```
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
```

```
#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;
}
```