/\*

\* PCD8544 - Interface with Philips PCD8544 (or compatible) LCDs.

\*

\* Copyright (c) 2010 Carlos Rodrigues <cefrodrigues@gmail.com>

\*

\* Permission is hereby granted, free of charge, to any person obtaining a copy

\* of this software and associated documentation files (the "Software"), to deal

\* in the Software without restriction, including without limitation the rights

\* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

\* copies of the Software, and to permit persons to whom the Software is

\* furnished to do so, subject to the following conditions:

\*

\* The above copyright notice and this permission notice shall be included in

\* all copies or substantial portions of the Software.

\*

\* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

\* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

\* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

\* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

\* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

\* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN

\* THE SOFTWARE.

\*/

#include "PCD8544.h"

#include <Arduino.h>

#include <avr/pgmspace.h>

extern const PROGMEM unsigned char font5x8[][5];

/\*

\* If this was a ".h", it would get added to sketches when using

\* the "Sketch -> Import Library..." menu on the Arduino IDE...

\*/

PCD8544::PCD8544(unsigned char sclk, unsigned char sdin,

unsigned char dc, unsigned char reset,

unsigned char sce):

pin\_sclk(sclk),

pin\_sdin(sdin),

pin\_dc(dc),

pin\_reset(reset),

pin\_sce(sce)

{}

void PCD8544::begin(unsigned char model)

{

this->column = 0;

this->line = 0;

// Sanitize the custom glyphs...

memset(this->custom, 0, sizeof(this->custom));

// All pins are outputs (these displays cannot be read)...

pinMode(this->pin\_sclk, OUTPUT);

pinMode(this->pin\_sdin, OUTPUT);

pinMode(this->pin\_dc, OUTPUT);

pinMode(this->pin\_reset, OUTPUT);

pinMode(this->pin\_sce, OUTPUT);

// Reset the controller state...

digitalWrite(this->pin\_reset, HIGH);

digitalWrite(this->pin\_sce, HIGH);

digitalWrite(this->pin\_reset, LOW);

delay(100);

digitalWrite(this->pin\_reset, HIGH);

// Set the LCD parameters...

this->send(PCD8544\_CMD, 0x21); // extended instruction set control (H=1)

this->send(PCD8544\_CMD, 0x13); // bias system (1:48)

if (model == CHIP\_ST7576) {

this->send(PCD8544\_CMD, 0xe0); // higher Vop, too faint at default

this->send(PCD8544\_CMD, 0x05); // partial display mode

} else {

this->send(PCD8544\_CMD, 0xc2); // default Vop (3.06 + 66 \* 0.06 = 7V)

}

this->send(PCD8544\_CMD, 0x20); // extended instruction set control (H=0)

this->send(PCD8544\_CMD, 0x09); // all display segments on

// Clear RAM contents...

this->clear();

// Activate LCD...

this->send(PCD8544\_CMD, 0x08); // display blank

this->send(PCD8544\_CMD, 0x0c); // normal mode (0x0d = inverse mode)

delay(100);

// Place the cursor at the origin...

this->send(PCD8544\_CMD, 0x80);

this->send(PCD8544\_CMD, 0x40);

}

void PCD8544::stop()

{

this->clear();

this->setPower(false);

}

void PCD8544::clear()

{

this->setCursor(0, 0);

for (unsigned short i = 0; i < PCD8544\_WIDTH \* (PCD8544\_HEIGHT/8); i++) {

this->send(PCD8544\_DATA, 0x00);

}

this->setCursor(0, 0);

}

void PCD8544::clearLine()

{

this->setCursor(0, this->line);

for (unsigned char i = 0; i < PCD8544\_WIDTH; i++) {

this->send(PCD8544\_DATA, 0x00);

}

this->setCursor(0, this->line);

}

void PCD8544::setPower(bool on)

{

this->send(PCD8544\_CMD, on ? 0x20 : 0x24);

}

inline void PCD8544::display()

{

this->setPower(true);

}

inline void PCD8544::noDisplay()

{

this->setPower(false);

}

void PCD8544::setInverse(bool inverse)

{

this->send(PCD8544\_CMD, inverse ? 0x0d : 0x0c);

}

void PCD8544::home()

{

this->setCursor(0, this->line);

}

void PCD8544::setCursor(unsigned char column, unsigned char line)

{

if (column > PCD8544\_WIDTH) {

column = 0;

line++;

}

if (line > PCD8544\_HEIGHT / 8)

line = 0;

this->column = column;

this->line = line;

this->send(PCD8544\_CMD, 0x80 | column);

this->send(PCD8544\_CMD, 0x40 | line);

}

void PCD8544::createChar(unsigned char chr, const unsigned char \*glyph)

{

// ASCII 0-31 only...

if (chr >= ' ') {

return;

}

this->custom[chr] = glyph;

}

size\_t PCD8544::write(uint8\_t chr)

{

// ASCII 7-bit only...

if (chr >= 0x7f) {

return 0;

}

if (chr == '\n') {

column = 0;

line = (line + 1) % (PCD8544\_HEIGHT/9 + 1);

return 0;

} else if (chr == '\r') {

column = 0;

return 0;

}

const unsigned char \*glyph;

unsigned char pgm\_buffer[5];

if (chr >= ' ') {

// Regular ASCII characters are kept in flash to save RAM...

memcpy\_P(pgm\_buffer, &font5x8[chr - ' '], sizeof(pgm\_buffer));

glyph = pgm\_buffer;

} else {

// Custom glyphs, on the other hand, are stored in RAM...

if (custom[chr]) {

glyph = custom[chr];

} else {

// Default to a space character if unset...

memcpy\_P(pgm\_buffer, &font5x8[0], sizeof(pgm\_buffer));

glyph = pgm\_buffer;

}

}

// Output one column at a time...

for (unsigned char i = 0; i < 5; i++) {

this->send(PCD8544\_DATA, glyph[i]);

}

// One column between characters...

this->send(PCD8544\_DATA, 0x00);

// Update the cursor position...

this->column = (this->column + 6) % PCD8544\_WIDTH;

if (this->column == 0) {

this->line = (this->line + 1) % (PCD8544\_HEIGHT/9 + 1);

}

#if ARDUINO >= 100

return 1;

#endif

}

void PCD8544::draw8x8(const unsigned char \*data)

{

// Output one column at a time...

for (unsigned char i = 0; i < 8; i++) {

this->send(PCD8544\_DATA, data[i]);

}

this->setCursor(column + 8, line);

}

void PCD8544::draw16x16(const unsigned char \*data)

{

unsigned char scolumn = this->column;

unsigned char sline = this->line;

// Output one column at a time...

for (unsigned char i = 0; i < 16; i++) {

this->send(PCD8544\_DATA, data[i]);

}

this->setCursor(scolumn, sline + 1);

for (unsigned char i = 0; i < 16; i++) {

this->send(PCD8544\_DATA, data[i + 16]);

}

// Update the cursor position...

this->setCursor(scolumn + 16, sline);

}

void PCD8544::drawColumn(unsigned char lines, unsigned char value)

{

unsigned char scolumn = this->column;

unsigned char sline = this->line;

// Keep "value" within range...

if (value > lines\*8) {

value = lines\*8;

}

// Find the line where "value" resides...

unsigned char mark = (lines\*8 - 1 - value)/8;

// Clear the lines above the mark...

for (unsigned char line = 0; line < mark; line++) {

this->setCursor(scolumn, sline + line);

this->send(PCD8544\_DATA, 0x00);

}

// Compute the byte to draw at the "mark" line...

unsigned char b = 0xff;

for (unsigned char i = 0; i < lines\*8 - mark\*8 - value; i++) {

b <<= 1;

}

this->setCursor(scolumn, sline + mark);

this->send(PCD8544\_DATA, b);

// Fill the lines below the mark...

for (unsigned char line = mark + 1; line < lines; line++) {

this->setCursor(scolumn, sline + line);

this->send(PCD8544\_DATA, 0xff);

}

// Leave the cursor in a consistent position...

this->setCursor(scolumn + 1, sline);

}

void PCD8544::send(unsigned char type, unsigned char data)

{

digitalWrite(this->pin\_dc, type);

digitalWrite(this->pin\_sce, LOW);

shiftOut(this->pin\_sdin, this->pin\_sclk, MSBFIRST, data);

digitalWrite(this->pin\_sce, HIGH);

}

/\* vim: set expandtab ts=4 sw=4: \*/