Electronic Design Project 2 -

Group P (Pulse Patrol)

# Ross Gallacher, Harvey Leicester, Axel Steinman and

# Alex Smith

# 1.0 Introduction

As a part of our second year team design course, group P (Pulse Patrol) were given a task. The design goals were to create a small 9-volt battery-powered sensor that would display when it is sensing a pulse, and the BPM of said pulse. Further goals were to gain experience working with a team on a project and practice proper report-writing, note-taking, teamwork and to discover how fun engineering can be. The device utilises technology that is largely already familiar. It was designed to measure subtle changes in light when a finger is covering the sensor of the device. The pulses are then counted and a value for the person’s BPM calculated. It is similar to most other pulse meters that have been made by our classmates, in terms of design and engineering.

The design will be slightly different to some commercially available devices.

Wiki

Modern heart rate monitors usually comprise two elements: a chest strap transmitter and a wrist receiver (which usually doubles as a watch) or mobile phone.

Some also making use of accelerometers measuring a persons activity to help analyse a person, ours does not have this complexity but still proved to pose a few issues which were useful in cementing our understanding of these devices and how they fit together.

During this project, data sheets of the following were utilised:

8x8 LED Display breakout board designed by Andrew Phillips

Maxim Integrated Max7221 chip

Newshine Optics’ XH–12088BEG display

National Semiconductor’s LM2937

Microchip MCP6292

NXP 74HCT04

TCRT1010

Along with the essential lectures, advice and help of J. Trinder, D.J. Muir and A. Phillips.

# 2.1 Principle of Operation

In our design for the heart rate monitor, the user places their finger onto a IR LED within a sensor circuit, which takes several readings from the varying light levels present in the user's finger. This is affected by the constantly changing amount of blood in the user's finger that the user's heartbeat sends meaning we are able to sense whenever the user's heart beats.

The signal picked up on the sensor from the user's heartbeat is then amplified and filtered as it was too small of a signal for proper use in the device otherwise and converted to a digital signal to be sent to the mbed through the Freedom board.

Within the mbed the code established allows it to react accordingly to the signals it receives by taking several readings and calculating an overall ratio that leads to the calculation of the user's bpm.

This is then converted to an analogue signal and displayed on 8x8 display in which each number and letter flashes up one at a time to make it easy for the user to read and interpret their bpm.

We chose to use the TCRT 1010 sensor mainly due to its recommendation, this and it's sufficiency for the task at hand bundled with its low cost made it a suitable and practical choice.

# 2.2 Overall Diagram (Need Block Diagram)

Sensor

The circuit for the sensor which can be seen in figure X consists of an IR LED, a transistor and two resistors. The values of the resistors were calculated using the IR LED data sheet and some assumptions about the circuit such as the collector voltage being zero.

Signal Processing

(Need diagrams for this and the rest of 2.2 so will complete in lab with diagrams to refer to)

Micro

" "

Display Blocks

" "

PSU

" "

# 2.3 Sensor

The circuit used for our TCRT1010 sensor seen in Figure Z was designed to receive the pulse of the user through the IR LED, this signal is then amplified and filtered to remove noise and allow a clearer trace on the oscilloscope before it is converted to a digital signal for the mbed.

An example of the trace seen on the oscilloscope can be seen in Figure Y, (Statement about the trace will be added here once I have the trace)

(Paragraph to be added concerning testing once I have the traces)

# 2.4 Amplifiers

The output of the sensor circuit requires amplification to provide a signal that isn't affected by noise and is strong enough for the ADC to receive.

This amplification is achieved from the MCP6292 amplifier, this uses two stages of amplification to achieve the stronger signal required for the rest of the circuit.

(Paragraph about frequencies here need to double check the values I have and why we chose those exact ones for high and low pass)

(Pictures/diagram of amplifier circuit and description)

(Traces showing the amplifier working and comments on it)

**2.5 Micro(pros and cons)**

With the amplification/analogue section of the design being chosen a microprocessor was required to take the pulses from the analogue side and then calculate the BPM value and output that along an SPI output to an LED. In this design the FRDM-KL25Z board was chosen. There were multiple reasons why this board was chosen as outlined below:

* This board has support for the mbed compiler which makes development a little simpler as it can be coded and compiled in the browser and then dragged on via USB to the board
* The board supports SPI which was required to send data to the LED’s to create the output in a way that a user can easily understand

# 2.6 Display

The display chosen was the Newshine Optics’ XH–12088BEG display. There were a few reasons this display was chosen. Firstly it was chosen because it is a very cheap component and costs of a design are an important consideration in any project. It was chosen due to it acting like individual LEDs which brought up some issues that had to be addressed in outputting a value. This helped solidify some of the fundamental c programming concepts.

# 2.7 PSU

The main goals of this project was to produce a portable, lively pulse meter. To keep this device portable a battery was the best option. The battery will last for ????? which is good enough to demo a working prototype of our design. A voltage regulator (LM2937) was required so when the battery is running down the voltage to the components stays the same until the battery is further depleted.

# 3.0 Software Design

### Pulse measuring algorithm

After some experimentation we found that a count based, trigger level system was the most accurate for calculating the total BPM. The main algorithm times how long it takes for the ADC to receive 8 signals above a set threshold level, and derives the total BPM from this time. The threshold is set relatively low at approximately 1V. This level was chosen as it can still detect when there is a pulse, but also 'ignore' the lower level, second peak that is characteristic of a pulse waveform. It also allows for fluctuations in the signal coming from the sensor as instead of having a high level trigger, where the pulse signal might not reach, it will almost always pass through the lower voltage level of 1V.

### Program Flow

With a light based sensor, it is always picking up a signal from background light levels. To stop the microcontroller from trying to calculate the BPM from this input, we put in a 'scan' loop. This is a function that does nothing but display the word 'waiting' until the user puts their finger on the sensor. When the sensor is first touched, the signal's voltage drops to approximately 0V. Thus with a simple 'do while' loop with the condition to check whether the ADC level was below 0.066V we fixed the issue of background light level effecting the calculation.

Once the user's finger is on the sensor, the program waits for a pulse. This is done using the 'measure' function. Again, this is a simple 'do while' loop in which the program waits for a pulse (ADC>1V), starts a timer, and counts how many times the input signal is greater than this level. Once there have been 8 pulses the timer stops, is read and returned by the function. The only difficulty in the trigger based system came when the program would count every instance where the input signal was above the threshold. This was solved by using a variable 'peak' to go high at a pulse and low at no pulse. This was then used within another 'do while' loop where the program checks if the signal is below the threshold (ADC<1V) for as long as there has been a pulse (peak=1). This effectively gets the program stuck in a loop checking if the input is below the threshold after every pulse is registered.

When the sample time has been returned to the main of the program, it is used as the argument for the 'calc\_and\_display' function. This again is a simple function that takes the sample time and works out its ratio to 1 minute. It then calculates the total BPM by multiplying the number of pulses (8) with this ratio and displays the 100's, 10's and units of this value. Splitting the total BPM into it's individual numbers is dine using simple operators (/ and %). I will explain how values are displayed on the LED matrix in the next section.

### Driving the Display

Using hexadecimal numbers, LED's can be set to be on or off. E.G. 10000001 = 0x81 = end left and right LED's on. Arrays are declared with the combination of hexadecimal values which can be selected to display characters on the screen, e.g. char numb\_array[10][8] = {{0x00, 0x38, 0x28, 0x28, 0x28, 0x38, 0x00, 0x00},{0x00, 0x08, 0x08...}} contains in it's rows the the information to display 0-9. Likewise 'bpm\_array[3][8]' contains the characters 'b', 'p' and 'm'. Each array used 8-bit numbers and 8 columns as the display is sized 8x8. Using Dr Trinder's template for setting up the Freedom board (SPI and read/write preferences) and his function 'pattern\_to\_display' it is relatively simple to display characters from these arrays.

In the 'calc\_and\_display' function, empty arrays are filled up with a selected row from the arrays containing multiple characters. This is done using 'for loops' to cycle through each column of the selected row, and fill the empty array with the column values. The program then uses the 'pattern\_to\_display' function with this new array as it's argument to turn the desired LED's on. E.g. to display the 100's of the total bpm, the calculation is done in the row position of the array containing the numbers 0-9. Lets say this equals 1. The empty array will then fill up with the information to display '1' on the screen by setting each one of its columns to the corresponding column of the array containing numbers. Once the information has been transferred to the new array, it can then be used with the 'pattern\_to\_display' function to drive the display to show the number '1'.

# 4.0 PCB Design

Our main considerations when designing the PCBs were to have a logical layout, decoupling capacitors near the chips and to have the sensor on the edge of the board. Our view of a logical layout was to have the analogue side of the board to be on the left, FRD board in the middle and the LED array on the right. This would mean input, then calculation and then output. The board was also routed on both sides in order to improve the spacing of tracks to make for easier and more reliable hand soldered connections.

# 6.0 Appendix

### Circuit Diagram

### Code

#include "mbed.h"

/\*

example of driving maxim chip for Glasgow Uni Projects.

Dr J.J.Trinder 2013,14

jon.trinder@glasgow.ac.uk

\*/

#include <stdio.h>

#include <stdlib.h> // extra includes!

#include <time.h>

/\* Get a random number from 0 to 0.9999999

(you don't need to understand this function)

\*\*\*\*\* DON'T MODIFY THIS FUNCTION \*\*\*\*\*

\*/

#define max7219\_reg\_noop 0x00

#define max7219\_reg\_digit0 0x01

#define max7219\_reg\_digit1 0x02

#define max7219\_reg\_digit2 0x03

#define max7219\_reg\_digit3 0x04

#define max7219\_reg\_digit4 0x05

#define max7219\_reg\_digit5 0x06

#define max7219\_reg\_digit6 0x07

#define max7219\_reg\_digit7 0x08

#define max7219\_reg\_decodeMode 0x09

#define max7219\_reg\_intensity 0x0a

#define max7219\_reg\_scanLimit 0x0b

#define max7219\_reg\_shutdown 0x0c

#define max7219\_reg\_displayTest 0x0f

#define LOW 0

#define HIGH 1

SPI max72\_spi(PTD2, NC, PTD1);

DigitalOut load(PTD0); //will provide the load signal

AnalogIn ADC(PTB0);

DigitalOut Do(PTD5);

char numb\_array[10][8] = {{0x00, 0x38, 0x28, 0x28, 0x28, 0x38, 0x00, 0x00},{0x00, 0x08, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00},{0x00, 0x38, 0x08, 0x38, 0x20, 0x38, 0x00, 0x00},

{0x00, 0x38, 0x08, 0x38, 0x08, 0x38, 0x00, 0x00},{0x00, 0x28, 0x28, 0x3C, 0x08, 0x08, 0x00, 0x00},{0x00, 0x38, 0x20, 0x38, 0x08, 0x38, 0x00, 0x00},

{0x00, 0x38, 0x20, 0x38, 0x28, 0x38, 0x00, 0x00},{0x00, 0x38, 0x08, 0x08, 0x08, 0x08, 0x00, 0x00},{0x00, 0x38, 0x28, 0x38, 0x28, 0x38, 0x00, 0x00},

{0x00, 0x38, 0x28, 0x38, 0x08, 0x38, 0x00, 0x00}

};

//array for numbers

char bpm\_array[3][8] = {{0x00, 0x10, 0x10, 0x10, 0x1C, 0x14, 0x1C, 0x00},{0x00, 0x1C, 0x14, 0x1C, 0x10, 0x10, 0x10, 0x00},{0x00, 0x22, 0x36, 0x2A, 0x22, 0x22, 0x22, 0x00}};

//array for bpm letters

char waiting\_array[7][8] = {{0x00, 0x00, 0x22, 0x22, 0x2A, 0x36, 0x22, 0x00}, {0x00, 0x00, 0x3E, 0x22, 0x3E, 0x22, 0x22, 0x00}, {0x00, 0x00, 0x08, 0x08, 0x08, 0x08, 0x08, 0x00}, {0x00, 0x00, 0x3E, 0x08, 0x08, 0x08, 0x08, 0x00}, {0x00, 0x00, 0x08, 0x08, 0x08, 0x08, 0x08, 0x00}, {0x00, 0x00, 0x22, 0x32, 0x2A, 0x26, 0x22, 0x00}, {0x00, 0x00, 0x3C, 0x20, 0x2E, 0x24, 0x3C, 0x00}};

//array for waiting letters

char empty\_pattern[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00}; //all lighs off

char big\_heart[8] = {0x00, 0x36, 0x49, 0x41, 0x22, 0x14, 0x08, 0x00};

char small\_heart[8] = {0x00, 0x14, 0x2A, 0x22, 0x14, 0x08, 0x00, 0x00};

/\*

Write to the maxim via SPI

args register and the column data

\*/

void write\_to\_max( int reg, int col)

{

load = LOW; // begin

max72\_spi.write(reg); // specify register

max72\_spi.write(col); // put data

load = HIGH; // make sure data is loaded (on rising edge of LOAD/CS)

}

//writes 8 bytes to the display

void pattern\_to\_display(char \*testdata)

{

int cdata;

for(int idx = 0; idx <= 7; idx++) {

cdata = testdata[idx];

write\_to\_max(idx+1,cdata);

}

}

void setup\_dot\_matrix ()

{

// initiation of the max 7219

// SPI setup: 8 bits, mode 0

max72\_spi.format(8, 0);

max72\_spi.frequency(100000); //down to 100khx easier to scope ;-)

write\_to\_max(max7219\_reg\_scanLimit, 0x07);

write\_to\_max(max7219\_reg\_decodeMode, 0x00); // using an led matrix (not digits)

write\_to\_max(max7219\_reg\_shutdown, 0x01); // not in shutdown mode

write\_to\_max(max7219\_reg\_displayTest, 0x00); // no display test

for (int e=1; e<=8; e++) { // empty registers, turn all LEDs off

write\_to\_max(e,0);

}

// maxAll(max7219\_reg\_intensity, 0x0f & 0x0f); // the first 0x0f is the value you can set

write\_to\_max(max7219\_reg\_intensity, 0x08);

}

void clear()

{

for (int e=1; e<=8; e++) { // empty registers, turn all LEDs off

write\_to\_max(e,0);

}

}

void scan()

{

char display\_wait[8] = {0,0,0,0,0,0,0,0};

do { //when finger isn't on sensor display 'waiting'

for (int x = 0; x<7; x++) { //when finger is on sensor break out of loop

for (int y=0; y<8; y++) {

display\_wait[y] = waiting\_array[x][y];

pattern\_to\_display(display\_wait);

if (ADC<0.02) {

break;

}

wait\_ms(100);

}

if (ADC<0.02) {

break;

}

wait\_ms(100);

}

} while(ADC>0.02);

}

int measure()

{

Timer t;

t.reset();

int count=0; //declare variables

int peak = 0;

do {

if (ADC > 0.31) { //if there is a pulse, start the timer and add to total count and turn all lights on

t.start();

peak = 1; //this allows the program to get stck in the 'do while' loop when there is no pulse. Otherwise the counter will count for every value above the threshold instead of just once

count = count++;

Do = 1; //this was just a test to see if the mbed was registering a pulse

pattern\_to\_display(big\_heart);

}

do {

if (ADC<0.31) {

peak = 0;

}

} while(peak ==1); //will be stuck checking if there is no pulse without adding to the total count

Do=0; //this was just a test to see if the mbed was registering a pulse

pattern\_to\_display(small\_heart); //all lights off for no pulse

} while(count<8); //do this whole thing until there are 8 pulses

t.stop(); //stop the timer

int sample\_time = t.read\_ms(); //read the time in ms

return sample\_time;

}

void calc\_and\_display(int x)

{

int j = 0;

int i = 0;

int z = 0;

char display\_num[8] = {0,0,0,0,0,0,0,0}; //create blank arrays to be able to display stuff

char display\_bpm[8] = {0,0,0,0,0,0,0,0};

float ratio = (60000.0 / x); //find ratio

int total\_bpm = (8 \* ratio); //calculate total bpm

for (z=0; z<2; z++) { //go round cycle twice

for (j=0; j<8; j++) {

display\_num[j] = numb\_array[total\_bpm / 100][j];

pattern\_to\_display(display\_num);

}

wait\_ms(500);

pattern\_to\_display(empty\_pattern); //have a blank screen between numbers to avoid confusion. eg - 111, 88 ect.

wait\_ms(100);

for (j=0; j<8; j++) {

display\_num[j] = numb\_array[(total\_bpm % 100) / 10][j];

pattern\_to\_display(display\_num);

}

wait\_ms(500);

pattern\_to\_display(empty\_pattern);

wait\_ms(100);

for (j=0; j<8; j++) {

display\_num[j] = numb\_array[total\_bpm % 10][j];

pattern\_to\_display(display\_num);

}

wait\_ms(500);

pattern\_to\_display(empty\_pattern);

wait\_ms(100);

for (i=0; i<3; i++) {

for (j=0; j<8; j++) {

display\_bpm[j] = bpm\_array[i][j];

pattern\_to\_display(display\_bpm);

}

wait\_ms(500);

}

}

}

int main() {

setup\_dot\_matrix (); /\* setup matric \*/

int time;

while(1) { //main loop

scan();

time = measure();

calc\_and\_display(time);

clear();

}

}