#include "ABlocks\_LiquidCrystal\_I2C.h"

#include <inttypes.h>

#include <Arduino.h>

#include <Wire.h>

// 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\_I2C::LiquidCrystal\_I2C(uint8\_t lcd\_addr, uint8\_t lcd\_cols, uint8\_t lcd\_rows, uint8\_t charsize)

{

\_addr = lcd\_addr;

\_cols = lcd\_cols;

\_rows = lcd\_rows;

\_charsize = charsize;

\_backlightval = LCD\_BACKLIGHT;

}

void LiquidCrystal\_I2C::begin() {

Wire.begin();

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

if (\_rows > 1) {

\_displayfunction |= LCD\_2LINE;

}

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

if ((\_charsize != 0) && (\_rows == 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

delay(50);

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

expanderWrite(\_backlightval); // reset expanderand turn backlight off (Bit 8 =1)

delay(1000);

//put the LCD into 4 bit mode

// 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 << 4);

delayMicroseconds(4500); // wait min 4.1ms

// second try

write4bits(0x03 << 4);

delayMicroseconds(4500); // wait min 4.1ms

// third go!

write4bits(0x03 << 4);

delayMicroseconds(150);

// finally, set to 4-bit interface

write4bits(0x02 << 4);

// 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 roman languages)

\_displaymode = LCD\_ENTRYLEFT | LCD\_ENTRYSHIFTDECREMENT;

// set the entry mode

command(LCD\_ENTRYMODESET | \_displaymode);

home();

}

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

void LiquidCrystal\_I2C::clear(){

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

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

}

void LiquidCrystal\_I2C::home(){

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

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

}

void LiquidCrystal\_I2C::setCursor(uint8\_t col, uint8\_t row){

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

if (row > \_rows) {

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

}

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

}

// Turn the display on/off (quickly)

void LiquidCrystal\_I2C::noDisplay() {

\_displaycontrol &= ~LCD\_DISPLAYON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

void LiquidCrystal\_I2C::display() {

\_displaycontrol |= LCD\_DISPLAYON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

// Turns the underline cursor on/off

void LiquidCrystal\_I2C::noCursor() {

\_displaycontrol &= ~LCD\_CURSORON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

void LiquidCrystal\_I2C::cursor() {

\_displaycontrol |= LCD\_CURSORON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

// Turn on and off the blinking cursor

void LiquidCrystal\_I2C::noBlink() {

\_displaycontrol &= ~LCD\_BLINKON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

void LiquidCrystal\_I2C::blink() {

\_displaycontrol |= LCD\_BLINKON;

command(LCD\_DISPLAYCONTROL | \_displaycontrol);

}

// These commands scroll the display without changing the RAM

void LiquidCrystal\_I2C::scrollDisplayLeft(void) {

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

}

void LiquidCrystal\_I2C::scrollDisplayRight(void) {

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

}

// This is for text that flows Left to Right

void LiquidCrystal\_I2C::leftToRight(void) {

\_displaymode |= LCD\_ENTRYLEFT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

// This is for text that flows Right to Left

void LiquidCrystal\_I2C::rightToLeft(void) {

\_displaymode &= ~LCD\_ENTRYLEFT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

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

void LiquidCrystal\_I2C::autoscroll(void) {

\_displaymode |= LCD\_ENTRYSHIFTINCREMENT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

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

void LiquidCrystal\_I2C::noAutoscroll(void) {

\_displaymode &= ~LCD\_ENTRYSHIFTINCREMENT;

command(LCD\_ENTRYMODESET | \_displaymode);

}

// Allows us to fill the first 8 CGRAM locations

// with custom characters

void LiquidCrystal\_I2C::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]);

}

}

// Turn the (optional) backlight off/on

void LiquidCrystal\_I2C::noBacklight(void) {

\_backlightval=LCD\_NOBACKLIGHT;

expanderWrite(0);

}

void LiquidCrystal\_I2C::backlight(void) {

\_backlightval=LCD\_BACKLIGHT;

expanderWrite(0);

}

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

inline void LiquidCrystal\_I2C::command(uint8\_t value) {

send(value, 0);

}

inline size\_t LiquidCrystal\_I2C::write(uint8\_t value) {

send(value, Rs);

return 1; //fixed for new compiler version IDE 1.6+ !!! by Juanjo

}

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

// write either command or data

void LiquidCrystal\_I2C::send(uint8\_t value, uint8\_t mode) {

uint8\_t highnib=value&0xf0;

uint8\_t lownib=(value<<4)&0xf0;

write4bits((highnib)|mode);

write4bits((lownib)|mode);

}

void LiquidCrystal\_I2C::write4bits(uint8\_t value) {

expanderWrite(value);

pulseEnable(value);

}

void LiquidCrystal\_I2C::expanderWrite(uint8\_t \_data){

Wire.beginTransmission(\_addr);

Wire.write((int)(\_data) | \_backlightval);

Wire.endTransmission();

}

void LiquidCrystal\_I2C::pulseEnable(uint8\_t \_data){

expanderWrite(\_data | En); // En high

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

expanderWrite(\_data & ~En); // En low

delayMicroseconds(50); // commands need > 37us to settle

}

void LiquidCrystal\_I2C::load\_custom\_character(uint8\_t char\_num, uint8\_t \*rows){

createChar(char\_num, rows);

}

void LiquidCrystal\_I2C::setBacklight(uint8\_t new\_val){

if (new\_val) {

backlight(); // turn backlight on

} else {

noBacklight(); // turn backlight off

}

}

void LiquidCrystal\_I2C::printstr(const char c[]){

//This function is not identical to the function used for "real" I2C displays

//it's here so the user sketch doesn't have to be changed

print(c);

}