# Progam code listings main.c

Only main.c was turned into a pdf file. The other files are from a library and had minimal or no changes done to them. The only changes to the libraries are in lcdpcf8574.h line 41 to 48, and pcf8574.h line 14. These changes are detailed in chapter 2.

//Name: LCD\_Atle\_Torstein

//Authors: Atle Undrum & Torstein Gaarder

//Library used to control LCD

//lcdpcf8574 lib sample

//copyright (c) Davide Gironi, 2013

//Released under GPLv3.

//Source: http://davidegironi.blogspot.no/2013/06/an-avr-atmega-library-for-hd44780-based.html#.WhLVyllaSUk

#include <avr/io.h>

#include <stdio.h>

#include <stdlib.h>

#include <avr/pgmspace.h>

#include <avr/interrupt.h>

#include "lcdpcf8574.h"

#include "i2cmaster.h"

#define F\_CPU 1000000

void InitialiseGeneral**();**

void InitialiseTimer1**();**

void InitialiseADC**();**

void InitialiseTimer3\_FastPWM\_Single**();**

void Initialise\_HW\_Interrupts**();**

//Variables for the PID controllers

struct PID FlowController**;**

double FlowPosition**;** //Current value

double FlowError**;** //Current value minus Set-point

double FlowOutput**;**

struct PID LevelController**;**

double LevelPosition**;**

double LevelError**;**

double LevelOutput**;**

double LevelSetpoint**;**

//Level alarms

unsigned char LAH\_triggered**;**

unsigned char LAL\_triggered**;**

unsigned char LED\_Pattern**;**

unsigned char Button\_value**;**

//LCD

char lcd\_int\_double\_to\_string**;**

//#define UART\_BAUD\_RATE 2400

//#include "uart.h"

int main**(**void**)**

**{**

///////////////////

//initializations//

///////////////////

//Initialization functions

InitialiseGeneral**();**

InitialiseTimer1**();**

InitialiseADC**();**

InitialiseTimer3\_FastPWM\_Single**();**

Initialise\_HW\_Interrupts**();**

//Initialize LCD

lcd\_init**(**LCD\_DISP\_ON\_BLINK**);**

lcd\_home**();**

uint8\_t led **=** 0**;**

lcd\_led**(**led**);** //set led

//Write text which doesn't change during operations

lcd\_gotoxy**(**0**,**0**);**//(0,1) =Starting at line 2 position 1.

lcd\_puts**(**"L= "**);**

lcd\_gotoxy**(**0**,** 1**);**

lcd\_puts**(**"F= "**);**

lcd\_gotoxy**(**8**,** 0**);**

lcd\_puts**(**"S= "**);**

lcd\_gotoxy**(**8**,** 1**);**

lcd\_puts**(**"O= "**);**

/////////////////////////

//Variable declarations//

/////////////////////////

//Used for input scaling

double FlowOutput\_inPercent**;**

double LevelPosition\_inPercent**;**

double FlowPosition\_inPercent**;**

char charray**[**5**];**

char charray1**[**5**];**

char charray2**[**4**];**

int b**;**

int c**;**

int d**;**

int e**;**

unsigned output\_unsigned**;**

double num1**;**

**while(**1**)**

**{**

///////////////

//Reset alarm//

///////////////

//Not completed.

//Because the LCD is so slow, this should be moved to an interrupt before being completed.

//Button\_value = DDRL;

**if** **(**1 **==** Button\_value**)**

**{**

LAH\_triggered **=** 0**;**

LAL\_triggered **=** 0**;**

**}**

//High alarm warning lamp turned off

**if** **(**0 **==** LAH\_triggered**)**

**{**

LED\_Pattern **&=** 0b01111111**;**

**}**

//Low alarm warning lamp turned off

**if** **(**0 **==** LAL\_triggered**)**

**{**

LED\_Pattern **&=** 0b10111111**;**

**}**

///////////////

//LCD display//

///////////////

// Input scaling

//Inputs:

//LevelPosition scale = 0..255

//FlowPosition scale = 0..255

//FlowOutput scale = -500..500 (This changes as the PID gains are changed, making the scaling more complex.)

//Desired outputs:

//All = 0-99

//for level and flow:

//wanted value = Position/2.55

//for output:

//wanted value = (FlowOutput+300)/7

LevelPosition\_inPercent **=** LevelPosition**/**2.55**;**

FlowPosition\_inPercent **=** FlowPosition**/**2.55**;**

FlowOutput\_inPercent **=** **(**FlowOutput**+**300**)/**7**;**

////////////////////////////////////////

//Level

//Code to split the double into two integers before it can be printed to the LCD

//Each input is one double which needs to be split into two char arrays, because that it the data type the LCD library accepts.

b **=** LevelPosition\_inPercent**;**

num1 **=** LevelPosition\_inPercent **\*** 100**;**

c **=** num1**;**

d **=** b **\*** 100**;**

e **=** c **-** d **;**

sprintf**(**charray**,** "%2.1hhi"**,** b**);** //Put integer into char array

sprintf**(**charray1**,** "%.1hhi"**,** e**);**

//Write level to LCD

lcd\_gotoxy**(**2**,** 0**);**

lcd\_puts**(**charray**);**

lcd\_gotoxy**(**4**,** 0**);**

lcd\_puts**(**","**);**

lcd\_gotoxy**(**5**,** 0**);**

lcd\_puts**(**charray1**);**

lcd\_gotoxy**(**8**,** 0**);**

////////////////////////////////////////

//Flow

//Code to split the double into two integers before it can be printed to the LCD

b **=** FlowPosition\_inPercent**;**

num1 **=** FlowPosition\_inPercent **\*** 100**;**

c **=** num1**;**

d **=** b **\*** 100**;**

e **=** c **-** d **;**

sprintf**(**charray**,** "%2.1hhi"**,** b**);**

sprintf**(**charray1**,** "%.1hhi"**,** e**);**

//Write flow to LCD

lcd\_gotoxy**(**2**,** 1**);**

lcd\_puts**(**charray**);**

lcd\_gotoxy**(**4**,** 1**);**

lcd\_puts**(**","**);**

lcd\_gotoxy**(**5**,** 1**);**

lcd\_puts**(**charray1**);**

lcd\_gotoxy**(**8**,** 1**);**

//\_delay\_ms(1000);

////////////////////////////////////////

//Set point

//Code to split the double into two integers before it can be printed to the LCD

b **=** LevelSetpoint**;**

num1 **=** LevelSetpoint **\*** 100**;**

c **=** num1**;**

d **=** b **\*** 100**;**

e **=** c **-** d **;**

sprintf**(**charray**,** "%2.1hhi"**,** b**);**

sprintf**(**charray1**,** "%.1hhi"**,** e**);**

//Write set point to LCD

lcd\_gotoxy**(**10**,** 0**);**

lcd\_puts**(**charray**);**

lcd\_gotoxy**(**12**,** 0**);**

lcd\_puts**(**","**);**

lcd\_gotoxy**(**13**,** 0**);**

lcd\_puts**(**charray1**);**

lcd\_gotoxy**(**15**,** 0**);**

////////////////////////////////////////

//Output

output\_unsigned **=** FlowOutput\_inPercent**;**

//Limits the output to 0..99

**if** **(**output\_unsigned**>**99**)**

**{**

output\_unsigned**=** 99**;**

**}**

**else** **if(**output\_unsigned**<**0**)**

**{**

output\_unsigned **=** 0**;**

**}**

sprintf**(**charray2**,** "%.1hi"**,** output\_unsigned**);**

//Write output to LCD

lcd\_gotoxy**(**10**,** 1**);**

lcd\_puts**(**charray2**);**

**}**

**}**

//Example code from https://www.embedded.com/design/prototyping-and-development/4211211/PID-without-a-PhD

//The only thing we did to this code was fixing a single bug and slightly restructure it for readability.

**typedef** struct PID //Struct for storing PID values.

**{**

double dState**;** // Last position input

double iState**;** // Integrator state

double iMax**,** iMin**;** // Maximum and minimum allowable integrator state

double iGain**,** pGain**,** dGain**;**// integral gain, proportional gain, derivative gain

**}**PID**;**

double UpdatePID**(**PID **\*** pid**,** double error**,** double position**)** //PID controller.

**{**

double pTerm**,** dTerm**,** iTerm**;**

// calculate the proportional term

pTerm **=** pid**->**pGain **\*** error**;**

// calculate the integral state with appropriate limiting and calculates the integral term

pid**->**iState **+=** error**;**

**if** **(**pid**->**iState **>** pid**->**iMax**)**pid**->**iState **=** pid**->**iMax**;**

**else** **if** **(**pid**->**iState **<** pid**->**iMin**)** pid**->**iState **=** pid**->**iMin**;**

iTerm **=** pid**->**iGain **\*** pid**->**iState**;**

// calculates the derivate term and stores the state

dTerm **=** pid**->**dGain **\*** **(**position **-** pid**->**dState**);**

pid**->**dState **=** position**;**

// returns result

**return** pTerm **+** iTerm **-** dTerm**;**

**}**

//End of example code

void InitialiseGeneral**()** //General stuff which doesn't go in the other initialization functions.

**{**

/\*

Ports used

input

LAH = PD0

LAL = PD1

reset alarm = some bit on PL

LevelPosition = PF2

FlowPosition = PF3

Output

LCD = PD0 & PD1

LAH LED = PA?

LAL LED = PA?

PWM = PE3

\*/

//Port declaration

//Buttons

DDRL **=** 0x00**;** //Port L input

PORTL **=** 0x00**;** //Pull up resistors

DDRE **=** 0b00001000**;**

PORTE **=** 0x00**;**

//LED

DDRA **=** 0xFF**;** //Port A output

PORTA **=** 0x00**;** //Initially off

//Variable initialization

//temp declaration

LevelSetpoint **=** 50**;**

//Temp. The position values will come from analog inputs

FlowPosition **=** 10**;**

LevelPosition **=** 10**;**

//Initialize values to flow controller

FlowController**.**iGain **=** 1**;**

FlowController**.**pGain **=** 1**;**

FlowController**.**dGain **=** 10**;**

FlowController**.**iMax **=** 100**;**

FlowController**.**iMin **=** 0**;**

//Again but for the level controller

LevelController**.**iGain **=** 1**;**

LevelController**.**pGain **=** 1**;**

LevelController**.**dGain **=** 10**;**

LevelController**.**iMax **=** 100**;**

LevelController**.**iMin **=** 0**;**

sei**();** //Enable interrupt

**}**

//All the

void InitialiseTimer1**()** //Copied from TimerDemo3. Generates interrupt on a one second interval. This will be changed.

**{**

TCCR1A **=** 0b00000000**;** // Normal port operation (OC1A, OC1B, OC1C), Clear Timer on 'Compare Match' (CTC) waveform mode)

TCCR1B **=** 0b00001101**;** // CTC waveform mode, use prescaler 1024

TCCR1C **=** 0b00000000**;**

OCR1AH **=** 0x03**;** // Output Compare Registers (16 bit) OCR1BH and OCR1BL

OCR1AL **=** 0xD0**;**

TCNT1H **=** 0b00000000**;** // Timer/Counter count/value registers (16 bit) TCNT1H and TCNT1L

TCNT1L **=** 0b00000000**;**

TIMSK1 **=** 0b00000010**;** // bit 1 OCIE1A Use 'Output Compare A Match' Interrupt, i.e. generate an interrupt

// when the timer reaches the set value (in the OCR1A register)

**}**

void InitialiseADC**()** //ADC. Copied from TwoPotentiometers. Most of the comments are removed, but the rest is unchanged. Converts level and flow.

**{**

ADMUX **=** 0b01100010**;** // AVCC REF, Left-adjust output (Read most-significant 8 bits via ADCH), Convert channel 2

ADCSRA **=** 0b10101101**;** // ADC enabled, Auto trigger, Interrupt enabled, Prescaler = 32

ADCSRB **&=** 0b11110000**;** // clear bits 3,2,1,0 (Free running mode)

DIDR0 **=** 0b00001100**;** // Disable digital input on bits 2 and 3

DIDR2 **=** 0b11111111**;** // Disable digital input on all bits (64-pin version of ATmega1281 does not even have these inputs)

ADCSRA **|=** 0b01000000**;** // start ADC conversion

**}**

void InitialiseTimer3\_FastPWM\_Single**()** //PWM. Copied from PWM\_Servo\_Singe\_Potentiometer. Controls the servo.

**{**

TCCR3A **=** 0b10000010**;** // Fast PWM non inverting, ICR3 used as TOP

TCCR3B **=** 0b00011001**;** // Fast PWM, Use Prescaler '1'

TCCR3C **=** 0b00000000**;**

ICR3 **=** 25000**;**

TCNT3H **=** 0**;** // 16-bit access (write high byte first, read low byte first)

TCNT3L **=** 0**;**

OCR3A **=** 1750**;**

TIMSK3 **=** 0b00000000**;** // No interrupts needed, PWM pulses appears directly on OC3A, OC3B (Port E Bits 3,4)

TIFR3 **=** 0b00101111**;** // Clear all interrupt flags

**}**

void Initialise\_HW\_Interrupts**()** //Hardware interrupts. Copied from TimerDemo4. Used as level alarms; high and low.

**{**

EICRA **=** 0b00000000**;** // INT 3,2 not used, Interrupt Sense (INT1, INT0) falling-edge triggered

EICRB **=** 0b00001010**;** // INT7 ... 4 not used

EIMSK **=** 0b00110000**;** // Enable INT1, INT0

EIFR **=** 0b00110000**;** // Clear INT1 and INT0 interrupt flags (in case a spurious interrupt has occurred during chip startup)

**}**

ISR**(**ADC\_vect**)** // ADC Interrupt Handler. Also from TwoPotentiometers with minimal changes. This interrupt handler is common for all ADC channels

**{**

// Need to alternate which channel is converted

unsigned char ADMUX\_temp **=** ADMUX**;**

unsigned char ADCH\_temp **=** ADCH**;**

ADMUX\_temp **&=** 0b00011111**;** // Mask off non-multiplexer bits

**if(**0b00000010 **==** ADMUX\_temp**)**

**{**

LevelPosition **=** ADCH\_temp**;**

ADMUX **=** 0b01100011**;** // Set ADMUX ADC register - next conversion is for ADC3

**}**

**else**

**{**

FlowPosition **=** ADCH\_temp**;**

ADMUX **=** 0b01100010**;** // Set ADMUX ADC register - next conversion is for ADC2

**}**

**}**

ISR**(**TIMER1\_COMPA\_vect**)** //Runs the PID regulators when timer 1 triggers.

**{**

LevelError **=** LevelPosition **-** LevelSetpoint**;** //Difference between wanted level and current level

LevelOutput **=** UpdatePID**(&**LevelController**,** LevelError**,** LevelPosition**);** //Master. In cascade regulation, the master regulator calculates the set point for the slave.

FlowError **=** FlowPosition **-** LevelOutput**;** //Difference between wanted flow and current flow

FlowOutput **=** UpdatePID**(&**FlowController**,** FlowError**,** FlowPosition**);** //Slave. Uses the set point from the master to regulate the servo.

//Servo output. Scaling and limiting the output.

unsigned PWM\_output **=** FlowOutput **+** 2000**;**

**if** **(**PWM\_output**>**2500**)**

**{**

PWM\_output**=** 2500**;**

**}**

**else** **if(**PWM\_output**<**1500**)**

**{**

PWM\_output **=** 1500**;**

**}**

OCR3A **=** PWM\_output**;**

**}**

//Simple input which triggers an alarm. In this case it's just a light to demonstrate that it works.

//These are not complete and doesn't work correctly. This might be a SW issue or a HW issue. More testing is needed.

ISR**(**INT4\_vect**)** //Level alarm high.

**{**

LAH\_triggered **=** 1**;**

LED\_Pattern **|=** 0b10000000**;**

**}**

ISR**(**INT5\_vect**)** //Level alarm low.

**{**

LAL\_triggered **=** 1**;**

LED\_Pattern **|=** 0b01000000**;**

**}**