ELEC3607/9607 Assignment Final Report

Bluetooth Pulse Meter

Group9		
Member	**Unikey**	**Student ID**
Liu, Yahong	yliu6900	440358628
Yan, Xiangyu	xyan5959	450069156

Abstract: In this paper, we described the designing of Bluetooth pulse meter based on Arduino Due. We present the theory and algorithm of calculating BPM (Beats per Minute) values. We also described the function of Android app which could communicate with the Arduino board. In the later parts, we depict the testing method and the problems we encountered during the project. The circuit schematic and main code is also attached in the appendix.

Keywords: Arduino Due; Pulse Meter; Bluetooth; Android;

Content

1 Introduction

In modern society, with the continuous improvement of people's living standard, great changes in dietary structure and lifestyle have taken in people daily life. The sick ratio cardiovascular and cerebrovascular diseases like hypertension, coronary heart disease and arteriosclerosis increase a lot. It is estimated that nearly one in five people have got such problems in cities of China. During the initial stages of these diseases, the patient cannot realize the symptom generally, but actually some parameters of the blood pressure, vascular resistance and blood viscosity have already changed stealthily. And these parameters perform in the changes of pulse firstly. So working through to the pulse of the collection and analysis can be very useful to diagnose the potential risks and to strive for the precious time for the further treatment.

Among all the applications of embedded system in healthcare, pulse meter plays an important role, not only recording the fluctuation of heart rate itself, but also providing simple and real-time signal for advanced application like clinic diagnosis [1], sleep-driving alarming [2] cardiovascular assessment and training assistance [3]. Meanwhile, emerging wireless technology like Bluetooth, and portable devices like smartphone or laptop computer have dramatically expanded the portability and functionality of pulse-rate monitoring devices because raw signal could be interpreted into diagnostic results or advices in real-time wherever the user is.

In addition, the pulse detection can be not only used in the field of medical services, but also in sport field, nowadays many high-tech companies have focused on this area, like FitBit and Jawbone UP. These companies designed some product to help people to take care of the most important organ in body. The devices can analyze the monitoring data and transmit information to the smart phone, which get huge potential benefits.

Figure 1.1 Jawbone UP and corresponding software [4]

In this project, our group focuses on fabricating a simple pulse-meter, with functions including recording the variation of heart-rate in a given period of time, alarming low heart rate variation which implies cardiovascular disorder and pulse-rate-based

sleep-driving alarm. Bluetooth technology will also be employed to connect the device with an Android smartphone, achieving the goal of alarming and data storage.

2 System Components

2.1 Arduino Due

The Arduino Due is based on the ARM Cortex M3 CPU which is used in this project. It runs at 84 MHz, 96Kb Ram and has 512Kb on-board memory. It has 54 digital input/output pins, 12 analog input pins and 4 UART.

In this project, the Arduino due is used to collect the analogue data from pulse sensor and filter it. The analogue data is transformed into digital signals through the ADC (Analog-to-Digital Converter) function. After collecting the signal from sensor, the board will calculate the IBI (Inter Beat Interval) and BPM (beats per minute) values and transmit them to the Android app through Bluetooth.

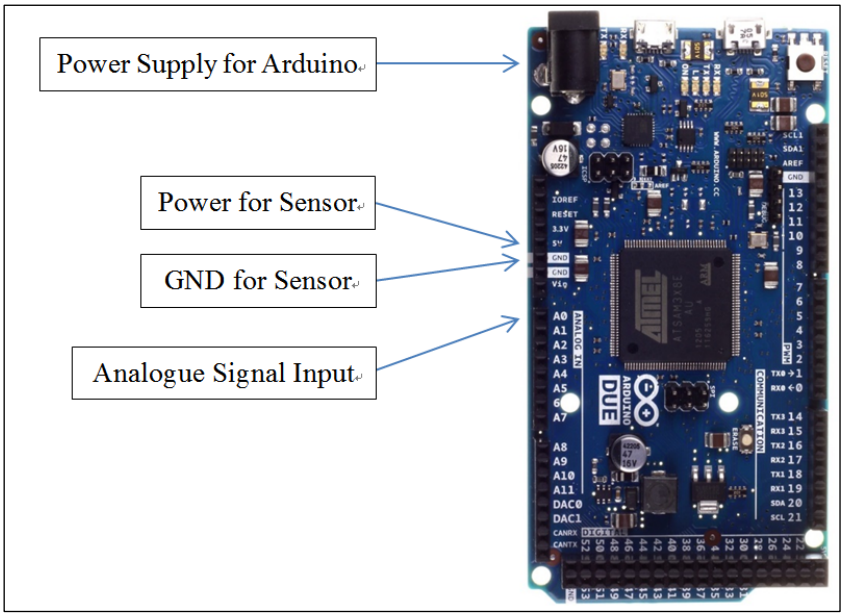


Figure 2.1 Arduino Due and the used pin

2.2 Bluetooth Shield

The Bluetooth Shield [5] used in this project is a great way to detach the Arduino from the computer and smart phone. It can be easily used with Arduino for transparent wireless serial communication. Plug the shield onto the Arduino board and the board would transmit data through the Bluetooth module. In this project, the Bluetooth Shield is used to receive request from the Android app and send the analyzed heart beat to the smart phone.

The jumper selects digital pin6 and pin7 as the Bluetooth RX and TX pin. These two pins have to be connected with the TX2 and RX2 pins on the Arduino board

respectively.

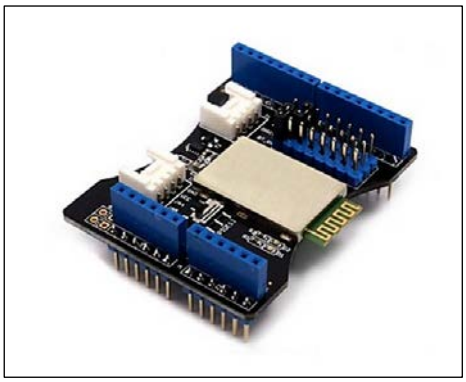


Figure 2.2 Seeedstudio Bluetooth Shield

2.3 Pulse Sensor

Pulse Sensor [6] is an open source heart rate sensor for Arduino. The sensor clips onto a fingertip or earlobe and plugs right into Arduino with some jumper cables. The sensor will transmit the data to Arduino board after detecting signals on body. There are three cables on the sensor to connect to the board:
- The Signal wire connects to Analogue In 0 port on board.
- The Power wire connects to the 3.3V port on board.
- The Ground wire connects to the GND port on board.

These three wires should link to the analog pin, VCC and GND on the Arduino board.

The pulse sensor uses an infrared light source to illuminate the finger on one side of PCB board, and a photo-detector on the other side to measure small variations in the transmitted light intensity due to changes in blood volume inside the tissue. So the pulse sensor is very sensitive to the light, and this is why hook dots is needed. In the practice, the pulse sensor is very tender so that is cannot survive under a high force pressure.

The circuit schematic is as follows.

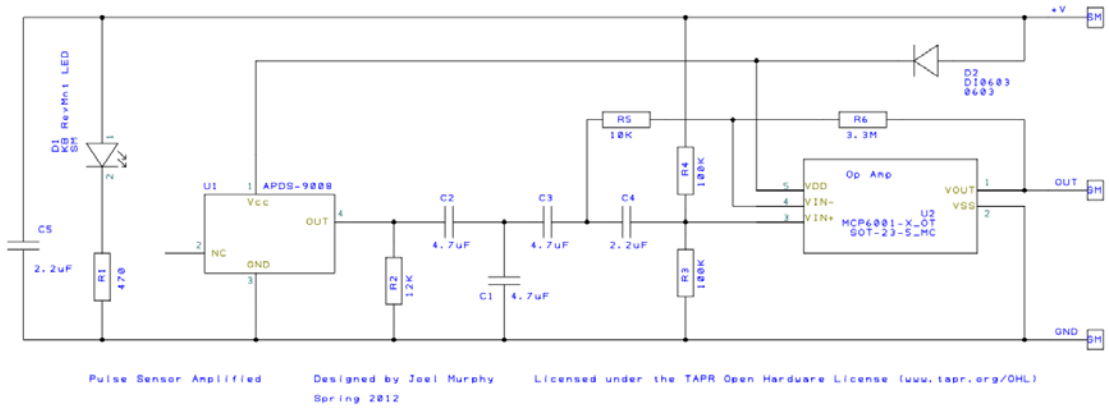


Figure 2.3 Circuit schematic of Pulse Sensor [7]

In this circuit, APDS-9008 is the ambient light sensor and KB RevMnt LED is the light source. Then the circuit uses a filter and amplifier to increase the amplitude of the pulse wave and normalize the signal around a reference point. When we are not using the sensor, the analog signal would hover around the mid-point of the voltage. When the sensor tests the pulse from fingertips, the signal would fluctuate around that reference point.

2.4 Android App

Android is a mobile operating system (OS) based on the Linux kernel and currently developed by Google. With a user interface based on direct manipulation, Android is designed primarily for touchscreen mobile devices such as smartphones and tablet computers, with specialized user interfaces for televisions (Android TV), cars (Android Auto), and wrist watches (Android Wear).

The application will have functions as shown below:
- Communicate with the Arduino board;
- Present the real-time heart rate value (BPM) received from Arduino board;
- Alarm when low heart rate is detected.

3 Designs

The whole system can be separated into two parts. One is the data collection and analysis on the Arduino board, the other is the Android application which could communicate with the Arduino board and display the data.

3.1 Arduino

3.1.1 Theory of Calculating BPM (Beats per Minute)

Blood is a highly opaque liquid and penetration of light is extremely low in blood. Blood volume of peripheral vascular (including capillaries, arterioles, venules) produces a corresponding change due to fluctuation of pulse in the cardiac cycle. Optical volume sensor can detect this change. Using this method to obtain the blood volume is called photoplethysmography. [8] The blood volume detected by photoplethysmography could ensure good stability and repeatability. Since the measurement is at fingertips with no patient activity restrictions, it is suitable for continuous measurement for a long time.

With the Pulse Sensor we are using, light comes from the LED and is reflected back to the sensor which changes during each pulse.

The pulse signal that measures from the sensor is an analog fluctuation in voltage, and it has a waveform as shown below.

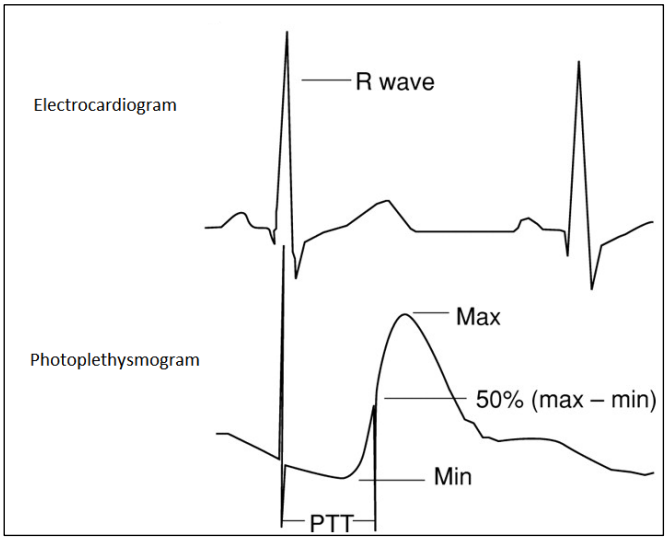


Figure 3.1 Waveform comparisons of electrocardiogram and photoplethysmogram [9]

As is shown above, PTT is called pulse transmit time. It refers to the time it takes a pulse wave to travel between two arterial sites. [9] Generally, the mid-point on the waveform is taken to indicate the arrival of the pulse wave.

We use the mid-point of waveform to represent the heart beat and thus the time between two beats is called the IBI (Inter Beat Interval). The BPM (Beats per Minute) value is derived from an average of 10 IBI times.

3.1.2 Algorithm of BPM Calculation

Firstly, we set up a Timer Counter and call the timer counter interrupt every 2ms to measure the signal of Pulse Sensor. Thus the sampling rate is 500Hz. The analogue signal is transformed into digital signal through the ADC (Analog-to-Digital Converter) function of Arduino Due.

Secondly, since the pulse reflection in the arteries, there would be dicrotic wave as shown below. To avoid this interference, we track the pulse rising up after 0.6 IBI times.

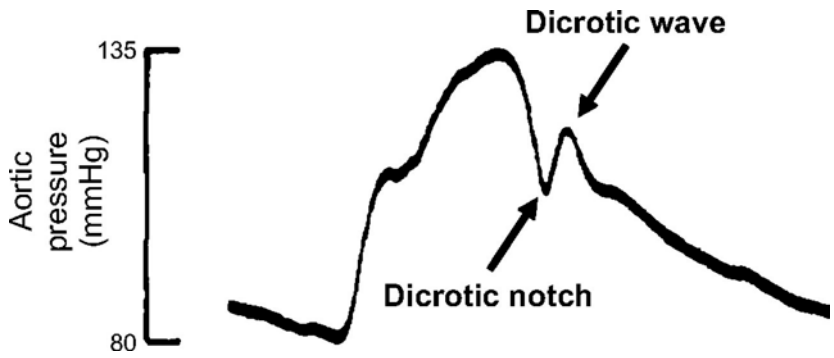


Figure 3.2 the Dicrotic notch and wave in the aortic blood pressure signal [10]

Thirdly, as the theory presented above, we need to calculate BPM through the average of 10 IBI times. When the signal rises to the mid-point of waveform, then we calculate the time since the last beat to get IBI. With IBI value, we use an array to store the previous 10 IBI values so that BPM could be calculated based on these 10 values.

3.1.3 Transmission of BPM Data

Since our system contains the Arduino Due and Android application, we use the Bluetooth module to communicate among the two devices. When the board receives the request sent by the app, the Arduino board would send the BPM value to Android app through Bluetooth serial.

3.2 Android

The Android application is the supporting software for the pulse meter. Commercial wearable devices often have these kinds of support.
We design the Android app with two interfaces. First interface is able to switch the smartphone Bluetooth on and off. We will search the surrounded Bluetooth devices and display them on the interface. When we find the desired device, we could press the link to jump to the second interface.
In the second interface, we could make a request to the Arduino board by pressing the test button. After the Bluetooth transmission, we could see the BPM value and a dynamic circle representing the value. When the BPM value is lower than 30, there would be a ‘low heart rate’ alarm at the top of the interface.
Here are the interfaces of Android application.

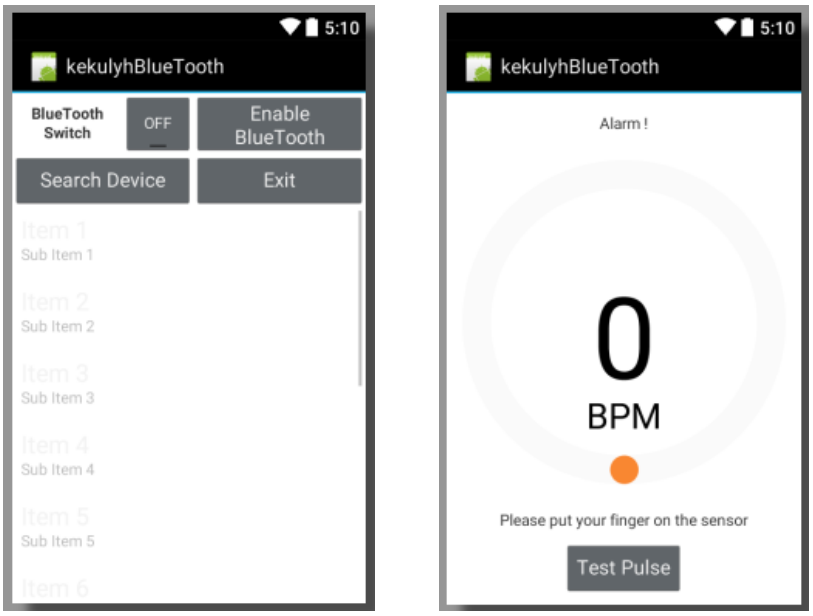


Figure 3.3 Interfaces of Android application

4 Testing Methods

The main idea is to test the functions individually. After ensuring that every part works well, the system should take an overall test.

4.1 Analogue Input

The pulse signal read from the sensor should be tested first. We have to ensure that the signal waveform is reflecting heart rate correctly. Otherwise, we have to adjust the circuit design or change another sensor.

4.2 Calculation of BPM

The algorithm of BPM calculation should be tested individually. Since the interference of dicrotic wave, we have to track the pulse rising up after a proportion of IBI times. The appropriate proportion has to be found due to the tests. Additionally, the algorithm is executed by timer counter interrupt, so that it may lead to some unexpected fault. We have to carefully test the calculation of BPM values since this is the principal function.

4.3 Android App

The Android app should be tested using other software means. With eclipse and Android SDK, we could test the interface and displaying functions. These functions are not related with the Arduino board so they could be tested independently. After that, we have to test the Bluetooth communication among the application and Arduino Due. We set a Bluetooth communication between the two components and test the communication without the calculation of BPM.

4.4 Overall Test

After every part was tested appropriately, we test the overall operation of the whole system. This is done by testing a calm pulse and a pulse after exercise. When using the Android application, both the application and the serial monitor of Arduino Due could display the correct value. This is what we expected for the project.

5 Problems Encountered

Here are some problems we encountered during the project.

First is the original analogue waveform. We tried many times to avoid noise and want to get better data. It has many noises at the beginning and we focused on adjusting our fingertip position on the sensor. But after referring some documents and papers, we found that the culprit is the AC frequency interferences. Since the signal of fingertips

is very small, when using computer which is powered by AC adapter to power the Arduino board, the waveform would be interfered by the power frequency noise. The solution is to use battery power instead of the AC adapter.

Second is the calculation of heartbeat based on the analogue data. This costs us lots of time to refer to papers so that we could find out the method of detecting the heartbeats. We have to refer to some algorithms of heart rate to summarize the appropriate algorithm.

The last problem is the Android application. Since the time is urgent, we can only finish an application with simple functions. Although writing Android application is not so difficult, the joint debugging with Arduino Due using Bluetooth is much more complicated.

6 Improvements

Our design has many deficiencies and we could make some improvements.

First, we could add a led screen onto the Arduino board to display the pulse waveforms and BPM value. This function would make the product more like a commercial product. Currently, the majority of products on the market also have such functions.

Second, we could add some other sensors to make it more integrated. People usually want to buy a medical product with multiple functions so that they could measure their body data without switching among numerous products. This could help the product improved into such kind of wearable devices.

Third, the function of Android app should be expanded. We would achieve the function that could display the varying data collected from the Arduino board. With the use of Internet, the medical data could be able to be stored and sent to some cloud service. Therefore, people who are using this product are able to see the analysis of their medical data and even comparisons of others' medical data.

7 Costs

Components	**Quantity**	**Price (AU$)**	**Total (AU$)**
Arduino Due	1	60	60
Pulse Sensor	1	31.35	31.35
Bluetooth Module	1	27.45	27.45
Total			**118.8**

8 Conclusion

In this project, we successfully achieve the goal of designing a Bluetooth pulse meter based on Arduino Due. During the project, there were many challenges for us but we have overcome them eventually. The experience of building a combined system is a great improvement for us and this enforced our understanding of embedded systems. Although the product is not such a powerful device, it indeed implements the basic functions. Regarding this as the beginning, we would develop more powerful devices from now on.

Reference

[1] Lishner, M., Akselrod, S., Avi, V. M., Oz, O., Divon, M., & Ravid, M. (1987). Spectral analysis of heart rate fluctuations. A non-invasive, sensitive method for the early diagnosis of autonomic neuropathy in diabetes mellitus. *Journal of the autonomic nervous system*, *19*(2), 119-125.

[2] Choi, S. J. (2004). Sleepy alarm system activated by heart pulse meter. *U.S. Patent No. 6,791,462*. Washington, DC: U.S. Patent and Trademark Office.

[3] Laukkanen, R. M., & Virtanen, P. K. (1998). Heart rate monitors: state of the art. *Journal of sports sciences*, *16*(sup1), 3-7.

[4] Wise Up! Jawbone UP24 vs UP features and differences. Retrieve from: https://jawbone.com/

[5] Seeedstudio Bluetooth shield. Retrieved from: http://www.seeedstudio.com/wiki/index.php?title=Bluetooth_Shield&uselang=en

[6] Pulse Sensor. Retrieved from: http://pulsesensor.com

[7] Open Hardware – Pulse Sensor. Retrieved from: http://pulsesensor.com/pages/open-hardware

[8] Allen, J. (2007). Photoplethysmography and its application in clinical physiological measurement. *Physiological measurement*, *28*(3), R1.

[9] Smith, R. P., Argod, J., Pépin, J. L., & Lévy, P. A. (1999). Pulse transit time: an appraisal of potential clinical applications. *Thorax*, *54*(5), 452-457.

[10] Sabbah, H. N., & Stein, P. D. (1978). Valve origin of the aortic incisura. *The American journal of cardiology*, *41*(1), 32-38.

[11] PulseSensor-Amped code. Retrieved from: https://github.com/WorldFamousElectronics/PulseSensor-Amped

Appendix 1 Block Diagram of System

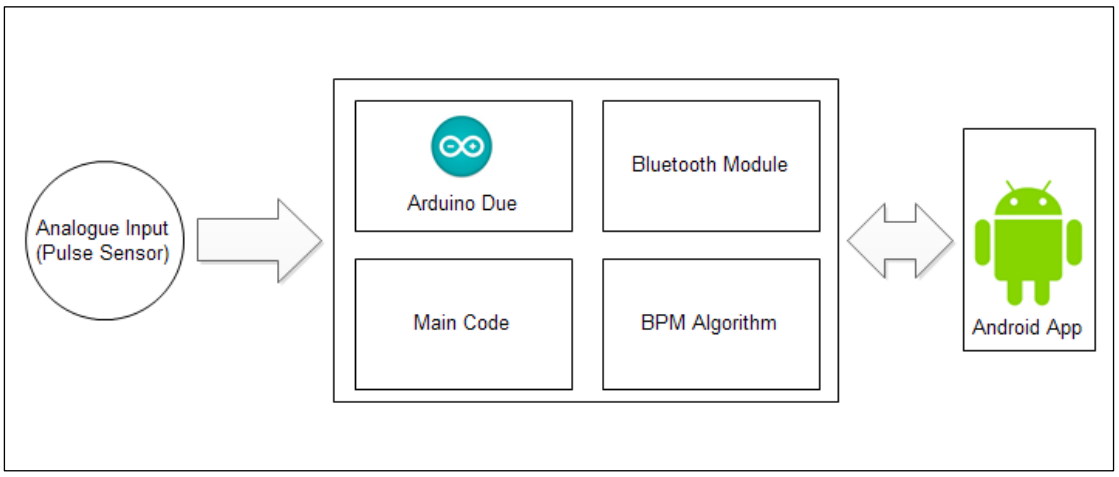


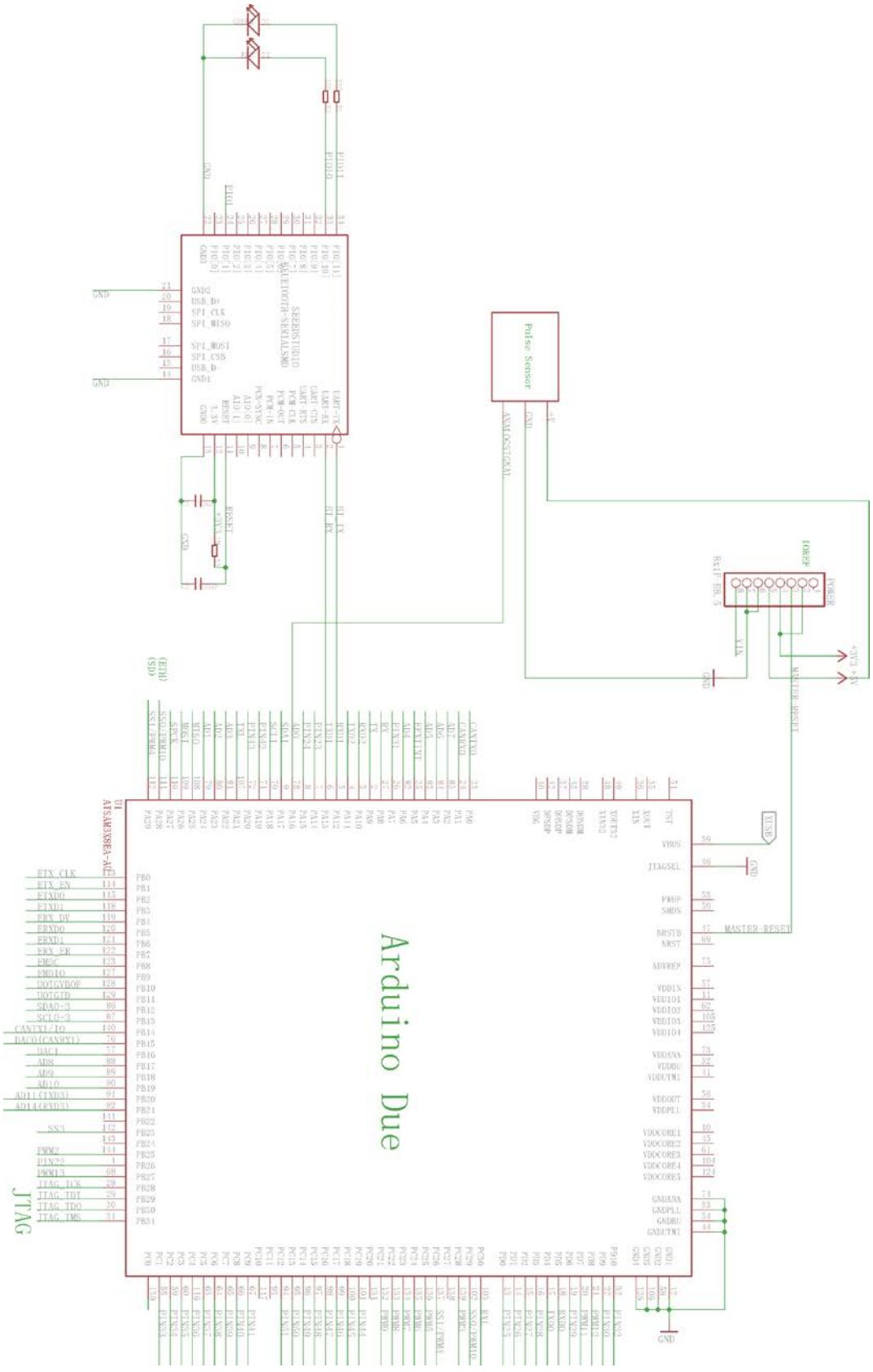
Figure appendix 1 Block Diagram of System

Appendix 2 Pseudo Code of Arduino Due

```
void setup(){
   set LED as output pin;
   serial begin;
   call interrupt function to read signal from sensor;

   set RxD as input pin;
   set TxD as output pin;
   call setupBluetooth function;
}


void interrupt(){
      Initialize timer to call an interrupt every period;
      call interrupt service routine to read data and calculate BPM every period;
}


void setupBluetooth()
{
       Bluetooth Serial begin;
       set baud rate;
       set the Bluetooth work mode;
       set the Bluetooth name;
```

```
        permit paired device to connect;
        auto-connection forbidden;
        make the slave Bluetooth inquirable;
        print message;
        flush Serial;
}


void loop(){
    if (heart beat detected){
       set LED high;
        call serialoutput function;
        set heart beat flag = 0;
    }
    else {
            set LED low;
    }

    if(Received request from Android app){
      Blurtooth serial print heart rate data;
    }
}
```

Appendix 3 Circuit Schematic Diagram



Appendix 4 Main Code for Arduino Due

```
#define FREQ 500                          // Sampling rate
volatile boolean state;

int pulsePin = 54;                        // Analog pin 0
int beatPin = 13;                         // Heart beat led

// Variables for BPM calculation
volatile int rate[10];                    // Array to store last ten IBI values
volatile unsigned long sampleCounter = 0; // Used to determine pulse timing
volatile unsigned long lastBeatTime = 0;  // Used to find IBI
volatile int P =512;                      // Peak of waveform
volatile int T = 512;                     // Trough of waveform
volatile int thresh = 525;                // Tresh of waveform when beat happens
volatile int amp = 100;                   // Amplitude of waveform
volatile boolean firstBeat = true;        // Used to seed rate array so we startup
                                          //with reasonable BPM
volatile boolean secondBeat = false;      // Used to seed rate array so we startup
                                          // with reasonable BPM

// Variables for interrupt service routine
volatile int BPM;                         // BPM (beats per minute)
volatile int Signal;                      // Pulse data collecting from sensor
volatile int IBI = 600;                   // IBI (inter beat interval)
volatile boolean Pulse = false;           // Heart beat is detected or not
volatile boolean QS = false;              // Quantified Self flag

void setup(){
   pinMode(beatPin,OUTPUT);
   Serial.begin(115200);                  // Serial tranmission speed
   startTimer(TC1, 0, TC3_IRQn, FREQ);    // Timer counter setup
   setupBlueToothConnection();            // Bluetooth module setup
   delay(20000);
}

void loop(){
   int recv;                              // Serial input from Android app through
                                          // Bluetooth serial
   if(Serial2.available()){               // Check if there's any data sent from the
                                          // remote bluetooth shield
      recv = Serial2.parseInt();          // Message(number '1') received from
                                          //Android app
```

```
      if(recv){
         Serial.println(recv);
            if (QS == true){
               digitalWrite(beatPin,HIGH);
               SerialOutput();                    // Serial output the BPM
               QS = false;                        // Reset Quantified Self flag
            }
            else{
               digitalWrite(beatPin,LOW);          // No beat tested
            }
            delay(20);
      }
   }
}

// Serial output
void SerialOutput(){
      Serial.println(BPM);
      Serial2.print(BPM);              // Without linebreak, because the Android input
                                       // stream has to deal with the socket stream data.
}

// Timer setup
void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t frequency) {
            pmc_set_writeprotect(false);
            pmc_enable_periph_clk((uint32_t)irq);
            TC_Configure(tc, channel, TC_CMR_WAVE |
                                      TC_CMR_WAVSEL_UP_RC |
                                      TC_CMR_TCCLKS_TIMER_CLOCK1);

            uint32_t rc = VARIANT_MCK/2/frequency;    // 2 because we selected
                                                      // TIMER_CLOCK1 above
            TC_SetRA(tc, channel, rc/2);               // 50% high, 50% low
            TC_SetRC(tc, channel, rc);
            TC_Start(tc, channel);

            tc->TC_CHANNEL[channel].TC_IER=TC_IER_CPCS; // TC Interrupt
                                                        // Enable Register
                                                        // RC Compare
                                                        // Interrupt
            tc->TC_CHANNEL[channel].TC_IDR=~TC_IER_CPCS; // TC Interrupt
                                                         // Disable
                                                         // Register
            NVIC_EnableIRQ(irq);
```

```
}

// Timer counter handler
void TC3_Handler()
{
   TC_GetStatus(TC1,0);
   state =! state;

   __disable_irq();                                  // Disable interrupt
   Signal = analogRead(pulsePin);                    // Read the analogue data
   sampleCounter += 2;                               // Keep track of the time in mS with
                                                     // this variable
   int N = sampleCounter - lastBeatTime;             // Monitor the time since the last beat
to avoid noise

   // Trough
   if(Signal < thresh && N > (IBI/5)*3){             // Avoid dichrotic interference by
                                                     // waiting 0.6 IBI
      if (Signal < T){                               // T is the trough
         T  = Signal;                                // Keep track of lowest point in pulse
                                                     // wave
      }
   }

   // Peak
   if(Signal > thresh && Signal > P){    // Thresh condition helps avoid noise
      P = Signal;                        // P is the peak
   }                                     // Keep track of highest point in pulse wave


   // Calculate BPM when heart beat happens
   if (N > 250){                                   // Avoid high frequency noise
      if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)*3) ){
         Pulse = true;                             // Set the Pulse flag when the beats happen
         IBI = sampleCounter - lastBeatTime;       // Measure time between beats in mS
         lastBeatTime = sampleCounter;             // Keep track of time for next pulse

         if(secondBeat){                           // If this is the second beat
            secondBeat = false;                    // Clear secondBeat flag
            for(int i=0; i<=9; i++){               // Seed the running total to get a
                                                   // realisitic BPM at startup

               rate[i]  =IBI;
            }
         }
```

```
      if(firstBeat){                        // If it's the first time we found a beat
        firstBeat = false;                  // Clear firstBeat flag
        secondBeat = true;                  // Set the second beat flag
        __enable_irq();                     // Enable interrupts again
        return;                             // IBI value is unreliable so discard it
      }

      // Keep a sum of the last 10 IBI values
      word runningTotal = 0;                // Clear the runningTotal variable

      for(int i=0; i<=8; i++){              // Shift data in the rate array
        rate[i] = rate[i+1];                // Drop the oldest IBI value
        runningTotal += rate[i];            // Add up the 9 oldest IBI values
      }

      rate[9] = IBI;                        // Add the latest IBI to the rate array
      runningTotal += rate[9];              // Add the latest IBI to runningTotal
      runningTotal /= 10;                   // Average the last 10 IBI values
      BPM = 60000/runningTotal;             // BPM calculation
      QS = true;                            // Set Quantified Self flag
    }
}

// Heart beat is over
if (Signal < thresh && Pulse == true){
   Pulse = false;                           // Reset the Pulse flag
   amp = P - T;                             // Get amplitude of the pulse wave
   thresh = amp/2 + T;                      // Set thresh at 50% of the amplitude
   P = thresh;                              // Reset these for next test
   T = thresh;
}

// Without heart beat for a long time
if (N > 2500){                              // If 2.5 seconds go by without a beat, set the
                                            // variables to default
   thresh = 512;
   P = 512;
   T = 512;
   lastBeatTime = sampleCounter;
   firstBeat = true;
   secondBeat = false;
}
```

```
  __enable_irq();                       // Enable interrupt

}

// Set Bluetooth module
void setupBlueToothConnection()
{
      Serial2.begin(115200);                    // Set serial2 to baud rate 115200
      Serial2.print("\r\n+STWMOD=0\r\n");    // Set the bluetooth work in slave mode
      Serial2.print("\r\n+STBD=115200\r\n");    // Set bluetooth module baud rate to
                                                // 115200
      Serial2.print("\r\n+STNA=Group9project\r\n");    // Set the bluetooth name
      Serial2.print("\r\n+STOAUT=1\r\n");         // Permit Paired device to connect me
      Serial2.print("\r\n+STAUTO=0\r\n");         // Auto-connection is forbidden
      Serial2.print("\r\n+STPIN=0000\r\n");       // Set pin code
      delay(2000);                                // This delay is required.
      Serial2.print("\r\n+INQ=1\r\n");             // Make the slave bluetooth inquirable
      Serial.println("The slave bluetooth is inquirable!");
      delay(2000);                                // This delay is required.
      Serial2.flush();                            // Flush serial2.
}
```