Adafruit_TFTLCD.cpp

```
// IMPORTANT: LIBRARY MUST BE SPECIFICALLY CONFIGURED FOR EITHER TFT SHIELD
// OR BREAKOUT BOARD USAGE.  SEE RELEVANT COMMENTS IN Adafruit_TFTLCD.h

// Graphics library by ladyada/adafruit with init code from Rossum
// MIT license

#if defined(__SAM3X8E__)
#include <include/pio.h>
#define PROGMEM
#define pgm_read_byte(addr) (*(const unsigned char *)(addr))
#define pgm_read_word(addr) (*(const unsigned short *)(addr))
#endif
#ifdef __AVR__
#include <avr/pgmspace.h>
#endif
#include "Adafruit_TFTLCD.h"
#include "pin_magic.h"
#include "pins_arduino.h"
#include "wiring_private.h"

//#define TFTWIDTH   320
//#define TFTHEIGHT  480

#define TFTWIDTH 240
#define TFTHEIGHT 320

// LCD controller chip identifiers
#define ID_932X 0
#define ID_7575 1
#define ID_9341 2
#define ID_HX8357D 3
#define ID_UNKNOWN 0xFF

#include "registers.h"

// Constructor for breakout board (configurable LCD control lines).
// Can still use this w/shield, but parameters are ignored.
Adafruit_TFTLCD::Adafruit_TFTLCD(uint8_t cs, uint8_t cd, uint8_t wr, uint8_t rd,
                                 uint8_t reset)
    : Adafruit_GFX(TFTWIDTH, TFTHEIGHT) {

#ifndef USE_ADAFRUIT_SHIELD_PINOUT
  // Convert pin numbers to registers and bitmasks
  _reset = reset;
#ifdef __AVR__
  csPort = portOutputRegister(digitalPinToPort(cs));
  cdPort = portOutputRegister(digitalPinToPort(cd));
  wrPort = portOutputRegister(digitalPinToPort(wr));
  rdPort = portOutputRegister(digitalPinToPort(rd));
#endif
#if defined(__SAM3X8E__)
  csPort = digitalPinToPort(cs);
  cdPort = digitalPinToPort(cd);
  wrPort = digitalPinToPort(wr);
  rdPort = digitalPinToPort(rd);
#endif
  csPinSet = digitalPinToBitMask(cs);
  cdPinSet = digitalPinToBitMask(cd);
  wrPinSet = digitalPinToBitMask(wr);
  rdPinSet = digitalPinToBitMask(rd);
  csPinUnset = ~csPinSet;
  cdPinUnset = ~cdPinSet;
  wrPinUnset = ~wrPinSet;
  rdPinUnset = ~rdPinSet;
#ifdef __AVR__
  *csPort |= csPinSet; // Set all control bits to HIGH (idle)
  *cdPort |= cdPinSet; // Signals are ACTIVE LOW
  *wrPort |= wrPinSet;
  *rdPort |= rdPinSet;
#endif
#if defined(__SAM3X8E__)
  csPort->PIO_SODR |= csPinSet; // Set all control bits to HIGH (idle)
  cdPort->PIO_SODR |= cdPinSet; // Signals are ACTIVE LOW
  wrPort->PIO_SODR |= wrPinSet;
  rdPort->PIO_SODR |= rdPinSet;
#endif
  pinMode(cs, OUTPUT); // Enable outputs
  pinMode(cd, OUTPUT);
  pinMode(wr, OUTPUT);
  pinMode(rd, OUTPUT);
  if (reset) {
    digitalWrite(reset, HIGH);
    pinMode(reset, OUTPUT);
  }
#endif

  init();
}
```

```
// Constructor for shield (fixed LCD control lines)
Adafruit_TFTLCD::Adafruit_TFTLCD(void) : Adafruit_GFX(TFTWIDTH, TFTHEIGHT) {
  init();
}

// Initialization common to both shield & breakout configs
void Adafruit_TFTLCD::init(void) {

#ifdef USE_ADAFRUIT_SHIELD_PINOUT
  CS_IDLE; // Set all control bits to idle state
  WR_IDLE;
  RD_IDLE;
  CD_DATA;
  digitalWrite(5, HIGH); // Reset line
  pinMode(A3, OUTPUT);   // Enable outputs
  pinMode(A2, OUTPUT);
  pinMode(A1, OUTPUT);
  pinMode(A0, OUTPUT);
  pinMode(5, OUTPUT);
#endif

  setWriteDir(); // Set up LCD data port(s) for WRITE operations

  rotation = 0;
  cursor_y = cursor_x = 0;
  textcolor = 0xFFFF;
  _width = TFTWIDTH;
  _height = TFTHEIGHT;
}

// Initialization command tables for different LCD controllers
#define TFTLCD_DELAY 0xFF
static const uint8_t HX8347G_regValues[] PROGMEM = {
    0x2E, 0x89, 0x29, 0x8F, 0x2B, 0x02, 0xE2, 0x00, 0xE4, 0x01, 0xE5, 0x10,
    0xE6, 0x01, 0xE7, 0x10, 0xE8, 0x70, 0xF2, 0x00, 0xEA, 0x00, 0xEB, 0x20,
    0xEC, 0x3C, 0xED, 0xC8, 0xE9, 0x38, 0xF1, 0x01,

    // skip gamma, do later

    0x1B, 0x1A, 0x1A, 0x02, 0x24, 0x61, 0x25, 0x5C,

    0x18, 0x36, 0x19, 0x01, 0x1F, 0x88, TFTLCD_DELAY, 5, // delay 5 ms
    0x1F, 0x80, TFTLCD_DELAY, 5, 0x1F, 0x90, TFTLCD_DELAY, 5, 0x1F, 0xD4,
    TFTLCD_DELAY, 5, 0x17, 0x05,

    0x36, 0x09, 0x28, 0x38, TFTLCD_DELAY, 40, 0x28, 0x3C,

    0x02, 0x00, 0x03, 0x00, 0x04, 0x00, 0x05, 0xEF, 0x06, 0x00, 0x07, 0x00,
    0x08, 0x01, 0x09, 0x3F};

static const uint8_t HX8357D_regValues[] PROGMEM = {
    HX8357_SWRESET,
    0,
    HX8357D_SETC,
    3,
    0xFF,
    0x83,
    0x57,
    TFTLCD_DELAY,
    250,
    HX8357_SETRGB,
    4,
    0x00,
    0x00,
    0x06,
    0x06,
    HX8357D_SETCOM,
    1,
    0x25, // -1.52V
    HX8357_SETOSC,
    1,
    0x68, // Normal mode 70Hz, Idle mode 55 Hz
    HX8357_SETPANEL,
    1,
    0x05, // BGR, Gate direction swapped
    HX8357_SETPWR1,
    6,
    0x00,
    0x15,
    0x1C,
    0x1C,
    0x83,
    0xAA,
    HX8357D_SETSTBA,
    6,
    0x50,
    0x50,
    0x01,
    0x3C,
```

```
    0x1E,
    0x08,
    // MEME GAMMA HERE
    HX8357D_SETCYC,
    7,
    0x02,
    0x40,
    0x00,
    0x2A,
    0x2A,
    0x0D,
    0x78,
    HX8357_COLMOD,
    1,
    0x55,
    HX8357_MADCTL,
    1,
    0xC0,
    HX8357_TEON,
    1,
    0x00,
    HX8357_TEARLINE,
    2,
    0x00,
    0x02,
    HX8357_SLPOUT,
    0,
    TFTLCD_DELAY,
    150,
    HX8357_DISPON,
    0,
    TFTLCD_DELAY,
    50,
};

static const uint16_t ILI932x_regValues[] PROGMEM = {
    ILI932X_START_OSC,
    0x0001, // Start oscillator
    TFTLCD_DELAY,
    50, // 50 millisecond delay
    ILI932X_DRIV_OUT_CTRL,
    0x0100,
    ILI932X_DRIV_WAV_CTRL,
    0x0700,
    ILI932X_ENTRY_MOD,
    0x1030,
    ILI932X_RESIZE_CTRL,
    0x0000,
    ILI932X_DISP_CTRL2,
    0x0202,
    ILI932X_DISP_CTRL3,
    0x0000,
    ILI932X_DISP_CTRL4,
    0x0000,
    ILI932X_RGB_DISP_IF_CTRL1,
    0x0,
    ILI932X_FRM_MARKER_POS,
    0x0,
    ILI932X_RGB_DISP_IF_CTRL2,
    0x0,
    ILI932X_POW_CTRL1,
    0x0000,
    ILI932X_POW_CTRL2,
    0x0007,
    ILI932X_POW_CTRL3,
    0x0000,
    ILI932X_POW_CTRL4,
    0x0000,
    TFTLCD_DELAY,
    200,
    ILI932X_POW_CTRL1,
    0x1690,
    ILI932X_POW_CTRL2,
    0x0227,
    TFTLCD_DELAY,
    50,
    ILI932X_POW_CTRL3,
    0x001A,
    TFTLCD_DELAY,
    50,
    ILI932X_POW_CTRL4,
    0x1800,
    ILI932X_POW_CTRL7,
    0x002A,
    TFTLCD_DELAY,
    50,
    ILI932X_GAMMA_CTRL1,
    0x0000,
    ILI932X_GAMMA_CTRL2,
    0x0000,
```

```
    ILI932X_GAMMA_CTRL3,
    0x0000,
    ILI932X_GAMMA_CTRL4,
    0x0206,
    ILI932X_GAMMA_CTRL5,
    0x0808,
    ILI932X_GAMMA_CTRL6,
    0x0007,
    ILI932X_GAMMA_CTRL7,
    0x0201,
    ILI932X_GAMMA_CTRL8,
    0x0000,
    ILI932X_GAMMA_CTRL9,
    0x0000,
    ILI932X_GAMMA_CTRL10,
    0x0000,
    ILI932X_GRAM_HOR_AD,
    0x0000,
    ILI932X_GRAM_VER_AD,
    0x0000,
    ILI932X_HOR_START_AD,
    0x0000,
    ILI932X_HOR_END_AD,
    0x00EF,
    ILI932X_VER_START_AD,
    0X0000,
    ILI932X_VER_END_AD,
    0x013F,
    ILI932X_GATE_SCAN_CTRL1,
    0xA700, // Driver Output Control (R60h)
    ILI932X_GATE_SCAN_CTRL2,
    0x0003, // Driver Output Control (R61h)
    ILI932X_GATE_SCAN_CTRL3,
    0x0000, // Driver Output Control (R62h)
    ILI932X_PANEL_IF_CTRL1,
    0X0010, // Panel Interface Control 1 (R90h)
    ILI932X_PANEL_IF_CTRL2,
    0X0000,
    ILI932X_PANEL_IF_CTRL3,
    0X0003,
    ILI932X_PANEL_IF_CTRL4,
    0X1100,
    ILI932X_PANEL_IF_CTRL5,
    0X0000,
    ILI932X_PANEL_IF_CTRL6,
    0X0000,
    ILI932X_DISP_CTRL1,
    0x0133, // Main screen turn on
};

void Adafruit_TFTLCD::begin(uint16_t id) {
  uint8_t i = 0;

  reset();

  delay(200);

  if ((id == 0x9325) || (id == 0x9328)) {

    uint16_t a, d;
    driver = ID_932X;
    CS_ACTIVE;
    while (i < sizeof(ILI932x_regValues) / sizeof(uint16_t)) {
      a = pgm_read_word(&ILI932x_regValues[i++]);
      d = pgm_read_word(&ILI932x_regValues[i++]);
      if (a == TFTLCD_DELAY)
        delay(d);
      else
        writeRegister16(a, d);
    }
    setRotation(rotation);
    setAddrWindow(0, 0, TFTWIDTH - 1, TFTHEIGHT - 1);

  } else if (id == 0x9341) {

    driver = ID_9341;
    CS_ACTIVE;
    writeRegister8(ILI9341_SOFTRESET, 0);
    delay(50);
    writeRegister8(ILI9341_DISPLAYOFF, 0);

    writeRegister8(ILI9341_POWERCONTROL1, 0x23);
    writeRegister8(ILI9341_POWERCONTROL2, 0x10);
    writeRegister16(ILI9341_VCOMCONTROL1, 0x2B2B);
    writeRegister8(ILI9341_VCOMCONTROL2, 0xC0);
    writeRegister8(ILI9341_MEMCONTROL, ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR);
    writeRegister8(ILI9341_PIXELFORMAT, 0x55);
    writeRegister16(ILI9341_FRAMECONTROL, 0x001B);

    writeRegister8(ILI9341_ENTRYMODE, 0x07);
```

```
    /* writeRegister32(ILI9341_DISPLAYFUNC, 0x0A822700);*/

    writeRegister8(ILI9341_SLEEPOUT, 0);
    delay(150);
    writeRegister8(ILI9341_DISPLAYON, 0);
    delay(500);
    setAddrWindow(0, 0, TFTWIDTH - 1, TFTHEIGHT - 1);
    return;

  } else if (id == 0x8357) {
    // HX8357D
    driver = ID_HX8357D;
    CS_ACTIVE;
    while (i < sizeof(HX8357D_regValues)) {
      uint8_t r = pgm_read_byte(&HX8357D_regValues[i++]);
      uint8_t len = pgm_read_byte(&HX8357D_regValues[i++]);
      if (r == TFTLCD_DELAY) {
        delay(len);
      } else {
        // Serial.print("Register $"); Serial.print(r, HEX);
        // Serial.print(" datalen "); Serial.println(len);

        CS_ACTIVE;
        CD_COMMAND;
        write8(r);
        CD_DATA;
        for (uint8_t d = 0; d < len; d++) {
          uint8_t x = pgm_read_byte(&HX8357D_regValues[i++]);
          write8(x);
        }
        CS_IDLE;
      }
    }
    return;

  } else if (id == 0x7575) {

    uint8_t a, d;
    driver = ID_7575;
    CS_ACTIVE;
    while (i < sizeof(HX8347G_regValues)) {
      a = pgm_read_byte(&HX8347G_regValues[i++]);
      d = pgm_read_byte(&HX8347G_regValues[i++]);
      if (a == TFTLCD_DELAY)
        delay(d);
      else
        writeRegister8(a, d);
    }
    setRotation(rotation);
    setLR(); // Lower-right corner of address window

  } else {
    driver = ID_UNKNOWN;
    return;
  }
}

void Adafruit_TFTLCD::reset(void) {

  CS_IDLE;
  //  CD_DATA;
  WR_IDLE;
  RD_IDLE;

#ifdef USE_ADAFRUIT_SHIELD_PINOUT
  digitalWrite(5, LOW);
  delay(2);
  digitalWrite(5, HIGH);
#else
  if (_reset) {
    digitalWrite(_reset, LOW);
    delay(2);
    digitalWrite(_reset, HIGH);
  }
#endif

  // Data transfer sync
  CS_ACTIVE;
  CD_COMMAND;
  write8(0x00);
  for (uint8_t i = 0; i < 3; i++)
    WR_STROBE; // Three extra 0x00s
  CS_IDLE;
}

// Sets the LCD address window (and address counter, on 932X).
// Relevant to rect/screen fills and H/V lines.  Input coordinates are
// assumed pre-sorted (e.g. x2 >= x1).
void Adafruit_TFTLCD::setAddrWindow(int x1, int y1, int x2, int y2) {
  CS_ACTIVE;
```

```
  if (driver == ID_932X) {

    // Values passed are in current (possibly rotated) coordinate
    // system.  932X requires hardware-native coords regardless of
    // MADCTL, so rotate inputs as needed.  The address counter is
    // set to the top-left corner -- although fill operations can be
    // done in any direction, the current screen rotation is applied
    // because some users find it disconcerting when a fill does not
    // occur top-to-bottom.
    int x, y, t;
    switch (rotation) {
    default:
      x = x1;
      y = y1;
      break;
    case 1:
      t = y1;
      y1 = x1;
      x1 = TFTWIDTH - 1 - y2;
      y2 = x2;
      x2 = TFTWIDTH - 1 - t;
      x = x2;
      y = y1;
      break;
    case 2:
      t = x1;
      x1 = TFTWIDTH - 1 - x2;
      x2 = TFTWIDTH - 1 - t;
      t = y1;
      y1 = TFTHEIGHT - 1 - y2;
      y2 = TFTHEIGHT - 1 - t;
      x = x2;
      y = y2;
      break;
    case 3:
      t = x1;
      x1 = y1;
      y1 = TFTHEIGHT - 1 - x2;
      x2 = y2;
      y2 = TFTHEIGHT - 1 - t;
      x = x1;
      y = y2;
      break;
    }
    writeRegister16(0x0050, x1); // Set address window
    writeRegister16(0x0051, x2);
    writeRegister16(0x0052, y1);
    writeRegister16(0x0053, y2);
    writeRegister16(0x0020, x); // Set address counter to top left
    writeRegister16(0x0021, y);

  } else if (driver == ID_7575) {

    writeRegisterPair(HX8347G_COLADDRSTART_HI, HX8347G_COLADDRSTART_LO, x1);
    writeRegisterPair(HX8347G_ROWADDRSTART_HI, HX8347G_ROWADDRSTART_LO, y1);
    writeRegisterPair(HX8347G_COLADDREND_HI, HX8347G_COLADDREND_LO, x2);
    writeRegisterPair(HX8347G_ROWADDREND_HI, HX8347G_ROWADDREND_LO, y2);

  } else if ((driver == ID_9341) || (driver == ID_HX8357D)) {
    uint32_t t;

    t = x1;
    t <<= 16;
    t |= x2;
    writeRegister32(ILI9341_COLADDRSET, t); // HX8357D uses same registers!
    t = y1;
    t <<= 16;
    t |= y2;
    writeRegister32(ILI9341_PAGEADDRSET, t); // HX8357D uses same registers!
  }
  CS_IDLE;
}

// Unlike the 932X drivers that set the address window to the full screen
// by default (using the address counter for drawPixel operations), the
// 7575 needs the address window set on all graphics operations.  In order
// to save a few register writes on each pixel drawn, the lower-right
// corner of the address window is reset after most fill operations, so
// that drawPixel only needs to change the upper left each time.
void Adafruit_TFTLCD::setLR(void) {
  CS_ACTIVE;
  writeRegisterPair(HX8347G_COLADDREND_HI, HX8347G_COLADDREND_LO, _width - 1);
  writeRegisterPair(HX8347G_ROWADDREND_HI, HX8347G_ROWADDREND_LO, _height - 1);
  CS_IDLE;
}

// Fast block fill operation for fillScreen, fillRect, H/V line, etc.
// Requires setAddrWindow() has previously been called to set the fill
// bounds.  'len' is inclusive, MUST be >= 1.
void Adafruit_TFTLCD::flood(uint16_t color, uint32_t len) {
```

```
  uint16_t blocks;
  uint8_t i, hi = color >> 8, lo = color;

  CS_ACTIVE;
  CD_COMMAND;
  if (driver == ID_9341) {
    write8(0x2C);
  } else if (driver == ID_932X) {
    write8(0x00); // High byte of GRAM register...
    write8(0x22); // Write data to GRAM
  } else if (driver == ID_HX8357D) {
    write8(HX8357_RAMWR);
  } else {
    write8(0x22); // Write data to GRAM
  }

  // Write first pixel normally, decrement counter by 1
  CD_DATA;
  write8(hi);
  write8(lo);
  len--;

  blocks = (uint16_t)(len / 64); // 64 pixels/block
  if (hi == lo) {
    // High and low bytes are identical.  Leave prior data
    // on the port(s) and just toggle the write strobe.
    while (blocks--) {
      i = 16; // 64 pixels/block / 4 pixels/pass
      do {
        WR_STROBE;
        WR_STROBE;
        WR_STROBE;
        WR_STROBE; // 2 bytes/pixel
        WR_STROBE;
        WR_STROBE;
        WR_STROBE;
        WR_STROBE; // x 4 pixels
      } while (--i);
    }
    // Fill any remaining pixels (1 to 64)
    for (i = (uint8_t)len & 63; i--;) {
      WR_STROBE;
      WR_STROBE;
    }
  } else {
    while (blocks--) {
      i = 16; // 64 pixels/block / 4 pixels/pass
      do {
        write8(hi);
        write8(lo);
        write8(hi);
        write8(lo);
        write8(hi);
        write8(lo);
        write8(hi);
        write8(lo);
      } while (--i);
    }
    for (i = (uint8_t)len & 63; i--;) {
      write8(hi);
      write8(lo);
    }
  }
  CS_IDLE;
}

void Adafruit_TFTLCD::drawFastHLine(int16_t x, int16_t y, int16_t length,
                                    uint16_t color) {
  int16_t x2;

  // Initial off-screen clipping
  if ((length <= 0) || (y < 0) || (y >= _height) || (x >= _width) ||
      ((x2 = (x + length - 1)) < 0))
    return;

  if (x < 0) { // Clip left
    length += x;
    x = 0;
  }
  if (x2 >= _width) { // Clip right
    x2 = _width - 1;
    length = x2 - x + 1;
  }

  setAddrWindow(x, y, x2, y);
  flood(color, length);
  if (driver == ID_932X)
    setAddrWindow(0, 0, _width - 1, _height - 1);
  else
    setLR();
```

```
}

void Adafruit_TFTLCD::drawFastVLine(int16_t x, int16_t y, int16_t length,
                                    uint16_t color) {
  int16_t y2;

  // Initial off-screen clipping
  if ((length <= 0) || (x < 0) || (x >= _width) || (y >= _height) ||
      ((y2 = (y + length - 1)) < 0))
    return;
  if (y < 0) { // Clip top
    length += y;
    y = 0;
  }
  if (y2 >= _height) { // Clip bottom
    y2 = _height - 1;
    length = y2 - y + 1;
  }

  setAddrWindow(x, y, x, y2);
  flood(color, length);
  if (driver == ID_932X)
    setAddrWindow(0, 0, _width - 1, _height - 1);
  else
    setLR();
}

void Adafruit_TFTLCD::fillRect(int16_t x1, int16_t y1, int16_t w, int16_t h,
                               uint16_t fillcolor) {
  int16_t x2, y2;

  // Initial off-screen clipping
  if ((w <= 0) || (h <= 0) || (x1 >= _width) || (y1 >= _height) ||
      ((x2 = x1 + w - 1) < 0) || ((y2 = y1 + h - 1) < 0))
    return;
  if (x1 < 0) { // Clip left
    w += x1;
    x1 = 0;
  }
  if (y1 < 0) { // Clip top
    h += y1;
    y1 = 0;
  }
  if (x2 >= _width) { // Clip right
    x2 = _width - 1;
    w = x2 - x1 + 1;
  }
  if (y2 >= _height) { // Clip bottom
    y2 = _height - 1;
    h = y2 - y1 + 1;
  }

  setAddrWindow(x1, y1, x2, y2);
  flood(fillcolor, (uint32_t)w * (uint32_t)h);
  if (driver == ID_932X)
    setAddrWindow(0, 0, _width - 1, _height - 1);
  else
    setLR();
}

void Adafruit_TFTLCD::fillScreen(uint16_t color) {

  if (driver == ID_932X) {

    // For the 932X, a full-screen address window is already the default
    // state, just need to set the address pointer to the top-left corner.
    // Although we could fill in any direction, the code uses the current
    // screen rotation because some users find it disconcerting when a
    // fill does not occur top-to-bottom.
    uint16_t x, y;
    switch (rotation) {
    default:
      x = 0;
      y = 0;
      break;
    case 1:
      x = TFTWIDTH - 1;
      y = 0;
      break;
    case 2:
      x = TFTWIDTH - 1;
      y = TFTHEIGHT - 1;
      break;
    case 3:
      x = 0;
      y = TFTHEIGHT - 1;
      break;
    }
    CS_ACTIVE;
    writeRegister16(0x0020, x);
```

```
    writeRegister16(0x0021, y);

  } else if ((driver == ID_9341) || (driver == ID_7575) ||
             (driver == ID_HX8357D)) {
    // For these, there is no settable address pointer, instead the
    // address window must be set for each drawing operation.  However,
    // this display takes rotation into account for the parameters, no
    // need to do extra rotation math here.
    setAddrWindow(0, 0, _width - 1, _height - 1);
  }
  flood(color, (long)TFTWIDTH * (long)TFTHEIGHT);
}

void Adafruit_TFTLCD::drawPixel(int16_t x, int16_t y, uint16_t color) {

  // Clip
  if ((x < 0) || (y < 0) || (x >= _width) || (y >= _height))
    return;

  CS_ACTIVE;
  if (driver == ID_932X) {
    int16_t t;
    switch (rotation) {
    case 1:
      t = x;
      x = TFTWIDTH - 1 - y;
      y = t;
      break;
    case 2:
      x = TFTWIDTH - 1 - x;
      y = TFTHEIGHT - 1 - y;
      break;
    case 3:
      t = x;
      x = y;
      y = TFTHEIGHT - 1 - t;
      break;
    }
    writeRegister16(0x0020, x);
    writeRegister16(0x0021, y);
    writeRegister16(0x0022, color);

  } else if (driver == ID_7575) {

    uint8_t hi, lo;
    switch (rotation) {
    default:
      lo = 0;
      break;
    case 1:
      lo = 0x60;
      break;
    case 2:
      lo = 0xc0;
      break;
    case 3:
      lo = 0xa0;
      break;
    }
    writeRegister8(HX8347G_MEMACCESS, lo);
    // Only upper-left is set -- bottom-right is full screen default
    writeRegisterPair(HX8347G_COLADDRSTART_HI, HX8347G_COLADDRSTART_LO, x);
    writeRegisterPair(HX8347G_ROWADDRSTART_HI, HX8347G_ROWADDRSTART_LO, y);
    hi = color >> 8;
    lo = color;
    CD_COMMAND;
    write8(0x22);
    CD_DATA;
    write8(hi);
    write8(lo);

  } else if ((driver == ID_9341) || (driver == ID_HX8357D)) {
    setAddrWindow(x, y, _width - 1, _height - 1);
    CS_ACTIVE;
    CD_COMMAND;
    write8(0x2C);
    CD_DATA;
    write8(color >> 8);
    write8(color);
  }

  CS_IDLE;
}

// Issues 'raw' an array of 16-bit color values to the LCD; used
// externally by BMP examples.  Assumes that setWindowAddr() has
// previously been set to define the bounds.  Max 255 pixels at
// a time (BMP examples read in small chunks due to limited RAM).
void Adafruit_TFTLCD::pushColors(uint16_t *data, uint8_t len, boolean first) {
  uint16_t color;
```

```
  uint8_t hi, lo;
  CS_ACTIVE;
  if (first == true) { // Issue GRAM write command only on first call
    CD_COMMAND;
    if (driver == ID_932X)
      write8(0x00);
    if ((driver == ID_9341) || (driver == ID_HX8357D)) {
      write8(0x2C);
    } else {
      write8(0x22);
    }
  }
  CD_DATA;
  while (len--) {
    color = *data++;
    hi = color >> 8; // Don't simplify or merge these
    lo = color;      // lines, there's macro shenanigans
    write8(hi);      // going on.
    write8(lo);
  }
  CS_IDLE;
}

void Adafruit_TFTLCD::setRotation(uint8_t x) {

  // Call parent rotation func first -- sets up rotation flags, etc.
  Adafruit_GFX::setRotation(x);
  // Then perform hardware-specific rotation operations...

  CS_ACTIVE;
  if (driver == ID_932X) {

    uint16_t t;
    switch (rotation) {
    default:
      t = 0x1030;
      break;
    case 1:
      t = 0x1028;
      break;
    case 2:
      t = 0x1000;
      break;
    case 3:
      t = 0x1018;
      break;
    }
    writeRegister16(0x0003, t); // MADCTL
    // For 932X, init default full-screen address window:
    setAddrWindow(0, 0, _width - 1, _height - 1); // CS_IDLE happens here
  }
  if (driver == ID_7575) {

    uint8_t t;
    switch (rotation) {
    default:
      t = 0;
      break;
    case 1:
      t = 0x60;
      break;
    case 2:
      t = 0xc0;
      break;
    case 3:
      t = 0xa0;
      break;
    }
    writeRegister8(HX8347G_MEMACCESS, t);
    // 7575 has to set the address window on most drawing operations.
    // drawPixel() cheats by setting only the top left...by default,
    // the lower right is always reset to the corner.
    setLR(); // CS_IDLE happens here
  }

  if (driver == ID_9341) {
    // MEME, HX8357D uses same registers as 9341 but different values
    uint16_t t = 0;

    switch (rotation) {
    case 2:
      t = ILI9341_MADCTL_MX | ILI9341_MADCTL_BGR;
      break;
    case 3:
      t = ILI9341_MADCTL_MV | ILI9341_MADCTL_BGR;
      break;
    case 0:
      t = ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR;
      break;
    case 1:
```

```
      t = ILI9341_MADCTL_MX | ILI9341_MADCTL_MY | ILI9341_MADCTL_MV |
          ILI9341_MADCTL_BGR;
      break;
    }
    writeRegister8(ILI9341_MADCTL, t); // MADCTL
    // For 9341, init default full-screen address window:
    setAddrWindow(0, 0, _width - 1, _height - 1); // CS_IDLE happens here
  }

  if (driver == ID_HX8357D) {
    // MEME, HX8357D uses same registers as 9341 but different values
    uint16_t t = 0;

    switch (rotation) {
    case 2:
      t = HX8357B_MADCTL_RGB;
      break;
    case 3:
      t = HX8357B_MADCTL_MX | HX8357B_MADCTL_MV | HX8357B_MADCTL_RGB;
      break;
    case 0:
      t = HX8357B_MADCTL_MX | HX8357B_MADCTL_MY | HX8357B_MADCTL_RGB;
      break;
    case 1:
      t = HX8357B_MADCTL_MY | HX8357B_MADCTL_MV | HX8357B_MADCTL_RGB;
      break;
    }
    writeRegister8(ILI9341_MADCTL, t); // MADCTL
    // For 8357, init default full-screen address window:
    setAddrWindow(0, 0, _width - 1, _height - 1); // CS_IDLE happens here
  }
}

#ifdef read8isFunctionalized
#define read8(x) x = read8fn()
#endif

// Because this function is used infrequently, it configures the ports for
// the read operation, reads the data, then restores the ports to the write
// configuration.  Write operations happen a LOT, so it's advantageous to
// leave the ports in that state as a default.
uint16_t Adafruit_TFTLCD::readPixel(int16_t x, int16_t y) {

  if ((x < 0) || (y < 0) || (x >= _width) || (y >= _height))
    return 0;

  CS_ACTIVE;
  if (driver == ID_932X) {

    uint8_t hi, lo;
    int16_t t;
    switch (rotation) {
    case 1:
      t = x;
      x = TFTWIDTH - 1 - y;
      y = t;
      break;
    case 2:
      x = TFTWIDTH - 1 - x;
      y = TFTHEIGHT - 1 - y;
      break;
    case 3:
      t = x;
      x = y;
      y = TFTHEIGHT - 1 - t;
      break;
    }
    writeRegister16(0x0020, x);
    writeRegister16(0x0021, y);
    // Inexplicable thing: sometimes pixel read has high/low bytes
    // reversed.  A second read fixes this.  Unsure of reason.  Have
    // tried adjusting timing in read8() etc. to no avail.
    for (uint8_t pass = 0; pass < 2; pass++) {
      CD_COMMAND;
      write8(0x00);
      write8(0x22); // Read data from GRAM
      CD_DATA;
      setReadDir(); // Set up LCD data port(s) for READ operations
      read8(hi);    // First 2 bytes back are a dummy read
      read8(hi);
      read8(hi); // Bytes 3, 4 are actual pixel value
      read8(lo);
      setWriteDir(); // Restore LCD data port(s) to WRITE configuration
    }
    CS_IDLE;
    return ((uint16_t)hi << 8) | lo;

  } else if (driver == ID_7575) {

    uint8_t r, g, b;
```

```
    writeRegisterPair(HX8347G_COLADDRSTART_HI, HX8347G_COLADDRSTART_LO, x);
    writeRegisterPair(HX8347G_ROWADDRSTART_HI, HX8347G_ROWADDRSTART_LO, y);
    CD_COMMAND;
    write8(0x22); // Read data from GRAM
    setReadDir(); // Set up LCD data port(s) for READ operations
    CD_DATA;
    read8(r); // First byte back is a dummy read
    read8(r);
    read8(g);
    read8(b);
    setWriteDir(); // Restore LCD data port(s) to WRITE configuration
    CS_IDLE;
    return (((uint16_t)r & B11111000) << 8) | (((uint16_t)g & B11111100) << 3) |
           (b >> 3);
  } else
    return 0;
}

// Ditto with the read/write port directions, as above.
uint16_t Adafruit_TFTLCD::readID(void) {
  uint16_t id;

  // retry a bunch!
  for (int i = 0; i < 5; i++) {
    id = (uint16_t)readReg(0xD3);
    delayMicroseconds(50);
    if (id == 0x9341) {
      return id;
    }
  }

  uint8_t hi, lo;

  /*
  for (uint8_t i=0; i<128; i++) {
    Serial.print("$"); Serial.print(i, HEX);
    Serial.print(" = 0x"); Serial.println(readReg(i), HEX);
  }
  */

  if (readReg(0x04) == 0x8000) { // eh close enough
    // setc!
    /*
      Serial.println("!");
      for (uint8_t i=0; i<254; i++) {
      Serial.print("$"); Serial.print(i, HEX);
      Serial.print(" = 0x"); Serial.println(readReg(i), HEX);
      }
    */
    writeRegister24(HX8357D_SETC, 0xFF8357);
    delay(300);
    // Serial.println(readReg(0xD0), HEX);
    if (readReg(0xD0) == 0x990000) {
      return 0x8357;
    }
  }

  CS_ACTIVE;
  CD_COMMAND;
  write8(0x00);
  WR_STROBE;    // Repeat prior byte (0x00)
  setReadDir(); // Set up LCD data port(s) for READ operations
  CD_DATA;
  read8(hi);
  read8(lo);
  setWriteDir(); // Restore LCD data port(s) to WRITE configuration
  CS_IDLE;

  id = hi;
  id <<= 8;
  id |= lo;
  return id;
}

uint32_t Adafruit_TFTLCD::readReg(uint8_t r) {
  uint32_t id;
  uint8_t x;

  // try reading register #4
  CS_ACTIVE;
  CD_COMMAND;
  write8(r);
  setReadDir(); // Set up LCD data port(s) for READ operations
  CD_DATA;
  delayMicroseconds(50);
  read8(x);
  id = x;   // Do not merge or otherwise simplify
  id <<= 8; // these lines.  It's an unfortunate
  read8(x);
  id |= x;  // shenanigans that are going on.
```

```
  id <<= 8; // these lines.  It's an unfortunate
  read8(x);
  id |= x;  // shenanigans that are going on.
  id <<= 8; // these lines.  It's an unfortunate
  read8(x);
  id |= x; // shenanigans that are going on.
  CS_IDLE;
  setWriteDir(); // Restore LCD data port(s) to WRITE configuration

  // Serial.print("Read $"); Serial.print(r, HEX);
  // Serial.print(":\t0x"); Serial.println(id, HEX);
  return id;
}

// Pass 8-bit (each) R,G,B, get back 16-bit packed color
uint16_t Adafruit_TFTLCD::color565(uint8_t r, uint8_t g, uint8_t b) {
  return ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | (b >> 3);
}

// For I/O macros that were left undefined, declare function
// versions that reference the inline macros just once:

#ifndef write8
void Adafruit_TFTLCD::write8(uint8_t value) { write8inline(value); }
#endif

#ifdef read8isFunctionalized
uint8_t Adafruit_TFTLCD::read8fn(void) {
  uint8_t result;
  read8inline(result);
  return result;
}
#endif

#ifndef setWriteDir
void Adafruit_TFTLCD::setWriteDir(void) { setWriteDirInline(); }
#endif

#ifndef setReadDir
void Adafruit_TFTLCD::setReadDir(void) { setReadDirInline(); }
#endif

#ifndef writeRegister8
void Adafruit_TFTLCD::writeRegister8(uint8_t a, uint8_t d) {
  writeRegister8inline(a, d);
}
#endif

#ifndef writeRegister16
void Adafruit_TFTLCD::writeRegister16(uint16_t a, uint16_t d) {
  writeRegister16inline(a, d);
}
#endif

#ifndef writeRegisterPair
void Adafruit_TFTLCD::writeRegisterPair(uint8_t aH, uint8_t aL, uint16_t d) {
  writeRegisterPairInline(aH, aL, d);
}
#endif

void Adafruit_TFTLCD::writeRegister24(uint8_t r, uint32_t d) {
  CS_ACTIVE;
  CD_COMMAND;
  write8(r);
  CD_DATA;
  delayMicroseconds(10);
  write8(d >> 16);
  delayMicroseconds(10);
  write8(d >> 8);
  delayMicroseconds(10);
  write8(d);
  CS_IDLE;
}

void Adafruit_TFTLCD::writeRegister32(uint8_t r, uint32_t d) {
  CS_ACTIVE;
  CD_COMMAND;
  write8(r);
  CD_DATA;
  delayMicroseconds(10);
  write8(d >> 24);
  delayMicroseconds(10);
  write8(d >> 16);
  delayMicroseconds(10);
  write8(d >> 8);
  delayMicroseconds(10);
  write8(d);
  CS_IDLE;
}
```

Adafruit_TFTLCD.h

```
// IMPORTANT: SEE COMMENTS @ LINE 15 REGARDING SHIELD VS BREAKOUT BOARD USAGE.

// Graphics library by ladyada/adafruit with init code from Rossum
// MIT license

#ifndef _ADAFRUIT_TFTLCD_H_
#define _ADAFRUIT_TFTLCD_H_

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

#include <Adafruit_GFX.h>

// **** IF USING THE LCD BREAKOUT BOARD, COMMENT OUT THIS NEXT LINE. ****
// **** IF USING THE LCD SHIELD, LEAVE THE LINE ENABLED:             ****

//#define USE_ADAFRUIT_SHIELD_PINOUT 1

class Adafruit_TFTLCD : public Adafruit_GFX {

public:
  Adafruit_TFTLCD(uint8_t cs, uint8_t cd, uint8_t wr, uint8_t rd, uint8_t rst);
  Adafruit_TFTLCD(void);

  void begin(uint16_t id = 0x9325);
  void drawPixel(int16_t x, int16_t y, uint16_t color);
  void drawFastHLine(int16_t x0, int16_t y0, int16_t w, uint16_t color);
  void drawFastVLine(int16_t x0, int16_t y0, int16_t h, uint16_t color);
  void fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t c);
  void fillScreen(uint16_t color);
  void reset(void);
  void setRegisters8(uint8_t *ptr, uint8_t n);
  void setRegisters16(uint16_t *ptr, uint8_t n);
  void setRotation(uint8_t x);
  // These methods are public in order for BMP examples to work:
  void setAddrWindow(int x1, int y1, int x2, int y2);
  void pushColors(uint16_t *data, uint8_t len, boolean first);

  uint16_t color565(uint8_t r, uint8_t g, uint8_t b),
      readPixel(int16_t x, int16_t y), readID(void);
  uint32_t readReg(uint8_t r);

private:
  void init(),
  // These items may have previously been defined as macros
  // in pin_magic.h.  If not, function versions are declared:
#ifndef write8
      write8(uint8_t value),
#endif
#ifndef setWriteDir
      setWriteDir(void),
#endif
#ifndef setReadDir
      setReadDir(void),
#endif
#ifndef writeRegister8
      writeRegister8(uint8_t a, uint8_t d),
#endif
#ifndef writeRegister16
      writeRegister16(uint16_t a, uint16_t d),
#endif
      writeRegister24(uint8_t a, uint32_t d),
      writeRegister32(uint8_t a, uint32_t d),
#ifndef writeRegisterPair
      writeRegisterPair(uint8_t aH, uint8_t aL, uint16_t d),
#endif
      setLR(void), flood(uint16_t color, uint32_t len);
  uint8_t driver;

#ifndef read8
  uint8_t read8fn(void);
#define read8isFunctionalized
#endif

#ifndef USE_ADAFRUIT_SHIELD_PINOUT

#ifdef __AVR__
  volatile uint8_t *csPort, *cdPort, *wrPort, *rdPort;
  uint8_t csPinSet, cdPinSet, wrPinSet, rdPinSet, csPinUnset, cdPinUnset,
      wrPinUnset, rdPinUnset, _reset;
#endif
#if defined(__SAM3X8E__)
  Pio *csPort, *cdPort, *wrPort, *rdPort;
  uint32_t csPinSet, cdPinSet, wrPinSet, rdPinSet, csPinUnset, cdPinUnset,
      wrPinUnset, rdPinUnset, _reset;
```

```
#endif

#endif
};

// For compatibility with sketches written for older versions of library.
// Color function name was changed to 'color565' for parity with 2.2" LCD
// library.
#define Color565 color565

#endif
```

pin_magic.h

```
#ifndef _pin_magic_
#define _pin_magic_

// This header file serves two purposes:
//
// 1) Isolate non-portable MCU port- and pin-specific identifiers and
//    operations so the library code itself remains somewhat agnostic
//    (PORTs and pin numbers are always referenced through macros).
//
// 2) GCC doesn't always respect the "inline" keyword, so this is a
//    ham-fisted manner of forcing the issue to minimize function calls.
//    This sometimes makes the library a bit bigger than before, but fast++.
//    However, because they're macros, we need to be SUPER CAREFUL about
//    parameters -- for example, write8(x) may expand to multiple PORT
//    writes that all refer to x, so it needs to be a constant or fixed
//    variable and not something like *ptr++ (which, after macro
//    expansion, may increment the pointer repeatedly and run off into
//    la-la land).  Macros also give us fine-grained control over which
//    operations are inlined on which boards (balancing speed against
//    available program space).

// When using the TFT shield, control and data pins exist in set physical
// locations, but the ports and bitmasks corresponding to each vary among
// boards.  A separate set of pin definitions is given for each supported
// board type.
// When using the TFT breakout board, control pins are configurable but
// the data pins are still fixed -- making every data pin configurable
// would be much too slow.  The data pin layouts are not the same between
// the shield and breakout configurations -- for the latter, pins were
// chosen to keep the tutorial wiring manageable more than making optimal
// use of ports and bitmasks.  So there's a second set of pin definitions
// given for each supported board.

// Shield pin usage:
// LCD Data Bit :    7    6    5    4    3    2    1    0
// Digital pin #:    7    6   13    4   11   10    9    8
// Uno port/pin :  PD7  PD6  PB5  PD4  PB3  PB2  PB1  PB0
// Mega port/pin:  PH4  PH3  PB7  PG5  PB5  PB4  PH6  PH5
// Leo port/pin :  PE6  PD7  PC7  PD4  PB7  PB6  PB5  PB4
// Due port/pin : PC23 PC24 PB27 PC26  PD7 PC29 PC21 PC22
// Breakout pin usage:
// LCD Data Bit :   7   6   5   4   3   2   1   0
// Uno dig. pin :   7   6   5   4   3   2   9   8
// Uno port/pin : PD7 PD6 PD5 PD4 PD3 PD2 PB1 PB0
// Mega dig. pin:  29  28  27  26  25  24  23  22
// Mega port/pin: PA7 PA6 PA5 PA4 PA3 PA2 PA1 PA0 (one contiguous PORT)
// Leo dig. pin :   7   6   5   4   3   2   9   8
// Leo port/pin : PE6 PD7 PC6 PD4 PD0 PD1 PB5 PB4
// Due dig. pin :  40  39  38  37  36  35  34  33
// Due port/pin : PC8 PC7 PC6 PC5 PC4 PC3 PC2 PC1 (one contiguous PORT. -ish…)

// Pixel read operations require a minimum 400 nS delay from RD_ACTIVE
// to polling the input pins.  At 16 MHz, one machine cycle is 62.5 nS.
// This code burns 7 cycles (437.5 nS) doing nothing; the RJMPs are
// equivalent to two NOPs each, final NOP burns the 7th cycle, and the
// last line is a radioactive mutant emoticon.
#define DELAY7                                                              \
  asm volatile("rjmp .+0"                                                   \
               "\n\t"                                                       \
               "rjmp .+0"                                                   \
               "\n\t"                                                       \
               "rjmp .+0"                                                   \
               "\n\t"                                                       \
               "nop"                                                        \
               "\n" ::);

#if defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__) ||            \
    defined(__AVR_ATmega328__) || defined(__AVR_ATmega8__)

// Arduino Uno, Duemilanove, etc.

#ifdef USE_ADAFRUIT_SHIELD_PINOUT

// LCD control lines:
// RD (read), WR (write), CD (command/data), CS (chip select)
#define RD_PORT PORTC /*pin A0 */
#define WR_PORT PORTC /*pin A1 */
#define CD_PORT PORTC /*pin A2 */
#define CS_PORT PORTC /*pin A3 */
#define RD_MASK B00000001
#define WR_MASK B00000010
#define CD_MASK B00000100
#define CS_MASK B00001000

// These are macros for I/O operations...

// Write 8-bit value to LCD data lines
#define write8inline(d)                                                     \
```

```
  {                                                                       \
    PORTD = (PORTD & B00101111) | ((d)&B11010000);                        \
    PORTB = (PORTB & B11010000) | ((d)&B00101111);                        \
    WR_STROBE;                                                            \
  } // STROBEs are defined later

// Read 8-bit value from LCD data lines.  The signle argument
// is a destination variable; this isn't a function and doesn't
// return a value in the conventional sense.
#define read8inline(result)                                               \
  {                                                                       \
    RD_ACTIVE;                                                            \
    DELAY7;                                                               \
    result = (PIND & B11010000) | (PINB & B00101111);                     \
    RD_IDLE;                                                              \
  }

// These set the PORT directions as required before the write and read
// operations.  Because write operations are much more common than reads,
// the data-reading functions in the library code set the PORT(s) to
// input before a read, and restore them back to the write state before
// returning.  This avoids having to set it for output inside every
// drawing method.  The default state has them initialized for writes.
#define setWriteDirInline()                                               \
  {                                                                       \
    DDRD |= B11010000;                                                    \
    DDRB |= B00101111;                                                    \
  }
#define setReadDirInline()                                                \
  {                                                                       \
    DDRD &= ~B11010000;                                                   \
    DDRB &= ~B00101111;                                                   \
  }

#else // Uno w/Breakout board

#define write8inline(d)                                                   \
  {                                                                       \
    PORTD = (PORTD & B00000011) | ((d)&B11111100);                        \
    PORTB = (PORTB & B11111100) | ((d)&B00000011);                        \
    WR_STROBE;                                                            \
  }
#define read8inline(result)                                               \
  {                                                                       \
    RD_ACTIVE;                                                            \
    DELAY7;                                                               \
    result = (PIND & B11111100) | (PINB & B00000011);                     \
    RD_IDLE;                                                              \
  }
#define setWriteDirInline()                                               \
  {                                                                       \
    DDRD |= B11111100;                                                    \
    DDRB |= B00000011;                                                    \
  }
#define setReadDirInline()                                                \
  {                                                                       \
    DDRD &= ~B11111100;                                                   \
    DDRB &= ~B00000011;                                                   \
  }

#endif

// As part of the inline control, macros reference other macros...if any
// of these are left undefined, an equivalent function version (non-inline)
// is declared later.  The Uno has a moderate amount of program space, so
// only write8() is inlined -- that one provides the most performance
// benefit, but unfortunately also generates the most bloat.  This is
// why only certain cases are inlined for each board.
#define write8 write8inline

#elif defined(__AVR_ATmega1281__) || defined(__AVR_ATmega2561__) ||       \
    defined(__AVR_ATmega2560__) || defined(__AVR_ATmega1280__)

// Arduino Mega, ADK, etc.

#ifdef USE_ADAFRUIT_SHIELD_PINOUT

#define RD_PORT PORTF
#define WR_PORT PORTF
#define CD_PORT PORTF
#define CS_PORT PORTF
#define RD_MASK B00000001
#define WR_MASK B00000010
#define CD_MASK B00000100
#define CS_MASK B00001000

#define write8inline(d)                                                   \
  {                                                                       \
    PORTH =                                                               \
        (PORTH & B10000111) | (((d)&B11000000) >> 3) | (((d)&B00000011) << 5); \
```

```
    PORTB = (PORTB & B01001111) | (((d)&B00101100) << 2);                    \
    PORTG = (PORTG & B11011111) | (((d)&B00010000) << 1);                    \
    WR_STROBE;                                                               \
  }
#define read8inline(result)                                                  \
  {                                                                          \
    RD_ACTIVE;                                                               \
    DELAY7;                                                                  \
    result = ((PINH & B00011000) << 3) | ((PINB & B10110000) >> 2) |         \
             ((PING & B00100000) >> 1) | ((PINH & B01100000) >> 5);          \
    RD_IDLE;                                                                 \
  }
#define setWriteDirInline()                                                  \
  {                                                                          \
    DDRH |= B01111000;                                                       \
    DDRB |= B10110000;                                                       \
    DDRG |= B00100000;                                                       \
  }
#define setReadDirInline()                                                   \
  {                                                                          \
    DDRH &= ~B01111000;                                                      \
    DDRB &= ~B10110000;                                                      \
    DDRG &= ~B00100000;                                                      \
  }

#else // Mega w/Breakout board

#define write8inline(d)                                                      \
  {                                                                          \
    PORTA = (d);                                                             \
    WR_STROBE;                                                               \
  }
#define read8inline(result)                                                  \
  {                                                                          \
    RD_ACTIVE;                                                               \
    DELAY7;                                                                  \
    result = PINA;                                                           \
    RD_IDLE;                                                                 \
  }
#define setWriteDirInline() DDRA = 0xff
#define setReadDirInline() DDRA = 0

#endif

// All of the functions are inlined on the Arduino Mega.  When using the
// breakout board, the macro versions aren't appreciably larger than the
// function equivalents, and they're super simple and fast.  When using
// the shield, the macros become pretty complicated...but this board has
// so much code space, the macros are used anyway.  If you need to free
// up program space, some macros can be removed, at a minor cost in speed.
#define write8 write8inline
#define read8 read8inline
#define setWriteDir setWriteDirInline
#define setReadDir setReadDirInline
#define writeRegister8 writeRegister8inline
#define writeRegister16 writeRegister16inline
#define writeRegisterPair writeRegisterPairInline

#elif defined(__AVR_ATmega32U4__)

// Arduino Leonardo

#ifdef USE_ADAFRUIT_SHIELD_PINOUT

#define RD_PORT PORTF
#define WR_PORT PORTF
#define CD_PORT PORTF
#define CS_PORT PORTF
#define RD_MASK B10000000
#define WR_MASK B01000000
#define CD_MASK B00100000
#define CS_MASK B00010000

#define write8inline(d)                                                      \
  {                                                                          \
    PORTE = (PORTE & B10111111) | (((d)&B10000000) >> 1);                    \
    PORTD = (PORTD & B01101111) | (((d)&B01000000) << 1) | ((d)&B00010000);  \
    PORTC = (PORTC & B01111111) | (((d)&B00100000) << 2);                    \
    PORTB = (PORTB & B00001111) | (((d)&B00001111) << 4);                    \
    WR_STROBE;                                                               \
  }
#define read8inline(result)                                                  \
  {                                                                          \
    RD_ACTIVE;                                                               \
    DELAY7;                                                                  \
    result = ((PINE & B01000000) << 1) | ((PIND & B10000000) >> 1) |         \
             ((PINC & B10000000) >> 2) | ((PINB & B11110000) >> 4) |         \
             (PIND & B00010000);                                             \
    RD_IDLE;                                                                 \
  }
```

```
#define setWriteDirInline()                                                  \
  {                                                                          \
    DDRE |= B01000000;                                                       \
    DDRD |= B10010000;                                                       \
    DDRC |= B10000000;                                                       \
    DDRB |= B11110000;                                                       \
  }
#define setReadDirInline()                                                   \
  {                                                                          \
    DDRE &= ~B01000000;                                                      \
    DDRD &= ~B10010000;                                                      \
    DDRC &= ~B10000000;                                                      \
    DDRB &= ~B11110000;                                                      \
  }

#else // Leonardo w/Breakout board

#define write8inline(d)                                                      \
  {                                                                          \
    uint8_t dr1 = (d) >> 1, dl1 = (d) << 1;                                  \
    PORTE = (PORTE & B10111111) | (dr1 & B01000000);                         \
    PORTD = (PORTD & B01101100) | (dl1 & B10000000) | (((d)&B00001000) >> 3) | \
            (dr1 & B00000010) | ((d)&B00010000);                             \
    PORTC = (PORTC & B10111111) | (dl1 & B01000000);                         \
    PORTB = (PORTB & B11001111) | (((d)&B00000011) << 4);                    \
    WR_STROBE;                                                               \
  }
#define read8inline(result)                                                  \
  {                                                                          \
    RD_ACTIVE;                                                               \
    DELAY7;                                                                  \
    result = (((PINE & B01000000) | (PIND & B00000010)) << 1) |              \
             (((PINC & B01000000) | (PIND & B10000000)) >> 1) |              \
             ((PIND & B00000001) << 3) | ((PINB & B00110000) >> 4) |         \
             (PIND & B00010000);                                             \
    RD_IDLE;                                                                 \
  }
#define setWriteDirInline()                                                  \
  {                                                                          \
    DDRE |= B01000000;                                                       \
    DDRD |= B10010011;                                                       \
    DDRC |= B01000000;                                                       \
    DDRB |= B00110000;                                                       \
  }
#define setReadDirInline()                                                   \
  {                                                                          \
    DDRE &= ~B01000000;                                                      \
    DDRD &= ~B10010011;                                                      \
    DDRC &= ~B01000000;                                                      \
    DDRB &= ~B00110000;                                                      \
  }

#endif

// On the Leonardo, only the write8() macro is used -- though even that
// might be excessive given the code size and available program space
// on this board.  You may need to disable this to get any sizable
// program to compile.
#define write8 write8inline

#elif defined(__SAM3X8E__)

// Arduino Due

#ifdef USE_ADAFRUIT_SHIELD_PINOUT

#define RD_PORT PIOA /*pin A0 */
#define WR_PORT PIOA /*pin A1 */
#define CD_PORT PIOA /*pin A2 */
#define CS_PORT PIOA /*pin A3 */
#define RD_MASK 0x00010000
#define WR_MASK 0x01000000
#define CD_MASK 0x00800000
#define CS_MASK 0x00400000

#define write8inline(d)                                                      \
  {                                                                          \
    PIO_Set(PIOD, (((d)&0x08) << (7 - 3)));                                  \
    PIO_Clear(PIOD, (((~d) & 0x08) << (7 - 3)));                             \
    PIO_Set(PIOC, (((d)&0x01) << (22 - 0)) | (((d)&0x02) << (21 - 1)) |      \
                      (((d)&0x04) << (29 - 2)) | (((d)&0x10) << (26 - 4)) |  \
                      (((d)&0x40) << (24 - 6)) | (((d)&0x80) << (23 - 7)));  \
    PIO_Clear(PIOC,                                                          \
              (((~d) & 0x01) << (22 - 0)) | (((~d) & 0x02) << (21 - 1)) |    \
                  (((~d) & 0x04) << (29 - 2)) | (((~d) & 0x10) << (26 - 4)) | \
                  (((~d) & 0x40) << (24 - 6)) | (((~d) & 0x80) << (23 - 7))); \
    PIO_Set(PIOB, (((d)&0x20) << (27 - 5)));                                 \
    PIO_Clear(PIOB, (((~d) & 0x20) << (27 - 5)));                            \
    WR_STROBE;                                                               \
  }
```

```
#define read8inline(result)                                                    \
  {                                                                            \
                                                                               \
    RD_ACTIVE;                                                                 \
    delayMicroseconds(1);                                                      \
    result = (((PIOC->PIO_PDSR & (1 << 23)) >> (23 - 7)) |                     \
              ((PIOC->PIO_PDSR & (1 << 24)) >> (24 - 6)) |                     \
              ((PIOB->PIO_PDSR & (1 << 27)) >> (27 - 5)) |                     \
              ((PIOC->PIO_PDSR & (1 << 26)) >> (26 - 4)) |                     \
              ((PIOD->PIO_PDSR & (1 << 7)) >> (7 - 3)) |                       \
              ((PIOC->PIO_PDSR & (1 << 29)) >> (29 - 2)) |                     \
              ((PIOC->PIO_PDSR & (1 << 21)) >> (21 - 1)) |                     \
              ((PIOC->PIO_PDSR & (1 << 22)) >> (22 - 0)));                     \
    RD_IDLE;                                                                   \
  }

#define setWriteDirInline()                                                    \
  {                                                                            \
    PIOD->PIO_MDDR |= 0x00000080; /*PIOD->PIO_SODR =  0x00000080;*/            \
    PIOD->PIO_OER |= 0x00000080;                                               \
    PIOD->PIO_PER |= 0x00000080;                                               \
    PIOC->PIO_MDDR |= 0x25E00000; /*PIOC->PIO_SODR =  0x25E00000;*/            \
    PIOC->PIO_OER |= 0x25E00000;                                               \
    PIOC->PIO_PER |= 0x25E00000;                                               \
    PIOB->PIO_MDDR |= 0x08000000; /*PIOB->PIO_SODR =  0x08000000;*/            \
    PIOB->PIO_OER |= 0x08000000;                                               \
    PIOB->PIO_PER |= 0x08000000;                                               \
  }

#define setReadDirInline()                                                     \
  {                                                                            \
    pmc_enable_periph_clk(ID_PIOD);                                            \
    pmc_enable_periph_clk(ID_PIOC);                                            \
    pmc_enable_periph_clk(ID_PIOB);                                            \
    PIOD->PIO_PUDR |= 0x00000080;                                              \
    PIOD->PIO_IFDR |= 0x00000080;                                              \
    PIOD->PIO_ODR |= 0x00000080;                                               \
    PIOD->PIO_PER |= 0x00000080;                                               \
    PIOC->PIO_PUDR |= 0x25E00000;                                              \
    PIOC->PIO_IFDR |= 0x25E00000;                                              \
    PIOC->PIO_ODR |= 0x25E00000;                                               \
    PIOC->PIO_PER |= 0x25E00000;                                               \
    PIOB->PIO_PUDR |= 0x08000000;                                              \
    PIOB->PIO_IFDR |= 0x08000000;                                              \
    PIOB->PIO_ODR |= 0x08000000;                                               \
    PIOB->PIO_PER |= 0x08000000;                                               \
  }

// Control signals are ACTIVE LOW (idle is HIGH)
// Command/Data: LOW = command, HIGH = data
// These are single-instruction operations and always inline
#define RD_ACTIVE RD_PORT->PIO_CODR |= RD_MASK
#define RD_IDLE RD_PORT->PIO_SODR |= RD_MASK
#define WR_ACTIVE WR_PORT->PIO_CODR |= WR_MASK
#define WR_IDLE WR_PORT->PIO_SODR |= WR_MASK
#define CD_COMMAND CD_PORT->PIO_CODR |= CD_MASK
#define CD_DATA CD_PORT->PIO_SODR |= CD_MASK
#define CS_ACTIVE CS_PORT->PIO_CODR |= CS_MASK
#define CS_IDLE CS_PORT->PIO_SODR |= CS_MASK

#else // Due w/Breakout board

#define write8inline(d)                                                        \
  {                                                                            \
    PIO_Set(PIOC, (((d)&0xFF) << 1));                                          \
    PIO_Clear(PIOC, (((~d) & 0xFF) << 1));                                     \
    WR_STROBE;                                                                 \
  }

#define read8inline(result)                                                    \
  {                                                                            \
    RD_ACTIVE;                                                                 \
    delayMicroseconds(1);                                                      \
    result = ((PIOC->PIO_PDSR & 0x1FE) >> 1);                                  \
    RD_IDLE;                                                                   \
  }

#define setWriteDirInline()                                                    \
  {                                                                            \
    PIOC->PIO_MDDR |= 0x000001FE; /*PIOC->PIO_SODR |=  0x000001FE;*/           \
    PIOC->PIO_OER |= 0x000001FE;                                               \
    PIOC->PIO_PER |= 0x000001FE;                                               \
  }

#define setReadDirInline()                                                     \
  {                                                                            \
    pmc_enable_periph_clk(ID_PIOC);                                            \
    PIOC->PIO_PUDR |= 0x000001FE;                                              \
    PIOC->PIO_IFDR |= 0x000001FE;                                              \
```

```
    PIOC->PIO_ODR |= 0x000001FE;                                      \
    PIOC->PIO_PER |= 0x000001FE;                                      \
  }

// When using the TFT breakout board, control pins are configurable.
#define RD_ACTIVE rdPort->PIO_CODR |= rdPinSet  // PIO_Clear(rdPort, rdPinSet)
#define RD_IDLE rdPort->PIO_SODR |= rdPinSet    // PIO_Set(rdPort, rdPinSet)
#define WR_ACTIVE wrPort->PIO_CODR |= wrPinSet  // PIO_Clear(wrPort, wrPinSet)
#define WR_IDLE wrPort->PIO_SODR |= wrPinSet    // PIO_Set(wrPort, wrPinSet)
#define CD_COMMAND cdPort->PIO_CODR |= cdPinSet // PIO_Clear(cdPort, cdPinSet)
#define CD_DATA cdPort->PIO_SODR |= cdPinSet    // PIO_Set(cdPort, cdPinSet)
#define CS_ACTIVE csPort->PIO_CODR |= csPinSet  // PIO_Clear(csPort, csPinSet)
#define CS_IDLE csPort->PIO_SODR |= csPinSet    // PIO_Set(csPort, csPinSet)

#endif

#else

#error "Board type unsupported / not recognized"

#endif

#if !defined(__SAM3X8E__)
// Stuff common to all Arduino AVR board types:

#ifdef USE_ADAFRUIT_SHIELD_PINOUT

// Control signals are ACTIVE LOW (idle is HIGH)
// Command/Data: LOW = command, HIGH = data
// These are single-instruction operations and always inline
#define RD_ACTIVE RD_PORT &= ~RD_MASK
#define RD_IDLE RD_PORT |= RD_MASK
#define WR_ACTIVE WR_PORT &= ~WR_MASK
#define WR_IDLE WR_PORT |= WR_MASK
#define CD_COMMAND CD_PORT &= ~CD_MASK
#define CD_DATA CD_PORT |= CD_MASK
#define CS_ACTIVE CS_PORT &= ~CS_MASK
#define CS_IDLE CS_PORT |= CS_MASK

#else // Breakout board

// When using the TFT breakout board, control pins are configurable.
#define RD_ACTIVE *rdPort &= rdPinUnset
#define RD_IDLE *rdPort |= rdPinSet
#define WR_ACTIVE *wrPort &= wrPinUnset
#define WR_IDLE *wrPort |= wrPinSet
#define CD_COMMAND *cdPort &= cdPinUnset
#define CD_DATA *cdPort |= cdPinSet
#define CS_ACTIVE *csPort &= csPinUnset
#define CS_IDLE *csPort |= csPinSet

#endif
#endif

// Data write strobe, ~2 instructions and always inline
#define WR_STROBE                                                     \
  {                                                                   \
    WR_ACTIVE;                                                        \
    WR_IDLE;                                                          \
  }

// These higher-level operations are usually functionalized,
// except on Mega where's there's gobs and gobs of program space.

// Set value of TFT register: 8-bit address, 8-bit value
#define writeRegister8inline(a, d)                                    \
  {                                                                   \
    CD_COMMAND;                                                       \
    write8(a);                                                        \
    CD_DATA;                                                          \
    write8(d);                                                        \
  }

// Set value of TFT register: 16-bit address, 16-bit value
// See notes at top about macro expansion, hence hi & lo temp vars
#define writeRegister16inline(a, d)                                   \
  {                                                                   \
    uint8_t hi, lo;                                                   \
    hi = (a) >> 8;                                                    \
    lo = (a);                                                         \
    CD_COMMAND;                                                       \
    write8(hi);                                                       \
    write8(lo);                                                       \
    hi = (d) >> 8;                                                    \
    lo = (d);                                                         \
    CD_DATA;                                                          \
    write8(hi);                                                       \
    write8(lo);                                                       \
  }
```

```
// Set value of 2 TFT registers: Two 8-bit addresses (hi & lo), 16-bit value
#define writeRegisterPairInline(aH, aL, d)                                \
  {                                                                       \
    uint8_t hi = (d) >> 8, lo = (d);                                      \
    CD_COMMAND;                                                           \
    write8(aH);                                                           \
    CD_DATA;                                                              \
    write8(hi);                                                           \
    CD_COMMAND;                                                           \
    write8(aL);                                                           \
    CD_DATA;                                                              \
    write8(lo);                                                           \
  }

#endif // _pin_magic_
```

registers.h

```
// Register names from Peter Barrett's Microtouch code
#define ILI932X_START_OSC 0x00
#define ILI932X_DRIV_OUT_CTRL 0x01
#define ILI932X_DRIV_WAV_CTRL 0x02
#define ILI932X_ENTRY_MOD 0x03
#define ILI932X_RESIZE_CTRL 0x04
#define ILI932X_DISP_CTRL1 0x07
#define ILI932X_DISP_CTRL2 0x08
#define ILI932X_DISP_CTRL3 0x09
#define ILI932X_DISP_CTRL4 0x0A
#define ILI932X_RGB_DISP_IF_CTRL1 0x0C
#define ILI932X_FRM_MARKER_POS 0x0D
#define ILI932X_RGB_DISP_IF_CTRL2 0x0F
#define ILI932X_POW_CTRL1 0x10
#define ILI932X_POW_CTRL2 0x11
#define ILI932X_POW_CTRL3 0x12
#define ILI932X_POW_CTRL4 0x13
#define ILI932X_GRAM_HOR_AD 0x20
#define ILI932X_GRAM_VER_AD 0x21
#define ILI932X_RW_GRAM 0x22
#define ILI932X_POW_CTRL7 0x29
#define ILI932X_FRM_RATE_COL_CTRL 0x2B
#define ILI932X_GAMMA_CTRL1 0x30
#define ILI932X_GAMMA_CTRL2 0x31
#define ILI932X_GAMMA_CTRL3 0x32
#define ILI932X_GAMMA_CTRL4 0x35
#define ILI932X_GAMMA_CTRL5 0x36
#define ILI932X_GAMMA_CTRL6 0x37
#define ILI932X_GAMMA_CTRL7 0x38
#define ILI932X_GAMMA_CTRL8 0x39
#define ILI932X_GAMMA_CTRL9 0x3C
#define ILI932X_GAMMA_CTRL10 0x3D
#define ILI932X_HOR_START_AD 0x50
#define ILI932X_HOR_END_AD 0x51
#define ILI932X_VER_START_AD 0x52
#define ILI932X_VER_END_AD 0x53
#define ILI932X_GATE_SCAN_CTRL1 0x60
#define ILI932X_GATE_SCAN_CTRL2 0x61
#define ILI932X_GATE_SCAN_CTRL3 0x6A
#define ILI932X_PART_IMG1_DISP_POS 0x80
#define ILI932X_PART_IMG1_START_AD 0x81
#define ILI932X_PART_IMG1_END_AD 0x82
#define ILI932X_PART_IMG2_DISP_POS 0x83
#define ILI932X_PART_IMG2_START_AD 0x84
#define ILI932X_PART_IMG2_END_AD 0x85
#define ILI932X_PANEL_IF_CTRL1 0x90
#define ILI932X_PANEL_IF_CTRL2 0x92
#define ILI932X_PANEL_IF_CTRL3 0x93
#define ILI932X_PANEL_IF_CTRL4 0x95
#define ILI932X_PANEL_IF_CTRL5 0x97
#define ILI932X_PANEL_IF_CTRL6 0x98

#define HX8347G_COLADDRSTART_HI 0x02
#define HX8347G_COLADDRSTART_LO 0x03
#define HX8347G_COLADDREND_HI 0x04
#define HX8347G_COLADDREND_LO 0x05
#define HX8347G_ROWADDRSTART_HI 0x06
#define HX8347G_ROWADDRSTART_LO 0x07
#define HX8347G_ROWADDREND_HI 0x08
#define HX8347G_ROWADDREND_LO 0x09
#define HX8347G_MEMACCESS 0x16

#define ILI9341_SOFTRESET 0x01
#define ILI9341_SLEEPIN 0x10
#define ILI9341_SLEEPOUT 0x11
#define ILI9341_NORMALDISP 0x13
#define ILI9341_INVERTOFF 0x20
#define ILI9341_INVERTON 0x21
#define ILI9341_GAMMASET 0x26
#define ILI9341_DISPLAYOFF 0x28
#define ILI9341_DISPLAYON 0x29
#define ILI9341_COLADDRSET 0x2A
#define ILI9341_PAGEADDRSET 0x2B
#define ILI9341_MEMORYWRITE 0x2C
#define ILI9341_PIXELFORMAT 0x3A
#define ILI9341_FRAMECONTROL 0xB1
#define ILI9341_DISPLAYFUNC 0xB6
#define ILI9341_ENTRYMODE 0xB7
#define ILI9341_POWERCONTROL1 0xC0
#define ILI9341_POWERCONTROL2 0xC1
#define ILI9341_VCOMCONTROL1 0xC5
#define ILI9341_VCOMCONTROL2 0xC7
#define ILI9341_MEMCONTROL 0x36
#define ILI9341_MADCTL 0x36

#define ILI9341_MADCTL_MY 0x80
#define ILI9341_MADCTL_MX 0x40
#define ILI9341_MADCTL_MV 0x20
```

```
#define ILI9341_MADCTL_ML 0x10
#define ILI9341_MADCTL_RGB 0x00
#define ILI9341_MADCTL_BGR 0x08
#define ILI9341_MADCTL_MH 0x04

#define HX8357_NOP 0x00
#define HX8357_SWRESET 0x01
#define HX8357_RDDID 0x04
#define HX8357_RDDST 0x09

#define HX8357B_RDPOWMODE 0x0A
#define HX8357B_RDMADCTL 0x0B
#define HX8357B_RDCOLMOD 0x0C
#define HX8357B_RDDIM 0x0D
#define HX8357B_RDDSDR 0x0F

#define HX8357_SLPIN 0x10
#define HX8357_SLPOUT 0x11
#define HX8357B_PTLON 0x12
#define HX8357B_NORON 0x13

#define HX8357_INVOFF 0x20
#define HX8357_INVON 0x21
#define HX8357_DISPOFF 0x28
#define HX8357_DISPON 0x29

#define HX8357_CASET 0x2A
#define HX8357_PASET 0x2B
#define HX8357_RAMWR 0x2C
#define HX8357_RAMRD 0x2E

#define HX8357B_PTLAR 0x30
#define HX8357_TEON 0x35
#define HX8357_TEARLINE 0x44
#define HX8357_MADCTL 0x36
#define HX8357_COLMOD 0x3A

#define HX8357_SETOSC 0xB0
#define HX8357_SETPWR1 0xB1
#define HX8357B_SETDISPLAY 0xB2
#define HX8357_SETRGB 0xB3
#define HX8357D_SETCOM 0xB6

#define HX8357B_SETDISPMODE 0xB4
#define HX8357D_SETCYC 0xB4
#define HX8357B_SETOTP 0xB7
#define HX8357D_SETC 0xB9

#define HX8357B_SET_PANEL_DRIVING 0xC0
#define HX8357D_SETSTBA 0xC0
#define HX8357B_SETDGC 0xC1
#define HX8357B_SETID 0xC3
#define HX8357B_SETDDB 0xC4
#define HX8357B_SETDISPLAYFRAME 0xC5
#define HX8357B_GAMMASET 0xC8
#define HX8357B_SETCABC 0xC9
#define HX8357_SETPANEL 0xCC

#define HX8357B_SETPOWER 0xD0
#define HX8357B_SETVCOM 0xD1
#define HX8357B_SETPWRNORMAL 0xD2

#define HX8357B_RDID1 0xDA
#define HX8357B_RDID2 0xDB
#define HX8357B_RDID3 0xDC
#define HX8357B_RDID4 0xDD

#define HX8357D_SETGAMMA 0xE0

#define HX8357B_SETGAMMA 0xC8
#define HX8357B_SETPANELRELATED 0xE9

#define HX8357B_MADCTL_MY 0x80
#define HX8357B_MADCTL_MX 0x40
#define HX8357B_MADCTL_MV 0x20
#define HX8357B_MADCTL_ML 0x10
#define HX8357B_MADCTL_RGB 0x00
#define HX8357B_MADCTL_BGR 0x08
#define HX8357B_MADCTL_MH 0x04
```

Graphicstest

```
// IMPORTANT: Adafruit_TFTLCD LIBRARY MUST BE SPECIFICALLY
// CONFIGURED FOR EITHER THE TFT SHIELD OR THE BREAKOUT BOARD.
// SEE RELEVANT COMMENTS IN Adafruit_TFTLCD.h FOR SETUP.

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_TFTLCD.h> // Hardware-specific library

// The control pins for the LCD can be assigned to any digital or
// analog pins...but we'll use the analog pins as this allows us to
// double up the pins with the touch screen (see the TFT paint example).
#define LCD_CS A3 // Chip Select goes to Analog 3
#define LCD_CD A2 // Command/Data goes to Analog 2
#define LCD_WR A1 // LCD Write goes to Analog 1
#define LCD_RD A0 // LCD Read goes to Analog 0

#define LCD_RESET A4 // Can alternately just connect to Arduino's reset pin

// When using the BREAKOUT BOARD only, use these 8 data lines to the LCD:
// For the Arduino Uno, Duemilanove, Diecimila, etc.:
//   D0 connects to digital pin 8  (Notice these are
//   D1 connects to digital pin 9   NOT in order!)
//   D2 connects to digital pin 2
//   D3 connects to digital pin 3
//   D4 connects to digital pin 4
//   D5 connects to digital pin 5
//   D6 connects to digital pin 6
//   D7 connects to digital pin 7
// For the Arduino Mega, use digital pins 22 through 29
// (on the 2-row header at the end of the board).

// Assign human-readable names to some common 16-bit color values:
#define  BLACK   0x0000
#define  BLUE    0x001F
#define  RED     0xF800
#define  GREEN   0x07E0
#define CYAN    0x07FF
#define MAGENTA 0xF81F
#define YELLOW  0xFFE0
#define WHITE   0xFFFF

Adafruit_TFTLCD tft(LCD_CS, LCD_CD, LCD_WR, LCD_RD, LCD_RESET);
// If using the shield, all control and data lines are fixed, and
// a simpler declaration can optionally be used:
// Adafruit_TFTLCD tft;

void setup(void) {
  Serial.begin(9600);
  Serial.println(F("TFT LCD test"));

#ifdef USE_ADAFRUIT_SHIELD_PINOUT
  Serial.println(F("Using Adafruit 2.8\" TFT Arduino Shield Pinout"));
#else
  Serial.println(F("Using Adafruit 2.8\" TFT Breakout Board Pinout"));
#endif

  Serial.print("TFT size is "); Serial.print(tft.width()); Serial.print("x"); Serial.println(tft.height());

  tft.reset();

  uint16_t identifier = tft.readID();

  if(identifier == 0x9325) {
    Serial.println(F("Found ILI9325 LCD driver"));
  } else if(identifier == 0x9328) {
    Serial.println(F("Found ILI9328 LCD driver"));
  } else if(identifier == 0x7575) {
    Serial.println(F("Found HX8347G LCD driver"));
  } else if(identifier == 0x9341) {
    Serial.println(F("Found ILI9341 LCD driver"));
  } else if(identifier == 0x8357) {
    Serial.println(F("Found HX8357D LCD driver"));
  } else {
    Serial.print(F("Unknown LCD driver chip: "));
    Serial.println(identifier, HEX);
    Serial.println(F("If using the Adafruit 2.8\" TFT Arduino shield, the line:"));
    Serial.println(F("  #define USE_ADAFRUIT_SHIELD_PINOUT"));
    Serial.println(F("should appear in the library header (Adafruit_TFT.h)."));
    Serial.println(F("If using the breakout board, it should NOT be #defined!"));
    Serial.println(F("Also if using the breakout, double-check that all wiring"));
    Serial.println(F("matches the tutorial."));
    return;
  }

  tft.begin(identifier);

  Serial.println(F("Benchmark                Time (microseconds)"));

  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(CYAN));
  delay(500);

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

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

  Serial.print(F("Rectangles (filled)      "));
  Serial.println(testFilledRects(YELLOW, MAGENTA));
  delay(500);

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

  Serial.print(F("Circles (outline)        "));
  Serial.println(testCircles(10, 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(2000);
  }
}

unsigned long testFillScreen() {
  unsigned long start = micros();
  tft.fillScreen(BLACK);
  tft.fillScreen(RED);
  tft.fillScreen(GREEN);
  tft.fillScreen(BLUE);
  tft.fillScreen(BLACK);
  return micros() - start;
}

unsigned long testText() {
  tft.fillScreen(BLACK);
  unsigned long start = micros();
  tft.setCursor(0, 0);
  tft.setTextColor(WHITE);  tft.setTextSize(1);
  tft.println("Hello World!");
  tft.setTextColor(YELLOW); tft.setTextSize(2);
  tft.println(1234.56);
  tft.setTextColor(RED);    tft.setTextSize(3);
  tft.println(0xDEADBEEF, HEX);
  tft.println();
  tft.setTextColor(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(BLACK);

  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

  tft.fillScreen(BLACK);

  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;

  tft.fillScreen(BLACK);

  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;

  tft.fillScreen(BLACK);

  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);

  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(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(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(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);
  }

  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(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(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(0, 0, 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(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, i));
    t += micros() - start;
    tft.drawTriangle(cx, cy - i, cx - i, cy + i, cx + i, cy + i,
      tft.color565(i, i, 0));
  }

  return t;
}

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

```
  tft.fillScreen(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(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));
  }

  return micros() - start;
}
```

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;
  }
}
```

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
```