Heart Rate Meter

VE373 Project Report

- UM-SJTU Joint Institute

Group 7:
Song Shiwei (5123709279)
Wei Yi (5123709286)
Zhang Wenxin (5123709288)

-August 7, 2015

Contents

Introduction

Heart rate is a really important index for the human body, especially for the athletes and the patient. Heart rate meter should be used at everywhere and all the time, for instance, some patients' heart rate should be monitored constantly. Hence, we need a device not only accuracy, but also stable and power saving. That's just what embedded system do. Our team designed a heart rate meter based on PIC32 by applying what we learned in the course VE373 including Interrupt (Timer, Change Notice), Serial Communication (UART) and ADC Module.

Project

Overview

Our system including four main parts, sampling, processing, LCD display and PC display. After we start, the censor module continuously collect the data and transmit them to PIC32 board. They would be converted to digital value by ADC module and then we will either calculate the heart rate or transmit them to PC, to display in different mode.

There are two operation mode, **Simple Mode** and **Realtime Mode (RT Mode)**. The simple mode is to measure the heart rate for 15 seconds, meanwhile the LED twinkling with the heartbeats, and then display the result on LCD screen. The realtime mode is to transmit the realtime voltage value to PC, plot in Matlab and calculate the heart rate in recent 13 seconds.

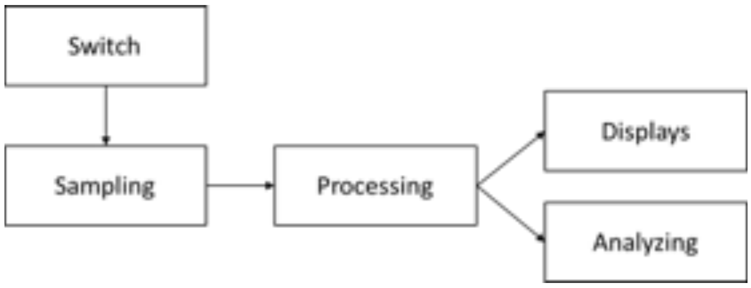

Fig.1 Functional block diagram

We are going to use several components, as shown in the Fig. 2. The IR LED and Optical sensor are integrated on one small PCB with the circuit. The are connected to the extension board of the PIC32. Besides there are button and LED on the extension board that we could use. The LCD is connected as what we do in the Lab 3. Moreover, a protocol converter, PL2303 which could convert RS232-USB, come between the PC and the MCU.

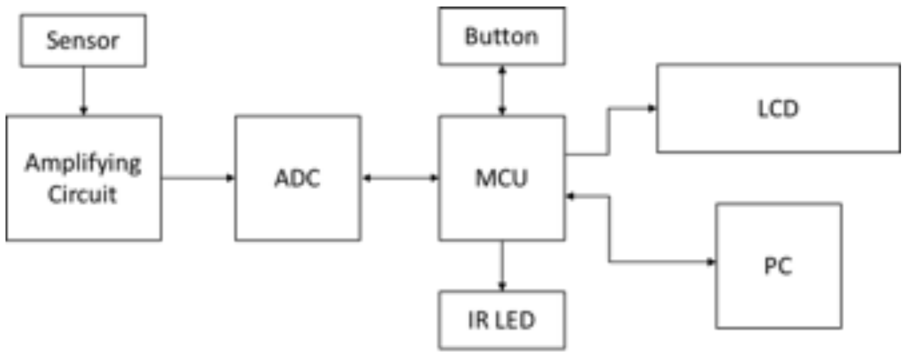

Fig. 2 Component level diagram

Sampling

Along with the heartbeats, the colour on fingertips will vary regularly. In other words, the light absorption of the blood on fingertips could be measured, although it is not that obvious. To get more precise data, we choose to use Infrared LED to illuminate the fingertip and measure the light intensity by an optical sensor.

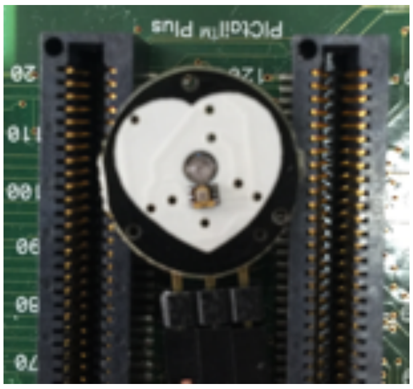

Fig. 3 Integrated Optical Sensor

The sensor has three pins, including Vcc, Ground and Signal. It will return the voltage it measured.

Processing

The first step is to convert the voltage obtained by the sensor to digital value. We use the ADC module integrated on the PIC32 board. The following is our configuration of ADC, set RB2 and CH0 as inputs, use internal voltage as reference voltage and T_AD = 2 * T_PB. And the corresponding timer are configured as prescaler 1:8 and Period Register as 19999 regarding 2 mSec under the PBCLK is 80MHz.

Before it start collect data, there is a process called ”Waiting for Stable”. Since the censor need some time to adjust, we consider continuously obtaining 20 times sample data under 150 is stable enough to proceed to measure.

The next step is to handle the data and figure out the heart rate. In simple mode, our algorithm is that if the voltage varies from above 250 to below 250, we consider it as a pulse. Every time we find a pulse, we turn the LED on. So while measuring, one could see the red light twinkling along with the heartbeats. After 15 seconds, we multiply the number we count by 4, which is considered as the measured heart rate in one minute.

LCD Display

We consider the LCD as a user interface, giving some hints and output result. So we use LCD all the time. As what we practice in Lab 3, we connect the LCD to PIC32 board and configure I/O ports. By the functions, for example `void LCD_puts(const uchar *s), void LCD_goto(uchar addr),` the LCD could display whatever text we want.

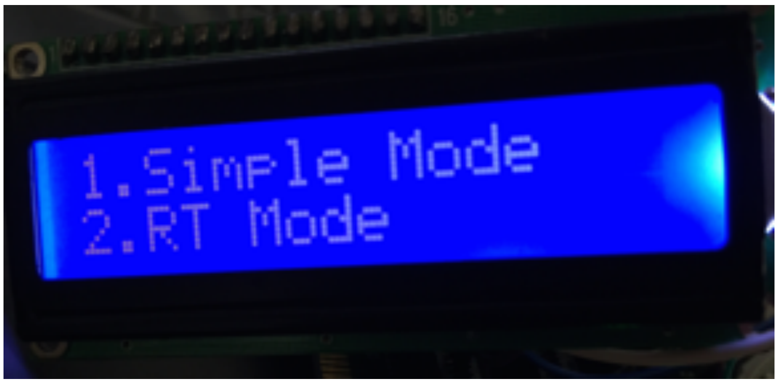

Fig. 4 LCD Display

PC Display

MCU-PC Communication

In our project, we need to communicate between PC and micro-controller, to send data from MCU to PC to display and calculate. We choose UART as the serial communication standard.

On the micro-controller, we use UART1A module. The pins are U1TX and U1RX. The baud rate is set as 9600 Hz and the high-speed mode is selected. When we need to transmit data, we write the data in U1ATXREG and the data will be automatically sent to the computer. On the PC, since we don’t have a RS232 port on our computer, we use a

PL2303 module to transform from RS232 to USB. Then we can use the COM on our computer to receive signals.

We use Matlab to receive and process the signals. We initialize a serial object and configure the correct PORT and baud rate (9600 Hz). Then we can read in data from the object as a stream using the `fread()` function and process the data.

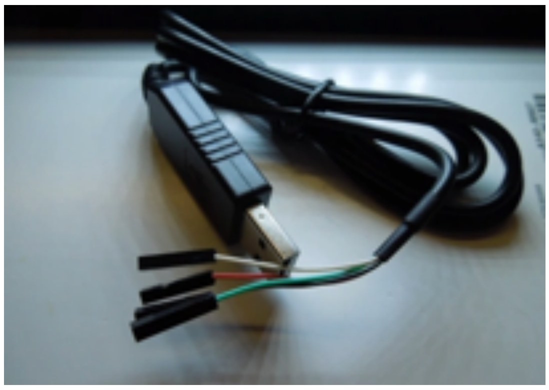
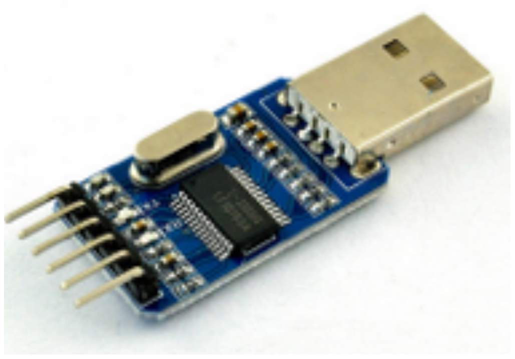

Fig. 5 PL2303 Module

Plot and Calculation

After the data from PIC32 transmitting to the computer, we use the data to plot graph and calculate the heart rate. The data from PIC32 is the voltage value detected by the sensor. We plot the graph for the heart rate. The lateral axis is the number of data, and the vertical axis is the voltage value from the PIC32. We make it by using

```
plot(t(1:length(data)), data, 'r', 'LineWidth',2.5)
```

in Matlab. In real-time mode, it will plot the graph continuously until we choose to stop. Then we need to get the heart rate from the data, we set one high peak and one low peak as one heart jump. We choose to measure 50 data in about 13 seconds which can make the graph better, so we calculate the heart rate in 13 seconds it needs to time 4.6. Thus, we get the heart rate of one people in 1 minute.

Test

We tested two parts of our system, functionality and accuracy. It turns out that our system work well as the pictures shown below. As for accuracy, we compared its result with the heart rate meter on cellphone and the result we count on ourselves. The result is very close.

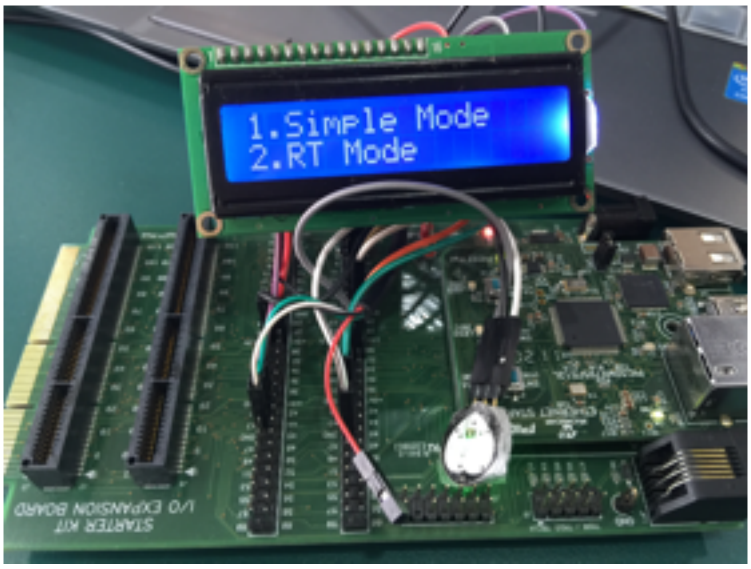

Fig. 6 Choose Mode

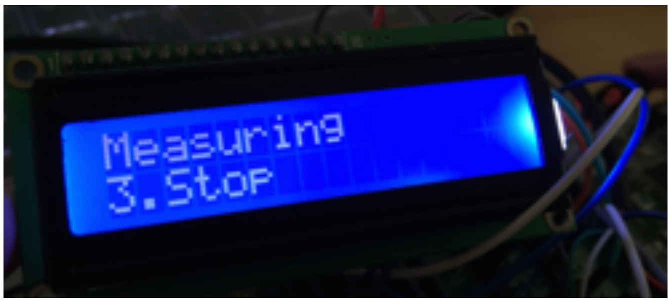

Fig. 7 Measuring in Simple Mode

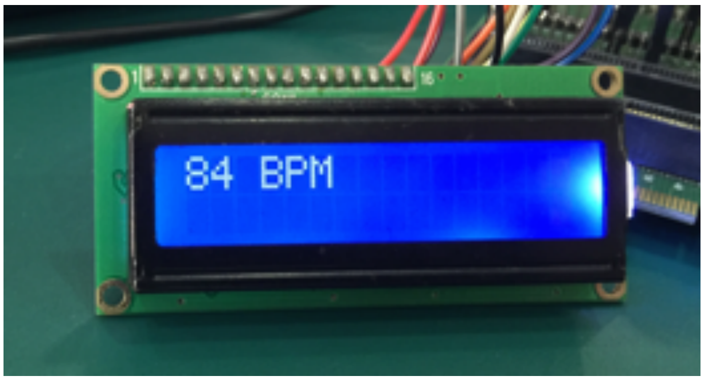

Fig. 8 Result in Simple Mode

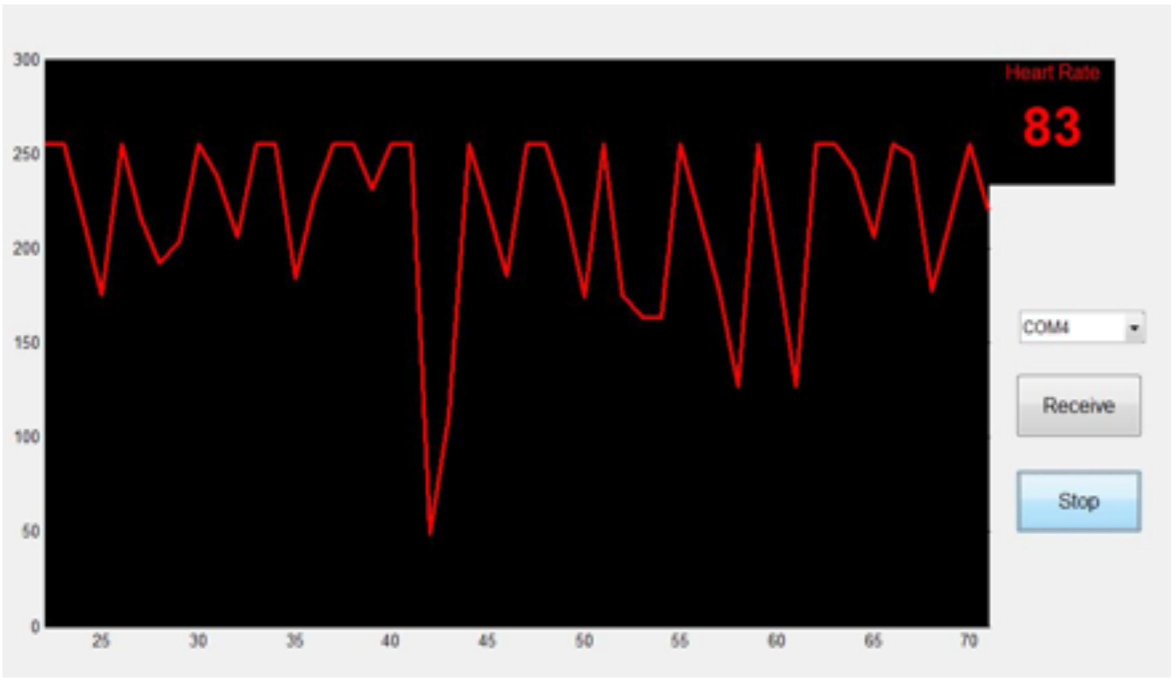

Fig. 9 Result and Graph in RT Mode

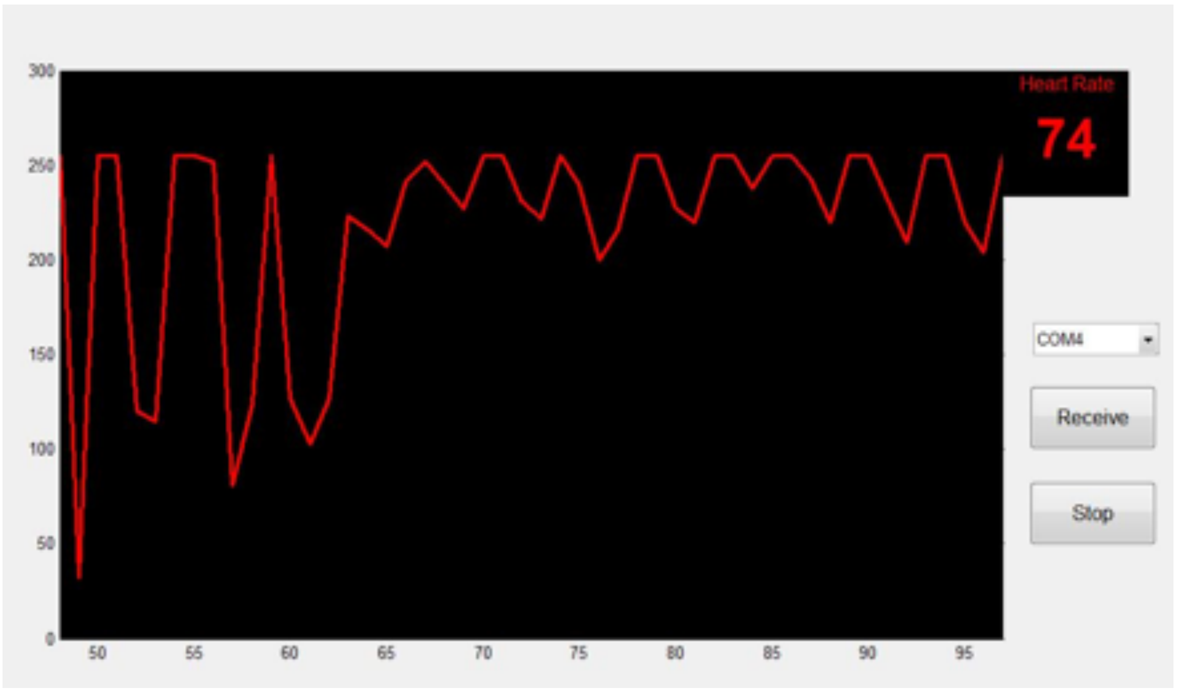

Fig. 10 Result and Graph in RT Mode

Conclusion

All in all, we successfully design and implement the heart rate meter, which could accurately measure the heart rate and plot the graph on PC. We really learned a lot from this project, not only get better understanding to the skills we learned from the course, but also get to know about embedded system. We may proceed studying on embedded system for our career.

Appendix

heartrate.c

```
#include <p32xxxx.h>
#include <plib.h>
#include "LCD.h"

/*Clock Configuration*/
#pragma config FNOSC = PRIPLL              // Oscillator Selection
#pragma config FPLLIDIV = DIV_2      // PLL Input Divider (PIC32 Starter
// Kit: use divide by 2 only)
#pragma config FPLLMUL = MUL_20      // PLL Multiplier
#pragma config FPLLODIV = DIV_1      // PLL Output Divider
#pragma config FPBDIV = DIV_1              // Peripheral Clock divisor
#pragma config FWDTEN = OFF          // Watchdog Timer
#pragma config WDTPS = PS1                 // Watchdog Timer Postscale
#pragma config FCKSM = CSDCMD              // Clock Switching & Fail Safe
// Clock Monitor
#pragma config OSCIOFNC = OFF              // CLKO Enable
#pragma config POSCMOD = XT          // Primary Oscillator
#pragma config IESO = OFF                  // Internal/External Switch-over
#pragma config FSOSCEN = OFF         // Secondary Oscillator Enable
#pragma config CP = OFF                    // Code Protect
#pragma config BWP = OFF                   // Boot Flash Write Protect
#pragma config PWP = OFF                   // Program Flash Write Protect
#pragma config ICESEL = ICS_PGx2     // ICE/ICD Comm Channel Select
#pragma config DEBUG = OFF                 // Debugger Disabled for StarterKit

/*Interrupt ISR*/
#pragma interrupt CN_ISR ipl7 vector 26
#pragma interrupt T5_ISR ipl7 vector 20

/*Global Variable*/
#define RATE 100
#define STABLE_TIME 20
#define BRATE 2000
int ADCValue;
int count;
struct bits {
      unsigned start : 1;
      unsigned stop: 1;
      unsigned mode: 1;
} flags;


/*Function Prototype*/
void T3_init();
void AD1_init();
void CN_init();
void CN_ISR();
void T45_init();
void T5_ISR();
void U1_init();

/*Main Function*/
```

```
int main()
{
	TRISDCLR = 0x1;
	INTCONbits.MVEC = 1;
	asm("ei");
	MCU_init();
	LCD_init();
	LCD_puts("Initialization");
	AD1_init();
	T3_init();
	CN_init();
	T45_init();
	U1_init();
	DelayMsec(1000);
	LCD_clear();
	LCD_puts("1.Simple Mode");
	LCD_goto(0x40);
	LCD_puts("2.RT Mode");
	flags.start = 0;

	while (1){
		while(flags.start == 0);
		LCD_clear();
		T3CONSET = 0x8000;		//start T3
		AD1CON1SET = 0x8004;	//start sample, auto sample
		int i = 0;
		int value = 0;
		uchar str[4];
		count = 0;
		flags.stop = 0;
		if (flags.mode == 0){
			LCD_goto(0x0);
			LCD_puts("Wait for stable");
			int t = 0;
			while(t < STABLE_TIME){			//wait for the signal to
be stable
				while(!IFS1bits.AD1IF);
				ADCValue = ADC1BUF0;
				if (i != RATE){
					i ++;
				}
				else{
					value = ADCValue/2;
					U1ATXREG = value;
					//sprintf(str, "%d", value);
					//LCD_goto(0x40);
					//LCD_puts(str);
					i = 0;
					if (value < 150)
						t = 0;
					else
						t ++;
				}
				IFS1CLR = 0x0002;
			}
			LCD_clear();
			LCD_goto(0x0);
			LCD_puts("Measuring");
		TMR4 = 0;
		T4CONSET = 0x8000;
		while(flags.stop == 0){
			while(!IFS1bits.AD1IF);
			ADCValue = ADC1BUF0;
			if (i != RATE){
				i ++;
			}
			else{
				value = ADCValue/2;
				U1ATXREG = value;
				//sprintf(str, "%d", value);
				//LCD_goto(0x40);
				//LCD_puts(str);
				if (value < 250){
					if (PORTDbits.RD0 == 0)
						count ++;
					PORTDbits.RD0 = 1;
				}
				else
					PORTDbits.RD0 = 0;
				value = 0;
				i = 0;
			}
			IFS1CLR = 0x0002;
		}
		PORTDbits.RD0 = 0;
		LCD_clear();
		LCD_goto(0x0);
		count *= 4;

		sprintf(str, "%d BPM", count);
		LCD_puts(str);
		DelayMsec(2000);
	}
	else{
		LCD_clear();
		LCD_goto(0x0);
		LCD_puts("Measuring");
		LCD_goto(0x40);
		LCD_puts("3.Stop");
		while(flags.stop == 0){
			while(!IFS1bits.AD1IF);
			ADCValue = ADC1BUF0;
			if (i != RATE){
				i ++;
			}
			else{
				value = ADCValue/2;
				U1ATXREG = value;
				if (value < 250){
					if (PORTDbits.RD0 == 0)
						count ++;
					PORTDbits.RD0 = 1;
				}
				else
					PORTDbits.RD0 = 0;
				value = 0;
```

```
                              i = 0;
                        }
                        IFS1CLR = 0x0002;
                  }
                  PORTDbits.RD0 = 0;
            }
            LCD_clear();
            LCD_puts("1.Simple Mode");
            LCD_goto(0x40);
            LCD_puts("2.RT Mode");
            flags.start = 0;
      }
}

/*Function Definition*/
void T3_init()
{
      T3CON = 0;
      T3CONSET = 0x30;  //prescaler 1:8
      TMR3 = 0;
      PR3 = 19999;            //period = 2ms under 80MHz
}

void AD1_init()
{
      AD1PCFG = 0xfffb; //configure AN2/RB2 as analog
      TRISB = 0x4;            //configure RB2 as input
      AD1CHS = 0x20000; //configure RB2 as CH0 input
      AD1CON1 = 0x40;         //T3 start conversion
      AD1CON2 = 0;            //use internal V as reference voltage
      AD1CON3 = 0;            //TAD = 2 * TPB
      AD1CSSL = 0;
}

void CN_init()
{
      TRISDbits.TRISD6 = 1;   //configure RD6 as input
      TRISDbits.TRISD7 = 1;   //configure RD7 as input
      TRISDbits.TRISD13 = 1;  //configure RD13 as input
      asm("di");
      CNCON = 0x8000;               //enable CN module
      CNEN = 0x98000;               //enable CN15, CN16, CN19
      CNPUE = 0x98000;
      IPC6SET = 0x1c0000;
      IFS1CLR = 0x01;               //clear interrupt flag
      IEC1SET = 0x01;
      asm("ei");
}

void CN_ISR()
{
      IEC1CLR = 0x01;
      if (PORTDbits.RD6 == 0){
            flags.start = 1;
            flags.mode = 0;
      }
      else if (PORTDbits.RD7 == 0){
            flags.start = 1;
            flags.mode = 1;
      }
      else if (PORTDbits.RD13 == 0){
            flags.stop = 1;
      }
      IEC1SET = 0x01;
      IFS1CLR = 0x01;
}

void T45_init()
{
      T4CON = 0;
      T5CON = 0;
      T4CONSET = 0x0078;      //32 bit timer, 1:256
      TMR4 = 0;
      PR4 = 4687499;          //15s

      IPC5SET = 0x001c;
      IEC0SET = 0x100000;
      IFS0CLR = 0x100000;
}

void T5_ISR()
{
      IFS0CLR = 0x100000;
      T4CONCLR = 0x8000;
      flags.stop = 1;
}

void U1_init()
{
      TRISF = 0x4;      //RF2 input, RF3 output
      U1ABRG = BRATE;
      U1AMODE = 0x8008;
      U1ASTA = 0x400;
}
```

gui.m

```
function varargout = gui(varargin)
% GUI MATLAB code for gui.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%      the existing singleton*.
%
%      GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI.M with the given input arguments.
%
%      GUI('Property','Value',...) creates a new GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to gui_OpeningFcn via varargin.
```

```
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui

% Last Modified by GUIDE v2.5 01-Aug-2015 00:19:15

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui (see VARARGIN)

% Choose default command line output for gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents
as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
global COM;
COM = 1;
val = get(hObject, 'value');
switch val
    case 1
        COM = 1;
    case 2
        COM = 2;
    case 3
        COM = 3;
    case 4
        COM = 4;
    case 5
        COM = 5;
end

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global s;
global stop;
global COM;
stop = 0;
switch COM
    case 1
        s = serial('COM1');
    case 2
        s = serial('COM2');
    case 3
        s = serial('COM3');
    case 4
        s = serial('COM4');
    case 5
        s = serial('COM5');
end
set(s, 'BaudRate', 9600);
fopen(s);
size = 50;
t = linspace(1,size,size);
data = [];
flag = 0;
i = 1;
rate=0;
hrate=0;
while (stop == 0)
    x = fread(s, 1);
    a = num2str(hrate);
    set(handles.text2, 'String', a);
    if flag == 0
        data(i) = x;
        i = i + 1;
    else
        data = [data(2:size), x];
        for j=1:size-1
            if data(j)<data(j+1)
                rate=rate+1;
            end
        end
        hrate=round(rate*4.6);
        rate=0;
        t = t + 1;
       % if hrate>100
        %    hrate=hrate-20;
        %end
    end
    if i == size + 1
        flag = 1;
    end

    plot(t(1:length(data)), data, 'r', 'LineWidth',2.5);
    set(gca, 'color', [0 0 0]);
    axis([t(1) t(size) 0 300]);
    guidata(hObject, handles);
    pause(0.01);
end
fclose(s);
delete(s);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global stop;
stop = 1;
```

LCD.c

```
/*************************************************************************
* LCD.c
*************************************************************************/
#include "LCD.h"
#include <plib.h>

/* initialize the PIC32 MCU */
void MCU_init() {
/* setup I/O ports to connect to the LCD module */
      TRISC = 0x0;
      TRISE = 0x0;
}
/* initialize the LCD module */
void LCD_init() {
      DelayMsec(15); //wait for 15 ms
      RS = 0; //send command
      Data = LCD_IDLE; //function set - 8 bit interface
      DelayMsec(5); //wait for 5 ms
      Data = LCD_IDLE; //function set - 8 bit interface
      DelayUsec(100); //wait for 100 us
      Data = LCD_IDLE; //function set
      DelayMsec(5);
      Data = LCD_IDLE;
      DelayUsec(100);
      LCD_putchar(LCD_2_LINE_4_BITS);
      DelayUsec(40);
      LCD_putchar(LCD_DSP_CSR);
      DelayUsec(40);
      LCD_putchar(LCD_CLR_DSP);
      DelayMsec(5);
      LCD_putchar(LCD_CSR_INC);
      DelayMsec(5);
}
void LCD_clear() {
      uchar c = 1;
      RS = 0;
      LCD_putchar(c);
      DelayMsec(1);
}
/* Send one byte c (instruction or data) to the LCD */
void LCD_putchar(uchar c) {
      E = 1;
      Data = c; //sending higher nibble
      E = 0; //producing falling edge on E
      E = 1;
      Data <<= 4; //sending lower nibble through higher 4 ports
      E = 0; //producing falling edge on E
}
/* Display a string of characters *s by continuously calling LCD_putchar() */
void LCD_puts(const uchar *s) {
      uchar i = 0;
      RS = 1;
      while (s[i] != '\0'){
            LCD_putchar(s[i]);
            i ++;
            DelayUsec(40);
      }
}
```

```
/* go to a specific DDRAM address addr */
void LCD_goto(uchar addr) {
      RS = 0;
      LCD_putchar(addr + 128);
      DelayUsec(40);
}
/* configure timer SFRs to generate num us delay*/
void DelayUsec(int num) {
      IFS0CLR = 0x100;

      T2CON = 0x0;
      T2CONSET = 0x0030;      //1:8
      TMR2 = 0x0;
      PR2 = num * 20;
      T2CONSET = 0x8000;
      while (IFS0bits.T2IF == 0);

}
/* configure timer SFRs to generate 1 ms delay*/
void GenMsec() {
      DelayUsec(250);
      DelayUsec(250);
      DelayUsec(250);
      DelayUsec(250);
}
/* Call GenMsec() num times to generate num ms delay*/
void DelayMsec(int num) {
      int i;
      for (i=0; i<num; i++) {
            GenMsec();
      }
}

/************end of LCD.c*************/
```

LCD.h

```
/***********************************************************************
* LCD.h
* Header file for the LCD Driver
***********************************************************************/
#include <p32xxxx.h>
// define macros for LCD instructions
#define LCD_IDLE 0x33
#define LCD_2_LINE_4_BITS 0x28
#define LCD_2_LINE_8_BITS 0x38
#define LCD_DSP_CSR 0x0c
#define LCD_CLR_DSP 0x01
#define LCD_CSR_INC 0x06
#define LCD_SFT_MOV 0x14
// define macros for interfacing ports
#define RS PORTCbits.RC1
#define E PORTCbits.RC3
#define Data PORTE
typedef unsigned char uchar;
```

```
/* Function prototypes */
void MCU_init(void);
void LCD_init(void);
void LCD_clear(void);
void LCD_putchar(uchar c);
void LCD_puts(const uchar *s);
void LCD_goto(uchar addr);
void GenMsec(void);
void DelayUsec(int num);
void DelayMsec(int num);
/*****************end of LCD.h*********************/
```