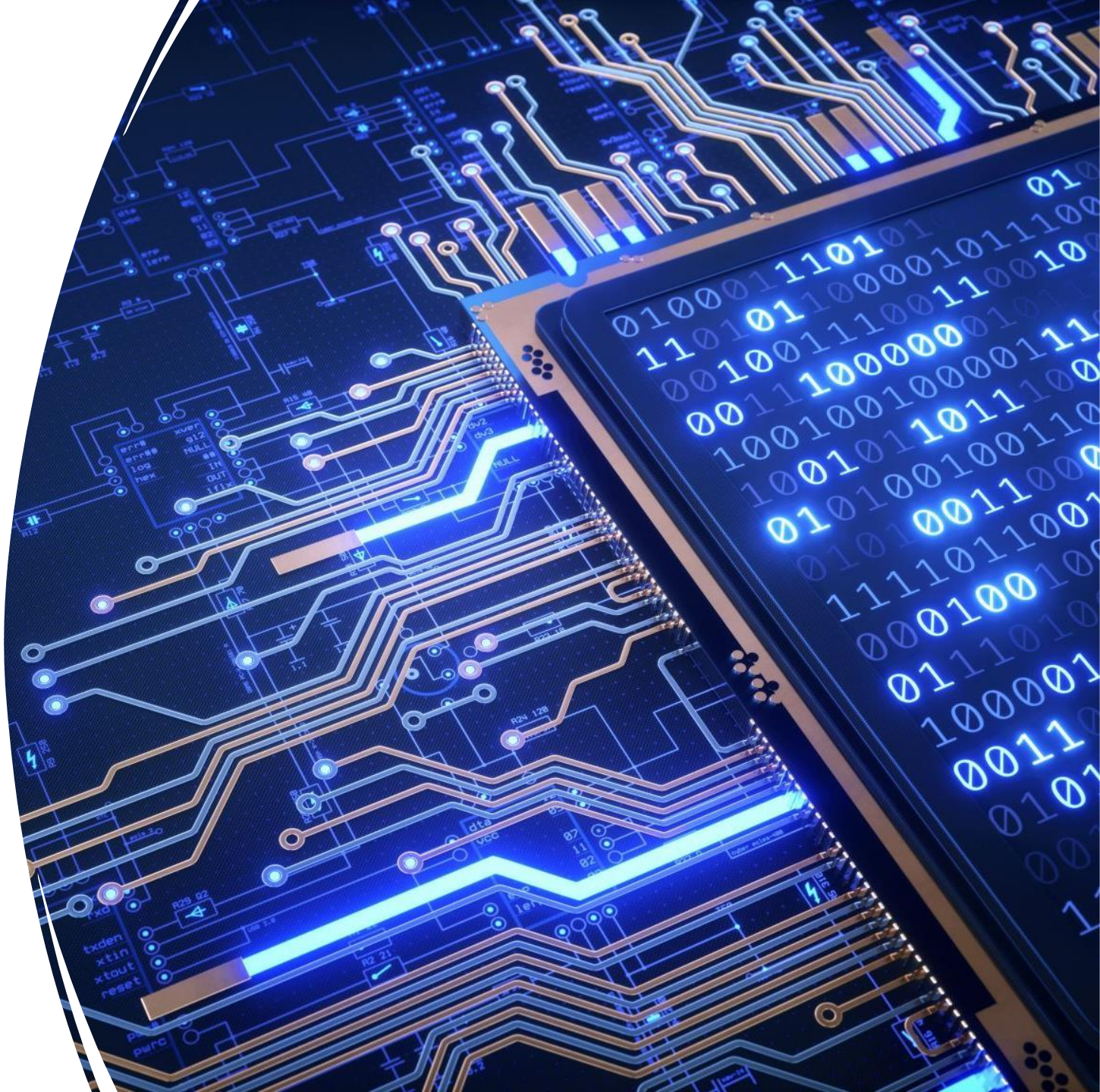
Heartbeat (Resting) (Moving Average)

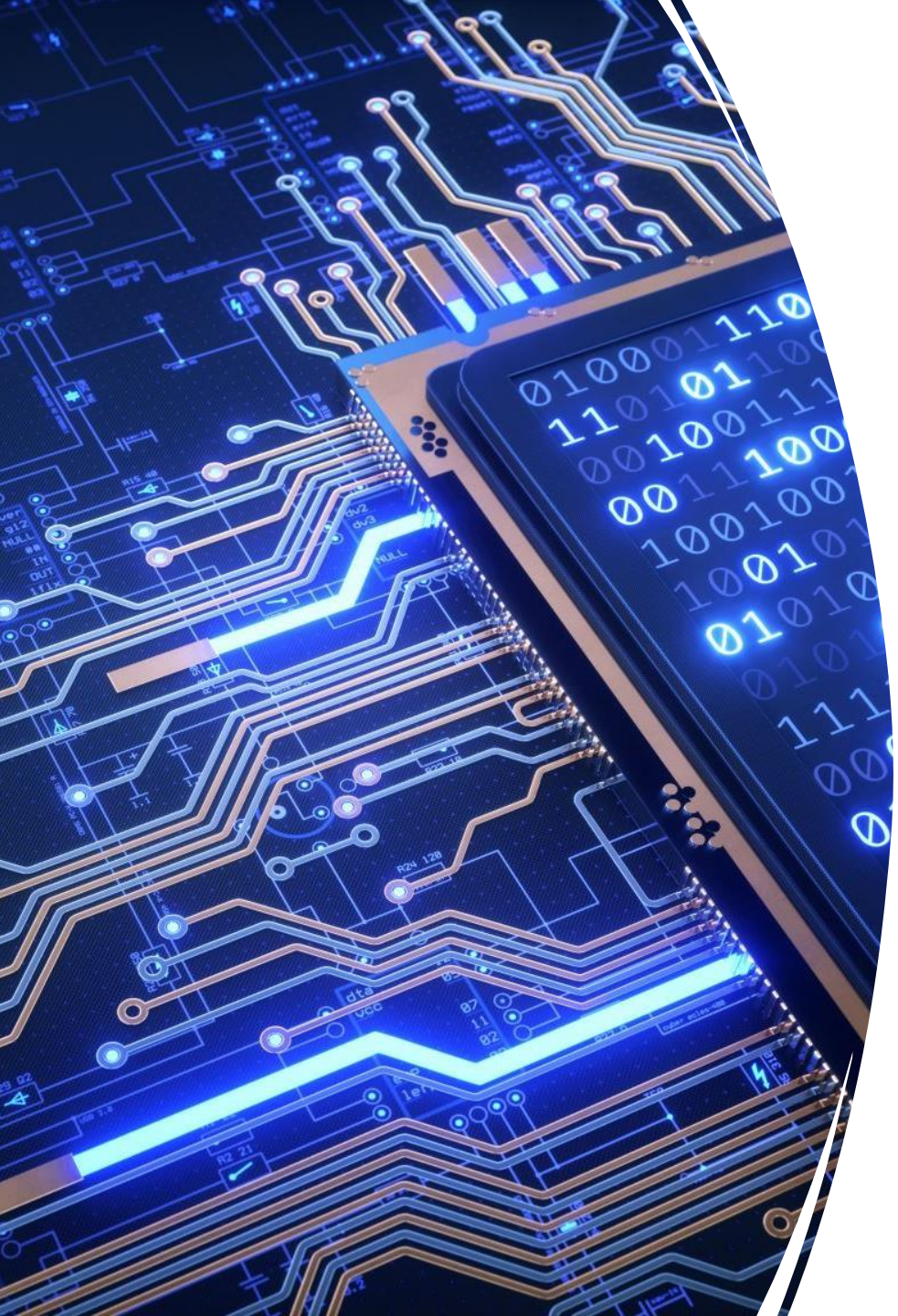


Challenges and Resolutions

Challenge

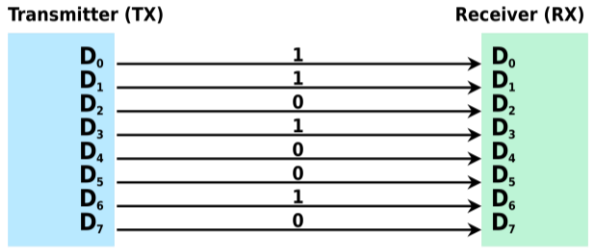
- LCD Screen
- First was just a 16x2 dot matrix display without soldered headers
- Moved to LCD
- Not updating fast enough
- Using different types of displays
- Started with serial display using SPI
- Misconfigured pins at first, using bit-banged SPI, fixed to use HW SPI
- Still too slow to update, messed with calculations

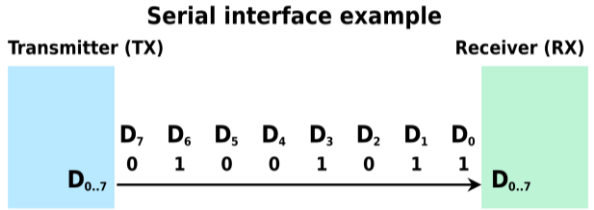


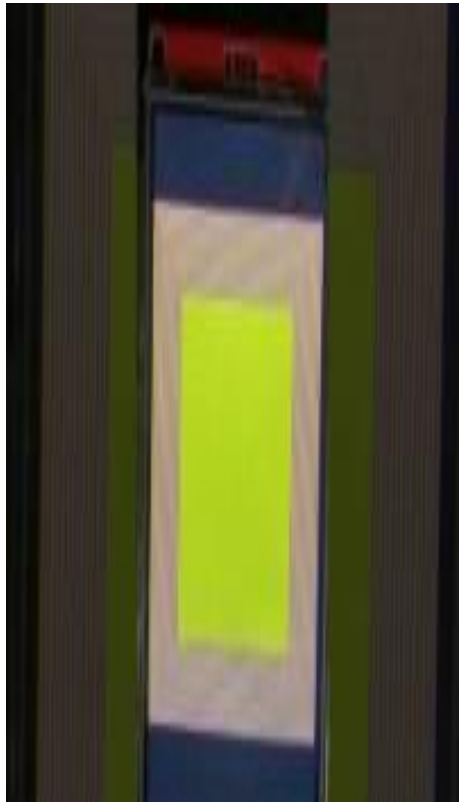


Resolution

- Using a parallel interface display
- Much faster refresh rate
- Less time sending signals to refresh display
- More time to do ADC conversion and do calculations like BPM







Example

Challenge

- Pulse Rate Calculation
- Was mostly caused by the previous problem
- When we got the HW SPI to work, still had wrong BPM
- There was error
- Tried to correct it, but error was nonlinear

Example

Challenge

- Wave Form clipping
- Waveform from sensor always clipping
- Does not appear on oscilloscope
- Perhaps due to board using supply power of USB as reference
- When measuring voltage of USB power, appears to be bit less than 5V
- The measured waveform on oscilloscope has peaks up to 5V
- Does not appear to mess with accuracy however

Benefits

- Measuring heartbeat activity
- The heart rate monitoring system can be a useful tool for educational purposes. It can be used to teach students about the cardiovascular system, heart rate monitoring, and the principles of electronics.
- The Arduino platform allows for customization of the heart rate monitoring system. Users can modify the code to include additional features or to adjust the display to suit their needs.