#include "LiquidCrystal.h"

#include <stdio.h>

#include <string.h>

#include <inttypes.h>

#include "Arduino.h"

#ifndef LiquidCrystal\_h

#define LiquidCrystal\_h

#include <inttypes.h>

#include "Print.h"

// commands

#define LCD\_CLEARDISPLAY 0x01

#define LCD\_RETURNHOME 0x02

#define LCD\_ENTRYMODESET 0x04

#define LCD\_DISPLAYCONTROL 0x08

#define LCD\_CURSORSHIFT 0x10

#define LCD\_FUNCTIONSET 0x20

#define LCD\_SETCGRAMADDR 0x40

#define LCD\_SETDDRAMADDR 0x80

// flags for display entry mode

#define LCD\_ENTRYRIGHT 0x00

#define LCD\_ENTRYLEFT 0x02

#define LCD\_ENTRYSHIFTINCREMENT 0x01

#define LCD\_ENTRYSHIFTDECREMENT 0x00

// flags for display on/off control

#define LCD\_DISPLAYON 0x04

#define LCD\_DISPLAYOFF 0x00

#define LCD\_CURSORON 0x02

#define LCD\_CURSOROFF 0x00

#define LCD\_BLINKON 0x01

#define LCD\_BLINKOFF 0x00

// flags for display/cursor shift

#define LCD\_DISPLAYMOVE 0x08

#define LCD\_CURSORMOVE 0x00

#define LCD\_MOVERIGHT 0x04

#define LCD\_MOVELEFT 0x00

// flags for function set

#define LCD\_8BITMODE 0x10

#define LCD\_4BITMODE 0x00

#define LCD\_2LINE 0x08

#define LCD\_1LINE 0x00

#define LCD\_5x10DOTS 0x04

#define LCD\_5x8DOTS 0x00

class LiquidCrystal : public Print {

public:

LiquidCrystal(uint8\_t rs, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3,

uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7);

LiquidCrystal(uint8\_t rs, uint8\_t rw, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3,

uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7);

LiquidCrystal(uint8\_t rs, uint8\_t rw, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3);

LiquidCrystal(uint8\_t rs, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3);

void init(uint8\_t fourbitmode, uint8\_t rs, uint8\_t rw, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3,

uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7);

void begin(uint8\_t cols, uint8\_t rows, uint8\_t charsize = LCD\_5x8DOTS);

void clear();

void home();

void noDisplay();

void display();

void noBlink();

void blink();

void noCursor();

void cursor();

void scrollDisplayLeft();

void scrollDisplayRight();

void leftToRight();

void rightToLeft();

void autoscroll();

void noAutoscroll();

void createChar(uint8\_t, uint8\_t[]);

void setCursor(uint8\_t, uint8\_t);

virtual size\_t write(uint8\_t);

void command(uint8\_t);

using Print::write;

private:

void send(uint8\_t, uint8\_t);

void write4bits(uint8\_t);

void write8bits(uint8\_t);

void pulseEnable();

uint8\_t \_rs\_pin; // LOW: command. HIGH: character.

uint8\_t \_rw\_pin; // LOW: write to LCD. HIGH: read from LCD.

uint8\_t \_enable\_pin; // activated by a HIGH pulse.

uint8\_t \_data\_pins[8];

uint8\_t \_displayfunction;

uint8\_t \_displaycontrol;

uint8\_t \_displaymode;

uint8\_t \_initialized;

uint8\_t \_numlines,\_currline;

};

#endif

#include <LiquidCrystal.h>

// Variables

int pulsePin = 0; // Pulse Sensor purple wire connected to analog pin 0

int blinkPin = 13; // pin to blink led at each beat

int fadePin = 8; // pin to do fancy classy fading blink at each beat

int fadeRate = 0; // used to fade LED on with PWM on fadePin

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

// Volatile Variables, used in the interrupt service routine!

volatile int BPM; // int that holds raw Analog in 0. updated every 2mS

volatile int Signal; // holds the incoming raw data

volatile int IBI = 600; // int that holds the time interval between beats! Must be seeded!

volatile boolean Pulse = false; // "True" when User's live heartbeat is detected. "False" when not a "live beat".

volatile boolean QS = false; // becomes true when Arduoino finds a beat.

// Regards Serial OutPut -- Set This Up to your needs

static boolean serialVisual = true; // Set to 'false' by Default. Re-set to 'true' to see Arduino Serial Monitor ASCII Visual Pulse

volatile int rate[10]; // array to hold last ten IBI values

volatile unsigned long sampleCounter = 0; // used to determine pulse timing

volatile unsigned long lastBeatTime = 0; // used to find IBI

volatile int P = 512; // used to find peak in pulse wave, seeded

volatile int T = 512; // used to find trough in pulse wave, seeded

volatile int thresh = 525; // used to find instant moment of heart beat, seeded

volatile int amp = 100; // used to hold amplitude of pulse waveform, seeded

volatile boolean firstBeat = true; // used to seed rate array so we startup with reasonable BPM

volatile boolean secondBeat = false; // used to seed rate array so we startup with reasonable BPM

void setup()

{

pinMode(blinkPin,OUTPUT); // pin that will blink to your heartbeat!

pinMode(fadePin,OUTPUT); // pin that will fade to your heartbeat!

Serial.begin(115200); // we agree to talk fast!

interruptSetup(); // sets up to read Pulse Sensor signal every 2mS

// IF YOU ARE POWERING The Pulse Sensor AT VOLTAGE LESS THAN THE BOARD VOLTAGE,

// UN-COMMENT THE NEXT LINE AND APPLY THAT VOLTAGE TO THE A-REF PIN

// analogReference(EXTERNAL);

}

// Where the Magic Happens

void loop()

{

serialOutput();

if (QS == true) // A Heartbeat Was Found

{

// BPM and IBI have been Determined

// Quantified Self "QS" true when arduino finds a heartbeat

fadeRate = 255; // Makes the LED Fade Effect Happen, Set 'fadeRate' Variable to 255 to fade LED with pulse

serialOutputWhenBeatHappens(); // A Beat Happened, Output that to serial.

QS = false; // reset the Quantified Self flag for next time

}

ledFadeToBeat(); // Makes the LED Fade Effect Happen

delay(20); // take a break

}

void ledFadeToBeat()

{

fadeRate -= 15; // set LED fade value

fadeRate = constrain(fadeRate,0,255); // keep LED fade value from going into negative numbers!

analogWrite(fadePin,fadeRate); // fade LED

}

void interruptSetup()

{

// Initializes Timer2 to throw an interrupt every 2mS.

TCCR2A = 0x02; // DISABLE PWM ON DIGITAL PINS 3 AND 11, AND GO INTO CTC MODE

TCCR2B = 0x06; // DON'T FORCE COMPARE, 256 PRESCALER

OCR2A = 0X7C; // SET THE TOP OF THE COUNT TO 124 FOR 500Hz SAMPLE RATE

TIMSK2 = 0x02; // ENABLE INTERRUPT ON MATCH BETWEEN TIMER2 AND OCR2A

sei(); // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED

}

void serialOutput()

{ // Decide How To Output Serial.

if (serialVisual == true)

{

arduinoSerialMonitorVisual('-', Signal); // goes to function that makes Serial Monitor Visualizer

}

else

{

sendDataToSerial('S', Signal); // goes to sendDataToSerial function

}

}

void serialOutputWhenBeatHappens()

{

if (serialVisual == true) // Code to Make the Serial Monitor Visualizer Work

{

Serial.print("\*\*\* Heart-Beat Happened \*\*\* "); //ASCII Art Madness

Serial.print("BPM: ");

Serial.println(BPM);

lcd.clear();

lcd.print("BPM: ");

lcd.print(BPM);

}

else

{

sendDataToSerial('B',BPM); // send heart rate with a 'B' prefix

sendDataToSerial('Q',IBI); // send time between beats with a 'Q' prefix

}

}

void arduinoSerialMonitorVisual(char symbol, int data )

{

const int sensorMin = 0; // sensor minimum, discovered through experiment

const int sensorMax = 1024; // sensor maximum, discovered through experiment

int sensorReading = data; // map the sensor range to a range of 12 options:

int range = map(sensorReading, sensorMin, sensorMax, 0, 11);

// do something different depending on the

// range value:

switch (range)

{

case 0:

Serial.println(""); /////ASCII Art Madness

break;

case 1:

Serial.println("---");

break;

case 2:

Serial.println("------");

break;

case 3:

Serial.println("---------");

break;

case 4:

Serial.println("------------");

break;

case 5:

Serial.println("--------------|-");

break;

case 6:

Serial.println("--------------|---");

break;

case 7:

Serial.println("--------------|-------");

break;

case 8:

Serial.println("--------------|----------");

break;

case 9:

Serial.println("--------------|----------------");

break;

case 10:

Serial.println("--------------|-------------------");

break;

case 11:

Serial.println("--------------|-----------------------");

break;

}

}

void sendDataToSerial(char symbol, int data )

{

Serial.print(symbol);

Serial.println(data);

}

ISR(TIMER2\_COMPA\_vect) //triggered when Timer2 counts to 124

{

cli(); // disable interrupts while we do this

Signal = analogRead(pulsePin); // read the Pulse Sensor

sampleCounter += 2; // keep track of the time in mS with this variable

int N = sampleCounter - lastBeatTime; // monitor the time since the last beat to avoid noise

// find the peak and trough of the pulse wave

if(Signal < thresh && N > (IBI/5)\*3) // avoid dichrotic noise by waiting 3/5 of last IBI

{

if (Signal < T) // T is the trough

{

T = Signal; // keep track of lowest point in pulse wave

}

}

if(Signal > thresh && Signal > P)

{ // thresh condition helps avoid noise

P = Signal; // P is the peak

} // keep track of highest point in pulse wave

// NOW IT'S TIME TO LOOK FOR THE HEART BEAT

// signal surges up in value every time there is a pulse

if (N > 250)

{ // avoid high frequency noise

if ( (Signal > thresh) && (Pulse == false) && (N > (IBI/5)\*3) )

{

Pulse = true; // set the Pulse flag when we think there is a pulse

digitalWrite(blinkPin,HIGH); // turn on pin 13 LED

IBI = sampleCounter - lastBeatTime; // measure time between beats in mS

lastBeatTime = sampleCounter; // keep track of time for next pulse

if(secondBeat)

{ // if this is the second beat, if secondBeat == TRUE

secondBeat = false; // clear secondBeat flag

for(int i=0; i<=9; i++) // seed the running total to get a realisitic BPM at startup

{

rate[i] = IBI;

}

}

if(firstBeat) // if it's the first time we found a beat, if firstBeat == TRUE

{

firstBeat = false; // clear firstBeat flag

secondBeat = true; // set the second beat flag

sei(); // enable interrupts again

return; // IBI value is unreliable so discard it

}

// keep a running total of the last 10 IBI values

word runningTotal = 0; // clear the runningTotal variable

for(int i=0; i<=8; i++)

{ // shift data in the rate array

rate[i] = rate[i+1]; // and drop the oldest IBI value

runningTotal += rate[i]; // add up the 9 oldest IBI values

}

rate[9] = IBI; // add the latest IBI to the rate array

runningTotal += rate[9]; // add the latest IBI to runningTotal

runningTotal /= 10; // average the last 10 IBI values

BPM = 60000/runningTotal; // how many beats can fit into a minute? that's BPM!

QS = true; // set Quantified Self flag

// QS FLAG IS NOT CLEARED INSIDE THIS ISR

}

}

if (Signal < thresh && Pulse == true)

{ // when the values are going down, the beat is over

digitalWrite(blinkPin,LOW); // turn off pin 13 LED

Pulse = false; // reset the Pulse flag so we can do it again

amp = P - T; // get amplitude of the pulse wave

thresh = amp/2 + T; // set thresh at 50% of the amplitude

P = thresh; // reset these for next time

T = thresh;

}

if (N > 2500)

{ // if 2.5 seconds go by without a beat

thresh = 512; // set thresh default

P = 512; // set P default

T = 512; // set T default

lastBeatTime = sampleCounter; // bring the lastBeatTime up to date

firstBeat = true; // set these to avoid noise

secondBeat = false; // when we get the heartbeat back

}

sei(); // enable interrupts when youre done!

}// end isr

// When the display powers up, it is configured as follows:

//

// 1. Display clear

// 2. Function set:

// DL = 1; 8-bit interface data

// N = 0; 1-line display

// F = 0; 5x8 dot character font

// 3. Display on/off control:

// D = 0; Display off

// C = 0; Cursor off

// B = 0; Blinking off

// 4. Entry mode set:

// I/D = 1; Increment by 1

// S = 0; No shift

//

// Note, however, that resetting the Arduino doesn't reset the LCD, so we

// can't assume that its in that state when a sketch starts (and the

// LiquidCrystal constructor is called).

LiquidCrystal::LiquidCrystal(uint8\_t rs, uint8\_t rw, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3,

uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7)

{

init(0, rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7);

}

LiquidCrystal::LiquidCrystal(uint8\_t rs, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3,

uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7)

{

init(0, rs, 255, enable, d0, d1, d2, d3, d4, d5, d6, d7);

}

LiquidCrystal::LiquidCrystal(uint8\_t rs, uint8\_t rw, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3)

{

init(1, rs, rw, enable, d0, d1, d2, d3, 0, 0, 0, 0);

}

LiquidCrystal::LiquidCrystal(uint8\_t rs, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3)

{

init(1, rs, 255, enable, d0, d1, d2, d3, 0, 0, 0, 0);

}

void LiquidCrystal::init(uint8\_t fourbitmode, uint8\_t rs, uint8\_t rw, uint8\_t enable,

uint8\_t d0, uint8\_t d1, uint8\_t d2, uint8\_t d3,

uint8\_t d4, uint8\_t d5, uint8\_t d6, uint8\_t d7)

{

\_rs\_pin = rs;

\_rw\_pin = rw;

\_enable\_pin = enable;

\_data\_pins[0] = d0;

\_data\_pins[1] = d1;

\_data\_pins[2] = d2;

\_data\_pins[3] = d3;

\_data\_pins[4] = d4;

\_data\_pins[5] = d5;

\_data\_pins[6] = d6;

\_data\_pins[7] = d7;

pinMode(\_rs\_pin, OUTPUT);

// we can save 1 pin by not using RW. Indicate by passing 255 instead of pin#

if (\_rw\_pin != 255) {

pinMode(\_rw\_pin, OUTPUT);

}

pinMode(\_enable\_pin, OUTPUT);

if (fourbitmode)

\_displayfunction = LCD\_4BITMODE | LCD\_1LINE | LCD\_5x8DOTS;

else

\_displayfunction = LCD\_8BITMODE | LCD\_1LINE | LCD\_5x8DOTS;

begin(16, 1);

}

void LiquidCrystal::begin(uint8\_t cols, uint8\_t lines, uint8\_t dotsize) {

if (lines > 1) {

\_displayfunction |= LCD\_2LINE;

}

\_numlines = lines;

\_currline = 0;

// for some 1 line displays you can select a 10 pixel high font

if ((dotsize != 0) && (lines == 1)) {

\_displayfunction |= LCD\_5x10DOTS;

}

// SEE PAGE 45/46 FOR INITIALIZATION SPECIFICATION!

// according to datasheet, we need at least 40ms after power rises above 2.7V

// before sending commands. Arduino can turn on way befer 4.5V so we'll wait 50

delayMicroseconds(50000);

// Now we pull both RS and R/W low to begin commands

digitalWrite(\_rs\_pin, LOW);

digitalWrite(\_enable\_pin, LOW);

if (\_rw\_pin != 255) {

digitalWrite(\_rw\_pin, LOW);

}

//put the LCD into 4 bit or 8 bit mode

if (! (\_displayfunction & LCD\_8BITMODE)) {

// this is according to the hitachi HD44780 datasheet

// figure 24, pg 46

// we start in 8bit mode, try to set 4 bit mode

write4bits(0x03);

delayMicroseconds(4500); // wait min 4.1ms

// second try

write4bits(0x03);

delayMicroseconds(4500); // wait min 4.1ms

// third go!

write4bits(0x03);

delayMicroseconds(150);

// finally, set to 4-bit interface

write4bits(0x02);

} else {

// this is according to the hitachi HD44780 datasheet

// page 45 figure 23

// Send function set command sequence

command(LCD\_FUNCTIONSET | \_displayfunction);

delayMicroseconds(4500); // wait more than 4.1ms

// second try

command(LCD\_FUNCTIONSET | \_displayfunction);

delayMicroseconds(150);

// third go

command(LCD\_FUNCTIONSET | \_displayfunction);

}

// finally, set # lines, font size, etc.

command(LCD\_FUNCTIONSET | \_displayfunction);

// turn the display on with no cursor or blinking default

\_displaycontrol = LCD\_DISPLAYON | LCD\_CURSOROFF | LCD\_BLINKOFF;

display();

// clear it off

clear();

// Initialize to default text direction (for romance languages)

\_displaymode = LCD\_ENTRYLEFT | LCD\_ENTRYSHIFTDECREMENT;

// set the entry mode

command(LCD\_ENTRYMODESET | \_displaymode);

}

/\*\*\*\*\*\*\*\*\*\* high level commands, for the user! \*/

void LiquidCrystal::clear()

{

command(LCD\_CLEARDISPLAY); // clear display, set cursor position to zero

delayMicroseconds(2000); // this command takes a long time!

}

void LiquidCrystal::home()

{

command(LCD\_RETURNHOME); // set cursor position to zero

delayMicroseconds(2000); // this command takes a long time!

}

void LiquidCrystal::setCursor(uint8\_t col, uint8\_t row)

{

int row\_offsets[] = { 0x00, 0x40, 0x14, 0x54 };

if ( row >= \_numlines ) {

row = \_numlines-1; // we count rows starting w/0

}

command(LCD\_SETDDRAMADDR | (col + row\_offsets[row]));

}

// Turn the display on/off (quickly)

void LiquidCrystal::noDisplay() {

\_displaycontrol &= ~LCD\_DISPLAYON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

void LiquidCrystal::display() {

\_displaycontrol |= LCD\_DISPLAYON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

// Turns the underline cursor on/off

void LiquidCrystal::noCursor() {

\_displaycontrol &= ~LCD\_CURSORON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

void LiquidCrystal::cursor() {

\_displaycontrol |= LCD\_CURSORON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

// Turn on and off the blinking cursor

void LiquidCrystal::noBlink() {

\_displaycontrol &= ~LCD\_BLINKON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

void LiquidCrystal::blink() {

\_displaycontrol |= LCD\_BLINKON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

// These commands scroll the display without changing the RAM

void LiquidCrystal::scrollDisplayLeft(void) {

command(LCD\_CURSORSHIFT | LCD\_DISPLAYMOVE | LCD\_MOVELEFT);

}

void LiquidCrystal::scrollDisplayRight(void) {

command(LCD\_CURSORSHIFT | LCD\_DISPLAYMOVE | LCD\_MOVERIGHT);

}

// This is for text that flows Left to Right

void LiquidCrystal::leftToRight(void) {

\_displaymode |= LCD\_ENTRYLEFT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

// This is for text that flows Right to Left

void LiquidCrystal::rightToLeft(void) {

\_displaymode &= ~LCD\_ENTRYLEFT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

// This will 'right justify' text from the cursor

void LiquidCrystal::autoscroll(void) {

\_displaymode |= LCD\_ENTRYSHIFTINCREMENT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

// This will 'left justify' text from the cursor

void LiquidCrystal::noAutoscroll(void) {

\_displaymode &= ~LCD\_ENTRYSHIFTINCREMENT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

// Allows us to fill the first 8 CGRAM locations

// with custom characters

void LiquidCrystal::createChar(uint8\_t location, uint8\_t charmap[]) {

location &= 0x7; // we only have 8 locations 0-7

command(LCD\_SETCGRAMADDR | (location << 3));

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

write(charmap[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\* mid level commands, for sending data/cmds \*/

inline void LiquidCrystal::command(uint8\_t value) {

send(value, LOW);

}

inline size\_t LiquidCrystal::write(uint8\_t value) {

send(value, HIGH);

return 1; // assume sucess

}

/\*\*\*\*\*\*\*\*\*\*\*\* low level data pushing commands \*\*\*\*\*\*\*\*\*\*/

// write either command or data, with automatic 4/8-bit selection

void LiquidCrystal::send(uint8\_t value, uint8\_t mode) {

digitalWrite(\_rs\_pin, mode);

// if there is a RW pin indicated, set it low to Write

if (\_rw\_pin != 255) {

digitalWrite(\_rw\_pin, LOW);

}

if (\_displayfunction & LCD\_8BITMODE) {

write8bits(value);

} else {

write4bits(value>>4);

write4bits(value);

}

}

void LiquidCrystal::pulseEnable(void) {

digitalWrite(\_enable\_pin, LOW);

delayMicroseconds(1);

digitalWrite(\_enable\_pin, HIGH);

delayMicroseconds(1); // enable pulse must be >450ns

digitalWrite(\_enable\_pin, LOW);

delayMicroseconds(100); // commands need > 37us to settle

}

void LiquidCrystal::write4bits(uint8\_t value) {

for (int i = 0; i < 4; i++) {

pinMode(\_data\_pins[i], OUTPUT);

digitalWrite(\_data\_pins[i], (value >> i) & 0x01);

}

pulseEnable();

}

void LiquidCrystal::write8bits(uint8\_t value) {

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

pinMode(\_data\_pins[i], OUTPUT);

digitalWrite(\_data\_pins[i], (value >> i) & 0x01);

}

pulseEnable();

}

int pinTemp = A1; //This is where our Output data goes

void setup() {

Serial.begin(9600);

}

void loop() {

int temp = analogRead(pinTemp); //Read the analog pin

temp = temp \* 0.48828125; // convert output (mv) to readable celcius

Serial.print("Temperature: ");

Serial.print(temp);

Serial.println("C"); //print the temperature status

delay(1000);

}