**STUDENT PLAGIARISM: COURSE WORK – POLICY AND PROCEDURE**

**MTRX 2700 COMPLIANCE STATEMENT**

**INDIVIDUAL / COLLABORATIVE WORK**

**I/We certify that:**

1. I/We have read and understood the *University of Sydney Student Plagiarism: Coursework Policy and Procedure*;

2. I/We understand that failure to comply with the *University of Sydney Student Plagiarism: Coursework Policy and Procedure* can lead to the University commencing proceedings against me/us for potential student misconduct under Chapter 8 of the *University of Sydney By-Law 1999* (as amended);

3. The Work undertaken in this course is substantially my/our own, and to the extent that any part of this Work is not my/our own I/we have indicated that it is not my/our own by Acknowledging the Source of that part or those parts of the Work.

Name(s) : Andrew Fredric Sekhar, Bevan Jones, Victor Zhang

Date : 04/06/15

**Final Report**

**ELECTROCARDIOGRAM**

**Group 6**

**ECG**

**MTRX 2700 Mechatronics 2**

**Major Project 2015**

**Group Certification**

We certify[1] that the design, implementation, and documentation presented by our Group are, unless otherwise acknowledged, the work of our group members only, and that the individuals named below were responsible for completing the following percentages of the modules named.

|  |  |  |
| --- | --- | --- |
| **Name** | **of the Module** | **Signature** |
| victor zhang  430415441 | 7 segment output display | victor |
| timer interrupt |
|  |
|  |
| Andrew Fredric  Sekhar  440285623 | SCI | Andrew F Sekhar |
| Timer interrupts |
| Electrodes |
| Calibration |
| Bevan  Jones  312144881 | Data processing | Bevan Jones |
| Data output on 7 segment display |
| Buzzer |
|  |
|  |  |  |
|  |
|  |
|  |
|  |  |  |
|  |
|  |
|  |
|  |  |  |
|  |
|  |
|  |

# 1. Introduction

## 1.1. Document Identification

This document describes the design of Robotic System for Jousting. This document is prepared by Group Leo Lou for assessment in MTRX3700 in 2015.

## 1.2. System Overview

The Robotic System is a system that can be controlled by the user or it can run in a fully automatic mode. Through the Xbee, the motor driver board (Pololu 707), the PWM motor, encoder and joysticks that controlled by the user. The robotic system (which is a mobile robot) can be operated in manual mode, auto assist mode and automatic mode. In addition, the status of the robotic can be displayed on a LCD screen.

## 1.3. Document Overview

* Part 1 of this document is an introduction of the report.
* Part 2 part of the report demonstrate the system overview and its expect performance. What its interface will be look like and show the design procedure.
* Part 3 of the document describe the user interface design and show how the system will be handle the error input.
* Part 4 describe the hardware components, the actuators, sensors and controller. It also point out how they can be integrated together
* Part 5 describe the software components of the robotic and controller system. The flow chart and data blocked will be explained. The robot motion algorism, serial communication (Xbee), controller algorism and robot mode will be shown. Other parts such as LCD user interface will be explained.
* Part 6 describe the testing procedure. It will explain how individual component will be test and demo it to the group member.

## 1.4. Reference Documents

The present document is prepared on the basis of the following reference documents, and should be read in conjunction with them.

—. MICROCHIP PIC18FXX2 Data Sheet

—. PIC18F452 to PIC18F4520 Migration Guide

—. Xbee manual.

—. Manual and circuit schematic for MNML\*PIC\*18 v2

—. STVHN3SP30 Data Sheet

—. Power Board Design Notes and Manual.

### 1.4.1. Acronyms and Abbreviations

Table 1-1 lists the acronyms and abbreviations used in this document.

Table 1-1: Acronyms and Abbreviations.

|  |  |
| --- | --- |
| **Acronym** | **Meaning** |
| PWM | Pulse Width Modulation |
| ISR | Interrupt service routine |
| ADC | Analog digital converter |
| Xbee | A kind of radio communication |

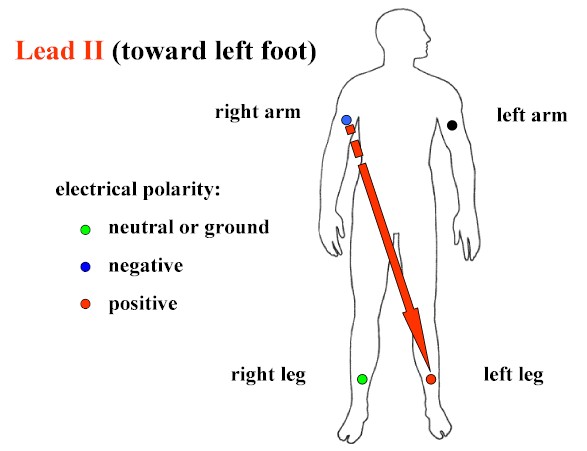
# 2. System Description

This section is intended to give a general overview of the robotic system and , of its division into hardware and software modules, and of its development and implementation.

## 2.1. Introduction

The system expect has the ability to measure the HBR using electrodes and calculate the heart rate per minute. Then display the value on the 7 segment display. Some extra functions can be added on the system such as beep sound \ emergency stop.

### 2.1.1 Hardware components introduction:

There have 3 hardware components, electrodes, amplifier and HCS12 dragon microcontroller. Electrodes are used to detect the voltage signal from human body due to the heart beat. Amplifier are used to scale the voltage signal from the electrodes so that it can be processed by the MCU easier. MCU is just like the CPU of the computer, it is used for determine the HBR and display it on the 7 segment display.

### 2.1.2. Software components:

Software contain 5 parts. ADC, peak detection function, 7 segment display function, timer interrupt system and SCI. ADC will convert the analog signal to digital signal. Peak detection will find out the peak value (the HBR) and using timer interrupt to calculate the HBR per/min, it will output a value of HBR to the function of 7 segment display. Timer interrupt system uses for control the sample rate to read data as well as peak detection. 7 segment display function will handle the HBR value to 7 segment LED. SCI is a user command input via putty on PC terminal.

## 2.2. Operational Scenarios

Electrodes use for measure the heart rate from human body. It will wrap on the human body in 3 different position, which is left hand pulse, right hand pulse and the bottom of left foot.

The electrode should wrap tightly so that the signal will be much clear. Connect the electrodes to the amplifier and connect the amplifier with the oscilloscope in parallel state so that the oscilloscope will display the raw ECG signal directly. The user will press either ‘c’, ’s’, ‘h’ on putty in order to control the system response.

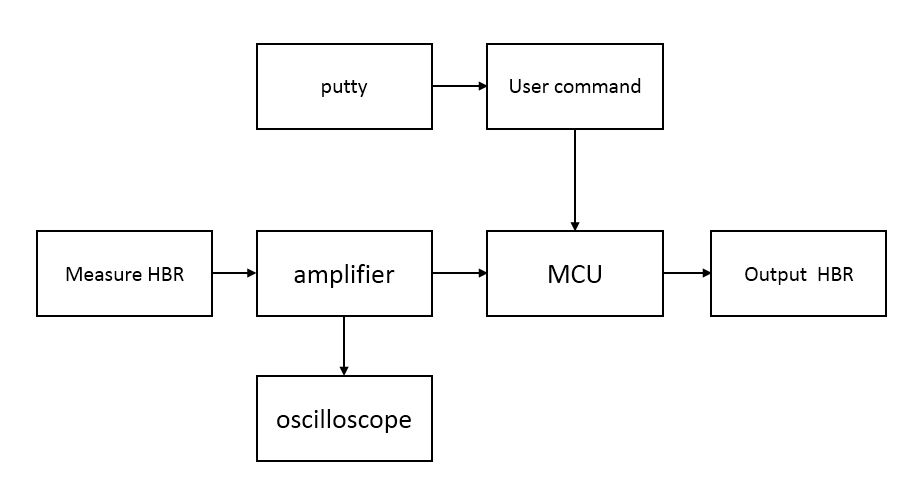
Some failure might occur in the system. Such that the electrodes might not warp on the arms/leg correctly so that the signal of the beat rate will contain lots of noise. Hence it is hard to measure the correct HBR.

Another failure occur is the timer interrupt system. It is easy to crash on the timer interrupt when different timer function implement together. When this happened, the system will not working normally, the sample rate/ 7 segment refresh rate/ HBR calculation will not working as expect.

## 2.3. System Requirements

The operational scenarios considered place certain requirements on the whole ECG system, and on the modules that comprise it.

**A ECG system general structure is shown below:**



**Requirement 1:**

The user interface must be implemented on PC.

**Requirement 2:**

A user must be able to initiate a calibration at any time. This shall be achieved using the SCI to PC. When a calibration has been initiated, the current offset voltage must be displayed on the 7 segment display for 3 seconds.

**Requirement 3:**

7 segment display is continuously output.

**Requirement 4:**

The audio tone need to be enable.

**Requirement 5:**

The system can be calibrate itself when power up.

**Requirement 6:**

SCI shall conform to RS-232C standard.

**Requirement 7:**

Press ‘h’, a list of available command should display on PC terminal.

### 2.3.1. Non-Functional (Quality of Service) Requirements

* The 7 segment display is continuous. Therefore a timer interrupt should be preset so that it can sample the HBR at a setting frequency.
* The input electrodes signal contain lots of noise. A filter need to be use in order to remove the high frequency noise.
* Input data should convert from analog to digital.
* Sample the input data should have a high frequency so that it can measure the HBR accuracy.

#### 2.3.1.1. Performance

* The HBR value should be less than +-2% torance.
* The audio tone should be enable for 0.1s when the peak value is detected.
* The 7 segment display should be displayed a value for 0.5s. This can make the value more readable.
* Calibration offset voltage correct to 1 decimal place when calibrate mode is ON.

#### 2.3.1.2. Interfaces

Using putty in PC terminal as command input / interface output.

When the program run, a guide message will be display on Putty.

When user press ‘c’ calibration mode will be trigger for 3s.

When user press ‘s’, start measure the HBR.

When user press ‘h’, show the help message.

Other input character such as ‘g’, ‘t’ etc. An error message will be trigger and display on the output interfaces.

#### 2.3.1.3. Design Constraints

The programming language should be either on C language or assembly language.

Any logic required shall be implemented principally by an Dragon12 Plus 12 Evaluation Board.

The ECG circuit provide is the main functional element of the system and it could not be modified. It must be used exactly as specified in the accompanying documentation.

# 3. User Interface Design

3.1 Overview

The user interface is composed of a keyboard to input commands to the microprocessor, a monitor to display the help function, the buzzer built onto the dragon board to give audio feedback to the user when invalid keys are pressed and to beep when there is a heart beat, a seven segment display capable of showing 4 characters at a time, and there are three electrodes that detect electrical impulses from the user’s body as input. 2 are placed inside elastic cotton armbands bands which are to be attached to the hands of the user and one placed inside a cotton elastic band similar to the arm bands but which has ends held together by velcro that is to placed on the user’s legs.

## 3.2. Interface Design

### 3.2.1. User Inputs and Outputs

User data is made available to the system by electrodes and keyboard. The electrode was used to measure the input data for calculating the heartbeat rate. Data was sampled by the system from the user’s at 160 Hz making available 160 readings per second from which the heart beat could be detected.The keyboard was used to allow the user to request help from the system, to force the system to recalibrate, and start or stop the system. When the system start to run the help page is automatically printed containing instruction to the user on how to start or stop the system operation, request the help page and force a calibration. The help message is a follows:

Help page:

Press: ‘h’ for help

‘c’ to calibrate

‘s’ to start or stop

When the user clicks the a valid key the associated function is carrier out.

Feedback is given to the user using the seven segment displays, and the audio tone played by the buzzer.

The seven segment display is used to communicate the heart rate and calibration voltage offset to the user. When the calibration function is run, at startup or during operation the voltage offset is calculated by the microprocessor and displayed on the seven segment display by the use of 2 characters. The first character represents the integer part of the voltage, this ranges from 0 to 5. The second character represents the first decimal place of the offset voltage, this ranges from 0 to 9. When in normal operation the seven segment display is used to display the heart rate. This makes use of 3 characters each with a possible range of 0 to 9. The first character displays the component of the heart beat rate that is in the order of hundreds, the second character displays the component of the heart beat rate that is in the order of tens, and the third character displays the component of the heartbeat that is in the order of single digits. The buzzer is used to play a beep sound each time there is a heartbeat detected or an invalid key is pressed by the user.

### 3.2.2. Input Validation and Error Trapping

When the user inputs a character input the input is validated before the request is fulfilled by the system. The input is handled by the serial communication interface interrupt subroutine. The subroutine checks the input character against a list of valid characters. If the input character is valid the associated function is executed else the buzzer tone is played to indicate to the user that the input is invalid. The list of valid characters are ‘h’, ‘H’, ‘c’, ‘C’, ‘s’, and ‘S’. The input character is checked against the list with an ‘if’ block. If the input is ‘h’ or ‘H’ the help function is set to run with the use of a flag, else if the input is ‘c’ or ‘C’ the calibrate function is set to run with the use of a flag, else if the input is ‘s’ or ‘S’ the system starts or stops operation. The last component of the ‘if’ block is an else statement which plays the audio tone, this part of the code is only reached if the input is invalid. After the tone to indicate the invalidity is played the system exits the interrupt subroutine and returns to its prior state of operation.

# 4. Hardware Design

## 4.1. Scope of the ECG System Hardware

The whole system is composed of the microcontroller, the signal amplification and filtration board, oscilloscope and probe, variable DC power supply, computer for user input, and the electrodes. Of these only the electrodes are designed and constructed by the team, all other components are provided in the form that they are used.

## 4.2. Hardware Design

### 4.2.1. Power Supply

Power is supplied to the system by the power cable of the development board, the amplification and filtration board is powered at 12 volts by the variable dc power supply, the computer is powered by the wall socket, and the electrodes are presumed to be powered by the human subject since that is where the input signal is coming from.

### 4.2.2. Computer Design

The hardware of the component is unknown to the design team. The interfacing component used are the serial port and what look to be a usb port. They are used for serial communication and loading programming data to the system.

### 4.2.3. Sensor Hardware

Each electrode is composed of a flat copper cuboid with the dimensions: length = 50 mm, width = 15 mm and depth of ~0.5mm. This is attached, by soldering, to the copper core of coaxial cable which is used to connect the electrode to the filtration and amplification board. On the end of the coaxial cable attached to the electrode the shielding is cut short by approximately a centimetre so that there is no chance of the shielding coming in contact with the electrode and causing extra noise from being picked up the electrode. On the other end where the cable connects to the amplification board the core is fed to the associated connector and the shielding from all the cables are fed to grounding pins. Then, finally after testing for operation, the electrodes were glued onto the arm and leg bands that held them to the user and/or operator.

### 4.2.4. Operator Input/Output Hardware

Input from the operator and/or user is obtained from the keyboard connected to the computer and fed through the serial communications interface to the microcontroller. Output is provided by via the serial communications interface to the screen of the attached computer when help is requested, auditory tone from the buzzer connected to microcontroller when the input is invalid or when a heartbeat is detected, and visual representations of the heart beat are provided on the oscilloscope.

## 4.3. Hardware Validation

The hardware provided were assumed to be working. As for the electrode the testing sequence was to attach the the constructed electrodes to the amplification and filtering board which is connected to the oscilloscope in the operational setup and look for a valid signal.

## 4.5. Hardware Maintenance and Adjustment

The required maintenance for the subsystems other than the electrode to be kept in working order and for the electrode surface to be cleaned as required to maintain good contance with the observed subject or user/operator.

# 5. Software Design

## 5.1. Software Design Process

The software was designed in a top down, divide and conquer method. Starting from the main it was broken down into all the required modules and each module was then designed individually.

## 5.1.1. Software Development Environment

Software: the software used during development was mainly the Freescale CodeWarrior IDE utilising its Full Chip Simulation for the HCS12 dragon microcontroller. Although MATLAB was initially used to test the method of filtering the sample data and calculating the beats per minute. Hardware: The designed software was tested and revised using the HCS12 dragon microcontroller.

### 5.1.2. Software Implementation Stages and Test Plans

The implementation and testing of the software was broken down into following stages:

1. A skeleton code was created with empty functions and interrupts outlining the intended overall integration of the project. This was generated using the Block diagram and discussed together as a group.
2. Then the following modules/functions were individually designed implemented and tested.
   1. The user interface with the help message utilising the SCI interrupt. This was implemented and tested on the HCS12 dragon board and worked.
   2. The sevenseg module was created to display arbitrary values of bpm. This utilised a timer interrupt and was tested using the HCS12 board.
   3. The software for finding the heart beats and beats per minute was first implemented in MATLAB using the provided sample data. This code was then converted in to C and again tested using the sample data in CodeWarrior Full Chip simulation.
3. After this we began to encounter problems as we attempted to integrate the components together. After integrating the above 3 components the calibration module, the ADC interrupt and the buzzer with its timer interrupt were then implemented directly into the integrated software. Then before testing on the HCS12 board but only the Full Chip simulation the sevenseg was changed to function for both the offset and bpm and to change the display only through the timer interrupt.
4. This completely integrated software was then tested on the HCS12 board but would not function. This led to continually crashing and having to reset the board while attempting to discern what the problem was. In an attempt to solve this, a number of minute and major changes to the software were made in an attempt to get everything to function.
5. In the end, the integrated software failed to function. The only working part was the user interface with the SCI interrupt and the program crashed whenever it tried to call on a timer interrupt.

After step 3 the software became a mess. In hindsight, we should have continued to develop, implement and test each module separately. Then we should have integrated in small stages. First by adding the ADC interrupt to the heartbeat module then test in real time with the electrodes. Then integrate the sevenseg onto this using the calculated value for bpm and test this using the board. Repeat this with the calibrate module while continually testing and then integrate the two together along with the user interface.

## 5.2. Software Quality Assurance

Measures taken to control/improve software quality:

* Comments were written during the writing of the code.
* Changes and improvements in code where noted in the header along with the date when they were made.
* Code was first compiled through CodeWarrior Full Chip simulation before being run on the board to remove any compilation errors.

## 5.3. Software Design Description

### 5.3.1. Architecture

High-level flow:

* Setup: configures the serial, ADC, timers and 7seg.
* Display Help: displays a help message through the serial explaining the user interface.
* Calibration: Sets the pulse oximeter chip to calibration mode and calculates the voltage offset.
* Heart Rate: filters the data from the ADC and calculates the heart beats per minute.
* 7seg Display: converts the value of bpm or voltage offset into a digit-by-digit array and then initiates the 7seg timer interrupt.

# 

The program either remains idle, displays a help message, calibrates or calculates the heart rate depending on the user input through the serial. When the program has calculated a value for beats per minute in the heart rate module, it displays it on the 7seg.

### 5.3.2. Software Interface

The only interface with the software the user has is through the serial. When a key is pressed on the computer, it triggers the serial interrupt and initiates one of three things:

* Display a help message.
* Start calibration.
* Start/Stop the program.

### 5.3.3. Software Components

Software modules:

Setup:

1. configures the serial, ADC, timers and 7seg.

2. Display Help: displays a help message through the serial explaining the user interface.

3. Calibration: Sets the pulse oximeter chip to calibration mode and calculates the voltage offset.

4. Heart Rate: filters the data from the ADC and calculates the heart beats per minute.

5. Seven Segment Display: converts the value of bpm or voltage offset into a digit-by-digit array and then initiates the 7seg timer interrupt.

6. Buzzer: Starts the buzzer and initiates the Buzzer interrupt.

7. SCI Interrupt: Triggers when there is a user input. Changes the current state depending on the input.

8. ADC Timer Interrupt: Triggers every 160Hz, reads value from ADC.

9. 7seg Timer Interrupt: Triggers every 100Hz, displays bpm or voltage offset value on 7seg display.

10. Buzzer Timer Interrupt: Turns buzzer off.

## 5.4. Conceptual Design

### 5.4.1. Software Module Setup

This is a culmination of all the setup requirements for all the other modules. This includes configuring the Serial communication interface, ADC, Timers, Seven-segment display and setting the calibration and buzzer pins to output. In addition, to enable interrupts.

1. SCI setup:

* Use SCI 1.
* Set baud rate to 9600 for 24MHz clock.
* Control Register 2: Turn on transmit and receive enable bits and initiate interrupts on receive.

2. ADC setup:

* Use ATD 1 as PAD 10 the input from the ECG chip is on channel bit 2 of ATD 2.
* Control Register 2: Allow normal operation.
* Control Register 3: Set conversion sequences to 1.
* Control Register 4: Set resolution to 8 bit and clock prescale to divide by 24 for the 24MHz clock.
* ATD1DIEN (Input Enable Register): Enable input from Pad 10.
* Control Register 5: Set right alignment, scan mode and set read to pad 10 (which is bit 2 on ATD 1).

3. Timer setup:

* Setup 3 output compare timer interrupts. One each for the ADC, 7seg and buzzer.
* Control Register 1: Set it to run normally and fast clear.
* Control Register 2: Set the timer prescale to 128.
* TIOS (input output select): Set TC0, TC2 TC4 to output compare.
* Initiate first output compare for all timers.
* TIE (interrupt enable): Enable interrupts for all timers.

4. 7seg setup: Set Port B, J and P to output. Then enable LEDs by setting Port J to = 0x00.

5. Calibration and Buzzer pins setup: Set Port T bit 5 (buzzer) and Port M bit 7 (calibrate pin) to output.

6. Enable Interrupts.

#### 5.4.1.1. Assumptions Made

* Assume the E-clock is running at 24MHz

### 5.4.2. Display Help

This function transmits a help message through the SCI to the user explaining the user interface.

# 

The following is the displayed helpMessage:

Help Page

Press: h for help

c to calibrate

s to start or stop

### 5.4.3. Calibration

This module initiates the calibration mode of the ECG chip and calculates the voltage offset. Once the offset is calculated it passes this value to the 7seg display, waits for 3 seconds then disables the 7seg display.

# 

The method used to calculate the offset is to wait until the wave generated from the chip levels out. It is determined that the wave has levelled out when two consecutive ADC values are equal.

To convert the value to a voltage offset between 0-5 volts the following calculation is made:

offset = (value\*50)/255

This gives a value between 0-50 but is late converted to 0-5 with a decimal point to be displayed by the 7seg display.

#### 5.4.3.1. Assumptions Made

Assume the ADC has already been configured and set up.Also, assume that the calibration pin has been set to output.

#### 5.4.3.1. Constraints on Module Calibration Performance

There is a problem with this method as there may not ever be two equal consecutive values because of noise. To counteract this we can make it so they are equal enough to each other within a certain range or just take the value after an arbitrary amount of time.

### 5.4.4. Heart Rate

In this module, the data collected through the ADC is first filtered then searched for any heart beats present. Whenever a heartbeat is found a new value for heart beats per minute is calculated and then passed on to the 7seg display.

The Heart rate function is continually run through by the main and filters the data whenever a new ADC sample is attained from the ADC timer interrupt. Once this value is filtered, it is then checked to see if there is a heartbeat. When a heartbeat is detected it activates the buzzer and calculates the beats per minute using the sub function Calculate bpm.

If no heartbeat is detected after a certain time, the value for bpm is set to zero. This time is calculated as the minimum healthy heart rate of 40bpm or 0.667Hz.

# 

# **Filter:**

# 

The filtering method in use is a high pass filter. The use of a high pass filter with a cut-off frequency of 50Hz removes the voltage offset from the signal and increases the value for high gradient portions of the wave. After the signal has passed through the high pass filter it is then squared.

Equations for a high pass filter:

Where:

* y is the array of filtered signal data
* x is the array of unfiltered signal data
* is the sampling period
* is the cut off frequency

In this case, the sampling rate is 160Hz and cut-off frequency is 50Hz giving the following value for alpha:

Example of filtered data using MATLAB:

|  |  |
| --- | --- |
|  |  |
|  |  |

Unfiltered sample data above and filtered and squared data below

# 

# **Heartbeat detection:**

To detect a heartbeat the filtered data has to pass the following 3 conditions:

1. previous filtered value < filtered value > next filtered value

2. filtered value > threshold

3. heart beat has occurred after a time equal to the max heart rate

The threshold value used was 200. Condition 3 was put in place to ignore the potential noise above the threshold; the max heart rate was considered as 150 bpm or 2.5Hz.

# **Calculate bpm:**

After a heartbeat has been detected, a new bpm value is calculated.

# 

This function uses the array peaktimes containing the value for last 5 times a heartbeat was detected. This time is in sample counts, a count incremented every time the ADC timer interrupt is triggered. This function then takes the average of these peaktimes (ignoring values of 0) to calculate a new value for beats per minute (bpm).

#### 5.4.4.1. Assumptions Made

* Assume the ADC has already been configured and set up.
* Assume the Sample Rate is at 160Hz

#### 5.4.4.2. Constraints on Module Heart Beat Performance

* For the filter to work the sampling rate must be at 160Hz and cannot be changed as this will affect the value of alpha.
* This filter completely distorts the signal meaning that no other calculations/observations can be made from the ECG signal except to calculate the heart rate.

### 5.4.5. Seven-Segment Display

This function takes a value of either bpm or voltage offset, converts it into digits and then initiates the 7seg timer interrupt.

# 

# 

#### 5.4.5.1. Assumptions Made

* Assume the 7seg timer has already been set up.
* Assume that the values for bpm and are between 0-999.

### 5.4.6. Buzzer

This simply turns the buzzer on then starts the buzzer interrupt. This function is called from the Heart Rate module whenever a heartbeat is detected.

# 

#### 5.4.6.1. Assumptions Made

Assume the Buzzer timer has already been set up.

## Interrupts/Timers:

### 5.4.7. SCI Interrupt

This interrupt triggers whenever there is a user input. The interrupt then reads the user input through the serial and changes the state of the program accordingly.

# 

### 5.4.8. ADC Timer Interrupt

This interrupt triggers every 160Hz, reads value from ADC and stores in a global variable.

160Hz = E-clock/(160\*Timer Prescale) = 1172 clock cycles with prescale of 128.

# 

#### 5.4.8.1. Assumptions Made

Assume the ADC timer has already been set up and started.

### 5.4.9. Seven-Segment Timer Interrupt

Triggers every 100Hz and displays current digit value of bpm or voltage offset on the seven-segment display. Then increments the digit for the next time this interrupt is triggered.

100Hz = E-clock/(100\*Timer Prescale) = 1875 clock cycles, with prescale of 128.

# 

#### 5.4.9.1. Assumptions Made

* Assume the 7seg timer has already been set up and started.
* Assume the bpm/offset values have been broken up into a size 3 array.

### 1.4.10. Buzzer Timer Interrupt

Triggered shortly after buzzer has been turned on. Turns buzzer off.

# 

# 

#### 5.4.10.1. Assumptions Made

* Assume the Buzzer timer has already been set up and started.
* Assume the buzzer has been turned on.

# 5.5. Software Module Requirements:

### 5.5.1. Setup Functional Requirements.

#### **5.5.1.2.** **Process**

Initialises SCI, ADC and Timer Interrupts.

#### **5.5.1.3.** **Outputs**

No direct Outputs. However, it does initiate interrupts.

#### **5.5.1.4.** **Timing**

Small delay of 10 cycles per timer interrupt is needed to initialize.

### 5.5.2. Display Help Functional Requirements

#### 5.5.2.1. Inputs

Takes the global char array helpMessage and the int helpMessageLength.

#### 5.5.2.2. Process

Goes through the helpMessage array and transfers each char through the SCI.

#### 5.5.2.3. Outputs

Outputs message through SCI for user.

#### 5.5.2.4. Timing

Waits for SCI data register empty flag before sending each char

### 5.5.3. Calibration Functional Requirements

#### 5.5.3.1. Inputs

Takes inputs from the ADC.

#### 5.5.3.2. Process

Uses ADC input to calculate the voltage offset. Then sends this offset to the 7seg display, delays 3 seconds then turns off 7seg.

#### 5.5.3.3. Outputs

Alters global variable offset.

#### 5.5.3.4. Timing

Waits until there are two consecutive values from the ADC. Then delays for 3 seconds before turning off 7seg.

#### 5.5.3.5. Failure modes

If there are no equal, consecutive values after a certain time it just takes the offset as the last value after that time.

### 5.5.4. Heart Rate Functional Requirements

#### 5.5.4.1. Inputs

Takes the ADC input generated through the corresponding ADC Timer Interrupt. Utilises 4 global arrays and 2 global variable as follows:

* unfiltered[2] – contains previous and current unfiltered ADC value to be used in filter.
* filtered[3] – contains 3 filtered values which are used in filter.
* squared\_filtered[3] – contains the 3 squared values of the above filter array, used to calculate the heartbeats.
* peaktimes[5] – contains the last 5 times (in samplecounts) when a heartbeat occurred, used to calculate bpm.
* bpm – global variable beats per minute displayed using the 7seg display.
* samplecount – a count incremented every ADC timer interrupt, used to calculate bpm.

#### 

#### 5.5.4.2. Process

Filters input signal from ADC. Checks for heartbeat, and if present calculates a new value for bpm.

#### 5.5.4.3. Outputs

Outputs value for bpm via a global variable.

#### 5.5.4.4. Timing

Only enters function when a new sample value from the ADC is attained. Must complete all processes in Heart Rate before next sample.

#### 5.5.4.5. Failure modes

If there is no heartbeat detected after the maximum healthy heart rate of 150Hz, the value for bpm is set to zero and the all the values in the array peaktimes are set back to zero.

### 5.5.5. Seven-Segment Display Functional Requirements

#### 5.5.5.1. Inputs

Takes the global variable bpm or offset.

#### 5.5.5.2. Process

Splits the value of bpm or offset into digits to be used in 7seg, then starts 7seg timer interrupt.

#### 5.5.5.3. Outputs

Outputs digit by digit value to be used in interrupt to global array of size 3. Starts 7seg timer interrupt.

#### 5.5.5.4. Timing

Only enters function when there is a new bpm value or it is in the calibrate state.

#### 5.5.5.5. Failure modes

N/A

### 5.5.6. Buzzer Functional Requirements

#### 5.5.6.1. Inputs

N/A

#### 5.5.6.2. Process

Turns buzzer on, starts buzzer timer interrupt.

#### 5.5.6.3. Outputs

Turns buzzer on, starts buzzer timer interrupt.

#### 5.5.6.4. Timing

N/A

#### 5.5.6.5. Failure modes

N/A

### 5.5.7. SCI Interrupt Functional Requirements

#### 5.5.7.1. Inputs

Takes user input through the Serial Data Receive register.

#### 5.5.7.2. Process

Changes the state of the program depending on the user input.

#### 5.5.7.3. Outputs

Changes state by changing the value of the global variable flags: helpFlag, calibrateFlag and runFlag.

#### 5.5.7.4. Timing

Interrupt triggers every time there is a user input trough the SCI.

#### **5.5.7.5.** **Failure modes**

N/A

### 5.5.8. ADC Timer Interrupt Functional Requirements

#### 5.5.8.1. Inputs

Reads value from ADC data register.

#### 5.5.8.2. Process

Stores value of ADC in global variable.

#### 5.5.8.3. Outputs

Stores value of ADC in global variable readValue.

#### 5.5.8.4. Timing

Interrupt triggers according to the sampling rate of 160Hz.

#### 5.5.8.5. Failure modes

N/A

### 5.5.9. 7seg Timer Interrupt Functional Requirements

#### 5.5.9.1. Inputs

Uses the 3 global arrays and 1 global variable:

* display[10] – constant array containing the Hex decimal values equivalent to the digits 0-9 applicable for the Port B LEDs.
* segment[4] – constant array containing the Hex decimal values which determine the segments 1-4 for Port P.
* sevensegValues[3] – the offset or bpm value separated into 3 digits.
* sevensegCount – a count that counts from 0-2 and repeating to display all 3 digits.

#### 5.5.9.2. Process

Displays the current digit value of offset or bpm determined by the sevensegCount onto the HCS12 dragon microcontroller board.

#### 5.5.9.3. Outputs

Outputs to the LED 7 segment display on the HCS12 dragon microcontroller board.

#### 5.5.9.4. Timing

Interrupt triggers every 100Hz.

#### 5.5.9.5. Failure modes

N/A

### 5.5.10. Buzzer Timer Interrupt Functional Requirements

#### 5.5.10.1. Inputs

N/A

#### 5.5.10.2. Process

Turns off Buzzer.

#### 5.5.10.3. Outputs

Turns off Buzzer.

#### 5.5.10.4. Timing

Interrupt triggers 100Hz.after started by Buzzer function. Does not start itself again.

#### 5.5.10.5. Failure modes

N/A

## 5.6. Preconditions for Software

### 5.6.1. Preconditions for System Startup

* Serial must be plugged into board.
* Electrodes must be connected via the ECG chip to the board.
* To use the user interface a program such as PUTTY must be set up and running.

### 5.6.2. Preconditions for System Shutdown

The system can be stopped by either pressing ‘s’ and waiting or by manually shutting down the board.

# 6. System Performance

### 6.1.1. Performance Testing

The following tests were performed in the following order:

1. The user interface with the help message wash implemented and tested using the HSC12 board. Functioned normally however, the SCI interrupt was implemented using the minimal startup method.
2. The Heart Rate module was first implanted and tested in MATLAB using all 5 sample datas provided (Andrew 160Hz, Brunner 160Hz,200Hz,300Hz,500Hz). The module functioned successfully and the required threshold was assigned from these simulations. This Heart Rate module was then translated into C and successfully tested with the Brunner 160Hz sample data. The ADC interrupt has not been implemented or tested yet.
3. The Seven-Segment Display module was implemented and tested on the HSC12 board successfully. However the delay between switching which seven-segment digit was in use was still not utilised in an interrupt format.
4. The electrodes and straps where developed and tested with the ECG chip to obtain a ECG signal. This successfully produced an ECG signal, however it was fairly noisy and there was quite a difference depending on who was measured. This was fixed by applying more pressure to the electrode onto the skin.
5. The software was then integrated combining the user interface the Heart Rate module and the Seven-Segment Display also implementing the ADC interrupt and 7seg interrupt. This was first tested on the CodeWarrior Full Chip Simulation and successfully compiled and ran. However when run directly onto the HSC12 board it crashed the system. This was assumed to be a problem with the interrupts. From now on 2 versions of the software were made, one using the minimal startup method for interrupts and the other implementing the interrupt method using ANSI startup with the isrvactors.c include.
6. Assuming that one of the interrupt methods would fix the problem, the calibrate module and buzzer where directly added into the integrated software without testing. This was then tested on the HSC board with neither interrupt method successful. In an attempt to solve and discover the problem a number of minor and major changes to the code were made.
7. A number of different versions of the code with a different method for interrupts(minimal or ANSI startup), the ADC (use cycles or scan mode) and with different methods of controlling the 7segment display (mainly relating how to turn it off after 3 second delay in calibration). There were also a few adjustments in the method for the buzzer and 3 second delay.
8. In the end none of these versions successfully functioned. The final product ended up being the version that successfully used the user interface and calculated a offset voltage. The 7seg, Buzzer, ADC interrupt and hence the Heart Rate module did not function successfully.

### 6.1.2. State of the System as Delivered

* Electrodes – The electrodes functioned and produced a ECG signal, although the signal was fairly noisy.
* User Interface – Printed Help message and ran as required.
* Calibration – Successfully calculated a voltage offset (accuracy unknown), however crashed the program when attempting to display the value on the 7segment display. Since it could not start the 7seg display it could not delay for 3 seconds.
* Heart Rate – Did not successfully get a value from the ADC interrupt and so did not calculate any value for beats per minute or detect any heartbeats.
* Seven-Segment Display – Did not display either the offset or bpm, crashed program when the 7sg interrupt was utilised.
* Buzzer – Did not activate or turn off the buzzer on the board.

This System did not meet the specifications requirements.

### 6.1.3. Future Improvements

1. Future improvements to the system are listed in order of priority:
2. The software must be broken up into all the individual modules. These modules must then be individually implemented and tested on the HSC12 dragon microcontroller board. Once each individually module is tested and functioning correctly the software can gradually be integrated in stages. The first stage would be to integrate the Heart Rate module successfully with the ADC interrupt, get a real time sample single and test to see if the method is truly sound. Then integrate the 7seg display with its interrupt to this and test again. Following on in this manner of integrating one part at a time and testing until fully integrating the software.
3. Once the system is functioning correctly and adheres to the required specifications some extra functionality may be added. The simplest extra functionality would be to provide a warning via the buzzer whenever a person’s bpm goes down to zero.
4. Another extra function which would be useful is for the user to be able to turn on/off the 7seg display and the buzzer via the SIC interrupt.
5. To further increase the functionality of the system a method for the user to alter the sample rate could be implemented. Tis also includes changing the parameters for the high pass filter in the Heart Rate Module.

# 3. Conclusions

Although the system did not achieve its required functionality, we have learned a lot about not only programming electrodes and electrocardiograms. We have learned the importance of the divide and conquer software design method and of continually testing each individual component. By not successfully implementing these techniques we wasted a lot of time trying to fix and discover why the system would not function and ultimately the finished product did not meet the required specification standards. In conjunction with this we have learnt the importance of proper planning and communication between the group before attempting to just write code and hope that it works.

# 

# Appendix A: C Code

#include "derivative.h" /\*\* derivative-specific definitions \*/

#include <hidef.h> /\*\* common defines and macros \*/

/\*\* ----- Start general definitions ----- \*/

#define setFlag 0x01

#define resetFlag 0x00

#define bit0 0x01

#define bit1 0x02

#define bit2 0x04

#define bit3 0x08

#define bit4 0x10

#define bit5 0x20

#define bit6 0x40

#define bit7 0x80

#define clear 0x00

#define fill 0xFF

/\*\* ----- End general definitions ----- \*/

/\*\* ----- Start set up definitions ----- \*/

#define MS\_DELAY E\_CLOCK/1000 //! ms delay for 7seg in clock cycles

#define DECIMALPOINT 0x80 //! decimal point for 7seg LEDs

//! Timer setup definitions

#define tscr1 bit7|bit4 //! Run normally

#define tscr2 bit2|bit1|bit0 //! Prescale 128

#define tios bit7|bit6|bit5|bit1|bit0 //! Enable TC7,TC1 and TC0 output compare and tc5 for buzzer

#define tieRegular bit1|bit0 //! Enable interrupt for TC1 and TC0

#define tieCalibrate bit7|bit6 //! Enable interrupt for TC7 and TC6

#define HzAdd100 1875 //! Addition value for operation at 100 Hz

#define HzAdd160 1172 //! Addition value for operation at 160 Hz. Actual value is 1171.875

#define HzAdd4000 47 //! Addition value for operation at 4 kHz. Actual value is 46.875

#define overflow 0 //! Set this to timer for overflow

//! Seven segment display setup definitions

#define ddrb fill //! Configure port B as output

#define ddrj fill //! Configure Port J as Output

#define ddrp fill //! enable the PTP

#define ptj clear //! enable LEDs

#define E\_CLOCK 24000000

#define SAMPLE\_RATE 160 //! Sample rate in Hz

#define SAMPLE\_DELAY E\_CLOCK/SAMPLE\_RATE //! Sample delay period in clock cycles

#define FILTER\_CUTOFF 50 //! cut off in Hz for high pass filter

#define THRESHOLD 200 //! threshold to check for peak

//! SCI setup definitions

#define BDH clear //! store 0 in the high byte of baud rate

#define BDL bit7|bit4|bit3|bit2 //! set baud rate to 9600

#define SCICR1 clear //! disable everything

#define SCICR2 bit5|bit3|bit2 //! turn on recieve and transmit enable bits

#define helpLength 70

//! ADT setup definitions

#define atd1ctl2 bit7|bit6 //! allowing normal operation and reading from output register will reset system and cause new conversion.

#define atd1ctl3 bit3 //! set control conversion sequence to 1

#define atd1ctl4 bit7|bit6|bit5|bit3|bit1|bit0 //! set resolution to 8 bit and clock prescale to 0, which divides by 2

#define atd1ctl5 bit7|bit1 //! set right allignment, set read to pad 10

#define atd1dien bit2 //! enable input from PAD10

#define ALPHA 34 //! alpha for high pass filter x10^-2

#define ALPHA\_DIVISOR 100 //! brings alpha to its correct value avoiding floats

#define MAX\_HEART\_RATE (SAMPLE\_RATE\*60)/150 //! max healthy heart rate 150bpm

#define MIN\_HEART\_RATE (SAMPLE\_RATE\*60)/40 //! min healthy heart rate 40 bpm

#define OC0F 1 //! flag for Output capture 0 interrupt

#define OC2F 1 << 2 //! flag for Output capture 2 interrupt

#define OC4F 1 << 4 //! flag for Output capture 4 interrupt

#define SPEAKER 1 << 5 //! speaker is 5th bit on PORT T

#define TONE\_TIME 500 //! delay in cycles to keep the speaker on

#define PEAKTIMES\_LENGTH 5 //! number of peaktimes to use to calculate bpm

/\*\* --- End set up definitions ----- \*/

/\*\* ----- Start variable declarations and initialisation ----- \*/

//! Heart Rate variables

int newbpmFlag = resetFlag;

int sample\_count = 0; //! the number that counts for every new ADC value

int unfiltered[2] = {0,0}; //! int array holding the current (1) and previous (0) unfiltered ADC values

int filtered[3] = {0,0,0}; //! int array holding the current (2) previous (1) and the next previous (0) filtered ADC values

int squared\_filt[3] = {0,0,0}; //! the filtered array values squared

#define PEAKTIMES\_LENGTH 5 //! number of peaktimes to use to calculate bpm

int peaktimes[PEAKTIMES\_LENGTH] = {0,0,0,0,0}; //! int array holding the peak times

//! General variables

int bpm = clear; //! Holds bpm

int bpm0 = clear; //! Holds the hundreds column of bpm

int bpm1 = clear; //! Holds the tens column of bpm

int bpm2 = clear; //! Holds the digits column of bpm

int readValue = clear;

//! SCI variables

int readSCIFlag = clear; //! Flag/variable to reset SCI status register

int input = clear; //! Variable to store input from SCI

int helpFlag = resetFlag; //! Flag to indicate that help has been requested

int calibrateFlag = resetFlag; //! Flag to indicate that calibration has been requested

int runFlag = resetFlag; //! Run setting

//! Timer variables

int tc7Count = clear;

int buzzCount = clear;

int buzzFlag = resetFlag;

//! Seven segment variables

unsigned char display[10]={63,6,91,79,102,109,125,7,127,111}; //! Values for 7 seg 0-9

int segment[4] = {0x0E, 0x0D, 0x0B, 0x07}; //! Code to turn on segments of the display

int position = 0; //! Position to display hundreds, tens, ones goes between 0 and 2

const char sevenseg\_LED[] = {0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F}; //! decimal representations on sevenseg LED Port B

const char sevenseg\_digit[] = {0xFD,0xFB,0xF7}; //! digit for seven seg on Port P {digit2,3,4}

char sevenseg\_values[3] = {0x3F,0x3F,0x3F}; //! array to store dec representations for values for port B to display bpm/offset on 7seg

int sevenseg\_count = 0; //! counter used during 7seg timer interrupt

//! Calibrate Variables

int offset = clear;

int offset0 = clear;

int offset1 = clear;

//! Help variables

/\*\* ----- Stop variable declarations ----- \*/

//! Function prototypes

void init(void);

void Setup(void);

void displayHelp(char helpMessage[70]);

void calibrate(void);

void delay\_3sec(void);

void heart\_rate(void);

void array\_shift(int\* array, int length);

int calculate\_bpm(int\* peaktimes);

void tone(void);

void seven\_seg(void);

interrupt 8 void TC0\_ISR(void);

interrupt 9 void TC1\_ISR(void);

interrupt 14 void TC6\_ISR(void);

interrupt 15 void TC7\_ISR(void);

interrupt 21 void SCI\_ISR(void);

interrupt 13 void TC5\_ISR(void);

char helpMessage[70] = {'\n','\r','H','e','l','p',' ','p','a','g','e',

'\n','\r','P','r','e','s','s',':','\t','h',' ','f','o','r',' ','h','e','l','p',

'\n','\r','\t','c',' ','t','o',' ','c','a','l','i','b','r','a','t','e',

'\n','\r','\t','s',' ','t','o',' ','s','t','a','r','t',' ','o','r',' ','s','t','o','p','\n','\r'};

void main(){

Setup();

displayHelp(helpMessage); //! Display help at start

while(1){

//! Main loop of program

while(runFlag){

if (helpFlag){

displayHelp(helpMessage);

helpFlag = resetFlag;

}else if (calibrateFlag){

calibrate();

calibrateFlag = resetFlag;

} else {

heart\_rate();

seven\_seg();

}

}

}

}

/\*\*

Setup function, configures SCI, ADC, 7seg, buzzer and Timer Interrupts.

no inputs/outputs

\*/

void Setup(void){

EnableInterrupts;

//! SCI setup

SCI1BDH = BDH;

SCI1BDL = BDL;

SCI1CR1 = SCICR1;

SCI1CR2 = SCICR2;

//! ADC setup

ATD1CTL2 = atd1ctl2;

ATD1CTL3 = atd1ctl3;

ATD1CTL4 = atd1ctl4;

ATD1DIEN = atd1dien;

ATD1CTL5 = atd1ctl5; //! Trigger the first conversion

//! Timer setup

TSCR1 = tscr1;

TSCR2 = tscr2;

TIOS = tios;

TIE = clear;

TC0 = TCNT + HzAdd160; //! Initiate TC0 with the first trigger time

TC1 = TCNT + HzAdd100; //! Initiate TC1 with the first trigger time

TC5 = TCNT + HzAdd4000; //! Initiate TC1 with the first trigger time

TC6 = TCNT + HzAdd100; //! Initiate TC6 with the first trigger time

TC7 = overflow; //! Initiate TC7 with to trigger when it overflows

//! Enable buzzer

DDRT = DDRT & bit5;

//! Seven segment setup

DDRB = ddrb;

DDRJ = ddrj;

PTJ = ptj;

DDRP = ddrp;

PORTB = clear; //! Turn all segments off

DDRT = 0xFF; //! set the speaker Port T bit 5 to output

DDRM = 0xFF;

EnableInterrupts;

}

/\*\*

Help function, transfers a help message over the SCI to the user.

@param strng takes help meassage as string

\*/

void displayHelp(char helpMessage[70]){

int helpCount = 0;

while(helpCount < 70){

if(SCI1SR1 & bit7){ //! poll to see if ready to transmit

SCI1DRL = helpMessage[helpCount]; //! transmit one character in help page

helpCount = helpCount + 1; //! increment counter

}

}

}

/\*\*

calibrate function, calculates the voltage offset and starts the 7seg.

disables 7seg after 3 second delay done by using interrupts.

no input

outputs offset to global variable.

\*/

void calibrate(void){

int value0 = 0;

int value1 = 0;

int dex = 0;

PTJ = clear;

PORTB = fill;

//! Stop Timer interrupts

TIE = clear;

//! Trigger calibration

PTM = PTM & ~bit7;

ATD1CTL5 = atd1ctl5; //! Trigger the first conversion

//! Loop until readings level

while(dex < 10000){

if(ATD1STAT0 & bit7){

value1 = value0;

value0 = ATD1DR0L; //! Read value read location can be ATD1DR0L

ATD1CTL5 = atd1ctl5; //! Trigger the next conversion

dex++;

}

}

//! Stop PM 7

PTM = PTM | bit7;

//! Calculate offset from 0 - 5 volts from 0 - 255 value.

offset = (value0\*50)/255;

offset0 = offset/10;

offset1 = offset%10;

//! Enable tc6 and tc7 interrupts

TIE = tieCalibrate;

//! While calibration flag is active(~ 3s) display offset

while(calibrateFlag);

//! Disable tc6 and tc7 while enabling tc0 and tc1

TIE = tieRegular;

}

/\*\*

This function plays the buzzer for 0.5 seconds

no inputs/outputs

\*/

void tone(void){

TIE = TIE | bit5;

TC5 = TCNT + HzAdd4000; //! Set the next trigger time.

}

/\*\*

function heart\_rate

takes an ADC value one at a time and calculates a new bpm every peak

Uses a high pass filter on the ADC values, squares the filtered values

then finds the peaks using 3 points with the middle point being greater than the other two and above a certain threshold

sub funtions:

1.calculate\_bpm

2.array\_shift

3.tone

\*/

void heart\_rate(void){

int i = 0;

int value = 0;

value = readValue;

unfiltered[1] = value; //! current unfiltered value equals new ADC value

//! make sure the following filter will give a positvie value

if (unfiltered[1]>unfiltered[0]){

filtered[2] = (ALPHA\*(filtered[2-1]+unfiltered[1]-unfiltered[1-1]))/ALPHA\_DIVISOR; //! high pass filter

} else {

filtered[2] = 0; //! if would be negative value become 0 instead

}

squared\_filt[2] = filtered[2]\*filtered[2]; //! square current filtered value

/\*find peak

if previous value <peak value > next value, peak value > threshold

and there is a reasonable gap in time before next peak in this case < 150 bpm(skip noise on peak) \*\*/

if ((squared\_filt[0]<squared\_filt[1]) & (squared\_filt[1]>squared\_filt[2]) & (squared\_filt[1]>THRESHOLD) & (sample\_count > peaktimes[PEAKTIMES\_LENGTH-2] + MAX\_HEART\_RATE)){

peaktimes[PEAKTIMES\_LENGTH-1] = sample\_count; //! current peak time equals sample counter

tone(); //! start speaker

bpm = calculate\_bpm(peaktimes); //! calculate new bpm

newbpmFlag = setFlag;

array\_shift(peaktimes,PEAKTIMES\_LENGTH); //! shift peaktimes array to the left so current value becomes previous

} else if (sample\_count > peaktimes[PEAKTIMES\_LENGTH-2] + MIN\_HEART\_RATE){ //! if no peak for set amount of time set bpm to 0

bpm = 0;

for(i =0;i<PEAKTIMES\_LENGTH-1;i++){ //! set all peaktimes to 0

peaktimes[i] = 0;

}

newbpmFlag = setFlag;

}

array\_shift(squared\_filt,3);//! shift array to left so current value becomes previous value

array\_shift(filtered,3);

array\_shift(unfiltered,2);

}

/\*\*

funtion array-shift

takes an address to first value of an array and array length

shifts all the values one space to the left and sets the last value to 0

@param int\* array takes a pointer to the start of an array to shift

@param int length - int defining length of the array

\*/

void array\_shift(int\* array, int length){

int i =0;

for(i = 0;i<length;i++){

if(i < (length-1)){

array[i] = array[i+1]; //!sets current value to equal next value

}

else{

array[i] = 0; //!sets last value to 0

}

}

}

/\*\*

funtion calcuate\_bpm

calculates a new bpm taking an array containing a set number of peaktimes

takes the average of the difference between following times (in sample counts) and converts to bpm

returns new bpm value

@param int\* peaktimes - a pointer to the start of array peaktimes holding values for times in sample counts of heartbeats

\*/

int calculate\_bpm(int\* peaktimes){

int i = 0;

int j = 0;

int new\_bpm = 0;

int time\_dif = 0;

for (i=0;i<PEAKTIMES\_LENGTH-1;i++){

if((peaktimes[i] > 0) & (peaktimes[i+1] > 0)){ //! make sure there will be a value

time\_dif = time\_dif + peaktimes[i+1] - peaktimes[i]; //! calculate and total time dif for following peak times

j++; //! count how many time intervals between peaks

}

}

time\_dif = time\_dif/j; //! calculate average time difference

new\_bpm = (60\*time\_dif)/SAMPLE\_RATE; //! convert from sample counts to bpm

return new\_bpm;

}

/\*\*

this function separates the value of bpm into digits whenever there is a new value

then starts the 7seg timer interrupt

\*/

void seven\_seg(void){

if((newbpmFlag == 1)){

bpm0 = bpm/100; //! Find the hundreds column

bpm1 = (bpm%100)/10; //! Find the tens column

bpm2 = (bpm%10); //! Find the ones column

TIE = tieRegular;

}

}

/\*\* ----- Start interrupt declaration and definitions ----- \*/

//! SCI interrupt

interrupt 21 void SCI\_ISR(void){ //! SCI ISR

readSCIFlag = SCI1SR1; //! Set flag to the serial register (to reset flag)

input = SCI1DRL; //! Get input from SCI 1 lower data register

if (input == 'h'|| input == 'H') { //! check if help has been requested

helpFlag = setFlag; //! Set flag to write help message

runFlag = setFlag;

} else if(input == 'c' || input == 'C'){ //! Check if recalibration requested

calibrateFlag = setFlag; //! Set flag to run calibration routine

runFlag = setFlag;

} else if(input == 's' || input == 'S'){ //! Check if stop/start requested

if (runFlag == setFlag){

runFlag = resetFlag;

TIE = clear;

}else{

runFlag = setFlag;

TIE = tieRegular;

}

} else {

tone();

}

}

//! Timer interrupts

interrupt 8 void TC0\_ISR(void){

//! This interrupt will tell the program when to get readings from the electrodes at 160Hz

//! Set next trigger time

TC0 = TCNT + HzAdd160;

//! Get converted value from previous conversion trigger and thus trigger the next conversion

readValue = ATD1DR0L; //! note: input register could be ATD1DRL0

ATD1CTL5 = atd1ctl5; //! Trigger the next conversion

sample\_count++;

}

/\*\* This interrupt will change the character being displayed and set the time for the next interrupt

The changes will happen at ~100HZ

\*/

interrupt 9 void TC1\_ISR(void){

//! Set next trigger time

TC1 = TCNT + HzAdd100;

//! Turn off seven segment display

PTP = fill;

//! Set or reset segment position

position = position + 1;

if (position > 2) position = 0;

//! Turn on the right value for display

if(position == 0){

PORTB = display[bpm2];

}

else if (position == 1){

PORTB = display[bpm1];

}

else if (position == 2){

PORTB = display[bpm0];

}

else{

position = 0;

PORTB = clear;

}

//! Turn on 7 segment display

PTP = segment[position];

}

//! This interrupt will toggle the value of TCTL1 bits 3 and 2 to create buzzer

interrupt 13 void TC5\_ISR(void){

//! Toggle TCTL and buzz flag

if (buzzFlag == resetFlag){

buzzFlag = setFlag;

TCTL1 = bit3|bit2; //! Set pt5 high

}else if (buzzFlag == setFlag){

buzzFlag = resetFlag;

TCTL1 = bit3; //! Set pt5 low

}

//! Increment buzz count

buzzCount++;

//! If buzz count is 2000 stop interrupt to stop buzzer

if (buzzCount >= 2000){

TIE = TIE & ~bit5;

}

}

/\*\* This interrupt will change the character being displayed to display calibration offset and set the time for the next interrupt

The changes will happen at ~100HZ.

\*/

interrupt 14 void TC6\_ISR(void){

//! Set next trigger time

TC6 = TCNT + HzAdd100;

//! Turn off seven segment display

PTP = fill;

//! Set or reset segment position

position = position + 1;

if (position > 1) position = 0;

//! Turn on the right value for display

if(position == 0) PORTB = display[offset1];

else if (position == 1) PORTB = display[offset0];

else{

position = 0;

PORTB = clear;

}

//! Turn on 7 segment display

PTP = segment[position];

}

/\*\* This interrupt count from 1 to 10 when triggered, on overflow.

Overflow happens approximately every 350ms, therefore this effectively counts to 3.5s

This is used to modulate the time period of displaying the offset voltage

Then this resets the calibrate flag stopping the infinite loop in the calibrate function and allowing it to exit.

\*/

interrupt 15 void TC7\_ISR(void){

tc7Count = tc7Count + 1;

if(tc7Count >= 10){

calibrateFlag = clear;

tc7Count = 0;

}

}

/\*\* ----- Stop interrupt declaration and definitions ----- \*/