/\*!

\* @file Adafruit\_GrayOLED.cpp

\*

\* This is documentation for Adafruit's generic library for grayscale

\* OLED displays: http://www.adafruit.com/category/63\_98

\*

\* These displays use I2C or SPI to communicate. I2C requires 2 pins

\* (SCL+SDA) and optionally a RESET pin. SPI requires 4 pins (MOSI, SCK,

\* select, data/command) and optionally a reset pin. Hardware SPI or

\* 'bitbang' software SPI are both supported.

\*

\* Adafruit invests time and resources providing this open source code,

\* please support Adafruit and open-source hardware by purchasing

\* products from Adafruit!

\*

\*/

#if !defined(\_\_AVR\_ATtiny85\_\_) // Not for ATtiny, at all

#include "Adafruit\_GrayOLED.h"

#include <Adafruit\_GFX.h>

// SOME DEFINES AND STATIC VARIABLES USED INTERNALLY -----------------------

#define grayoled\_swap(a, b) \

(((a) ^= (b)), ((b) ^= (a)), ((a) ^= (b))) ///< No-temp-var swap operation

// CONSTRUCTORS, DESTRUCTOR ------------------------------------------------

/\*!

@brief Constructor for I2C-interfaced OLED displays.

@param bpp Bits per pixel, 1 for monochrome, 4 for 16-gray

@param w

Display width in pixels

@param h

Display height in pixels

@param twi

Pointer to an existing TwoWire instance (e.g. &Wire, the

microcontroller's primary I2C bus).

@param rst\_pin

Reset pin (using Arduino pin numbering), or -1 if not used

(some displays might be wired to share the microcontroller's

reset pin).

@param clkDuring

Speed (in Hz) for Wire transmissions in library calls.

Defaults to 400000 (400 KHz), a known 'safe' value for most

microcontrollers, and meets the OLED datasheet spec.

Some systems can operate I2C faster (800 KHz for ESP32, 1 MHz

for many other 32-bit MCUs), and some (perhaps not all)

Many OLED's can work with this -- so it's optionally be specified

here and is not a default behavior. (Ignored if using pre-1.5.7

Arduino software, which operates I2C at a fixed 100 KHz.)

@param clkAfter

Speed (in Hz) for Wire transmissions following library

calls. Defaults to 100000 (100 KHz), the default Arduino Wire

speed. This is done rather than leaving it at the 'during' speed

because other devices on the I2C bus might not be compatible

with the faster rate. (Ignored if using pre-1.5.7 Arduino

software, which operates I2C at a fixed 100 KHz.)

@note Call the object's begin() function before use -- buffer

allocation is performed there!

\*/

Adafruit\_GrayOLED::Adafruit\_GrayOLED(uint8\_t bpp, uint16\_t w, uint16\_t h,

TwoWire \*twi, int8\_t rst\_pin,

uint32\_t clkDuring, uint32\_t clkAfter)

: Adafruit\_GFX(w, h), i2c\_preclk(clkDuring), i2c\_postclk(clkAfter),

buffer(NULL), dcPin(-1), csPin(-1), rstPin(rst\_pin), \_bpp(bpp) {

i2c\_dev = NULL;

\_theWire = twi;

}

/\*!

@brief Constructor for SPI GrayOLED displays, using software (bitbang)

SPI.

@param bpp Bits per pixel, 1 for monochrome, 4 for 16-gray

@param w

Display width in pixels

@param h

Display height in pixels

@param mosi\_pin

MOSI (master out, slave in) pin (using Arduino pin numbering).

This transfers serial data from microcontroller to display.

@param sclk\_pin

SCLK (serial clock) pin (using Arduino pin numbering).

This clocks each bit from MOSI.

@param dc\_pin

Data/command pin (using Arduino pin numbering), selects whether

display is receiving commands (low) or data (high).

@param rst\_pin

Reset pin (using Arduino pin numbering), or -1 if not used

(some displays might be wired to share the microcontroller's

reset pin).

@param cs\_pin

Chip-select pin (using Arduino pin numbering) for sharing the

bus with other devices. Active low.

@note Call the object's begin() function before use -- buffer

allocation is performed there!

\*/

Adafruit\_GrayOLED::Adafruit\_GrayOLED(uint8\_t bpp, uint16\_t w, uint16\_t h,

int8\_t mosi\_pin, int8\_t sclk\_pin,

int8\_t dc\_pin, int8\_t rst\_pin,

int8\_t cs\_pin)

: Adafruit\_GFX(w, h), dcPin(dc\_pin), csPin(cs\_pin), rstPin(rst\_pin),

\_bpp(bpp) {

spi\_dev = new Adafruit\_SPIDevice(cs\_pin, sclk\_pin, -1, mosi\_pin, 1000000);

}

/\*!

@brief Constructor for SPI GrayOLED displays, using native hardware SPI.

@param bpp Bits per pixel, 1 for monochrome, 4 for 16-gray

@param w

Display width in pixels

@param h

Display height in pixels

@param spi

Pointer to an existing SPIClass instance (e.g. &SPI, the

microcontroller's primary SPI bus).

@param dc\_pin

Data/command pin (using Arduino pin numbering), selects whether

display is receiving commands (low) or data (high).

@param rst\_pin

Reset pin (using Arduino pin numbering), or -1 if not used

(some displays might be wired to share the microcontroller's

reset pin).

@param cs\_pin

Chip-select pin (using Arduino pin numbering) for sharing the

bus with other devices. Active low.

@param bitrate

SPI clock rate for transfers to this display. Default if

unspecified is 8000000UL (8 MHz).

@note Call the object's begin() function before use -- buffer

allocation is performed there!

\*/

Adafruit\_GrayOLED::Adafruit\_GrayOLED(uint8\_t bpp, uint16\_t w, uint16\_t h,

SPIClass \*spi, int8\_t dc\_pin,

int8\_t rst\_pin, int8\_t cs\_pin,

uint32\_t bitrate)

: Adafruit\_GFX(w, h), dcPin(dc\_pin), csPin(cs\_pin), rstPin(rst\_pin),

\_bpp(bpp) {

spi\_dev = new Adafruit\_SPIDevice(cs\_pin, bitrate, SPI\_BITORDER\_MSBFIRST,

SPI\_MODE0, spi);

}

/\*!

@brief Destructor for Adafruit\_GrayOLED object.

\*/

Adafruit\_GrayOLED::~Adafruit\_GrayOLED(void) {

if (buffer) {

free(buffer);

buffer = NULL;

}

if (spi\_dev)

delete spi\_dev;

if (i2c\_dev)

delete i2c\_dev;

}

// LOW-LEVEL UTILS ---------------------------------------------------------

/\*!

@brief Issue single command byte to OLED, using I2C or hard/soft SPI as

needed.

@param c The single byte command

\*/

void Adafruit\_GrayOLED::oled\_command(uint8\_t c) {

if (i2c\_dev) { // I2C

uint8\_t buf[2] = {0x00, c}; // Co = 0, D/C = 0

i2c\_dev->write(buf, 2);

} else { // SPI (hw or soft) -- transaction started in calling function

digitalWrite(dcPin, LOW);

spi\_dev->write(&c, 1);

}

}

// Issue list of commands to GrayOLED

/\*!

@brief Issue multiple bytes of commands OLED, using I2C or hard/soft SPI as

needed.

@param c Pointer to the command array

@param n The number of bytes in the command array

@returns True for success on ability to write the data in I2C.

\*/

bool Adafruit\_GrayOLED::oled\_commandList(const uint8\_t \*c, uint8\_t n) {

if (i2c\_dev) { // I2C

uint8\_t dc\_byte = 0x00; // Co = 0, D/C = 0

if (!i2c\_dev->write((uint8\_t \*)c, n, true, &dc\_byte, 1)) {

return false;

}

} else { // SPI -- transaction started in calling function

digitalWrite(dcPin, LOW);

if (!spi\_dev->write((uint8\_t \*)c, n)) {

return false;

}

}

return true;

}

// ALLOCATE & INIT DISPLAY -------------------------------------------------

/\*!

@brief Allocate RAM for image buffer, initialize peripherals and pins.

Note that subclasses must call this before other begin() init

@param addr

I2C address of corresponding oled display.

SPI displays (hardware or software) do not use addresses, but

this argument is still required. Default if unspecified is 0x3C.

@param reset

If true, and if the reset pin passed to the constructor is

valid, a hard reset will be performed before initializing the

display. If using multiple oled displays on the same bus, and

if they all share the same reset pin, you should only pass true

on the first display being initialized, false on all others,

else the already-initialized displays would be reset. Default if

unspecified is true.

@return true on successful allocation/init, false otherwise.

Well-behaved code should check the return value before

proceeding.

@note MUST call this function before any drawing or updates!

\*/

bool Adafruit\_GrayOLED::\_init(uint8\_t addr, bool reset) {

// attempt to malloc the bitmap framebuffer

if ((!buffer) &&

!(buffer = (uint8\_t \*)malloc(\_bpp \* WIDTH \* ((HEIGHT + 7) / 8)))) {

return false;

}

// Reset OLED if requested and reset pin specified in constructor

if (reset && (rstPin >= 0)) {

pinMode(rstPin, OUTPUT);

digitalWrite(rstPin, HIGH);

delay(10); // VDD goes high at start, pause

digitalWrite(rstPin, LOW); // Bring reset low

delay(10); // Wait 10 ms

digitalWrite(rstPin, HIGH); // Bring out of reset

delay(10);

}

// Setup pin directions

if (\_theWire) { // using I2C

i2c\_dev = new Adafruit\_I2CDevice(addr, \_theWire);

// look for i2c address:

if (!i2c\_dev || !i2c\_dev->begin()) {

return false;

}

} else { // Using one of the SPI modes, either soft or hardware

if (!spi\_dev || !spi\_dev->begin()) {

return false;

}

pinMode(dcPin, OUTPUT); // Set data/command pin as output

}

clearDisplay();

// set max dirty window

window\_x1 = 0;

window\_y1 = 0;

window\_x2 = WIDTH - 1;

window\_y2 = HEIGHT - 1;

return true; // Success

}

// DRAWING FUNCTIONS -------------------------------------------------------

/\*!

@brief Set/clear/invert a single pixel. This is also invoked by the

Adafruit\_GFX library in generating many higher-level graphics

primitives.

@param x

Column of display -- 0 at left to (screen width - 1) at right.

@param y

Row of display -- 0 at top to (screen height -1) at bottom.

@param color

Pixel color, one of: MONOOLED\_BLACK, MONOOLED\_WHITE or

MONOOLED\_INVERT.

@note Changes buffer contents only, no immediate effect on display.

Follow up with a call to display(), or with other graphics

commands as needed by one's own application.

\*/

void Adafruit\_GrayOLED::drawPixel(int16\_t x, int16\_t y, uint16\_t color) {

if ((x >= 0) && (x < width()) && (y >= 0) && (y < height())) {

// Pixel is in-bounds. Rotate coordinates if needed.

switch (getRotation()) {

case 1:

grayoled\_swap(x, y);

x = WIDTH - x - 1;

break;

case 2:

x = WIDTH - x - 1;

y = HEIGHT - y - 1;

break;

case 3:

grayoled\_swap(x, y);

y = HEIGHT - y - 1;

break;

}

// adjust dirty window

window\_x1 = min(window\_x1, x);

window\_y1 = min(window\_y1, y);

window\_x2 = max(window\_x2, x);

window\_y2 = max(window\_y2, y);

if (\_bpp == 1) {

switch (color) {

case MONOOLED\_WHITE:

buffer[x + (y / 8) \* WIDTH] |= (1 << (y & 7));

break;

case MONOOLED\_BLACK:

buffer[x + (y / 8) \* WIDTH] &= ~(1 << (y & 7));

break;

case MONOOLED\_INVERSE:

buffer[x + (y / 8) \* WIDTH] ^= (1 << (y & 7));

break;

}

}

if (\_bpp == 4) {

uint8\_t \*pixelptr = &buffer[x / 2 + (y \* WIDTH / 2)];

// Serial.printf("(%d, %d) -> offset %d\n", x, y, x/2 + (y \* WIDTH / 2));

if (x % 2 == 0) { // even, left nibble

uint8\_t t = pixelptr[0] & 0x0F;

t |= (color & 0xF) << 4;

pixelptr[0] = t;

} else { // odd, right lower nibble

uint8\_t t = pixelptr[0] & 0xF0;

t |= color & 0xF;

pixelptr[0] = t;

}

}

}

}

/\*!

@brief Clear contents of display buffer (set all pixels to off).

@note Changes buffer contents only, no immediate effect on display.

Follow up with a call to display(), or with other graphics

commands as needed by one's own application.

\*/

void Adafruit\_GrayOLED::clearDisplay(void) {

memset(buffer, 0, \_bpp \* WIDTH \* ((HEIGHT + 7) / 8));

// set max dirty window

window\_x1 = 0;

window\_y1 = 0;

window\_x2 = WIDTH - 1;

window\_y2 = HEIGHT - 1;

}

/\*!

@brief Return color of a single pixel in display buffer.

@param x

Column of display -- 0 at left to (screen width - 1) at right.

@param y

Row of display -- 0 at top to (screen height -1) at bottom.

@return true if pixel is set (usually MONOOLED\_WHITE, unless display invert

mode is enabled), false if clear (MONOOLED\_BLACK).

@note Reads from buffer contents; may not reflect current contents of

screen if display() has not been called.

\*/

bool Adafruit\_GrayOLED::getPixel(int16\_t x, int16\_t y) {

if ((x >= 0) && (x < width()) && (y >= 0) && (y < height())) {

// Pixel is in-bounds. Rotate coordinates if needed.

switch (getRotation()) {

case 1:

grayoled\_swap(x, y);

x = WIDTH - x - 1;

break;

case 2:

x = WIDTH - x - 1;

y = HEIGHT - y - 1;

break;

case 3:

grayoled\_swap(x, y);

y = HEIGHT - y - 1;

break;

}

return (buffer[x + (y / 8) \* WIDTH] & (1 << (y & 7)));

}

return false; // Pixel out of bounds

}

/\*!

@brief Get base address of display buffer for direct reading or writing.

@return Pointer to an unsigned 8-bit array, column-major, columns padded

to full byte boundary if needed.

\*/

uint8\_t \*Adafruit\_GrayOLED::getBuffer(void) { return buffer; }

// OTHER HARDWARE SETTINGS -------------------------------------------------

/\*!

@brief Enable or disable display invert mode (white-on-black vs

black-on-white). Handy for testing!

@param i

If true, switch to invert mode (black-on-white), else normal

mode (white-on-black).

@note This has an immediate effect on the display, no need to call the

display() function -- buffer contents are not changed, rather a

different pixel mode of the display hardware is used. When

enabled, drawing MONOOLED\_BLACK (value 0) pixels will actually draw

white, MONOOLED\_WHITE (value 1) will draw black.

\*/

void Adafruit\_GrayOLED::invertDisplay(bool i) {

oled\_command(i ? GRAYOLED\_INVERTDISPLAY : GRAYOLED\_NORMALDISPLAY);

}

/\*!

@brief Adjust the display contrast.

@param level The contrast level from 0 to 0x7F

@note This has an immediate effect on the display, no need to call the

display() function -- buffer contents are not changed.

\*/

void Adafruit\_GrayOLED::setContrast(uint8\_t level) {

uint8\_t cmd[] = {GRAYOLED\_SETCONTRAST, level};

oled\_commandList(cmd, 2);

}

#endif /\* ATTIN85 not supported \*/