***** graphicstest.ino

```
/***************************************************
  This is our GFX example for the Adafruit ILI9341 Breakout and Shield
  ----> http://www.adafruit.com/products/1651

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!

  Written by Limor Fried/Ladyada for Adafruit Industries.
  MIT license, all text above must be included in any redistribution
 ****************************************************/


#include "SPI.h"
#include "Adafruit_GFX.h"
#include "Adafruit_ILI9341.h"

// For the Adafruit shield, these are the default.
#define TFT_DC 9
#define TFT_CS 10

// Use hardware SPI (on Uno, #13, #12, #11) and the above for CS/DC
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);
// If using the breakout, change pins as desired
//Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_MOSI, TFT_CLK, TFT_RST, TFT_MISO);

void setup() {
  Serial.begin(9600);
  Serial.println("ILI9341 Test!");

  tft.begin();

  // read diagnostics (optional but can help debug problems)
  uint8_t x = tft.readcommand8(ILI9341_RDMODE);
  Serial.print("Display Power Mode: 0x"); Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDMADCTL);
  Serial.print("MADCTL Mode: 0x"); Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDPIXFMT);
  Serial.print("Pixel Format: 0x"); Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDIMGFMT);
  Serial.print("Image Format: 0x"); Serial.println(x, HEX);
  x = tft.readcommand8(ILI9341_RDSELFDIAG);
  Serial.print("Self Diagnostic: 0x"); Serial.println(x, HEX);

  Serial.println(F("Benchmark                Time (microseconds)"));
  delay(10);
  Serial.print(F("Screen fill              "));
  Serial.println(testFillScreen());
  delay(500);

  Serial.print(F("Text                     "));
  Serial.println(testText());
  delay(3000);

  Serial.print(F("Lines                    "));
  Serial.println(testLines(ILI9341_CYAN));
  delay(500);

  Serial.print(F("Horiz/Vert Lines         "));
  Serial.println(testFastLines(ILI9341_RED, ILI9341_BLUE));
  delay(500);

  Serial.print(F("Rectangles (outline)     "));
  Serial.println(testRects(ILI9341_GREEN));
  delay(500);

  Serial.print(F("Rectangles (filled)      "));
  Serial.println(testFilledRects(ILI9341_YELLOW, ILI9341_MAGENTA));
  delay(500);

  Serial.print(F("Circles (filled)         "));
  Serial.println(testFilledCircles(10, ILI9341_MAGENTA));

  Serial.print(F("Circles (outline)        "));
  Serial.println(testCircles(10, ILI9341_WHITE));
  delay(500);

  Serial.print(F("Triangles (outline)      "));
  Serial.println(testTriangles());
  delay(500);

  Serial.print(F("Triangles (filled)       "));
  Serial.println(testFilledTriangles());
  delay(500);

  Serial.print(F("Rounded rects (outline)  "));
  Serial.println(testRoundRects());
  delay(500);

  Serial.print(F("Rounded rects (filled)   "));
  Serial.println(testFilledRoundRects());
  delay(500);

  Serial.println(F("Done!"));

}


void loop(void) {
  for(uint8_t rotation=0; rotation<4; rotation++) {
    tft.setRotation(rotation);
    testText();
    delay(1000);
  }
}

unsigned long testFillScreen() {
  unsigned long start = micros();
  tft.fillScreen(ILI9341_BLACK);
  yield();
  tft.fillScreen(ILI9341_RED);
  yield();
```

```
  tft.fillScreen(ILI9341_GREEN);
  yield();
  tft.fillScreen(ILI9341_BLUE);
  yield();
  tft.fillScreen(ILI9341_BLACK);
  yield();
  return micros() - start;
}

unsigned long testText() {
  tft.fillScreen(ILI9341_BLACK);
  unsigned long start = micros();
  tft.setCursor(0, 0);
  tft.setTextColor(ILI9341_WHITE);  tft.setTextSize(1);
  tft.println("Hello World!");
  tft.setTextColor(ILI9341_YELLOW); tft.setTextSize(2);
  tft.println(1234.56);
  tft.setTextColor(ILI9341_RED);    tft.setTextSize(3);
  tft.println(0xDEADBEEF, HEX);
  tft.println();
  tft.setTextColor(ILI9341_GREEN);
  tft.setTextSize(5);
  tft.println("Groop");
  tft.setTextSize(2);
  tft.println("I implore thee,");
  tft.setTextSize(1);
  tft.println("my foonting turlingdromes.");
  tft.println("And hooptiously drangle me");
  tft.println("with crinkly bindlewurdles,");
  tft.println("Or I will rend thee");
  tft.println("in the gobberwarts");
  tft.println("with my blurglecruncheon,");
  tft.println("see if I don't!");
  return micros() - start;
}

unsigned long testLines(uint16_t color) {
  unsigned long start, t;
  int           x1, y1, x2, y2,
                w = tft.width(),
                h = tft.height();

  tft.fillScreen(ILI9341_BLACK);
  yield();

  x1 = y1 = 0;
  y2    = h - 1;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = w - 1;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);
  t     = micros() - start; // fillScreen doesn't count against timing

  yield();
  tft.fillScreen(ILI9341_BLACK);
  yield();

  x1    = w - 1;
  y1    = 0;
  y2    = h - 1;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = 0;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);
  t    += micros() - start;

  yield();
  tft.fillScreen(ILI9341_BLACK);
  yield();

  x1    = 0;
  y1    = h - 1;
  y2    = 0;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = w - 1;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);
  t    += micros() - start;

  yield();
  tft.fillScreen(ILI9341_BLACK);
  yield();

  x1    = w - 1;
  y1    = h - 1;
  y2    = 0;
  start = micros();
  for(x2=0; x2<w; x2+=6) tft.drawLine(x1, y1, x2, y2, color);
  x2    = 0;
  for(y2=0; y2<h; y2+=6) tft.drawLine(x1, y1, x2, y2, color);

  yield();
  return micros() - start;
}

unsigned long testFastLines(uint16_t color1, uint16_t color2) {
  unsigned long start;
  int           x, y, w = tft.width(), h = tft.height();

  tft.fillScreen(ILI9341_BLACK);
  start = micros();
  for(y=0; y<h; y+=5) tft.drawFastHLine(0, y, w, color1);
  for(x=0; x<w; x+=5) tft.drawFastVLine(x, 0, h, color2);

  return micros() - start;
}

unsigned long testRects(uint16_t color) {
  unsigned long start;
  int           n, i, i2,
                cx = tft.width()  / 2,
                cy = tft.height() / 2;

  tft.fillScreen(ILI9341_BLACK);
  n     = min(tft.width(), tft.height());
  start = micros();
  for(i=2; i<n; i+=6) {
```

```
    i2 = i / 2;
    tft.drawRect(cx-i2, cy-i2, i, i, color);
  }

  return micros() - start;
}

unsigned long testFilledRects(uint16_t color1, uint16_t color2) {
  unsigned long start, t = 0;
  int           n, i, i2,
                cx = tft.width()  / 2 - 1,
                cy = tft.height() / 2 - 1;

  tft.fillScreen(ILI9341_BLACK);
  n = min(tft.width(), tft.height());
  for(i=n; i>0; i-=6) {
    i2    = i / 2;
    start = micros();
    tft.fillRect(cx-i2, cy-i2, i, i, color1);
    t    += micros() - start;
    // Outlines are not included in timing results
    tft.drawRect(cx-i2, cy-i2, i, i, color2);
    yield();
  }

  return t;
}

unsigned long testFilledCircles(uint8_t radius, uint16_t color) {
  unsigned long start;
  int x, y, w = tft.width(), h = tft.height(), r2 = radius * 2;

  tft.fillScreen(ILI9341_BLACK);
  start = micros();
  for(x=radius; x<w; x+=r2) {
    for(y=radius; y<h; y+=r2) {
      tft.fillCircle(x, y, radius, color);
    }
  }

  return micros() - start;
}

unsigned long testCircles(uint8_t radius, uint16_t color) {
  unsigned long start;
  int           x, y, r2 = radius * 2,
                w = tft.width()  + radius,
                h = tft.height() + radius;

  // Screen is not cleared for this one -- this is
  // intentional and does not affect the reported time.
  start = micros();
  for(x=0; x<w; x+=r2) {
    for(y=0; y<h; y+=r2) {
      tft.drawCircle(x, y, radius, color);
    }
  }

  return micros() - start;
}

unsigned long testTriangles() {
  unsigned long start;
  int           n, i, cx = tft.width()  / 2 - 1,
                      cy = tft.height() / 2 - 1;

  tft.fillScreen(ILI9341_BLACK);
  n     = min(cx, cy);
  start = micros();
  for(i=0; i<n; i+=5) {
    tft.drawTriangle(
      cx    , cy - i, // peak
      cx - i, cy + i, // bottom left
      cx + i, cy + i, // bottom right
      tft.color565(i, i, i));
  }

  return micros() - start;
}

unsigned long testFilledTriangles() {
  unsigned long start, t = 0;
  int           i, cx = tft.width()  / 2 - 1,
                   cy = tft.height() / 2 - 1;

  tft.fillScreen(ILI9341_BLACK);
  start = micros();
  for(i=min(cx,cy); i>10; i-=5) {
    start = micros();
    tft.fillTriangle(cx, cy - i, cx - i, cy + i, cx + i, cy + i,
      tft.color565(0, i*10, i*10));
    t += micros() - start;
    tft.drawTriangle(cx, cy - i, cx - i, cy + i, cx + i, cy + i,
      tft.color565(i*10, i*10, 0));
    yield();
  }

  return t;
}

unsigned long testRoundRects() {
  unsigned long start;
  int           w, i, i2,
                cx = tft.width()  / 2 - 1,
                cy = tft.height() / 2 - 1;

  tft.fillScreen(ILI9341_BLACK);
  w     = min(tft.width(), tft.height());
  start = micros();
  for(i=0; i<w; i+=6) {
    i2 = i / 2;
    tft.drawRoundRect(cx-i2, cy-i2, i, i, i/8, tft.color565(i, 0, 0));
  }

  return micros() - start;
}
```

```
unsigned long testFilledRoundRects() {
  unsigned long start;
  int           i, i2,
                cx = tft.width()  / 2 - 1,
                cy = tft.height() / 2 - 1;

  tft.fillScreen(ILI9341_BLACK);
  start = micros();
  for(i=min(tft.width(), tft.height()); i>20; i-=6) {
    i2 = i / 2;
    tft.fillRoundRect(cx-i2, cy-i2, i, i, i/8, tft.color565(0, i, 0));
    yield();
  }

  return micros() - start;
}
```

***** Adafruit_ILI9341.h

```
#ifndef _ADAFRUIT_ILI9341H_
#define _ADAFRUIT_ILI9341H_

#include "Adafruit_GFX.h"
#include "Arduino.h"
#include "Print.h"
#include <Adafruit_SPITFT.h>
#include <SPI.h>

#define ILI9341_TFTWIDTH 240  ///< ILI9341 max TFT width
#define ILI9341_TFTHEIGHT 320 ///< ILI9341 max TFT height

#define ILI9341_NOP 0x00     ///< No-op register
#define ILI9341_SWRESET 0x01 ///< Software reset register
#define ILI9341_RDDID 0x04   ///< Read display identification information
#define ILI9341_RDDST 0x09   ///< Read Display Status

#define ILI9341_SLPIN 0x10  ///< Enter Sleep Mode
#define ILI9341_SLPOUT 0x11 ///< Sleep Out
#define ILI9341_PTLON 0x12  ///< Partial Mode ON
#define ILI9341_NORON 0x13  ///< Normal Display Mode ON

#define ILI9341_RDMODE 0x0A     ///< Read Display Power Mode
#define ILI9341_RDMADCTL 0x0B   ///< Read Display MADCTL
#define ILI9341_RDPIXFMT 0x0C   ///< Read Display Pixel Format
#define ILI9341_RDIMGFMT 0x0D   ///< Read Display Image Format
#define ILI9341_RDSELFDIAG 0x0F ///< Read Display Self-Diagnostic Result

#define ILI9341_INVOFF 0x20   ///< Display Inversion OFF
#define ILI9341_INVON 0x21    ///< Display Inversion ON
#define ILI9341_GAMMASET 0x26 ///< Gamma Set
#define ILI9341_DISPOFF 0x28  ///< Display OFF
#define ILI9341_DISPON 0x29   ///< Display ON

#define ILI9341_CASET 0x2A ///< Column Address Set
#define ILI9341_PASET 0x2B ///< Page Address Set
#define ILI9341_RAMWR 0x2C ///< Memory Write
#define ILI9341_RAMRD 0x2E ///< Memory Read

#define ILI9341_PTLAR 0x30    ///< Partial Area
#define ILI9341_VSCRDEF 0x33  ///< Vertical Scrolling Definition
#define ILI9341_MADCTL 0x36   ///< Memory Access Control
#define ILI9341_VSCRSADD 0x37 ///< Vertical Scrolling Start Address
#define ILI9341_PIXFMT 0x3A   ///< COLMOD: Pixel Format Set

#define ILI9341_FRMCTR1                                                        \
  0xB1 ///< Frame Rate Control (In Normal Mode/Full Colors)
#define ILI9341_FRMCTR2 0xB2 ///< Frame Rate Control (In Idle Mode/8 colors)
#define ILI9341_FRMCTR3                                                        \
  0xB3 ///< Frame Rate control (In Partial Mode/Full Colors)
#define ILI9341_INVCTR 0xB4  ///< Display Inversion Control
#define ILI9341_DFUNCTR 0xB6 ///< Display Function Control

#define ILI9341_PWCTR1 0xC0 ///< Power Control 1
#define ILI9341_PWCTR2 0xC1 ///< Power Control 2
#define ILI9341_PWCTR3 0xC2 ///< Power Control 3
#define ILI9341_PWCTR4 0xC3 ///< Power Control 4
#define ILI9341_PWCTR5 0xC4 ///< Power Control 5
#define ILI9341_VMCTR1 0xC5 ///< VCOM Control 1
#define ILI9341_VMCTR2 0xC7 ///< VCOM Control 2

#define ILI9341_RDID1 0xDA ///< Read ID 1
#define ILI9341_RDID2 0xDB ///< Read ID 2
#define ILI9341_RDID3 0xDC ///< Read ID 3
#define ILI9341_RDID4 0xDD ///< Read ID 4

#define ILI9341_GMCTRP1 0xE0 ///< Positive Gamma Correction
#define ILI9341_GMCTRN1 0xE1 ///< Negative Gamma Correction
//#define ILI9341_PWCTR6     0xFC

// Color definitions
#define ILI9341_BLACK 0x0000       ///<   0,   0,   0
#define ILI9341_NAVY 0x000F        ///<   0,   0, 123
#define ILI9341_DARKGREEN 0x03E0   ///<   0, 125,   0
#define ILI9341_DARKCYAN 0x03EF    ///<   0, 125, 123
#define ILI9341_MAROON 0x7800      ///< 123,   0,   0
#define ILI9341_PURPLE 0x780F      ///< 123,   0, 123
#define ILI9341_OLIVE 0x7BE0       ///< 123, 125,   0
#define ILI9341_LIGHTGREY 0xC618   ///< 198, 195, 198
#define ILI9341_DARKGREY 0x7BEF    ///< 123, 125, 123
#define ILI9341_BLUE 0x001F        ///<   0,   0, 255
#define ILI9341_GREEN 0x07E0       ///<   0, 255,   0
#define ILI9341_CYAN 0x07FF        ///<   0, 255, 255
#define ILI9341_RED 0xF800         ///< 255,   0,   0
#define ILI9341_MAGENTA 0xF81F     ///< 255,   0, 255
#define ILI9341_YELLOW 0xFFE0      ///< 255, 255,   0
#define ILI9341_WHITE 0xFFFF       ///< 255, 255, 255
#define ILI9341_ORANGE 0xFD20      ///< 255, 165,   0
#define ILI9341_GREENYELLOW 0xAFE5 ///< 173, 255,  41
#define ILI9341_PINK 0xFC18        ///< 255, 130, 198

/*************************************************************************/
/*!
@brief Class to manage hardware interface with ILI9341 chipset (also seems to
work with ILI9340)
*/
/*************************************************************************/
```

```
class Adafruit_ILI9341 : public Adafruit_SPITFT {
public:
  Adafruit_ILI9341(int8_t _CS, int8_t _DC, int8_t _MOSI, int8_t _SCLK,
                   int8_t _RST = -1, int8_t _MISO = -1);
  Adafruit_ILI9341(int8_t _CS, int8_t _DC, int8_t _RST = -1);
#if !defined(ESP8266)
  Adafruit_ILI9341(SPIClass *spiClass, int8_t dc, int8_t cs = -1,
                   int8_t rst = -1);
#endif // end !ESP8266
  Adafruit_ILI9341(tftBusWidth busWidth, int8_t d0, int8_t wr, int8_t dc,
                   int8_t cs = -1, int8_t rst = -1, int8_t rd = -1);

  void begin(uint32_t freq = 0);
  void setRotation(uint8_t r);
  void invertDisplay(bool i);
  void scrollTo(uint16_t y);
  void setScrollMargins(uint16_t top, uint16_t bottom);

  // Transaction API not used by GFX
  void setAddrWindow(uint16_t x, uint16_t y, uint16_t w, uint16_t h);

  uint8_t readcommand8(uint8_t reg, uint8_t index = 0);
};

#endif // _ADAFRUIT_ILI9341H_
```

***** Adafruit_ILI9341.cpp

```
#include "Adafruit_ILI9341.h"
#ifndef ARDUINO_STM32_FEATHER
#include "pins_arduino.h"
#ifndef RASPI
#include "wiring_private.h"
#endif
#endif
#include <limits.h>

#if defined(ARDUINO_ARCH_ARC32) || defined(ARDUINO_MAXIM)
#define SPI_DEFAULT_FREQ 16000000
// Teensy 3.0, 3.1/3.2, 3.5, 3.6
#elif defined(__MK20DX128__) || defined(__MK20DX256__) ||                  \
    defined(__MK64FX512__) || defined(__MK66FX1M0__)
#define SPI_DEFAULT_FREQ 40000000
#elif defined(__AVR__) || defined(TEENSYDUINO)
#define SPI_DEFAULT_FREQ 8000000
#elif defined(ESP8266) || defined(ESP32)
#define SPI_DEFAULT_FREQ 40000000
#elif defined(RASPI)
#define SPI_DEFAULT_FREQ 80000000
#elif defined(ARDUINO_ARCH_STM32F1)
#define SPI_DEFAULT_FREQ 36000000
#else
#define SPI_DEFAULT_FREQ 24000000 ///< Default SPI data clock frequency
#endif

#define MADCTL_MY 0x80  ///< Bottom to top
#define MADCTL_MX 0x40  ///< Right to left
#define MADCTL_MV 0x20  ///< Reverse Mode
#define MADCTL_ML 0x10  ///< LCD refresh Bottom to top
#define MADCTL_RGB 0x00 ///< Red-Green-Blue pixel order
#define MADCTL_BGR 0x08 ///< Blue-Green-Red pixel order
#define MADCTL_MH 0x04  ///< LCD refresh right to left

/***************************************************************************/
/*!
    @brief  Instantiate Adafruit ILI9341 driver with software SPI
    @param    cs    Chip select pin #
    @param    dc    Data/Command pin #
    @param    mosi  SPI MOSI pin #
    @param    sclk  SPI Clock pin #
    @param    rst   Reset pin # (optional, pass -1 if unused)
    @param    miso  SPI MISO pin # (optional, pass -1 if unused)
*/
/***************************************************************************/
Adafruit_ILI9341::Adafruit_ILI9341(int8_t cs, int8_t dc, int8_t mosi,
                                   int8_t sclk, int8_t rst, int8_t miso)
    : Adafruit_SPITFT(ILI9341_TFTWIDTH, ILI9341_TFTHEIGHT, cs, dc, mosi, sclk,
                      rst, miso) {}

/***************************************************************************/
/*!
    @brief  Instantiate Adafruit ILI9341 driver with hardware SPI using the
            default SPI peripheral.
    @param  cs   Chip select pin # (OK to pass -1 if CS tied to GND).
    @param  dc   Data/Command pin # (required).
    @param  rst  Reset pin # (optional, pass -1 if unused).
*/
/***************************************************************************/
Adafruit_ILI9341::Adafruit_ILI9341(int8_t cs, int8_t dc, int8_t rst)
    : Adafruit_SPITFT(ILI9341_TFTWIDTH, ILI9341_TFTHEIGHT, cs, dc, rst) {}

#if !defined(ESP8266)
/***************************************************************************/
/*!
    @brief  Instantiate Adafruit ILI9341 driver with hardware SPI using
            a specific SPI peripheral (not necessarily default).
    @param  spiClass  Pointer to SPI peripheral (e.g. &SPI or &SPI1).
    @param  dc        Data/Command pin # (required).
    @param  cs        Chip select pin # (optional, pass -1 if unused and
                      CS is tied to GND).
    @param  rst       Reset pin # (optional, pass -1 if unused).
*/
/***************************************************************************/
Adafruit_ILI9341::Adafruit_ILI9341(SPIClass *spiClass, int8_t dc, int8_t cs,
                                   int8_t rst)
    : Adafruit_SPITFT(ILI9341_TFTWIDTH, ILI9341_TFTHEIGHT, spiClass, cs, dc,
                      rst) {}
#endif // end !ESP8266

/***************************************************************************/
/*!
    @brief  Instantiate Adafruit ILI9341 driver using parallel interface.
    @param  busWidth  If tft16 (enumeration in Adafruit_SPITFT.h), is a
                      16-bit interface, else 8-bit.
    @param  d0        Data pin 0 (MUST be a byte- or word-aligned LSB of a
                      PORT register -- pins 1-n are extrapolated from this).
```

```
    @param  wr        Write strobe pin # (required).
    @param  dc        Data/Command pin # (required).
    @param  cs        Chip select pin # (optional, pass -1 if unused and CS
                      is tied to GND).
    @param  rst       Reset pin # (optional, pass -1 if unused).
    @param  rd        Read strobe pin # (optional, pass -1 if unused).
*/
/**************************************************************************/
Adafruit_ILI9341::Adafruit_ILI9341(tftBusWidth busWidth, int8_t d0, int8_t wr,
                                   int8_t dc, int8_t cs, int8_t rst, int8_t rd)
    : Adafruit_SPITFT(ILI9341_TFTWIDTH, ILI9341_TFTHEIGHT, busWidth, d0, wr, dc,
                      cs, rst, rd) {}

// clang-format off
static const uint8_t PROGMEM initcmd[] = {
  0xEF, 3, 0x03, 0x80, 0x02,
  0xCF, 3, 0x00, 0xC1, 0x30,
  0xED, 4, 0x64, 0x03, 0x12, 0x81,
  0xE8, 3, 0x85, 0x00, 0x78,
  0xCB, 5, 0x39, 0x2C, 0x00, 0x34, 0x02,
  0xF7, 1, 0x20,
  0xEA, 2, 0x00, 0x00,
  ILI9341_PWCTR1  , 1, 0x23,             // Power control VRH[5:0]
  ILI9341_PWCTR2  , 1, 0x10,             // Power control SAP[2:0];BT[3:0]
  ILI9341_VMCTR1  , 2, 0x3e, 0x28,       // VCM control
  ILI9341_VMCTR2  , 1, 0x86,             // VCM control2
  ILI9341_MADCTL  , 1, 0x48,             // Memory Access Control
  ILI9341_VSCRSADD, 1, 0x00,             // Vertical scroll zero
  ILI9341_PIXFMT  , 1, 0x55,
  ILI9341_FRMCTR1 , 2, 0x00, 0x18,
  ILI9341_DFUNCTR , 3, 0x08, 0x82, 0x27, // Display Function Control
  0xF2, 1, 0x00,                         // 3Gamma Function Disable
  ILI9341_GAMMASET , 1, 0x01,             // Gamma curve selected
  ILI9341_GMCTRP1 , 15, 0x0F, 0x31, 0x2B, 0x0C, 0x0E, 0x08, // Set Gamma
    0x4E, 0xF1, 0x37, 0x07, 0x10, 0x03, 0x0E, 0x09, 0x00,
  ILI9341_GMCTRN1 , 15, 0x00, 0x0E, 0x14, 0x03, 0x11, 0x07, // Set Gamma
    0x31, 0xC1, 0x48, 0x08, 0x0F, 0x0C, 0x31, 0x36, 0x0F,
  ILI9341_SLPOUT  , 0x80,                // Exit Sleep
  ILI9341_DISPON  , 0x80,                // Display on
  0x00                                   // End of list
};
// clang-format on

/**************************************************************************/
/*!
    @brief   Initialize ILI9341 chip
    Connects to the ILI9341 over SPI and sends initialization procedure commands
    @param    freq  Desired SPI clock frequency
*/
/**************************************************************************/
void Adafruit_ILI9341::begin(uint32_t freq) {

  if (!freq)
    freq = SPI_DEFAULT_FREQ;
  initSPI(freq);

  if (_rst < 0) {                // If no hardware reset pin...
    sendCommand(ILI9341_SWRESET); // Engage software reset
    delay(150);
  }

  uint8_t cmd, x, numArgs;
  const uint8_t *addr = initcmd;
  while ((cmd = pgm_read_byte(addr++)) > 0) {
    x = pgm_read_byte(addr++);
    numArgs = x & 0x7F;
    sendCommand(cmd, addr, numArgs);
    addr += numArgs;
    if (x & 0x80)
      delay(150);
  }

  _width = ILI9341_TFTWIDTH;
  _height = ILI9341_TFTHEIGHT;
}

/**************************************************************************/
/*!
    @brief   Set origin of (0,0) and orientation of TFT display
    @param   m  The index for rotation, from 0-3 inclusive
*/
/**************************************************************************/
void Adafruit_ILI9341::setRotation(uint8_t m) {
  rotation = m % 4; // can't be higher than 3
  switch (rotation) {
  case 0:
    m = (MADCTL_MX | MADCTL_BGR);
    _width = ILI9341_TFTWIDTH;
    _height = ILI9341_TFTHEIGHT;
    break;
  case 1:
    m = (MADCTL_MV | MADCTL_BGR);
    _width = ILI9341_TFTHEIGHT;
    _height = ILI9341_TFTWIDTH;
    break;
  case 2:
    m = (MADCTL_MY | MADCTL_BGR);
    _width = ILI9341_TFTWIDTH;
    _height = ILI9341_TFTHEIGHT;
    break;
  case 3:
    m = (MADCTL_MX | MADCTL_MY | MADCTL_MV | MADCTL_BGR);
    _width = ILI9341_TFTHEIGHT;
    _height = ILI9341_TFTWIDTH;
    break;
  }

  sendCommand(ILI9341_MADCTL, &m, 1);
}

/**************************************************************************/
/*!
    @brief   Enable/Disable display color inversion
    @param   invert True to invert, False to have normal color
*/
/**************************************************************************/
void Adafruit_ILI9341::invertDisplay(bool invert) {
```

```
  sendCommand(invert ? ILI9341_INVON : ILI9341_INVOFF);
}

/********************************************************************/
/*!
    @brief   Scroll display memory
    @param   y How many pixels to scroll display by
*/
/********************************************************************/
void Adafruit_ILI9341::scrollTo(uint16_t y) {
  uint8_t data[2];
  data[0] = y >> 8;
  data[1] = y & 0xff;
  sendCommand(ILI9341_VSCRSADD, (uint8_t *)data, 2);
}

/********************************************************************/
/*!
    @brief   Set the height of the Top and Bottom Scroll Margins
    @param   top The height of the Top scroll margin
    @param   bottom The height of the Bottom scroll margin
 */
/********************************************************************/
void Adafruit_ILI9341::setScrollMargins(uint16_t top, uint16_t bottom) {
  // TFA+VSA+BFA must equal 320
  if (top + bottom <= ILI9341_TFTHEIGHT) {
    uint16_t middle = ILI9341_TFTHEIGHT - (top + bottom);
    uint8_t data[6];
    data[0] = top >> 8;
    data[1] = top & 0xff;
    data[2] = middle >> 8;
    data[3] = middle & 0xff;
    data[4] = bottom >> 8;
    data[5] = bottom & 0xff;
    sendCommand(ILI9341_VSCRDEF, (uint8_t *)data, 6);
  }
}

/********************************************************************/
/*!
    @brief   Set the "address window" - the rectangle we will write to RAM with
   the next chunk of      SPI data writes. The ILI9341 will automatically wrap
   the data as each row is filled
    @param   x1  TFT memory 'x' origin
    @param   y1  TFT memory 'y' origin
    @param   w   Width of rectangle
    @param   h   Height of rectangle
*/
/********************************************************************/
void Adafruit_ILI9341::setAddrWindow(uint16_t x1, uint16_t y1, uint16_t w,
                                     uint16_t h) {
  static uint16_t old_x1 = 0xffff, old_x2 = 0xffff;
  static uint16_t old_y1 = 0xffff, old_y2 = 0xffff;

  uint16_t x2 = (x1 + w - 1), y2 = (y1 + h - 1);
  if (x1 != old_x1 || x2 != old_x2) {
    writeCommand(ILI9341_CASET); // Column address set
    SPI_WRITE16(x1);
    SPI_WRITE16(x2);
    old_x1 = x1;
    old_x2 = x2;
  }
  if (y1 != old_y1 || y2 != old_y2) {
    writeCommand(ILI9341_PASET); // Row address set
    SPI_WRITE16(y1);
    SPI_WRITE16(y2);
    old_y1 = y1;
    old_y2 = y2;
  }
  writeCommand(ILI9341_RAMWR); // Write to RAM
}

/********************************************************************/
/*!
    @brief  Read 8 bits of data from ILI9341 configuration memory. NOT from RAM!
            This is highly undocumented/supported, it's really a hack but kinda
   works?
    @param    commandByte  The command register to read data from
    @param    index  The byte index into the command to read from
    @return   Unsigned 8-bit data read from ILI9341 register
 */
/********************************************************************/
uint8_t Adafruit_ILI9341::readcommand8(uint8_t commandByte, uint8_t index) {
  uint8_t data = 0x10 + index;
  sendCommand(0xD9, &data, 1); // Set Index Register
  return Adafruit_SPITFT::readcommand8(commandByte);
}
```

***** Adafruit_GFX.h

```
#ifndef _ADAFRUIT_GFX_H
#define _ADAFRUIT_GFX_H

#if ARDUINO >= 100
#include "Arduino.h"
#include "Print.h"
#else
#include "WProgram.h"
#endif
#include "gfxfont.h"

#include <Adafruit_I2CDevice.h>
#include <Adafruit_SPIDevice.h>

/// A generic graphics superclass that can handle all sorts of drawing. At a
/// minimum you can subclass and provide drawPixel(). At a maximum you can do a
/// ton of overriding to optimize. Used for any/all Adafruit displays!
class Adafruit_GFX : public Print {

public:
  Adafruit_GFX(int16_t w, int16_t h); // Constructor

  /********************************************************************/
  /*!
    @brief  Draw to the screen/framebuffer/etc.
    Must be overridden in subclass.
```

```
  @param  x    X coordinate in pixels
  @param  y    Y coordinate in pixels
  @param color  16-bit pixel color.
*/
/**************************************************************************/
virtual void drawPixel(int16_t x, int16_t y, uint16_t color) = 0;

// TRANSACTION API / CORE DRAW API
// These MAY be overridden by the subclass to provide device-specific
// optimized code.  Otherwise 'generic' versions are used.
virtual void startWrite(void);
virtual void writePixel(int16_t x, int16_t y, uint16_t color);
virtual void writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                           uint16_t color);
virtual void writeFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
virtual void writeFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
virtual void writeLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                       uint16_t color);
virtual void endWrite(void);

// CONTROL API
// These MAY be overridden by the subclass to provide device-specific
// optimized code.  Otherwise 'generic' versions are used.
virtual void setRotation(uint8_t r);
virtual void invertDisplay(bool i);

// BASIC DRAW API
// These MAY be overridden by the subclass to provide device-specific
// optimized code.  Otherwise 'generic' versions are used.

// It's good to implement those, even if using transaction API
virtual void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
virtual void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
virtual void fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                      uint16_t color);
virtual void fillScreen(uint16_t color);
// Optional and probably not necessary to change
virtual void drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                      uint16_t color);
virtual void drawRect(int16_t x, int16_t y, int16_t w, int16_t h,
                      uint16_t color);

// These exist only with Adafruit_GFX (no subclass overrides)
void drawCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color);
void drawCircleHelper(int16_t x0, int16_t y0, int16_t r, uint8_t cornername,
                      uint16_t color);
void fillCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color);
void fillCircleHelper(int16_t x0, int16_t y0, int16_t r, uint8_t cornername,
                      int16_t delta, uint16_t color);
void drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2,
                  int16_t y2, uint16_t color);
void fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2,
                  int16_t y2, uint16_t color);
void drawRoundRect(int16_t x0, int16_t y0, int16_t w, int16_t h,
                   int16_t radius, uint16_t color);
void fillRoundRect(int16_t x0, int16_t y0, int16_t w, int16_t h,
                   int16_t radius, uint16_t color);
void drawBitmap(int16_t x, int16_t y, const uint8_t bitmap[], int16_t w,
                int16_t h, uint16_t color);
void drawBitmap(int16_t x, int16_t y, const uint8_t bitmap[], int16_t w,
                int16_t h, uint16_t color, uint16_t bg);
void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w, int16_t h,
                uint16_t color);
void drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w, int16_t h,
                uint16_t color, uint16_t bg);
void drawXBitmap(int16_t x, int16_t y, const uint8_t bitmap[], int16_t w,
                 int16_t h, uint16_t color);
void drawGrayscaleBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
                         int16_t w, int16_t h);
void drawGrayscaleBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w,
                         int16_t h);
void drawGrayscaleBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
                         const uint8_t mask[], int16_t w, int16_t h);
void drawGrayscaleBitmap(int16_t x, int16_t y, uint8_t *bitmap, uint8_t *mask,
                         int16_t w, int16_t h);
void drawRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[], int16_t w,
                   int16_t h);
void drawRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap, int16_t w,
                   int16_t h);
void drawRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[],
                   const uint8_t mask[], int16_t w, int16_t h);
void drawRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap, uint8_t *mask,
                   int16_t w, int16_t h);
void drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color,
              uint16_t bg, uint8_t size);
void drawChar(int16_t x, int16_t y, unsigned char c, uint16_t color,
              uint16_t bg, uint8_t size_x, uint8_t size_y);
void getTextBounds(const char *string, int16_t x, int16_t y, int16_t *x1,
                   int16_t *y1, uint16_t *w, uint16_t *h);
void getTextBounds(const __FlashStringHelper *s, int16_t x, int16_t y,
                   int16_t *x1, int16_t *y1, uint16_t *w, uint16_t *h);
void getTextBounds(const String &str, int16_t x, int16_t y, int16_t *x1,
                   int16_t *y1, uint16_t *w, uint16_t *h);
void setTextSize(uint8_t s);
void setTextSize(uint8_t sx, uint8_t sy);
void setFont(const GFXfont *f = NULL);

/**************************************************************************/
/*!
  @brief  Set text cursor location
  @param  x    X coordinate in pixels
  @param  y    Y coordinate in pixels
*/
/**************************************************************************/
void setCursor(int16_t x, int16_t y) {
  cursor_x = x;
  cursor_y = y;
}

/**************************************************************************/
/*!
  @brief   Set text font color with transparant background
  @param   c   16-bit 5-6-5 Color to draw text with
  @note    For 'transparent' background, background and foreground
           are set to same color rather than using a separate flag.
*/
/**************************************************************************/
```

```
  void setTextColor(uint16_t c) { textcolor = textbgcolor = c; }

  /**************************************************************************/
  /*!
    @brief   Set text font color with custom background color
    @param   c   16-bit 5-6-5 Color to draw text with
    @param   bg  16-bit 5-6-5 Color to draw background/fill with
  */
  /**************************************************************************/
  void setTextColor(uint16_t c, uint16_t bg) {
    textcolor = c;
    textbgcolor = bg;
  }

  /**************************************************************************/
  /*!
  @brief  Set whether text that is too long for the screen width should
          automatically wrap around to the next line (else clip right).
  @param  w  true for wrapping, false for clipping
  */
  /**************************************************************************/
  void setTextWrap(bool w) { wrap = w; }

  /**************************************************************************/
  /*!
    @brief  Enable (or disable) Code Page 437-compatible charset.
            There was an error in glcdfont.c for the longest time -- one
            character (#176, the 'light shade' block) was missing -- this
            threw off the index of every character that followed it.
            But a TON of code has been written with the erroneous
            character indices. By default, the library uses the original
            'wrong' behavior and old sketches will still work. Pass
            'true' to this function to use correct CP437 character values
            in your code.
    @param  x  true = enable (new behavior), false = disable (old behavior)
  */
  /**************************************************************************/
  void cp437(bool x = true) { _cp437 = x; }

  using Print::write;
#if ARDUINO >= 100
  virtual size_t write(uint8_t);
#else
  virtual void write(uint8_t);
#endif

  /**************************************************************************/
  /*!
    @brief      Get width of the display, accounting for current rotation
    @returns    Width in pixels
  */
  /**************************************************************************/
  int16_t width(void) const { return _width; };

  /**************************************************************************/
  /*!
    @brief      Get height of the display, accounting for current rotation
    @returns    Height in pixels
  */
  /**************************************************************************/
  int16_t height(void) const { return _height; }

  /**************************************************************************/
  /*!
    @brief      Get rotation setting for display
    @returns    0 thru 3 corresponding to 4 cardinal rotations
  */
  /**************************************************************************/
  uint8_t getRotation(void) const { return rotation; }

  // get current cursor position (get rotation safe maximum values,
  // using: width() for x, height() for y)
  /**************************************************************************/
  /*!
    @brief  Get text cursor X location
    @returns    X coordinate in pixels
  */
  /**************************************************************************/
  int16_t getCursorX(void) const { return cursor_x; }

  /**************************************************************************/
  /*!
    @brief      Get text cursor Y location
    @returns    Y coordinate in pixels
  */
  /**************************************************************************/
  int16_t getCursorY(void) const { return cursor_y; };

protected:
  void charBounds(unsigned char c, int16_t *x, int16_t *y, int16_t *minx,
                  int16_t *miny, int16_t *maxx, int16_t *maxy);
  int16_t WIDTH;        ///< This is the 'raw' display width - never changes
  int16_t HEIGHT;       ///< This is the 'raw' display height - never changes
  int16_t _width;       ///< Display width as modified by current rotation
  int16_t _height;      ///< Display height as modified by current rotation
  int16_t cursor_x;     ///< x location to start print()ing text
  int16_t cursor_y;     ///< y location to start print()ing text
  uint16_t textcolor;   ///< 16-bit background color for print()
  uint16_t textbgcolor; ///< 16-bit text color for print()
  uint8_t textsize_x;   ///< Desired magnification in X-axis of text to print()
  uint8_t textsize_y;   ///< Desired magnification in Y-axis of text to print()
  uint8_t rotation;     ///< Display rotation (0 thru 3)
  bool wrap;            ///< If set, 'wrap' text at right edge of display
  bool _cp437;          ///< If set, use correct CP437 charset (default is off)
  GFXfont *gfxFont;     ///< Pointer to special font
};

/// A simple drawn button UI element
class Adafruit_GFX_Button {

public:
  Adafruit_GFX_Button(void);
  // "Classic" initButton() uses center & size
  void initButton(Adafruit_GFX *gfx, int16_t x, int16_t y, uint16_t w,
                  uint16_t h, uint16_t outline, uint16_t fill,
                  uint16_t textcolor, char *label, uint8_t textsize);
  void initButton(Adafruit_GFX *gfx, int16_t x, int16_t y, uint16_t w,
```

```
                uint16_t h, uint16_t outline, uint16_t fill,
                uint16_t textcolor, char *label, uint8_t textsize_x,
                uint8_t textsize_y);
  // New/alt initButton() uses upper-left corner & size
  void initButtonUL(Adafruit_GFX *gfx, int16_t x1, int16_t y1, uint16_t w,
                    uint16_t h, uint16_t outline, uint16_t fill,
                    uint16_t textcolor, char *label, uint8_t textsize);
  void initButtonUL(Adafruit_GFX *gfx, int16_t x1, int16_t y1, uint16_t w,
                    uint16_t h, uint16_t outline, uint16_t fill,
                    uint16_t textcolor, char *label, uint8_t textsize_x,
                    uint8_t textsize_y);
  void drawButton(bool inverted = false);
  bool contains(int16_t x, int16_t y);

  /*************************************************************************/
  /*!
    @brief    Sets button state, should be done by some touch function
    @param    p  True for pressed, false for not.
  */
  /*************************************************************************/
  void press(bool p) {
    laststate = currstate;
    currstate = p;
  }

  bool justPressed();
  bool justReleased();

  /*************************************************************************/
  /*!
    @brief    Query whether the button is currently pressed
    @returns  True if pressed
  */
  /*************************************************************************/
  bool isPressed(void) { return currstate; };

private:
  Adafruit_GFX *_gfx;
  int16_t _x1, _y1; // Coordinates of top-left corner
  uint16_t _w, _h;
  uint8_t _textsize_x;
  uint8_t _textsize_y;
  uint16_t _outlinecolor, _fillcolor, _textcolor;
  char _label[10];

  bool currstate, laststate;
};

/// A GFX 1-bit canvas context for graphics
class GFXcanvas1 : public Adafruit_GFX {
public:
  GFXcanvas1(uint16_t w, uint16_t h);
  ~GFXcanvas1(void);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillScreen(uint16_t color);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  bool getPixel(int16_t x, int16_t y) const;
  /*************************************************************************/
  /*!
    @brief    Get a pointer to the internal buffer memory
    @returns  A pointer to the allocated buffer
  */
  /*************************************************************************/
  uint8_t *getBuffer(void) const { return buffer; }

protected:
  bool getRawPixel(int16_t x, int16_t y) const;
  void drawFastRawVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastRawHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint8_t *buffer; ///< Raster data: no longer private, allow subclass access

private:
#ifdef __AVR__
  // Bitmask tables of 0x80>>X and ~(0x80>>X), because X>>Y is slow on AVR
  static const uint8_t PROGMEM GFXsetBit[], GFXclrBit[];
#endif
};

/// A GFX 8-bit canvas context for graphics
class GFXcanvas8 : public Adafruit_GFX {
public:
  GFXcanvas8(uint16_t w, uint16_t h);
  ~GFXcanvas8(void);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillScreen(uint16_t color);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint8_t getPixel(int16_t x, int16_t y) const;
  /*************************************************************************/
  /*!
   @brief    Get a pointer to the internal buffer memory
   @returns  A pointer to the allocated buffer
  */
  /*************************************************************************/
  uint8_t *getBuffer(void) const { return buffer; }

protected:
  uint8_t getRawPixel(int16_t x, int16_t y) const;
  void drawFastRawVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastRawHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint8_t *buffer; ///< Raster data: no longer private, allow subclass access
};

///  A GFX 16-bit canvas context for graphics
class GFXcanvas16 : public Adafruit_GFX {
public:
  GFXcanvas16(uint16_t w, uint16_t h);
  ~GFXcanvas16(void);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillScreen(uint16_t color);
  void byteSwap(void);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint16_t getPixel(int16_t x, int16_t y) const;
  /*************************************************************************/
  /*!
```

```
    @brief    Get a pointer to the internal buffer memory
    @returns  A pointer to the allocated buffer
  */
  /**************************************************************************/
  uint16_t *getBuffer(void) const { return buffer; }

protected:
  uint16_t getRawPixel(int16_t x, int16_t y) const;
  void drawFastRawVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  void drawFastRawHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  uint16_t *buffer; ///< Raster data: no longer private, allow subclass access
};

#endif // _ADAFRUIT_GFX_H
```

***** Adafruit_GFX.cpp

```
#include "Adafruit_GFX.h"
#include "glcdfont.c"
#ifdef __AVR__
#include <avr/pgmspace.h>
#elif defined(ESP8266) || defined(ESP32)
#include <pgmspace.h>
#endif

// Many (but maybe not all) non-AVR board installs define macros
// for compatibility with existing PROGMEM-reading AVR code.
// Do our own checks and defines here for good measure...

#ifndef pgm_read_byte
#define pgm_read_byte(addr) (*(const unsigned char *)(addr))
#endif
#ifndef pgm_read_word
#define pgm_read_word(addr) (*(const unsigned short *)(addr))
#endif
#ifndef pgm_read_dword
#define pgm_read_dword(addr) (*(const unsigned long *)(addr))
#endif

// Pointers are a peculiar case...typically 16-bit on AVR boards,
// 32 bits elsewhere.  Try to accommodate both...

#if !defined(__INT_MAX__) || (__INT_MAX__ > 0xFFFF)
#define pgm_read_pointer(addr) ((void *)pgm_read_dword(addr))
#else
#define pgm_read_pointer(addr) ((void *)pgm_read_word(addr))
#endif

inline GFXglyph *pgm_read_glyph_ptr(const GFXfont *gfxFont, uint8_t c) {
#ifdef __AVR__
  return &(((GFXglyph *)pgm_read_pointer(&gfxFont->glyph))[c]);
#else
  // expression in __AVR__ section may generate "dereferencing type-punned
  // pointer will break strict-aliasing rules" warning In fact, on other
  // platforms (such as STM32) there is no need to do this pointer magic as
  // program memory may be read in a usual way So expression may be simplified
  return gfxFont->glyph + c;
#endif //__AVR__
}

inline uint8_t *pgm_read_bitmap_ptr(const GFXfont *gfxFont) {
#ifdef __AVR__
  return (uint8_t *)pgm_read_pointer(&gfxFont->bitmap);
#else
  // expression in __AVR__ section generates "dereferencing type-punned pointer
  // will break strict-aliasing rules" warning In fact, on other platforms (such
  // as STM32) there is no need to do this pointer magic as program memory may
  // be read in a usual way So expression may be simplified
  return gfxFont->bitmap;
#endif //__AVR__
}

#ifndef min
#define min(a, b) (((a) < (b)) ? (a) : (b))
#endif

#ifndef _swap_int16_t
#define _swap_int16_t(a, b)                                                    \
  {                                                                            \
    int16_t t = a;                                                             \
    a = b;                                                                     \
    b = t;                                                                     \
  }
#endif

/**************************************************************************/
/*!
   @brief    Instatiate a GFX context for graphics! Can only be done by a
   superclass
   @param    w   Display width, in pixels
   @param    h   Display height, in pixels
*/
/**************************************************************************/
Adafruit_GFX::Adafruit_GFX(int16_t w, int16_t h) : WIDTH(w), HEIGHT(h) {
  _width = WIDTH;
  _height = HEIGHT;
  rotation = 0;
  cursor_y = cursor_x = 0;
  textsize_x = textsize_y = 1;
  textcolor = textbgcolor = 0xFFFF;
  wrap = true;
  _cp437 = false;
  gfxFont = NULL;
}

/**************************************************************************/
/*!
   @brief    Write a line.  Bresenham's algorithm - thx wikpedia
    @param    x0  Start point x coordinate
    @param    y0  Start point y coordinate
    @param    x1  End point x coordinate
    @param    y1  End point y coordinate
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::writeLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
```

```
                              uint16_t color) {
#if defined(ESP8266)
  yield();
#endif
  int16_t steep = abs(y1 - y0) > abs(x1 - x0);
  if (steep) {
    _swap_int16_t(x0, y0);
    _swap_int16_t(x1, y1);
  }

  if (x0 > x1) {
    _swap_int16_t(x0, x1);
    _swap_int16_t(y0, y1);
  }

  int16_t dx, dy;
  dx = x1 - x0;
  dy = abs(y1 - y0);

  int16_t err = dx / 2;
  int16_t ystep;

  if (y0 < y1) {
    ystep = 1;
  } else {
    ystep = -1;
  }

  for (; x0 <= x1; x0++) {
    if (steep) {
      writePixel(y0, x0, color);
    } else {
      writePixel(x0, y0, color);
    }
    err -= dy;
    if (err < 0) {
      y0 += ystep;
      err += dx;
    }
  }
}

/***************************************************************************/
/*!
   @brief    Start a display-writing routine, overwrite in subclasses.
*/
/***************************************************************************/
void Adafruit_GFX::startWrite() {}

/***************************************************************************/
/*!
   @brief    Write a pixel, overwrite in subclasses if startWrite is defined!
    @param   x   x coordinate
    @param   y   y coordinate
   @param    color 16-bit 5-6-5 Color to fill with
*/
/***************************************************************************/
void Adafruit_GFX::writePixel(int16_t x, int16_t y, uint16_t color) {
  drawPixel(x, y, color);
}

/***************************************************************************/
/*!
   @brief    Write a perfectly vertical line, overwrite in subclasses if
   startWrite is defined!
    @param    x   Top-most x coordinate
    @param    y   Top-most y coordinate
    @param    h   Height in pixels
   @param    color 16-bit 5-6-5 Color to fill with
*/
/***************************************************************************/
void Adafruit_GFX::writeFastVLine(int16_t x, int16_t y, int16_t h,
                                  uint16_t color) {
  // Overwrite in subclasses if startWrite is defined!
  // Can be just writeLine(x, y, x, y+h-1, color);
  // or writeFillRect(x, y, 1, h, color);
  drawFastVLine(x, y, h, color);
}

/***************************************************************************/
/*!
   @brief    Write a perfectly horizontal line, overwrite in subclasses if
   startWrite is defined!
    @param    x   Left-most x coordinate
    @param    y   Left-most y coordinate
    @param    w   Width in pixels
   @param    color 16-bit 5-6-5 Color to fill with
*/
/***************************************************************************/
void Adafruit_GFX::writeFastHLine(int16_t x, int16_t y, int16_t w,
                                  uint16_t color) {
  // Overwrite in subclasses if startWrite is defined!
  // Example: writeLine(x, y, x+w-1, y, color);
  // or writeFillRect(x, y, w, 1, color);
  drawFastHLine(x, y, w, color);
}

/***************************************************************************/
/*!
   @brief    Write a rectangle completely with one color, overwrite in
   subclasses if startWrite is defined!
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    w   Width in pixels
    @param    h   Height in pixels
   @param    color 16-bit 5-6-5 Color to fill with
*/
/***************************************************************************/
void Adafruit_GFX::writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                                 uint16_t color) {
  // Overwrite in subclasses if desired!
  fillRect(x, y, w, h, color);
}

/***************************************************************************/
/*!
   @brief    End a display-writing routine, overwrite in subclasses if
```

```
   startWrite is defined!
*/
/**************************************************************************/
void Adafruit_GFX::endWrite() {}

/**************************************************************************/
/*!
   @brief    Draw a perfectly vertical line (this is often optimized in a
   subclass!)
    @param    x   Top-most x coordinate
    @param    y   Top-most y coordinate
    @param    h   Height in pixels
   @param    color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void Adafruit_GFX::drawFastVLine(int16_t x, int16_t y, int16_t h,
                                 uint16_t color) {
  startWrite();
  writeLine(x, y, x, y + h - 1, color);
  endWrite();
}

/**************************************************************************/
/*!
   @brief    Draw a perfectly horizontal line (this is often optimized in a
   subclass!)
    @param    x   Left-most x coordinate
    @param    y   Left-most y coordinate
    @param    w   Width in pixels
   @param    color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void Adafruit_GFX::drawFastHLine(int16_t x, int16_t y, int16_t w,
                                 uint16_t color) {
  startWrite();
  writeLine(x, y, x + w - 1, y, color);
  endWrite();
}

/**************************************************************************/
/*!
   @brief    Fill a rectangle completely with one color. Update in subclasses if
   desired!
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    w   Width in pixels
    @param    h   Height in pixels
   @param    color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void Adafruit_GFX::fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                            uint16_t color) {
  startWrite();
  for (int16_t i = x; i < x + w; i++) {
    writeFastVLine(i, y, h, color);
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief    Fill the screen completely with one color. Update in subclasses if
   desired!
    @param    color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void Adafruit_GFX::fillScreen(uint16_t color) {
  fillRect(0, 0, _width, _height, color);
}

/**************************************************************************/
/*!
   @brief    Draw a line
    @param    x0  Start point x coordinate
    @param    y0  Start point y coordinate
    @param    x1  End point x coordinate
    @param    y1  End point y coordinate
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawLine(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                            uint16_t color) {
  // Update in subclasses if desired!
  if (x0 == x1) {
    if (y0 > y1)
      _swap_int16_t(y0, y1);
    drawFastVLine(x0, y0, y1 - y0 + 1, color);
  } else if (y0 == y1) {
    if (x0 > x1)
      _swap_int16_t(x0, x1);
    drawFastHLine(x0, y0, x1 - x0 + 1, color);
  } else {
    startWrite();
    writeLine(x0, y0, x1, y1, color);
    endWrite();
  }
}

/**************************************************************************/
/*!
   @brief    Draw a circle outline
    @param    x0   Center-point x coordinate
    @param    y0   Center-point y coordinate
    @param    r   Radius of circle
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawCircle(int16_t x0, int16_t y0, int16_t r,
                              uint16_t color) {
#if defined(ESP8266)
  yield();
#endif
  int16_t f = 1 - r;
  int16_t ddF_x = 1;
  int16_t ddF_y = -2 * r;
  int16_t x = 0;
  int16_t y = r;
```

```
  startWrite();
  writePixel(x0, y0 + r, color);
  writePixel(x0, y0 - r, color);
  writePixel(x0 + r, y0, color);
  writePixel(x0 - r, y0, color);

  while (x < y) {
    if (f >= 0) {
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x;

    writePixel(x0 + x, y0 + y, color);
    writePixel(x0 - x, y0 + y, color);
    writePixel(x0 + x, y0 - y, color);
    writePixel(x0 - x, y0 - y, color);
    writePixel(x0 + y, y0 + x, color);
    writePixel(x0 - y, y0 + x, color);
    writePixel(x0 + y, y0 - x, color);
    writePixel(x0 - y, y0 - x, color);
  }
  endWrite();
}

/**************************************************************************/
/*!
    @brief    Quarter-circle drawer, used to do circles and roundrects
    @param    x0   Center-point x coordinate
    @param    y0   Center-point y coordinate
    @param    r   Radius of circle
    @param    cornername  Mask bit #1 or bit #2 to indicate which quarters of
   the circle we're doing
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawCircleHelper(int16_t x0, int16_t y0, int16_t r,
                                    uint8_t cornername, uint16_t color) {
  int16_t f = 1 - r;
  int16_t ddF_x = 1;
  int16_t ddF_y = -2 * r;
  int16_t x = 0;
  int16_t y = r;

  while (x < y) {
    if (f >= 0) {
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x;
    if (cornername & 0x4) {
      writePixel(x0 + x, y0 + y, color);
      writePixel(x0 + y, y0 + x, color);
    }
    if (cornername & 0x2) {
      writePixel(x0 + x, y0 - y, color);
      writePixel(x0 + y, y0 - x, color);
    }
    if (cornername & 0x8) {
      writePixel(x0 - y, y0 + x, color);
      writePixel(x0 - x, y0 + y, color);
    }
    if (cornername & 0x1) {
      writePixel(x0 - y, y0 - x, color);
      writePixel(x0 - x, y0 - y, color);
    }
  }
}

/**************************************************************************/
/*!
   @brief    Draw a circle with filled color
    @param    x0   Center-point x coordinate
    @param    y0   Center-point y coordinate
    @param    r   Radius of circle
    @param    color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void Adafruit_GFX::fillCircle(int16_t x0, int16_t y0, int16_t r,
                              uint16_t color) {
  startWrite();
  writeFastVLine(x0, y0 - r, 2 * r + 1, color);
  fillCircleHelper(x0, y0, r, 3, 0, color);
  endWrite();
}

/**************************************************************************/
/*!
    @brief  Quarter-circle drawer with fill, used for circles and roundrects
    @param  x0       Center-point x coordinate
    @param  y0       Center-point y coordinate
    @param  r        Radius of circle
    @param  corners  Mask bits indicating which quarters we're doing
    @param  delta    Offset from center-point, used for round-rects
    @param  color    16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void Adafruit_GFX::fillCircleHelper(int16_t x0, int16_t y0, int16_t r,
                                    uint8_t corners, int16_t delta,
                                    uint16_t color) {

  int16_t f = 1 - r;
  int16_t ddF_x = 1;
  int16_t ddF_y = -2 * r;
  int16_t x = 0;
  int16_t y = r;
  int16_t px = x;
  int16_t py = y;

  delta++; // Avoid some +1's in the loop
```

```
  while (x < y) {
    if (f >= 0) {
      y--;
      ddF_y += 2;
      f += ddF_y;
    }
    x++;
    ddF_x += 2;
    f += ddF_x;
    // These checks avoid double-drawing certain lines, important
    // for the SSD1306 library which has an INVERT drawing mode.
    if (x < (y + 1)) {
      if (corners & 1)
        writeFastVLine(x0 + x, y0 - y, 2 * y + delta, color);
      if (corners & 2)
        writeFastVLine(x0 - x, y0 - y, 2 * y + delta, color);
    }
    if (y != py) {
      if (corners & 1)
        writeFastVLine(x0 + py, y0 - px, 2 * px + delta, color);
      if (corners & 2)
        writeFastVLine(x0 - py, y0 - px, 2 * px + delta, color);
      py = y;
    }
    px = x;
  }
}

/**************************************************************************/
/*!
   @brief   Draw a rectangle with no fill color
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    w   Width in pixels
    @param    h   Height in pixels
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawRect(int16_t x, int16_t y, int16_t w, int16_t h,
                            uint16_t color) {
  startWrite();
  writeFastHLine(x, y, w, color);
  writeFastHLine(x, y + h - 1, w, color);
  writeFastVLine(x, y, h, color);
  writeFastVLine(x + w - 1, y, h, color);
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a rounded rectangle with no fill color
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    w   Width in pixels
    @param    h   Height in pixels
    @param    r   Radius of corner rounding
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawRoundRect(int16_t x, int16_t y, int16_t w, int16_t h,
                                 int16_t r, uint16_t color) {
  int16_t max_radius = ((w < h) ? w : h) / 2; // 1/2 minor axis
  if (r > max_radius)
    r = max_radius;
  // smarter version
  startWrite();
  writeFastHLine(x + r, y, w - 2 * r, color);         // Top
  writeFastHLine(x + r, y + h - 1, w - 2 * r, color); // Bottom
  writeFastVLine(x, y + r, h - 2 * r, color);         // Left
  writeFastVLine(x + w - 1, y + r, h - 2 * r, color); // Right
  // draw four corners
  drawCircleHelper(x + r, y + r, r, 1, color);
  drawCircleHelper(x + w - r - 1, y + r, r, 2, color);
  drawCircleHelper(x + w - r - 1, y + h - r - 1, r, 4, color);
  drawCircleHelper(x + r, y + h - r - 1, r, 8, color);
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a rounded rectangle with fill color
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    w   Width in pixels
    @param    h   Height in pixels
    @param    r   Radius of corner rounding
    @param    color 16-bit 5-6-5 Color to draw/fill with
*/
/**************************************************************************/
void Adafruit_GFX::fillRoundRect(int16_t x, int16_t y, int16_t w, int16_t h,
                                 int16_t r, uint16_t color) {
  int16_t max_radius = ((w < h) ? w : h) / 2; // 1/2 minor axis
  if (r > max_radius)
    r = max_radius;
  // smarter version
  startWrite();
  writeFillRect(x + r, y, w - 2 * r, h, color);
  // draw four corners
  fillCircleHelper(x + w - r - 1, y + r, r, 1, h - 2 * r - 1, color);
  fillCircleHelper(x + r, y + r, r, 2, h - 2 * r - 1, color);
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a triangle with no fill color
    @param    x0  Vertex #0 x coordinate
    @param    y0  Vertex #0 y coordinate
    @param    x1  Vertex #1 x coordinate
    @param    y1  Vertex #1 y coordinate
    @param    x2  Vertex #2 x coordinate
    @param    y2  Vertex #2 y coordinate
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
```

```
                              int16_t x2, int16_t y2, uint16_t color) {
  drawLine(x0, y0, x1, y1, color);
  drawLine(x1, y1, x2, y2, color);
  drawLine(x2, y2, x0, y0, color);
}

/**************************************************************************/
/*!
   @brief     Draw a triangle with color-fill
    @param    x0  Vertex #0 x coordinate
    @param    y0  Vertex #0 y coordinate
    @param    x1  Vertex #1 x coordinate
    @param    y1  Vertex #1 y coordinate
    @param    x2  Vertex #2 x coordinate
    @param    y2  Vertex #2 y coordinate
    @param    color 16-bit 5-6-5 Color to fill/draw with
*/
/**************************************************************************/
void Adafruit_GFX::fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1,
                              int16_t x2, int16_t y2, uint16_t color) {

  int16_t a, b, y, last;

  // Sort coordinates by Y order (y2 >= y1 >= y0)
  if (y0 > y1) {
    _swap_int16_t(y0, y1);
    _swap_int16_t(x0, x1);
  }
  if (y1 > y2) {
    _swap_int16_t(y2, y1);
    _swap_int16_t(x2, x1);
  }
  if (y0 > y1) {
    _swap_int16_t(y0, y1);
    _swap_int16_t(x0, x1);
  }

  startWrite();
  if (y0 == y2) { // Handle awkward all-on-same-line case as its own thing
    a = b = x0;
    if (x1 < a)
      a = x1;
    else if (x1 > b)
      b = x1;
    if (x2 < a)
      a = x2;
    else if (x2 > b)
      b = x2;
    writeFastHLine(a, y0, b - a + 1, color);
    endWrite();
    return;
  }

  int16_t dx01 = x1 - x0, dy01 = y1 - y0, dx02 = x2 - x0, dy02 = y2 - y0,
          dx12 = x2 - x1, dy12 = y2 - y1;
  int32_t sa = 0, sb = 0;

  // For upper part of triangle, find scanline crossings for segments
  // 0-1 and 0-2.  If y1=y2 (flat-bottomed triangle), the scanline y1
  // is included here (and second loop will be skipped, avoiding a /0
  // error there), otherwise scanline y1 is skipped here and handled
  // in the second loop...which also avoids a /0 error here if y0=y1
  // (flat-topped triangle).
  if (y1 == y2)
    last = y1; // Include y1 scanline
  else
    last = y1 - 1; // Skip it

  for (y = y0; y <= last; y++) {
    a = x0 + sa / dy01;
    b = x0 + sb / dy02;
    sa += dx01;
    sb += dx02;
    /* longhand:
    a = x0 + (x1 - x0) * (y - y0) / (y1 - y0);
    b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
    */
    if (a > b)
      _swap_int16_t(a, b);
    writeFastHLine(a, y, b - a + 1, color);
  }

  // For lower part of triangle, find scanline crossings for segments
  // 0-2 and 1-2.  This loop is skipped if y1=y2.
  sa = (int32_t)dx12 * (y - y1);
  sb = (int32_t)dx02 * (y - y0);
  for (; y <= y2; y++) {
    a = x1 + sa / dy12;
    b = x0 + sb / dy02;
    sa += dx12;
    sb += dx02;
    /* longhand:
    a = x1 + (x2 - x1) * (y - y1) / (y2 - y1);
    b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
    */
    if (a > b)
      _swap_int16_t(a, b);
    writeFastHLine(a, y, b - a + 1, color);
  }
  endWrite();
}

// BITMAP / XBITMAP / GRAYSCALE / RGB BITMAP FUNCTIONS ---------------------

/**************************************************************************/
/*!
   @brief      Draw a PROGMEM-resident 1-bit image at the specified (x,y)
   position, using the specified foreground color (unset bits are transparent).
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with monochrome bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
```

```
                              int16_t w, int16_t h, uint16_t color) {

  int16_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte
  uint8_t b = 0;

  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = pgm_read_byte(&bitmap[j * byteWidth + i / 8]);
      if (b & 0x80)
        writePixel(x + i, y, color);
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief      Draw a PROGMEM-resident 1-bit image at the specified (x,y)
   position, using the specified foreground (for set bits) and background (unset
   bits) colors.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with monochrome bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
    @param    color 16-bit 5-6-5 Color to draw pixels with
    @param    bg 16-bit 5-6-5 Color to draw background with
*/
/**************************************************************************/
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
                              int16_t w, int16_t h, uint16_t color,
                              uint16_t bg) {

  int16_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte
  uint8_t b = 0;

  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = pgm_read_byte(&bitmap[j * byteWidth + i / 8]);
      writePixel(x + i, y, (b & 0x80) ? color : bg);
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief      Draw a RAM-resident 1-bit image at the specified (x,y) position,
   using the specified foreground color (unset bits are transparent).
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with monochrome bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
    @param    color 16-bit 5-6-5 Color to draw with
*/
/**************************************************************************/
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w,
                              int16_t h, uint16_t color) {

  int16_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte
  uint8_t b = 0;

  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = bitmap[j * byteWidth + i / 8];
      if (b & 0x80)
        writePixel(x + i, y, color);
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief      Draw a RAM-resident 1-bit image at the specified (x,y) position,
   using the specified foreground (for set bits) and background (unset bits)
   colors.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with monochrome bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
    @param    color 16-bit 5-6-5 Color to draw pixels with
    @param    bg 16-bit 5-6-5 Color to draw background with
*/
/**************************************************************************/
void Adafruit_GFX::drawBitmap(int16_t x, int16_t y, uint8_t *bitmap, int16_t w,
                              int16_t h, uint16_t color, uint16_t bg) {

  int16_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte
  uint8_t b = 0;

  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = bitmap[j * byteWidth + i / 8];
      writePixel(x + i, y, (b & 0x80) ? color : bg);
    }
  }
  endWrite();
}
```

```
/**************************************************************************/
/*!
   @brief      Draw PROGMEM-resident XBitMap Files (*.xbm), exported from GIMP.
   Usage: Export from GIMP to *.xbm, rename *.xbm to *.c and open in editor.
   C Array can be directly used with this function.
   There is no RAM-resident version of this function; if generating bitmaps
   in RAM, use the format defined by drawBitmap() and call that instead.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with monochrome bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
    @param    color 16-bit 5-6-5 Color to draw pixels with
*/
/**************************************************************************/
void Adafruit_GFX::drawXBitmap(int16_t x, int16_t y, const uint8_t bitmap[],
                               int16_t w, int16_t h, uint16_t color) {

  int16_t byteWidth = (w + 7) / 8; // Bitmap scanline pad = whole byte
  uint8_t b = 0;

  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b >>= 1;
      else
        b = pgm_read_byte(&bitmap[j * byteWidth + i / 8]);
      // Nearly identical to drawBitmap(), only the bit order
      // is reversed here (left-to-right = LSB to MSB):
      if (b & 0x01)
        writePixel(x + i, y, color);
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a PROGMEM-resident 8-bit image (grayscale) at the specified
   (x,y) pos. Specifically for 8-bit display devices such as IS31FL3731; no
   color reduction/expansion is performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with grayscale bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawGrayscaleBitmap(int16_t x, int16_t y,
                                       const uint8_t bitmap[], int16_t w,
                                       int16_t h) {
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      writePixel(x + i, y, (uint8_t)pgm_read_byte(&bitmap[j * w + i]));
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a RAM-resident 8-bit image (grayscale) at the specified (x,y)
   pos. Specifically for 8-bit display devices such as IS31FL3731; no color
   reduction/expansion is performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with grayscale bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawGrayscaleBitmap(int16_t x, int16_t y, uint8_t *bitmap,
                                       int16_t w, int16_t h) {
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      writePixel(x + i, y, bitmap[j * w + i]);
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a PROGMEM-resident 8-bit image (grayscale) with a 1-bit mask
   (set bits = opaque, unset bits = clear) at the specified (x,y) position.
   BOTH buffers (grayscale and mask) must be PROGMEM-resident.
   Specifically for 8-bit display devices such as IS31FL3731; no color
   reduction/expansion is performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with grayscale bitmap
    @param    mask  byte array with mask bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawGrayscaleBitmap(int16_t x, int16_t y,
                                       const uint8_t bitmap[],
                                       const uint8_t mask[], int16_t w,
                                       int16_t h) {
  int16_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte
  uint8_t b = 0;
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = pgm_read_byte(&mask[j * bw + i / 8]);
      if (b & 0x80) {
        writePixel(x + i, y, (uint8_t)pgm_read_byte(&bitmap[j * w + i]));
      }
    }
```

```
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a RAM-resident 8-bit image (grayscale) with a 1-bit mask
   (set bits = opaque, unset bits = clear) at the specified (x,y) position.
   BOTH buffers (grayscale and mask) must be RAM-residentt, no mix-and-match
   Specifically for 8-bit display devices such as IS31FL3731; no color
   reduction/expansion is performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with grayscale bitmap
    @param    mask  byte array with mask bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawGrayscaleBitmap(int16_t x, int16_t y, uint8_t *bitmap,
                                       uint8_t *mask, int16_t w, int16_t h) {
  int16_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte
  uint8_t b = 0;
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = mask[j * bw + i / 8];
      if (b & 0x80) {
        writePixel(x + i, y, bitmap[j * w + i]);
      }
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a PROGMEM-resident 16-bit image (RGB 5/6/5) at the specified
   (x,y) position. For 16-bit display devices; no color reduction performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with 16-bit color bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[],
                                 int16_t w, int16_t h) {
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      writePixel(x + i, y, pgm_read_word(&bitmap[j * w + i]));
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a RAM-resident 16-bit image (RGB 5/6/5) at the specified (x,y)
   position. For 16-bit display devices; no color reduction performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with 16-bit color bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap,
                                 int16_t w, int16_t h) {
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      writePixel(x + i, y, bitmap[j * w + i]);
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
   @brief   Draw a PROGMEM-resident 16-bit image (RGB 5/6/5) with a 1-bit mask
   (set bits = opaque, unset bits = clear) at the specified (x,y) position. BOTH
   buffers (color and mask) must be PROGMEM-resident. For 16-bit display
   devices; no color reduction performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with 16-bit color bitmap
    @param    mask  byte array with monochrome mask bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawRGBBitmap(int16_t x, int16_t y, const uint16_t bitmap[],
                                 const uint8_t mask[], int16_t w, int16_t h) {
  int16_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte
  uint8_t b = 0;
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = pgm_read_byte(&mask[j * bw + i / 8]);
      if (b & 0x80) {
        writePixel(x + i, y, pgm_read_word(&bitmap[j * w + i]));
      }
    }
  }
  endWrite();
}

/**************************************************************************/
/*!
```

```
   @brief   Draw a RAM-resident 16-bit image (RGB 5/6/5) with a 1-bit mask (set
   bits = opaque, unset bits = clear) at the specified (x,y) position. BOTH
   buffers (color and mask) must be RAM-resident. For 16-bit display devices; no
   color reduction performed.
    @param    x   Top left corner x coordinate
    @param    y   Top left corner y coordinate
    @param    bitmap  byte array with 16-bit color bitmap
    @param    mask  byte array with monochrome mask bitmap
    @param    w   Width of bitmap in pixels
    @param    h   Height of bitmap in pixels
*/
/**************************************************************************/
void Adafruit_GFX::drawRGBBitmap(int16_t x, int16_t y, uint16_t *bitmap,
                                 uint8_t *mask, int16_t w, int16_t h) {
  int16_t bw = (w + 7) / 8; // Bitmask scanline pad = whole byte
  uint8_t b = 0;
  startWrite();
  for (int16_t j = 0; j < h; j++, y++) {
    for (int16_t i = 0; i < w; i++) {
      if (i & 7)
        b <<= 1;
      else
        b = mask[j * bw + i / 8];
      if (b & 0x80) {
        writePixel(x + i, y, bitmap[j * w + i]);
      }
    }
  }
  endWrite();
}

// TEXT- AND CHARACTER-HANDLING FUNCTIONS ----------------------------------

// Draw a character
/**************************************************************************/
/*!
   @brief   Draw a single character
    @param    x   Bottom left corner x coordinate
    @param    y   Bottom left corner y coordinate
    @param    c   The 8-bit font-indexed character (likely ascii)
    @param    color 16-bit 5-6-5 Color to draw chraracter with
    @param    bg 16-bit 5-6-5 Color to fill background with (if same as color,
   no background)
    @param    size  Font magnification level, 1 is 'original' size
*/
/**************************************************************************/
void Adafruit_GFX::drawChar(int16_t x, int16_t y, unsigned char c,
                            uint16_t color, uint16_t bg, uint8_t size) {
  drawChar(x, y, c, color, bg, size, size);
}

// Draw a character
/**************************************************************************/
/*!
   @brief   Draw a single character
    @param    x   Bottom left corner x coordinate
    @param    y   Bottom left corner y coordinate
    @param    c   The 8-bit font-indexed character (likely ascii)
    @param    color 16-bit 5-6-5 Color to draw chraracter with
    @param    bg 16-bit 5-6-5 Color to fill background with (if same as color,
   no background)
    @param    size_x  Font magnification level in X-axis, 1 is 'original' size
    @param    size_y  Font magnification level in Y-axis, 1 is 'original' size
*/
/**************************************************************************/
void Adafruit_GFX::drawChar(int16_t x, int16_t y, unsigned char c,
                            uint16_t color, uint16_t bg, uint8_t size_x,
                            uint8_t size_y) {

  if (!gfxFont) { // 'Classic' built-in font

    if ((x >= _width) ||              // Clip right
        (y >= _height) ||             // Clip bottom
        ((x + 6 * size_x - 1) < 0) || // Clip left
        ((y + 8 * size_y - 1) < 0))   // Clip top
      return;

    if (!_cp437 && (c >= 176))
      c++; // Handle 'classic' charset behavior

    startWrite();
    for (int8_t i = 0; i < 5; i++) { // Char bitmap = 5 columns
      uint8_t line = pgm_read_byte(&font[c * 5 + i]);
      for (int8_t j = 0; j < 8; j++, line >>= 1) {
        if (line & 1) {
          if (size_x == 1 && size_y == 1)
            writePixel(x + i, y + j, color);
          else
            writeFillRect(x + i * size_x, y + j * size_y, size_x, size_y,
                          color);
        } else if (bg != color) {
          if (size_x == 1 && size_y == 1)
            writePixel(x + i, y + j, bg);
          else
            writeFillRect(x + i * size_x, y + j * size_y, size_x, size_y, bg);
        }
      }
    }
    if (bg != color) { // If opaque, draw vertical line for last column
      if (size_x == 1 && size_y == 1)
        writeFastVLine(x + 5, y, 8, bg);
      else
        writeFillRect(x + 5 * size_x, y, size_x, 8 * size_y, bg);
    }
    endWrite();

  } else { // Custom font

    // Character is assumed previously filtered by write() to eliminate
    // newlines, returns, non-printable characters, etc.  Calling
    // drawChar() directly with 'bad' characters of font may cause mayhem!

    c -= (uint8_t)pgm_read_byte(&gfxFont->first);
    GFXglyph *glyph = pgm_read_glyph_ptr(gfxFont, c);
    uint8_t *bitmap = pgm_read_bitmap_ptr(gfxFont);

    uint16_t bo = pgm_read_word(&glyph->bitmapOffset);
```

```
    uint8_t w = pgm_read_byte(&glyph->width), h = pgm_read_byte(&glyph->height);
    int8_t xo = pgm_read_byte(&glyph->xOffset),
           yo = pgm_read_byte(&glyph->yOffset);
    uint8_t xx, yy, bits = 0, bit = 0;
    int16_t xo16 = 0, yo16 = 0;

    if (size_x > 1 || size_y > 1) {
      xo16 = xo;
      yo16 = yo;
    }

    // Todo: Add character clipping here

    // NOTE: THERE IS NO 'BACKGROUND' COLOR OPTION ON CUSTOM FONTS.
    // THIS IS ON PURPOSE AND BY DESIGN.  The background color feature
    // has typically been used with the 'classic' font to overwrite old
    // screen contents with new data.  This ONLY works because the
    // characters are a uniform size; it's not a sensible thing to do with
    // proportionally-spaced fonts with glyphs of varying sizes (and that
    // may overlap).  To replace previously-drawn text when using a custom
    // font, use the getTextBounds() function to determine the smallest
    // rectangle encompassing a string, erase the area with fillRect(),
    // then draw new text.  This WILL infortunately 'blink' the text, but
    // is unavoidable.  Drawing 'background' pixels will NOT fix this,
    // only creates a new set of problems.  Have an idea to work around
    // this (a canvas object type for MCUs that can afford the RAM and
    // displays supporting setAddrWindow() and pushColors()), but haven't
    // implemented this yet.

    startWrite();
    for (yy = 0; yy < h; yy++) {
      for (xx = 0; xx < w; xx++) {
        if (!(bit++ & 7)) {
          bits = pgm_read_byte(&bitmap[bo++]);
        }
        if (bits & 0x80) {
          if (size_x == 1 && size_y == 1) {
            writePixel(x + xo + xx, y + yo + yy, color);
          } else {
            writeFillRect(x + (xo16 + xx) * size_x, y + (yo16 + yy) * size_y,
                          size_x, size_y, color);
          }
        }
        bits <<= 1;
      }
    }
    endWrite();

  } // End classic vs custom font
}
/**************************************************************************/
/*!
    @brief  Print one byte/character of data, used to support print()
    @param  c  The 8-bit ascii character to write
*/
/**************************************************************************/
size_t Adafruit_GFX::write(uint8_t c) {
  if (!gfxFont) { // 'Classic' built-in font

    if (c == '\n') {              // Newline?
      cursor_x = 0;               // Reset x to zero,
      cursor_y += textsize_y * 8; // advance y one line
    } else if (c != '\r') {       // Ignore carriage returns
      if (wrap && ((cursor_x + textsize_x * 6) > _width)) { // Off right?
        cursor_x = 0;                                       // Reset x to zero,
        cursor_y += textsize_y * 8; // advance y one line
      }
      drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize_x,
               textsize_y);
      cursor_x += textsize_x * 6; // Advance x one char
    }

  } else { // Custom font

    if (c == '\n') {
      cursor_x = 0;
      cursor_y +=
          (int16_t)textsize_y * (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
    } else if (c != '\r') {
      uint8_t first = pgm_read_byte(&gfxFont->first);
      if ((c >= first) && (c <= (uint8_t)pgm_read_byte(&gfxFont->last))) {
        GFXglyph *glyph = pgm_read_glyph_ptr(gfxFont, c - first);
        uint8_t w = pgm_read_byte(&glyph->width),
                h = pgm_read_byte(&glyph->height);
        if ((w > 0) && (h > 0)) { // Is there an associated bitmap?
          int16_t xo = (int8_t)pgm_read_byte(&glyph->xOffset); // sic
          if (wrap && ((cursor_x + textsize_x * (xo + w)) > _width)) {
            cursor_x = 0;
            cursor_y += (int16_t)textsize_y *
                        (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
          }
          drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize_x,
                   textsize_y);
        }
        cursor_x +=
            (uint8_t)pgm_read_byte(&glyph->xAdvance) * (int16_t)textsize_x;
      }
    }
  }
  return 1;
}

/**************************************************************************/
/*!
    @brief   Set text 'magnification' size. Each increase in s makes 1 pixel
   that much bigger.
    @param  s  Desired text size. 1 is default 6x8, 2 is 12x16, 3 is 18x24, etc
*/
/**************************************************************************/
void Adafruit_GFX::setTextSize(uint8_t s) { setTextSize(s, s); }

/**************************************************************************/
/*!
    @brief   Set text 'magnification' size. Each increase in s makes 1 pixel
   that much bigger.
    @param  s_x  Desired text width magnification level in X-axis. 1 is default
    @param  s_y  Desired text width magnification level in Y-axis. 1 is default
```

```
*/
/**************************************************************************/
void Adafruit_GFX::setTextSize(uint8_t s_x, uint8_t s_y) {
  textsize_x = (s_x > 0) ? s_x : 1;
  textsize_y = (s_y > 0) ? s_y : 1;
}

/**************************************************************************/
/*!
    @brief      Set rotation setting for display
    @param  x   0 thru 3 corresponding to 4 cardinal rotations
*/
/**************************************************************************/
void Adafruit_GFX::setRotation(uint8_t x) {
  rotation = (x & 3);
  switch (rotation) {
  case 0:
  case 2:
    _width = WIDTH;
    _height = HEIGHT;
    break;
  case 1:
  case 3:
    _width = HEIGHT;
    _height = WIDTH;
    break;
  }
}

/**************************************************************************/
/*!
    @brief Set the font to display when print()ing, either custom or default
    @param  f  The GFXfont object, if NULL use built in 6x8 font
*/
/**************************************************************************/
void Adafruit_GFX::setFont(const GFXfont *f) {
  if (f) {          // Font struct pointer passed in?
    if (!gfxFont) { // And no current font struct?
      // Switching from classic to new font behavior.
      // Move cursor pos down 6 pixels so it's on baseline.
      cursor_y += 6;
    }
  } else if (gfxFont) { // NULL passed.  Current font struct defined?
    // Switching from new to classic font behavior.
    // Move cursor pos up 6 pixels so it's at top-left of char.
    cursor_y -= 6;
  }
  gfxFont = (GFXfont *)f;
}

/**************************************************************************/
/*!
    @brief  Helper to determine size of a character with current font/size.
            Broke this out as it's used by both the PROGMEM- and RAM-resident
            getTextBounds() functions.
    @param  c     The ASCII character in question
    @param  x     Pointer to x location of character. Value is modified by
                  this function to advance to next character.
    @param  y     Pointer to y location of character. Value is modified by
                  this function to advance to next character.
    @param  minx  Pointer to minimum X coordinate, passed in to AND returned
                  by this function -- this is used to incrementally build a
                  bounding rectangle for a string.
    @param  miny  Pointer to minimum Y coord, passed in AND returned.
    @param  maxx  Pointer to maximum X coord, passed in AND returned.
    @param  maxy  Pointer to maximum Y coord, passed in AND returned.
*/
/**************************************************************************/
void Adafruit_GFX::charBounds(unsigned char c, int16_t *x, int16_t *y,
                              int16_t *minx, int16_t *miny, int16_t *maxx,
                              int16_t *maxy) {

  if (gfxFont) {

    if (c == '\n') { // Newline?
      *x = 0;        // Reset x to zero, advance y by one line
      *y += textsize_y * (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
    } else if (c != '\r') { // Not a carriage return; is normal char
      uint8_t first = pgm_read_byte(&gfxFont->first),
              last = pgm_read_byte(&gfxFont->last);
      if ((c >= first) && (c <= last)) { // Char present in this font?
        GFXglyph *glyph = pgm_read_glyph_ptr(gfxFont, c - first);
        uint8_t gw = pgm_read_byte(&glyph->width),
                gh = pgm_read_byte(&glyph->height),
                xa = pgm_read_byte(&glyph->xAdvance);
        int8_t xo = pgm_read_byte(&glyph->xOffset),
               yo = pgm_read_byte(&glyph->yOffset);
        if (wrap && ((*x + (((int16_t)xo + gw) * textsize_x)) > _width)) {
          *x = 0; // Reset x to zero, advance y by one line
          *y += textsize_y * (uint8_t)pgm_read_byte(&gfxFont->yAdvance);
        }
        int16_t tsx = (int16_t)textsize_x, tsy = (int16_t)textsize_y,
                x1 = *x + xo * tsx, y1 = *y + yo * tsy, x2 = x1 + gw * tsx - 1,
                y2 = y1 + gh * tsy - 1;
        if (x1 < *minx)
          *minx = x1;
        if (y1 < *miny)
          *miny = y1;
        if (x2 > *maxx)
          *maxx = x2;
        if (y2 > *maxy)
          *maxy = y2;
        *x += xa * tsx;
      }
    }

  } else { // Default font

    if (c == '\n') {        // Newline?
      *x = 0;               // Reset x to zero,
      *y += textsize_y * 8; // advance y one line
      // min/max x/y unchaged -- that waits for next 'normal' character
    } else if (c != '\r') { // Normal char; ignore carriage returns
      if (wrap && ((*x + textsize_x * 6) > _width)) { // Off right?
        *x = 0;                                       // Reset x to zero,
        *y += textsize_y * 8;                         // advance y one line
      }
```

```
      int x2 = *x + textsize_x * 6 - 1, // Lower-right pixel of char
          y2 = *y + textsize_y * 8 - 1;
      if (x2 > *maxx)
        *maxx = x2; // Track max x, y
      if (y2 > *maxy)
        *maxy = y2;
      if (*x < *minx)
        *minx = *x; // Track min x, y
      if (*y < *miny)
        *miny = *y;
      *x += textsize_x * 6; // Advance x one char
    }
  }
}

/**************************************************************************/
/*!
    @brief  Helper to determine size of a string with current font/size.
            Pass string and a cursor position, returns UL corner and W,H.
    @param  str  The ASCII string to measure
    @param  x    The current cursor X
    @param  y    The current cursor Y
    @param  x1   The boundary X coordinate, returned by function
    @param  y1   The boundary Y coordinate, returned by function
    @param  w    The boundary width, returned by function
    @param  h    The boundary height, returned by function
*/
/**************************************************************************/
void Adafruit_GFX::getTextBounds(const char *str, int16_t x, int16_t y,
                                 int16_t *x1, int16_t *y1, uint16_t *w,
                                 uint16_t *h) {

  uint8_t c; // Current character
  int16_t minx = 0x7FFF, miny = 0x7FFF, maxx = -1, maxy = -1; // Bound rect
  // Bound rect is intentionally initialized inverted, so 1st char sets it

  *x1 = x; // Initial position is value passed in
  *y1 = y;
  *w = *h = 0; // Initial size is zero

  while ((c = *str++)) {
    // charBounds() modifies x/y to advance for each character,
    // and min/max x/y are updated to incrementally build bounding rect.
    charBounds(c, &x, &y, &minx, &miny, &maxx, &maxy);
  }

  if (maxx >= minx) {     // If legit string bounds were found...
    *x1 = minx;           // Update x1 to least X coord,
    *w = maxx - minx + 1; // And w to bound rect width
  }
  if (maxy >= miny) { // Same for height
    *y1 = miny;
    *h = maxy - miny + 1;
  }
}

/**************************************************************************/
/*!
    @brief    Helper to determine size of a string with current font/size. Pass
   string and a cursor position, returns UL corner and W,H.
    @param    str    The ascii string to measure (as an arduino String() class)
    @param    x      The current cursor X
    @param    y      The current cursor Y
    @param    x1     The boundary X coordinate, set by function
    @param    y1     The boundary Y coordinate, set by function
    @param    w      The boundary width, set by function
    @param    h      The boundary height, set by function
*/
/**************************************************************************/
void Adafruit_GFX::getTextBounds(const String &str, int16_t x, int16_t y,
                                 int16_t *x1, int16_t *y1, uint16_t *w,
                                 uint16_t *h) {
  if (str.length() != 0) {
    getTextBounds(const_cast<char *>(str.c_str()), x, y, x1, y1, w, h);
  }
}

/**************************************************************************/
/*!
    @brief    Helper to determine size of a PROGMEM string with current
   font/size. Pass string and a cursor position, returns UL corner and W,H.
    @param    str     The flash-memory ascii string to measure
    @param    x       The current cursor X
    @param    y       The current cursor Y
    @param    x1      The boundary X coordinate, set by function
    @param    y1      The boundary Y coordinate, set by function
    @param    w      The boundary width, set by function
    @param    h      The boundary height, set by function
*/
/**************************************************************************/
void Adafruit_GFX::getTextBounds(const __FlashStringHelper *str, int16_t x,
                                 int16_t y, int16_t *x1, int16_t *y1,
                                 uint16_t *w, uint16_t *h) {
  uint8_t *s = (uint8_t *)str, c;

  *x1 = x;
  *y1 = y;
  *w = *h = 0;

  int16_t minx = _width, miny = _height, maxx = -1, maxy = -1;

  while ((c = pgm_read_byte(s++)))
    charBounds(c, &x, &y, &minx, &miny, &maxx, &maxy);

  if (maxx >= minx) {
    *x1 = minx;
    *w = maxx - minx + 1;
  }
  if (maxy >= miny) {
    *y1 = miny;
    *h = maxy - miny + 1;
  }
}

/**************************************************************************/
/*!
    @brief      Invert the display (ideally using built-in hardware command)
```

```
    @param   i  True if you want to invert, false to make 'normal'
*/
/**************************************************************************/
void Adafruit_GFX::invertDisplay(bool i) {
  // Do nothing, must be subclassed if supported by hardware
  (void)i; // disable -Wunused-parameter warning
}

/**************************************************************************/

/**************************************************************************/
/*!
   @brief    Create a simple drawn button UI element
*/
/**************************************************************************/
Adafruit_GFX_Button::Adafruit_GFX_Button(void) { _gfx = 0; }

/**************************************************************************/
/*!
   @brief    Initialize button with our desired color/size/settings
   @param    gfx     Pointer to our display so we can draw to it!
   @param    x       The X coordinate of the center of the button
   @param    y       The Y coordinate of the center of the button
   @param    w       Width of the buttton
   @param    h       Height of the buttton
   @param    outline  Color of the outline (16-bit 5-6-5 standard)
   @param    fill  Color of the button fill (16-bit 5-6-5 standard)
   @param    textcolor  Color of the button label (16-bit 5-6-5 standard)
   @param    label  Ascii string of the text inside the button
   @param    textsize The font magnification of the label text
*/
/**************************************************************************/
// Classic initButton() function: pass center & size
void Adafruit_GFX_Button::initButton(Adafruit_GFX *gfx, int16_t x, int16_t y,
                                     uint16_t w, uint16_t h, uint16_t outline,
                                     uint16_t fill, uint16_t textcolor,
                                     char *label, uint8_t textsize) {
  // Tweak arguments and pass to the newer initButtonUL() function...
  initButtonUL(gfx, x - (w / 2), y - (h / 2), w, h, outline, fill, textcolor,
               label, textsize);
}

/**************************************************************************/
/*!
   @brief    Initialize button with our desired color/size/settings
   @param    gfx     Pointer to our display so we can draw to it!
   @param    x       The X coordinate of the center of the button
   @param    y       The Y coordinate of the center of the button
   @param    w       Width of the buttton
   @param    h       Height of the buttton
   @param    outline  Color of the outline (16-bit 5-6-5 standard)
   @param    fill  Color of the button fill (16-bit 5-6-5 standard)
   @param    textcolor  Color of the button label (16-bit 5-6-5 standard)
   @param    label  Ascii string of the text inside the button
   @param    textsize_x The font magnification in X-axis of the label text
   @param    textsize_y The font magnification in Y-axis of the label text
*/
/**************************************************************************/
// Classic initButton() function: pass center & size
void Adafruit_GFX_Button::initButton(Adafruit_GFX *gfx, int16_t x, int16_t y,
                                     uint16_t w, uint16_t h, uint16_t outline,
                                     uint16_t fill, uint16_t textcolor,
                                     char *label, uint8_t textsize_x,
                                     uint8_t textsize_y) {
  // Tweak arguments and pass to the newer initButtonUL() function...
  initButtonUL(gfx, x - (w / 2), y - (h / 2), w, h, outline, fill, textcolor,
               label, textsize_x, textsize_y);
}

/**************************************************************************/
/*!
   @brief    Initialize button with our desired color/size/settings, with
   upper-left coordinates
   @param    gfx     Pointer to our display so we can draw to it!
   @param    x1       The X coordinate of the Upper-Left corner of the button
   @param    y1       The Y coordinate of the Upper-Left corner of the button
   @param    w       Width of the buttton
   @param    h       Height of the buttton
   @param    outline  Color of the outline (16-bit 5-6-5 standard)
   @param    fill  Color of the button fill (16-bit 5-6-5 standard)
   @param    textcolor  Color of the button label (16-bit 5-6-5 standard)
   @param    label  Ascii string of the text inside the button
   @param    textsize The font magnification of the label text
*/
/**************************************************************************/
void Adafruit_GFX_Button::initButtonUL(Adafruit_GFX *gfx, int16_t x1,
                                       int16_t y1, uint16_t w, uint16_t h,
                                       uint16_t outline, uint16_t fill,
                                       uint16_t textcolor, char *label,
                                       uint8_t textsize) {
  initButtonUL(gfx, x1, y1, w, h, outline, fill, textcolor, label, textsize,
               textsize);
}

/**************************************************************************/
/*!
   @brief    Initialize button with our desired color/size/settings, with
   upper-left coordinates
   @param    gfx     Pointer to our display so we can draw to it!
   @param    x1       The X coordinate of the Upper-Left corner of the button
   @param    y1       The Y coordinate of the Upper-Left corner of the button
   @param    w       Width of the buttton
   @param    h       Height of the buttton
   @param    outline  Color of the outline (16-bit 5-6-5 standard)
   @param    fill  Color of the button fill (16-bit 5-6-5 standard)
   @param    textcolor  Color of the button label (16-bit 5-6-5 standard)
   @param    label  Ascii string of the text inside the button
   @param    textsize_x The font magnification in X-axis of the label text
   @param    textsize_y The font magnification in Y-axis of the label text
*/
/**************************************************************************/
void Adafruit_GFX_Button::initButtonUL(Adafruit_GFX *gfx, int16_t x1,
                                       int16_t y1, uint16_t w, uint16_t h,
                                       uint16_t outline, uint16_t fill,
                                       uint16_t textcolor, char *label,
                                       uint8_t textsize_x, uint8_t textsize_y) {
  _x1 = x1;
```

```
  _y1 = y1;
  _w = w;
  _h = h;
  _outlinecolor = outline;
  _fillcolor = fill;
  _textcolor = textcolor;
  _textsize_x = textsize_x;
  _textsize_y = textsize_y;
  _gfx = gfx;
  strncpy(_label, label, 9);
  _label[9] = 0; // strncpy does not place a null at the end.
                 // When 'label' is >9 characters, _label is not terminated.
}

/***************************************************************************/
/*!
   @brief    Draw the button on the screen
   @param    inverted Whether to draw with fill/text swapped to indicate
   'pressed'
*/
/***************************************************************************/
void Adafruit_GFX_Button::drawButton(bool inverted) {
  uint16_t fill, outline, text;

  if (!inverted) {
    fill = _fillcolor;
    outline = _outlinecolor;
    text = _textcolor;
  } else {
    fill = _textcolor;
    outline = _outlinecolor;
    text = _fillcolor;
  }

  uint8_t r = min(_w, _h) / 4; // Corner radius
  _gfx->fillRoundRect(_x1, _y1, _w, _h, r, fill);
  _gfx->drawRoundRect(_x1, _y1, _w, _h, r, outline);

  _gfx->setCursor(_x1 + (_w / 2) - (strlen(_label) * 3 * _textsize_x),
                  _y1 + (_h / 2) - (4 * _textsize_y));
  _gfx->setTextColor(text);
  _gfx->setTextSize(_textsize_x, _textsize_y);
  _gfx->print(_label);
}

/***************************************************************************/
/*!
    @brief    Helper to let us know if a coordinate is within the bounds of the
   button
    @param    x       The X coordinate to check
    @param    y       The Y coordinate to check
    @returns  True if within button graphics outline
*/
/***************************************************************************/
bool Adafruit_GFX_Button::contains(int16_t x, int16_t y) {
  return ((x >= _x1) && (x < (int16_t)(_x1 + _w)) && (y >= _y1) &&
          (y < (int16_t)(_y1 + _h)));
}

/***************************************************************************/
/*!
   @brief    Query whether the button was pressed since we last checked state
   @returns  True if was not-pressed before, now is.
*/
/***************************************************************************/
bool Adafruit_GFX_Button::justPressed() { return (currstate && !laststate); }

/***************************************************************************/
/*!
   @brief    Query whether the button was released since we last checked state
   @returns  True if was pressed before, now is not.
*/
/***************************************************************************/
bool Adafruit_GFX_Button::justReleased() { return (!currstate && laststate); }

// -------------------------------------------------------------------------

// GFXcanvas1, GFXcanvas8 and GFXcanvas16 (currently a WIP, don't get too
// comfy with the implementation) provide 1-, 8- and 16-bit offscreen
// canvases, the address of which can be passed to drawBitmap() or
// pushColors() (the latter appears only in a couple of GFX-subclassed TFT
// libraries at this time).  This is here mostly to help with the recently-
// added proportionally-spaced fonts; adds a way to refresh a section of the
// screen without a massive flickering clear-and-redraw...but maybe you'll
// find other uses too.  VERY RAM-intensive, since the buffer is in MCU
// memory and not the display driver...GXFcanvas1 might be minimally useful
// on an Uno-class board, but this and the others are much more likely to
// require at least a Mega or various recent ARM-type boards (recommended,
// as the text+bitmap draw can be pokey).  GFXcanvas1 requires 1 bit per
// pixel (rounded up to nearest byte per scanline), GFXcanvas8 is 1 byte
// per pixel (no scanline pad), and GFXcanvas16 uses 2 bytes per pixel (no
// scanline pad).
// NOT EXTENSIVELY TESTED YET.  MAY CONTAIN WORST BUGS KNOWN TO HUMANKIND.

#ifdef __AVR__
// Bitmask tables of 0x80>>X and ~(0x80>>X), because X>>Y is slow on AVR
const uint8_t PROGMEM GFXcanvas1::GFXsetBit[] = {0x80, 0x40, 0x20, 0x10,
                                                 0x08, 0x04, 0x02, 0x01};
const uint8_t PROGMEM GFXcanvas1::GFXclrBit[] = {0x7F, 0xBF, 0xDF, 0xEF,
                                                 0xF7, 0xFB, 0xFD, 0xFE};
#endif

/***************************************************************************/
/*!
   @brief    Instatiate a GFX 1-bit canvas context for graphics
   @param    w   Display width, in pixels
   @param    h   Display height, in pixels
*/
/***************************************************************************/
GFXcanvas1::GFXcanvas1(uint16_t w, uint16_t h) : Adafruit_GFX(w, h) {
  uint32_t bytes = ((w + 7) / 8) * h;
  if ((buffer = (uint8_t *)malloc(bytes))) {
    memset(buffer, 0, bytes);
  }
}

/***************************************************************************/
```

```
/*!
   @brief    Delete the canvas, free memory
*/
/**************************************************************************/
GFXcanvas1::~GFXcanvas1(void) {
  if (buffer)
    free(buffer);
}

/**************************************************************************/
/*!
    @brief  Draw a pixel to the canvas framebuffer
    @param  x     x coordinate
    @param  y     y coordinate
    @param  color Binary (on or off) color to fill with
*/
/**************************************************************************/
void GFXcanvas1::drawPixel(int16_t x, int16_t y, uint16_t color) {
  if (buffer) {
    if ((x < 0) || (y < 0) || (x >= _width) || (y >= _height))
      return;

    int16_t t;
    switch (rotation) {
    case 1:
      t = x;
      x = WIDTH - 1 - y;
      y = t;
      break;
    case 2:
      x = WIDTH - 1 - x;
      y = HEIGHT - 1 - y;
      break;
    case 3:
      t = x;
      x = y;
      y = HEIGHT - 1 - t;
      break;
    }

    uint8_t *ptr = &buffer[(x / 8) + y * ((WIDTH + 7) / 8)];
#ifdef __AVR__
    if (color)
      *ptr |= pgm_read_byte(&GFXsetBit[x & 7]);
    else
      *ptr &= pgm_read_byte(&GFXclrBit[x & 7]);
#else
    if (color)
      *ptr |= 0x80 >> (x & 7);
    else
      *ptr &= ~(0x80 >> (x & 7));
#endif
  }
}

/**************************************************************************/
/*!
        @brief    Get the pixel color value at a given coordinate
        @param    x   x coordinate
        @param    y   y coordinate
        @returns  The desired pixel's binary color value, either 0x1 (on) or 0x0
   (off)
*/
/**************************************************************************/
bool GFXcanvas1::getPixel(int16_t x, int16_t y) const {
  int16_t t;
  switch (rotation) {
  case 1:
    t = x;
    x = WIDTH - 1 - y;
    y = t;
    break;
  case 2:
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;
    break;
  case 3:
    t = x;
    x = y;
    y = HEIGHT - 1 - t;
    break;
  }
  return getRawPixel(x, y);
}

/**************************************************************************/
/*!
        @brief    Get the pixel color value at a given, unrotated coordinate.
              This method is intended for hardware drivers to get pixel value
              in physical coordinates.
        @param    x   x coordinate
        @param    y   y coordinate
        @returns  The desired pixel's binary color value, either 0x1 (on) or 0x0
   (off)
*/
/**************************************************************************/
bool GFXcanvas1::getRawPixel(int16_t x, int16_t y) const {
  if ((x < 0) || (y < 0) || (x >= WIDTH) || (y >= HEIGHT))
    return 0;
  if (buffer) {
    uint8_t *ptr = &buffer[(x / 8) + y * ((WIDTH + 7) / 8)];

#ifdef __AVR__
    return ((*ptr) & pgm_read_byte(&GFXsetBit[x & 7])) != 0;
#else
    return ((*ptr) & (0x80 >> (x & 7))) != 0;
#endif
  }
  return 0;
}

/**************************************************************************/
/*!
    @brief  Fill the framebuffer completely with one color
    @param  color Binary (on or off) color to fill with
*/
/**************************************************************************/
```

```
void GFXcanvas1::fillScreen(uint16_t color) {
  if (buffer) {
    uint32_t bytes = ((WIDTH + 7) / 8) * HEIGHT;
    memset(buffer, color ? 0xFF : 0x00, bytes);
  }
}

/**************************************************************************/
/*!
   @brief  Speed optimized vertical line drawing
   @param  x      Line horizontal start point
   @param  y      Line vertical start point
   @param  h      Length of vertical line to be drawn, including first point
   @param  color  Color to fill with
*/
/**************************************************************************/
void GFXcanvas1::drawFastVLine(int16_t x, int16_t y, int16_t h,
                               uint16_t color) {

  if (h < 0) { // Convert negative heights to positive equivalent
    h *= -1;
    y -= h - 1;
    if (y < 0) {
      h += y;
      y = 0;
    }
  }

  // Edge rejection (no-draw if totally off canvas)
  if ((x < 0) || (x >= width()) || (y >= height()) || ((y + h - 1) < 0)) {
    return;
  }

  if (y < 0) { // Clip top
    h += y;
    y = 0;
  }
  if (y + h > height()) { // Clip bottom
    h = height() - y;
  }

  if (getRotation() == 0) {
    drawFastRawVLine(x, y, h, color);
  } else if (getRotation() == 1) {
    int16_t t = x;
    x = WIDTH - 1 - y;
    y = t;
    x -= h - 1;
    drawFastRawHLine(x, y, h, color);
  } else if (getRotation() == 2) {
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;

    y -= h - 1;
    drawFastRawVLine(x, y, h, color);
  } else if (getRotation() == 3) {
    int16_t t = x;
    x = y;
    y = HEIGHT - 1 - t;
    drawFastRawHLine(x, y, h, color);
  }
}

/**************************************************************************/
/*!
   @brief  Speed optimized horizontal line drawing
   @param  x      Line horizontal start point
   @param  y      Line vertical start point
   @param  w      Length of horizontal line to be drawn, including first point
   @param  color  Color to fill with
*/
/**************************************************************************/
void GFXcanvas1::drawFastHLine(int16_t x, int16_t y, int16_t w,
                               uint16_t color) {
  if (w < 0) { // Convert negative widths to positive equivalent
    w *= -1;
    x -= w - 1;
    if (x < 0) {
      w += x;
      x = 0;
    }
  }

  // Edge rejection (no-draw if totally off canvas)
  if ((y < 0) || (y >= height()) || (x >= width()) || ((x + w - 1) < 0)) {
    return;
  }

  if (x < 0) { // Clip left
    w += x;
    x = 0;
  }
  if (x + w >= width()) { // Clip right
    w = width() - x;
  }

  if (getRotation() == 0) {
    drawFastRawHLine(x, y, w, color);
  } else if (getRotation() == 1) {
    int16_t t = x;
    x = WIDTH - 1 - y;
    y = t;
    drawFastRawVLine(x, y, w, color);
  } else if (getRotation() == 2) {
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;

    x -= w - 1;
    drawFastRawHLine(x, y, w, color);
  } else if (getRotation() == 3) {
    int16_t t = x;
    x = y;
    y = HEIGHT - 1 - t;
    y -= w - 1;
    drawFastRawVLine(x, y, w, color);
  }
}
```

```
/**************************************************************************/
/*!
   @brief    Speed optimized vertical line drawing into the raw canvas buffer
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    h   length of vertical line to be drawn, including first point
   @param    color   Binary (on or off) color to fill with
*/
/**************************************************************************/
void GFXcanvas1::drawFastRawVLine(int16_t x, int16_t y, int16_t h,
                                  uint16_t color) {
  // x & y already in raw (rotation 0) coordinates, no need to transform.
  int16_t row_bytes = ((WIDTH + 7) / 8);
  uint8_t *ptr = &buffer[(x / 8) + y * row_bytes];

  if (color > 0) {
#ifdef __AVR__
    uint8_t bit_mask = pgm_read_byte(&GFXsetBit[x & 7]);
#else
    uint8_t bit_mask = (0x80 >> (x & 7));
#endif
    for (int16_t i = 0; i < h; i++) {
      *ptr |= bit_mask;
      ptr += row_bytes;
    }
  } else {
#ifdef __AVR__
    uint8_t bit_mask = pgm_read_byte(&GFXclrBit[x & 7]);
#else
    uint8_t bit_mask = ~(0x80 >> (x & 7));
#endif
    for (int16_t i = 0; i < h; i++) {
      *ptr &= bit_mask;
      ptr += row_bytes;
    }
  }
}

/**************************************************************************/
/*!
   @brief    Speed optimized horizontal line drawing into the raw canvas buffer
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    w   length of horizontal line to be drawn, including first point
   @param    color   Binary (on or off) color to fill with
*/
/**************************************************************************/
void GFXcanvas1::drawFastRawHLine(int16_t x, int16_t y, int16_t w,
                                  uint16_t color) {
  // x & y already in raw (rotation 0) coordinates, no need to transform.
  int16_t rowBytes = ((WIDTH + 7) / 8);
  uint8_t *ptr = &buffer[(x / 8) + y * rowBytes];
  size_t remainingWidthBits = w;

  // check to see if first byte needs to be partially filled
  if ((x & 7) > 0) {
    // create bit mask for first byte
    uint8_t startByteBitMask = 0x00;
    for (int8_t i = (x & 7); ((i < 8) && (remainingWidthBits > 0)); i++) {
#ifdef __AVR__
      startByteBitMask |= pgm_read_byte(&GFXsetBit[i]);
#else
      startByteBitMask |= (0x80 >> i);
#endif
      remainingWidthBits--;
    }
    if (color > 0) {
      *ptr |= startByteBitMask;
    } else {
      *ptr &= ~startByteBitMask;
    }

    ptr++;
  }

  // do the next remainingWidthBits bits
  if (remainingWidthBits > 0) {
    size_t remainingWholeBytes = remainingWidthBits / 8;
    size_t lastByteBits = remainingWidthBits % 8;
    uint8_t wholeByteColor = color > 0 ? 0xFF : 0x00;

    memset(ptr, wholeByteColor, remainingWholeBytes);

    if (lastByteBits > 0) {
      uint8_t lastByteBitMask = 0x00;
      for (size_t i = 0; i < lastByteBits; i++) {
#ifdef __AVR__
        lastByteBitMask |= pgm_read_byte(&GFXsetBit[i]);
#else
        lastByteBitMask |= (0x80 >> i);
#endif
      }
      ptr += remainingWholeBytes;

      if (color > 0) {
        *ptr |= lastByteBitMask;
      } else {
        *ptr &= ~lastByteBitMask;
      }
    }
  }
}

/**************************************************************************/
/*!
   @brief    Instatiate a GFX 8-bit canvas context for graphics
   @param    w   Display width, in pixels
   @param    h   Display height, in pixels
*/
/**************************************************************************/
GFXcanvas8::GFXcanvas8(uint16_t w, uint16_t h) : Adafruit_GFX(w, h) {
  uint32_t bytes = w * h;
  if ((buffer = (uint8_t *)malloc(bytes))) {
    memset(buffer, 0, bytes);
  }
}
```

```
/**************************************************************************/
/*!
   @brief    Delete the canvas, free memory
*/
/**************************************************************************/
GFXcanvas8::~GFXcanvas8(void) {
  if (buffer)
    free(buffer);
}

/**************************************************************************/
/*!
    @brief  Draw a pixel to the canvas framebuffer
    @param  x   x coordinate
    @param  y   y coordinate
    @param  color 8-bit Color to fill with. Only lower byte of uint16_t is used.
*/
/**************************************************************************/
void GFXcanvas8::drawPixel(int16_t x, int16_t y, uint16_t color) {
  if (buffer) {
    if ((x < 0) || (y < 0) || (x >= _width) || (y >= _height))
      return;

    int16_t t;
    switch (rotation) {
    case 1:
      t = x;
      x = WIDTH - 1 - y;
      y = t;
      break;
    case 2:
      x = WIDTH - 1 - x;
      y = HEIGHT - 1 - y;
      break;
    case 3:
      t = x;
      x = y;
      y = HEIGHT - 1 - t;
      break;
    }

    buffer[x + y * WIDTH] = color;
  }
}

/**************************************************************************/
/*!
        @brief    Get the pixel color value at a given coordinate
        @param    x   x coordinate
        @param    y   y coordinate
        @returns  The desired pixel's 8-bit color value
*/
/**************************************************************************/
uint8_t GFXcanvas8::getPixel(int16_t x, int16_t y) const {
  int16_t t;
  switch (rotation) {
  case 1:
    t = x;
    x = WIDTH - 1 - y;
    y = t;
    break;
  case 2:
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;
    break;
  case 3:
    t = x;
    x = y;
    y = HEIGHT - 1 - t;
    break;
  }
  return getRawPixel(x, y);
}

/**************************************************************************/
/*!
        @brief    Get the pixel color value at a given, unrotated coordinate.
              This method is intended for hardware drivers to get pixel value
              in physical coordinates.
        @param    x   x coordinate
        @param    y   y coordinate
        @returns  The desired pixel's 8-bit color value
*/
/**************************************************************************/
uint8_t GFXcanvas8::getRawPixel(int16_t x, int16_t y) const {
  if ((x < 0) || (y < 0) || (x >= WIDTH) || (y >= HEIGHT))
    return 0;
  if (buffer) {
    return buffer[x + y * WIDTH];
  }
  return 0;
}

/**************************************************************************/
/*!
    @brief  Fill the framebuffer completely with one color
    @param  color 8-bit Color to fill with. Only lower byte of uint16_t is used.
*/
/**************************************************************************/
void GFXcanvas8::fillScreen(uint16_t color) {
  if (buffer) {
    memset(buffer, color, WIDTH * HEIGHT);
  }
}

/**************************************************************************/
/*!
   @brief  Speed optimized vertical line drawing
   @param  x      Line horizontal start point
   @param  y      Line vertical start point
   @param  h      Length of vertical line to be drawn, including first point
   @param  color  8-bit Color to fill with. Only lower byte of uint16_t is
                  used.
*/
/**************************************************************************/
void GFXcanvas8::drawFastVLine(int16_t x, int16_t y, int16_t h,
```

```
                                uint16_t color) {
  if (h < 0) { // Convert negative heights to positive equivalent
    h *= -1;
    y -= h - 1;
    if (y < 0) {
      h += y;
      y = 0;
    }
  }

  // Edge rejection (no-draw if totally off canvas)
  if ((x < 0) || (x >= width()) || (y >= height()) || ((y + h - 1) < 0)) {
    return;
  }

  if (y < 0) { // Clip top
    h += y;
    y = 0;
  }
  if (y + h > height()) { // Clip bottom
    h = height() - y;
  }

  if (getRotation() == 0) {
    drawFastRawVLine(x, y, h, color);
  } else if (getRotation() == 1) {
    int16_t t = x;
    x = WIDTH - 1 - y;
    y = t;
    x -= h - 1;
    drawFastRawHLine(x, y, h, color);
  } else if (getRotation() == 2) {
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;

    y -= h - 1;
    drawFastRawVLine(x, y, h, color);
  } else if (getRotation() == 3) {
    int16_t t = x;
    x = y;
    y = HEIGHT - 1 - t;
    drawFastRawHLine(x, y, h, color);
  }
}

/***************************************************************************/
/*!
   @brief  Speed optimized horizontal line drawing
   @param  x      Line horizontal start point
   @param  y      Line vertical start point
   @param  w      Length of horizontal line to be drawn, including 1st point
   @param  color  8-bit Color to fill with. Only lower byte of uint16_t is
                  used.
*/
/***************************************************************************/
void GFXcanvas8::drawFastHLine(int16_t x, int16_t y, int16_t w,
                               uint16_t color) {

  if (w < 0) { // Convert negative widths to positive equivalent
    w *= -1;
    x -= w - 1;
    if (x < 0) {
      w += x;
      x = 0;
    }
  }

  // Edge rejection (no-draw if totally off canvas)
  if ((y < 0) || (y >= height()) || (x >= width()) || ((x + w - 1) < 0)) {
    return;
  }

  if (x < 0) { // Clip left
    w += x;
    x = 0;
  }
  if (x + w >= width()) { // Clip right
    w = width() - x;
  }

  if (getRotation() == 0) {
    drawFastRawHLine(x, y, w, color);
  } else if (getRotation() == 1) {
    int16_t t = x;
    x = WIDTH - 1 - y;
    y = t;
    drawFastRawVLine(x, y, w, color);
  } else if (getRotation() == 2) {
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;

    x -= w - 1;
    drawFastRawHLine(x, y, w, color);
  } else if (getRotation() == 3) {
    int16_t t = x;
    x = y;
    y = HEIGHT - 1 - t;
    y -= w - 1;
    drawFastRawVLine(x, y, w, color);
  }
}

/***************************************************************************/
/*!
   @brief    Speed optimized vertical line drawing into the raw canvas buffer
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    h   length of vertical line to be drawn, including first point
   @param    color   8-bit Color to fill with. Only lower byte of uint16_t is
   used.
*/
/***************************************************************************/
void GFXcanvas8::drawFastRawVLine(int16_t x, int16_t y, int16_t h,
                                  uint16_t color) {
  // x & y already in raw (rotation 0) coordinates, no need to transform.
  uint8_t *buffer_ptr = buffer + y * WIDTH + x;
  for (int16_t i = 0; i < h; i++) {
```

```
    (*buffer_ptr) = color;
    buffer_ptr += WIDTH;
  }
}

/**************************************************************************/
/*!
   @brief    Speed optimized horizontal line drawing into the raw canvas buffer
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    w   length of horizontal line to be drawn, including first point
   @param    color   8-bit Color to fill with. Only lower byte of uint16_t is
   used.
*/
/**************************************************************************/
void GFXcanvas8::drawFastRawHLine(int16_t x, int16_t y, int16_t w,
                                  uint16_t color) {
  // x & y already in raw (rotation 0) coordinates, no need to transform.
  memset(buffer + y * WIDTH + x, color, w);
}

/**************************************************************************/
/*!
   @brief    Instatiate a GFX 16-bit canvas context for graphics
   @param    w   Display width, in pixels
   @param    h   Display height, in pixels
*/
/**************************************************************************/
GFXcanvas16::GFXcanvas16(uint16_t w, uint16_t h) : Adafruit_GFX(w, h) {
  uint32_t bytes = w * h * 2;
  if ((buffer = (uint16_t *)malloc(bytes))) {
    memset(buffer, 0, bytes);
  }
}

/**************************************************************************/
/*!
   @brief    Delete the canvas, free memory
*/
/**************************************************************************/
GFXcanvas16::~GFXcanvas16(void) {
  if (buffer)
    free(buffer);
}

/**************************************************************************/
/*!
    @brief  Draw a pixel to the canvas framebuffer
    @param  x   x coordinate
    @param  y   y coordinate
    @param  color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void GFXcanvas16::drawPixel(int16_t x, int16_t y, uint16_t color) {
  if (buffer) {
    if ((x < 0) || (y < 0) || (x >= _width) || (y >= _height))
      return;

    int16_t t;
    switch (rotation) {
    case 1:
      t = x;
      x = WIDTH - 1 - y;
      y = t;
      break;
    case 2:
      x = WIDTH - 1 - x;
      y = HEIGHT - 1 - y;
      break;
    case 3:
      t = x;
      x = y;
      y = HEIGHT - 1 - t;
      break;
    }

    buffer[x + y * WIDTH] = color;
  }
}

/**************************************************************************/
/*!
        @brief    Get the pixel color value at a given coordinate
        @param    x   x coordinate
        @param    y   y coordinate
        @returns  The desired pixel's 16-bit 5-6-5 color value
*/
/**************************************************************************/
uint16_t GFXcanvas16::getPixel(int16_t x, int16_t y) const {
  int16_t t;
  switch (rotation) {
  case 1:
    t = x;
    x = WIDTH - 1 - y;
    y = t;
    break;
  case 2:
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;
    break;
  case 3:
    t = x;
    x = y;
    y = HEIGHT - 1 - t;
    break;
  }
  return getRawPixel(x, y);
}

/**************************************************************************/
/*!
        @brief    Get the pixel color value at a given, unrotated coordinate.
              This method is intended for hardware drivers to get pixel value
              in physical coordinates.
        @param    x   x coordinate
        @param    y   y coordinate
        @returns  The desired pixel's 16-bit 5-6-5 color value
```

```
*/
/**************************************************************************/
uint16_t GFXcanvas16::getRawPixel(int16_t x, int16_t y) const {
  if ((x < 0) || (y < 0) || (x >= WIDTH) || (y >= HEIGHT))
    return 0;
  if (buffer) {
    return buffer[x + y * WIDTH];
  }
  return 0;
}

/**************************************************************************/
/*!
    @brief  Fill the framebuffer completely with one color
    @param  color 16-bit 5-6-5 Color to fill with
*/
/**************************************************************************/
void GFXcanvas16::fillScreen(uint16_t color) {
  if (buffer) {
    uint8_t hi = color >> 8, lo = color & 0xFF;
    if (hi == lo) {
      memset(buffer, lo, WIDTH * HEIGHT * 2);
    } else {
      uint32_t i, pixels = WIDTH * HEIGHT;
      for (i = 0; i < pixels; i++)
        buffer[i] = color;
    }
  }
}

/**************************************************************************/
/*!
    @brief  Reverses the "endian-ness" of each 16-bit pixel within the
            canvas; little-endian to big-endian, or big-endian to little.
            Most microcontrollers (such as SAMD) are little-endian, while
            most displays tend toward big-endianness. All the drawing
            functions (including RGB bitmap drawing) take care of this
            automatically, but some specialized code (usually involving
            DMA) can benefit from having pixel data already in the
            display-native order. Note that this does NOT convert to a
            SPECIFIC endian-ness, it just flips the bytes within each word.
*/
/**************************************************************************/
void GFXcanvas16::byteSwap(void) {
  if (buffer) {
    uint32_t i, pixels = WIDTH * HEIGHT;
    for (i = 0; i < pixels; i++)
      buffer[i] = __builtin_bswap16(buffer[i]);
  }
}

/**************************************************************************/
/*!
   @brief    Speed optimized vertical line drawing
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    h   length of vertical line to be drawn, including first point
   @param    color   color 16-bit 5-6-5 Color to draw line with
*/
/**************************************************************************/
void GFXcanvas16::drawFastVLine(int16_t x, int16_t y, int16_t h,
                                uint16_t color) {
  if (h < 0) { // Convert negative heights to positive equivalent
    h *= -1;
    y -= h - 1;
    if (y < 0) {
      h += y;
      y = 0;
    }
  }

  // Edge rejection (no-draw if totally off canvas)
  if ((x < 0) || (x >= width()) || (y >= height()) || ((y + h - 1) < 0)) {
    return;
  }

  if (y < 0) { // Clip top
    h += y;
    y = 0;
  }
  if (y + h > height()) { // Clip bottom
    h = height() - y;
  }

  if (getRotation() == 0) {
    drawFastRawVLine(x, y, h, color);
  } else if (getRotation() == 1) {
    int16_t t = x;
    x = WIDTH - 1 - y;
    y = t;
    x -= h - 1;
    drawFastRawHLine(x, y, h, color);
  } else if (getRotation() == 2) {
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;

    y -= h - 1;
    drawFastRawVLine(x, y, h, color);
  } else if (getRotation() == 3) {
    int16_t t = x;
    x = y;
    y = HEIGHT - 1 - t;
    drawFastRawHLine(x, y, h, color);
  }
}

/**************************************************************************/
/*!
   @brief  Speed optimized horizontal line drawing
   @param  x      Line horizontal start point
   @param  y      Line vertical start point
   @param  w      Length of horizontal line to be drawn, including 1st point
   @param  color  Color 16-bit 5-6-5 Color to draw line with
*/
/**************************************************************************/
void GFXcanvas16::drawFastHLine(int16_t x, int16_t y, int16_t w,
                                uint16_t color) {
```

```
  if (w < 0) { // Convert negative widths to positive equivalent
    w *= -1;
    x -= w - 1;
    if (x < 0) {
      w += x;
      x = 0;
    }
  }

  // Edge rejection (no-draw if totally off canvas)
  if ((y < 0) || (y >= height()) || (x >= width()) || ((x + w - 1) < 0)) {
    return;
  }

  if (x < 0) { // Clip left
    w += x;
    x = 0;
  }
  if (x + w >= width()) { // Clip right
    w = width() - x;
  }

  if (getRotation() == 0) {
    drawFastRawHLine(x, y, w, color);
  } else if (getRotation() == 1) {
    int16_t t = x;
    x = WIDTH - 1 - y;
    y = t;
    drawFastRawVLine(x, y, w, color);
  } else if (getRotation() == 2) {
    x = WIDTH - 1 - x;
    y = HEIGHT - 1 - y;

    x -= w - 1;
    drawFastRawHLine(x, y, w, color);
  } else if (getRotation() == 3) {
    int16_t t = x;
    x = y;
    y = HEIGHT - 1 - t;
    y -= w - 1;
    drawFastRawVLine(x, y, w, color);
  }
}

/***************************************************************************/
/*!
   @brief    Speed optimized vertical line drawing into the raw canvas buffer
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    h   length of vertical line to be drawn, including first point
   @param    color   color 16-bit 5-6-5 Color to draw line with
*/
/***************************************************************************/
void GFXcanvas16::drawFastRawVLine(int16_t x, int16_t y, int16_t h,
                                   uint16_t color) {
  // x & y already in raw (rotation 0) coordinates, no need to transform.
  uint16_t *buffer_ptr = buffer + y * WIDTH + x;
  for (int16_t i = 0; i < h; i++) {
    (*buffer_ptr) = color;
    buffer_ptr += WIDTH;
  }
}

/***************************************************************************/
/*!
   @brief    Speed optimized horizontal line drawing into the raw canvas buffer
   @param    x   Line horizontal start point
   @param    y   Line vertical start point
   @param    w   length of horizontal line to be drawn, including first point
   @param    color   color 16-bit 5-6-5 Color to draw line with
*/
/***************************************************************************/
void GFXcanvas16::drawFastRawHLine(int16_t x, int16_t y, int16_t w,
                                   uint16_t color) {
  // x & y already in raw (rotation 0) coordinates, no need to transform.
  uint32_t buffer_index = y * WIDTH + x;
  for (uint32_t i = buffer_index; i < buffer_index + w; i++) {
    buffer[i] = color;
  }
}
```

***** SPI.h

```
/*
 * Copyright (c) 2010 by Cristian Maglie <c.maglie@arduino.cc>
 * Copyright (c) 2014 by Paul Stoffregen <paul@pjrc.com> (Transaction API)
 * Copyright (c) 2014 by Matthijs Kooijman <matthijs@stdin.nl> (SPISettings AVR)
 * Copyright (c) 2014 by Andrew J. Kroll <xxxajk@gmail.com> (atomicity fixes)
 * SPI Master library for arduino.
 *
 * This file is free software; you can redistribute it and/or modify
 * it under the terms of either the GNU General Public License version 2
 * or the GNU Lesser General Public License version 2.1, both as
 * published by the Free Software Foundation.
 */

#ifndef _SPI_H_INCLUDED
#define _SPI_H_INCLUDED

#include <Arduino.h>

// SPI_HAS_TRANSACTION means SPI has beginTransaction(), endTransaction(),
// usingInterrupt(), and SPISetting(clock, bitOrder, dataMode)
#define SPI_HAS_TRANSACTION 1

// SPI_HAS_NOTUSINGINTERRUPT means that SPI has notUsingInterrupt() method
#define SPI_HAS_NOTUSINGINTERRUPT 1

// SPI_ATOMIC_VERSION means that SPI has atomicity fixes and what version.
// This way when there is a bug fix you can check this define to alert users
// of your code if it uses better version of this library.
// This also implies everything that SPI_HAS_TRANSACTION as documented above is
// available too.
#define SPI_ATOMIC_VERSION 1

// Uncomment this line to add detection of mismatched begin/end transactions.
```

```
// A mismatch occurs if other libraries fail to use SPI.endTransaction() for
// each SPI.beginTransaction().  Connect an LED to this pin.  The LED will turn
// on if any mismatch is ever detected.
//#define SPI_TRANSACTION_MISMATCH_LED 5

#ifndef LSBFIRST
#define LSBFIRST 0
#endif
#ifndef MSBFIRST
#define MSBFIRST 1
#endif

#define SPI_CLOCK_DIV4 0x00
#define SPI_CLOCK_DIV16 0x01
#define SPI_CLOCK_DIV64 0x02
#define SPI_CLOCK_DIV128 0x03
#define SPI_CLOCK_DIV2 0x04
#define SPI_CLOCK_DIV8 0x05
#define SPI_CLOCK_DIV32 0x06

#define SPI_MODE0 0x00
#define SPI_MODE1 0x04
#define SPI_MODE2 0x08
#define SPI_MODE3 0x0C

#define SPI_MODE_MASK 0x0C  // CPOL = bit 3, CPHA = bit 2 on SPCR
#define SPI_CLOCK_MASK 0x03  // SPR1 = bit 1, SPR0 = bit 0 on SPCR
#define SPI_2XCLOCK_MASK 0x01  // SPI2X = bit 0 on SPSR

// define SPI_AVR_EIMSK for AVR boards with external interrupt pins
#if defined(EIMSK)
  #define SPI_AVR_EIMSK  EIMSK
#elif defined(GICR)
  #define SPI_AVR_EIMSK  GICR
#elif defined(GIMSK)
  #define SPI_AVR_EIMSK  GIMSK
#endif

class SPISettings {
public:
  SPISettings(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
    if (__builtin_constant_p(clock)) {
      init_AlwaysInline(clock, bitOrder, dataMode);
    } else {
      init_MightInline(clock, bitOrder, dataMode);
    }
  }
  SPISettings() {
    init_AlwaysInline(4000000, MSBFIRST, SPI_MODE0);
  }
private:
  void init_MightInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode) {
    init_AlwaysInline(clock, bitOrder, dataMode);
  }
  void init_AlwaysInline(uint32_t clock, uint8_t bitOrder, uint8_t dataMode)
    __attribute__((__always_inline__)) {
    // Clock settings are defined as follows. Note that this shows SPI2X
    // inverted, so the bits form increasing numbers. Also note that
    // fosc/64 appears twice
    // SPR1 SPR0 ~SPI2X Freq
    //   0    0     0   fosc/2
    //   0    0     1   fosc/4
    //   0    1     0   fosc/8
    //   0    1     1   fosc/16
    //   1    0     0   fosc/32
    //   1    0     1   fosc/64
    //   1    1     0   fosc/64
    //   1    1     1   fosc/128

    // We find the fastest clock that is less than or equal to the
    // given clock rate. The clock divider that results in clock_setting
    // is 2 ^^ (clock_div + 1). If nothing is slow enough, we'll use the
    // slowest (128 == 2 ^^ 7, so clock_div = 6).
    uint8_t clockDiv;

    // When the clock is known at compile time, use this if-then-else
    // cascade, which the compiler knows how to completely optimize
    // away. When clock is not known, use a loop instead, which generates
    // shorter code.
    if (__builtin_constant_p(clock)) {
      if (clock >= F_CPU / 2) {
        clockDiv = 0;
      } else if (clock >= F_CPU / 4) {
        clockDiv = 1;
      } else if (clock >= F_CPU / 8) {
        clockDiv = 2;
      } else if (clock >= F_CPU / 16) {
        clockDiv = 3;
      } else if (clock >= F_CPU / 32) {
        clockDiv = 4;
      } else if (clock >= F_CPU / 64) {
        clockDiv = 5;
      } else {
        clockDiv = 6;
      }
    } else {
      uint32_t clockSetting = F_CPU / 2;
      clockDiv = 0;
      while (clockDiv < 6 && clock < clockSetting) {
        clockSetting /= 2;
        clockDiv++;
      }
    }

    // Compensate for the duplicate fosc/64
    if (clockDiv == 6)
    clockDiv = 7;

    // Invert the SPI2X bit
    clockDiv ^= 0x1;

    // Pack into the SPISettings class
    spcr = _BV(SPE) | _BV(MSTR) | ((bitOrder == LSBFIRST) ? _BV(DORD) : 0) |
      (dataMode & SPI_MODE_MASK) | ((clockDiv >> 1) & SPI_CLOCK_MASK);
    spsr = clockDiv & SPI_2XCLOCK_MASK;
  }
  uint8_t spcr;
```

```
  uint8_t spsr;
  friend class SPIClass;
};


class SPIClass {
public:
  // Initialize the SPI library
  static void begin();

  // If SPI is used from within an interrupt, this function registers
  // that interrupt with the SPI library, so beginTransaction() can
  // prevent conflicts.  The input interruptNumber is the number used
  // with attachInterrupt.  If SPI is used from a different interrupt
  // (eg, a timer), interruptNumber should be 255.
  static void usingInterrupt(uint8_t interruptNumber);
  // And this does the opposite.
  static void notUsingInterrupt(uint8_t interruptNumber);
  // Note: the usingInterrupt and notUsingInterrupt functions should
  // not to be called from ISR context or inside a transaction.
  // For details see:
  // https://github.com/arduino/Arduino/pull/2381
  // https://github.com/arduino/Arduino/pull/2449

  // Before using SPI.transfer() or asserting chip select pins,
  // this function is used to gain exclusive access to the SPI bus
  // and configure the correct settings.
  inline static void beginTransaction(SPISettings settings) {
    if (interruptMode > 0) {
      uint8_t sreg = SREG;
      noInterrupts();

      #ifdef SPI_AVR_EIMSK
      if (interruptMode == 1) {
        interruptSave = SPI_AVR_EIMSK;
        SPI_AVR_EIMSK &= ~interruptMask;
        SREG = sreg;
      } else
      #endif
      {
        interruptSave = sreg;
      }
    }

    #ifdef SPI_TRANSACTION_MISMATCH_LED
    if (inTransactionFlag) {
      pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
      digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
    }
    inTransactionFlag = 1;
    #endif

    SPCR = settings.spcr;
    SPSR = settings.spsr;
  }

  // Write to the SPI bus (MOSI pin) and also receive (MISO pin)
  inline static uint8_t transfer(uint8_t data) {
    SPDR = data;
    /*
     * The following NOP introduces a small delay that can prevent the wait
     * loop form iterating when running at the maximum speed. This gives
     * about 10% more speed, even if it seems counter-intuitive. At lower
     * speeds it is unnoticed.
     */
    asm volatile("nop");
    while (!(SPSR & _BV(SPIF))) ; // wait
    return SPDR;
  }
  inline static uint16_t transfer16(uint16_t data) {
    union { uint16_t val; struct { uint8_t lsb; uint8_t msb; }; } in, out;
    in.val = data;
    if (!(SPCR & _BV(DORD))) {
      SPDR = in.msb;
      asm volatile("nop"); // See transfer(uint8_t) function
      while (!(SPSR & _BV(SPIF))) ;
      out.msb = SPDR;
      SPDR = in.lsb;
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.lsb = SPDR;
    } else {
      SPDR = in.lsb;
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.lsb = SPDR;
      SPDR = in.msb;
      asm volatile("nop");
      while (!(SPSR & _BV(SPIF))) ;
      out.msb = SPDR;
    }
    return out.val;
  }
  inline static void transfer(void *buf, size_t count) {
    if (count == 0) return;
    uint8_t *p = (uint8_t *)buf;
    SPDR = *p;
    while (--count > 0) {
      uint8_t out = *(p + 1);
      while (!(SPSR & _BV(SPIF))) ;
      uint8_t in = SPDR;
      SPDR = out;
      *p++ = in;
    }
    while (!(SPSR & _BV(SPIF))) ;
    *p = SPDR;
  }
  // After performing a group of transfers and releasing the chip select
  // signal, this function allows others to access the SPI bus
  inline static void endTransaction(void) {
    #ifdef SPI_TRANSACTION_MISMATCH_LED
    if (!inTransactionFlag) {
      pinMode(SPI_TRANSACTION_MISMATCH_LED, OUTPUT);
      digitalWrite(SPI_TRANSACTION_MISMATCH_LED, HIGH);
    }
    inTransactionFlag = 0;
    #endif
```

```
    if (interruptMode > 0) {
      #ifdef SPI_AVR_EIMSK
      uint8_t sreg = SREG;
      #endif
      noInterrupts();
      #ifdef SPI_AVR_EIMSK
      if (interruptMode == 1) {
        SPI_AVR_EIMSK = interruptSave;
        SREG = sreg;
      } else
      #endif
      {
        SREG = interruptSave;
      }
    }
  }

  // Disable the SPI bus
  static void end();

  // This function is deprecated.  New applications should use
  // beginTransaction() to configure SPI settings.
  inline static void setBitOrder(uint8_t bitOrder) {
    if (bitOrder == LSBFIRST) SPCR |= _BV(DORD);
    else SPCR &= ~(_BV(DORD));
  }
  // This function is deprecated.  New applications should use
  // beginTransaction() to configure SPI settings.
  inline static void setDataMode(uint8_t dataMode) {
    SPCR = (SPCR & ~SPI_MODE_MASK) | dataMode;
  }
  // This function is deprecated.  New applications should use
  // beginTransaction() to configure SPI settings.
  inline static void setClockDivider(uint8_t clockDiv) {
    SPCR = (SPCR & ~SPI_CLOCK_MASK) | (clockDiv & SPI_CLOCK_MASK);
    SPSR = (SPSR & ~SPI_2XCLOCK_MASK) | ((clockDiv >> 2) & SPI_2XCLOCK_MASK);
  }
  // These undocumented functions should not be used.  SPI.transfer()
  // polls the hardware flag which is automatically cleared as the
  // AVR responds to SPI's interrupt
  inline static void attachInterrupt() { SPCR |= _BV(SPIE); }
  inline static void detachInterrupt() { SPCR &= ~_BV(SPIE); }

private:
  static uint8_t initialized;
  static uint8_t interruptMode; // 0=none, 1=mask, 2=global
  static uint8_t interruptMask; // which interrupts to mask
  static uint8_t interruptSave; // temp storage, to restore state
  #ifdef SPI_TRANSACTION_MISMATCH_LED
  static uint8_t inTransactionFlag;
  #endif
};

extern SPIClass SPI;

#endif
```

***** SPI.cpp

```
/*
 * Copyright (c) 2010 by Cristian Maglie <c.maglie@arduino.cc>
 * Copyright (c) 2014 by Paul Stoffregen <paul@pjrc.com> (Transaction API)
 * Copyright (c) 2014 by Matthijs Kooijman <matthijs@stdin.nl> (SPISettings AVR)
 * Copyright (c) 2014 by Andrew J. Kroll <xxxajk@gmail.com> (atomicity fixes)
 * SPI Master library for arduino.
 *
 * This file is free software; you can redistribute it and/or modify
 * it under the terms of either the GNU General Public License version 2
 * or the GNU Lesser General Public License version 2.1, both as
 * published by the Free Software Foundation.
 */

#include "SPI.h"

SPIClass SPI;

uint8_t SPIClass::initialized = 0;
uint8_t SPIClass::interruptMode = 0;
uint8_t SPIClass::interruptMask = 0;
uint8_t SPIClass::interruptSave = 0;
#ifdef SPI_TRANSACTION_MISMATCH_LED
uint8_t SPIClass::inTransactionFlag = 0;
#endif

void SPIClass::begin()
{
  uint8_t sreg = SREG;
  noInterrupts(); // Protect from a scheduler and prevent transactionBegin
  if (!initialized) {
    // Set SS to high so a connected chip will be "deselected" by default
    uint8_t port = digitalPinToPort(SS);
    uint8_t bit = digitalPinToBitMask(SS);
    volatile uint8_t *reg = portModeRegister(port);

    // if the SS pin is not already configured as an output
    // then set it high (to enable the internal pull-up resistor)
    if(!(*reg & bit)){
      digitalWrite(SS, HIGH);
    }

    // When the SS pin is set as OUTPUT, it can be used as
    // a general purpose output port (it doesn't influence
    // SPI operations).
    pinMode(SS, OUTPUT);

    // Warning: if the SS pin ever becomes a LOW INPUT then SPI
    // automatically switches to Slave, so the data direction of
    // the SS pin MUST be kept as OUTPUT.
    SPCR |= _BV(MSTR);
    SPCR |= _BV(SPE);

    // Set direction register for SCK and MOSI pin.
    // MISO pin automatically overrides to INPUT.
    // By doing this AFTER enabling SPI, we avoid accidentally
    // clocking in a single bit since the lines go directly
```

```
    // from "input" to SPI control.
    // http://code.google.com/p/arduino/issues/detail?id=888
    pinMode(SCK, OUTPUT);
    pinMode(MOSI, OUTPUT);
  }
  initialized++; // reference count
  SREG = sreg;
}

void SPIClass::end() {
  uint8_t sreg = SREG;
  noInterrupts(); // Protect from a scheduler and prevent transactionBegin
  // Decrease the reference counter
  if (initialized)
    initialized--;
  // If there are no more references disable SPI
  if (!initialized) {
    SPCR &= ~_BV(SPE);
    interruptMode = 0;
    #ifdef SPI_TRANSACTION_MISMATCH_LED
    inTransactionFlag = 0;
    #endif
  }
  SREG = sreg;
}

// mapping of interrupt numbers to bits within SPI_AVR_EIMSK
#if defined(__AVR_ATmega32U4__)
  #define SPI_INT0_MASK  (1<<INT0)
  #define SPI_INT1_MASK  (1<<INT1)
  #define SPI_INT2_MASK  (1<<INT2)
  #define SPI_INT3_MASK  (1<<INT3)
  #define SPI_INT4_MASK  (1<<INT6)
#elif defined(__AVR_AT90USB646__) || defined(__AVR_AT90USB1286__)
  #define SPI_INT0_MASK  (1<<INT0)
  #define SPI_INT1_MASK  (1<<INT1)
  #define SPI_INT2_MASK  (1<<INT2)
  #define SPI_INT3_MASK  (1<<INT3)
  #define SPI_INT4_MASK  (1<<INT4)
  #define SPI_INT5_MASK  (1<<INT5)
  #define SPI_INT6_MASK  (1<<INT6)
  #define SPI_INT7_MASK  (1<<INT7)
#elif defined(EICRA) && defined(EICRB) && defined(EIMSK)
  #define SPI_INT0_MASK  (1<<INT4)
  #define SPI_INT1_MASK  (1<<INT5)
  #define SPI_INT2_MASK  (1<<INT0)
  #define SPI_INT3_MASK  (1<<INT1)
  #define SPI_INT4_MASK  (1<<INT2)
  #define SPI_INT5_MASK  (1<<INT3)
  #define SPI_INT6_MASK  (1<<INT6)
  #define SPI_INT7_MASK  (1<<INT7)
#else
  #ifdef INT0
  #define SPI_INT0_MASK  (1<<INT0)
  #endif
  #ifdef INT1
  #define SPI_INT1_MASK  (1<<INT1)
  #endif
  #ifdef INT2
  #define SPI_INT2_MASK  (1<<INT2)
  #endif
#endif

void SPIClass::usingInterrupt(uint8_t interruptNumber)
{
  uint8_t mask = 0;
  uint8_t sreg = SREG;
  noInterrupts(); // Protect from a scheduler and prevent transactionBegin
  switch (interruptNumber) {
  #ifdef SPI_INT0_MASK
  case 0: mask = SPI_INT0_MASK; break;
  #endif
  #ifdef SPI_INT1_MASK
  case 1: mask = SPI_INT1_MASK; break;
  #endif
  #ifdef SPI_INT2_MASK
  case 2: mask = SPI_INT2_MASK; break;
  #endif
  #ifdef SPI_INT3_MASK
  case 3: mask = SPI_INT3_MASK; break;
  #endif
  #ifdef SPI_INT4_MASK
  case 4: mask = SPI_INT4_MASK; break;
  #endif
  #ifdef SPI_INT5_MASK
  case 5: mask = SPI_INT5_MASK; break;
  #endif
  #ifdef SPI_INT6_MASK
  case 6: mask = SPI_INT6_MASK; break;
  #endif
  #ifdef SPI_INT7_MASK
  case 7: mask = SPI_INT7_MASK; break;
  #endif
  default:
    interruptMode = 2;
    break;
  }
  interruptMask |= mask;
  if (!interruptMode)
    interruptMode = 1;
  SREG = sreg;
}

void SPIClass::notUsingInterrupt(uint8_t interruptNumber)
{
  // Once in mode 2 we can't go back to 0 without a proper reference count
  if (interruptMode == 2)
    return;
  uint8_t mask = 0;
  uint8_t sreg = SREG;
  noInterrupts(); // Protect from a scheduler and prevent transactionBegin
  switch (interruptNumber) {
  #ifdef SPI_INT0_MASK
  case 0: mask = SPI_INT0_MASK; break;
  #endif
  #ifdef SPI_INT1_MASK
  case 1: mask = SPI_INT1_MASK; break;
```

```
  #endif
  #ifdef SPI_INT2_MASK
  case 2: mask = SPI_INT2_MASK; break;
  #endif
  #ifdef SPI_INT3_MASK
  case 3: mask = SPI_INT3_MASK; break;
  #endif
  #ifdef SPI_INT4_MASK
  case 4: mask = SPI_INT4_MASK; break;
  #endif
  #ifdef SPI_INT5_MASK
  case 5: mask = SPI_INT5_MASK; break;
  #endif
  #ifdef SPI_INT6_MASK
  case 6: mask = SPI_INT6_MASK; break;
  #endif
  #ifdef SPI_INT7_MASK
  case 7: mask = SPI_INT7_MASK; break;
  #endif
  default:
    break;
    // this case can't be reached
  }
  interruptMask &= ~mask;
  if (!interruptMask)
    interruptMode = 0;
  SREG = sreg;
}
```

***** Adafruit_SPITFT.h

```
#ifndef _ADAFRUIT_SPITFT_H_
#define _ADAFRUIT_SPITFT_H_

#if !defined(__AVR_ATtiny85__) // Not for ATtiny, at all

#include "Adafruit_GFX.h"
#include <SPI.h>

// HARDWARE CONFIG -----------------------------------------------------------

#if defined(__AVR__)
typedef uint8_t ADAGFX_PORT_t;       ///< PORT values are 8-bit
#define USE_FAST_PINIO               ///< Use direct PORT register access
#elif defined(ARDUINO_STM32_FEATHER) // WICED
typedef class HardwareSPI SPIClass;        ///< SPI is a bit odd on WICED
typedef uint32_t ADAGFX_PORT_t;            ///< PORT values are 32-bit
#elif defined(__arm__)
#if defined(ARDUINO_ARCH_SAMD)
// Adafruit M0, M4
typedef uint32_t ADAGFX_PORT_t; ///< PORT values are 32-bit
#define USE_FAST_PINIO   ///< Use direct PORT register access
#define HAS_PORT_SET_CLR ///< PORTs have set & clear registers
#elif defined(CORE_TEENSY)
// PJRC Teensy 4.x
#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
typedef uint32_t ADAGFX_PORT_t; ///< PORT values are 32-bit
                                // PJRC Teensy 3.x
#else
typedef uint8_t ADAGFX_PORT_t; ///< PORT values are 8-bit
#endif
#define USE_FAST_PINIO   ///< Use direct PORT register access
#define HAS_PORT_SET_CLR ///< PORTs have set & clear registers
#else
// Arduino Due?
typedef uint32_t ADAGFX_PORT_t; ///< PORT values are 32-bit
// USE_FAST_PINIO not available here (yet)...Due has a totally different
// GPIO register set and will require some changes elsewhere (e.g. in
// constructors especially).
#endif
#else                                      // !ARM
// Probably ESP8266 or ESP32. USE_FAST_PINIO is not available here (yet)
// but don't worry about it too much...the digitalWrite() implementation
// on these platforms is reasonably efficient and already RAM-resident,
// only gotcha then is no parallel connection support for now.
typedef uint32_t ADAGFX_PORT_t; ///< PORT values are 32-bit
#endif                                     // end !ARM
typedef volatile ADAGFX_PORT_t *PORTreg_t; ///< PORT register type

#if defined(__AVR__)
#define DEFAULT_SPI_FREQ 8000000L ///< Hardware SPI default speed
#else
#define DEFAULT_SPI_FREQ 16000000L ///< Hardware SPI default speed
#endif

#if defined(ADAFRUIT_PYPORTAL) || defined(ADAFRUIT_PYPORTAL_M4_TITANO) ||      \
    defined(ADAFRUIT_PYBADGE_M4_EXPRESS) ||                                    \
    defined(ADAFRUIT_PYGAMER_M4_EXPRESS) ||                                    \
    defined(ADAFRUIT_MONSTER_M4SK_EXPRESS) || defined(NRF52_SERIES) ||         \
    defined(ADAFRUIT_CIRCUITPLAYGROUND_M0)
#define USE_SPI_DMA ///< Auto DMA
#else
                                           //#define USE_SPI_DMA ///< If set,
                                           // use DMA if available
#endif
// Another "oops" name -- this now also handles parallel DMA.
// If DMA is enabled, Arduino sketch MUST #include <Adafruit_ZeroDMA.h>
// Estimated RAM usage:
// 4 bytes/pixel on display major axis + 8 bytes/pixel on minor axis,
// e.g. 320x240 pixels = 320 * 4 + 240 * 8 = 3,200 bytes.

#if defined(USE_SPI_DMA) && (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
#include <Adafruit_ZeroDMA.h>
#endif

// This is kind of a kludge. Needed a way to disambiguate the software SPI
// and parallel constructors via their argument lists. Originally tried a
// bool as the first argument to the parallel constructor (specifying 8-bit
// vs 16-bit interface) but the compiler regards this as equivalent to an
// integer and thus still ambiguous. SO...the parallel constructor requires
// an enumerated type as the first argument: tft8 (for 8-bit parallel) or
// tft16 (for 16-bit)...even though 16-bit isn't fully implemented or tested
// and might never be, still needed that disambiguation from soft SPI.
/*! For first arg to parallel constructor */
enum tftBusWidth { tft8bitbus, tft16bitbus };
```

```
// CLASS DEFINITION ----------------------------------------------------------

/*!
  @brief  Adafruit_SPITFT is an intermediary class between Adafruit_GFX
          and various hardware-specific subclasses for different displays.
          It handles certain operations that are common to a range of
          displays (address window, area fills, etc.). Originally these were
          all color TFT displays interfaced via SPI, but it's since expanded
          to include color OLEDs and parallel-interfaced TFTs. THE NAME HAS
          BEEN KEPT TO AVOID BREAKING A LOT OF SUBCLASSES AND EXAMPLE CODE.
          Many of the class member functions similarly live on with names
          that don't necessarily accurately describe what they're doing,
          again to avoid breaking a lot of other code. If in doubt, read
          the comments.
*/
class Adafruit_SPITFT : public Adafruit_GFX {

public:
  // CONSTRUCTORS --------------------------------------------------------

  // Software SPI constructor: expects width & height (at default rotation
  // setting 0), 4 signal pins (cs, dc, mosi, sclk), 2 optional pins
  // (reset, miso). cs argument is required but can be -1 if unused --
  // rather than moving it to the optional arguments, it was done this way
  // to avoid breaking existing code (-1 option was a later addition).
  Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc, int8_t mosi,
                  int8_t sck, int8_t rst = -1, int8_t miso = -1);

  // Hardware SPI constructor using the default SPI port: expects width &
  // height (at default rotation setting 0), 2 signal pins (cs, dc),
  // optional reset pin. cs is required but can be -1 if unused -- rather
  // than moving it to the optional arguments, it was done this way to
  // avoid breaking existing code (-1 option was a later addition).
  Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc,
                  int8_t rst = -1);

#if !defined(ESP8266) // See notes in .cpp
  // Hardware SPI constructor using an arbitrary SPI peripheral: expects
  // width & height (rotation 0), SPIClass pointer, 2 signal pins (cs, dc)
  // and optional reset pin. cs is required but can be -1 if unused.
  Adafruit_SPITFT(uint16_t w, uint16_t h, SPIClass *spiClass, int8_t cs,
                  int8_t dc, int8_t rst = -1);
#endif // end !ESP8266

  // Parallel constructor: expects width & height (rotation 0), flag
  // indicating whether 16-bit (true) or 8-bit (false) interface, 3 signal
  // pins (d0, wr, dc), 3 optional pins (cs, rst, rd). 16-bit parallel
  // isn't even fully implemented but the 'wide' flag was added as a
  // required argument to avoid ambiguity with other constructors.
  Adafruit_SPITFT(uint16_t w, uint16_t h, tftBusWidth busWidth, int8_t d0,
                  int8_t wr, int8_t dc, int8_t cs = -1, int8_t rst = -1,
                  int8_t rd = -1);

  // DESTRUCTOR ----------------------------------------------------------

  ~Adafruit_SPITFT(){};

  // CLASS MEMBER FUNCTIONS ----------------------------------------------

  // These first two functions MUST be declared by subclasses:

  /*!
      @brief  Display-specific initialization function.
      @param  freq  SPI frequency, in hz (or 0 for default or unused).
  */
  virtual void begin(uint32_t freq) = 0;

  /*!
      @brief  Set up the specific display hardware's "address window"
              for subsequent pixel-pushing operations.
      @param  x  Leftmost pixel of area to be drawn (MUST be within
                 display bounds at current rotation setting).
      @param  y  Topmost pixel of area to be drawn (MUST be within
                 display bounds at current rotation setting).
      @param  w  Width of area to be drawn, in pixels (MUST be >0 and,
                 added to x, within display bounds at current rotation).
      @param  h  Height of area to be drawn, in pixels (MUST be >0 and,
                 added to x, within display bounds at current rotation).
  */
  virtual void setAddrWindow(uint16_t x, uint16_t y, uint16_t w,
                             uint16_t h) = 0;

  // Remaining functions do not need to be declared in subclasses
  // unless they wish to provide hardware-specific optimizations.
  // Brief comments here...documented more thoroughly in .cpp file.

  // Subclass' begin() function invokes this to initialize hardware.
  // freq=0 to use default SPI speed. spiMode must be one of the SPI_MODEn
  // values defined in SPI.h, which are NOT the same as 0 for SPI_MODE0,
  // 1 for SPI_MODE1, etc...use ONLY the SPI_MODEn defines! Only!
  // Name is outdated (interface may be parallel) but for compatibility:
  void initSPI(uint32_t freq = 0, uint8_t spiMode = SPI_MODE0);
  void setSPISpeed(uint32_t freq);
  // Chip select and/or hardware SPI transaction start as needed:
  void startWrite(void);
  // Chip deselect and/or hardware SPI transaction end as needed:
  void endWrite(void);
  void sendCommand(uint8_t commandByte, uint8_t *dataBytes,
                   uint8_t numDataBytes);
  void sendCommand(uint8_t commandByte, const uint8_t *dataBytes = NULL,
                   uint8_t numDataBytes = 0);
  void sendCommand16(uint16_t commandWord, const uint8_t *dataBytes = NULL,
                     uint8_t numDataBytes = 0);
  uint8_t readcommand8(uint8_t commandByte, uint8_t index = 0);
  uint16_t readcommand16(uint16_t addr);

  // These functions require a chip-select and/or SPI transaction
  // around them. Higher-level graphics primitives might start a
  // single transaction and then make multiple calls to these functions
  // (e.g. circle or text rendering might make repeated lines or rects)
  // before ending the transaction. It's more efficient than starting a
  // transaction every time.
  void writePixel(int16_t x, int16_t y, uint16_t color);
  void writePixels(uint16_t *colors, uint32_t len, bool block = true,
                   bool bigEndian = false);
  void writeColor(uint16_t color, uint32_t len);
  void writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
```

```
                    uint16_t color);
  void writeFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  void writeFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  // This is a new function, similar to writeFillRect() except that
  // all arguments MUST be onscreen, sorted and clipped. If higher-level
  // primitives can handle their own sorting/clipping, it avoids repeating
  // such operations in the low-level code, making it potentially faster.
  // CALLING THIS WITH UNCLIPPED OR NEGATIVE VALUES COULD BE DISASTROUS.
  inline void writeFillRectPreclipped(int16_t x, int16_t y, int16_t w,
                                      int16_t h, uint16_t color);
  // Another new function, companion to the new non-blocking
  // writePixels() variant.
  void dmaWait(void);
  // Used by writePixels() in some situations, but might have rare need in
  // user code, so it's public...
  bool dmaBusy(void) const; // true if DMA is used and busy, false otherwise
  void swapBytes(uint16_t *src, uint32_t len, uint16_t *dest = NULL);

  // These functions are similar to the 'write' functions above, but with
  // a chip-select and/or SPI transaction built-in. They're typically used
  // solo -- that is, as graphics primitives in themselves, not invoked by
  // higher-level primitives (which should use the functions above).
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);
  void drawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);
  void drawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);
  // A single-pixel push encapsulated in a transaction. I don't think
  // this is used anymore (BMP demos might've used it?) but is provided
  // for backward compatibility, consider it deprecated:
  void pushColor(uint16_t color);

  using Adafruit_GFX::drawRGBBitmap; // Check base class first
  void drawRGBBitmap(int16_t x, int16_t y, uint16_t *pcolors, int16_t w,
                     int16_t h);

  void invertDisplay(bool i);
  uint16_t color565(uint8_t r, uint8_t g, uint8_t b);

  // Despite parallel additions, function names kept for compatibility:
  void spiWrite(uint8_t b);           // Write single byte as DATA
  void writeCommand(uint8_t cmd);     // Write single byte as COMMAND
  uint8_t spiRead(void);              // Read single byte of data
  void write16(uint16_t w);           // Write 16-bit value as DATA
  void writeCommand16(uint16_t cmd); // Write 16-bit value as COMMAND
  uint16_t read16(void);              // Read single 16-bit value

  // Most of these low-level functions were formerly macros in
  // Adafruit_SPITFT_Macros.h. Some have been made into inline functions
  // to avoid macro mishaps. Despite the addition of code for a parallel
  // display interface, the names have been kept for backward
  // compatibility (some subclasses may be invoking these):
  void SPI_WRITE16(uint16_t w); // Not inline
  void SPI_WRITE32(uint32_t l); // Not inline
  // Old code had both a spiWrite16() function and SPI_WRITE16 macro
  // in addition to the SPI_WRITE32 macro. The latter two have been
  // made into functions here, and spiWrite16() removed (use SPI_WRITE16()
  // instead). It looks like most subclasses had gotten comfortable with
  // SPI_WRITE16 and SPI_WRITE32 anyway so those names were kept rather
  // than the less-obnoxious camelcase variants, oh well.

  // Placing these functions entirely in the class definition inlines
  // them implicitly them while allowing their use in other code:

  /*!
      @brief  Set the chip-select line HIGH. Does NOT check whether CS pin
              is set (>=0), that should be handled in calling function.
              Despite function name, this is used even if the display
              connection is parallel.
  */
  void SPI_CS_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *csPortSet = 1;
#else  // !KINETISK
    *csPortSet = csPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *csPort |= csPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_cs, HIGH);
#endif // end !USE_FAST_PINIO
  }

  /*!
      @brief  Set the chip-select line LOW. Does NOT check whether CS pin
              is set (>=0), that should be handled in calling function.
              Despite function name, this is used even if the display
              connection is parallel.
  */
  void SPI_CS_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *csPortClr = 1;
#else  // !KINETISK
    *csPortClr = csPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *csPort &= csPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_cs, LOW);
#endif // end !USE_FAST_PINIO
  }

  /*!
      @brief  Set the data/command line HIGH (data mode).
  */
  void SPI_DC_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *dcPortSet = 1;
#else  // !KINETISK
    *dcPortSet = dcPinMask;
```

```
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *dcPort |= dcPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_dc, HIGH);
#endif // end !USE_FAST_PINIO
  }

  /*!
      @brief  Set the data/command line LOW (command mode).
  */
  void SPI_DC_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
    *dcPortClr = 1;
#else  // !KINETISK
    *dcPortClr = dcPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
    *dcPort &= dcPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
    digitalWrite(_dc, LOW);
#endif // end !USE_FAST_PINIO
  }

protected:
  // A few more low-level member functions -- some may have previously
  // been macros. Shouldn't have a need to access these externally, so
  // they've been moved to the protected section. Additionally, they're
  // declared inline here and the code is in the .cpp file, since outside
  // code doesn't need to see these.
  inline void SPI_MOSI_HIGH(void);
  inline void SPI_MOSI_LOW(void);
  inline void SPI_SCK_HIGH(void);
  inline void SPI_SCK_LOW(void);
  inline bool SPI_MISO_READ(void);
  inline void SPI_BEGIN_TRANSACTION(void);
  inline void SPI_END_TRANSACTION(void);
  inline void TFT_WR_STROBE(void); // Parallel interface write strobe
  inline void TFT_RD_HIGH(void);   // Parallel interface read high
  inline void TFT_RD_LOW(void);    // Parallel interface read low

  // CLASS INSTANCE VARIABLES --------------------------------------------

  // Here be dragons! There's a big union of three structures here --
  // one each for hardware SPI, software (bitbang) SPI, and parallel
  // interfaces. This is to save some memory, since a display's connection
  // will be only one of these. The order of some things is a little weird
  // in an attempt to get values to align and pack better in RAM.

#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
  PORTreg_t csPortSet; ///< PORT register for chip select SET
  PORTreg_t csPortClr; ///< PORT register for chip select CLEAR
  PORTreg_t dcPortSet; ///< PORT register for data/command SET
  PORTreg_t dcPortClr; ///< PORT register for data/command CLEAR
#else                  // !HAS_PORT_SET_CLR
  PORTreg_t csPort;                ///< PORT register for chip select
  PORTreg_t dcPort;                ///< PORT register for data/command
#endif                 // end HAS_PORT_SET_CLR
#endif                 // end USE_FAST_PINIO
#if defined(__cplusplus) && (__cplusplus >= 201100)
  union {
#endif
    struct {          //   Values specific to HARDWARE SPI:
      SPIClass *_spi; ///< SPI class pointer
#if defined(SPI_HAS_TRANSACTION)
      SPISettings settings; ///< SPI transaction settings
#else
    uint32_t _freq; ///< SPI bitrate (if no SPI transactions)
#endif
      uint32_t _mode; ///< SPI data mode (transactions or no)
    } hwspi;          ///< Hardware SPI values
    struct {          //   Values specific to SOFTWARE SPI:
#if defined(USE_FAST_PINIO)
      PORTreg_t misoPort; ///< PORT (PIN) register for MISO
#if defined(HAS_PORT_SET_CLR)
      PORTreg_t mosiPortSet; ///< PORT register for MOSI SET
      PORTreg_t mosiPortClr; ///< PORT register for MOSI CLEAR
      PORTreg_t sckPortSet;  ///< PORT register for SCK SET
      PORTreg_t sckPortClr;  ///< PORT register for SCK CLEAR
#if !defined(KINETISK)
      ADAGFX_PORT_t mosiPinMask; ///< Bitmask for MOSI
      ADAGFX_PORT_t sckPinMask;  ///< Bitmask for SCK
#endif                           // end !KINETISK
#else                            // !HAS_PORT_SET_CLR
      PORTreg_t mosiPort;           ///< PORT register for MOSI
      PORTreg_t sckPort;            ///< PORT register for SCK
      ADAGFX_PORT_t mosiPinMaskSet; ///< Bitmask for MOSI SET (OR)
      ADAGFX_PORT_t mosiPinMaskClr; ///< Bitmask for MOSI CLEAR (AND)
      ADAGFX_PORT_t sckPinMaskSet;  ///< Bitmask for SCK SET (OR bitmask)
      ADAGFX_PORT_t sckPinMaskClr;  ///< Bitmask for SCK CLEAR (AND)
#endif                           // end HAS_PORT_SET_CLR
#if !defined(KINETISK)
      ADAGFX_PORT_t misoPinMask; ///< Bitmask for MISO
#endif                           // end !KINETISK
#endif                           // end USE_FAST_PINIO
      int8_t _mosi;              ///< MOSI pin #
      int8_t _miso;              ///< MISO pin #
      int8_t _sck;               ///< SCK pin #
    } swspi;                     ///< Software SPI values
    struct {                     //   Values specific to 8-bit parallel:
#if defined(USE_FAST_PINIO)

#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
      volatile uint32_t *writePort; ///< PORT register for DATA WRITE
      volatile uint32_t *readPort;  ///< PORT (PIN) register for DATA READ
#else
      volatile uint8_t *writePort;  ///< PORT register for DATA WRITE
      volatile uint8_t *readPort;   ///< PORT (PIN) register for DATA READ
#endif
#if defined(HAS_PORT_SET_CLR)
      // Port direction register pointers are always 8-bit regardless of
      // PORTreg_t -- even if 32-bit port, we modify a byte-aligned 8 bits.
```

```
#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
      volatile uint32_t *dirSet; ///< PORT byte data direction SET
      volatile uint32_t *dirClr; ///< PORT byte data direction CLEAR
#else
      volatile uint8_t *dirSet; ///< PORT byte data direction SET
      volatile uint8_t *dirClr; ///< PORT byte data direction CLEAR
#endif
      PORTreg_t wrPortSet; ///< PORT register for write strobe SET
      PORTreg_t wrPortClr; ///< PORT register for write strobe CLEAR
      PORTreg_t rdPortSet; ///< PORT register for read strobe SET
      PORTreg_t rdPortClr; ///< PORT register for read strobe CLEAR
#if !defined(KINETISK)
      ADAGFX_PORT_t wrPinMask; ///< Bitmask for write strobe
#endif                         // end !KINETISK
      ADAGFX_PORT_t rdPinMask; ///< Bitmask for read strobe
#else                          // !HAS_PORT_SET_CLR
      // Port direction register pointer is always 8-bit regardless of
      // PORTreg_t -- even if 32-bit port, we modify a byte-aligned 8 bits.
      volatile uint8_t *portDir;  ///< PORT direction register
      PORTreg_t wrPort;           ///< PORT register for write strobe
      PORTreg_t rdPort;           ///< PORT register for read strobe
      ADAGFX_PORT_t wrPinMaskSet; ///< Bitmask for write strobe SET (OR)
      ADAGFX_PORT_t wrPinMaskClr; ///< Bitmask for write strobe CLEAR (AND)
      ADAGFX_PORT_t rdPinMaskSet; ///< Bitmask for read strobe SET (OR)
      ADAGFX_PORT_t rdPinMaskClr; ///< Bitmask for read strobe CLEAR (AND)
#endif                         // end HAS_PORT_SET_CLR
#endif                         // end USE_FAST_PINIO
      int8_t _d0;              ///< Data pin 0 #
      int8_t _wr;              ///< Write strobe pin #
      int8_t _rd;              ///< Read strobe pin # (or -1)
      bool wide = 0;           ///< If true, is 16-bit interface
    } tft8;                    ///< Parallel interface settings
#if defined(__cplusplus) && (__cplusplus >= 201100)
  }; ///< Only one interface is active
#endif
#if defined(USE_SPI_DMA) &&                                                    \
    (defined(__SAMD51__) ||                                                    \
     defined(ARDUINO_SAMD_ZERO))     // Used by hardware SPI and tft8
  Adafruit_ZeroDMA dma;              ///< DMA instance
  DmacDescriptor *dptr = NULL;       ///< 1st descriptor
  DmacDescriptor *descriptor = NULL; ///< Allocated descriptor list
  uint16_t *pixelBuf[2];             ///< Working buffers
  uint16_t maxFillLen;               ///< Max pixels per DMA xfer
  uint16_t lastFillColor = 0;        ///< Last color used w/fill
  uint32_t lastFillLen = 0;          ///< # of pixels w/last fill
  uint8_t onePixelBuf;               ///< For hi==lo fill
#endif
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if !defined(KINETISK)
  ADAGFX_PORT_t csPinMask; ///< Bitmask for chip select
  ADAGFX_PORT_t dcPinMask; ///< Bitmask for data/command
#endif                     // end !KINETISK
#else                      // !HAS_PORT_SET_CLR
  ADAGFX_PORT_t csPinMaskSet;     ///< Bitmask for chip select SET (OR)
  ADAGFX_PORT_t csPinMaskClr;     ///< Bitmask for chip select CLEAR (AND)
  ADAGFX_PORT_t dcPinMaskSet;     ///< Bitmask for data/command SET (OR)
  ADAGFX_PORT_t dcPinMaskClr;     ///< Bitmask for data/command CLEAR (AND)
#endif                     // end HAS_PORT_SET_CLR
#endif                     // end USE_FAST_PINIO
  uint8_t connection;      ///< TFT_HARD_SPI, TFT_SOFT_SPI, etc.
  int8_t _rst;             ///< Reset pin # (or -1)
  int8_t _cs;              ///< Chip select pin # (or -1)
  int8_t _dc;              ///< Data/command pin #

  int16_t _xstart = 0;          ///< Internal framebuffer X offset
  int16_t _ystart = 0;          ///< Internal framebuffer Y offset
  uint8_t invertOnCommand = 0;  ///< Command to enable invert mode
  uint8_t invertOffCommand = 0; ///< Command to disable invert mode

  uint32_t _freq = 0; ///< Dummy var to keep subclasses happy
};

#endif // end __AVR_ATtiny85__
#endif // end _ADAFRUIT_SPITFT_H_
```

***** Adafruit_SPITFT.cpp

```
#if !defined(__AVR_ATtiny85__) // Not for ATtiny, at all

#include "Adafruit_SPITFT.h"

#if defined(__AVR__)
#if defined(__AVR_XMEGA__) // only tested with __AVR_ATmega4809__
#define AVR_WRITESPI(x)                                                        \
  for (SPI0_DATA = (x); (!(SPI0_INTFLAGS & _BV(SPI_IF_bp)));)
#else
#define AVR_WRITESPI(x) for (SPDR = (x); (!(SPSR & _BV(SPIF)));)
#endif
#endif

#if defined(PORT_IOBUS)
// On SAMD21, redefine digitalPinToPort() to use the slightly-faster
// PORT_IOBUS rather than PORT (not needed on SAMD51).
#undef digitalPinToPort
#define digitalPinToPort(P) (&(PORT_IOBUS->Group[g_APinDescription[P].ulPort]))
#endif // end PORT_IOBUS

#if defined(USE_SPI_DMA) && (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
// #pragma message ("GFX DMA IS ENABLED. HIGHLY EXPERIMENTAL.")
#include "wiring_private.h" // pinPeripheral() function
#include <Adafruit_ZeroDMA.h>
#include <malloc.h>          // memalign() function
#define tcNum 2              // Timer/Counter for parallel write strobe PWM
#define wrPeripheral PIO_CCL // Use CCL to invert write strobe

// DMA transfer-in-progress indicator and callback
static volatile bool dma_busy = false;
static void dma_callback(Adafruit_ZeroDMA *dma) { dma_busy = false; }

#if defined(__SAMD51__)
// Timer/counter info by index #
static const struct {
  Tc *tc;   // -> Timer/Counter base address
  int gclk; // GCLK ID
  int evu;  // EVSYS user ID
```

```
} tcList[] = {{TC0, TC0_GCLK_ID, EVSYS_ID_USER_TC0_EVU},
              {TC1, TC1_GCLK_ID, EVSYS_ID_USER_TC1_EVU},
              {TC2, TC2_GCLK_ID, EVSYS_ID_USER_TC2_EVU},
              {TC3, TC3_GCLK_ID, EVSYS_ID_USER_TC3_EVU},
#if defined(TC4)
              {TC4, TC4_GCLK_ID, EVSYS_ID_USER_TC4_EVU},
#endif
#if defined(TC5)
              {TC5, TC5_GCLK_ID, EVSYS_ID_USER_TC5_EVU},
#endif
#if defined(TC6)
              {TC6, TC6_GCLK_ID, EVSYS_ID_USER_TC6_EVU},
#endif
#if defined(TC7)
              {TC7, TC7_GCLK_ID, EVSYS_ID_USER_TC7_EVU}
#endif
};
#define NUM_TIMERS (sizeof tcList / sizeof tcList[0]) ///< # timer/counters
#endif                                                // end __SAMD51__

#endif // end USE_SPI_DMA

// Possible values for Adafruit_SPITFT.connection:
#define TFT_HARD_SPI 0 ///< Display interface = hardware SPI
#define TFT_SOFT_SPI 1 ///< Display interface = software SPI
#define TFT_PARALLEL 2 ///< Display interface = 8- or 16-bit parallel

// CONSTRUCTORS ------------------------------------------------------------

/*!
    @brief   Adafruit_SPITFT constructor for software (bitbang) SPI.
    @param   w     Display width in pixels at default rotation setting (0).
    @param   h     Display height in pixels at default rotation setting (0).
    @param   cs    Arduino pin # for chip-select (-1 if unused, tie CS low).
    @param   dc    Arduino pin # for data/command select (required).
    @param   mosi  Arduino pin # for bitbang SPI MOSI signal (required).
    @param   sck   Arduino pin # for bitbang SPI SCK signal (required).
    @param   rst   Arduino pin # for display reset (optional, display reset
                   can be tied to MCU reset, default of -1 means unused).
    @param   miso  Arduino pin # for bitbang SPI MISO signal (optional,
                   -1 default, many displays don't support SPI read).
    @note    Output pins are not initialized; application typically will
             need to call subclass' begin() function, which in turn calls
             this library's initSPI() function to initialize pins.
*/
Adafruit_SPITFT::Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc,
                                 int8_t mosi, int8_t sck, int8_t rst,
                                 int8_t miso)
    : Adafruit_GFX(w, h), connection(TFT_SOFT_SPI), _rst(rst), _cs(cs),
      _dc(dc) {
  swspi._sck = sck;
  swspi._mosi = mosi;
  swspi._miso = miso;
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(CORE_TEENSY)
#if !defined(KINETISK)
  dcPinMask = digitalPinToBitMask(dc);
  swspi.sckPinMask = digitalPinToBitMask(sck);
  swspi.mosiPinMask = digitalPinToBitMask(mosi);
#endif
  dcPortSet = portSetRegister(dc);
  dcPortClr = portClearRegister(dc);
  swspi.sckPortSet = portSetRegister(sck);
  swspi.sckPortClr = portClearRegister(sck);
  swspi.mosiPortSet = portSetRegister(mosi);
  swspi.mosiPortClr = portClearRegister(mosi);
  if (cs >= 0) {
#if !defined(KINETISK)
    csPinMask = digitalPinToBitMask(cs);
#endif
    csPortSet = portSetRegister(cs);
    csPortClr = portClearRegister(cs);
  } else {
#if !defined(KINETISK)
    csPinMask = 0;
#endif
    csPortSet = dcPortSet;
    csPortClr = dcPortClr;
  }
  if (miso >= 0) {
    swspi.misoPort = portInputRegister(miso);
#if !defined(KINETISK)
    swspi.misoPinMask = digitalPinToBitMask(miso);
#endif
  } else {
    swspi.misoPort = portInputRegister(dc);
  }
#else  // !CORE_TEENSY
  dcPinMask = digitalPinToBitMask(dc);
  swspi.sckPinMask = digitalPinToBitMask(sck);
  swspi.mosiPinMask = digitalPinToBitMask(mosi);
  dcPortSet = &(PORT->Group[g_APinDescription[dc].ulPort].OUTSET.reg);
  dcPortClr = &(PORT->Group[g_APinDescription[dc].ulPort].OUTCLR.reg);
  swspi.sckPortSet = &(PORT->Group[g_APinDescription[sck].ulPort].OUTSET.reg);
  swspi.sckPortClr = &(PORT->Group[g_APinDescription[sck].ulPort].OUTCLR.reg);
  swspi.mosiPortSet = &(PORT->Group[g_APinDescription[mosi].ulPort].OUTSET.reg);
  swspi.mosiPortClr = &(PORT->Group[g_APinDescription[mosi].ulPort].OUTCLR.reg);
  if (cs >= 0) {
    csPinMask = digitalPinToBitMask(cs);
    csPortSet = &(PORT->Group[g_APinDescription[cs].ulPort].OUTSET.reg);
    csPortClr = &(PORT->Group[g_APinDescription[cs].ulPort].OUTCLR.reg);
  } else {
    // No chip-select line defined; might be permanently tied to GND.
    // Assign a valid GPIO register (though not used for CS), and an
    // empty pin bitmask...the nonsense bit-twiddling might be faster
    // than checking _cs and possibly branching.
    csPortSet = dcPortSet;
    csPortClr = dcPortClr;
    csPinMask = 0;
  }
  if (miso >= 0) {
    swspi.misoPinMask = digitalPinToBitMask(miso);
    swspi.misoPort = (PORTreg_t)portInputRegister(digitalPinToPort(miso));
  } else {
    swspi.misoPinMask = 0;
    swspi.misoPort = (PORTreg_t)portInputRegister(digitalPinToPort(dc));
```

```
  }
#endif // end !CORE_TEENSY
#else  // !HAS_PORT_SET_CLR
  dcPort = (PORTreg_t)portOutputRegister(digitalPinToPort(dc));
  dcPinMaskSet = digitalPinToBitMask(dc);
  swspi.sckPort = (PORTreg_t)portOutputRegister(digitalPinToPort(sck));
  swspi.sckPinMaskSet = digitalPinToBitMask(sck);
  swspi.mosiPort = (PORTreg_t)portOutputRegister(digitalPinToPort(mosi));
  swspi.mosiPinMaskSet = digitalPinToBitMask(mosi);
  if (cs >= 0) {
    csPort = (PORTreg_t)portOutputRegister(digitalPinToPort(cs));
    csPinMaskSet = digitalPinToBitMask(cs);
  } else {
    // No chip-select line defined; might be permanently tied to GND.
    // Assign a valid GPIO register (though not used for CS), and an
    // empty pin bitmask...the nonsense bit-twiddling might be faster
    // than checking _cs and possibly branching.
    csPort = dcPort;
    csPinMaskSet = 0;
  }
  if (miso >= 0) {
    swspi.misoPort = (PORTreg_t)portInputRegister(digitalPinToPort(miso));
    swspi.misoPinMask = digitalPinToBitMask(miso);
  } else {
    swspi.misoPort = (PORTreg_t)portInputRegister(digitalPinToPort(dc));
    swspi.misoPinMask = 0;
  }
  csPinMaskClr = ~csPinMaskSet;
  dcPinMaskClr = ~dcPinMaskSet;
  swspi.sckPinMaskClr = ~swspi.sckPinMaskSet;
  swspi.mosiPinMaskClr = ~swspi.mosiPinMaskSet;
#endif // !end HAS_PORT_SET_CLR
#endif // end USE_FAST_PINIO
}

/*!
    @brief   Adafruit_SPITFT constructor for hardware SPI using the board's
             default SPI peripheral.
    @param   w     Display width in pixels at default rotation setting (0).
    @param   h     Display height in pixels at default rotation setting (0).
    @param   cs    Arduino pin # for chip-select (-1 if unused, tie CS low).
    @param   dc    Arduino pin # for data/command select (required).
    @param   rst   Arduino pin # for display reset (optional, display reset
                   can be tied to MCU reset, default of -1 means unused).
    @note    Output pins are not initialized; application typically will
             need to call subclass' begin() function, which in turn calls
             this library's initSPI() function to initialize pins.
*/
#if defined(ESP8266) // See notes below
Adafruit_SPITFT::Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc,
                                 int8_t rst)
    : Adafruit_GFX(w, h), connection(TFT_HARD_SPI), _rst(rst), _cs(cs),
      _dc(dc) {
  hwspi._spi = &SPI;
}
#else  // !ESP8266
Adafruit_SPITFT::Adafruit_SPITFT(uint16_t w, uint16_t h, int8_t cs, int8_t dc,
                                 int8_t rst)
    : Adafruit_SPITFT(w, h, &SPI, cs, dc, rst) {
  // This just invokes the hardware SPI constructor below,
  // passing the default SPI device (&SPI).
}
#endif // end !ESP8266

#if !defined(ESP8266)
// ESP8266 compiler freaks out at this constructor -- it can't disambiguate
// beteween the SPIClass pointer (argument #3) and a regular integer.
// Solution here it to just not offer this variant on the ESP8266. You can
// use the default hardware SPI peripheral, or you can use software SPI,
// but if there's any library out there that creates a 'virtual' SPIClass
// peripheral and drives it with software bitbanging, that's not supported.
/*!
    @brief   Adafruit_SPITFT constructor for hardware SPI using a specific
             SPI peripheral.
    @param   w         Display width in pixels at default rotation (0).
    @param   h         Display height in pixels at default rotation (0).
    @param   spiClass  Pointer to SPIClass type (e.g. &SPI or &SPI1).
    @param   cs        Arduino pin # for chip-select (-1 if unused, tie CS low).
    @param   dc        Arduino pin # for data/command select (required).
    @param   rst       Arduino pin # for display reset (optional, display reset
                       can be tied to MCU reset, default of -1 means unused).
    @note    Output pins are not initialized in constructor; application
             typically will need to call subclass' begin() function, which
             in turn calls this library's initSPI() function to initialize
             pins. EXCEPT...if you have built your own SERCOM SPI peripheral
             (calling the SPIClass constructor) rather than one of the
             built-in SPI devices (e.g. &SPI, &SPI1 and so forth), you will
             need to call the begin() function for your object as well as
             pinPeripheral() for the MOSI, MISO and SCK pins to configure
             GPIO manually. Do this BEFORE calling the display-specific
             begin or init function. Unfortunate but unavoidable.
*/
Adafruit_SPITFT::Adafruit_SPITFT(uint16_t w, uint16_t h, SPIClass *spiClass,
                                 int8_t cs, int8_t dc, int8_t rst)
    : Adafruit_GFX(w, h), connection(TFT_HARD_SPI), _rst(rst), _cs(cs),
      _dc(dc) {
  hwspi._spi = spiClass;
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(CORE_TEENSY)
#if !defined(KINETISK)
  dcPinMask = digitalPinToBitMask(dc);
#endif
  dcPortSet = portSetRegister(dc);
  dcPortClr = portClearRegister(dc);
  if (cs >= 0) {
#if !defined(KINETISK)
    csPinMask = digitalPinToBitMask(cs);
#endif
    csPortSet = portSetRegister(cs);
    csPortClr = portClearRegister(cs);
  } else { // see comments below
#if !defined(KINETISK)
    csPinMask = 0;
#endif
    csPortSet = dcPortSet;
    csPortClr = dcPortClr;
```

```
  }
#else  // !CORE_TEENSY
  dcPinMask = digitalPinToBitMask(dc);
  dcPortSet = &(PORT->Group[g_APinDescription[dc].ulPort].OUTSET.reg);
  dcPortClr = &(PORT->Group[g_APinDescription[dc].ulPort].OUTCLR.reg);
  if (cs >= 0) {
    csPinMask = digitalPinToBitMask(cs);
    csPortSet = &(PORT->Group[g_APinDescription[cs].ulPort].OUTSET.reg);
    csPortClr = &(PORT->Group[g_APinDescription[cs].ulPort].OUTCLR.reg);
  } else {
    // No chip-select line defined; might be permanently tied to GND.
    // Assign a valid GPIO register (though not used for CS), and an
    // empty pin bitmask...the nonsense bit-twiddling might be faster
    // than checking _cs and possibly branching.
    csPortSet = dcPortSet;
    csPortClr = dcPortClr;
    csPinMask = 0;
  }
#endif // end !CORE_TEENSY
#else  // !HAS_PORT_SET_CLR
  dcPort = (PORTreg_t)portOutputRegister(digitalPinToPort(dc));
  dcPinMaskSet = digitalPinToBitMask(dc);
  if (cs >= 0) {
    csPort = (PORTreg_t)portOutputRegister(digitalPinToPort(cs));
    csPinMaskSet = digitalPinToBitMask(cs);
  } else {
    // No chip-select line defined; might be permanently tied to GND.
    // Assign a valid GPIO register (though not used for CS), and an
    // empty pin bitmask...the nonsense bit-twiddling might be faster
    // than checking _cs and possibly branching.
    csPort = dcPort;
    csPinMaskSet = 0;
  }
  csPinMaskClr = ~csPinMaskSet;
  dcPinMaskClr = ~dcPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#endif // end USE_FAST_PINIO
}
#endif // end !ESP8266

/*!
    @brief   Adafruit_SPITFT constructor for parallel display connection.
    @param   w         Display width in pixels at default rotation (0).
    @param   h         Display height in pixels at default rotation (0).
    @param   busWidth  If tft16 (enumeration in header file), is a 16-bit
                       parallel connection, else 8-bit.
                       16-bit isn't fully implemented or tested yet so
                       applications should pass "tft8bitbus" for now...needed to
                       stick a required enum argument in there to
                       disambiguate this constructor from the soft-SPI case.
                       Argument is ignored on 8-bit architectures (no 'wide'
                       support there since PORTs are 8 bits anyway).
    @param   d0        Arduino pin # for data bit 0 (1+ are extrapolated).
                       The 8 (or 16) data bits MUST be contiguous and byte-
                       aligned (or word-aligned for wide interface) within
                       the same PORT register (might not correspond to
                       Arduino pin sequence).
    @param   wr        Arduino pin # for write strobe (required).
    @param   dc        Arduino pin # for data/command select (required).
    @param   cs        Arduino pin # for chip-select (optional, -1 if unused,
                       tie CS low).
    @param   rst       Arduino pin # for display reset (optional, display reset
                       can be tied to MCU reset, default of -1 means unused).
    @param   rd        Arduino pin # for read strobe (optional, -1 if unused).
    @note    Output pins are not initialized; application typically will need
             to call subclass' begin() function, which in turn calls this
             library's initSPI() function to initialize pins.
             Yes, the name is a misnomer...this library originally handled
             only SPI displays, parallel being a recent addition (but not
             wanting to break existing code).
*/
Adafruit_SPITFT::Adafruit_SPITFT(uint16_t w, uint16_t h, tftBusWidth busWidth,
                                 int8_t d0, int8_t wr, int8_t dc, int8_t cs,
                                 int8_t rst, int8_t rd)
    : Adafruit_GFX(w, h), connection(TFT_PARALLEL), _rst(rst), _cs(cs),
      _dc(dc) {
  tft8._d0 = d0;
  tft8._wr = wr;
  tft8._rd = rd;
  tft8.wide = (busWidth == tft16bitbus);
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(CORE_TEENSY)
  tft8.wrPortSet = portSetRegister(wr);
  tft8.wrPortClr = portClearRegister(wr);
#if !defined(KINETISK)
  dcPinMask = digitalPinToBitMask(dc);
#endif
  dcPortSet = portSetRegister(dc);
  dcPortClr = portClearRegister(dc);
  if (cs >= 0) {
#if !defined(KINETISK)
    csPinMask = digitalPinToBitMask(cs);
#endif
    csPortSet = portSetRegister(cs);
    csPortClr = portClearRegister(cs);
  } else { // see comments below
#if !defined(KINETISK)
    csPinMask = 0;
#endif
    csPortSet = dcPortSet;
    csPortClr = dcPortClr;
  }
  if (rd >= 0) { // if read-strobe pin specified...
#if defined(KINETISK)
    tft8.rdPinMask = 1;
#else // !KINETISK
    tft8.rdPinMask = digitalPinToBitMask(rd);
#endif
    tft8.rdPortSet = portSetRegister(rd);
    tft8.rdPortClr = portClearRegister(rd);
  } else {
    tft8.rdPinMask = 0;
    tft8.rdPortSet = dcPortSet;
    tft8.rdPortClr = dcPortClr;
  }
  // These are all uint8_t* pointers -- elsewhere they're recast
```

```
  // as necessary if a 'wide' 16-bit interface is in use.
  tft8.writePort = portOutputRegister(d0);
  tft8.readPort = portInputRegister(d0);
  tft8.dirSet = portModeRegister(d0);
  tft8.dirClr = portModeRegister(d0);
#else  // !CORE_TEENSY
  tft8.wrPinMask = digitalPinToBitMask(wr);
  tft8.wrPortSet = &(PORT->Group[g_APinDescription[wr].ulPort].OUTSET.reg);
  tft8.wrPortClr = &(PORT->Group[g_APinDescription[wr].ulPort].OUTCLR.reg);
  dcPinMask = digitalPinToBitMask(dc);
  dcPortSet = &(PORT->Group[g_APinDescription[dc].ulPort].OUTSET.reg);
  dcPortClr = &(PORT->Group[g_APinDescription[dc].ulPort].OUTCLR.reg);
  if (cs >= 0) {
    csPinMask = digitalPinToBitMask(cs);
    csPortSet = &(PORT->Group[g_APinDescription[cs].ulPort].OUTSET.reg);
    csPortClr = &(PORT->Group[g_APinDescription[cs].ulPort].OUTCLR.reg);
  } else {
    // No chip-select line defined; might be permanently tied to GND.
    // Assign a valid GPIO register (though not used for CS), and an
    // empty pin bitmask...the nonsense bit-twiddling might be faster
    // than checking _cs and possibly branching.
    csPortSet = dcPortSet;
    csPortClr = dcPortClr;
    csPinMask = 0;
  }
  if (rd >= 0) { // if read-strobe pin specified...
    tft8.rdPinMask = digitalPinToBitMask(rd);
    tft8.rdPortSet = &(PORT->Group[g_APinDescription[rd].ulPort].OUTSET.reg);
    tft8.rdPortClr = &(PORT->Group[g_APinDescription[rd].ulPort].OUTCLR.reg);
  } else {
    tft8.rdPinMask = 0;
    tft8.rdPortSet = dcPortSet;
    tft8.rdPortClr = dcPortClr;
  }
  // Get pointers to PORT write/read/dir bytes within 32-bit PORT
  uint8_t dBit = g_APinDescription[d0].ulPin; // d0 bit # in PORT
  PortGroup *p = (&(PORT->Group[g_APinDescription[d0].ulPort]));
  uint8_t offset = dBit / 8; // d[7:0] byte # within PORT
  if (tft8.wide)
    offset &= ~1; // d[15:8] byte # within PORT
  // These are all uint8_t* pointers -- elsewhere they're recast
  // as necessary if a 'wide' 16-bit interface is in use.
  tft8.writePort = (volatile uint8_t *)&(p->OUT.reg) + offset;
  tft8.readPort = (volatile uint8_t *)&(p->IN.reg) + offset;
  tft8.dirSet = (volatile uint8_t *)&(p->DIRSET.reg) + offset;
  tft8.dirClr = (volatile uint8_t *)&(p->DIRCLR.reg) + offset;
#endif // end !CORE_TEENSY
#else  // !HAS_PORT_SET_CLR
  tft8.wrPort = (PORTreg_t)portOutputRegister(digitalPinToPort(wr));
  tft8.wrPinMaskSet = digitalPinToBitMask(wr);
  dcPort = (PORTreg_t)portOutputRegister(digitalPinToPort(dc));
  dcPinMaskSet = digitalPinToBitMask(dc);
  if (cs >= 0) {
    csPort = (PORTreg_t)portOutputRegister(digitalPinToPort(cs));
    csPinMaskSet = digitalPinToBitMask(cs);
  } else {
    // No chip-select line defined; might be permanently tied to GND.
    // Assign a valid GPIO register (though not used for CS), and an
    // empty pin bitmask...the nonsense bit-twiddling might be faster
    // than checking _cs and possibly branching.
    csPort = dcPort;
    csPinMaskSet = 0;
  }
  if (rd >= 0) { // if read-strobe pin specified...
    tft8.rdPort = (PORTreg_t)portOutputRegister(digitalPinToPort(rd));
    tft8.rdPinMaskSet = digitalPinToBitMask(rd);
  } else {
    tft8.rdPort = dcPort;
    tft8.rdPinMaskSet = 0;
  }
  csPinMaskClr = ~csPinMaskSet;
  dcPinMaskClr = ~dcPinMaskSet;
  tft8.wrPinMaskClr = ~tft8.wrPinMaskSet;
  tft8.rdPinMaskClr = ~tft8.rdPinMaskSet;
  tft8.writePort = (PORTreg_t)portOutputRegister(digitalPinToPort(d0));
  tft8.readPort = (PORTreg_t)portInputRegister(digitalPinToPort(d0));
  tft8.portDir = (PORTreg_t)portModeRegister(digitalPinToPort(d0));
#endif // end !HAS_PORT_SET_CLR
#endif // end USE_FAST_PINIO
}

// end constructors -------

// CLASS MEMBER FUNCTIONS -----------------------------------------------------

// begin() and setAddrWindow() MUST be declared by any subclass.

/*!
    @brief  Configure microcontroller pins for TFT interfacing. Typically
            called by a subclass' begin() function.
    @param  freq     SPI frequency when using hardware SPI. If default (0)
                     is passed, will fall back on a device-specific value.
                     Value is ignored when using software SPI or parallel
                     connection.
    @param  spiMode  SPI mode when using hardware SPI. MUST be one of the
                     values SPI_MODE0, SPI_MODE1, SPI_MODE2 or SPI_MODE3
                     defined in SPI.h. Do NOT attempt to pass '0' for
                     SPI_MODE0 and so forth...the values are NOT the same!
                     Use ONLY the defines! (Pity it's not an enum.)
    @note   Another anachronistically-named function; this is called even
            when the display connection is parallel (not SPI). Also, this
            could probably be made private...quite a few class functions
            were generously put in the public section.
*/
void Adafruit_SPITFT::initSPI(uint32_t freq, uint8_t spiMode) {

  if (!freq)
    freq = DEFAULT_SPI_FREQ; // If no freq specified, use default

  // Init basic control pins common to all connection types
  if (_cs >= 0) {
    pinMode(_cs, OUTPUT);
    digitalWrite(_cs, HIGH); // Deselect
  }
  pinMode(_dc, OUTPUT);
  digitalWrite(_dc, HIGH); // Data mode
```

```
  if (connection == TFT_HARD_SPI) {

#if defined(SPI_HAS_TRANSACTION)
    hwspi.settings = SPISettings(freq, MSBFIRST, spiMode);
#else
    hwspi._freq = freq; // Save freq value for later
#endif
    hwspi._mode = spiMode; // Save spiMode value for later
    // Call hwspi._spi->begin() ONLY if this is among the 'established'
    // SPI interfaces in variant.h. For DIY roll-your-own SERCOM SPIs,
    // begin() and pinPeripheral() calls MUST be made in one's calling
    // code, BEFORE the screen-specific begin/init function is called.
    // Reason for this is that SPI::begin() makes its own calls to
    // pinPeripheral() based on g_APinDescription[n].ulPinType, which
    // on non-established SPI interface pins will always be PIO_DIGITAL
    // or similar, while we need PIO_SERCOM or PIO_SERCOM_ALT...it's
    // highly unique between devices and variants for each pin or
    // SERCOM so we can't make those calls ourselves here. And the SPI
    // device needs to be set up before calling this because it's
    // immediately followed with initialization commands. Blargh.
    if (
#if !defined(SPI_INTERFACES_COUNT)
        1
#else
#if SPI_INTERFACES_COUNT > 0
        (hwspi._spi == &SPI)
#endif
#if SPI_INTERFACES_COUNT > 1
        || (hwspi._spi == &SPI1)
#endif
#if SPI_INTERFACES_COUNT > 2
        || (hwspi._spi == &SPI2)
#endif
#if SPI_INTERFACES_COUNT > 3
        || (hwspi._spi == &SPI3)
#endif
#if SPI_INTERFACES_COUNT > 4
        || (hwspi._spi == &SPI4)
#endif
#if SPI_INTERFACES_COUNT > 5
        || (hwspi._spi == &SPI5)
#endif
#endif // end SPI_INTERFACES_COUNT
    ) {
      hwspi._spi->begin();
    }
  } else if (connection == TFT_SOFT_SPI) {

    pinMode(swspi._mosi, OUTPUT);
    digitalWrite(swspi._mosi, LOW);
    pinMode(swspi._sck, OUTPUT);
    digitalWrite(swspi._sck, LOW);
    if (swspi._miso >= 0) {
      pinMode(swspi._miso, INPUT);
    }

  } else { // TFT_PARALLEL
           // Initialize data pins.  We were only passed d0, so scan
           // the pin description list looking for the other pins.
           // They'll be on the same PORT, and within the next 7 (or 15) bits
           // (because we need to write to a contiguous PORT byte or word).
#if defined(__AVR__)
    // PORT registers are 8 bits wide, so just need a register match...
    for (uint8_t i = 0; i < NUM_DIGITAL_PINS; i++) {
      if ((PORTreg_t)portOutputRegister(digitalPinToPort(i)) ==
          tft8.writePort) {
        pinMode(i, OUTPUT);
        digitalWrite(i, LOW);
      }
    }
#elif defined(USE_FAST_PINIO)
#if defined(CORE_TEENSY)
    if (!tft8.wide) {
      *tft8.dirSet = 0xFF;    // Set port to output
      *tft8.writePort = 0x00; // Write all 0s
    } else {
      *(volatile uint16_t *)tft8.dirSet = 0xFFFF;
      *(volatile uint16_t *)tft8.writePort = 0x0000;
    }
#else  // !CORE_TEENSY
    uint8_t portNum = g_APinDescription[tft8._d0].ulPort, // d0 PORT #
        dBit = g_APinDescription[tft8._d0].ulPin,         // d0 bit in PORT
        lastBit = dBit + (tft8.wide ? 15 : 7);
    for (uint8_t i = 0; i < PINS_COUNT; i++) {
      if ((g_APinDescription[i].ulPort == portNum) &&
          (g_APinDescription[i].ulPin >= dBit) &&
          (g_APinDescription[i].ulPin <= (uint32_t)lastBit)) {
        pinMode(i, OUTPUT);
        digitalWrite(i, LOW);
      }
    }
#endif // end !CORE_TEENSY
#endif
    pinMode(tft8._wr, OUTPUT);
    digitalWrite(tft8._wr, HIGH);
    if (tft8._rd >= 0) {
      pinMode(tft8._rd, OUTPUT);
      digitalWrite(tft8._rd, HIGH);
    }
  }

  if (_rst >= 0) {
    // Toggle _rst low to reset
    pinMode(_rst, OUTPUT);
    digitalWrite(_rst, HIGH);
    delay(100);
    digitalWrite(_rst, LOW);
    delay(100);
    digitalWrite(_rst, HIGH);
    delay(200);
  }

#if defined(USE_SPI_DMA) && (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
  if (((connection == TFT_HARD_SPI) || (connection == TFT_PARALLEL)) &&
      (dma.allocate() == DMA_STATUS_OK)) { // Allocate channel
    // The DMA library needs to alloc at least one valid descriptor,
    // so we do that here. It's not used in the usual sense though,
```

```
    // just before a transfer we copy descriptor[0] to this address.
    if (dptr = dma.addDescriptor(NULL, NULL, 42, DMA_BEAT_SIZE_BYTE, false,
                                 false)) {
      // Alloc 2 scanlines worth of pixels on display's major axis,
      // whichever that is, rounding each up to 2-pixel boundary.
      int major = (WIDTH > HEIGHT) ? WIDTH : HEIGHT;
      major += (major & 1);   // -> next 2-pixel bound, if needed.
      maxFillLen = major * 2; // 2 scanlines
      // Note to future self: if you decide to make the pixel buffer
      // much larger, remember that DMA transfer descriptors can't
      // exceed 65,535 bytes (not 65,536), meaning 32,767 pixels max.
      // Not that we have that kind of RAM to throw around right now.
      if ((pixelBuf[0] = (uint16_t *)malloc(maxFillLen * sizeof(uint16_t)))) {
        // Alloc OK. Get pointer to start of second scanline.
        pixelBuf[1] = &pixelBuf[0][major];
        // Determine number of DMA descriptors needed to cover
        // entire screen when entire 2-line pixelBuf is used
        // (round up for fractional last descriptor).
        int numDescriptors = (WIDTH * HEIGHT + (maxFillLen - 1)) / maxFillLen;
        // DMA descriptors MUST be 128-bit (16 byte) aligned.
        // memalign() is considered obsolete but it's replacements
        // (aligned_alloc() or posix_memalign()) are not currently
        // available in the version of ARM GCC in use, but this
        // is, so here we are.
        if ((descriptor = (DmacDescriptor *)memalign(
                16, numDescriptors * sizeof(DmacDescriptor)))) {
          int dmac_id;
          volatile uint32_t *data_reg;

          if (connection == TFT_HARD_SPI) {
            // THIS IS AN AFFRONT TO NATURE, but I don't know
            // any "clean" way to get the sercom number from the
            // the SPIClass pointer (e.g. &SPI or &SPI1), which
            // is all we have to work with. SPIClass does contain
            // a SERCOM pointer but it is a PRIVATE member!
            // Doing an UNSPEAKABLY HORRIBLE THING here, directly
            // accessing the first 32-bit value in the SPIClass
            // structure, knowing that's (currently) where the
            // SERCOM pointer lives, but this ENTIRELY DEPENDS on
            // that structure not changing nor the compiler
            // rearranging things. Oh the humanity!

            if (*(SERCOM **)hwspi._spi == &sercom0) {
              dmac_id = SERCOM0_DMAC_ID_TX;
              data_reg = &SERCOM0->SPI.DATA.reg;
#if defined SERCOM1
            } else if (*(SERCOM **)hwspi._spi == &sercom1) {
              dmac_id = SERCOM1_DMAC_ID_TX;
              data_reg = &SERCOM1->SPI.DATA.reg;
#endif
#if defined SERCOM2
            } else if (*(SERCOM **)hwspi._spi == &sercom2) {
              dmac_id = SERCOM2_DMAC_ID_TX;
              data_reg = &SERCOM2->SPI.DATA.reg;
#endif
#if defined SERCOM3
            } else if (*(SERCOM **)hwspi._spi == &sercom3) {
              dmac_id = SERCOM3_DMAC_ID_TX;
              data_reg = &SERCOM3->SPI.DATA.reg;
#endif
#if defined SERCOM4
            } else if (*(SERCOM **)hwspi._spi == &sercom4) {
              dmac_id = SERCOM4_DMAC_ID_TX;
              data_reg = &SERCOM4->SPI.DATA.reg;
#endif
#if defined SERCOM5
            } else if (*(SERCOM **)hwspi._spi == &sercom5) {
              dmac_id = SERCOM5_DMAC_ID_TX;
              data_reg = &SERCOM5->SPI.DATA.reg;
#endif
#if defined SERCOM6
            } else if (*(SERCOM **)hwspi._spi == &sercom6) {
              dmac_id = SERCOM6_DMAC_ID_TX;
              data_reg = &SERCOM6->SPI.DATA.reg;
#endif
#if defined SERCOM7
            } else if (*(SERCOM **)hwspi._spi == &sercom7) {
              dmac_id = SERCOM7_DMAC_ID_TX;
              data_reg = &SERCOM7->SPI.DATA.reg;
#endif
            }
            dma.setPriority(DMA_PRIORITY_3);
            dma.setTrigger(dmac_id);
            dma.setAction(DMA_TRIGGER_ACTON_BEAT);

            // Initialize descriptor list.
            for (int d = 0; d < numDescriptors; d++) {
              // No need to set SRCADDR, DESCADDR or BTCNT --
              // those are done in the pixel-writing functions.
              descriptor[d].BTCTRL.bit.VALID = true;
              descriptor[d].BTCTRL.bit.EVOSEL = DMA_EVENT_OUTPUT_DISABLE;
              descriptor[d].BTCTRL.bit.BLOCKACT = DMA_BLOCK_ACTION_NOACT;
              descriptor[d].BTCTRL.bit.BEATSIZE = DMA_BEAT_SIZE_BYTE;
              descriptor[d].BTCTRL.bit.DSTINC = 0;
              descriptor[d].BTCTRL.bit.STEPSEL = DMA_STEPSEL_SRC;
              descriptor[d].BTCTRL.bit.STEPSIZE =
                  DMA_ADDRESS_INCREMENT_STEP_SIZE_1;
              descriptor[d].DSTADDR.reg = (uint32_t)data_reg;
            }

          } else { // Parallel connection

#if defined(__SAMD51__)
            int dmaChannel = dma.getChannel();
            // Enable event output, use EVOSEL output
            DMAC->Channel[dmaChannel].CHEVCTRL.bit.EVOE = 1;
            DMAC->Channel[dmaChannel].CHEVCTRL.bit.EVOMODE = 0;

            // CONFIGURE TIMER/COUNTER (for write strobe)

            Tc *timer = tcList[tcNum].tc; // -> Timer struct
            int id = tcList[tcNum].gclk;  // Timer GCLK ID
            GCLK_PCHCTRL_Type pchctrl;

            // Set up timer clock source from GCLK
            GCLK->PCHCTRL[id].bit.CHEN = 0; // Stop timer
            while (GCLK->PCHCTRL[id].bit.CHEN)
```

```
  ; // Wait for it
pchctrl.bit.GEN = GCLK_PCHCTRL_GEN_GCLK0_Val;
pchctrl.bit.CHEN = 1; // Enable
GCLK->PCHCTRL[id].reg = pchctrl.reg;
while (!GCLK->PCHCTRL[id].bit.CHEN)
  ; // Wait for it

// Disable timer/counter before configuring it
timer->COUNT8.CTRLA.bit.ENABLE = 0;
while (timer->COUNT8.SYNCBUSY.bit.STATUS)
  ;

timer->COUNT8.WAVE.bit.WAVEGEN = 2;    // NPWM
timer->COUNT8.CTRLA.bit.MODE = 1;      // 8-bit
timer->COUNT8.CTRLA.bit.PRESCALER = 0; // 1:1
while (timer->COUNT8.SYNCBUSY.bit.STATUS)
  ;

timer->COUNT8.CTRLBCLR.bit.DIR = 1; // Count UP
while (timer->COUNT8.SYNCBUSY.bit.CTRLB)
  ;
timer->COUNT8.CTRLBSET.bit.ONESHOT = 1; // One-shot
while (timer->COUNT8.SYNCBUSY.bit.CTRLB)
  ;
timer->COUNT8.PER.reg = 6; // PWM top
while (timer->COUNT8.SYNCBUSY.bit.PER)
  ;
timer->COUNT8.CC[0].reg = 2; // Compare
while (timer->COUNT8.SYNCBUSY.bit.CC0)
  ;
// Enable async input events,
// event action = restart.
timer->COUNT8.EVCTRL.bit.TCEI = 1;
timer->COUNT8.EVCTRL.bit.EVACT = 1;

// Enable timer
timer->COUNT8.CTRLA.reg |= TC_CTRLA_ENABLE;
while (timer->COUNT8.SYNCBUSY.bit.STATUS)
  ;

#if (wrPeripheral == PIO_CCL)
            // CONFIGURE CCL (inverts timer/counter output)

            MCLK->APBCMASK.bit.CCL_ = 1;         // Enable CCL clock
            CCL->CTRL.bit.ENABLE = 0;            // Disable to config
            CCL->CTRL.bit.SWRST = 1;             // Reset CCL registers
            CCL->LUTCTRL[tcNum].bit.ENABLE = 0;  // Disable LUT
            CCL->LUTCTRL[tcNum].bit.FILTSEL = 0; // No filter
            CCL->LUTCTRL[tcNum].bit.INSEL0 = 6;  // TC input
            CCL->LUTCTRL[tcNum].bit.INSEL1 = 0;  // MASK
            CCL->LUTCTRL[tcNum].bit.INSEL2 = 0;  // MASK
            CCL->LUTCTRL[tcNum].bit.TRUTH = 1;   // Invert in 0
            CCL->LUTCTRL[tcNum].bit.ENABLE = 1;  // Enable LUT
            CCL->CTRL.bit.ENABLE = 1;            // Enable CCL
#endif

            // CONFIGURE EVENT SYSTEM

            // Set up event system clock source from GCLK...
            // Disable EVSYS, wait for disable
            GCLK->PCHCTRL[EVSYS_GCLK_ID_0].bit.CHEN = 0;
            while (GCLK->PCHCTRL[EVSYS_GCLK_ID_0].bit.CHEN)
              ;
            pchctrl.bit.GEN = GCLK_PCHCTRL_GEN_GCLK0_Val;
            pchctrl.bit.CHEN = 1; // Re-enable
            GCLK->PCHCTRL[EVSYS_GCLK_ID_0].reg = pchctrl.reg;
            // Wait for it, then enable EVSYS clock
            while (!GCLK->PCHCTRL[EVSYS_GCLK_ID_0].bit.CHEN)
              ;
            MCLK->APBBMASK.bit.EVSYS_ = 1;

            // Connect Timer EVU to ch 0
            EVSYS->USER[tcList[tcNum].evu].reg = 1;
            // Datasheet recommends single write operation;
            // reg instead of bit. Also datasheet: PATH bits
            // must be zero when using async!
            EVSYS_CHANNEL_Type ev;
            ev.reg = 0;
            ev.bit.PATH = 2;                    // Asynchronous
            ev.bit.EVGEN = 0x22 + dmaChannel; // DMA channel 0+
            EVSYS->Channel[0].CHANNEL.reg = ev.reg;

            // Initialize descriptor list.
            for (int d = 0; d < numDescriptors; d++) {
              // No need to set SRCADDR, DESCADDR or BTCNT --
              // those are done in the pixel-writing functions.
              descriptor[d].BTCTRL.bit.VALID = true;
              // Event strobe on beat xfer:
              descriptor[d].BTCTRL.bit.EVOSEL = 0x3;
              descriptor[d].BTCTRL.bit.BLOCKACT = DMA_BLOCK_ACTION_NOACT;
              descriptor[d].BTCTRL.bit.BEATSIZE =
                  tft8.wide ? DMA_BEAT_SIZE_HWORD : DMA_BEAT_SIZE_BYTE;
              descriptor[d].BTCTRL.bit.SRCINC = 1;
              descriptor[d].BTCTRL.bit.DSTINC = 0;
              descriptor[d].BTCTRL.bit.STEPSEL = DMA_STEPSEL_SRC;
              descriptor[d].BTCTRL.bit.STEPSIZE =
                  DMA_ADDRESS_INCREMENT_STEP_SIZE_1;
              descriptor[d].DSTADDR.reg = (uint32_t)tft8.writePort;
            }
#endif      // __SAMD51
          } // end parallel-specific DMA setup

          lastFillColor = 0x0000;
          lastFillLen = 0;
          dma.setCallback(dma_callback);
          return; // Success!
                  // else clean up any partial allocation...
        }         // end descriptor memalign()
        free(pixelBuf[0]);
        pixelBuf[0] = pixelBuf[1] = NULL;
      }         // end pixelBuf malloc()
                // Don't currently have a descriptor delete function in
                // ZeroDMA lib, but if we did, it would be called here.
    }           // end addDescriptor()
    dma.free(); // Deallocate DMA channel
  }
#endif // end USE_SPI_DMA
```

```
}

/*!
    @brief  Allow changing the SPI clock speed after initialization
    @param  freq Desired frequency of SPI clock, may not be the
    end frequency you get based on what the chip can do!
*/
void Adafruit_SPITFT::setSPISpeed(uint32_t freq) {
#if defined(SPI_HAS_TRANSACTION)
  hwspi.settings = SPISettings(freq, MSBFIRST, hwspi._mode);
#else
  hwspi._freq = freq; // Save freq value for later
#endif
}

/*!
    @brief  Call before issuing command(s) or data to display. Performs
            chip-select (if required) and starts an SPI transaction (if
            using hardware SPI and transactions are supported). Required
            for all display types; not an SPI-specific function.
*/
void Adafruit_SPITFT::startWrite(void) {
  SPI_BEGIN_TRANSACTION();
  if (_cs >= 0)
    SPI_CS_LOW();
}

/*!
    @brief  Call after issuing command(s) or data to display. Performs
            chip-deselect (if required) and ends an SPI transaction (if
            using hardware SPI and transactions are supported). Required
            for all display types; not an SPI-specific function.
*/
void Adafruit_SPITFT::endWrite(void) {
  if (_cs >= 0)
    SPI_CS_HIGH();
  SPI_END_TRANSACTION();
}

// ------------------------------------------------------------------------
// Lower-level graphics operations. These functions require a chip-select
// and/or SPI transaction around them (via startWrite(), endWrite() above).
// Higher-level graphics primitives might start a single transaction and
// then make multiple calls to these functions (e.g. circle or text
// rendering might make repeated lines or rects) before ending the
// transaction. It's more efficient than starting a transaction every time.

/*!
    @brief  Draw a single pixel to the display at requested coordinates.
            Not self-contained; should follow a startWrite() call.
    @param  x      Horizontal position (0 = left).
    @param  y      Vertical position   (0 = top).
    @param  color  16-bit pixel color in '565' RGB format.
*/
void Adafruit_SPITFT::writePixel(int16_t x, int16_t y, uint16_t color) {
  if ((x >= 0) && (x < _width) && (y >= 0) && (y < _height)) {
    setAddrWindow(x, y, 1, 1);
    SPI_WRITE16(color);
  }
}

/*!
    @brief  Swap bytes in an array of pixels; converts little-to-big or
            big-to-little endian. Used by writePixels() below in some
            situations, but may also be helpful for user code occasionally.
    @param  src   Source address of 16-bit pixels buffer.
    @param  len   Number of pixels to byte-swap.
    @param  dest  Optional destination address if different than src --
                  otherwise, if NULL (default) or same address is passed,
                  pixel buffer is overwritten in-place.
*/
void Adafruit_SPITFT::swapBytes(uint16_t *src, uint32_t len, uint16_t *dest) {
  if (!dest)
    dest = src; // NULL -> overwrite src buffer
  for (uint32_t i = 0; i < len; i++) {
    dest[i] = __builtin_bswap16(src[i]);
  }
}

/*!
    @brief  Issue a series of pixels from memory to the display. Not self-
            contained; should follow startWrite() and setAddrWindow() calls.
    @param  colors     Pointer to array of 16-bit pixel values in '565' RGB
                       format.
    @param  len        Number of elements in 'colors' array.
    @param  block      If true (default case if unspecified), function blocks
                       until DMA transfer is complete. This is simply IGNORED
                       if DMA is not enabled. If false, the function returns
                       immediately after the last DMA transfer is started,
                       and one should use the dmaWait() function before
                       doing ANY other display-related activities (or even
                       any SPI-related activities, if using an SPI display
                       that shares the bus with other devices).
    @param  bigEndian  If true, bitmap in memory is in big-endian order (most
                       significant byte first). By default this is false, as
                       most microcontrollers seem to be little-endian and
                       16-bit pixel values must be byte-swapped before
                       issuing to the display (which tend toward big-endian
                       when using SPI or 8-bit parallel). If an application
                       can optimize around this -- for example, a bitmap in a
                       uint16_t array having the byte values already ordered
                       big-endian, this can save time here, ESPECIALLY if
                       using this function's non-blocking DMA mode.
*/
void Adafruit_SPITFT::writePixels(uint16_t *colors, uint32_t len, bool block,
                                  bool bigEndian) {

  if (!len)
    return; // Avoid 0-byte transfers

  // avoid paramater-not-used complaints
  (void)block;
  (void)bigEndian;

#if defined(ESP32)
  if (connection == TFT_HARD_SPI) {
    if (!bigEndian) {
```

```
      hwspi._spi->writePixels(colors, len * 2); // Inbuilt endian-swap
    } else {
      hwspi._spi->writeBytes((uint8_t *)colors, len * 2); // Issue bytes direct
    }
    return;
  }
#elif defined(ARDUINO_NRF52_ADAFRUIT) &&                                      \
    defined(NRF52840_XXAA) // Adafruit nRF52 use SPIM3 DMA at 32Mhz
  if (!bigEndian) {
    swapBytes(colors, len); // convert little-to-big endian for display
  }
  hwspi._spi->transfer(colors, NULL, 2 * len); // NULL RX to avoid overwrite
  if (!bigEndian) {
    swapBytes(colors, len); // big-to-little endian to restore pixel buffer
  }

  return;
#elif defined(ARDUINO_ARCH_RP2040)
  spi_inst_t *pi_spi = hwspi._spi == &SPI ? spi0 : spi1;

  if (!bigEndian) {
    // switch to 16-bit writes
    hw_write_masked(&spi_get_hw(pi_spi)->cr0, 15 << SPI_SSPCR0_DSS_LSB,
                    SPI_SSPCR0_DSS_BITS);
    spi_write16_blocking(pi_spi, colors, len);
    // switch back to 8-bit
    hw_write_masked(&spi_get_hw(pi_spi)->cr0, 7 << SPI_SSPCR0_DSS_LSB,
                    SPI_SSPCR0_DSS_BITS);
  } else {
    spi_write_blocking(pi_spi, (uint8_t *)colors, len * 2);
  }
  return;
#elif defined(USE_SPI_DMA) &&                                                 \
    (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
  if ((connection == TFT_HARD_SPI) || (connection == TFT_PARALLEL)) {
    int maxSpan = maxFillLen / 2; // One scanline max
    uint8_t pixelBufIdx = 0;      // Active pixel buffer number
#if defined(__SAMD51__)
    if (connection == TFT_PARALLEL) {
      // Switch WR pin to PWM or CCL
      pinPeripheral(tft8._wr, wrPeripheral);
    }
#endif // end __SAMD51__
    if (!bigEndian) { // Normal little-endian situation...
      while (len) {
        int count = (len < maxSpan) ? len : maxSpan;

        // Because TFT and SAMD endianisms are different, must swap
        // bytes from the 'colors' array passed into a DMA working
        // buffer. This can take place while the prior DMA transfer
        // is in progress, hence the need for two pixelBufs.
        swapBytes(colors, count, pixelBuf[pixelBufIdx]);
        colors += count;

        // The transfers themselves are relatively small, so we don't
        // need a long descriptor list. We just alternate between the
        // first two, sharing pixelBufIdx for that purpose.
        descriptor[pixelBufIdx].SRCADDR.reg =
            (uint32_t)pixelBuf[pixelBufIdx] + count * 2;
        descriptor[pixelBufIdx].BTCTRL.bit.SRCINC = 1;
        descriptor[pixelBufIdx].BTCNT.reg = count * 2;
        descriptor[pixelBufIdx].DESCADDR.reg = 0;

        while (dma_busy)
          ; // Wait for prior line to finish

        // Move new descriptor into place...
        memcpy(dptr, &descriptor[pixelBufIdx], sizeof(DmacDescriptor));
        dma_busy = true;
        dma.startJob(); // Trigger SPI DMA transfer
        if (connection == TFT_PARALLEL)
          dma.trigger();
        pixelBufIdx = 1 - pixelBufIdx; // Swap DMA pixel buffers

        len -= count;
      }
    } else { // bigEndian == true
      // With big-endian pixel data, this can be handled as a single
      // DMA transfer using chained descriptors. Even full screen, this
      // needs only a relatively short descriptor list, each
      // transferring a max of 32,767 (not 32,768) pixels. The list
      // was allocated large enough to accommodate a full screen's
      // worth of data, so this won't run past the end of the list.
      int d, numDescriptors = (len + 32766) / 32767;
      for (d = 0; d < numDescriptors; d++) {
        int count = (len < 32767) ? len : 32767;
        descriptor[d].SRCADDR.reg = (uint32_t)colors + count * 2;
        descriptor[d].BTCTRL.bit.SRCINC = 1;
        descriptor[d].BTCNT.reg = count * 2;
        descriptor[d].DESCADDR.reg = (uint32_t)&descriptor[d + 1];
        len -= count;
        colors += count;
      }
      descriptor[d - 1].DESCADDR.reg = 0;

      while (dma_busy)
        ; // Wait for prior transfer (if any) to finish

      // Move first descriptor into place and start transfer...
      memcpy(dptr, &descriptor[0], sizeof(DmacDescriptor));
      dma_busy = true;
      dma.startJob(); // Trigger SPI DMA transfer
      if (connection == TFT_PARALLEL)
        dma.trigger();
    } // end bigEndian

    lastFillColor = 0x0000; // pixelBuf has been sullied
    lastFillLen = 0;
    if (block) {
      while (dma_busy)
        ; // Wait for last line to complete
#if defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO)
      if (connection == TFT_HARD_SPI) {
        // See SAMD51/21 note in writeColor()
        hwspi._spi->setDataMode(hwspi._mode);
      } else {
        pinPeripheral(tft8._wr, PIO_OUTPUT); // Switch WR back to GPIO
```

```
      }
#endif // end __SAMD51__ || ARDUINO_SAMD_ZERO
    }
    return;
  }
#endif // end USE_SPI_DMA

  // All other cases (bitbang SPI or non-DMA hard SPI or parallel),
  // use a loop with the normal 16-bit data write function:

  if (!bigEndian) {
    while (len--) {
      SPI_WRITE16(*colors++);
    }
  } else {
    // Well this is awkward. SPI_WRITE16() was designed for little-endian
    // hosts and big-endian displays as that's nearly always the typical
    // case. If the bigEndian flag was set, data is already in display's
    // order...so each pixel needs byte-swapping before being issued.
    // Rather than having a separate big-endian SPI_WRITE16 (adding more
    // bloat), it's preferred if calling function is smart and only uses
    // bigEndian where DMA is supported. But we gotta handle this...
    while (len--) {
      SPI_WRITE16(__builtin_bswap16(*colors++));
    }
  }
}

/*!
    @brief  Wait for the last DMA transfer in a prior non-blocking
            writePixels() call to complete. This does nothing if DMA
            is not enabled, and is not needed if blocking writePixels()
            was used (as is the default case).
*/
void Adafruit_SPITFT::dmaWait(void) {
#if defined(USE_SPI_DMA) && (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
  while (dma_busy)
    ;
#if defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO)
  if (connection == TFT_HARD_SPI) {
    // See SAMD51/21 note in writeColor()
    hwspi._spi->setDataMode(hwspi._mode);
  } else {
    pinPeripheral(tft8._wr, PIO_OUTPUT); // Switch WR back to GPIO
  }
#endif // end __SAMD51__ || ARDUINO_SAMD_ZERO
#endif
}

/*!
    @brief  Check if DMA transfer is active. Always returts false if DMA
            is not enabled.
    @return true if DMA is enabled and transmitting data, false otherwise.
*/
bool Adafruit_SPITFT::dmaBusy(void) const {
#if defined(USE_SPI_DMA) && (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
  return dma_busy;
#else
  return false;
#endif
}

/*!
    @brief  Issue a series of pixels, all the same color. Not self-
            contained; should follow startWrite() and setAddrWindow() calls.
    @param  color  16-bit pixel color in '565' RGB format.
    @param  len    Number of pixels to draw.
*/
void Adafruit_SPITFT::writeColor(uint16_t color, uint32_t len) {

  if (!len)
    return; // Avoid 0-byte transfers

  uint8_t hi = color >> 8, lo = color;

#if defined(ESP32) // ESP32 has a special SPI pixel-writing function...
  if (connection == TFT_HARD_SPI) {
#define SPI_MAX_PIXELS_AT_ONCE 32
#define TMPBUF_LONGWORDS (SPI_MAX_PIXELS_AT_ONCE + 1) / 2
#define TMPBUF_PIXELS (TMPBUF_LONGWORDS * 2)
    static uint32_t temp[TMPBUF_LONGWORDS];
    uint32_t c32 = color * 0x00010001;
    uint16_t bufLen = (len < TMPBUF_PIXELS) ? len : TMPBUF_PIXELS, xferLen,
             fillLen;
    // Fill temp buffer 32 bits at a time
    fillLen = (bufLen + 1) / 2; // Round up to next 32-bit boundary
    for (uint32_t t = 0; t < fillLen; t++) {
      temp[t] = c32;
    }
    // Issue pixels in blocks from temp buffer
    while (len) {                           // While pixels remain
      xferLen = (bufLen < len) ? bufLen : len; // How many this pass?
      writePixels((uint16_t *)temp, xferLen);
      len -= xferLen;
    }
    return;
  }
#elif defined(ARDUINO_NRF52_ADAFRUIT) &&                                       \
    defined(NRF52840_XXAA) // Adafruit nRF52840 use SPIM3 DMA at 32Mhz
  // at most 2 scan lines
  uint32_t const pixbufcount = min(len, ((uint32_t)2 * width()));
  uint16_t *pixbuf = (uint16_t *)rtos_malloc(2 * pixbufcount);

  // use SPI3 DMA if we could allocate buffer, else fall back to writing each
  // pixel loop below
  if (pixbuf) {
    uint16_t const swap_color = __builtin_bswap16(color);

    // fill buffer with color
    for (uint32_t i = 0; i < pixbufcount; i++) {
      pixbuf[i] = swap_color;
    }

    while (len) {
      uint32_t const count = min(len, pixbufcount);
      writePixels(pixbuf, count, true, true);
      len -= count;
```

```
    }

    rtos_free(pixbuf);
    return;
  }
#else                       // !ESP32
#if defined(USE_SPI_DMA) && (defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO))
  if (((connection == TFT_HARD_SPI) || (connection == TFT_PARALLEL)) &&
      (len >= 16)) { // Don't bother with DMA on short pixel runs
    int i, d, numDescriptors;
    if (hi == lo) { // If high & low bytes are same...
      onePixelBuf = color;
      // Can do this with a relatively short descriptor list,
      // each transferring a max of 32,767 (not 32,768) pixels.
      // This won't run off the end of the allocated descriptor list,
      // since we're using much larger chunks per descriptor here.
      numDescriptors = (len + 32766) / 32767;
      for (d = 0; d < numDescriptors; d++) {
        int count = (len < 32767) ? len : 32767;
        descriptor[d].SRCADDR.reg = (uint32_t)&onePixelBuf;
        descriptor[d].BTCTRL.bit.SRCINC = 0;
        descriptor[d].BTCNT.reg = count * 2;
        descriptor[d].DESCADDR.reg = (uint32_t)&descriptor[d + 1];
        len -= count;
      }
      descriptor[d - 1].DESCADDR.reg = 0;
    } else {
      // If high and low bytes are distinct, it's necessary to fill
      // a buffer with pixel data (swapping high and low bytes because
      // TFT and SAMD are different endianisms) and create a longer
      // descriptor list pointing repeatedly to this data. We can do
      // this slightly faster working 2 pixels (32 bits) at a time.
      uint32_t *pixelPtr = (uint32_t *)pixelBuf[0],
               twoPixels = __builtin_bswap16(color) * 0x00010001;
      // We can avoid some or all of the buffer-filling if the color
      // is the same as last time...
      if (color == lastFillColor) {
        // If length is longer than prior instance, fill only the
        // additional pixels in the buffer and update lastFillLen.
        if (len > lastFillLen) {
          int fillStart = lastFillLen / 2,
              fillEnd = (((len < maxFillLen) ? len : maxFillLen) + 1) / 2;
          for (i = fillStart; i < fillEnd; i++)
            pixelPtr[i] = twoPixels;
          lastFillLen = fillEnd * 2;
        } // else do nothing, don't set pixels or change lastFillLen
      } else {
        int fillEnd = (((len < maxFillLen) ? len : maxFillLen) + 1) / 2;
        for (i = 0; i < fillEnd; i++)
          pixelPtr[i] = twoPixels;
        lastFillLen = fillEnd * 2;
        lastFillColor = color;
      }

      numDescriptors = (len + maxFillLen - 1) / maxFillLen;
      for (d = 0; d < numDescriptors; d++) {
        int pixels = (len < maxFillLen) ? len : maxFillLen, bytes = pixels * 2;
        descriptor[d].SRCADDR.reg = (uint32_t)pixelPtr + bytes;
        descriptor[d].BTCTRL.bit.SRCINC = 1;
        descriptor[d].BTCNT.reg = bytes;
        descriptor[d].DESCADDR.reg = (uint32_t)&descriptor[d + 1];
        len -= pixels;
      }
      descriptor[d - 1].DESCADDR.reg = 0;
    }
    memcpy(dptr, &descriptor[0], sizeof(DmacDescriptor));
#if defined(__SAMD51__)
    if (connection == TFT_PARALLEL) {
      // Switch WR pin to PWM or CCL
      pinPeripheral(tft8._wr, wrPeripheral);
    }
#endif // end __SAMD51__

    dma_busy = true;
    dma.startJob();
    if (connection == TFT_PARALLEL)
      dma.trigger();
    while (dma_busy)
      ; // Wait for completion
      // Unfortunately blocking is necessary. An earlier version returned
      // immediately and checked dma_busy on startWrite() instead, but it
      // turns out to be MUCH slower on many graphics operations (as when
      // drawing lines, pixel-by-pixel), perhaps because it's a volatile
      // type and doesn't cache. Working on this.
#if defined(__SAMD51__) || defined(ARDUINO_SAMD_ZERO)
    if (connection == TFT_HARD_SPI) {
      // SAMD51: SPI DMA seems to leave the SPI peripheral in a freaky
      // state on completion. Workaround is to explicitly set it back...
      // (5/17/2019: apparently SAMD21 too, in certain cases, observed
      // with ST7789 display.)
      hwspi._spi->setDataMode(hwspi._mode);
    } else {
      pinPeripheral(tft8._wr, PIO_OUTPUT); // Switch WR back to GPIO
    }
#endif // end __SAMD51__
    return;
  }
#endif // end USE_SPI_DMA
#endif // end !ESP32

  // All other cases (non-DMA hard SPI, bitbang SPI, parallel)...

  if (connection == TFT_HARD_SPI) {
#if defined(ESP8266)
    do {
      uint32_t pixelsThisPass = len;
      if (pixelsThisPass > 50000)
        pixelsThisPass = 50000;
      len -= pixelsThisPass;
      yield(); // Periodic yield() on long fills
      while (pixelsThisPass--) {
        hwspi._spi->write(hi);
        hwspi._spi->write(lo);
      }
    } while (len);
#elif defined(ARDUINO_ARCH_RP2040)
    spi_inst_t *pi_spi = hwspi._spi == &SPI ? spi0 : spi1;
```

```
    color = __builtin_bswap16(color);

    while (len--)
      spi_write_blocking(pi_spi, (uint8_t *)&color, 2);
#else // !ESP8266 && !ARDUINO_ARCH_RP2040
    while (len--) {
#if defined(__AVR__)
      AVR_WRITESPI(hi);
      AVR_WRITESPI(lo);
#elif defined(ESP32)
      hwspi._spi->write(hi);
      hwspi._spi->write(lo);
#else
      hwspi._spi->transfer(hi);
      hwspi._spi->transfer(lo);
#endif
    }
#endif // end !ESP8266
  } else if (connection == TFT_SOFT_SPI) {
#if defined(ESP8266)
    do {
      uint32_t pixelsThisPass = len;
      if (pixelsThisPass > 20000)
        pixelsThisPass = 20000;
      len -= pixelsThisPass;
      yield(); // Periodic yield() on long fills
      while (pixelsThisPass--) {
        for (uint16_t bit = 0, x = color; bit < 16; bit++) {
          if (x & 0x8000)
            SPI_MOSI_HIGH();
          else
            SPI_MOSI_LOW();
          SPI_SCK_HIGH();
          SPI_SCK_LOW();
          x <<= 1;
        }
      }
    } while (len);
#else // !ESP8266
    while (len--) {
#if defined(__AVR__)
      for (uint8_t bit = 0, x = hi; bit < 8; bit++) {
        if (x & 0x80)
          SPI_MOSI_HIGH();
        else
          SPI_MOSI_LOW();
        SPI_SCK_HIGH();
        SPI_SCK_LOW();
        x <<= 1;
      }
      for (uint8_t bit = 0, x = lo; bit < 8; bit++) {
        if (x & 0x80)
          SPI_MOSI_HIGH();
        else
          SPI_MOSI_LOW();
        SPI_SCK_HIGH();
        SPI_SCK_LOW();
        x <<= 1;
      }
#else     // !__AVR__
      for (uint16_t bit = 0, x = color; bit < 16; bit++) {
        if (x & 0x8000)
          SPI_MOSI_HIGH();
        else
          SPI_MOSI_LOW();
        SPI_SCK_HIGH();
        x <<= 1;
        SPI_SCK_LOW();
      }
#endif    // end !__AVR__
    }
#endif    // end !ESP8266
  } else { // PARALLEL
    if (hi == lo) {
#if defined(__AVR__)
      len *= 2;
      *tft8.writePort = hi;
      while (len--) {
        TFT_WR_STROBE();
      }
#elif defined(USE_FAST_PINIO)
      if (!tft8.wide) {
        len *= 2;
        *tft8.writePort = hi;
      } else {
        *(volatile uint16_t *)tft8.writePort = color;
      }
      while (len--) {
        TFT_WR_STROBE();
      }
#endif
    } else {
      while (len--) {
#if defined(__AVR__)
        *tft8.writePort = hi;
        TFT_WR_STROBE();
        *tft8.writePort = lo;
#elif defined(USE_FAST_PINIO)
        if (!tft8.wide) {
          *tft8.writePort = hi;
          TFT_WR_STROBE();
          *tft8.writePort = lo;
        } else {
          *(volatile uint16_t *)tft8.writePort = color;
        }
#endif
        TFT_WR_STROBE();
      }
    }
  }
}

/*!
    @brief  Draw a filled rectangle to the display. Not self-contained;
            should follow startWrite(). Typically used by higher-level
            graphics primitives; user code shouldn't need to call this and
            is likely to use the self-contained fillRect() instead.
```

```
            writeFillRect() performs its own edge clipping and rejection;
            see writeFillRectPreclipped() for a more 'raw' implementation.
    @param  x      Horizontal position of first corner.
    @param  y      Vertical position of first corner.
    @param  w      Rectangle width in pixels (positive = right of first
                   corner, negative = left of first corner).
    @param  h      Rectangle height in pixels (positive = below first
                   corner, negative = above first corner).
    @param  color  16-bit fill color in '565' RGB format.
    @note   Written in this deep-nested way because C by definition will
            optimize for the 'if' case, not the 'else' -- avoids branches
            and rejects clipped rectangles at the least-work possibility.
*/
void Adafruit_SPITFT::writeFillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                                    uint16_t color) {
  if (w && h) {   // Nonzero width and height?
    if (w < 0) {  // If negative width...
      x += w + 1; //   Move X to left edge
      w = -w;     //   Use positive width
    }
    if (x < _width) { // Not off right
      if (h < 0) {    // If negative height...
        y += h + 1;   //   Move Y to top edge
        h = -h;       //   Use positive height
      }
      if (y < _height) { // Not off bottom
        int16_t x2 = x + w - 1;
        if (x2 >= 0) { // Not off left
          int16_t y2 = y + h - 1;
          if (y2 >= 0) { // Not off top
            // Rectangle partly or fully overlaps screen
            if (x < 0) {
              x = 0;
              w = x2 + 1;
            } // Clip left
            if (y < 0) {
              y = 0;
              h = y2 + 1;
            } // Clip top
            if (x2 >= _width) {
              w = _width - x;
            } // Clip right
            if (y2 >= _height) {
              h = _height - y;
            } // Clip bottom
            writeFillRectPreclipped(x, y, w, h, color);
          }
        }
      }
    }
  }
}

/*!
    @brief  Draw a horizontal line on the display. Performs edge clipping
            and rejection. Not self-contained; should follow startWrite().
            Typically used by higher-level graphics primitives; user code
            shouldn't need to call this and is likely to use the self-
            contained drawFastHLine() instead.
    @param  x      Horizontal position of first point.
    @param  y      Vertical position of first point.
    @param  w      Line width in pixels (positive = right of first point,
                   negative = point of first corner).
    @param  color  16-bit line color in '565' RGB format.
*/
void inline Adafruit_SPITFT::writeFastHLine(int16_t x, int16_t y, int16_t w,
                                            uint16_t color) {
  if ((y >= 0) && (y < _height) && w) { // Y on screen, nonzero width
    if (w < 0) {                        // If negative width...
      x += w + 1;                       //   Move X to left edge
      w = -w;                           //   Use positive width
    }
    if (x < _width) { // Not off right
      int16_t x2 = x + w - 1;
      if (x2 >= 0) { // Not off left
        // Line partly or fully overlaps screen
        if (x < 0) {
          x = 0;
          w = x2 + 1;
        } // Clip left
        if (x2 >= _width) {
          w = _width - x;
        } // Clip right
        writeFillRectPreclipped(x, y, w, 1, color);
      }
    }
  }
}

/*!
    @brief  Draw a vertical line on the display. Performs edge clipping and
            rejection. Not self-contained; should follow startWrite().
            Typically used by higher-level graphics primitives; user code
            shouldn't need to call this and is likely to use the self-
            contained drawFastVLine() instead.
    @param  x      Horizontal position of first point.
    @param  y      Vertical position of first point.
    @param  h      Line height in pixels (positive = below first point,
                   negative = above first point).
    @param  color  16-bit line color in '565' RGB format.
*/
void inline Adafruit_SPITFT::writeFastVLine(int16_t x, int16_t y, int16_t h,
                                            uint16_t color) {
  if ((x >= 0) && (x < _width) && h) { // X on screen, nonzero height
    if (h < 0) {                       // If negative height...
      y += h + 1;                      //   Move Y to top edge
      h = -h;                          //   Use positive height
    }
    if (y < _height) { // Not off bottom
      int16_t y2 = y + h - 1;
      if (y2 >= 0) { // Not off top
        // Line partly or fully overlaps screen
        if (y < 0) {
          y = 0;
          h = y2 + 1;
        } // Clip top
        if (y2 >= _height) {
```

```
          h = _height - y;
        } // Clip bottom
        writeFillRectPreclipped(x, y, 1, h, color);
      }
    }
  }
}

/*!
    @brief  A lower-level version of writeFillRect(). This version requires
            all inputs are in-bounds, that width and height are positive,
            and no part extends offscreen. NO EDGE CLIPPING OR REJECTION IS
            PERFORMED. If higher-level graphics primitives are written to
            handle their own clipping earlier in the drawing process, this
            can avoid unnecessary function calls and repeated clipping
            operations in the lower-level functions.
    @param  x      Horizontal position of first corner. MUST BE WITHIN
                   SCREEN BOUNDS.
    @param  y      Vertical position of first corner. MUST BE WITHIN SCREEN
                   BOUNDS.
    @param  w      Rectangle width in pixels. MUST BE POSITIVE AND NOT
                   EXTEND OFF SCREEN.
    @param  h      Rectangle height in pixels. MUST BE POSITIVE AND NOT
                   EXTEND OFF SCREEN.
    @param  color  16-bit fill color in '565' RGB format.
    @note   This is a new function, no graphics primitives besides rects
            and horizontal/vertical lines are written to best use this yet.
*/
inline void Adafruit_SPITFT::writeFillRectPreclipped(int16_t x, int16_t y,
                                                     int16_t w, int16_t h,
                                                     uint16_t color) {
  setAddrWindow(x, y, w, h);
  writeColor(color, (uint32_t)w * h);
}

// -----------------------------------------------------------------------
// Ever-so-slightly higher-level graphics operations. Similar to the 'write'
// functions above, but these contain their own chip-select and SPI
// transactions as needed (via startWrite(), endWrite()). They're typically
// used solo -- as graphics primitives in themselves, not invoked by higher-
// level primitives (which should use the functions above for better
// performance).

/*!
    @brief  Draw a single pixel to the display at requested coordinates.
            Self-contained and provides its own transaction as needed
            (see writePixel(x,y,color) for a lower-level variant).
            Edge clipping is performed here.
    @param  x      Horizontal position (0 = left).
    @param  y      Vertical position   (0 = top).
    @param  color  16-bit pixel color in '565' RGB format.
*/
void Adafruit_SPITFT::drawPixel(int16_t x, int16_t y, uint16_t color) {
  // Clip first...
  if ((x >= 0) && (x < _width) && (y >= 0) && (y < _height)) {
    // THEN set up transaction (if needed) and draw...
    startWrite();
    setAddrWindow(x, y, 1, 1);
    SPI_WRITE16(color);
    endWrite();
  }
}

/*!
    @brief  Draw a filled rectangle to the display. Self-contained and
            provides its own transaction as needed (see writeFillRect() or
            writeFillRectPreclipped() for lower-level variants). Edge
            clipping and rejection is performed here.
    @param  x      Horizontal position of first corner.
    @param  y      Vertical position of first corner.
    @param  w      Rectangle width in pixels (positive = right of first
                   corner, negative = left of first corner).
    @param  h      Rectangle height in pixels (positive = below first
                   corner, negative = above first corner).
    @param  color  16-bit fill color in '565' RGB format.
    @note   This repeats the writeFillRect() function almost in its entirety,
            with the addition of a transaction start/end. It's done this way
            (rather than starting the transaction and calling writeFillRect()
            to handle clipping and so forth) so that the transaction isn't
            performed at all if the rectangle is rejected. It's really not
            that much code.
*/
void Adafruit_SPITFT::fillRect(int16_t x, int16_t y, int16_t w, int16_t h,
                               uint16_t color) {
  if (w && h) {   // Nonzero width and height?
    if (w < 0) {  // If negative width...
      x += w + 1; //   Move X to left edge
      w = -w;     //   Use positive width
    }
    if (x < _width) { // Not off right
      if (h < 0) {    // If negative height...
        y += h + 1;   //   Move Y to top edge
        h = -h;       //   Use positive height
      }
      if (y < _height) { // Not off bottom
        int16_t x2 = x + w - 1;
        if (x2 >= 0) { // Not off left
          int16_t y2 = y + h - 1;
          if (y2 >= 0) { // Not off top
            // Rectangle partly or fully overlaps screen
            if (x < 0) {
              x = 0;
              w = x2 + 1;
            } // Clip left
            if (y < 0) {
              y = 0;
              h = y2 + 1;
            } // Clip top
            if (x2 >= _width) {
              w = _width - x;
            } // Clip right
            if (y2 >= _height) {
              h = _height - y;
            } // Clip bottom
            startWrite();
            writeFillRectPreclipped(x, y, w, h, color);
            endWrite();
```

```
        }
      }
    }
  }
}

/*!
    @brief  Draw a horizontal line on the display. Self-contained and
            provides its own transaction as needed (see writeFastHLine() for
            a lower-level variant). Edge clipping and rejection is performed
            here.
    @param  x      Horizontal position of first point.
    @param  y      Vertical position of first point.
    @param  w      Line width in pixels (positive = right of first point,
                   negative = point of first corner).
    @param  color  16-bit line color in '565' RGB format.
    @note   This repeats the writeFastHLine() function almost in its
            entirety, with the addition of a transaction start/end. It's
            done this way (rather than starting the transaction and calling
            writeFastHLine() to handle clipping and so forth) so that the
            transaction isn't performed at all if the line is rejected.
*/
void Adafruit_SPITFT::drawFastHLine(int16_t x, int16_t y, int16_t w,
                                    uint16_t color) {
  if ((y >= 0) && (y < _height) && w) { // Y on screen, nonzero width
    if (w < 0) {                        // If negative width...
      x += w + 1;                       //   Move X to left edge
      w = -w;                           //   Use positive width
    }
    if (x < _width) { // Not off right
      int16_t x2 = x + w - 1;
      if (x2 >= 0) { // Not off left
        // Line partly or fully overlaps screen
        if (x < 0) {
          x = 0;
          w = x2 + 1;
        } // Clip left
        if (x2 >= _width) {
          w = _width - x;
        } // Clip right
        startWrite();
        writeFillRectPreclipped(x, y, w, 1, color);
        endWrite();
      }
    }
  }
}

/*!
    @brief  Draw a vertical line on the display. Self-contained and provides
            its own transaction as needed (see writeFastHLine() for a lower-
            level variant). Edge clipping and rejection is performed here.
    @param  x      Horizontal position of first point.
    @param  y      Vertical position of first point.
    @param  h      Line height in pixels (positive = below first point,
                   negative = above first point).
    @param  color  16-bit line color in '565' RGB format.
    @note   This repeats the writeFastVLine() function almost in its
            entirety, with the addition of a transaction start/end. It's
            done this way (rather than starting the transaction and calling
            writeFastVLine() to handle clipping and so forth) so that the
            transaction isn't performed at all if the line is rejected.
*/
void Adafruit_SPITFT::drawFastVLine(int16_t x, int16_t y, int16_t h,
                                    uint16_t color) {
  if ((x >= 0) && (x < _width) && h) { // X on screen, nonzero height
    if (h < 0) {                       // If negative height...
      y += h + 1;                      //   Move Y to top edge
      h = -h;                          //   Use positive height
    }
    if (y < _height) { // Not off bottom
      int16_t y2 = y + h - 1;
      if (y2 >= 0) { // Not off top
        // Line partly or fully overlaps screen
        if (y < 0) {
          y = 0;
          h = y2 + 1;
        } // Clip top
        if (y2 >= _height) {
          h = _height - y;
        } // Clip bottom
        startWrite();
        writeFillRectPreclipped(x, y, 1, h, color);
        endWrite();
      }
    }
  }
}

/*!
    @brief  Essentially writePixel() with a transaction around it. I don't
            think this is in use by any of our code anymore (believe it was
            for some older BMP-reading examples), but is kept here in case
            any user code relies on it. Consider it DEPRECATED.
    @param  color  16-bit pixel color in '565' RGB format.
*/
void Adafruit_SPITFT::pushColor(uint16_t color) {
  startWrite();
  SPI_WRITE16(color);
  endWrite();
}

/*!
    @brief  Draw a 16-bit image (565 RGB) at the specified (x,y) position.
            For 16-bit display devices; no color reduction performed.
            Adapted from https://github.com/PaulStoffregen/ILI9341_t3
            by Marc MERLIN. See examples/pictureEmbed to use this.
            5/6/2017: function name and arguments have changed for
            compatibility with current GFX library and to avoid naming
            problems in prior implementation.  Formerly drawBitmap() with
            arguments in different order. Handles its own transaction and
            edge clipping/rejection.
    @param  x       Top left corner horizontal coordinate.
    @param  y       Top left corner vertical coordinate.
    @param  pcolors  Pointer to 16-bit array of pixel values.
    @param  w       Width of bitmap in pixels.
```

```
    @param  h       Height of bitmap in pixels.
*/
void Adafruit_SPITFT::drawRGBBitmap(int16_t x, int16_t y, uint16_t *pcolors,
                                    int16_t w, int16_t h) {

  int16_t x2, y2;                 // Lower-right coord
  if ((x >= _width) ||            // Off-edge right
      (y >= _height) ||           // " top
      ((x2 = (x + w - 1)) < 0) || // " left
      ((y2 = (y + h - 1)) < 0))
    return; // " bottom

  int16_t bx1 = 0, by1 = 0, // Clipped top-left within bitmap
      saveW = w;            // Save original bitmap width value
  if (x < 0) {              // Clip left
    w += x;
    bx1 = -x;
    x = 0;
  }
  if (y < 0) { // Clip top
    h += y;
    by1 = -y;
    y = 0;
  }
  if (x2 >= _width)
    w = _width - x; // Clip right
  if (y2 >= _height)
    h = _height - y; // Clip bottom

  pcolors += by1 * saveW + bx1; // Offset bitmap ptr to clipped top-left
  startWrite();
  setAddrWindow(x, y, w, h); // Clipped area
  while (h--) {              // For each (clipped) scanline...
    writePixels(pcolors, w); // Push one (clipped) row
    pcolors += saveW;        // Advance pointer by one full (unclipped) line
  }
  endWrite();
}

// ---------------------------------------------------------------------------
// Miscellaneous class member functions that don't draw anything.

/*!
    @brief  Invert the colors of the display (if supported by hardware).
            Self-contained, no transaction setup required.
    @param  i  true = inverted display, false = normal display.
*/
void Adafruit_SPITFT::invertDisplay(bool i) {
  startWrite();
  writeCommand(i ? invertOnCommand : invertOffCommand);
  endWrite();
}

/*!
    @brief   Given 8-bit red, green and blue values, return a 'packed'
             16-bit color value in '565' RGB format (5 bits red, 6 bits
             green, 5 bits blue). This is just a mathematical operation,
             no hardware is touched.
    @param   red    8-bit red brightnesss (0 = off, 255 = max).
    @param   green  8-bit green brightnesss (0 = off, 255 = max).
    @param   blue   8-bit blue brightnesss (0 = off, 255 = max).
    @return  'Packed' 16-bit color value (565 format).
*/
uint16_t Adafruit_SPITFT::color565(uint8_t red, uint8_t green, uint8_t blue) {
  return ((red & 0xF8) << 8) | ((green & 0xFC) << 3) | (blue >> 3);
}

/*!
@brief   Adafruit_SPITFT Send Command handles complete sending of commands and
data
@param   commandByte       The Command Byte
@param   dataBytes         A pointer to the Data bytes to send
@param   numDataBytes      The number of bytes we should send
*/
void Adafruit_SPITFT::sendCommand(uint8_t commandByte, uint8_t *dataBytes,
                                  uint8_t numDataBytes) {
  SPI_BEGIN_TRANSACTION();
  if (_cs >= 0)
    SPI_CS_LOW();

  SPI_DC_LOW();           // Command mode
  spiWrite(commandByte); // Send the command byte

  SPI_DC_HIGH();
  for (int i = 0; i < numDataBytes; i++) {
    if ((connection == TFT_PARALLEL) && tft8.wide) {
      SPI_WRITE16(*(uint16_t *)dataBytes);
      dataBytes += 2;
    } else {
      spiWrite(*dataBytes); // Send the data bytes
      dataBytes++;
    }
  }

  if (_cs >= 0)
    SPI_CS_HIGH();
  SPI_END_TRANSACTION();
}

/*!
 @brief   Adafruit_SPITFT Send Command handles complete sending of commands and
 data
 @param   commandByte       The Command Byte
 @param   dataBytes         A pointer to the Data bytes to send
 @param   numDataBytes      The number of bytes we should send
 */
void Adafruit_SPITFT::sendCommand(uint8_t commandByte, const uint8_t *dataBytes,
                                  uint8_t numDataBytes) {
  SPI_BEGIN_TRANSACTION();
  if (_cs >= 0)
    SPI_CS_LOW();

  SPI_DC_LOW();           // Command mode
  spiWrite(commandByte); // Send the command byte

  SPI_DC_HIGH();
  for (int i = 0; i < numDataBytes; i++) {
```

```
    if ((connection == TFT_PARALLEL) && tft8.wide) {
      SPI_WRITE16(*(uint16_t *)dataBytes);
      dataBytes += 2;
    } else {
      spiWrite(pgm_read_byte(dataBytes++));
    }
  }

  if (_cs >= 0)
    SPI_CS_HIGH();
  SPI_END_TRANSACTION();
}

/*!
 @brief  Adafruit_SPITFT sendCommand16 handles complete sending of
         commands and data for 16-bit parallel displays. Currently somewhat
         rigged for the NT35510, which has the odd behavior of wanting
         commands 16-bit, but subsequent data as 8-bit values, despite
         the 16-bit bus (high byte is always 0). Also seems to require
         issuing and incrementing address with each transfer.
 @param  commandWord   The command word (16 bits)
 @param  dataBytes     A pointer to the data bytes to send
 @param  numDataBytes  The number of bytes we should send
 */
void Adafruit_SPITFT::sendCommand16(uint16_t commandWord,
                                    const uint8_t *dataBytes,
                                    uint8_t numDataBytes) {
  SPI_BEGIN_TRANSACTION();
  if (_cs >= 0)
    SPI_CS_LOW();

  if (numDataBytes == 0) {
    SPI_DC_LOW();             // Command mode
    SPI_WRITE16(commandWord); // Send the command word
    SPI_DC_HIGH();            // Data mode
  }
  for (int i = 0; i < numDataBytes; i++) {
    SPI_DC_LOW();             // Command mode
    SPI_WRITE16(commandWord); // Send the command word
    SPI_DC_HIGH();            // Data mode
    commandWord++;
    SPI_WRITE16((uint16_t)pgm_read_byte(dataBytes++));
  }

  if (_cs >= 0)
    SPI_CS_HIGH();
  SPI_END_TRANSACTION();
}

/*!
 @brief   Read 8 bits of data from display configuration memory (not RAM).
 This is highly undocumented/supported and should be avoided,
 function is only included because some of the examples use it.
 @param   commandByte
 The command register to read data from.
 @param   index
 The byte index into the command to read from.
 @return  Unsigned 8-bit data read from display register.
 */
/**************************************************************************/
uint8_t Adafruit_SPITFT::readcommand8(uint8_t commandByte, uint8_t index) {
  uint8_t result;
  startWrite();
  SPI_DC_LOW(); // Command mode
  spiWrite(commandByte);
  SPI_DC_HIGH(); // Data mode
  do {
    result = spiRead();
  } while (index--); // Discard bytes up to index'th
  endWrite();
  return result;
}

/*!
 @brief   Read 16 bits of data from display register.
          For 16-bit parallel displays only.
 @param   addr  Command/register to access.
 @return  Unsigned 16-bit data.
 */
uint16_t Adafruit_SPITFT::readcommand16(uint16_t addr) {
#if defined(USE_FAST_PINIO) // NOT SUPPORTED without USE_FAST_PINIO
  uint16_t result = 0;
  if ((connection == TFT_PARALLEL) && tft8.wide) {
    startWrite();
    SPI_DC_LOW(); // Command mode
    SPI_WRITE16(addr);
    SPI_DC_HIGH(); // Data mode
    TFT_RD_LOW();  // Read line LOW
#if defined(HAS_PORT_SET_CLR)
    *(volatile uint16_t *)tft8.dirClr = 0xFFFF;   // Input state
    result = *(volatile uint16_t *)tft8.readPort; // 16-bit read
    *(volatile uint16_t *)tft8.dirSet = 0xFFFF;   // Output state
#else                                             // !HAS_PORT_SET_CLR
    *(volatile uint16_t *)tft8.portDir = 0x0000;    // Input state
    result = *(volatile uint16_t *)tft8.readPort;   // 16-bit read
    *(volatile uint16_t *)tft8.portDir = 0xFFFF;    // Output state
#endif                                            // end !HAS_PORT_SET_CLR
    TFT_RD_HIGH();                                // Read line HIGH
    endWrite();
  }
  return result;
#else
  (void)addr; // disable -Wunused-parameter warning
  return 0;
#endif // end !USE_FAST_PINIO
}

// ------------------------------------------------------------------------
// Lowest-level hardware-interfacing functions. Many of these are inline and
// compile to different things based on #defines -- typically just a few
// instructions. Others, not so much, those are not inlined.

/*!
    @brief  Start an SPI transaction if using the hardware SPI interface to
            the display. If using an earlier version of the Arduino platform
            (before the addition of SPI transactions), this instead attempts
            to set up the SPI clock and mode. No action is taken if the
```

```
            connection is not hardware SPI-based. This does NOT include a
            chip-select operation -- see startWrite() for a function that
            encapsulated both actions.
*/
inline void Adafruit_SPITFT::SPI_BEGIN_TRANSACTION(void) {
  if (connection == TFT_HARD_SPI) {
#if defined(SPI_HAS_TRANSACTION)
    hwspi._spi->beginTransaction(hwspi.settings);
#else // No transactions, configure SPI manually...
#if defined(__AVR__) || defined(TEENSYDUINO) || defined(ARDUINO_ARCH_STM32F1)
    hwspi._spi->setClockDivider(SPI_CLOCK_DIV2);
#elif defined(__arm__)
    hwspi._spi->setClockDivider(11);
#elif defined(ESP8266) || defined(ESP32)
    hwspi._spi->setFrequency(hwspi._freq);
#elif defined(RASPI) || defined(ARDUINO_ARCH_STM32F1)
    hwspi._spi->setClock(hwspi._freq);
#endif
    hwspi._spi->setBitOrder(MSBFIRST);
    hwspi._spi->setDataMode(hwspi._mode);
#endif // end !SPI_HAS_TRANSACTION
  }
}

/*!
    @brief  End an SPI transaction if using the hardware SPI interface to
            the display. No action is taken if the connection is not
            hardware SPI-based or if using an earlier version of the Arduino
            platform (before the addition of SPI transactions). This does
            NOT include a chip-deselect operation -- see endWrite() for a
            function that encapsulated both actions.
*/
inline void Adafruit_SPITFT::SPI_END_TRANSACTION(void) {
#if defined(SPI_HAS_TRANSACTION)
  if (connection == TFT_HARD_SPI) {
    hwspi._spi->endTransaction();
  }
#endif
}

/*!
    @brief  Issue a single 8-bit value to the display. Chip-select,
            transaction and data/command selection must have been
            previously set -- this ONLY issues the byte. This is another of
            those functions in the library with a now-not-accurate name
            that's being maintained for compatibility with outside code.
            This function is used even if display connection is parallel.
    @param  b  8-bit value to write.
*/
void Adafruit_SPITFT::spiWrite(uint8_t b) {
  if (connection == TFT_HARD_SPI) {
#if defined(__AVR__)
    AVR_WRITESPI(b);
#elif defined(ESP8266) || defined(ESP32)
    hwspi._spi->write(b);
#elif defined(ARDUINO_ARCH_RP2040)
    spi_inst_t *pi_spi = hwspi._spi == &SPI ? spi0 : spi1;
    spi_write_blocking(pi_spi, &b, 1);
#else
    hwspi._spi->transfer(b);
#endif
  } else if (connection == TFT_SOFT_SPI) {
    for (uint8_t bit = 0; bit < 8; bit++) {
      if (b & 0x80)
        SPI_MOSI_HIGH();
      else
        SPI_MOSI_LOW();
      SPI_SCK_HIGH();
      b <<= 1;
      SPI_SCK_LOW();
    }
  } else { // TFT_PARALLEL
#if defined(__AVR__)
    *tft8.writePort = b;
#elif defined(USE_FAST_PINIO)
    if (!tft8.wide)
      *tft8.writePort = b;
    else
      *(volatile uint16_t *)tft8.writePort = b;
#endif
    TFT_WR_STROBE();
  }
}

/*!
    @brief  Write a single command byte to the display. Chip-select and
            transaction must have been previously set -- this ONLY sets
            the device to COMMAND mode, issues the byte and then restores
            DATA mode. There is no corresponding explicit writeData()
            function -- just use spiWrite().
    @param  cmd  8-bit command to write.
*/
void Adafruit_SPITFT::writeCommand(uint8_t cmd) {
  SPI_DC_LOW();
  spiWrite(cmd);
  SPI_DC_HIGH();
}

/*!
    @brief   Read a single 8-bit value from the display. Chip-select and
             transaction must have been previously set -- this ONLY reads
             the byte. This is another of those functions in the library
             with a now-not-accurate name that's being maintained for
             compatibility with outside code. This function is used even if
             display connection is parallel.
    @return  Unsigned 8-bit value read (always zero if USE_FAST_PINIO is
             not supported by the MCU architecture).
*/
uint8_t Adafruit_SPITFT::spiRead(void) {
  uint8_t b = 0;
  uint16_t w = 0;
  if (connection == TFT_HARD_SPI) {
    return hwspi._spi->transfer((uint8_t)0);
  } else if (connection == TFT_SOFT_SPI) {
    if (swspi._miso >= 0) {
      for (uint8_t i = 0; i < 8; i++) {
        SPI_SCK_HIGH();
```

```
        b <<= 1;
        if (SPI_MISO_READ())
          b++;
        SPI_SCK_LOW();
      }
    }
    return b;
  } else { // TFT_PARALLEL
    if (tft8._rd >= 0) {
#if defined(USE_FAST_PINIO)
      TFT_RD_LOW(); // Read line LOW
#if defined(__AVR__)
      *tft8.portDir = 0x00; // Set port to input state
      w = *tft8.readPort;   // Read value from port
      *tft8.portDir = 0xFF; // Restore port to output
#else                       // !__AVR__
      if (!tft8.wide) {                           // 8-bit TFT connection
#if defined(HAS_PORT_SET_CLR)
        *tft8.dirClr = 0xFF;                      // Set port to input state
        w = *tft8.readPort;                       // Read value from port
        *tft8.dirSet = 0xFF;                      // Restore port to output
#else  // !HAS_PORT_SET_CLR
        *tft8.portDir = 0x00;                      // Set port to input state
        w = *tft8.readPort;                        // Read value from port
        *tft8.portDir = 0xFF;                      // Restore port to output
#endif // end HAS_PORT_SET_CLR
      } else {                                    // 16-bit TFT connection
#if defined(HAS_PORT_SET_CLR)
        *(volatile uint16_t *)tft8.dirClr = 0xFFFF; // Input state
        w = *(volatile uint16_t *)tft8.readPort;    // 16-bit read
        *(volatile uint16_t *)tft8.dirSet = 0xFFFF; // Output state
#else  // !HAS_PORT_SET_CLR
        *(volatile uint16_t *)tft8.portDir = 0x0000; // Input state
        w = *(volatile uint16_t *)tft8.readPort;     // 16-bit read
        *(volatile uint16_t *)tft8.portDir = 0xFFFF; // Output state
#endif // end !HAS_PORT_SET_CLR
      }
      TFT_RD_HIGH();                               // Read line HIGH
#endif // end !__AVR__
#else  // !USE_FAST_PINIO
      w = 0; // Parallel TFT is NOT SUPPORTED without USE_FAST_PINIO
#endif // end !USE_FAST_PINIO
    }
    return w;
  }
}

/*!
    @brief  Issue a single 16-bit value to the display. Chip-select,
            transaction and data/command selection must have been
            previously set -- this ONLY issues the word.
            Thus operates ONLY on 'wide' (16-bit) parallel displays!
    @param  w  16-bit value to write.
*/
void Adafruit_SPITFT::write16(uint16_t w) {
  if (connection == TFT_PARALLEL) {
#if defined(USE_FAST_PINIO)
    if (tft8.wide)
      *(volatile uint16_t *)tft8.writePort = w;
#else
    (void)w; // disable -Wunused-parameter warning
#endif
    TFT_WR_STROBE();
  }
}

/*!
    @brief  Write a single command word to the display. Chip-select and
            transaction must have been previously set -- this ONLY sets
            the device to COMMAND mode, issues the byte and then restores
            DATA mode. This operates ONLY on 'wide' (16-bit) parallel
            displays!
    @param  cmd  16-bit command to write.
*/
void Adafruit_SPITFT::writeCommand16(uint16_t cmd) {
  SPI_DC_LOW();
  write16(cmd);
  SPI_DC_HIGH();
}

/*!
    @brief   Read a single 16-bit value from the display. Chip-select and
             transaction must have been previously set -- this ONLY reads
             the byte. This operates ONLY on 'wide' (16-bit) parallel
             displays!
    @return  Unsigned 16-bit value read (always zero if USE_FAST_PINIO is
             not supported by the MCU architecture).
*/
uint16_t Adafruit_SPITFT::read16(void) {
  uint16_t w = 0;
  if (connection == TFT_PARALLEL) {
    if (tft8._rd >= 0) {
#if defined(USE_FAST_PINIO)
      TFT_RD_LOW();    // Read line LOW
      if (tft8.wide) { // 16-bit TFT connection
#if defined(HAS_PORT_SET_CLR)
        *(volatile uint16_t *)tft8.dirClr = 0xFFFF; // Input state
        w = *(volatile uint16_t *)tft8.readPort;    // 16-bit read
        *(volatile uint16_t *)tft8.dirSet = 0xFFFF; // Output state
#else                                               // !HAS_PORT_SET_CLR
        *(volatile uint16_t *)tft8.portDir = 0x0000; // Input state
        w = *(volatile uint16_t *)tft8.readPort;     // 16-bit read
        *(volatile uint16_t *)tft8.portDir = 0xFFFF; // Output state
#endif                                              // end !HAS_PORT_SET_CLR
      }
      TFT_RD_HIGH(); // Read line HIGH
#else                // !USE_FAST_PINIO
      w = 0; // Parallel TFT is NOT SUPPORTED without USE_FAST_PINIO
#endif               // end !USE_FAST_PINIO
    }
  }
  return w;
}

/*!
    @brief  Set the software (bitbang) SPI MOSI line HIGH.
*/
```

```
inline void Adafruit_SPITFT::SPI_MOSI_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
  *swspi.mosiPortSet = 1;
#else // !KINETISK
  *swspi.mosiPortSet = swspi.mosiPinMask;
#endif
#else  // !HAS_PORT_SET_CLR
  *swspi.mosiPort |= swspi.mosiPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(swspi._mosi, HIGH);
#if defined(ESP32)
  for (volatile uint8_t i = 0; i < 1; i++)
    ;
#endif // end ESP32
#endif // end !USE_FAST_PINIO
}

/*!
    @brief  Set the software (bitbang) SPI MOSI line LOW.
*/
inline void Adafruit_SPITFT::SPI_MOSI_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
  *swspi.mosiPortClr = 1;
#else // !KINETISK
  *swspi.mosiPortClr = swspi.mosiPinMask;
#endif
#else  // !HAS_PORT_SET_CLR
  *swspi.mosiPort &= swspi.mosiPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(swspi._mosi, LOW);
#if defined(ESP32)
  for (volatile uint8_t i = 0; i < 1; i++)
    ;
#endif // end ESP32
#endif // end !USE_FAST_PINIO
}

/*!
    @brief  Set the software (bitbang) SPI SCK line HIGH.
*/
inline void Adafruit_SPITFT::SPI_SCK_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
  *swspi.sckPortSet = 1;
#else                                              // !KINETISK
  *swspi.sckPortSet = swspi.sckPinMask;
#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
  for (volatile uint8_t i = 0; i < 1; i++)
    ;
#endif
#endif
#else  // !HAS_PORT_SET_CLR
  *swspi.sckPort |= swspi.sckPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(swspi._sck, HIGH);
#if defined(ESP32)
  for (volatile uint8_t i = 0; i < 1; i++)
    ;
#endif // end ESP32
#endif // end !USE_FAST_PINIO
}

/*!
    @brief  Set the software (bitbang) SPI SCK line LOW.
*/
inline void Adafruit_SPITFT::SPI_SCK_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
  *swspi.sckPortClr = 1;
#else                                              // !KINETISK
  *swspi.sckPortClr = swspi.sckPinMask;
#if defined(__IMXRT1052__) || defined(__IMXRT1062__) // Teensy 4.x
  for (volatile uint8_t i = 0; i < 1; i++)
    ;
#endif
#endif
#else  // !HAS_PORT_SET_CLR
  *swspi.sckPort &= swspi.sckPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(swspi._sck, LOW);
#if defined(ESP32)
  for (volatile uint8_t i = 0; i < 1; i++)
    ;
#endif // end ESP32
#endif // end !USE_FAST_PINIO
}

/*!
    @brief   Read the state of the software (bitbang) SPI MISO line.
    @return  true if HIGH, false if LOW.
*/
inline bool Adafruit_SPITFT::SPI_MISO_READ(void) {
#if defined(USE_FAST_PINIO)
#if defined(KINETISK)
  return *swspi.misoPort;
#else  // !KINETISK
  return *swspi.misoPort & swspi.misoPinMask;
#endif // end !KINETISK
#else  // !USE_FAST_PINIO
  return digitalRead(swspi._miso);
#endif // end !USE_FAST_PINIO
}

/*!
    @brief  Issue a single 16-bit value to the display. Chip-select,
            transaction and data/command selection must have been
            previously set -- this ONLY issues the word. Despite the name,
```

```
            this function is used even if display connection is parallel;
            name was maintaned for backward compatibility. Naming is also
            not consistent with the 8-bit version, spiWrite(). Sorry about
            that. Again, staying compatible with outside code.
    @param  w  16-bit value to write.
*/
void Adafruit_SPITFT::SPI_WRITE16(uint16_t w) {
  if (connection == TFT_HARD_SPI) {
#if defined(__AVR__)
    AVR_WRITESPI(w >> 8);
    AVR_WRITESPI(w);
#elif defined(ESP8266) || defined(ESP32)
    hwspi._spi->write16(w);
#elif defined(ARDUINO_ARCH_RP2040)
    spi_inst_t *pi_spi = hwspi._spi == &SPI ? spi0 : spi1;
    w = __builtin_bswap16(w);
    spi_write_blocking(pi_spi, (uint8_t *)&w, 2);
#else
    // MSB, LSB because TFTs are generally big-endian
    hwspi._spi->transfer(w >> 8);
    hwspi._spi->transfer(w);
#endif
  } else if (connection == TFT_SOFT_SPI) {
    for (uint8_t bit = 0; bit < 16; bit++) {
      if (w & 0x8000)
        SPI_MOSI_HIGH();
      else
        SPI_MOSI_LOW();
      SPI_SCK_HIGH();
      SPI_SCK_LOW();
      w <<= 1;
    }
  } else { // TFT_PARALLEL
#if defined(__AVR__)
    *tft8.writePort = w >> 8;
    TFT_WR_STROBE();
    *tft8.writePort = w;
#elif defined(USE_FAST_PINIO)
    if (!tft8.wide) {
      *tft8.writePort = w >> 8;
      TFT_WR_STROBE();
      *tft8.writePort = w;
    } else {
      *(volatile uint16_t *)tft8.writePort = w;
    }
#endif
    TFT_WR_STROBE();
  }
}

/*!
    @brief  Issue a single 32-bit value to the display. Chip-select,
            transaction and data/command selection must have been
            previously set -- this ONLY issues the longword. Despite the
            name, this function is used even if display connection is
            parallel; name was maintaned for backward compatibility. Naming
            is also not consistent with the 8-bit version, spiWrite().
            Sorry about that. Again, staying compatible with outside code.
    @param  l  32-bit value to write.
*/
void Adafruit_SPITFT::SPI_WRITE32(uint32_t l) {
  if (connection == TFT_HARD_SPI) {
#if defined(__AVR__)
    AVR_WRITESPI(l >> 24);
    AVR_WRITESPI(l >> 16);
    AVR_WRITESPI(l >> 8);
    AVR_WRITESPI(l);
#elif defined(ESP8266) || defined(ESP32)
    hwspi._spi->write32(l);
#elif defined(ARDUINO_ARCH_RP2040)
    spi_inst_t *pi_spi = hwspi._spi == &SPI ? spi0 : spi1;
    l = __builtin_bswap32(l);
    spi_write_blocking(pi_spi, (uint8_t *)&l, 4);
#else
    hwspi._spi->transfer(l >> 24);
    hwspi._spi->transfer(l >> 16);
    hwspi._spi->transfer(l >> 8);
    hwspi._spi->transfer(l);
#endif
  } else if (connection == TFT_SOFT_SPI) {
    for (uint8_t bit = 0; bit < 32; bit++) {
      if (l & 0x80000000)
        SPI_MOSI_HIGH();
      else
        SPI_MOSI_LOW();
      SPI_SCK_HIGH();
      SPI_SCK_LOW();
      l <<= 1;
    }
  } else { // TFT_PARALLEL
#if defined(__AVR__)
    *tft8.writePort = l >> 24;
    TFT_WR_STROBE();
    *tft8.writePort = l >> 16;
    TFT_WR_STROBE();
    *tft8.writePort = l >> 8;
    TFT_WR_STROBE();
    *tft8.writePort = l;
#elif defined(USE_FAST_PINIO)
    if (!tft8.wide) {
      *tft8.writePort = l >> 24;
      TFT_WR_STROBE();
      *tft8.writePort = l >> 16;
      TFT_WR_STROBE();
      *tft8.writePort = l >> 8;
      TFT_WR_STROBE();
      *tft8.writePort = l;
    } else {
      *(volatile uint16_t *)tft8.writePort = l >> 16;
      TFT_WR_STROBE();
      *(volatile uint16_t *)tft8.writePort = l;
    }
#endif
    TFT_WR_STROBE();
  }
}
```

```
/*!
    @brief  Set the WR line LOW, then HIGH. Used for parallel-connected
            interfaces when writing data.
*/
inline void Adafruit_SPITFT::TFT_WR_STROBE(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
#if defined(KINETISK)
  *tft8.wrPortClr = 1;
  *tft8.wrPortSet = 1;
#else  // !KINETISK
  *tft8.wrPortClr = tft8.wrPinMask;
  *tft8.wrPortSet = tft8.wrPinMask;
#endif // end !KINETISK
#else  // !HAS_PORT_SET_CLR
  *tft8.wrPort &= tft8.wrPinMaskClr;
  *tft8.wrPort |= tft8.wrPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(tft8._wr, LOW);
  digitalWrite(tft8._wr, HIGH);
#endif // end !USE_FAST_PINIO
}

/*!
    @brief  Set the RD line HIGH. Used for parallel-connected interfaces
            when reading data.
*/
inline void Adafruit_SPITFT::TFT_RD_HIGH(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
  *tft8.rdPortSet = tft8.rdPinMask;
#else  // !HAS_PORT_SET_CLR
  *tft8.rdPort |= tft8.rdPinMaskSet;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(tft8._rd, HIGH);
#endif // end !USE_FAST_PINIO
}

/*!
    @brief  Set the RD line LOW. Used for parallel-connected interfaces
            when reading data.
*/
inline void Adafruit_SPITFT::TFT_RD_LOW(void) {
#if defined(USE_FAST_PINIO)
#if defined(HAS_PORT_SET_CLR)
  *tft8.rdPortClr = tft8.rdPinMask;
#else  // !HAS_PORT_SET_CLR
  *tft8.rdPort &= tft8.rdPinMaskClr;
#endif // end !HAS_PORT_SET_CLR
#else  // !USE_FAST_PINIO
  digitalWrite(tft8._rd, LOW);
#endif // end !USE_FAST_PINIO
}

#endif // end __AVR_ATtiny85__
```

***** Adafruit_SPIDevice.h

```
#ifndef Adafruit_SPIDevice_h
#define Adafruit_SPIDevice_h

#include <Arduino.h>

#if !defined(SPI_INTERFACES_COUNT) ||                                          \
    (defined(SPI_INTERFACES_COUNT) && (SPI_INTERFACES_COUNT > 0))
// HW SPI available
#include <SPI.h>
#define BUSIO_HAS_HW_SPI
#else
// SW SPI ONLY
enum { SPI_MODE0, SPI_MODE1, SPI_MODE2, _SPI_MODE4 };
typedef uint8_t SPIClass;
#endif

// some modern SPI definitions don't have BitOrder enum
#if (defined(__AVR__) && !defined(ARDUINO_ARCH_MEGAAVR)) ||                    \
    defined(ESP8266) || defined(TEENSYDUINO) || defined(SPARK) ||              \
    defined(ARDUINO_ARCH_SPRESENSE) || defined(MEGATINYCORE) ||                \
    defined(DXCORE) || defined(ARDUINO_AVR_ATmega4809) ||                      \
    defined(ARDUINO_AVR_ATmega4808) || defined(ARDUINO_AVR_ATmega3209) ||      \
    defined(ARDUINO_AVR_ATmega3208) || defined(ARDUINO_AVR_ATmega1609) ||      \
    defined(ARDUINO_AVR_ATmega1608) || defined(ARDUINO_AVR_ATmega809) ||       \
    defined(ARDUINO_AVR_ATmega808) || defined(ARDUINO_ARCH_ARC32)

typedef enum _BitOrder {
  SPI_BITORDER_MSBFIRST = MSBFIRST,
  SPI_BITORDER_LSBFIRST = LSBFIRST,
} BusIOBitOrder;

#elif defined(ESP32) || defined(__ASR6501__) || defined(__ASR6502__)

// some modern SPI definitions don't have BitOrder enum and have different SPI
// mode defines
typedef enum _BitOrder {
  SPI_BITORDER_MSBFIRST = SPI_MSBFIRST,
  SPI_BITORDER_LSBFIRST = SPI_LSBFIRST,
} BusIOBitOrder;

#else
// Some platforms have a BitOrder enum but its named MSBFIRST/LSBFIRST
#define SPI_BITORDER_MSBFIRST MSBFIRST
#define SPI_BITORDER_LSBFIRST LSBFIRST
typedef BitOrder BusIOBitOrder;
#endif

#if defined(__IMXRT1062__) // Teensy 4.x
// *Warning* I disabled the usage of FAST_PINIO as the set/clear operations
// used in the cpp file are not atomic and can effect multiple IO pins
// and if an interrupt happens in between the time the code reads the register
//  and writes out the updated value, that changes one or more other IO pins
// on that same IO port, those change will be clobbered when the updated
// values are written back.  A fast version can be implemented that uses the
// ports set and clear registers which are atomic.
// typedef volatile uint32_t BusIO_PortReg;
```

```
// typedef uint32_t BusIO_PortMask;
//#define BUSIO_USE_FAST_PINIO

#elif defined(__AVR__) || defined(TEENSYDUINO)
typedef volatile uint8_t BusIO_PortReg;
typedef uint8_t BusIO_PortMask;
#define BUSIO_USE_FAST_PINIO

#elif defined(ESP8266) || defined(ESP32) || defined(__SAM3X8E__) ||            \
    defined(ARDUINO_ARCH_SAMD)
typedef volatile uint32_t BusIO_PortReg;
typedef uint32_t BusIO_PortMask;
#define BUSIO_USE_FAST_PINIO

#elif (defined(__arm__) || defined(ARDUINO_FEATHER52)) &&                      \
    !defined(ARDUINO_ARCH_MBED) && !defined(ARDUINO_ARCH_RP2040)
typedef volatile uint32_t BusIO_PortReg;
typedef uint32_t BusIO_PortMask;
#if !defined(__ASR6501__) && !defined(__ASR6502__)
#define BUSIO_USE_FAST_PINIO
#endif

#else
#undef BUSIO_USE_FAST_PINIO
#endif

/**! The class which defines how we will talk to this device over SPI **/
class Adafruit_SPIDevice {
public:
#ifdef BUSIO_HAS_HW_SPI
  Adafruit_SPIDevice(int8_t cspin, uint32_t freq = 1000000,
                     BusIOBitOrder dataOrder = SPI_BITORDER_MSBFIRST,
                     uint8_t dataMode = SPI_MODE0, SPIClass *theSPI = &SPI);
#else
  Adafruit_SPIDevice(int8_t cspin, uint32_t freq = 1000000,
                     BusIOBitOrder dataOrder = SPI_BITORDER_MSBFIRST,
                     uint8_t dataMode = SPI_MODE0, SPIClass *theSPI = nullptr);
#endif
  Adafruit_SPIDevice(int8_t cspin, int8_t sck, int8_t miso, int8_t mosi,
                     uint32_t freq = 1000000,
                     BusIOBitOrder dataOrder = SPI_BITORDER_MSBFIRST,
                     uint8_t dataMode = SPI_MODE0);
  ~Adafruit_SPIDevice();

  bool begin(void);
  bool read(uint8_t *buffer, size_t len, uint8_t sendvalue = 0xFF);
  bool write(const uint8_t *buffer, size_t len,
             const uint8_t *prefix_buffer = nullptr, size_t prefix_len = 0);
  bool write_then_read(const uint8_t *write_buffer, size_t write_len,
                       uint8_t *read_buffer, size_t read_len,
                       uint8_t sendvalue = 0xFF);
  bool write_and_read(uint8_t *buffer, size_t len);

  uint8_t transfer(uint8_t send);
  void transfer(uint8_t *buffer, size_t len);
  void beginTransaction(void);
  void endTransaction(void);
  void beginTransactionWithAssertingCS();
  void endTransactionWithDeassertingCS();

private:
#ifdef BUSIO_HAS_HW_SPI
  SPIClass *_spi = nullptr;
  SPISettings *_spiSetting = nullptr;
#else
  uint8_t *_spi = nullptr;
  uint8_t *_spiSetting = nullptr;
#endif
  uint32_t _freq;
  BusIOBitOrder _dataOrder;
  uint8_t _dataMode;
  void setChipSelect(int value);

  int8_t _cs, _sck, _mosi, _miso;
#ifdef BUSIO_USE_FAST_PINIO
  BusIO_PortReg *mosiPort, *clkPort, *misoPort, *csPort;
  BusIO_PortMask mosiPinMask, misoPinMask, clkPinMask, csPinMask;
#endif
  bool _begun;
};

#endif // Adafruit_SPIDevice_h
```

***** Adafruit_SPIDevice.cpp

```
#include "Adafruit_SPIDevice.h"

//#define DEBUG_SERIAL Serial

/*!
 *    @brief  Create an SPI device with the given CS pin and settings
 *    @param  cspin The arduino pin number to use for chip select
 *    @param  freq The SPI clock frequency to use, defaults to 1MHz
 *    @param  dataOrder The SPI data order to use for bits within each byte,
 * defaults to SPI_BITORDER_MSBFIRST
 *    @param  dataMode The SPI mode to use, defaults to SPI_MODE0
 *    @param  theSPI The SPI bus to use, defaults to &theSPI
 */
Adafruit_SPIDevice::Adafruit_SPIDevice(int8_t cspin, uint32_t freq,
                                       BusIOBitOrder dataOrder,
                                       uint8_t dataMode, SPIClass *theSPI) {
#ifdef BUSIO_HAS_HW_SPI
  _cs = cspin;
  _sck = _mosi = _miso = -1;
  _spi = theSPI;
  _begun = false;
  _spiSetting = new SPISettings(freq, dataOrder, dataMode);
  _freq = freq;
  _dataOrder = dataOrder;
  _dataMode = dataMode;
#else
  // unused, but needed to suppress compiler warns
  (void)cspin;
  (void)freq;
  (void)dataOrder;
  (void)dataMode;
```

```
  (void)theSPI;
#endif
}

/*!
 *    @brief  Create an SPI device with the given CS pin and settings
 *    @param  cspin The arduino pin number to use for chip select
 *    @param  sckpin The arduino pin number to use for SCK
 *    @param  misopin The arduino pin number to use for MISO, set to -1 if not
 * used
 *    @param  mosipin The arduino pin number to use for MOSI, set to -1 if not
 * used
 *    @param  freq The SPI clock frequency to use, defaults to 1MHz
 *    @param  dataOrder The SPI data order to use for bits within each byte,
 * defaults to SPI_BITORDER_MSBFIRST
 *    @param  dataMode The SPI mode to use, defaults to SPI_MODE0
 */
Adafruit_SPIDevice::Adafruit_SPIDevice(int8_t cspin, int8_t sckpin,
                                       int8_t misopin, int8_t mosipin,
                                       uint32_t freq, BusIOBitOrder dataOrder,
                                       uint8_t dataMode) {
  _cs = cspin;
  _sck = sckpin;
  _miso = misopin;
  _mosi = mosipin;

#ifdef BUSIO_USE_FAST_PINIO
  csPort = (BusIO_PortReg *)portOutputRegister(digitalPinToPort(cspin));
  csPinMask = digitalPinToBitMask(cspin);
  if (mosipin != -1) {
    mosiPort = (BusIO_PortReg *)portOutputRegister(digitalPinToPort(mosipin));
    mosiPinMask = digitalPinToBitMask(mosipin);
  }
  if (misopin != -1) {
    misoPort = (BusIO_PortReg *)portInputRegister(digitalPinToPort(misopin));
    misoPinMask = digitalPinToBitMask(misopin);
  }
  clkPort = (BusIO_PortReg *)portOutputRegister(digitalPinToPort(sckpin));
  clkPinMask = digitalPinToBitMask(sckpin);
#endif

  _freq = freq;
  _dataOrder = dataOrder;
  _dataMode = dataMode;
  _begun = false;
}

/*!
 *    @brief  Release memory allocated in constructors
 */
Adafruit_SPIDevice::~Adafruit_SPIDevice() {
  if (_spiSetting)
    delete _spiSetting;
}

/*!
 *    @brief  Initializes SPI bus and sets CS pin high
 *    @return Always returns true because there's no way to test success of SPI
 * init
 */
bool Adafruit_SPIDevice::begin(void) {
  if (_cs != -1) {
    pinMode(_cs, OUTPUT);
    digitalWrite(_cs, HIGH);
  }

  if (_spi) { // hardware SPI
#ifdef BUSIO_HAS_HW_SPI
    _spi->begin();
#endif
  } else {
    pinMode(_sck, OUTPUT);

    if ((_dataMode == SPI_MODE0) || (_dataMode == SPI_MODE1)) {
      // idle low on mode 0 and 1
      digitalWrite(_sck, LOW);
    } else {
      // idle high on mode 2 or 3
      digitalWrite(_sck, HIGH);
    }
    if (_mosi != -1) {
      pinMode(_mosi, OUTPUT);
      digitalWrite(_mosi, HIGH);
    }
    if (_miso != -1) {
      pinMode(_miso, INPUT);
    }
  }

  _begun = true;
  return true;
}

/*!
 *    @brief  Transfer (send/receive) a buffer over hard/soft SPI, without
 * transaction management
 *    @param  buffer The buffer to send and receive at the same time
 *    @param  len    The number of bytes to transfer
 */
void Adafruit_SPIDevice::transfer(uint8_t *buffer, size_t len) {
  //
  // HARDWARE SPI
  //
  if (_spi) {
#ifdef BUSIO_HAS_HW_SPI
#if defined(SPARK)
    _spi->transfer(buffer, buffer, len, nullptr);
#elif defined(STM32)
    for (size_t i = 0; i < len; i++) {
      _spi->transfer(buffer[i]);
    }
#else
    _spi->transfer(buffer, len);
#endif
    return;
#endif
  }
```

```
  //
  // SOFTWARE SPI
  //
  uint8_t startbit;
  if (_dataOrder == SPI_BITORDER_LSBFIRST) {
    startbit = 0x1;
  } else {
    startbit = 0x80;
  }

  bool towrite, lastmosi = !(buffer[0] & startbit);
  uint8_t bitdelay_us = (1000000 / _freq) / 2;

  for (size_t i = 0; i < len; i++) {
    uint8_t reply = 0;
    uint8_t send = buffer[i];

    /*
    Serial.print("\tSending software SPI byte 0x");
    Serial.print(send, HEX);
    Serial.print(" -> 0x");
    */

    // Serial.print(send, HEX);
    for (uint8_t b = startbit; b != 0;
         b = (_dataOrder == SPI_BITORDER_LSBFIRST) ? b << 1 : b >> 1) {

      if (bitdelay_us) {
        delayMicroseconds(bitdelay_us);
      }

      if (_dataMode == SPI_MODE0 || _dataMode == SPI_MODE2) {
        towrite = send & b;
        if ((_mosi != -1) && (lastmosi != towrite)) {
#ifdef BUSIO_USE_FAST_PINIO
          if (towrite)
            *mosiPort |= mosiPinMask;
          else
            *mosiPort &= ~mosiPinMask;
#else
          digitalWrite(_mosi, towrite);
#endif
          lastmosi = towrite;
        }

#ifdef BUSIO_USE_FAST_PINIO
        *clkPort |= clkPinMask; // Clock high
#else
        digitalWrite(_sck, HIGH);
#endif

        if (bitdelay_us) {
          delayMicroseconds(bitdelay_us);
        }

        if (_miso != -1) {
#ifdef BUSIO_USE_FAST_PINIO
          if (*misoPort & misoPinMask) {
#else
          if (digitalRead(_miso)) {
#endif
            reply |= b;
          }
        }

#ifdef BUSIO_USE_FAST_PINIO
        *clkPort &= ~clkPinMask; // Clock low
#else
        digitalWrite(_sck, LOW);
#endif
      } else { // if (_dataMode == SPI_MODE1 || _dataMode == SPI_MODE3)

#ifdef BUSIO_USE_FAST_PINIO
        *clkPort |= clkPinMask; // Clock high
#else
        digitalWrite(_sck, HIGH);
#endif

        if (bitdelay_us) {
          delayMicroseconds(bitdelay_us);
        }

        if (_mosi != -1) {
#ifdef BUSIO_USE_FAST_PINIO
          if (send & b)
            *mosiPort |= mosiPinMask;
          else
            *mosiPort &= ~mosiPinMask;
#else
          digitalWrite(_mosi, send & b);
#endif
        }

#ifdef BUSIO_USE_FAST_PINIO
        *clkPort &= ~clkPinMask; // Clock low
#else
        digitalWrite(_sck, LOW);
#endif

        if (_miso != -1) {
#ifdef BUSIO_USE_FAST_PINIO
          if (*misoPort & misoPinMask) {
#else
          if (digitalRead(_miso)) {
#endif
            reply |= b;
          }
        }
      }
      if (_miso != -1) {
        buffer[i] = reply;
      }
    }
  }
  return;
}
```

```
/*!
 *    @brief  Transfer (send/receive) one byte over hard/soft SPI, without
 * transaction management
 *    @param  send The byte to send
 *    @return The byte received while transmitting
 */
uint8_t Adafruit_SPIDevice::transfer(uint8_t send) {
  uint8_t data = send;
  transfer(&data, 1);
  return data;
}

/*!
 *    @brief  Manually begin a transaction (calls beginTransaction if hardware
 * SPI)
 */
void Adafruit_SPIDevice::beginTransaction(void) {
  if (_spi) {
#ifdef BUSIO_HAS_HW_SPI
    _spi->beginTransaction(*_spiSetting);
#endif
  }
}

/*!
 *    @brief  Manually end a transaction (calls endTransaction if hardware SPI)
 */
void Adafruit_SPIDevice::endTransaction(void) {
  if (_spi) {
#ifdef BUSIO_HAS_HW_SPI
    _spi->endTransaction();
#endif
  }
}

/*!
 *    @brief  Assert/Deassert the CS pin if it is defined
 *    @param  value The state the CS is set to
 */
void Adafruit_SPIDevice::setChipSelect(int value) {
  if (_cs != -1) {
    digitalWrite(_cs, value);
  }
}

/*!
 *    @brief  Write a buffer or two to the SPI device, with transaction
 * management.
 *    @brief  Manually begin a transaction (calls beginTransaction if hardware
 *            SPI) with asserting the CS pin
 */
void Adafruit_SPIDevice::beginTransactionWithAssertingCS() {
  beginTransaction();
  setChipSelect(LOW);
}

/*!
 *    @brief  Manually end a transaction (calls endTransaction if hardware SPI)
 *            with deasserting the CS pin
 */
void Adafruit_SPIDevice::endTransactionWithDeassertingCS() {
  setChipSelect(HIGH);
  endTransaction();
}

/*!
 *    @brief  Write a buffer or two to the SPI device, with transaction
 * management.
 *    @param  buffer Pointer to buffer of data to write
 *    @param  len Number of bytes from buffer to write
 *    @param  prefix_buffer Pointer to optional array of data to write before
 * buffer.
 *    @param  prefix_len Number of bytes from prefix buffer to write
 *    @return Always returns true because there's no way to test success of SPI
 * writes
 */
bool Adafruit_SPIDevice::write(const uint8_t *buffer, size_t len,
                               const uint8_t *prefix_buffer,
                               size_t prefix_len) {
  beginTransactionWithAssertingCS();

  // do the writing
#if defined(ARDUINO_ARCH_ESP32)
  if (_spi) {
    if (prefix_len > 0) {
      _spi->transferBytes(prefix_buffer, nullptr, prefix_len);
    }
    if (len > 0) {
      _spi->transferBytes(buffer, nullptr, len);
    }
  } else
#endif
  {
    for (size_t i = 0; i < prefix_len; i++) {
      transfer(prefix_buffer[i]);
    }
    for (size_t i = 0; i < len; i++) {
      transfer(buffer[i]);
    }
  }
  endTransactionWithDeassertingCS();

#ifdef DEBUG_SERIAL
  DEBUG_SERIAL.print(F("\tSPIDevice Wrote: "));
  if ((prefix_len != 0) && (prefix_buffer != nullptr)) {
    for (uint16_t i = 0; i < prefix_len; i++) {
      DEBUG_SERIAL.print(F("0x"));
      DEBUG_SERIAL.print(prefix_buffer[i], HEX);
      DEBUG_SERIAL.print(F(", "));
    }
  }
  for (uint16_t i = 0; i < len; i++) {
    DEBUG_SERIAL.print(F("0x"));
    DEBUG_SERIAL.print(buffer[i], HEX);
    DEBUG_SERIAL.print(F(", "));
    if (i % 32 == 31) {
```

```
      DEBUG_SERIAL.println();
    }
  }
  DEBUG_SERIAL.println();
#endif

  return true;
}

/*!
 *    @brief  Read from SPI into a buffer from the SPI device, with transaction
 * management.
 *    @param  buffer Pointer to buffer of data to read into
 *    @param  len Number of bytes from buffer to read.
 *    @param  sendvalue The 8-bits of data to write when doing the data read,
 * defaults to 0xFF
 *    @return Always returns true because there's no way to test success of SPI
 * writes
 */
bool Adafruit_SPIDevice::read(uint8_t *buffer, size_t len, uint8_t sendvalue) {
  memset(buffer, sendvalue, len); // clear out existing buffer

  beginTransactionWithAssertingCS();
  transfer(buffer, len);
  endTransactionWithDeassertingCS();

#ifdef DEBUG_SERIAL
  DEBUG_SERIAL.print(F("\tSPIDevice Read: "));
  for (uint16_t i = 0; i < len; i++) {
    DEBUG_SERIAL.print(F("0x"));
    DEBUG_SERIAL.print(buffer[i], HEX);
    DEBUG_SERIAL.print(F(", "));
    if (len % 32 == 31) {
      DEBUG_SERIAL.println();
    }
  }
  DEBUG_SERIAL.println();
#endif

  return true;
}

/*!
 *    @brief  Write some data, then read some data from SPI into another buffer,
 * with transaction management. The buffers can point to same/overlapping
 * locations. This does not transmit-receive at the same time!
 *    @param  write_buffer Pointer to buffer of data to write from
 *    @param  write_len Number of bytes from buffer to write.
 *    @param  read_buffer Pointer to buffer of data to read into.
 *    @param  read_len Number of bytes from buffer to read.
 *    @param  sendvalue The 8-bits of data to write when doing the data read,
 * defaults to 0xFF
 *    @return Always returns true because there's no way to test success of SPI
 * writes
 */
bool Adafruit_SPIDevice::write_then_read(const uint8_t *write_buffer,
                                         size_t write_len, uint8_t *read_buffer,
                                         size_t read_len, uint8_t sendvalue) {
  beginTransactionWithAssertingCS();
  // do the writing
#if defined(ARDUINO_ARCH_ESP32)
  if (_spi) {
    if (write_len > 0) {
      _spi->transferBytes(write_buffer, nullptr, write_len);
    }
  } else
#endif
  {
    for (size_t i = 0; i < write_len; i++) {
      transfer(write_buffer[i]);
    }
  }

#ifdef DEBUG_SERIAL
  DEBUG_SERIAL.print(F("\tSPIDevice Wrote: "));
  for (uint16_t i = 0; i < write_len; i++) {
    DEBUG_SERIAL.print(F("0x"));
    DEBUG_SERIAL.print(write_buffer[i], HEX);
    DEBUG_SERIAL.print(F(", "));
    if (write_len % 32 == 31) {
      DEBUG_SERIAL.println();
    }
  }
  DEBUG_SERIAL.println();
#endif

  // do the reading
  for (size_t i = 0; i < read_len; i++) {
    read_buffer[i] = transfer(sendvalue);
  }

#ifdef DEBUG_SERIAL
  DEBUG_SERIAL.print(F("\tSPIDevice Read: "));
  for (uint16_t i = 0; i < read_len; i++) {
    DEBUG_SERIAL.print(F("0x"));
    DEBUG_SERIAL.print(read_buffer[i], HEX);
    DEBUG_SERIAL.print(F(", "));
    if (read_len % 32 == 31) {
      DEBUG_SERIAL.println();
    }
  }
  DEBUG_SERIAL.println();
#endif

  endTransactionWithDeassertingCS();

  return true;
}

/*!
 *    @brief  Write some data and read some data at the same time from SPI
 * into the same buffer, with transaction management. This is basicaly a wrapper
 * for transfer() with CS-pin and transaction management. This /does/
 * transmit-receive at the same time!
 *    @param  buffer Pointer to buffer of data to write/read to/from
 *    @param  len Number of bytes from buffer to write/read.
 *    @return Always returns true because there's no way to test success of SPI
```

```
 * writes
 */
bool Adafruit_SPIDevice::write_and_read(uint8_t *buffer, size_t len) {
  beginTransactionWithAssertingCS();
  transfer(buffer, len);
  endTransactionWithDeassertingCS();

  return true;
}
```

***** Arduino.h

```
/*
  Arduino.h - Main include file for the Arduino SDK
  Copyright (c) 2005-2013 Arduino Team.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef Arduino_h
#define Arduino_h

#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <math.h>

#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/interrupt.h>

#include "binary.h"

#ifdef __cplusplus
extern "C"{
#endif

void yield(void);

#define HIGH 0x1
#define LOW  0x0

#define INPUT 0x0
#define OUTPUT 0x1
#define INPUT_PULLUP 0x2

#define PI 3.14159265358979323846264338327950
#define HALF_PI 1.57079632679489661923132169163975
#define TWO_PI 6.28318530717958647692528676655900
#define DEG_TO_RAD 0.01745329251994329576923690768488612
#define RAD_TO_DEG 57.29577951308232087679815481410517
#define EULER 2.71828182845904523536028747135266

#define SERIAL  0x0
#define DISPLAY 0x1

#define LSBFIRST 0
#define MSBFIRST 1

#define CHANGE 1
#define FALLING 2
#define RISING 3

#if defined(__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
  #define DEFAULT 0
  #define EXTERNAL 1
  #define INTERNAL1V1 2
  #define INTERNAL INTERNAL1V1
#elif defined(__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
  #define DEFAULT 0
  #define EXTERNAL 4
  #define INTERNAL1V1 8
  #define INTERNAL INTERNAL1V1
  #define INTERNAL2V56 9
  #define INTERNAL2V56_EXTCAP 13
#else
#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(__AVR_ATmega1284__) || defined(__AVR_ATmega1284P__) ||
defined(__AVR_ATmega644__) || defined(__AVR_ATmega644A__) || defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644PA__)
#define INTERNAL1V1 2
#define INTERNAL2V56 3
#else
#define INTERNAL 3
#endif
#define DEFAULT 1
#define EXTERNAL 0
#endif

// undefine stdlib's abs if encountered
#ifdef abs
#undef abs
#endif

#define min(a,b) ((a)<(b)?(a):(b))
#define max(a,b) ((a)>(b)?(a):(b))
#define abs(x) ((x)>0?(x):-(x))
#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
#define round(x)     ((x)>=0?(long)((x)+0.5):(long)((x)-0.5))
#define radians(deg) ((deg)*DEG_TO_RAD)
#define degrees(rad) ((rad)*RAD_TO_DEG)
#define sq(x) ((x)*(x))

#define interrupts() sei()
#define noInterrupts() cli()
```

```
#define clockCyclesPerMicrosecond() ( F_CPU / 1000000L )
#define clockCyclesToMicroseconds(a) ( (a) / clockCyclesPerMicrosecond() )
#define microsecondsToClockCycles(a) ( (a) * clockCyclesPerMicrosecond() )

#define lowByte(w) ((uint8_t) ((w) & 0xff))
#define highByte(w) ((uint8_t) ((w) >> 8))

#define bitRead(value, bit) (((value) >> (bit)) & 0x01)
#define bitSet(value, bit) ((value) |= (1UL << (bit)))
#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
#define bitToggle(value, bit) ((value) ^= (1UL << (bit)))
#define bitWrite(value, bit, bitvalue) ((bitvalue) ? bitSet(value, bit) : bitClear(value, bit))

// avr-libc defines _NOP() since 1.6.2
#ifndef _NOP
#define _NOP() do { __asm__ volatile ("nop"); } while (0)
#endif

typedef unsigned int word;

#define bit(b) (1UL << (b))

typedef bool boolean;
typedef uint8_t byte;

void init(void);
void initVariant(void);

int atexit(void (*func)()) __attribute__((weak));

void pinMode(uint8_t pin, uint8_t mode);
void digitalWrite(uint8_t pin, uint8_t val);
int digitalRead(uint8_t pin);
int analogRead(uint8_t pin);
void analogReference(uint8_t mode);
void analogWrite(uint8_t pin, int val);

unsigned long millis(void);
unsigned long micros(void);
void delay(unsigned long ms);
void delayMicroseconds(unsigned int us);
unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout);

void shiftOut(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder, uint8_t val);
uint8_t shiftIn(uint8_t dataPin, uint8_t clockPin, uint8_t bitOrder);

void attachInterrupt(uint8_t interruptNum, void (*userFunc)(void), int mode);
void detachInterrupt(uint8_t interruptNum);

void setup(void);
void loop(void);

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.

#define analogInPinToBit(P) (P)

// On the ATmega1280, the addresses of some of the port registers are
// greater than 255, so we can't store them in uint8_t's.
extern const uint16_t PROGMEM port_to_mode_PGM[];
extern const uint16_t PROGMEM port_to_input_PGM[];
extern const uint16_t PROGMEM port_to_output_PGM[];

extern const uint8_t PROGMEM digital_pin_to_port_PGM[];
// extern const uint8_t PROGMEM digital_pin_to_bit_PGM[];
extern const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[];
extern const uint8_t PROGMEM digital_pin_to_timer_PGM[];

// Get the bit location within the hardware port of the given virtual pin.
// This comes from the pins_*.c file for the active board configuration.
//
// These perform slightly better as macros compared to inline functions
//
#define digitalPinToPort(P) ( pgm_read_byte( digital_pin_to_port_PGM + (P) ) )
#define digitalPinToBitMask(P) ( pgm_read_byte( digital_pin_to_bit_mask_PGM + (P) ) )
#define digitalPinToTimer(P) ( pgm_read_byte( digital_pin_to_timer_PGM + (P) ) )
#define analogInPinToBit(P) (P)
#define portOutputRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_output_PGM + (P))) )
#define portInputRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_input_PGM + (P))) )
#define portModeRegister(P) ( (volatile uint8_t *)( pgm_read_word( port_to_mode_PGM + (P))) )

#define NOT_A_PIN 0
#define NOT_A_PORT 0

#define NOT_AN_INTERRUPT -1

#ifdef ARDUINO_MAIN
#define PA 1
#define PB 2
#define PC 3
#define PD 4
#define PE 5
#define PF 6
#define PG 7
#define PH 8
#define PJ 10
#define PK 11
#define PL 12
#endif

#define NOT_ON_TIMER 0
#define TIMER0A 1
#define TIMER0B 2
#define TIMER1A 3
#define TIMER1B 4
#define TIMER1C 5
#define TIMER2  6
#define TIMER2A 7
#define TIMER2B 8

#define TIMER3A 9
#define TIMER3B 10
#define TIMER3C 11
#define TIMER4A 12
#define TIMER4B 13
#define TIMER4C 14
```

```
#define TIMER4D 15
#define TIMER5A 16
#define TIMER5B 17
#define TIMER5C 18

#ifdef __cplusplus
} // extern "C"
#endif

#ifdef __cplusplus
#include "WCharacter.h"
#include "WString.h"
#include "HardwareSerial.h"
#include "USBAPI.h"
#if defined(HAVE_HWSERIAL0) && defined(HAVE_CDCSERIAL)
#error "Targets with both UART0 and CDC serial not supported"
#endif

uint16_t makeWord(uint16_t w);
uint16_t makeWord(byte h, byte l);

#define word(...) makeWord(__VA_ARGS__)

unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout = 1000000L);
unsigned long pulseInLong(uint8_t pin, uint8_t state, unsigned long timeout = 1000000L);

void tone(uint8_t _pin, unsigned int frequency, unsigned long duration = 0);
void noTone(uint8_t _pin);

// WMath prototypes
long random(long);
long random(long, long);
void randomSeed(unsigned long);
long map(long, long, long, long, long);

#endif

#include "pins_arduino.h"

#endif
```

***** pins_arduino.h

```
/*
  pins_arduino.h - Pin definition functions for Arduino
  Part of Arduino - http://www.arduino.cc/

  Copyright (c) 2007 David A. Mellis

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA
*/

#ifndef Pins_Arduino_h
#define Pins_Arduino_h

#include <avr/pgmspace.h>

#define NUM_DIGITAL_PINS            20
#define NUM_ANALOG_INPUTS           6
#define analogInputToDigitalPin(p)  ((p < 6) ? (p) + 14 : -1)

#if defined(__AVR_ATmega8__)
#define digitalPinHasPWM(p)         ((p) == 9 || (p) == 10 || (p) == 11)
#else
#define digitalPinHasPWM(p)         ((p) == 3 || (p) == 5 || (p) == 6 || (p) == 9 || (p) == 10 || (p) == 11)
#endif

#define PIN_SPI_SS    (10)
#define PIN_SPI_MOSI  (11)
#define PIN_SPI_MISO  (12)
#define PIN_SPI_SCK   (13)

static const uint8_t SS   = PIN_SPI_SS;
static const uint8_t MOSI = PIN_SPI_MOSI;
static const uint8_t MISO = PIN_SPI_MISO;
static const uint8_t SCK  = PIN_SPI_SCK;

#define PIN_WIRE_SDA        (18)
#define PIN_WIRE_SCL        (19)

static const uint8_t SDA = PIN_WIRE_SDA;
static const uint8_t SCL = PIN_WIRE_SCL;

#define LED_BUILTIN 13

#define PIN_A0   (14)
#define PIN_A1   (15)
#define PIN_A2   (16)
#define PIN_A3   (17)
#define PIN_A4   (18)
#define PIN_A5   (19)
#define PIN_A6   (20)
#define PIN_A7   (21)

static const uint8_t A0 = PIN_A0;
static const uint8_t A1 = PIN_A1;
static const uint8_t A2 = PIN_A2;
static const uint8_t A3 = PIN_A3;
static const uint8_t A4 = PIN_A4;
static const uint8_t A5 = PIN_A5;
static const uint8_t A6 = PIN_A6;
static const uint8_t A7 = PIN_A7;
```

```
#define digitalPinToPCICR(p)    (((p) >= 0 && (p) <= 21) ? (&PCICR) : ((uint8_t *)0))
#define digitalPinToPCICRbit(p) (((p) <= 7) ? 2 : (((p) <= 13) ? 0 : 1))
#define digitalPinToPCMSK(p)    (((p) <= 7) ? (&PCMSK2) : (((p) <= 13) ? (&PCMSK0) : (((p) <= 21) ? (&PCMSK1) : ((uint8_t *)0))))
#define digitalPinToPCMSKbit(p) (((p) <= 7) ? (p) : (((p) <= 13) ? ((p) - 8) : ((p) - 14)))

#define digitalPinToInterrupt(p)  ((p) == 2 ? 0 : ((p) == 3 ? 1 : NOT_AN_INTERRUPT))

#ifdef ARDUINO_MAIN

// On the Arduino board, digital pins are also used
// for the analog output (software PWM).  Analog input
// pins are a separate set.

// ATMEL ATMEGA8 & 168 / ARDUINO
//
//                  +-\/-+
//            PC6  1|    |28  PC5 (AI 5)
//      (D 0) PD0  2|    |27  PC4 (AI 4)
//      (D 1) PD1  3|    |26  PC3 (AI 3)
//      (D 2) PD2  4|    |25  PC2 (AI 2)
// PWM+ (D 3) PD3  5|    |24  PC1 (AI 1)
//      (D 4) PD4  6|    |23  PC0 (AI 0)
//            VCC  7|    |22  GND
//            GND  8|    |21  AREF
//            PB6  9|    |20  AVCC
//            PB7 10|    |19  PB5 (D 13)
// PWM+ (D 5) PD5 11|    |18  PB4 (D 12)
// PWM+ (D 6) PD6 12|    |17  PB3 (D 11) PWM
//      (D 7) PD7 13|    |16  PB2 (D 10) PWM
//      (D 8) PB0 14|    |15  PB1 (D 9) PWM
//                  +----+
//
// (PWM+ indicates the additional PWM pins on the ATmega168.)

// ATMEL ATMEGA1280 / ARDUINO
//
// 0-7 PE0-PE7   works
// 8-13 PB0-PB5  works
// 14-21 PA0-PA7 works
// 22-29 PH0-PH7 works
// 30-35 PG5-PG0 works
// 36-43 PC7-PC0 works
// 44-51 PJ7-PJ0 works
// 52-59 PL7-PL0 works
// 60-67 PD7-PD0 works
// A0-A7 PF0-PF7
// A8-A15 PK0-PK7


// these arrays map port names (e.g. port B) to the
// appropriate addresses for various functions (e.g. reading
// and writing)
const uint16_t PROGMEM port_to_mode_PGM[] = {
          NOT_A_PORT,
          NOT_A_PORT,
          (uint16_t) &DDRB,
          (uint16_t) &DDRC,
          (uint16_t) &DDRD,
};

const uint16_t PROGMEM port_to_output_PGM[] = {
          NOT_A_PORT,
          NOT_A_PORT,
          (uint16_t) &PORTB,
          (uint16_t) &PORTC,
          (uint16_t) &PORTD,
};

const uint16_t PROGMEM port_to_input_PGM[] = {
          NOT_A_PORT,
          NOT_A_PORT,
          (uint16_t) &PINB,
          (uint16_t) &PINC,
          (uint16_t) &PIND,
};

const uint8_t PROGMEM digital_pin_to_port_PGM[] = {
          PD, /* 0 */
          PD,
          PD,
          PD,
          PD,
          PD,
          PD,
          PD,
          PB, /* 8 */
          PB,
          PB,
          PB,
          PB,
          PB,
          PC, /* 14 */
          PC,
          PC,
          PC,
          PC,
          PC,
};

const uint8_t PROGMEM digital_pin_to_bit_mask_PGM[] = {
          _BV(0), /* 0, port D */
          _BV(1),
          _BV(2),
          _BV(3),
          _BV(4),
          _BV(5),
          _BV(6),
          _BV(7),
          _BV(0), /* 8, port B */
          _BV(1),
          _BV(2),
          _BV(3),
          _BV(4),
          _BV(5),
          _BV(0), /* 14, port C */
          _BV(1),
          _BV(2),
```

```
        _BV(3),
        _BV(4),
        _BV(5),
};

const uint8_t PROGMEM digital_pin_to_timer_PGM[] = {
        NOT_ON_TIMER, /* 0 - port D */
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        // on the ATmega168, digital pin 3 has hardware pwm
#if defined(__AVR_ATmega8__)
        NOT_ON_TIMER,
#else
        TIMER2B,
#endif
        NOT_ON_TIMER,
        // on the ATmega168, digital pins 5 and 6 have hardware pwm
#if defined(__AVR_ATmega8__)
        NOT_ON_TIMER,
        NOT_ON_TIMER,
#else
        TIMER0B,
        TIMER0A,
#endif
        NOT_ON_TIMER,
        NOT_ON_TIMER, /* 8 - port B */
        TIMER1A,
        TIMER1B,
#if defined(__AVR_ATmega8__)
        TIMER2,
#else
        TIMER2A,
#endif
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER, /* 14 - port C */
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER,
        NOT_ON_TIMER,
};

#endif

// These serial port names are intended to allow libraries and architecture-neutral
// sketches to automatically default to the correct port name for a particular type
// of use.  For example, a GPS module would normally connect to SERIAL_PORT_HARDWARE_OPEN,
// the first hardware serial port whose RX/TX pins are not dedicated to another use.
//
// SERIAL_PORT_MONITOR        Port which normally prints to the Arduino Serial Monitor
//
// SERIAL_PORT_USBVIRTUAL     Port which is USB virtual serial
//
// SERIAL_PORT_LINUXBRIDGE    Port which connects to a Linux system via Bridge library
//
// SERIAL_PORT_HARDWARE       Hardware serial port, physical RX & TX pins.
//
// SERIAL_PORT_HARDWARE_OPEN  Hardware serial ports which are open for use.  Their RX & TX
//                            pins are NOT connected to anything by default.
#define SERIAL_PORT_MONITOR   Serial
#define SERIAL_PORT_HARDWARE  Serial

#endif
```

***** wiring_private.h

```
/*
  wiring_private.h - Internal header file.
  Part of Arduino - http://www.arduino.cc/

  Copyright (c) 2005-2006 David A. Mellis

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General
  Public License along with this library; if not, write to the
  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
  Boston, MA  02111-1307  USA
*/

#ifndef WiringPrivate_h
#define WiringPrivate_h

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdarg.h>

#include "Arduino.h"

#ifdef __cplusplus
extern "C"{
#endif

#ifndef cbi
#define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#endif
#ifndef sbi
#define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#endif

uint32_t countPulseASM(volatile uint8_t *port, uint8_t bit, uint8_t stateMask, unsigned long maxloops);

#define EXTERNAL_INT_0 0
#define EXTERNAL_INT_1 1
#define EXTERNAL_INT_2 2
#define EXTERNAL_INT_3 3
```

```
#define EXTERNAL_INT_4 4
#define EXTERNAL_INT_5 5
#define EXTERNAL_INT_6 6
#define EXTERNAL_INT_7 7

#if defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__) || defined(__AVR_ATmega128RFA1__) || defined(__AVR_ATmega256RFR2__) || \
    defined(__AVR_AT90USB82__) || defined(__AVR_AT90USB162__) || defined(__AVR_ATmega32U2__) || defined(__AVR_ATmega16U2__) ||
defined(__AVR_ATmega8U2__)
#define EXTERNAL_NUM_INTERRUPTS 8
#elif defined(__AVR_ATmega1284__) || defined(__AVR_ATmega1284P__) || defined(__AVR_ATmega644__) || defined(__AVR_ATmega644A__) ||
defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644PA__)
#define EXTERNAL_NUM_INTERRUPTS 3
#elif defined(__AVR_ATmega32U4__)
#define EXTERNAL_NUM_INTERRUPTS 5
#else
#define EXTERNAL_NUM_INTERRUPTS 2
#endif

typedef void (*voidFuncPtr)(void);

#ifdef __cplusplus
} // extern "C"
#endif

#endif
```

***** Print.h

```
/*
  Print.h - Base class that provides print() and println()
  Copyright (c) 2008 David A. Mellis.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#ifndef Print_h
#define Print_h

#include <inttypes.h>
#include <stdio.h> // for size_t

#include "WString.h"
#include "Printable.h"

#define DEC 10
#define HEX 16
#define OCT 8
#ifdef BIN // Prevent warnings if BIN is previously defined in "iotnx4.h" or similar
#undef BIN
#endif
#define BIN 2

class Print
{
  private:
    int write_error;
    size_t printNumber(unsigned long, uint8_t);
    size_t printFloat(double, uint8_t);
  protected:
    void setWriteError(int err = 1) { write_error = err; }
  public:
    Print() : write_error(0) {}

    int getWriteError() { return write_error; }
    void clearWriteError() { setWriteError(0); }

    virtual size_t write(uint8_t) = 0;
    size_t write(const char *str) {
      if (str == NULL) return 0;
      return write((const uint8_t *)str, strlen(str));
    }
    virtual size_t write(const uint8_t *buffer, size_t size);
    size_t write(const char *buffer, size_t size) {
      return write((const uint8_t *)buffer, size);
    }

    // default to zero, meaning "a single write may block"
    // should be overridden by subclasses with buffering
    virtual int availableForWrite() { return 0; }

    size_t print(const __FlashStringHelper *);
    size_t print(const String &);
    size_t print(const char[]);
    size_t print(char);
    size_t print(unsigned char, int = DEC);
    size_t print(int, int = DEC);
    size_t print(unsigned int, int = DEC);
    size_t print(long, int = DEC);
    size_t print(unsigned long, int = DEC);
    size_t print(double, int = 2);
    size_t print(const Printable&);

    size_t println(const __FlashStringHelper *);
    size_t println(const String &s);
    size_t println(const char[]);
    size_t println(char);
    size_t println(unsigned char, int = DEC);
    size_t println(int, int = DEC);
    size_t println(unsigned int, int = DEC);
    size_t println(long, int = DEC);
    size_t println(unsigned long, int = DEC);
    size_t println(double, int = 2);
    size_t println(const Printable&);
    size_t println(void);
```

```
    virtual void flush() { /* Empty implementation for backward compatibility */ }
};

#endif
```

***** Print.cpp

```
/*
 Print.cpp - Base class that provides print() and println()
 Copyright (c) 2008 David A. Mellis.  All right reserved.

 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

 Modified 23 November 2006 by David A. Mellis
 Modified 03 August 2015 by Chuck Todd
 */

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include "Arduino.h"

#include "Print.h"

// Public Methods //////////////////////////////////////////////////////////////

/* default implementation: may be overridden */
size_t Print::write(const uint8_t *buffer, size_t size)
{
  size_t n = 0;
  while (size--) {
    if (write(*buffer++)) n++;
    else break;
  }
  return n;
}

size_t Print::print(const __FlashStringHelper *ifsh)
{
  PGM_P p = reinterpret_cast<PGM_P>(ifsh);
  size_t n = 0;
  while (1) {
    unsigned char c = pgm_read_byte(p++);
    if (c == 0) break;
    if (write(c)) n++;
    else break;
  }
  return n;
}

size_t Print::print(const String &s)
{
  return write(s.c_str(), s.length());
}

size_t Print::print(const char str[])
{
  return write(str);
}

size_t Print::print(char c)
{
  return write(c);
}

size_t Print::print(unsigned char b, int base)
{
  return print((unsigned long) b, base);
}

size_t Print::print(int n, int base)
{
  return print((long) n, base);
}

size_t Print::print(unsigned int n, int base)
{
  return print((unsigned long) n, base);
}

size_t Print::print(long n, int base)
{
  if (base == 0) {
    return write(n);
  } else if (base == 10) {
    if (n < 0) {
      int t = print('-');
      n = -n;
      return printNumber(n, 10) + t;
    }
    return printNumber(n, 10);
  } else {
    return printNumber(n, base);
  }
}

size_t Print::print(unsigned long n, int base)
{
  if (base == 0) return write(n);
  else return printNumber(n, base);
}
```

```
size_t Print::print(double n, int digits)
{
  return printFloat(n, digits);
}

size_t Print::println(const __FlashStringHelper *ifsh)
{
  size_t n = print(ifsh);
  n += println();
  return n;
}

size_t Print::print(const Printable& x)
{
  return x.printTo(*this);
}

size_t Print::println(void)
{
  return write("\r\n");
}

size_t Print::println(const String &s)
{
  size_t n = print(s);
  n += println();
  return n;
}

size_t Print::println(const char c[])
{
  size_t n = print(c);
  n += println();
  return n;
}

size_t Print::println(char c)
{
  size_t n = print(c);
  n += println();
  return n;
}

size_t Print::println(unsigned char b, int base)
{
  size_t n = print(b, base);
  n += println();
  return n;
}

size_t Print::println(int num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(unsigned int num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(long num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(unsigned long num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(double num, int digits)
{
  size_t n = print(num, digits);
  n += println();
  return n;
}

size_t Print::println(const Printable& x)
{
  size_t n = print(x);
  n += println();
  return n;
}

// Private Methods /////////////////////////////////////////////////////////////

size_t Print::printNumber(unsigned long n, uint8_t base)
{
  char buf[8 * sizeof(long) + 1]; // Assumes 8-bit chars plus zero byte.
  char *str = &buf[sizeof(buf) - 1];

  *str = '\0';

  // prevent crash if called with base == 1
  if (base < 2) base = 10;

  do {
    char c = n % base;
    n /= base;

    *--str = c < 10 ? c + '0' : c + 'A' - 10;
  } while(n);

  return write(str);
}

size_t Print::printFloat(double number, uint8_t digits)
{
  size_t n = 0;
```

```
  if (isnan(number)) return print("nan");
  if (isinf(number)) return print("inf");
  if (number > 4294967040.0) return print ("ovf");  // constant determined empirically
  if (number <-4294967040.0) return print ("ovf");  // constant determined empirically

  // Handle negative numbers
  if (number < 0.0)
  {
     n += print('-');
     number = -number;
  }

  // Round correctly so that print(1.999, 2) prints as "2.00"
  double rounding = 0.5;
  for (uint8_t i=0; i<digits; ++i)
    rounding /= 10.0;

  number += rounding;

  // Extract the integer part of the number and print it
  unsigned long int_part = (unsigned long)number;
  double remainder = number - (double)int_part;
  n += print(int_part);

  // Print the decimal point, but only if there are digits beyond
  if (digits > 0) {
    n += print('.');
  }

  // Extract digits from the remainder one at a time
  while (digits-- > 0)
  {
    remainder *= 10.0;
    unsigned int toPrint = (unsigned int)(remainder);
    n += print(toPrint);
    remainder -= toPrint;
  }

  return n;
}
```

***** pgmspace.h

```
/* $Id$ */

/*
   pgmspace.h

   Contributors:
     Created by Marek Michalkiewicz <marekm@linux.org.pl>
     Eric B. Weddington <eric@ecentral.com>
     Wolfgang Haidinger <wh@vmars.tuwien.ac.at> (pgm_read_dword())
     Ivanov Anton <anton@arc.com.ru> (pgm_read_float())
 */

/** \file */
/** \defgroup avr_pgmspace <avr/pgmspace.h>: Program Space Utilities
    \code
    #include <avr/io.h>
    #include <avr/pgmspace.h>
    \endcode

    The functions in this module provide interfaces for a program to access
    data stored in program space (flash memory) of the device.  In order to
    use these functions, the target device must support either the \c LPM or
    \c ELPM instructions.

    \note These functions are an attempt to provide some compatibility with
    header files that come with IAR C, to make porting applications between
    different compilers easier.  This is not 100% compatibility though (GCC
    does not have full support for multiple address spaces yet).

    \note If you are working with strings which are completely based in ram,
    use the standard string functions described in \ref avr_string.

    \note If possible, put your constant tables in the lower 64 KB and use
    pgm_read_byte_near() or pgm_read_word_near() instead of
    pgm_read_byte_far() or pgm_read_word_far() since it is more efficient that
    way, and you can still use the upper 64K for executable code.
    All functions that are suffixed with a \c _P \e require their
    arguments to be in the lower 64 KB of the flash ROM, as they do
    not use ELPM instructions.  This is normally not a big concern as
    the linker setup arranges any program space constants declared
    using the macros from this header file so they are placed right after
    the interrupt vectors, and in front of any executable code.  However,
    it can become a problem if there are too many of these constants, or
    for bootloaders on devices with more than 64 KB of ROM.
    <em>All these functions will not work in that situation.</em>

    \note For <b>Xmega</b> devices, make sure the NVM controller
    command register (\c NVM.CMD or \c NVM_CMD) is set to 0x00 (NOP)
    before using any of these functions.
*/

#ifndef __PGMSPACE_H_
#define __PGMSPACE_H_ 1

#ifndef __DOXYGEN__
#define __need_size_t
#endif
#include <inttypes.h>
#include <stddef.h>
#include <avr/io.h>

#ifndef __DOXYGEN__
#ifndef __ATTR_CONST__
#define __ATTR_CONST__ __attribute__((__const__))
#endif

#ifndef __ATTR_PROGMEM__
#define __ATTR_PROGMEM__ __attribute__((__progmem__))
#endif

#ifndef __ATTR_PURE__
#define __ATTR_PURE__ __attribute__((__pure__))
```

```
#endif
#endif    /* !__DOXYGEN__ */

/**
   \ingroup avr_pgmspace
   \def PROGMEM

   Attribute to use in order to declare an object being located in
   flash ROM.
 */
#define PROGMEM __ATTR_PROGMEM__

#ifdef __cplusplus
extern "C" {
#endif

#if defined(__DOXYGEN__)
/*
 * Doxygen doesn't grok the appended attribute syntax of
 * GCC, and confuses the typedefs with function decls, so
 * supply a doxygen-friendly view.
 */

/**
   \ingroup avr_pgmspace
   \typedef prog_void
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of a "void" object located in flash ROM.  Does not make much
   sense by itself, but can be used to declare a "void *" object in
   flash ROM.
*/
typedef void PROGMEM prog_void;

/**
   \ingroup avr_pgmspace
   \typedef prog_char
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of a "char" object located in flash ROM.
*/
typedef char PROGMEM prog_char;

/**
   \ingroup avr_pgmspace
   \typedef prog_uchar
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "unsigned char" object located in flash ROM.
*/
typedef unsigned char PROGMEM prog_uchar;

/**
   \ingroup avr_pgmspace
   \typedef prog_int8_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "int8_t" object located in flash ROM.
*/
typedef int8_t PROGMEM prog_int8_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint8_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "uint8_t" object located in flash ROM.
*/
typedef uint8_t PROGMEM prog_uint8_t;

/**
   \ingroup avr_pgmspace
```

```
   \typedef prog_int16_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "int16_t" object located in flash ROM.
*/
typedef int16_t PROGMEM prog_int16_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint16_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "uint16_t" object located in flash ROM.
*/
typedef uint16_t PROGMEM prog_uint16_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_int32_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "int32_t" object located in flash ROM.
*/
typedef int32_t PROGMEM prog_int32_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint32_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "uint32_t" object located in flash ROM.
*/
typedef uint32_t PROGMEM prog_uint32_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_int64_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "int64_t" object located in flash ROM.

   \note This type is not available when the compiler
   option -mint8 is in effect.
*/
typedef int64_t PROGMEM prog_int64_t;

/**
   \ingroup avr_pgmspace
   \typedef prog_uint64_t
   \note DEPRECATED

   This typedef is now deprecated because the usage of the __progmem__
   attribute on a type is not supported in GCC. However, the use of the
   __progmem__ attribute on a variable declaration is supported, and this is
   now the recommended usage.

   The typedef is only visible if the macro __PROG_TYPES_COMPAT__
   has been defined before including <avr/pgmspace.h> (either by a
   \c \#define directive, or by a -D compiler option.)

   Type of an "uint64_t" object located in flash ROM.

   \note This type is not available when the compiler
   option -mint8 is in effect.
*/
typedef uint64_t PROGMEM prog_uint64_t;

/** \ingroup avr_pgmspace
    \def PGM_P

    Used to declare a variable that is a pointer to a string in program
    space. */
```

```
#ifndef PGM_P
#define PGM_P const char *
#endif

/** \ingroup avr_pgmspace
    \def PGM_VOID_P

    Used to declare a generic pointer to an object in program space. */

#ifndef PGM_VOID_P
#define PGM_VOID_P const void *
#endif

#elif defined(__PROG_TYPES_COMPAT__)  /* !DOXYGEN */

typedef void prog_void __attribute__((__progmem__,deprecated("prog_void type is deprecated.")));
typedef char prog_char __attribute__((__progmem__,deprecated("prog_char type is deprecated.")));
typedef unsigned char prog_uchar __attribute__((__progmem__,deprecated("prog_uchar type is deprecated.")));
typedef int8_t    prog_int8_t   __attribute__((__progmem__,deprecated("prog_int8_t type is deprecated.")));
typedef uint8_t   prog_uint8_t  __attribute__((__progmem__,deprecated("prog_uint8_t type is deprecated.")));
typedef int16_t   prog_int16_t  __attribute__((__progmem__,deprecated("prog_int16_t type is deprecated.")));
typedef uint16_t  prog_uint16_t __attribute__((__progmem__,deprecated("prog_uint16_t type is deprecated.")));
typedef int32_t   prog_int32_t  __attribute__((__progmem__,deprecated("prog_int32_t type is deprecated.")));
typedef uint32_t  prog_uint32_t __attribute__((__progmem__,deprecated("prog_uint32_t type is deprecated.")));
#if !__USING_MINT8
typedef int64_t   prog_int64_t  __attribute__((__progmem__,deprecated("prog_int64_t type is deprecated.")));
typedef uint64_t  prog_uint64_t __attribute__((__progmem__,deprecated("prog_uint64_t type is deprecated.")));
#endif

#ifndef PGM_P
#define PGM_P const prog_char *
#endif

#ifndef PGM_VOID_P
#define PGM_VOID_P const prog_void *
#endif

#else /* !defined(__DOXYGEN__), !defined(__PROG_TYPES_COMPAT__) */

#ifndef PGM_P
#define PGM_P const char *
#endif

#ifndef PGM_VOID_P
#define PGM_VOID_P const void *
#endif
#endif /* defined(__DOXYGEN__), defined(__PROG_TYPES_COMPAT__) */

/* Although in C, we can get away with just using __c, it does not work in
   C++. We need to use &__c[0] to avoid the compiler puking. Dave Hylands
   explaned it thusly,

     Let's suppose that we use PSTR("Test"). In this case, the type returned
     by __c is a prog_char[5] and not a prog_char *. While these are
     compatible, they aren't the same thing (especially in C++). The type
     returned by &__c[0] is a prog_char *, which explains why it works
     fine. */

#if defined(__DOXYGEN__)
/*
 * The #define below is just a dummy that serves documentation
 * purposes only.
 */
/** \ingroup avr_pgmspace
    \def PSTR(s)

    Used to declare a static pointer to a string in program space. */
# define PSTR(s) ((const PROGMEM char *)(s))
#else  /* !DOXYGEN */
/* The real thing. */
# define PSTR(s) (__extension__({static const char __c[] PROGMEM = (s); &__c[0];}))
#endif /* DOXYGEN */

#ifndef __DOXYGEN__ /* Internal macros, not documented. */
#define __LPM_classic__(addr)   \
(__extension__({                \
    uint16_t __addr16 = (uint16_t)(addr); \
    uint8_t __result;           \
    __asm__ __volatile__        \
    (                           \
        "lpm" "\n\t"            \
        "mov %0, r0" "\n\t"     \
        : "=r" (__result)       \
        : "z" (__addr16)        \
        : "r0"                  \
    );                          \
    __result;                   \
}))

#define __LPM_tiny__(addr)      \
(__extension__({                \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    uint8_t __result;           \
    __asm__                     \
    (                           \
        "ld %0, z" "\n\t"       \
        : "=r" (__result)       \
        : "z" (__addr16)        \
    );                          \
    __result;                   \
}))

#define __LPM_enhanced__(addr)  \
(__extension__({                \
    uint16_t __addr16 = (uint16_t)(addr); \
    uint8_t __result;           \
    __asm__ __volatile__        \
    (                           \
        "lpm %0, Z" "\n\t"      \
        : "=r" (__result)       \
        : "z" (__addr16)        \
    );                          \
    __result;                   \
}))

#define __LPM_word_classic__(addr)          \
```

```
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint16_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm"           "\n\t"              \
        "mov %A0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %B0, r0"   "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
        : "r0"                              \
    );                                      \
    __result;                               \
}))

#define __LPM_word_tiny__(addr)             \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    uint16_t __result;                      \
    __asm__                                 \
    (                                       \
        "ld %A0, z+"     "\n\t"             \
        "ld %B0, z"      "\n\t"             \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_word_enhanced__(addr)         \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint16_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm %A0, Z+"   "\n\t"              \
        "lpm %B0, Z"    "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_dword_classic__(addr)         \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint32_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm"           "\n\t"              \
        "mov %A0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %B0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %C0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %D0, r0"   "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
        : "r0"                              \
    );                                      \
    __result;                               \
}))

#define __LPM_dword_tiny__(addr)            \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    uint32_t __result;                      \
    __asm__                                 \
    (                                       \
        "ld %A0, z+"    "\n\t"              \
        "ld %B0, z+"    "\n\t"              \
        "ld %C0, z+"    "\n\t"              \
        "ld %D0, z"     "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_dword_enhanced__(addr)        \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    uint32_t __result;                      \
    __asm__ __volatile__                    \
    (                                       \
        "lpm %A0, Z+"   "\n\t"              \
        "lpm %B0, Z+"   "\n\t"              \
        "lpm %C0, Z+"   "\n\t"              \
        "lpm %D0, Z"    "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_float_classic__(addr)         \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    float __result;                         \
    __asm__ __volatile__                    \
    (                                       \
        "lpm"           "\n\t"              \
        "mov %A0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %B0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
        "mov %C0, r0"   "\n\t"              \
        "adiw r30, 1"   "\n\t"              \
        "lpm"           "\n\t"              \
```

```
        "mov %D0, r0"   "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
        : "r0"                              \
    );                                      \
    __result;                               \
}))

#define __LPM_float_tiny__(addr)            \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr) + __AVR_TINY_PM_BASE_ADDRESS__; \
    float __result;                         \
    __asm__                                 \
    (                                       \
        "ld %A0, z+"   "\n\t"               \
        "ld %B0, z+"   "\n\t"               \
        "ld %C0, z+"   "\n\t"               \
        "ld %D0, z"    "\n\t"               \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#define __LPM_float_enhanced__(addr)        \
(__extension__({                            \
    uint16_t __addr16 = (uint16_t)(addr);   \
    float __result;                         \
    __asm__ __volatile__                    \
    (                                       \
        "lpm %A0, Z+"   "\n\t"              \
        "lpm %B0, Z+"   "\n\t"              \
        "lpm %C0, Z+"   "\n\t"              \
        "lpm %D0, Z"    "\n\t"              \
        : "=r" (__result), "=z" (__addr16)  \
        : "1" (__addr16)                    \
    );                                      \
    __result;                               \
}))

#if defined (__AVR_HAVE_LPMX__)
#define __LPM(addr)          __LPM_enhanced__(addr)
#define __LPM_word(addr)     __LPM_word_enhanced__(addr)
#define __LPM_dword(addr)    __LPM_dword_enhanced__(addr)
#define __LPM_float(addr)    __LPM_float_enhanced__(addr)
/*
Macro to read data from program memory for avr tiny parts(tiny 4/5/9/10/20/40).
why:
- LPM instruction is not available in AVR_TINY instruction set.
- Programs are executed starting from address 0x0000 in program memory.
But it must be addressed starting from 0x4000 when accessed via data memory.
Reference: TINY device (ATTiny 4,5,9,10,20 and 40) datasheets
Bug: avrtc-536
*/
#elif defined (__AVR_TINY__)
#define __LPM(addr)          __LPM_tiny__(addr)
#define __LPM_word(addr)     __LPM_word_tiny__(addr)
#define __LPM_dword(addr)    __LPM_dword_tiny__(addr)
#define __LPM_float(addr)    __LPM_float_tiny__(addr)
#else
#define __LPM(addr)          __LPM_classic__(addr)
#define __LPM_word(addr)     __LPM_word_classic__(addr)
#define __LPM_dword(addr)    __LPM_dword_classic__(addr)
#define __LPM_float(addr)    __LPM_float_classic__(addr)
#endif

#endif  /* !__DOXYGEN__ */

/** \ingroup avr_pgmspace
    \def pgm_read_byte_near(address_short)
    Read a byte from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_byte_near(address_short) __LPM((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_word_near(address_short)
    Read a word from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_word_near(address_short) __LPM_word((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_dword_near(address_short)
    Read a double word from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_dword_near(address_short) \
    __LPM_dword((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_float_near(address_short)
    Read a float from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_float_near(address_short) \
    __LPM_float((uint16_t)(address_short))

/** \ingroup avr_pgmspace
    \def pgm_read_ptr_near(address_short)
    Read a pointer from the program space with a 16-bit (near) address.
    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_ptr_near(address_short) \
    (void*)__LPM_word((uint16_t)(address_short))

#if defined(RAMPZ) || defined(__DOXYGEN__)

/* Only for devices with more than 64K of program memory.
   RAMPZ must be defined (see iom103.h, iom128.h).
*/
```

```
/* The classic functions are needed for ATmega103. */
#ifndef __DOXYGEN__           /* These are internal macros, avoid "is
                                 not documented" warnings. */
#define __ELPM_classic__(addr)      \
(__extension__({                    \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;               \
    __asm__ __volatile__            \
    (                               \
        "out %2, %C1" "\n\t"        \
        "mov r31, %B1" "\n\t"       \
        "mov r30, %A1" "\n\t"       \
        "elpm" "\n\t"               \
        "mov %0, r0" "\n\t"         \
        : "=r" (__result)           \
        : "r" (__addr32),           \
          "I" (_SFR_IO_ADDR(RAMPZ)) \
        : "r0", "r30", "r31"        \
    );                              \
    __result;                       \
}))

#define __ELPM_enhanced__(addr)     \
(__extension__({                    \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;               \
    __asm__ __volatile__            \
    (                               \
        "out %2, %C1" "\n\t"        \
        "movw r30, %1" "\n\t"       \
        "elpm %0, Z+" "\n\t"        \
        : "=r" (__result)           \
        : "r" (__addr32),           \
          "I" (_SFR_IO_ADDR(RAMPZ)) \
        : "r30", "r31"              \
    );                              \
    __result;                       \
}))

#define __ELPM_xmega__(addr)        \
(__extension__({                    \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint8_t __result;               \
    __asm__ __volatile__            \
    (                               \
        "in __tmp_reg__, %2" "\n\t" \
        "out %2, %C1" "\n\t"        \
        "movw r30, %1" "\n\t"       \
        "elpm %0, Z+" "\n\t"        \
        "out %2, __tmp_reg__"       \
        : "=r" (__result)           \
        : "r" (__addr32),           \
          "I" (_SFR_IO_ADDR(RAMPZ)) \
        : "r30", "r31"              \
    );                              \
    __result;                       \
}))

#define __ELPM_word_classic__(addr)     \
(__extension__({                        \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint16_t __result;                  \
    __asm__ __volatile__                \
    (                                   \
        "out %2, %C1"   "\n\t"          \
        "mov r31, %B1"  "\n\t"          \
        "mov r30, %A1"  "\n\t"          \
        "elpm"          "\n\t"          \
        "mov %A0, r0"   "\n\t"          \
        "in r0, %2"     "\n\t"          \
        "adiw r30, 1"   "\n\t"          \
        "adc r0, __zero_reg__" "\n\t"   \
        "out %2, r0"    "\n\t"          \
        "elpm"          "\n\t"          \
        "mov %B0, r0"   "\n\t"          \
        : "=r" (__result)               \
        : "r" (__addr32),               \
          "I" (_SFR_IO_ADDR(RAMPZ))     \
        : "r0", "r30", "r31"            \
    );                                  \
    __result;                           \
}))

#define __ELPM_word_enhanced__(addr)    \
(__extension__({                        \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint16_t __result;                  \
    __asm__ __volatile__                \
    (                                   \
        "out %2, %C1"   "\n\t"          \
        "movw r30, %1"  "\n\t"          \
        "elpm %A0, Z+"  "\n\t"          \
        "elpm %B0, Z"   "\n\t"          \
        : "=r" (__result)               \
        : "r" (__addr32),               \
          "I" (_SFR_IO_ADDR(RAMPZ))     \
        : "r30", "r31"                  \
    );                                  \
    __result;                           \
}))

#define __ELPM_word_xmega__(addr)       \
(__extension__({                        \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint16_t __result;                  \
    __asm__ __volatile__                \
    (                                   \
        "in __tmp_reg__, %2" "\n\t"     \
        "out %2, %C1"   "\n\t"          \
        "movw r30, %1"  "\n\t"          \
        "elpm %A0, Z+"  "\n\t"          \
        "elpm %B0, Z"   "\n\t"          \
        "out %2, __tmp_reg__"           \
        : "=r" (__result)               \
        : "r" (__addr32),               \
          "I" (_SFR_IO_ADDR(RAMPZ))     \
        : "r30", "r31"                  \
```

```
    );                                  \
    __result;                           \
}))

#define __ELPM_dword_classic__(addr)      \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint32_t __result;                    \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"          "\n\t"     \
        "mov r31, %B1"         "\n\t"     \
        "mov r30, %A1"         "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %A0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %B0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %C0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %D0, r0"          "\n\t"     \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r0", "r30", "r31"              \
    );                                    \
    __result;                             \
}))

#define __ELPM_dword_enhanced__(addr)     \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint32_t __result;                    \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

#define __ELPM_dword_xmega__(addr)        \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    uint32_t __result;                    \
    __asm__ __volatile__                  \
    (                                     \
        "in __tmp_reg__, %2" "\n\t"       \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        "out %2, __tmp_reg__"             \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

#define __ELPM_float_classic__(addr)      \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    float __result;                       \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"          "\n\t"     \
        "mov r31, %B1"         "\n\t"     \
        "mov r30, %A1"         "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %A0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %B0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %C0, r0"          "\n\t"     \
        "in r0, %2"            "\n\t"     \
        "adiw r30, 1"          "\n\t"     \
        "adc r0, __zero_reg__" "\n\t"     \
        "out %2, r0"           "\n\t"     \
        "elpm"                 "\n\t"     \
        "mov %D0, r0"          "\n\t"     \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r0", "r30", "r31"              \
    );                                    \
```

```
    __result;                                \
}))

#define __ELPM_float_enhanced__(addr)     \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    float __result;                       \
    __asm__ __volatile__                  \
    (                                     \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

#define __ELPM_float_xmega__(addr)        \
(__extension__({                          \
    uint32_t __addr32 = (uint32_t)(addr); \
    float __result;                       \
    __asm__ __volatile__                  \
    (                                     \
        "in __tmp_reg__, %2" "\n\t"       \
        "out %2, %C1"   "\n\t"            \
        "movw r30, %1"  "\n\t"            \
        "elpm %A0, Z+"  "\n\t"            \
        "elpm %B0, Z+"  "\n\t"            \
        "elpm %C0, Z+"  "\n\t"            \
        "elpm %D0, Z"   "\n\t"            \
        "out %2, __tmp_reg__"             \
        : "=r" (__result)                 \
        : "r" (__addr32),                 \
          "I" (_SFR_IO_ADDR(RAMPZ))       \
        : "r30", "r31"                    \
    );                                    \
    __result;                             \
}))

/*
Check for architectures that implement RAMPD (avrxmega5, avrxmega7)
as they need to save/restore RAMPZ for ELPM macros so it does
not interfere with data accesses.
*/
#if defined (__AVR_HAVE_RAMPD__)

#define __ELPM(addr)         __ELPM_xmega__(addr)
#define __ELPM_word(addr)    __ELPM_word_xmega__(addr)
#define __ELPM_dword(addr)   __ELPM_dword_xmega__(addr)
#define __ELPM_float(addr)   __ELPM_float_xmega__(addr)

#else

#if defined (__AVR_HAVE_LPMX__)

#define __ELPM(addr)         __ELPM_enhanced__(addr)
#define __ELPM_word(addr)    __ELPM_word_enhanced__(addr)
#define __ELPM_dword(addr)   __ELPM_dword_enhanced__(addr)
#define __ELPM_float(addr)   __ELPM_float_enhanced__(addr)

#else

#define __ELPM(addr)         __ELPM_classic__(addr)
#define __ELPM_word(addr)    __ELPM_word_classic__(addr)
#define __ELPM_dword(addr)   __ELPM_dword_classic__(addr)
#define __ELPM_float(addr)   __ELPM_float_classic__(addr)

#endif  /* __AVR_HAVE_LPMX__ */

#endif  /* __AVR_HAVE_RAMPD__ */

#endif    /* !__DOXYGEN__ */

/** \ingroup avr_pgmspace
    \def pgm_read_byte_far(address_long)
    Read a byte from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_byte_far(address_long)  __ELPM((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_word_far(address_long)
    Read a word from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_word_far(address_long)  __ELPM_word((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_dword_far(address_long)
    Read a double word from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_dword_far(address_long) __ELPM_dword((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_float_far(address_long)
    Read a float from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_float_far(address_long) __ELPM_float((uint32_t)(address_long))

/** \ingroup avr_pgmspace
    \def pgm_read_ptr_far(address_long)
```

```
    Read a pointer from the program space with a 32-bit (far) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_ptr_far(address_long) (void*)__ELPM_word((uint32_t)(address_long))

#endif /* RAMPZ or __DOXYGEN__ */

/** \ingroup avr_pgmspace
    \def pgm_read_byte(address_short)
    Read a byte from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_byte(address_short)    pgm_read_byte_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_word(address_short)
    Read a word from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_word(address_short)    pgm_read_word_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_dword(address_short)
    Read a double word from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_dword(address_short)   pgm_read_dword_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_float(address_short)
    Read a float from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_float(address_short)   pgm_read_float_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_read_ptr(address_short)
    Read a pointer from the program space with a 16-bit (near) address.

    \note The address is a byte address.
    The address is in the program space. */

#define pgm_read_ptr(address_short)     pgm_read_ptr_near(address_short)

/** \ingroup avr_pgmspace
    \def pgm_get_far_address(var)

   This macro facilitates the obtention of a 32 bit "far" pointer (only 24 bits
   used) to data even passed the 64KB limit for the 16 bit ordinary pointer. It
   is similar to the '&' operator, with some limitations.

   Comments:

   - The overhead is minimal and it's mainly due to the 32 bit size operation.

   - 24 bit sizes guarantees the code compatibility for use in future devices.

   - hh8() is an undocumented feature but seems to give the third significant byte
     of a 32 bit data and accepts symbols, complementing the functionality of hi8()
     and lo8(). There is not an equivalent assembler function to get the high
     significant byte.

   - 'var' has to be resolved at linking time as an existing symbol, i.e, a simple
     type variable name, an array name (not an indexed element of the array, if the
     index is a constant the compiler does not complain but fails to get the address
     if optimization is enabled), a struct name or a struct field name, a function
     identifier, a linker defined identifier,...

   - The returned value is the identifier's VMA (virtual memory address) determined
     by the linker and falls in the corresponding memory region. The AVR Harvard
     architecture requires non overlapping VMA areas for the multiple address spaces
     in the processor: Flash ROM, RAM, and EEPROM. Typical offset for this are
     0x00000000, 0x00800xx0, and 0x00810000 respectively, derived from the linker
     script used and linker options. The value returned can be seen then as a
     universal pointer.
*/

#define pgm_get_far_address(var)                        \
({                                                      \
        uint_farptr_t tmp;                              \
                                                        \
        __asm__ __volatile__(                           \
                                                        \
                        "ldi    %A0, lo8(%1)"           "\n\t"  \
                        "ldi    %B0, hi8(%1)"           "\n\t"  \
                        "ldi    %C0, hh8(%1)"           "\n\t"  \
                        "clr    %D0"                    "\n\t"  \
                :                                       \
                        "=d" (tmp)                              \
                :                                       \
                        "p"  (&(var))                           \
        );                                              \
        tmp;                                            \
})


/** \ingroup avr_pgmspace
    \fn const void * memchr_P(const void *s, int val, size_t len)
    \brief Scan flash memory for a character.

    The memchr_P() function scans the first \p len bytes of the flash
    memory area pointed to by \p s for the character \p val.  The first
    byte to match \p val (interpreted as an unsigned character) stops
    the operation.

    \return The memchr_P() function returns a pointer to the matching
```

```
    byte or \c NULL if the character does not occur in the given memory
    area. */
extern const void * memchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;

/** \ingroup avr_pgmspace
    \fn int memcmp_P(const void *s1, const void *s2, size_t len)
    \brief Compare memory areas

    The memcmp_P() function compares the first \p len bytes of the memory
    areas \p s1 and flash \p s2. The comparision is performed using unsigned
    char operations.

    \returns The memcmp_P() function returns an integer less than, equal
    to, or greater than zero if the first \p len bytes of \p s1 is found,
    respectively, to be less than, to match, or be greater than the first
    \p len bytes of \p s2.    */
extern int memcmp_P(const void *, const void *, size_t) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn void *memccpy_P (void *dest, const void *src, int val, size_t len)

    This function is similar to memccpy() except that \p src is pointer
    to a string in program space.      */
extern void *memccpy_P(void *, const void *, int __val, size_t);

/** \ingroup avr_pgmspace
    \fn void *memcpy_P(void *dest, const void *src, size_t n)

    The memcpy_P() function is similar to memcpy(), except the src string
    resides in program space.

    \returns The memcpy_P() function returns a pointer to dest.  */
extern void *memcpy_P(void *, const void *, size_t);

/** \ingroup avr_pgmspace
    \fn void *memmem_P(const void *s1, size_t len1, const void *s2, size_t len2)

    The memmem_P() function is similar to memmem() except that \p s2 is
    pointer to a string in program space.       */
extern void *memmem_P(const void *, size_t, const void *, size_t) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn const void +memrchr_P(const void *src, int val, size_t len)

    The memrchr_P() function is like the memchr_P() function, except
    that it searches backwards from the end of the \p len bytes pointed
    to by \p src instead of forwards from the front. (Glibc, GNU extension.)

    \return The memrchr_P() function returns a pointer to the matching
    byte or \c NULL if the character does not occur in the given memory
    area. */
extern const void * memrchr_P(const void *, int __val, size_t __len) __ATTR_CONST__;

/** \ingroup avr_pgmspace
    \fn char *strcat_P(char *dest, const char *src)

    The strcat_P() function is similar to strcat() except that the \e src
    string must be located in program space (flash).

    \returns The strcat() function returns a pointer to the resulting string
    \e dest. */
extern char *strcat_P(char *, const char *);

/** \ingroup avr_pgmspace
    \fn const char *strchr_P(const char *s, int val)
    \brief Locate character in program space string.

    The strchr_P() function locates the first occurrence of \p val
    (converted to a char) in the string pointed to by \p s in program
    space. The terminating null character is considered to be part of
    the string.

    The strchr_P() function is similar to strchr() except that \p s is
    pointer to a string in program space.

    \returns The strchr_P() function returns a pointer to the matched
    character or \c NULL if the character is not found. */
extern const char * strchr_P(const char *, int __val) __ATTR_CONST__;

/** \ingroup avr_pgmspace
    \fn const char *strchrnul_P(const char *s, int c)

    The strchrnul_P() function is like strchr_P() except that if \p c is
    not found in \p s, then it returns a pointer to the null byte at the
    end of \p s, rather than \c NULL. (Glibc, GNU extension.)

    \return The strchrnul_P() function returns a pointer to the matched
    character, or a pointer to the null byte at the end of \p s (i.e.,
    \c s+strlen(s)) if the character is not found.        */
extern const char * strchrnul_P(const char *, int __val) __ATTR_CONST__;

/** \ingroup avr_pgmspace
    \fn int strcmp_P(const char *s1, const char *s2)

    The strcmp_P() function is similar to strcmp() except that \p s2 is
    pointer to a string in program space.

    \returns The strcmp_P() function returns an integer less than, equal
    to, or greater than zero if \p s1 is found, respectively, to be less
    than, to match, or be greater than \p s2. A consequence of the
    ordering used by strcmp_P() is that if \p s1 is an initial substring
    of \p s2, then \p s1 is considered to be "less than" \p s2. */
extern int strcmp_P(const char *, const char *) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn char *strcpy_P(char *dest, const char *src)

    The strcpy_P() function is similar to strcpy() except that src is a
    pointer to a string in program space.

    \returns The strcpy_P() function returns a pointer to the destination
    string dest. */
extern char *strcpy_P(char *, const char *);

/** \ingroup avr_pgmspace
    \fn int strcasecmp_P(const char *s1, const char *s2)
    \brief Compare two strings ignoring case.
```

```
    The strcasecmp_P() function compares the two strings \p s1 and \p s2,
    ignoring the case of the characters.

    \param s1 A pointer to a string in the devices SRAM.
    \param s2 A pointer to a string in the devices Flash.

    \returns The strcasecmp_P() function returns an integer less than,
    equal to, or greater than zero if \p s1 is found, respectively, to
    be less than, to match, or be greater than \p s2. A consequence of
    the ordering used by strcasecmp_P() is that if \p s1 is an initial
    substring of \p s2, then \p s1 is considered to be "less than" \p s2. */
extern int strcasecmp_P(const char *, const char *) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn char *strcasestr_P(const char *s1, const char *s2)

    This funtion is similar to strcasestr() except that \p s2 is pointer
    to a string in program space.      */
extern char *strcasestr_P(const char *, const char *) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn size_t strcspn_P(const char *s, const char *reject)

    The strcspn_P() function calculates the length of the initial segment
    of \p s which consists entirely of characters not in \p reject. This
    function is similar to strcspn() except that \p reject is a pointer
    to a string in program space.

    \return The strcspn_P() function returns the number of characters in
    the initial segment of \p s which are not in the string \p reject.
    The terminating zero is not considered as a part of string.      */
extern size_t strcspn_P(const char *__s, const char * __reject) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn size_t strlcat_P(char *dst, const char *src, size_t siz)
    \brief Concatenate two strings.

    The strlcat_P() function is similar to strlcat(), except that the \p src
    string must be located in program space (flash).

    Appends \p src to string \p dst of size \p siz (unlike strncat(),
    \p siz is the full size of \p dst, not space left).  At most \p siz-1
    characters will be copied.  Always NULL terminates (unless \p siz <=
    \p strlen(dst)).

    \returns The strlcat_P() function returns strlen(src) + MIN(siz,
    strlen(initial dst)).  If retval >= siz, truncation occurred.    */
extern size_t strlcat_P (char *, const char *, size_t );

/** \ingroup avr_pgmspace
    \fn size_t strlcpy_P(char *dst, const char *src, size_t siz)
    \brief Copy a string from progmem to RAM.

    Copy \p src to string \p dst of size \p siz.  At most \p siz-1
    characters will be copied.  Always NULL terminates (unless \p siz == 0).
    The strlcpy_P() function is similar to strlcpy() except that the
    \p src is pointer to a string in memory space.

    \returns The strlcpy_P() function returns strlen(src). If
    retval >= siz, truncation occurred.  */
extern size_t strlcpy_P (char *, const char *, size_t );

/** \ingroup avr_pgmspace
    \fn size_t strnlen_P(const char *src, size_t len)
    \brief Determine the length of a fixed-size string.

    The strnlen_P() function is similar to strnlen(), except that \c src is a
    pointer to a string in program space.

    \returns The strnlen_P function returns strlen_P(src), if that is less than
    \c len, or \c len if there is no '\\0' character among the first \c len
    characters pointed to by \c src. */
extern size_t strnlen_P(const char *, size_t) __ATTR_CONST__; /* program memory can't change */

/** \ingroup avr_pgmspace
    \fn int strncmp_P(const char *s1, const char *s2, size_t n)

    The strncmp_P() function is similar to strcmp_P() except it only compares
    the first (at most) n characters of s1 and s2.

    \returns The strncmp_P() function returns an integer less than, equal to,
    or greater than zero if s1 (or the first n bytes thereof) is found,
    respectively, to be less than, to match, or be greater than s2.  */
extern int strncmp_P(const char *, const char *, size_t) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn int strncasecmp_P(const char *s1, const char *s2, size_t n)
    \brief Compare two strings ignoring case.

    The strncasecmp_P() function is similar to strcasecmp_P(), except it
    only compares the first \p n characters of \p s1.

    \param s1 A pointer to a string in the devices SRAM.
    \param s2 A pointer to a string in the devices Flash.
    \param n The maximum number of bytes to compare.

    \returns The strncasecmp_P() function returns an integer less than,
    equal to, or greater than zero if \p s1 (or the first \p n bytes
    thereof) is found, respectively, to be less than, to match, or be
    greater than \p s2. A consequence of the ordering used by
    strncasecmp_P() is that if \p s1 is an initial substring of \p s2,
    then \p s1 is considered to be "less than" \p s2.  */
extern int strncasecmp_P(const char *, const char *, size_t) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn char *strncat_P(char *dest, const char *src, size_t len)
    \brief Concatenate two strings.

    The strncat_P() function is similar to strncat(), except that the \e src
    string must be located in program space (flash).

    \returns The strncat_P() function returns a pointer to the resulting string
    dest.  */
extern char *strncat_P(char *, const char *, size_t);

/** \ingroup avr_pgmspace
```

```
    \fn char *strncpy_P(char *dest, const char *src, size_t n)

    The strncpy_P() function is similar to strcpy_P() except that not more
    than n bytes of src are copied.  Thus, if there is no null byte among the
    first n bytes of src, the result will not be null-terminated.

    In the case where the length of src is less than that of n, the remainder
    of dest will be padded with nulls.

    \returns The strncpy_P() function returns a pointer to the destination
    string dest.  */
extern char *strncpy_P(char *, const char *, size_t);

/** \ingroup avr_pgmspace
    \fn char *strpbrk_P(const char *s, const char *accept)

    The strpbrk_P() function locates the first occurrence in the string
    \p s of any of the characters in the flash string \p accept. This
    function is similar to strpbrk() except that \p accept is a pointer
    to a string in program space.

    \return  The strpbrk_P() function returns a pointer to the character
    in \p s that matches one of the characters in \p accept, or \c NULL
    if no such character is found. The terminating zero is not considered
    as a part of string: if one or both args are empty, the result will
    \c NULL. */
extern char *strpbrk_P(const char *__s, const char * __accept) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn const char *strrchr_P(const char *s, int val)
    \brief Locate character in string.

    The strrchr_P() function returns a pointer to the last occurrence of
    the character \p val in the flash string \p s.

    \return The strrchr_P() function returns a pointer to the matched
    character or \c NULL if the character is not found. */
extern const char * strrchr_P(const char *, int __val) __ATTR_CONST__;

/** \ingroup avr_pgmspace
    \fn char *strsep_P(char **sp, const char *delim)
    \brief Parse a string into tokens.

    The strsep_P() function locates, in the string referenced by \p *sp,
    the first occurrence of any character in the string \p delim (or the
    terminating '\\0' character) and replaces it with a '\\0'.  The
    location of the next character after the delimiter character (or \c
    NULL, if the end of the string was reached) is stored in \p *sp. An
    ``empty'' field, i.e. one caused by two adjacent delimiter
    characters, can be detected by comparing the location referenced by
    the pointer returned in \p *sp to '\\0'. This function is similar to
    strsep() except that \p delim is a pointer to a string in program
    space.

    \return The strsep_P() function returns a pointer to the original
    value of \p *sp. If \p *sp is initially \c NULL, strsep_P() returns
    \c NULL. */
extern char *strsep_P(char **__sp, const char * __delim);

/** \ingroup avr_pgmspace
    \fn size_t strspn_P(const char *s, const char *accept)

    The strspn_P() function calculates the length of the initial segment
    of \p s which consists entirely of characters in \p accept. This
    function is similar to strspn() except that \p accept is a pointer
    to a string in program space.

    \return  The strspn_P() function returns the number of characters in
    the initial segment of \p s which consist only of characters from \p
    accept. The terminating zero is not considered as a part of string.       */
extern size_t strspn_P(const char *__s, const char * __accept) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn char *strstr_P(const char *s1, const char *s2)
    \brief Locate a substring.

    The strstr_P() function finds the first occurrence of the substring
    \p s2 in the string \p s1.  The terminating '\\0' characters are not
    compared. The strstr_P() function is similar to strstr() except that
    \p s2 is pointer to a string in program space.

    \returns The strstr_P() function returns a pointer to the beginning
    of the substring, or NULL if the substring is not found. If \p s2
    points to a string of zero length, the function returns \p s1. */
extern char *strstr_P(const char *, const char *) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn char *strtok_P(char *s, const char * delim)
    \brief Parses the string into tokens.

    strtok_P() parses the string \p s into tokens. The first call to
    strtok_P() should have \p s as its first argument. Subsequent calls
    should have the first argument set to NULL. If a token ends with a
    delimiter, this delimiting character is overwritten with a '\\0' and a
    pointer to the next character is saved for the next call to strtok_P().
    The delimiter string \p delim may be different for each call.

    The strtok_P() function is similar to strtok() except that \p delim
    is pointer to a string in program space.

    \returns The strtok_P() function returns a pointer to the next token or
    NULL when no more tokens are found.

    \note strtok_P() is NOT reentrant. For a reentrant version of this
    function see strtok_rP().
 */
extern char *strtok_P(char *__s, const char * __delim);

/** \ingroup avr_pgmspace
    \fn char *strtok_rP (char *string, const char *delim, char **last)
    \brief Parses string into tokens.

    The strtok_rP() function parses \p string into tokens. The first call to
    strtok_rP() should have string as its first argument. Subsequent calls
    should have the first argument set to NULL. If a token ends with a
    delimiter, this delimiting character is overwritten with a '\\0' and a
    pointer to the next character is saved for the next call to strtok_rP().
```

```
    The delimiter string \p delim may be different for each call. \p last is
    a user allocated char* pointer. It must be the same while parsing the
    same string. strtok_rP() is a reentrant version of strtok_P().

    The strtok_rP() function is similar to strtok_r() except that \p delim
    is pointer to a string in program space.

    \returns The strtok_rP() function returns a pointer to the next token or
    NULL when no more tokens are found. */
extern char *strtok_rP(char *__s, const char * __delim, char **__last);

/** \ingroup avr_pgmspace
    \fn size_t strlen_PF(uint_farptr_t s)
    \brief Obtain the length of a string

    The strlen_PF() function is similar to strlen(), except that \e s is a
    far pointer to a string in program space.

    \param s A far pointer to the string in flash

    \returns The strlen_PF() function returns the number of characters in
    \e s. The contents of RAMPZ SFR are undefined when the function returns. */
extern size_t strlen_PF(uint_farptr_t src) __ATTR_CONST__; /* program memory can't change */

/** \ingroup avr_pgmspace
    \fn size_t strnlen_PF(uint_farptr_t s, size_t len)
    \brief Determine the length of a fixed-size string

    The strnlen_PF() function is similar to strnlen(), except that \e s is a
    far pointer to a string in program space.

    \param s A far pointer to the string in Flash
    \param len The maximum number of length to return

    \returns The strnlen_PF function returns strlen_P(\e s), if that is less
    than \e len, or \e len if there is no '\\0' character among the first \e
    len characters pointed to by \e s. The contents of RAMPZ SFR are
    undefined when the function returns. */
extern size_t strnlen_PF(uint_farptr_t src, size_t len) __ATTR_CONST__; /* program memory can't change */

/** \ingroup avr_pgmspace
    \fn void *memcpy_PF(void *dest, uint_farptr_t src, size_t n)
    \brief Copy a memory block from flash to SRAM

    The memcpy_PF() function is similar to memcpy(), except the data
    is copied from the program space and is addressed using a far pointer.

    \param dest A pointer to the destination buffer
    \param src A far pointer to the origin of data in flash memory
    \param n The number of bytes to be copied

    \returns The memcpy_PF() function returns a pointer to \e dst. The contents
    of RAMPZ SFR are undefined when the function returns. */
extern void *memcpy_PF(void *dest, uint_farptr_t src, size_t len);

/** \ingroup avr_pgmspace
    \fn char *strcpy_PF(char *dst, uint_farptr_t src)
    \brief Duplicate a string

    The strcpy_PF() function is similar to strcpy() except that \e src is a far
    pointer to a string in program space.

    \param dst A pointer to the destination string in SRAM
    \param src A far pointer to the source string in Flash

    \returns The strcpy_PF() function returns a pointer to the destination
    string \e dst. The contents of RAMPZ SFR are undefined when the funcion
    returns. */
extern char *strcpy_PF(char *dest, uint_farptr_t src);

/** \ingroup avr_pgmspace
    \fn char *strncpy_PF(char *dst, uint_farptr_t src, size_t n)
    \brief Duplicate a string until a limited length

    The strncpy_PF() function is similar to strcpy_PF() except that not more
    than \e n bytes of \e src are copied.  Thus, if there is no null byte among
    the first \e n bytes of \e src, the result will not be null-terminated.

    In the case where the length of \e src is less than that of \e n, the
    remainder of \e dst will be padded with nulls.

    \param dst A pointer to the destination string in SRAM
    \param src A far pointer to the source string in Flash
    \param n The maximum number of bytes to copy

    \returns The strncpy_PF() function returns a pointer to the destination
    string \e dst. The contents of RAMPZ SFR are undefined when the function
    returns. */
extern char *strncpy_PF(char *dest, uint_farptr_t src, size_t len);

/** \ingroup avr_pgmspace
    \fn char *strcat_PF(char *dst, uint_farptr_t src)
    \brief Concatenates two strings

    The strcat_PF() function is similar to strcat() except that the \e src
    string must be located in program space (flash) and is addressed using
    a far pointer

    \param dst A pointer to the destination string in SRAM
    \param src A far pointer to the string to be appended in Flash

    \returns The strcat_PF() function returns a pointer to the resulting
    string \e dst. The contents of RAMPZ SFR are undefined when the function
    returns */
extern char *strcat_PF(char *dest, uint_farptr_t src);

/** \ingroup avr_pgmspace
    \fn size_t strlcat_PF(char *dst, uint_farptr_t src, size_t n)
    \brief Concatenate two strings

    The strlcat_PF() function is similar to strlcat(), except that the \e src
    string must be located in program space (flash) and is addressed using
    a far pointer.

    Appends src to string dst of size \e n (unlike strncat(), \e n is the
    full size of \e dst, not space left).  At most \e n-1 characters
    will be copied.  Always NULL terminates (unless \e n <= strlen(\e dst)).
```

```
    \param dst A pointer to the destination string in SRAM
    \param src A far pointer to the source string in Flash
    \param n The total number of bytes allocated to the destination string

    \returns The strlcat_PF() function returns strlen(\e src) + MIN(\e n,
    strlen(initial \e dst)).  If retval >= \e n, truncation occurred. The
    contents of RAMPZ SFR are undefined when the funcion returns. */
extern size_t strlcat_PF(char *dst, uint_farptr_t src, size_t siz);

/** \ingroup avr_pgmspace
    \fn char *strncat_PF(char *dst, uint_farptr_t src, size_t n)
    \brief Concatenate two strings

    The strncat_PF() function is similar to strncat(), except that the \e src
    string must be located in program space (flash) and is addressed using a
    far pointer.

    \param dst A pointer to the destination string in SRAM
    \param src A far pointer to the source string in Flash
    \param n The maximum number of bytes to append

    \returns The strncat_PF() function returns a pointer to the resulting
    string \e dst. The contents of RAMPZ SFR are undefined when the function
    returns. */
extern char *strncat_PF(char *dest, uint_farptr_t src, size_t len);

/** \ingroup avr_pgmspace
    \fn int strcmp_PF(const char *s1, uint_farptr_t s2)
    \brief Compares two strings

    The strcmp_PF() function is similar to strcmp() except that \e s2 is a far
    pointer to a string in program space.

    \param s1 A pointer to the first string in SRAM
    \param s2 A far pointer to the second string in Flash

    \returns The strcmp_PF() function returns an integer less than, equal to,
    or greater than zero if \e s1 is found, respectively, to be less than, to
    match, or be greater than \e s2. The contents of RAMPZ SFR are undefined
    when the function returns. */
extern int strcmp_PF(const char *s1, uint_farptr_t s2) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn int strncmp_PF(const char *s1, uint_farptr_t s2, size_t n)
    \brief Compare two strings with limited length

    The strncmp_PF() function is similar to strcmp_PF() except it only
    compares the first (at most) \e n characters of \e s1 and \e s2.

    \param s1 A pointer to the first string in SRAM
    \param s2 A far pointer to the second string in Flash
    \param n The maximum number of bytes to compare

    \returns The strncmp_PF() function returns an integer less than, equal
    to, or greater than zero if \e s1 (or the first \e n bytes thereof) is found,
    respectively, to be less than, to match, or be greater than \e s2. The
    contents of RAMPZ SFR are undefined when the function returns. */
extern int strncmp_PF(const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn int strcasecmp_PF(const char *s1, uint_farptr_t s2)
    \brief Compare two strings ignoring case

    The strcasecmp_PF() function compares the two strings \e s1 and \e s2, ignoring
    the case of the characters.

    \param s1 A pointer to the first string in SRAM
    \param s2 A far pointer to the second string in Flash

    \returns The strcasecmp_PF() function returns an integer less than, equal
    to, or greater than zero if \e s1 is found, respectively, to be less than, to
    match, or be greater than \e s2. The contents of RAMPZ SFR are undefined
    when the function returns. */
extern int strcasecmp_PF(const char *s1, uint_farptr_t s2) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn int strncasecmp_PF(const char *s1, uint_farptr_t s2, size_t n)
    \brief Compare two strings ignoring case

    The strncasecmp_PF() function is similar to strcasecmp_PF(), except it
    only compares the first \e n characters of \e s1 and the string in flash is
    addressed using a far pointer.

    \param s1 A pointer to a string in SRAM
    \param s2 A far pointer to a string in Flash
    \param n The maximum number of bytes to compare

    \returns The strncasecmp_PF() function returns an integer less than, equal
    to, or greater than zero if \e s1 (or the first \e n bytes thereof) is found,
    respectively, to be less than, to match, or be greater than \e s2. The
    contents of RAMPZ SFR are undefined when the function returns.  */
extern int strncasecmp_PF(const char *s1, uint_farptr_t s2, size_t n) __ATTR_PURE__;

/** \ingroup avr_pgmspace
    \fn char *strstr_PF(const char *s1, uint_farptr_t s2)
    \brief Locate a substring.

    The strstr_PF() function finds the first occurrence of the substring \c s2
    in the string \c s1.  The terminating '\\0' characters are not
    compared.
    The strstr_PF() function is similar to strstr() except that \c s2 is a
    far pointer to a string in program space.

    \returns The strstr_PF() function returns a pointer to the beginning of the
    substring, or NULL if the substring is not found.
    If \c s2 points to a string of zero length, the function returns \c s1. The
    contents of RAMPZ SFR are undefined when the function returns. */
extern char *strstr_PF(const char *s1, uint_farptr_t s2);

/** \ingroup avr_pgmspace
    \fn size_t strlcpy_PF(char *dst, uint_farptr_t src, size_t siz)
    \brief Copy a string from progmem to RAM.

    Copy src to string dst of size siz.  At most siz-1 characters will be
    copied. Always NULL terminates (unless siz == 0).
```

```
    \returns The strlcpy_PF() function returns strlen(src). If retval >= siz,
    truncation occurred.  The contents of RAMPZ SFR are undefined when the
    function returns. */
extern size_t strlcpy_PF(char *dst, uint_farptr_t src, size_t siz);

/** \ingroup avr_pgmspace
    \fn int memcmp_PF(const void *s1, uint_farptr_t s2, size_t len)
    \brief Compare memory areas

    The memcmp_PF() function compares the first \p len bytes of the memory
    areas \p s1 and flash \p s2. The comparision is performed using unsigned
    char operations. It is an equivalent of memcmp_P() function, except
    that it is capable working on all FLASH including the exteded area
    above 64kB.

    \returns The memcmp_PF() function returns an integer less than, equal
    to, or greater than zero if the first \p len bytes of \p s1 is found,
    respectively, to be less than, to match, or be greater than the first
    \p len bytes of \p s2.  */
extern int memcmp_PF(const void *, uint_farptr_t, size_t) __ATTR_PURE__;

#ifdef __DOXYGEN__
/** \ingroup avr_pgmspace
    \fn size_t strlen_P(const char *src)

    The strlen_P() function is similar to strlen(), except that src is a
    pointer to a string in program space.

    \returns The strlen_P() function returns the number of characters in src.

    \note strlen_P() is implemented as an inline function in the avr/pgmspace.h
    header file, which will check if the length of the string is a constant
    and known at compile time. If it is not known at compile time, the macro
    will issue a call to __strlen_P() which will then calculate the length
    of the string as normal.
*/
static inline size_t strlen_P(const char * s);
#else
extern size_t __strlen_P(const char *) __ATTR_CONST__;  /* internal helper function */
__attribute__((__always_inline__)) static __inline__ size_t strlen_P(const char * s);
static __inline__ size_t strlen_P(const char *s) {
  return __builtin_constant_p(__builtin_strlen(s))
     ? __builtin_strlen(s) : __strlen_P(s);
}
#endif

#ifdef __cplusplus
}
#endif

#endif /* __PGMSPACE_H_ */
```

***** io.h

```
/* $Id$ */

/** \file */
/** \defgroup avr_io <avr/io.h>: AVR device-specific IO definitions
    \code #include <avr/io.h> \endcode

    This header file includes the apropriate IO definitions for the
    device that has been specified by the <tt>-mmcu=</tt> compiler
    command-line switch.  This is done by diverting to the appropriate
    file <tt>&lt;avr/io</tt><em>XXXX</em><tt>.h&gt;</tt> which should
    never be included directly.  Some register names common to all
    AVR devices are defined directly within <tt>&lt;avr/common.h&gt;</tt>,
    which is included in <tt>&lt;avr/io.h&gt;</tt>,
    but most of the details come from the respective include file.

    Note that this file always includes the following files:
    \code
    #include <avr/sfr_defs.h>
    #include <avr/portpins.h>
    #include <avr/common.h>
    #include <avr/version.h>
    \endcode
    See \ref avr_sfr for more details about that header file.

    Included are definitions of the IO register set and their
    respective bit values as specified in the Atmel documentation.
    Note that inconsistencies in naming conventions,
    so even identical functions sometimes get different names on
    different devices.

    Also included are the specific names useable for interrupt
    function definitions as documented
    \ref avr_signames "here".

    Finally, the following macros are defined:

    - \b RAMEND
    <br>
    The last on-chip RAM address.
    <br>
    - \b XRAMEND
    <br>
    The last possible RAM location that is addressable. This is equal to
    RAMEND for devices that do not allow for external RAM. For devices
    that allow external RAM, this will be larger than RAMEND.
    <br>
    - \b E2END
    <br>
    The last EEPROM address.
    <br>
    - \b FLASHEND
    <br>
    The last byte address in the Flash program space.
    <br>
    - \b SPM_PAGESIZE
    <br>
    For devices with bootloader support, the flash pagesize
    (in bytes) to be used for the \c SPM instruction.
    - \b E2PAGESIZE
    <br>
    The size of the EEPROM page.
```

```
*/

#ifndef _AVR_IO_H_
#define _AVR_IO_H_

#include <avr/sfr_defs.h>

#if defined (__AVR_AT94K__)
#  include <avr/ioat94k.h>
#elif defined (__AVR_AT43USB320__)
#  include <avr/io43u32x.h>
#elif defined (__AVR_AT43USB355__)
#  include <avr/io43u35x.h>
#elif defined (__AVR_AT76C711__)
#  include <avr/io76c711.h>
#elif defined (__AVR_AT86RF401__)
#  include <avr/io86r401.h>
#elif defined (__AVR_AT90PWM1__)
#  include <avr/io90pwm1.h>
#elif defined (__AVR_AT90PWM2__)
#  include <avr/io90pwmx.h>
#elif defined (__AVR_AT90PWM2B__)
#  include <avr/io90pwm2b.h>
#elif defined (__AVR_AT90PWM3__)
#  include <avr/io90pwmx.h>
#elif defined (__AVR_AT90PWM3B__)
#  include <avr/io90pwm3b.h>
#elif defined (__AVR_AT90PWM216__)
#  include <avr/io90pwm216.h>
#elif defined (__AVR_AT90PWM316__)
#  include <avr/io90pwm316.h>
#elif defined (__AVR_AT90PWM161__)
#  include <avr/io90pwm161.h>
#elif defined (__AVR_AT90PWM81__)
#  include <avr/io90pwm81.h>
#elif defined (__AVR_ATmega8U2__)
#  include <avr/iom8u2.h>
#elif defined (__AVR_ATmega16M1__)
#  include <avr/iom16m1.h>
#elif defined (__AVR_ATmega16U2__)
#  include <avr/iom16u2.h>
#elif defined (__AVR_ATmega16U4__)
#  include <avr/iom16u4.h>
#elif defined (__AVR_ATmega32C1__)
#  include <avr/iom32c1.h>
#elif defined (__AVR_ATmega32M1__)
#  include <avr/iom32m1.h>
#elif defined (__AVR_ATmega32U2__)
#  include <avr/iom32u2.h>
#elif defined (__AVR_ATmega32U4__)
#  include <avr/iom32u4.h>
#elif defined (__AVR_ATmega32U6__)
#  include <avr/iom32u6.h>
#elif defined (__AVR_ATmega64C1__)
#  include <avr/iom64c1.h>
#elif defined (__AVR_ATmega64M1__)
#  include <avr/iom64m1.h>
#elif defined (__AVR_ATmega128__)
#  include <avr/iom128.h>
#elif defined (__AVR_ATmega128A__)
#  include <avr/iom128a.h>
#elif defined (__AVR_ATmega1280__)
#  include <avr/iom1280.h>
#elif defined (__AVR_ATmega1281__)
#  include <avr/iom1281.h>
#elif defined (__AVR_ATmega1284__)
#  include <avr/iom1284.h>
#elif defined (__AVR_ATmega1284P__)
#  include <avr/iom1284p.h>
#elif defined (__AVR_ATmega128RFA1__)
#  include <avr/iom128rfa1.h>
#elif defined (__AVR_ATmega1284RFR2__)
#  include <avr/iom1284rfr2.h>
#elif defined (__AVR_ATmega128RFR2__)
#  include <avr/iom128rfr2.h>
#elif defined (__AVR_ATmega2564RFR2__)
#  include <avr/iom2564rfr2.h>
#elif defined (__AVR_ATmega256RFR2__)
#  include <avr/iom256rfr2.h>
#elif defined (__AVR_ATmega2560__)
#  include <avr/iom2560.h>
#elif defined (__AVR_ATmega2561__)
#  include <avr/iom2561.h>
#elif defined (__AVR_AT90CAN32__)
#  include <avr/iocan32.h>
#elif defined (__AVR_AT90CAN64__)
#  include <avr/iocan64.h>
#elif defined (__AVR_AT90CAN128__)
#  include <avr/iocan128.h>
#elif defined (__AVR_AT90USB82__)
#  include <avr/iousb82.h>
#elif defined (__AVR_AT90USB162__)
#  include <avr/iousb162.h>
#elif defined (__AVR_AT90USB646__)
#  include <avr/iousb646.h>
#elif defined (__AVR_AT90USB647__)
#  include <avr/iousb647.h>
#elif defined (__AVR_AT90USB1286__)
#  include <avr/iousb1286.h>
#elif defined (__AVR_AT90USB1287__)
#  include <avr/iousb1287.h>
#elif defined (__AVR_ATmega644RFR2__)
#  include <avr/iom644rfr2.h>
#elif defined (__AVR_ATmega64RFR2__)
#  include <avr/iom64rfr2.h>
#elif defined (__AVR_ATmega64__)
#  include <avr/iom64.h>
#elif defined (__AVR_ATmega64A__)
#  include <avr/iom64a.h>
#elif defined (__AVR_ATmega640__)
#  include <avr/iom640.h>
#elif defined (__AVR_ATmega644__)
#  include <avr/iom644.h>
#elif defined (__AVR_ATmega644A__)
#  include <avr/iom644a.h>
#elif defined (__AVR_ATmega644P__)
#  include <avr/iom644p.h>
#elif defined (__AVR_ATmega644PA__)
#  include <avr/iom644pa.h>
#elif defined (__AVR_ATmega645__)
#  include <avr/iom645.h>
#elif (defined __AVR_ATmega645A__)
#include <avr/iom645a.h>
#elif (defined __AVR_ATmega645P__)
#include <avr/iom645p.h>
#elif defined (__AVR_ATmega6450__)
#  include <avr/iom6450.h>
#elif (defined __AVR_ATmega6450A__)
#include <avr/iom6450a.h>
#elif (defined __AVR_ATmega6450P__)
#include <avr/iom6450p.h>
#elif defined (__AVR_ATmega649__)
#  include <avr/iom649.h>
#elif (defined __AVR_ATmega649A__)
#include <avr/iom649a.h>
#elif defined (__AVR_ATmega6490__)
#  include <avr/iom6490.h>
#elif (defined __AVR_ATmega6490A__)
#include <avr/iom6490a.h>
```

```
#elif (defined __AVR_ATmega6490P__)
#include <avr/iom6490p.h>
#elif defined (__AVR_ATmega649P__)
#  include <avr/iom649p.h>
#elif defined (__AVR_ATmega64HVE__)
#  include <avr/iom64hve.h>
#elif defined (__AVR_ATmega64HVE2__)
#  include <avr/iom64hve2.h>
#elif defined (__AVR_ATmega103__)
#  include <avr/iom103.h>
#elif defined (__AVR_ATmega32__)
#  include <avr/iom32.h>
#elif defined (__AVR_ATmega32A__)
#  include <avr/iom32a.h>
#elif defined (__AVR_ATmega323__)
#  include <avr/iom323.h>
#elif defined (__AVR_ATmega324P__)
#  include <avr/iom324p.h>
#elif (defined __AVR_ATmega324A__)
#include <avr/iom324a.h>
#elif defined (__AVR_ATmega324PA__)
#  include <avr/iom324pa.h>
#elif defined (__AVR_ATmega325__)
#  include <avr/iom325.h>
#elif (defined __AVR_ATmega325A__)
#include <avr/iom325a.h>
#elif defined (__AVR_ATmega325P__)
#  include <avr/iom325p.h>
#elif defined (__AVR_ATmega325PA__)
#  include <avr/iom325pa.h>
#elif defined (__AVR_ATmega3250__)
#  include <avr/iom3250.h>
#elif (defined __AVR_ATmega3250A__)
#include <avr/iom3250a.h>
#elif defined (__AVR_ATmega3250P__)
#  include <avr/iom3250p.h>
#elif defined (__AVR_ATmega3250PA__)
#  include <avr/iom3250pa.h>
#elif defined (__AVR_ATmega328P__)
#  include <avr/iom328p.h>
#elif (defined __AVR_ATmega328__)
#include <avr/iom328.h>
#elif defined (__AVR_ATmega329__)
#  include <avr/iom329.h>
#elif (defined __AVR_ATmega329A__)
#include <avr/iom329a.h>
#elif defined (__AVR_ATmega329P__)
#  include <avr/iom329p.h>
#elif (defined __AVR_ATmega329PA__)
#include <avr/iom329pa.h>
#elif (defined __AVR_ATmega3290PA__)
#include <avr/iom3290pa.h>
#elif defined (__AVR_ATmega3290__)
#  include <avr/iom3290.h>
#elif (defined __AVR_ATmega3290A__)
#include <avr/iom3290a.h>
#elif defined (__AVR_ATmega3290P__)
#  include <avr/iom3290.h>
#elif defined (__AVR_ATmega32HVB__)
#  include <avr/iom32hvb.h>
#elif defined (__AVR_ATmega32HVBREVB__)
#  include <avr/iom32hvbrevb.h>
#elif defined (__AVR_ATmega406__)
#  include <avr/iom406.h>
#elif defined (__AVR_ATmega16__)
#  include <avr/iom16.h>
#elif defined (__AVR_ATmega16A__)
#  include <avr/iom16a.h>
#elif defined (__AVR_ATmega161__)
#  include <avr/iom161.h>
#elif defined (__AVR_ATmega162__)
#  include <avr/iom162.h>
#elif defined (__AVR_ATmega163__)
#  include <avr/iom163.h>
#elif defined (__AVR_ATmega164P__)
#  include <avr/iom164p.h>
#elif (defined __AVR_ATmega164A__)
#include <avr/iom164a.h>
#elif defined (__AVR_ATmega164PA__)
#  include <avr/iom164pa.h>
#elif defined (__AVR_ATmega165__)
#  include <avr/iom165.h>
#elif defined (__AVR_ATmega165A__)
#  include <avr/iom165a.h>
#elif defined (__AVR_ATmega165P__)
#  include <avr/iom165p.h>
#elif defined (__AVR_ATmega165PA__)
#  include <avr/iom165pa.h>
#elif defined (__AVR_ATmega168__)
#  include <avr/iom168.h>
#elif defined (__AVR_ATmega168A__)
#  include <avr/iom168a.h>
#elif defined (__AVR_ATmega168P__)
#  include <avr/iom168p.h>
#elif defined (__AVR_ATmega168PA__)
#  include <avr/iom168pa.h>
#elif defined (__AVR_ATmega168PB__)
#  include <avr/iom168pb.h>
#elif defined (__AVR_ATmega169__)
#  include <avr/iom169.h>
#elif (defined __AVR_ATmega169A__)
#include <avr/iom169a.h>
#elif defined (__AVR_ATmega169P__)
#  include <avr/iom169p.h>
#elif defined (__AVR_ATmega169PA__)
#  include <avr/iom169pa.h>
#elif defined (__AVR_ATmega8HVA__)
#  include <avr/iom8hva.h>
#elif defined (__AVR_ATmega16HVA__)
#  include <avr/iom16hva.h>
#elif defined (__AVR_ATmega16HVA2__)
#  include <avr/iom16hva2.h>
#elif defined (__AVR_ATmega16HVB__)
#  include <avr/iom16hvb.h>
#elif defined (__AVR_ATmega16HVBREVB__)
#  include <avr/iom16hvbrevb.h>
#elif defined (__AVR_ATmega8__)
#  include <avr/iom8.h>
#elif defined (__AVR_ATmega8A__)
#  include <avr/iom8a.h>
#elif defined (__AVR_ATmega48__)
#  include <avr/iom48.h>
#elif defined (__AVR_ATmega48A__)
#  include <avr/iom48a.h>
#elif defined (__AVR_ATmega48PA__)
#  include <avr/iom48pa.h>
#elif defined (__AVR_ATmega48PB__)
#  include <avr/iom48pb.h>
#elif defined (__AVR_ATmega48P__)
#  include <avr/iom48p.h>
#elif defined (__AVR_ATmega88__)
#  include <avr/iom88.h>
#elif defined (__AVR_ATmega88A__)
#  include <avr/iom88a.h>
#elif defined (__AVR_ATmega88P__)
#  include <avr/iom88p.h>
#elif defined (__AVR_ATmega88PA__)
#  include <avr/iom88pa.h>
#elif defined (__AVR_ATmega88PB__)
#  include <avr/iom88pb.h>
#elif defined (__AVR_ATmega8515__)
```

```
#  include <avr/iom8515.h>
#elif defined (__AVR_ATmega8535__)
#  include <avr/iom8535.h>
#elif defined (__AVR_AT90S8535__)
#  include <avr/io8535.h>
#elif defined (__AVR_AT90C8534__)
#  include <avr/io8534.h>
#elif defined (__AVR_AT90S8515__)
#  include <avr/io8515.h>
#elif defined (__AVR_AT90S4434__)
#  include <avr/io4434.h>
#elif defined (__AVR_AT90S4433__)
#  include <avr/io4433.h>
#elif defined (__AVR_AT90S4414__)
#  include <avr/io4414.h>
#elif defined (__AVR_ATtiny22__)
#  include <avr/iotn22.h>
#elif defined (__AVR_ATtiny26__)
#  include <avr/iotn26.h>
#elif defined (__AVR_AT90S2343__)
#  include <avr/io2343.h>
#elif defined (__AVR_AT90S2333__)
#  include <avr/io2333.h>
#elif defined (__AVR_AT90S2323__)
#  include <avr/io2323.h>
#elif defined (__AVR_AT90S2313__)
#  include <avr/io2313.h>
#elif defined (__AVR_ATtiny4__)
#  include <avr/iotn4.h>
#elif defined (__AVR_ATtiny5__)
#  include <avr/iotn5.h>
#elif defined (__AVR_ATtiny9__)
#  include <avr/iotn9.h>
#elif defined (__AVR_ATtiny10__)
#  include <avr/iotn10.h>
#elif defined (__AVR_ATtiny20__)
#  include <avr/iotn20.h>
#elif defined (__AVR_ATtiny40__)
#  include <avr/iotn40.h>
#elif defined (__AVR_ATtiny2313__)
#  include <avr/iotn2313.h>
#elif defined (__AVR_ATtiny2313A__)
#  include <avr/iotn2313a.h>
#elif defined (__AVR_ATtiny13__)
#  include <avr/iotn13.h>
#elif defined (__AVR_ATtiny13A__)
#  include <avr/iotn13a.h>
#elif defined (__AVR_ATtiny25__)
#  include <avr/iotn25.h>
#elif defined (__AVR_ATtiny4313__)
#  include <avr/iotn4313.h>
#elif defined (__AVR_ATtiny45__)
#  include <avr/iotn45.h>
#elif defined (__AVR_ATtiny85__)
#  include <avr/iotn85.h>
#elif defined (__AVR_ATtiny24__)
#  include <avr/iotn24.h>
#elif defined (__AVR_ATtiny24A__)
#  include <avr/iotn24a.h>
#elif defined (__AVR_ATtiny44__)
#  include <avr/iotn44.h>
#elif defined (__AVR_ATtiny44A__)
#  include <avr/iotn44a.h>
#elif defined (__AVR_ATtiny441__)
#  include <avr/iotn441.h>
#elif defined (__AVR_ATtiny84__)
#  include <avr/iotn84.h>
#elif defined (__AVR_ATtiny84A__)
#  include <avr/iotn84a.h>
#elif defined (__AVR_ATtiny841__)
#  include <avr/iotn841.h>
#elif defined (__AVR_ATtiny261__)
#  include <avr/iotn261.h>
#elif defined (__AVR_ATtiny261A__)
#  include <avr/iotn261a.h>
#elif defined (__AVR_ATtiny461__)
#  include <avr/iotn461.h>
#elif defined (__AVR_ATtiny461A__)
#  include <avr/iotn461a.h>
#elif defined (__AVR_ATtiny861__)
#  include <avr/iotn861.h>
#elif defined (__AVR_ATtiny861A__)
#  include <avr/iotn861a.h>
#elif defined (__AVR_ATtiny43U__)
#  include <avr/iotn43u.h>
#elif defined (__AVR_ATtiny48__)
#  include <avr/iotn48.h>
#elif defined (__AVR_ATtiny88__)
#  include <avr/iotn88.h>
#elif defined (__AVR_ATtiny828__)
#  include <avr/iotn828.h>
#elif defined (__AVR_ATtiny87__)
#  include <avr/iotn87.h>
#elif defined (__AVR_ATtiny167__)
#  include <avr/iotn167.h>
#elif defined (__AVR_ATtiny1634__)
#  include <avr/iotn1634.h>
#elif defined (__AVR_AT90SCR100__)
#  include <avr/io90scr100.h>
#elif defined (__AVR_ATxmega8E5__)
#  include <avr/iox8e5.h>
#elif defined (__AVR_ATxmega16A4__)
#  include <avr/iox16a4.h>
#elif defined (__AVR_ATxmega16A4U__)
#  include <avr/iox16a4u.h>
#elif defined (__AVR_ATxmega16C4__)
#  include <avr/iox16c4.h>
#elif defined (__AVR_ATxmega16D4__)
#  include <avr/iox16d4.h>
#elif defined (__AVR_ATxmega16E5__)
#  include <avr/iox16e5.h>
#elif defined (__AVR_ATxmega32A4__)
#  include <avr/iox32a4.h>
#elif defined (__AVR_ATxmega32A4U__)
#  include <avr/iox32a4u.h>
#elif defined (__AVR_ATxmega32C3__)
#  include <avr/iox32c3.h>
#elif defined (__AVR_ATxmega32C4__)
#  include <avr/iox32c4.h>
#elif defined (__AVR_ATxmega32D3__)
#  include <avr/iox32d3.h>
#elif defined (__AVR_ATxmega32D4__)
#  include <avr/iox32d4.h>
#elif defined (__AVR_ATxmega32E5__)
#  include <avr/iox32e5.h>
#elif defined (__AVR_ATxmega64A1__)
#  include <avr/iox64a1.h>
#elif defined (__AVR_ATxmega64A1U__)
#  include <avr/iox64a1u.h>
#elif defined (__AVR_ATxmega64A3__)
#  include <avr/iox64a3.h>
#elif defined (__AVR_ATxmega64A3U__)
#  include <avr/iox64a3u.h>
#elif defined (__AVR_ATxmega64A4U__)
#  include <avr/iox64a4u.h>
#elif defined (__AVR_ATxmega64B1__)
#  include <avr/iox64b1.h>
#elif defined (__AVR_ATxmega64B3__)
#  include <avr/iox64b3.h>
#elif defined (__AVR_ATxmega64C3__)
#  include <avr/iox64c3.h>
```

```
#elif defined (__AVR_ATxmega64D3__)
#  include <avr/iox64d3.h>
#elif defined (__AVR_ATxmega64D4__)
#  include <avr/iox64d4.h>
#elif defined (__AVR_ATxmega128A1__)
#  include <avr/iox128a1.h>
#elif defined (__AVR_ATxmega128A1U__)
#  include <avr/iox128a1u.h>
#elif defined (__AVR_ATxmega128A4U__)
#  include <avr/iox128a4u.h>
#elif defined (__AVR_ATxmega128A3__)
#  include <avr/iox128a3.h>
#elif defined (__AVR_ATxmega128A3U__)
#  include <avr/iox128a3u.h>
#elif defined (__AVR_ATxmega128B1__)
#  include <avr/iox128b1.h>
#elif defined (__AVR_ATxmega128B3__)
#  include <avr/iox128b3.h>
#elif defined (__AVR_ATxmega128C3__)
#  include <avr/iox128c3.h>
#elif defined (__AVR_ATxmega128D3__)
#  include <avr/iox128d3.h>
#elif defined (__AVR_ATxmega128D4__)
#  include <avr/iox128d4.h>
#elif defined (__AVR_ATxmega192A3__)
#  include <avr/iox192a3.h>
#elif defined (__AVR_ATxmega192A3U__)
#  include <avr/iox192a3u.h>
#elif defined (__AVR_ATxmega192C3__)
#  include <avr/iox192c3.h>
#elif defined (__AVR_ATxmega192D3__)
#  include <avr/iox192d3.h>
#elif defined (__AVR_ATxmega256A3__)
#  include <avr/iox256a3.h>
#elif defined (__AVR_ATxmega256A3U__)
#  include <avr/iox256a3u.h>
#elif defined (__AVR_ATxmega256A3B__)
#  include <avr/iox256a3b.h>
#elif defined (__AVR_ATxmega256A3BU__)
#  include <avr/iox256a3bu.h>
#elif defined (__AVR_ATxmega256C3__)
#  include <avr/iox256c3.h>
#elif defined (__AVR_ATxmega256D3__)
#  include <avr/iox256d3.h>
#elif defined (__AVR_ATxmega384C3__)
#  include <avr/iox384c3.h>
#elif defined (__AVR_ATxmega384D3__)
#  include <avr/iox384d3.h>
#elif defined (__AVR_ATA5702M322__)
#  include <avr/ioa5702m322.h>
#elif defined (__AVR_ATA5782__)
#  include <avr/ioa5782.h>
#elif defined (__AVR_ATA5790__)
#  include <avr/ioa5790.h>
#elif defined (__AVR_ATA5790N__)
#  include <avr/ioa5790n.h>
#elif defined (__AVR_ATA5791__)
#  include <avr/ioa5791.h>
#elif defined (__AVR_ATA5831__)
#  include <avr/ioa5831.h>
#elif defined (__AVR_ATA5272__)
#  include <avr/ioa5272.h>
#elif defined (__AVR_ATA5505__)
#  include <avr/ioa5505.h>
#elif defined (__AVR_ATA5795__)
#  include <avr/ioa5795.h>
#elif defined (__AVR_ATA6285__)
#  include <avr/ioa6285.h>
#elif defined (__AVR_ATA6286__)
#  include <avr/ioa6286.h>
#elif defined (__AVR_ATA6289__)
#  include <avr/ioa6289.h>
#elif defined (__AVR_ATA6612C__)
#  include <avr/ioa6612c.h>
#elif defined (__AVR_ATA6613C__)
#  include <avr/ioa6613c.h>
#elif defined (__AVR_ATA6614Q__)
#  include <avr/ioa6614q.h>
#elif defined (__AVR_ATA6616C__)
#  include <avr/ioa6616c.h>
#elif defined (__AVR_ATA6617C__)
#  include <avr/ioa6617c.h>
#elif defined (__AVR_ATA664251__)
#  include <avr/ioa664251.h>
#elif defined (__AVR_ATA8210__)
#  include <avr/ioa8210.h>
#elif defined (__AVR_ATA8510__)
#  include <avr/ioa8510.h>
/* avr1: the following only supported for assembler programs */
#elif defined (__AVR_ATtiny28__)
#  include <avr/iotn28.h>
#elif defined (__AVR_AT90S1200__)
#  include <avr/io1200.h>
#elif defined (__AVR_ATtiny15__)
#  include <avr/iotn15.h>
#elif defined (__AVR_ATtiny12__)
#  include <avr/iotn12.h>
#elif defined (__AVR_ATtiny11__)
#  include <avr/iotn11.h>
#elif defined (__AVR_M3000__)
#  include <avr/iom3000.h>
#elif defined (__AVR_AVR128DA64__)
#  include <avr/ioavr128da64.h>
#elif defined (__AVR_AVR128DA48__)
#  include <avr/ioavr128da48.h>
#elif defined (__AVR_AVR128DA32__)
#  include <avr/ioavr128da32.h>
#elif defined (__AVR_AVR128DA28__)
#  include <avr/ioavr128da28.h>
#elif defined (__AVR_ATtiny817__)
#  include <avr/iotn817.h>
#elif defined (__AVR_ATtiny816__)
#  include <avr/iotn816.h>
#elif defined (__AVR_ATtiny814__)
#  include <avr/iotn814.h>
#elif defined (__AVR_ATtiny807__)
#  include <avr/iotn807.h>
#elif defined (__AVR_ATtiny806__)
#  include <avr/iotn806.h>
#elif defined (__AVR_ATtiny804__)
#  include <avr/iotn804.h>
#elif defined (__AVR_ATtiny417__)
#  include <avr/iotn417.h>
#elif defined (__AVR_ATtiny416__)
#  include <avr/iotn416.h>
#elif defined (__AVR_ATtiny414__)
#  include <avr/iotn414.h>
#elif defined (__AVR_ATtiny412__)
#  include <avr/iotn412.h>
#elif defined (__AVR_ATtiny406__)
#  include <avr/iotn406.h>
#elif defined (__AVR_ATtiny404__)
#  include <avr/iotn404.h>
#elif defined (__AVR_ATtiny402__)
#  include <avr/iotn402.h>
#elif defined (__AVR_ATtiny3217__)
#  include <avr/iotn3217.h>
#elif defined (__AVR_ATtiny3216__)
#  include <avr/iotn3216.h>
#elif defined (__AVR_ATtiny214__)
#  include <avr/iotn214.h>
```

```
#elif defined (__AVR_ATtiny212__)
#  include <avr/iotn212.h>
#elif defined (__AVR_ATtiny204__)
#  include <avr/iotn204.h>
#elif defined (__AVR_ATtiny202__)
#  include <avr/iotn202.h>
#elif defined (__AVR_ATtiny1627__)
#  include <avr/iotn1627.h>
#elif defined (__AVR_ATtiny1626__)
#  include <avr/iotn1626.h>
#elif defined (__AVR_ATtiny1624__)
#  include <avr/iotn1624.h>
#elif defined (__AVR_ATtiny1617__)
#  include <avr/iotn1617.h>
#elif defined (__AVR_ATtiny1616__)
#  include <avr/iotn1616.h>
#elif defined (__AVR_ATtiny1614__)
#  include <avr/iotn1614.h>
#elif defined (__AVR_ATtiny1607__)
#  include <avr/iotn1607.h>
#elif defined (__AVR_ATtiny1606__)
#  include <avr/iotn1606.h>
#elif defined (__AVR_ATtiny1604__)
#  include <avr/iotn1604.h>
#elif defined (__AVR_ATtiny104__)
#  include <avr/iotn104.h>
#elif defined (__AVR_ATtiny102__)
#  include <avr/iotn102.h>
#elif defined (__AVR_ATmega809__)
#  include <avr/iom809.h>
#elif defined (__AVR_ATmega808__)
#  include <avr/iom808.h>
#elif defined (__AVR_ATmega4809__)
#  include <avr/iom4809.h>
#elif defined (__AVR_ATmega4808__)
#  include <avr/iom4808.h>
#elif defined (__AVR_ATmega328PB__)
#  include <avr/iom328pb.h>
#elif defined (__AVR_ATmega324PB__)
#  include <avr/iom324pb.h>
#elif defined (__AVR_ATmega3209__)
#  include <avr/iom3209.h>
#elif defined (__AVR_ATmega3208__)
#  include <avr/iom3208.h>
#elif defined (__AVR_ATmega1609__)
#  include <avr/iom1609.h>
#elif defined (__AVR_ATmega1608__)
#  include <avr/iom1608.h>
#elif defined (__AVR_DEV_LIB_NAME__)
#  define __concat__(a,b) a##b
#  define __header1__(a,b) __concat__(a,b)
#  define __AVR_DEVICE_HEADER__ <avr/__header1__(io,__AVR_DEV_LIB_NAME__).h>
#  include __AVR_DEVICE_HEADER__
#else
#  if !defined(__COMPILING_AVR_LIBC__)
#    warning "device type not defined"
#  endif
#endif

#include <avr/portpins.h>

#include <avr/common.h>

#include <avr/version.h>

#if __AVR_ARCH__ >= 100
#  include <avr/xmega.h>
#endif

/* Include fuse.h after individual IO header files. */
#include <avr/fuse.h>

/* Include lock.h after individual IO header files. */
#include <avr/lock.h>

#endif /* _AVR_IO_H_ */
```