/\*!

\* @file Adafruit\_SSD1306.cpp

\*

\* @mainpage Arduino library for monochrome OLEDs based on SSD1306 drivers.

\*

\* @section intro\_sec Introduction

\*

\* This is documentation for Adafruit's SSD1306 library for monochrome

\* 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!

\*

\* @section dependencies Dependencies

\*

\* This library depends on <a

\* href="https://github.com/adafruit/Adafruit-GFX-Library"> Adafruit\_GFX</a>

\* being present on your system. Please make sure you have installed the latest

\* version before using this library.

\*

\* @section author Author

\*

\* Written by Limor Fried/Ladyada for Adafruit Industries, with

\* contributions from the open source community.

\*

\* @section license License

\*

\* BSD license, all text above, and the splash screen included below,

\* must be included in any redistribution.

\*

\*/

#ifdef \_\_AVR\_\_

#include <avr/pgmspace.h>

#elif defined(ESP8266) || defined(ESP32) || defined(ARDUINO\_ARCH\_RP2040)

#include <pgmspace.h>

#else

#define pgm\_read\_byte(addr) \

(\*(const unsigned char \*)(addr)) ///< PROGMEM workaround for non-AVR

#endif

#if !defined(\_\_ARM\_ARCH) && !defined(ENERGIA) && !defined(ESP8266) && \

!defined(ESP32) && !defined(\_\_arc\_\_)

#include <util/delay.h>

#endif

#include "Adafruit\_SSD1306.h"

#include "splash.h"

#include <Adafruit\_GFX.h>

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

#if defined(I2C\_BUFFER\_LENGTH)

#define WIRE\_MAX min(256, I2C\_BUFFER\_LENGTH) ///< Particle or similar Wire lib

#elif defined(BUFFER\_LENGTH)

#define WIRE\_MAX min(256, BUFFER\_LENGTH) ///< AVR or similar Wire lib

#elif defined(SERIAL\_BUFFER\_SIZE)

#define WIRE\_MAX \

min(255, SERIAL\_BUFFER\_SIZE - 1) ///< Newer Wire uses RingBuffer

#else

#define WIRE\_MAX 32 ///< Use common Arduino core default

#endif

#define ssd1306\_swap(a, b) \

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

#if ARDUINO >= 100

#define WIRE\_WRITE wire->write ///< Wire write function in recent Arduino lib

#else

#define WIRE\_WRITE wire->send ///< Wire write function in older Arduino lib

#endif

#ifdef HAVE\_PORTREG

#define SSD1306\_SELECT \*csPort &= ~csPinMask; ///< Device select

#define SSD1306\_DESELECT \*csPort |= csPinMask; ///< Device deselect

#define SSD1306\_MODE\_COMMAND \*dcPort &= ~dcPinMask; ///< Command mode

#define SSD1306\_MODE\_DATA \*dcPort |= dcPinMask; ///< Data mode

#else

#define SSD1306\_SELECT digitalWrite(csPin, LOW); ///< Device select

#define SSD1306\_DESELECT digitalWrite(csPin, HIGH); ///< Device deselect

#define SSD1306\_MODE\_COMMAND digitalWrite(dcPin, LOW); ///< Command mode

#define SSD1306\_MODE\_DATA digitalWrite(dcPin, HIGH); ///< Data mode

#endif

#if (ARDUINO >= 157) && !defined(ARDUINO\_STM32\_FEATHER)

#define SETWIRECLOCK wire->setClock(wireClk) ///< Set before I2C transfer

#define RESWIRECLOCK wire->setClock(restoreClk) ///< Restore after I2C xfer

#else // setClock() is not present in older Arduino Wire lib (or WICED)

#define SETWIRECLOCK ///< Dummy stand-in define

#define RESWIRECLOCK ///< keeps compiler happy

#endif

#if defined(SPI\_HAS\_TRANSACTION)

#define SPI\_TRANSACTION\_START spi->beginTransaction(spiSettings) ///< Pre-SPI

#define SPI\_TRANSACTION\_END spi->endTransaction() ///< Post-SPI

#else // SPI transactions likewise not present in older Arduino SPI lib

#define SPI\_TRANSACTION\_START ///< Dummy stand-in define

#define SPI\_TRANSACTION\_END ///< keeps compiler happy

#endif

// The definition of 'transaction' is broadened a bit in the context of

// this library -- referring not just to SPI transactions (if supported

// in the version of the SPI library being used), but also chip select

// (if SPI is being used, whether hardware or soft), and also to the

// beginning and end of I2C transfers (the Wire clock may be sped up before

// issuing data to the display, then restored to the default rate afterward

// so other I2C device types still work). All of these are encapsulated

// in the TRANSACTION\_\* macros.

// Check first if Wire, then hardware SPI, then soft SPI:

#define TRANSACTION\_START \

if (wire) { \

SETWIRECLOCK; \

} else { \

if (spi) { \

SPI\_TRANSACTION\_START; \

} \

SSD1306\_SELECT; \

} ///< Wire, SPI or bitbang transfer setup

#define TRANSACTION\_END \

if (wire) { \

RESWIRECLOCK; \

} else { \

SSD1306\_DESELECT; \

if (spi) { \

SPI\_TRANSACTION\_END; \

} \

} ///< Wire, SPI or bitbang transfer end

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

/\*!

@brief Constructor for I2C-interfaced SSD1306 displays.

@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 SSD1306 library calls.

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

microcontrollers, and meets the SSD1306 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)

SSD1306'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 SSD1306 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.)

@return Adafruit\_SSD1306 object.

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

allocation is performed there!

\*/

Adafruit\_SSD1306::Adafruit\_SSD1306(uint8\_t w, uint8\_t h, TwoWire \*twi,

int8\_t rst\_pin, uint32\_t clkDuring,

uint32\_t clkAfter)

: Adafruit\_GFX(w, h), spi(NULL), wire(twi ? twi : &Wire), buffer(NULL),

mosiPin(-1), clkPin(-1), dcPin(-1), csPin(-1), rstPin(rst\_pin)

#if ARDUINO >= 157

,

wireClk(clkDuring), restoreClk(clkAfter)

#endif

{

}

/\*!

@brief Constructor for SPI SSD1306 displays, using software (bitbang)

SPI.

@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.

@return Adafruit\_SSD1306 object.

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

allocation is performed there!

\*/

Adafruit\_SSD1306::Adafruit\_SSD1306(uint8\_t w, uint8\_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), spi(NULL), wire(NULL), buffer(NULL),

mosiPin(mosi\_pin), clkPin(sclk\_pin), dcPin(dc\_pin), csPin(cs\_pin),

rstPin(rst\_pin) {}

/\*!

@brief Constructor for SPI SSD1306 displays, using native hardware SPI.

@param w

Display width in pixels

@param h

Display height in pixels

@param spi\_ptr

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).

@return Adafruit\_SSD1306 object.

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

allocation is performed there!

\*/

Adafruit\_SSD1306::Adafruit\_SSD1306(uint8\_t w, uint8\_t h, SPIClass \*spi\_ptr,

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

uint32\_t bitrate)

: Adafruit\_GFX(w, h), spi(spi\_ptr ? spi\_ptr : &SPI), wire(NULL),

buffer(NULL), mosiPin(-1), clkPin(-1), dcPin(dc\_pin), csPin(cs\_pin),

rstPin(rst\_pin) {

#ifdef SPI\_HAS\_TRANSACTION

spiSettings = SPISettings(bitrate, MSBFIRST, SPI\_MODE0);

#endif

}

/\*!

@brief DEPRECATED constructor for SPI SSD1306 displays, using software

(bitbang) SPI. Provided for older code to maintain compatibility

with the current library. Screen size is determined by enabling

one of the SSD1306\_\* size defines in Adafruit\_SSD1306.h. New

code should NOT use this.

@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.

@return Adafruit\_SSD1306 object.

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

allocation is performed there!

\*/

Adafruit\_SSD1306::Adafruit\_SSD1306(int8\_t mosi\_pin, int8\_t sclk\_pin,

int8\_t dc\_pin, int8\_t rst\_pin, int8\_t cs\_pin)

: Adafruit\_GFX(SSD1306\_LCDWIDTH, SSD1306\_LCDHEIGHT), spi(NULL), wire(NULL),

buffer(NULL), mosiPin(mosi\_pin), clkPin(sclk\_pin), dcPin(dc\_pin),

csPin(cs\_pin), rstPin(rst\_pin) {}

/\*!

@brief DEPRECATED constructor for SPI SSD1306 displays, using native

hardware SPI. Provided for older code to maintain compatibility

with the current library. Screen size is determined by enabling

one of the SSD1306\_\* size defines in Adafruit\_SSD1306.h. New

code should NOT use this. Only the primary SPI bus is supported,

and bitrate is fixed at 8 MHz.

@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.

@return Adafruit\_SSD1306 object.

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

allocation is performed there!

\*/

Adafruit\_SSD1306::Adafruit\_SSD1306(int8\_t dc\_pin, int8\_t rst\_pin, int8\_t cs\_pin)

: Adafruit\_GFX(SSD1306\_LCDWIDTH, SSD1306\_LCDHEIGHT), spi(&SPI), wire(NULL),

buffer(NULL), mosiPin(-1), clkPin(-1), dcPin(dc\_pin), csPin(cs\_pin),

rstPin(rst\_pin) {

#ifdef SPI\_HAS\_TRANSACTION

spiSettings = SPISettings(8000000, MSBFIRST, SPI\_MODE0);

#endif

}

/\*!

@brief DEPRECATED constructor for I2C SSD1306 displays. Provided for

older code to maintain compatibility with the current library.

Screen size is determined by enabling one of the SSD1306\_\* size

defines in Adafruit\_SSD1306.h. New code should NOT use this.

Only the primary I2C bus is supported.

@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).

@return Adafruit\_SSD1306 object.

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

allocation is performed there!

\*/

Adafruit\_SSD1306::Adafruit\_SSD1306(int8\_t rst\_pin)

: Adafruit\_GFX(SSD1306\_LCDWIDTH, SSD1306\_LCDHEIGHT), spi(NULL), wire(&Wire),

buffer(NULL), mosiPin(-1), clkPin(-1), dcPin(-1), csPin(-1),

rstPin(rst\_pin) {}

/\*!

@brief Destructor for Adafruit\_SSD1306 object.

\*/

Adafruit\_SSD1306::~Adafruit\_SSD1306(void) {

if (buffer) {

free(buffer);

buffer = NULL;

}

}

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

// Issue single byte out SPI, either soft or hardware as appropriate.

// SPI transaction/selection must be performed in calling function.

/\*!

@brief Write a single byte to the SPI port.

@param d

Data byte to be written.

@return void

@note See HAVE\_PORTREG which defines if the method uses a port or bit-bang

method

\*/

inline void Adafruit\_SSD1306::SPIwrite(uint8\_t d) {

if (spi) {

(void)spi->transfer(d);

} else {

for (uint8\_t bit = 0x80; bit; bit >>= 1) {

#ifdef HAVE\_PORTREG

if (d & bit)

\*mosiPort |= mosiPinMask;

else

\*mosiPort &= ~mosiPinMask;

\*clkPort |= clkPinMask; // Clock high

\*clkPort &= ~clkPinMask; // Clock low

#else

digitalWrite(mosiPin, d & bit);

digitalWrite(clkPin, HIGH);

digitalWrite(clkPin, LOW);

#endif

}

}

}

/\*!

@brief Issue single command to SSD1306, using I2C or hard/soft SPI as

needed. Because command calls are often grouped, SPI transaction and

selection must be started/ended in calling function for efficiency. This is a

protected function, not exposed (see ssd1306\_command() instead).

@param c

the command character to send to the display.

Refer to ssd1306 data sheet for commands

@return None (void).

@note

\*/

void Adafruit\_SSD1306::ssd1306\_command1(uint8\_t c) {

if (wire) { // I2C

wire->beginTransmission(i2caddr);

WIRE\_WRITE((uint8\_t)0x00); // Co = 0, D/C = 0

WIRE\_WRITE(c);

wire->endTransmission();

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

SSD1306\_MODE\_COMMAND

SPIwrite(c);

}

}

/\*!

@brief Issue list of commands to SSD1306, same rules as above re:

transactions. This is a protected function, not exposed.

@param c

pointer to list of commands

@param n

number of commands in the list

@return None (void).

@note

\*/

void Adafruit\_SSD1306::ssd1306\_commandList(const uint8\_t \*c, uint8\_t n) {

if (wire) { // I2C

wire->beginTransmission(i2caddr);

WIRE\_WRITE((uint8\_t)0x00); // Co = 0, D/C = 0

uint16\_t bytesOut = 1;

while (n--) {

if (bytesOut >= WIRE\_MAX) {

wire->endTransmission();

wire->beginTransmission(i2caddr);

WIRE\_WRITE((uint8\_t)0x00); // Co = 0, D/C = 0

bytesOut = 1;

}

WIRE\_WRITE(pgm\_read\_byte(c++));

bytesOut++;

}

wire->endTransmission();

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

SSD1306\_MODE\_COMMAND

while (n--)

SPIwrite(pgm\_read\_byte(c++));

}

}

// A public version of ssd1306\_command1(), for existing user code that

// might rely on that function. This encapsulates the command transfer

// in a transaction start/end, similar to old library's handling of it.

/\*!

@brief Issue a single low-level command directly to the SSD1306

display, bypassing the library.

@param c

Command to issue (0x00 to 0xFF, see datasheet).

@return None (void).

\*/

void Adafruit\_SSD1306::ssd1306\_command(uint8\_t c) {

TRANSACTION\_START

ssd1306\_command1(c);

TRANSACTION\_END

}

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

/\*!

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

@param vcs

VCC selection. Pass SSD1306\_SWITCHCAPVCC to generate the display

voltage (step up) from the 3.3V source, or SSD1306\_EXTERNALVCC

otherwise. Most situations with Adafruit SSD1306 breakouts will

want SSD1306\_SWITCHCAPVCC.

@param addr

I2C address of corresponding SSD1306 display (or pass 0 to use

default of 0x3C for 128x32 display, 0x3D for all others).

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

this argument is still required (pass 0 or any value really,

it will simply be ignored). Default if unspecified is 0.

@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 SSD1306 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.

@param periphBegin

If true, and if a hardware peripheral is being used (I2C or SPI,

but not software SPI), call that peripheral's begin() function,

else (false) it has already been done in one's sketch code.

Cases where false might be used include multiple displays or

other devices sharing a common bus, or situations on some

platforms where a nonstandard begin() function is available

(e.g. a TwoWire interface on non-default pins, as can be done

on the ESP8266 and perhaps others).

@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\_SSD1306::begin(uint8\_t vcs, uint8\_t addr, bool reset,

bool periphBegin) {

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

return false;

clearDisplay();

#ifndef SSD1306\_NO\_SPLASH

if (HEIGHT > 32) {

drawBitmap((WIDTH - splash1\_width) / 2, (HEIGHT - splash1\_height) / 2,

splash1\_data, splash1\_width, splash1\_height, 1);

} else {

drawBitmap((WIDTH - splash2\_width) / 2, (HEIGHT - splash2\_height) / 2,

splash2\_data, splash2\_width, splash2\_height, 1);

}

#endif

vccstate = vcs;

// Setup pin directions

if (wire) { // Using I2C

// If I2C address is unspecified, use default

// (0x3C for 32-pixel-tall displays, 0x3D for all others).

i2caddr = addr ? addr : ((HEIGHT == 32) ? 0x3C : 0x3D);

// TwoWire begin() function might be already performed by the calling

// function if it has unusual circumstances (e.g. TWI variants that

// can accept different SDA/SCL pins, or if two SSD1306 instances

// with different addresses -- only a single begin() is needed).

if (periphBegin)

wire->begin();

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

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

pinMode(csPin, OUTPUT); // Same for chip select

#ifdef HAVE\_PORTREG

dcPort = (PortReg \*)portOutputRegister(digitalPinToPort(dcPin));

dcPinMask = digitalPinToBitMask(dcPin);

csPort = (PortReg \*)portOutputRegister(digitalPinToPort(csPin));

csPinMask = digitalPinToBitMask(csPin);

#endif

SSD1306\_DESELECT

if (spi) { // Hardware SPI

// SPI peripheral begin same as wire check above.

if (periphBegin)

spi->begin();

} else { // Soft SPI

pinMode(mosiPin, OUTPUT); // MOSI and SCLK outputs

pinMode(clkPin, OUTPUT);

#ifdef HAVE\_PORTREG

mosiPort = (PortReg \*)portOutputRegister(digitalPinToPort(mosiPin));

mosiPinMask = digitalPinToBitMask(mosiPin);

clkPort = (PortReg \*)portOutputRegister(digitalPinToPort(clkPin));

clkPinMask = digitalPinToBitMask(clkPin);

\*clkPort &= ~clkPinMask; // Clock low

#else

digitalWrite(clkPin, LOW); // Clock low

#endif

}

}

// Reset SSD1306 if requested and reset pin specified in constructor

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

pinMode(rstPin, OUTPUT);

digitalWrite(rstPin, HIGH);

delay(1); // VDD goes high at start, pause for 1 ms

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

delay(10); // Wait 10 ms

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

}

TRANSACTION\_START

// Init sequence

static const uint8\_t PROGMEM init1[] = {SSD1306\_DISPLAYOFF, // 0xAE

SSD1306\_SETDISPLAYCLOCKDIV, // 0xD5

0x80, // the suggested ratio 0x80

SSD1306\_SETMULTIPLEX}; // 0xA8

ssd1306\_commandList(init1, sizeof(init1));

ssd1306\_command1(HEIGHT - 1);

static const uint8\_t PROGMEM init2[] = {SSD1306\_SETDISPLAYOFFSET, // 0xD3

0x0, // no offset

SSD1306\_SETSTARTLINE | 0x0, // line #0

SSD1306\_CHARGEPUMP}; // 0x8D

ssd1306\_commandList(init2, sizeof(init2));

ssd1306\_command1((vccstate == SSD1306\_EXTERNALVCC) ? 0x10 : 0x14);

static const uint8\_t PROGMEM init3[] = {SSD1306\_MEMORYMODE, // 0x20

0x00, // 0x0 act like ks0108

SSD1306\_SEGREMAP | 0x1,

SSD1306\_COMSCANDEC};

ssd1306\_commandList(init3, sizeof(init3));

uint8\_t comPins = 0x02;

contrast = 0x8F;

if ((WIDTH == 128) && (HEIGHT == 32)) {

comPins = 0x02;

contrast = 0x8F;

} else if ((WIDTH == 128) && (HEIGHT == 64)) {

comPins = 0x12;

contrast = (vccstate == SSD1306\_EXTERNALVCC) ? 0x9F : 0xCF;

} else if ((WIDTH == 96) && (HEIGHT == 16)) {

comPins = 0x2; // ada x12

contrast = (vccstate == SSD1306\_EXTERNALVCC) ? 0x10 : 0xAF;

} else {

// Other screen varieties -- TBD

}

ssd1306\_command1(SSD1306\_SETCOMPINS);

ssd1306\_command1(comPins);

ssd1306\_command1(SSD1306\_SETCONTRAST);

ssd1306\_command1(contrast);

ssd1306\_command1(SSD1306\_SETPRECHARGE); // 0xd9

ssd1306\_command1((vccstate == SSD1306\_EXTERNALVCC) ? 0x22 : 0xF1);

static const uint8\_t PROGMEM init5[] = {

SSD1306\_SETVCOMDETECT, // 0xDB

0x40,

SSD1306\_DISPLAYALLON\_RESUME, // 0xA4

SSD1306\_NORMALDISPLAY, // 0xA6

SSD1306\_DEACTIVATE\_SCROLL,

SSD1306\_DISPLAYON}; // Main screen turn on

ssd1306\_commandList(init5, sizeof(init5));

TRANSACTION\_END

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: SSD1306\_BLACK, SSD1306\_WHITE or

SSD1306\_INVERSE.

@return None (void).

@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\_SSD1306::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:

ssd1306\_swap(x, y);

x = WIDTH - x - 1;

break;

case 2:

x = WIDTH - x - 1;

y = HEIGHT - y - 1;

break;

case 3:

ssd1306\_swap(x, y);

y = HEIGHT - y - 1;

break;

}

switch (color) {

case SSD1306\_WHITE:

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

break;

case SSD1306\_BLACK:

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

break;

case SSD1306\_INVERSE:

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

break;

}

}

}

/\*!

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

@return None (void).

@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\_SSD1306::clearDisplay(void) {

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

}

/\*!

@brief Draw a horizontal line. This is also invoked by the Adafruit\_GFX

library in generating many higher-level graphics primitives.

@param x

Leftmost column -- 0 at left to (screen width - 1) at right.

@param y

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

@param w

Width of line, in pixels.

@param color

Line color, one of: SSD1306\_BLACK, SSD1306\_WHITE or SSD1306\_INVERSE.

@return None (void).

@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\_SSD1306::drawFastHLine(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

bool bSwap = false;

switch (rotation) {

case 1:

// 90 degree rotation, swap x & y for rotation, then invert x

bSwap = true;

ssd1306\_swap(x, y);

x = WIDTH - x - 1;

break;

case 2:

// 180 degree rotation, invert x and y, then shift y around for height.

x = WIDTH - x - 1;

y = HEIGHT - y - 1;

x -= (w - 1);

break;

case 3:

// 270 degree rotation, swap x & y for rotation,

// then invert y and adjust y for w (not to become h)

bSwap = true;

ssd1306\_swap(x, y);

y = HEIGHT - y - 1;

y -= (w - 1);

break;

}

if (bSwap)

drawFastVLineInternal(x, y, w, color);

else

drawFastHLineInternal(x, y, w, color);

}

/\*!

@brief Draw a horizontal line with a width and color. Used by public

methods drawFastHLine,drawFastVLine

@param x

Leftmost column -- 0 at left to (screen width - 1) at right.

@param y

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

@param w

Width of line, in pixels.

@param color

Line color, one of: SSD1306\_BLACK, SSD1306\_WHITE or

SSD1306\_INVERSE.

@return None (void).

@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\_SSD1306::drawFastHLineInternal(int16\_t x, int16\_t y, int16\_t w,

uint16\_t color) {

if ((y >= 0) && (y < HEIGHT)) { // Y coord in bounds?

if (x < 0) { // Clip left

w += x;

x = 0;

}

if ((x + w) > WIDTH) { // Clip right

w = (WIDTH - x);

}

if (w > 0) { // Proceed only if width is positive

uint8\_t \*pBuf = &buffer[(y / 8) \* WIDTH + x], mask = 1 << (y & 7);

switch (color) {

case SSD1306\_WHITE:

while (w--) {

\*pBuf++ |= mask;

};

break;

case SSD1306\_BLACK:

mask = ~mask;

while (w--) {

\*pBuf++ &= mask;

};

break;

case SSD1306\_INVERSE:

while (w--) {

\*pBuf++ ^= mask;

};

break;

}

}

}

}

/\*!

@brief Draw a vertical line. 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

Topmost row -- 0 at top to (screen height - 1) at bottom.

@param h

Height of line, in pixels.

@param color

Line color, one of: SSD1306\_BLACK, SSD1306\_WHITE or SSD1306\_INVERSE.

@return None (void).

@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\_SSD1306::drawFastVLine(int16\_t x, int16\_t y, int16\_t h,

uint16\_t color) {

bool bSwap = false;

switch (rotation) {

case 1:

// 90 degree rotation, swap x & y for rotation,

// then invert x and adjust x for h (now to become w)

bSwap = true;

ssd1306\_swap(x, y);

x = WIDTH - x - 1;

x -= (h - 1);

break;

case 2:

// 180 degree rotation, invert x and y, then shift y around for height.

x = WIDTH - x - 1;

y = HEIGHT - y - 1;

y -= (h - 1);

break;

case 3:

// 270 degree rotation, swap x & y for rotation, then invert y

bSwap = true;

ssd1306\_swap(x, y);

y = HEIGHT - y - 1;

break;

}

if (bSwap)

drawFastHLineInternal(x, y, h, color);

else

drawFastVLineInternal(x, y, h, color);

}

/\*!

@brief Draw a vertical line with a width and color. Used by public method

drawFastHLine,drawFastVLine

@param x

Leftmost column -- 0 at left to (screen width - 1) at right.

@param \_\_y

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

@param \_\_h height of the line in pixels

@param color

Line color, one of: SSD1306\_BLACK, SSD1306\_WHITE or

SSD1306\_INVERSE.

@return None (void).

@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\_SSD1306::drawFastVLineInternal(int16\_t x, int16\_t \_\_y,

int16\_t \_\_h, uint16\_t color) {

if ((x >= 0) && (x < WIDTH)) { // X coord in bounds?

if (\_\_y < 0) { // Clip top

\_\_h += \_\_y;

\_\_y = 0;

}

if ((\_\_y + \_\_h) > HEIGHT) { // Clip bottom

\_\_h = (HEIGHT - \_\_y);

}

if (\_\_h > 0) { // Proceed only if height is now positive

// this display doesn't need ints for coordinates,

// use local byte registers for faster juggling

uint8\_t y = \_\_y, h = \_\_h;

uint8\_t \*pBuf = &buffer[(y / 8) \* WIDTH + x];

// do the first partial byte, if necessary - this requires some masking

uint8\_t mod = (y & 7);

if (mod) {

// mask off the high n bits we want to set

mod = 8 - mod;

// note - lookup table results in a nearly 10% performance

// improvement in fill\* functions

// uint8\_t mask = ~(0xFF >> mod);

static const uint8\_t PROGMEM premask[8] = {0x00, 0x80, 0xC0, 0xE0,

0xF0, 0xF8, 0xFC, 0xFE};

uint8\_t mask = pgm\_read\_byte(&premask[mod]);

// adjust the mask if we're not going to reach the end of this byte

if (h < mod)

mask &= (0XFF >> (mod - h));

switch (color) {

case SSD1306\_WHITE:

\*pBuf |= mask;

break;

case SSD1306\_BLACK:

\*pBuf &= ~mask;

break;

case SSD1306\_INVERSE:

\*pBuf ^= mask;

break;

}

pBuf += WIDTH;

}

if (h >= mod) { // More to go?

h -= mod;

// Write solid bytes while we can - effectively 8 rows at a time

if (h >= 8) {

if (color == SSD1306\_INVERSE) {

// separate copy of the code so we don't impact performance of

// black/white write version with an extra comparison per loop

do {

\*pBuf ^= 0xFF; // Invert byte

pBuf += WIDTH; // Advance pointer 8 rows

h -= 8; // Subtract 8 rows from height

} while (h >= 8);

} else {

// store a local value to work with

uint8\_t val = (color != SSD1306\_BLACK) ? 255 : 0;

do {

\*pBuf = val; // Set byte

pBuf += WIDTH; // Advance pointer 8 rows

h -= 8; // Subtract 8 rows from height

} while (h >= 8);

}

}

if (h) { // Do the final partial byte, if necessary

mod = h & 7;

// this time we want to mask the low bits of the byte,

// vs the high bits we did above

// uint8\_t mask = (1 << mod) - 1;

// note - lookup table results in a nearly 10% performance

// improvement in fill\* functions

static const uint8\_t PROGMEM postmask[8] = {0x00, 0x01, 0x03, 0x07,

0x0F, 0x1F, 0x3F, 0x7F};

uint8\_t mask = pgm\_read\_byte(&postmask[mod]);

switch (color) {

case SSD1306\_WHITE:

\*pBuf |= mask;

break;

case SSD1306\_BLACK:

\*pBuf &= ~mask;

break;

case SSD1306\_INVERSE:

\*pBuf ^= mask;

break;

}

}

}

} // endif positive height

} // endif x in bounds

}

/\*!

@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 SSD1306\_WHITE, unless display invert

mode is enabled), false if clear (SSD1306\_BLACK).

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

screen if display() has not been called.

\*/

bool Adafruit\_SSD1306::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:

ssd1306\_swap(x, y);

x = WIDTH - x - 1;

break;

case 2:

x = WIDTH - x - 1;

y = HEIGHT - y - 1;

break;

case 3:

ssd1306\_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\_SSD1306::getBuffer(void) { return buffer; }

// REFRESH DISPLAY ---------------------------------------------------------

/\*!

@brief Push data currently in RAM to SSD1306 display.

@return None (void).

@note Drawing operations are not visible until this function is

called. Call after each graphics command, or after a whole set

of graphics commands, as best needed by one's own application.

\*/

void Adafruit\_SSD1306::display(void) {

TRANSACTION\_START

static const uint8\_t PROGMEM dlist1[] = {

SSD1306\_PAGEADDR,

0, // Page start address

0xFF, // Page end (not really, but works here)

SSD1306\_COLUMNADDR, 0}; // Column start address

ssd1306\_commandList(dlist1, sizeof(dlist1));

ssd1306\_command1(WIDTH - 1); // Column end address

#if defined(ESP8266)

// ESP8266 needs a periodic yield() call to avoid watchdog reset.

// With the limited size of SSD1306 displays, and the fast bitrate

// being used (1 MHz or more), I think one yield() immediately before

// a screen write and one immediately after should cover it. But if

// not, if this becomes a problem, yields() might be added in the

// 32-byte transfer condition below.

yield();

#endif

uint16\_t count = WIDTH \* ((HEIGHT + 7) / 8);

uint8\_t \*ptr = buffer;

if (wire) { // I2C

wire->beginTransmission(i2caddr);

WIRE\_WRITE((uint8\_t)0x40);

uint16\_t bytesOut = 1;

while (count--) {

if (bytesOut >= WIRE\_MAX) {

wire->endTransmission();

wire->beginTransmission(i2caddr);

WIRE\_WRITE((uint8\_t)0x40);

bytesOut = 1;

}

WIRE\_WRITE(\*ptr++);

bytesOut++;

}

wire->endTransmission();

} else { // SPI

SSD1306\_MODE\_DATA

while (count--)

SPIwrite(\*ptr++);

}

TRANSACTION\_END

#if defined(ESP8266)

yield();

#endif

}

// SCROLLING FUNCTIONS -----------------------------------------------------

/\*!

@brief Activate a right-handed scroll for all or part of the display.

@param start

First row.

@param stop

Last row.

@return None (void).

\*/

// To scroll the whole display, run: display.startscrollright(0x00, 0x0F)

void Adafruit\_SSD1306::startscrollright(uint8\_t start, uint8\_t stop) {

TRANSACTION\_START

static const uint8\_t PROGMEM scrollList1a[] = {

SSD1306\_RIGHT\_HORIZONTAL\_SCROLL, 0X00};

ssd1306\_commandList(scrollList1a, sizeof(scrollList1a));

ssd1306\_command1(start);

ssd1306\_command1(0X00);

ssd1306\_command1(stop);

static const uint8\_t PROGMEM scrollList1b[] = {0X00, 0XFF,

SSD1306\_ACTIVATE\_SCROLL};

ssd1306\_commandList(scrollList1b, sizeof(scrollList1b));

TRANSACTION\_END

}

/\*!

@brief Activate a left-handed scroll for all or part of the display.

@param start

First row.

@param stop

Last row.

@return None (void).

\*/

// To scroll the whole display, run: display.startscrollleft(0x00, 0x0F)

void Adafruit\_SSD1306::startscrollleft(uint8\_t start, uint8\_t stop) {

TRANSACTION\_START

static const uint8\_t PROGMEM scrollList2a[] = {SSD1306\_LEFT\_HORIZONTAL\_SCROLL,

0X00};

ssd1306\_commandList(scrollList2a, sizeof(scrollList2a));

ssd1306\_command1(start);

ssd1306\_command1(0X00);

ssd1306\_command1(stop);

static const uint8\_t PROGMEM scrollList2b[] = {0X00, 0XFF,

SSD1306\_ACTIVATE\_SCROLL};

ssd1306\_commandList(scrollList2b, sizeof(scrollList2b));

TRANSACTION\_END

}

/\*!

@brief Activate a diagonal scroll for all or part of the display.

@param start

First row.

@param stop

Last row.

@return None (void).

\*/

// display.startscrolldiagright(0x00, 0x0F)

void Adafruit\_SSD1306::startscrolldiagright(uint8\_t start, uint8\_t stop) {

TRANSACTION\_START

static const uint8\_t PROGMEM scrollList3a[] = {

SSD1306\_SET\_VERTICAL\_SCROLL\_AREA, 0X00};

ssd1306\_commandList(scrollList3a, sizeof(scrollList3a));

ssd1306\_command1(HEIGHT);

static const uint8\_t PROGMEM scrollList3b[] = {

SSD1306\_VERTICAL\_AND\_RIGHT\_HORIZONTAL\_SCROLL, 0X00};

ssd1306\_commandList(scrollList3b, sizeof(scrollList3b));

ssd1306\_command1(start);

ssd1306\_command1(0X00);

ssd1306\_command1(stop);

static const uint8\_t PROGMEM scrollList3c[] = {0X01, SSD1306\_ACTIVATE\_SCROLL};

ssd1306\_commandList(scrollList3c, sizeof(scrollList3c));

TRANSACTION\_END

}

/\*!

@brief Activate alternate diagonal scroll for all or part of the display.

@param start

First row.

@param stop

Last row.

@return None (void).

\*/

// To scroll the whole display, run: display.startscrolldiagleft(0x00, 0x0F)

void Adafruit\_SSD1306::startscrolldiagleft(uint8\_t start, uint8\_t stop) {

TRANSACTION\_START

static const uint8\_t PROGMEM scrollList4a[] = {

SSD1306\_SET\_VERTICAL\_SCROLL\_AREA, 0X00};

ssd1306\_commandList(scrollList4a, sizeof(scrollList4a));

ssd1306\_command1(HEIGHT);

static const uint8\_t PROGMEM scrollList4b[] = {

SSD1306\_VERTICAL\_AND\_LEFT\_HORIZONTAL\_SCROLL, 0X00};

ssd1306\_commandList(scrollList4b, sizeof(scrollList4b));

ssd1306\_command1(start);

ssd1306\_command1(0X00);

ssd1306\_command1(stop);

static const uint8\_t PROGMEM scrollList4c[] = {0X01, SSD1306\_ACTIVATE\_SCROLL};

ssd1306\_commandList(scrollList4c, sizeof(scrollList4c));

TRANSACTION\_END

}

/\*!

@brief Cease a previously-begun scrolling action.

@return None (void).

\*/

void Adafruit\_SSD1306::stopscroll(void) {

TRANSACTION\_START

ssd1306\_command1(SSD1306\_DEACTIVATE\_SCROLL);

TRANSACTION\_END

}

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

/\*!

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

black-on-white).

@param i

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

mode (white-on-black).

@return None (void).

@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 SSD1306\_BLACK (value 0) pixels will actually draw

white, SSD1306\_WHITE (value 1) will draw black.

\*/

void Adafruit\_SSD1306::invertDisplay(bool i) {

TRANSACTION\_START

ssd1306\_command1(i ? SSD1306\_INVERTDISPLAY : SSD1306\_NORMALDISPLAY);

TRANSACTION\_END

}

/\*!

@brief Dim the display.

@param dim

true to enable lower brightness mode, false for full brightness.

@return None (void).

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

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

\*/

void Adafruit\_SSD1306::dim(bool dim) {

// the range of contrast to too small to be really useful

// it is useful to dim the display

TRANSACTION\_START

ssd1306\_command1(SSD1306\_SETCONTRAST);

ssd1306\_command1(dim ? 0 : contrast);

TRANSACTION\_END

}