/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

  Arduino TFT graphics library targeted at 32 bit

  processors such as ESP32, ESP8266 and STM32.

  This is a stand-alone library that contains the

  hardware driver, the graphics functions and the

  proportional fonts.

  The larger fonts are Run Length Encoded to reduce their

  size.

  Created by Bodmer 2/12/16

  Last update by Bodmer 20/03/20

 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "TFT\_eSPI.h"

#if defined (ESP32)

  #if defined(CONFIG\_IDF\_TARGET\_ESP32S3)

    #include "Processors/TFT\_eSPI\_ESP32\_S3.c" // Tested with SPI and 8 bit parallel

  #elif defined(CONFIG\_IDF\_TARGET\_ESP32C3)

    #include "Processors/TFT\_eSPI\_ESP32\_C3.c" // Tested with SPI (8 bit parallel will probably work too!)

  #else

    #include "Processors/TFT\_eSPI\_ESP32.c"

  #endif

#elif defined (ARDUINO\_ARCH\_ESP8266)

  #include "Processors/TFT\_eSPI\_ESP8266.c"

#elif defined (STM32) // (\_VARIANT\_ARDUINO\_STM32\_) stm32\_def.h

  #include "Processors/TFT\_eSPI\_STM32.c"

#elif defined (ARDUINO\_ARCH\_RP2040)  || defined (ARDUINO\_ARCH\_MBED) // Raspberry Pi Pico

  #include "Processors/TFT\_eSPI\_RP2040.c"

#else

  #include "Processors/TFT\_eSPI\_Generic.c"

#endif

#ifndef SPI\_BUSY\_CHECK

  #define SPI\_BUSY\_CHECK

#endif

// Clipping macro for pushImage

#define PI\_CLIP                                        \

  if (\_vpOoB) return;                                  \

  x+= \_xDatum;                                         \

  y+= \_yDatum;                                         \

                                                       \

  if ((x >= \_vpW) || (y >= \_vpH)) return;              \

                                                       \

  int32\_t dx = 0;                                      \

  int32\_t dy = 0;                                      \

  int32\_t dw = w;                                      \

  int32\_t dh = h;                                      \

                                                       \

  if (x < \_vpX) { dx = \_vpX - x; dw -= dx; x = \_vpX; } \

  if (y < \_vpY) { dy = \_vpY - y; dh -= dy; y = \_vpY; } \

                                                       \

  if ((x + dw) > \_vpW ) dw = \_vpW - x;                 \

  if ((y + dh) > \_vpH ) dh = \_vpH - y;                 \

                                                       \

  if (dw < 1 || dh < 1) return;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           Legacy - deprecated

\*\* Description:             Start/end transaction

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

  void TFT\_eSPI::spi\_begin()       {begin\_tft\_write();}

  void TFT\_eSPI::spi\_end()         {  end\_tft\_write();}

  void TFT\_eSPI::spi\_begin\_read()  {begin\_tft\_read(); }

  void TFT\_eSPI::spi\_end\_read()    {  end\_tft\_read(); }

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           begin\_tft\_write (was called spi\_begin)

\*\* Description:             Start SPI transaction for writes and select TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline void TFT\_eSPI::begin\_tft\_write(void){

  if (locked) {

    locked = false; // Flag to show SPI access now unlocked

#if defined (SPI\_HAS\_TRANSACTION) && defined (SUPPORT\_TRANSACTIONS) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

    spi.beginTransaction(SPISettings(SPI\_FREQUENCY, MSBFIRST, TFT\_SPI\_MODE));

#endif

    CS\_L;

    SET\_BUS\_WRITE\_MODE;  // Some processors (e.g. ESP32) allow recycling the tx buffer when rx is not used

  }

}

// Non-inlined version to permit override

void TFT\_eSPI::begin\_nin\_write(void){

  if (locked) {

    locked = false; // Flag to show SPI access now unlocked

#if defined (SPI\_HAS\_TRANSACTION) && defined (SUPPORT\_TRANSACTIONS) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

    spi.beginTransaction(SPISettings(SPI\_FREQUENCY, MSBFIRST, TFT\_SPI\_MODE));

#endif

    CS\_L;

    SET\_BUS\_WRITE\_MODE;  // Some processors (e.g. ESP32) allow recycling the tx buffer when rx is not used

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           end\_tft\_write (was called spi\_end)

\*\* Description:             End transaction for write and deselect TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline void TFT\_eSPI::end\_tft\_write(void){

  if(!inTransaction) {      // Flag to stop ending transaction during multiple graphics calls

    if (!locked) {          // Locked when beginTransaction has been called

      locked = true;        // Flag to show SPI access now locked

      SPI\_BUSY\_CHECK;       // Check send complete and clean out unused rx data

      CS\_H;

      SET\_BUS\_READ\_MODE;    // In case bus has been configured for tx only

#if defined (SPI\_HAS\_TRANSACTION) && defined (SUPPORT\_TRANSACTIONS) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

      spi.endTransaction();

#endif

    }

  }

}

// Non-inlined version to permit override

inline void TFT\_eSPI::end\_nin\_write(void){

  if(!inTransaction) {      // Flag to stop ending transaction during multiple graphics calls

    if (!locked) {          // Locked when beginTransaction has been called

      locked = true;        // Flag to show SPI access now locked

      SPI\_BUSY\_CHECK;       // Check send complete and clean out unused rx data

      CS\_H;

      SET\_BUS\_READ\_MODE;    // In case SPI has been configured for tx only

#if defined (SPI\_HAS\_TRANSACTION) && defined (SUPPORT\_TRANSACTIONS) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

      spi.endTransaction();

#endif

    }

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           begin\_tft\_read  (was called spi\_begin\_read)

\*\* Description:             Start transaction for reads and select TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Reads require a lower SPI clock rate than writes

inline void TFT\_eSPI::begin\_tft\_read(void){

  DMA\_BUSY\_CHECK; // Wait for any DMA transfer to complete before changing SPI settings

#if defined (SPI\_HAS\_TRANSACTION) && defined (SUPPORT\_TRANSACTIONS) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

  if (locked) {

    locked = false;

    spi.beginTransaction(SPISettings(SPI\_READ\_FREQUENCY, MSBFIRST, TFT\_SPI\_MODE));

    CS\_L;

  }

#else

  #if !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

    spi.setFrequency(SPI\_READ\_FREQUENCY);

  #endif

   CS\_L;

#endif

  SET\_BUS\_READ\_MODE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           end\_tft\_read (was called spi\_end\_read)

\*\* Description:             End transaction for reads and deselect TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline void TFT\_eSPI::end\_tft\_read(void){

#if defined (SPI\_HAS\_TRANSACTION) && defined (SUPPORT\_TRANSACTIONS) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

  if(!inTransaction) {

    if (!locked) {

      locked = true;

      CS\_H;

      spi.endTransaction();

    }

  }

#else

  #if !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

    spi.setFrequency(SPI\_FREQUENCY);

  #endif

   if(!inTransaction) {CS\_H;}

#endif

  SET\_BUS\_WRITE\_MODE;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setViewport

\*\* Description:             Set the clipping region for the TFT screen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setViewport(int32\_t x, int32\_t y, int32\_t w, int32\_t h, bool vpDatum)

{

  // Viewport metrics (not clipped)

  \_xDatum  = x; // Datum x position in screen coordinates

  \_yDatum  = y; // Datum y position in screen coordinates

  \_xWidth  = w; // Viewport width

  \_yHeight = h; // Viewport height

  // Full size default viewport

  \_vpDatum = false; // Datum is at top left corner of screen (true = top left of viewport)

  \_vpOoB   = false; // Out of Bounds flag (true is all of viewport is off screen)

  \_vpX = 0;         // Viewport top left corner x coordinate

  \_vpY = 0;         // Viewport top left corner y coordinate

  \_vpW = width();   // Equivalent of TFT width  (Nb: viewport right edge coord + 1)

  \_vpH = height();  // Equivalent of TFT height (Nb: viewport bottom edge coord + 1)

  // Clip viewport to screen area

  if (x<0) { w += x; x = 0; }

  if (y<0) { h += y; y = 0; }

  if ((x + w) > width() ) { w = width()  - x; }

  if ((y + h) > height() ) { h = height() - y; }

  //Serial.print(" x=");Serial.print( x);Serial.print(", y=");Serial.print( y);

  //Serial.print(", w=");Serial.print(w);Serial.print(", h=");Serial.println(h);

  // Check if viewport is entirely out of bounds

  if (w < 1 || h < 1)

  {

    // Set default values and Out of Bounds flag in case of error

    \_xDatum = 0;

    \_yDatum = 0;

    \_xWidth  = width();

    \_yHeight = height();

    \_vpOoB = true;      // Set Out of Bounds flag to inhibit all drawing

    return;

  }

  if (!vpDatum)

  {

    \_xDatum = 0; // Reset to top left of screen if not using a viewport datum

    \_yDatum = 0;

    \_xWidth  = width();

    \_yHeight = height();

  }

  // Store the clipped screen viewport metrics and datum position

  \_vpX = x;

  \_vpY = y;

  \_vpW = x + w;

  \_vpH = y + h;

  \_vpDatum = vpDatum;

  //Serial.print(" \_xDatum=");Serial.print( \_xDatum);Serial.print(", \_yDatum=");Serial.print( \_yDatum);

  //Serial.print(", \_xWidth=");Serial.print(\_xWidth);Serial.print(", \_yHeight=");Serial.println(\_yHeight);

  //Serial.print(" \_vpX=");Serial.print( \_vpX);Serial.print(", \_vpY=");Serial.print( \_vpY);

  //Serial.print(", \_vpW=");Serial.print(\_vpW);Serial.print(", \_vpH=");Serial.println(\_vpH);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           checkViewport

\*\* Description:             Check if any part of specified area is visible in viewport

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Note: Setting w and h to 1 will check if coordinate x,y is in area

bool TFT\_eSPI::checkViewport(int32\_t x, int32\_t y, int32\_t w, int32\_t h)

{

  if (\_vpOoB) return false;

  x+= \_xDatum;

  y+= \_yDatum;

  if ((x >= \_vpW) || (y >= \_vpH)) return false;

  int32\_t dx = 0;

  int32\_t dy = 0;

  int32\_t dw = w;

  int32\_t dh = h;

  if (x < \_vpX) { dx = \_vpX - x; dw -= dx; x = \_vpX; }

  if (y < \_vpY) { dy = \_vpY - y; dh -= dy; y = \_vpY; }

  if ((x + dw) > \_vpW ) dw = \_vpW - x;

  if ((y + dh) > \_vpH ) dh = \_vpH - y;

  if (dw < 1 || dh < 1) return false;

  return true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           resetViewport

\*\* Description:             Reset viewport to whole TFT screen, datum at 0,0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::resetViewport(void)

{

  // Reset viewport to the whole screen (or sprite) area

  \_vpDatum = false;

  \_vpOoB   = false;

  \_xDatum = 0;

  \_yDatum = 0;

  \_vpX = 0;

  \_vpY = 0;

  \_vpW = width();

  \_vpH = height();

  \_xWidth  = width();

  \_yHeight = height();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getViewportX

\*\* Description:             Get x position of the viewport datum

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int32\_t  TFT\_eSPI::getViewportX(void)

{

  return \_xDatum;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getViewportY

\*\* Description:             Get y position of the viewport datum

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int32\_t  TFT\_eSPI::getViewportY(void)

{

  return \_yDatum;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getViewportWidth

\*\* Description:             Get width of the viewport

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int32\_t TFT\_eSPI::getViewportWidth(void)

{

  return \_xWidth;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getViewportHeight

\*\* Description:             Get height of the viewport

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int32\_t TFT\_eSPI::getViewportHeight(void)

{

  return \_yHeight;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getViewportDatum

\*\* Description:             Get datum flag of the viewport (true = viewport corner)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool  TFT\_eSPI::getViewportDatum(void)

{

  return \_vpDatum;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           frameViewport

\*\* Description:             Draw a frame inside or outside the viewport of width w

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::frameViewport(uint16\_t color, int32\_t w)

{

  // Save datum position

  bool \_dT = \_vpDatum;

  // If w is positive the frame is drawn inside the viewport

  // a large positive width will clear the screen inside the viewport

  if (w>0)

  {

    // Set vpDatum true to simplify coordinate derivation

    \_vpDatum = true;

    fillRect(0, 0, \_vpW - \_vpX, w, color);                // Top

    fillRect(0, w, w, \_vpH - \_vpY - w - w, color);        // Left

    fillRect(\_xWidth - w, w, w, \_yHeight - w - w, color); // Right

    fillRect(0, \_yHeight - w, \_xWidth, w, color);         // Bottom

  }

  else

  // If w is negative the frame is drawn outside the viewport

  // a large negative width will clear the screen outside the viewport

  {

    w = -w;

    // Save old values

    int32\_t \_xT = \_vpX; \_vpX = 0;

    int32\_t \_yT = \_vpY; \_vpY = 0;

    int32\_t \_wT = \_vpW;

    int32\_t \_hT = \_vpH;

    // Set vpDatum false so frame can be drawn outside window

    \_vpDatum = false; // When false the full width and height is accessed

    \_vpH = height();

    \_vpW = width();

    // Draw frame

    fillRect(\_xT - w - \_xDatum, \_yT - w - \_yDatum, \_wT - \_xT + w + w, w, color); // Top

    fillRect(\_xT - w - \_xDatum, \_yT - \_yDatum, w, \_hT - \_yT, color);             // Left

    fillRect(\_wT - \_xDatum, \_yT - \_yDatum, w, \_hT - \_yT, color);                 // Right

    fillRect(\_xT - w - \_xDatum, \_hT - \_yDatum, \_wT - \_xT + w + w, w, color);     // Bottom

    // Restore old values

    \_vpX = \_xT;

    \_vpY = \_yT;

    \_vpW = \_wT;

    \_vpH = \_hT;

  }

  // Restore vpDatum

  \_vpDatum = \_dT;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           clipAddrWindow

\*\* Description:             Clip address window x,y,w,h to screen and viewport

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool TFT\_eSPI::clipAddrWindow(int32\_t \*x, int32\_t \*y, int32\_t \*w, int32\_t \*h)

{

  if (\_vpOoB) return false; // Area is outside of viewport

  \*x+= \_xDatum;

  \*y+= \_yDatum;

  if ((\*x >= \_vpW) || (\*y >= \_vpH)) return false;  // Area is outside of viewport

  // Crop drawing area bounds

  if (\*x < \_vpX) { \*w -= \_vpX - \*x; \*x = \_vpX; }

  if (\*y < \_vpY) { \*h -= \_vpY - \*y; \*y = \_vpY; }

  if ((\*x + \*w) > \_vpW ) \*w = \_vpW - \*x;

  if ((\*y + \*h) > \_vpH ) \*h = \_vpH - \*y;

  if (\*w < 1 || \*h < 1) return false; // No area is inside viewport

  return true;  // Area is wholly or partially inside viewport

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           clipWindow

\*\* Description:             Clip window xs,yx,xe,ye to screen and viewport

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool TFT\_eSPI::clipWindow(int32\_t \*xs, int32\_t \*ys, int32\_t \*xe, int32\_t \*ye)

{

  if (\_vpOoB) return false; // Area is outside of viewport

  \*xs+= \_xDatum;

  \*ys+= \_yDatum;

  \*xe+= \_xDatum;

  \*ye+= \_yDatum;

  if ((\*xs >= \_vpW) || (\*ys >= \_vpH)) return false;  // Area is outside of viewport

  if ((\*xe <  \_vpX) || (\*ye <  \_vpY)) return false;  // Area is outside of viewport

  // Crop drawing area bounds

  if (\*xs < \_vpX) \*xs = \_vpX;

  if (\*ys < \_vpY) \*ys = \_vpY;

  if (\*xe > \_vpW) \*xe = \_vpW - 1;

  if (\*ye > \_vpH) \*ye = \_vpH - 1;

  return true;  // Area is wholly or partially inside viewport

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           TFT\_eSPI

\*\* Description:             Constructor , we must use hardware SPI pins

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

TFT\_eSPI::TFT\_eSPI(int16\_t w, int16\_t h)

{

  \_init\_width  = \_width  = w; // Set by specific xxxxx\_Defines.h file or by users sketch

  \_init\_height = \_height = h; // Set by specific xxxxx\_Defines.h file or by users sketch

  // Reset the viewport to the whole screen

  resetViewport();

  rotation  = 0;

  cursor\_y  = cursor\_x  = last\_cursor\_x = bg\_cursor\_x = 0;

  textfont  = 1;

  textsize  = 1;

  textcolor   = bitmap\_fg = 0xFFFF; // White

  textbgcolor = bitmap\_bg = 0x0000; // Black

  padX        = 0;                  // No padding

  \_fillbg    = false;   // Smooth font only at the moment, force text background fill

  isDigits   = false;   // No bounding box adjustment

  textwrapX  = true;    // Wrap text at end of line when using print stream

  textwrapY  = false;   // Wrap text at bottom of screen when using print stream

  textdatum = TL\_DATUM; // Top Left text alignment is default

  fontsloaded = 0;

  \_swapBytes = false;   // Do not swap colour bytes by default

  locked = true;           // Transaction mutex lock flag to ensure begin/endTranaction pairing

  inTransaction = false;   // Flag to prevent multiple sequential functions to keep bus access open

  lockTransaction = false; // start/endWrite lock flag to allow sketch to keep SPI bus access open

  \_booted   = true;     // Default attributes

  \_cp437    = true;     // Legacy GLCD font bug fix

  \_utf8     = true;     // UTF8 decoding enabled

#if defined (FONT\_FS\_AVAILABLE) && defined (SMOOTH\_FONT)

  fs\_font  = true;     // Smooth font filing system or array (fs\_font = false) flag

#endif

#if defined (ESP32) && defined (CONFIG\_SPIRAM\_SUPPORT)

  if (psramFound()) \_psram\_enable = true; // Enable the use of PSRAM (if available)

  else

#endif

  \_psram\_enable = false;

  addr\_row = 0xFFFF;  // drawPixel command length optimiser

  addr\_col = 0xFFFF;  // drawPixel command length optimiser

  \_xPivot = 0;

  \_yPivot = 0;

// Legacy support for bit GPIO masks

  cspinmask = 0;

  dcpinmask = 0;

  wrpinmask = 0;

  sclkpinmask = 0;

// Flags for which fonts are loaded

#ifdef LOAD\_GLCD

  fontsloaded  = 0x0002; // Bit 1 set

#endif

#ifdef LOAD\_FONT2

  fontsloaded |= 0x0004; // Bit 2 set

#endif

#ifdef LOAD\_FONT4

  fontsloaded |= 0x0010; // Bit 4 set

#endif

#ifdef LOAD\_FONT6

  fontsloaded |= 0x0040; // Bit 6 set

#endif

#ifdef LOAD\_FONT7

  fontsloaded |= 0x0080; // Bit 7 set

#endif

#ifdef LOAD\_FONT8

  fontsloaded |= 0x0100; // Bit 8 set

#endif

#ifdef LOAD\_FONT8N

  fontsloaded |= 0x0200; // Bit 9 set

#endif

#ifdef SMOOTH\_FONT

  fontsloaded |= 0x8000; // Bit 15 set

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           initBus

\*\* Description:             initialise the SPI or parallel bus

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::initBus(void) {

#ifdef TFT\_CS

  pinMode(TFT\_CS, OUTPUT);

  digitalWrite(TFT\_CS, HIGH); // Chip select high (inactive)

#endif

// Configure chip select for touchscreen controller if present

#ifdef TOUCH\_CS

  pinMode(TOUCH\_CS, OUTPUT);

  digitalWrite(TOUCH\_CS, HIGH); // Chip select high (inactive)

#endif

// In parallel mode and with the RP2040 processor, the TFT\_WR line is handled in the  PIO

#if defined (TFT\_WR) && !defined (ARDUINO\_ARCH\_RP2040) && !defined (ARDUINO\_ARCH\_MBED)

  pinMode(TFT\_WR, OUTPUT);

  digitalWrite(TFT\_WR, HIGH); // Set write strobe high (inactive)

#endif

#ifdef TFT\_DC

  pinMode(TFT\_DC, OUTPUT);

  digitalWrite(TFT\_DC, HIGH); // Data/Command high = data mode

#endif

#ifdef TFT\_RST

  if (TFT\_RST >= 0) {

    pinMode(TFT\_RST, OUTPUT);

    digitalWrite(TFT\_RST, HIGH); // Set high, do not share pin with another SPI device

  }

#endif

#if defined (TFT\_PARALLEL\_8\_BIT)

  // Make sure read is high before we set the bus to output

  pinMode(TFT\_RD, OUTPUT);

  digitalWrite(TFT\_RD, HIGH);

  #if  !defined (ARDUINO\_ARCH\_RP2040)  && !defined (ARDUINO\_ARCH\_MBED)// PIO manages pins

    // Set TFT data bus lines to output

    pinMode(TFT\_D0, OUTPUT); digitalWrite(TFT\_D0, HIGH);

    pinMode(TFT\_D1, OUTPUT); digitalWrite(TFT\_D1, HIGH);

    pinMode(TFT\_D2, OUTPUT); digitalWrite(TFT\_D2, HIGH);

    pinMode(TFT\_D3, OUTPUT); digitalWrite(TFT\_D3, HIGH);

    pinMode(TFT\_D4, OUTPUT); digitalWrite(TFT\_D4, HIGH);

    pinMode(TFT\_D5, OUTPUT); digitalWrite(TFT\_D5, HIGH);

    pinMode(TFT\_D6, OUTPUT); digitalWrite(TFT\_D6, HIGH);

    pinMode(TFT\_D7, OUTPUT); digitalWrite(TFT\_D7, HIGH);

  #endif

  PARALLEL\_INIT\_TFT\_DATA\_BUS;

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           begin

\*\* Description:             Included for backwards compatibility

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::begin(uint8\_t tc)

{

 init(tc);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           init (tc is tab colour for ST7735 displays only)

\*\* Description:             Reset, then initialise the TFT display registers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::init(uint8\_t tc)

{

  if (\_booted)

  {

    initBus();

#if !defined (ESP32) && !defined(TFT\_PARALLEL\_8\_BIT) && !defined(ARDUINO\_ARCH\_RP2040) && !defined (ARDUINO\_ARCH\_MBED)

  // Legacy bitmasks for GPIO

  #if defined (TFT\_CS) && (TFT\_CS >= 0)

    cspinmask = (uint32\_t) digitalPinToBitMask(TFT\_CS);

  #endif

  #if defined (TFT\_DC) && (TFT\_DC >= 0)

    dcpinmask = (uint32\_t) digitalPinToBitMask(TFT\_DC);

  #endif

  #if defined (TFT\_WR) && (TFT\_WR >= 0)

    wrpinmask = (uint32\_t) digitalPinToBitMask(TFT\_WR);

  #endif

  #if defined (TFT\_SCLK) && (TFT\_SCLK >= 0)

    sclkpinmask = (uint32\_t) digitalPinToBitMask(TFT\_SCLK);

  #endif

  #if defined (TFT\_SPI\_OVERLAP) && defined (ARDUINO\_ARCH\_ESP8266)

    // Overlap mode SD0=MISO, SD1=MOSI, CLK=SCLK must use D3 as CS

    //    pins(int8\_t sck, int8\_t miso, int8\_t mosi, int8\_t ss);

    //spi.pins(        6,          7,           8,          0);

    spi.pins(6, 7, 8, 0);

  #endif

  spi.begin(); // This will set HMISO to input

#else

  #if !defined(TFT\_PARALLEL\_8\_BIT) && !defined(RP2040\_PIO\_INTERFACE)

    #if defined (TFT\_MOSI) && !defined (TFT\_SPI\_OVERLAP) && !defined(ARDUINO\_ARCH\_RP2040) && !defined (ARDUINO\_ARCH\_MBED)

      spi.begin(TFT\_SCLK, TFT\_MISO, TFT\_MOSI, -1); // This will set MISO to input

    #else

      spi.begin(); // This will set MISO to input

    #endif

  #endif

#endif

    lockTransaction = false;

    inTransaction = false;

    locked = true;

    INIT\_TFT\_DATA\_BUS;

#if defined (TFT\_CS) && !defined(RP2040\_PIO\_INTERFACE)

  // Set to output once again in case MISO is used for CS

  pinMode(TFT\_CS, OUTPUT);

  digitalWrite(TFT\_CS, HIGH); // Chip select high (inactive)

#elif defined (ARDUINO\_ARCH\_ESP8266) && !defined (TFT\_PARALLEL\_8\_BIT) && !defined (RP2040\_PIO\_SPI)

  spi.setHwCs(1); // Use hardware SS toggling

#endif

  // Set to output once again in case MISO is used for DC

#if defined (TFT\_DC) && !defined(RP2040\_PIO\_INTERFACE)

    pinMode(TFT\_DC, OUTPUT);

    digitalWrite(TFT\_DC, HIGH); // Data/Command high = data mode

#endif

    \_booted = false;

    end\_tft\_write();

  } // end of: if just \_booted

  // Toggle RST low to reset

#ifdef TFT\_RST

  #if !defined(RP2040\_PIO\_INTERFACE)

    // Set to output once again in case MISO is used for TFT\_RST

    pinMode(TFT\_RST, OUTPUT);

  #endif

  if (TFT\_RST >= 0) {

    writecommand(0x00); // Put SPI bus in known state for TFT with CS tied low

    digitalWrite(TFT\_RST, HIGH);

    delay(5);

    digitalWrite(TFT\_RST, LOW);

    delay(20);

    digitalWrite(TFT\_RST, HIGH);

  }

  else writecommand(TFT\_SWRST); // Software reset

#else

  writecommand(TFT\_SWRST); // Software reset

#endif

  delay(150); // Wait for reset to complete

  begin\_tft\_write();

  tc = tc; // Suppress warning

  // This loads the driver specific initialisation code  <<<<<<<<<<<<<<<<<<<<< ADD NEW DRIVERS TO THE LIST HERE <<<<<<<<<<<<<<<<<<<<<<<

#if   defined (ILI9341\_DRIVER) || defined(ILI9341\_2\_DRIVER) || defined (ILI9342\_DRIVER)

    #include "TFT\_Drivers/ILI9341\_Init.h"

#elif defined (ST7735\_DRIVER)

    tabcolor = tc;

    #include "TFT\_Drivers/ST7735\_Init.h"

#elif defined (ILI9163\_DRIVER)

    #include "TFT\_Drivers/ILI9163\_Init.h"

#elif defined (S6D02A1\_DRIVER)

    #include "TFT\_Drivers/S6D02A1\_Init.h"

#elif defined (ST7796\_DRIVER)

    #include "TFT\_Drivers/ST7796\_Init.h"

#elif defined (ILI9486\_DRIVER)

    #include "TFT\_Drivers/ILI9486\_Init.h"

#elif defined (ILI9481\_DRIVER)

    #include "TFT\_Drivers/ILI9481\_Init.h"

#elif defined (ILI9488\_DRIVER)

    #include "TFT\_Drivers/ILI9488\_Init.h"

#elif defined (HX8357D\_DRIVER)

    #include "TFT\_Drivers/HX8357D\_Init.h"

#elif defined (ST7789\_DRIVER)

    #include "TFT\_Drivers/ST7789\_Init.h"

#elif defined (R61581\_DRIVER)

    #include "TFT\_Drivers/R61581\_Init.h"

#elif defined (RM68140\_DRIVER)

  #include "TFT\_Drivers/RM68140\_Init.h"

#elif defined (ST7789\_2\_DRIVER)

    #include "TFT\_Drivers/ST7789\_2\_Init.h"

#elif defined (SSD1351\_DRIVER)

    #include "TFT\_Drivers/SSD1351\_Init.h"

#elif defined (SSD1963\_DRIVER)

    #include "TFT\_Drivers/SSD1963\_Init.h"

#elif defined (GC9A01\_DRIVER)

     #include "TFT\_Drivers/GC9A01\_Init.h"

#elif defined (ILI9225\_DRIVER)

     #include "TFT\_Drivers/ILI9225\_Init.h"

#elif defined (RM68120\_DRIVER)

     #include "TFT\_Drivers/RM68120\_Init.h"

#elif defined (HX8357B\_DRIVER)

    #include "TFT\_Drivers/HX8357B\_Init.h"

#elif defined (HX8357C\_DRIVER)

    #include "TFT\_Drivers/HX8357C\_Init.h"

#endif

#ifdef TFT\_INVERSION\_ON

  writecommand(TFT\_INVON);

#endif

#ifdef TFT\_INVERSION\_OFF

  writecommand(TFT\_INVOFF);

#endif

  end\_tft\_write();

  setRotation(rotation);

#if defined (TFT\_BL) && defined (TFT\_BACKLIGHT\_ON)

  pinMode(TFT\_BL, OUTPUT);

  digitalWrite(TFT\_BL, TFT\_BACKLIGHT\_ON);

#else

  #if defined (TFT\_BL) && defined (M5STACK)

    // Turn on the back-light LED

    pinMode(TFT\_BL, OUTPUT);

    digitalWrite(TFT\_BL, HIGH);

  #endif

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setRotation

\*\* Description:             rotate the screen orientation m = 0-3 or 4-7 for BMP drawing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setRotation(uint8\_t m)

{

  begin\_tft\_write();

    // This loads the driver specific rotation code  <<<<<<<<<<<<<<<<<<<<< ADD NEW DRIVERS TO THE LIST HERE <<<<<<<<<<<<<<<<<<<<<<<

#if   defined (ILI9341\_DRIVER) || defined(ILI9341\_2\_DRIVER) || defined (ILI9342\_DRIVER)

    #include "TFT\_Drivers/ILI9341\_Rotation.h"

#elif defined (ST7735\_DRIVER)

    #include "TFT\_Drivers/ST7735\_Rotation.h"

#elif defined (ILI9163\_DRIVER)

    #include "TFT\_Drivers/ILI9163\_Rotation.h"

#elif defined (S6D02A1\_DRIVER)

    #include "TFT\_Drivers/S6D02A1\_Rotation.h"

#elif defined (ST7796\_DRIVER)

    #include "TFT\_Drivers/ST7796\_Rotation.h"

#elif defined (ILI9486\_DRIVER)

    #include "TFT\_Drivers/ILI9486\_Rotation.h"

#elif defined (ILI9481\_DRIVER)

    #include "TFT\_Drivers/ILI9481\_Rotation.h"

#elif defined (ILI9488\_DRIVER)

    #include "TFT\_Drivers/ILI9488\_Rotation.h"

#elif defined (HX8357D\_DRIVER)

    #include "TFT\_Drivers/HX8357D\_Rotation.h"

#elif defined (ST7789\_DRIVER)

    #include "TFT\_Drivers/ST7789\_Rotation.h"

#elif defined (R61581\_DRIVER)

    #include "TFT\_Drivers/R61581\_Rotation.h"

#elif defined (RM68140\_DRIVER)

  #include "TFT\_Drivers/RM68140\_Rotation.h"

#elif defined (ST7789\_2\_DRIVER)

    #include "TFT\_Drivers/ST7789\_2\_Rotation.h"

#elif defined (SSD1351\_DRIVER)

    #include "TFT\_Drivers/SSD1351\_Rotation.h"

#elif defined (SSD1963\_DRIVER)

    #include "TFT\_Drivers/SSD1963\_Rotation.h"

#elif defined (GC9A01\_DRIVER)

     #include "TFT\_Drivers/GC9A01\_Rotation.h"

#elif defined (ILI9225\_DRIVER)

     #include "TFT\_Drivers/ILI9225\_Rotation.h"

#elif defined (RM68120\_DRIVER)

     #include "TFT\_Drivers/RM68120\_Rotation.h"

#elif defined (HX8357B\_DRIVER)

    #include "TFT\_Drivers/HX8357B\_Rotation.h"

#elif defined (HX8357C\_DRIVER)

    #include "TFT\_Drivers/HX8357C\_Rotation.h"

#endif

  delayMicroseconds(10);

  end\_tft\_write();

  addr\_row = 0xFFFF;

  addr\_col = 0xFFFF;

  // Reset the viewport to the whole screen

  resetViewport();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           commandList, used for FLASH based lists only (e.g. ST7735)

\*\* Description:             Get initialisation commands from FLASH and send to TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::commandList (const uint8\_t \*addr)

{

  uint8\_t  numCommands;

  uint8\_t  numArgs;

  uint8\_t  ms;

  numCommands = pgm\_read\_byte(addr++);   // Number of commands to follow

  while (numCommands--)                  // For each command...

  {

    writecommand(pgm\_read\_byte(addr++)); // Read, issue command

    numArgs = pgm\_read\_byte(addr++);     // Number of args to follow

    ms = numArgs & TFT\_INIT\_DELAY;       // If hibit set, delay follows args

    numArgs &= ~TFT\_INIT\_DELAY;          // Mask out delay bit

    while (numArgs--)                    // For each argument...

    {

      writedata(pgm\_read\_byte(addr++));  // Read, issue argument

    }

    if (ms)

    {

      ms = pgm\_read\_byte(addr++);        // Read post-command delay time (ms)

      delay( (ms==255 ? 500 : ms) );

    }

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           spiwrite

\*\* Description:             Write 8 bits to SPI port (legacy support only)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::spiwrite(uint8\_t c)

{

  begin\_tft\_write();

  tft\_Write\_8(c);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           writecommand

\*\* Description:             Send an 8 bit command to the TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef RM68120\_DRIVER

void TFT\_eSPI::writecommand(uint8\_t c)

{

  begin\_tft\_write();

  DC\_C;

  tft\_Write\_8(c);

  DC\_D;

  end\_tft\_write();

}

#else

void TFT\_eSPI::writecommand(uint16\_t c)

{

  begin\_tft\_write();

  DC\_C;

  tft\_Write\_16(c);

  DC\_D;

  end\_tft\_write();

}

void TFT\_eSPI::writeRegister(uint16\_t c, uint8\_t d)

{

  begin\_tft\_write();

  DC\_C;

  tft\_Write\_16(c);

  DC\_D;

  tft\_Write\_8(d);

  end\_tft\_write();

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           writedata

\*\* Description:             Send a 8 bit data value to the TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::writedata(uint8\_t d)

{

  begin\_tft\_write();

  DC\_D;        // Play safe, but should already be in data mode

  tft\_Write\_8(d);

  CS\_L;        // Allow more hold time for low VDI rail

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           readcommand8

\*\* Description:             Read a 8 bit data value from an indexed command register

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t TFT\_eSPI::readcommand8(uint8\_t cmd\_function, uint8\_t index)

{

  uint8\_t reg = 0;

#if defined(TFT\_PARALLEL\_8\_BIT) || defined(RP2040\_PIO\_INTERFACE)

  writecommand(cmd\_function); // Sets DC and CS high

  busDir(GPIO\_DIR\_MASK, INPUT);

  CS\_L;

  // Read nth parameter (assumes caller discards 1st parameter or points index to 2nd)

  while(index--) reg = readByte();

  busDir(GPIO\_DIR\_MASK, OUTPUT);

  CS\_H;

#else // SPI interface

  // Tested with ILI9341 set to Interface II i.e. IM [3:0] = "1101"

  begin\_tft\_read();

  index = 0x10 + (index & 0x0F);

  DC\_C; tft\_Write\_8(0xD9);

  DC\_D; tft\_Write\_8(index);

  CS\_H; // Some displays seem to need CS to be pulsed here, or is just a delay needed?

  CS\_L;

  DC\_C; tft\_Write\_8(cmd\_function);

  DC\_D;

  reg = tft\_Read\_8();

  end\_tft\_read();

#endif

  return reg;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           readcommand16

\*\* Description:             Read a 16 bit data value from an indexed command register

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t TFT\_eSPI::readcommand16(uint8\_t cmd\_function, uint8\_t index)

{

  uint32\_t reg;

  reg  = (readcommand8(cmd\_function, index + 0) <<  8);

  reg |= (readcommand8(cmd\_function, index + 1) <<  0);

  return reg;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           readcommand32

\*\* Description:             Read a 32 bit data value from an indexed command register

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t TFT\_eSPI::readcommand32(uint8\_t cmd\_function, uint8\_t index)

{

  uint32\_t reg;

  reg  = ((uint32\_t)readcommand8(cmd\_function, index + 0) << 24);

  reg |= ((uint32\_t)readcommand8(cmd\_function, index + 1) << 16);

  reg |= ((uint32\_t)readcommand8(cmd\_function, index + 2) <<  8);

  reg |= ((uint32\_t)readcommand8(cmd\_function, index + 3) <<  0);

  return reg;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           read pixel (for SPI Interface II i.e. IM [3:0] = "1101")

\*\* Description:             Read 565 pixel colours from a pixel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t TFT\_eSPI::readPixel(int32\_t x0, int32\_t y0)

{

  if (\_vpOoB) return 0;

  x0+= \_xDatum;

  y0+= \_yDatum;

  // Range checking

  if ((x0 < \_vpX) || (y0 < \_vpY) ||(x0 >= \_vpW) || (y0 >= \_vpH)) return 0;

#if defined(TFT\_PARALLEL\_8\_BIT) || defined(RP2040\_PIO\_INTERFACE)

  if (!inTransaction) { CS\_L; } // CS\_L can be multi-statement

  readAddrWindow(x0, y0, 1, 1);

  // Set masked pins D0- D7 to input

  busDir(GPIO\_DIR\_MASK, INPUT);

  #if  !defined (SSD1963\_DRIVER)

  // Dummy read to throw away don't care value

  readByte();

  #endif

  // Fetch the 16 bit BRG pixel

  //uint16\_t rgb = (readByte() << 8) | readByte();

  #if defined (ILI9341\_DRIVER)  || defined(ILI9341\_2\_DRIVER) || defined (ILI9488\_DRIVER) || defined (SSD1963\_DRIVER)// Read 3 bytes

    // Read window pixel 24 bit RGB values and fill in LS bits

    uint16\_t rgb = ((readByte() & 0xF8) << 8) | ((readByte() & 0xFC) << 3) | (readByte() >> 3);

    if (!inTransaction) { CS\_H; } // CS\_H can be multi-statement

    // Set masked pins D0- D7 to output

    busDir(GPIO\_DIR\_MASK, OUTPUT);

    return rgb;

  #else // ILI9481 or ILI9486 16 bit read

    // Fetch the 16 bit BRG pixel

    uint16\_t bgr = (readByte() << 8) | readByte();

    if (!inTransaction) { CS\_H; } // CS\_H can be multi-statement

    // Set masked pins D0- D7 to output

    busDir(GPIO\_DIR\_MASK, OUTPUT);

    #ifdef ILI9486\_DRIVER

      return  bgr;

    #else

      // Swap Red and Blue (could check MADCTL setting to see if this is needed)

      return  (bgr>>11) | (bgr<<11) | (bgr & 0x7E0);

    #endif

  #endif

#else // Not TFT\_PARALLEL\_8\_BIT

  // This function can get called during anti-aliased font rendering

  // so a transaction may be in progress

  bool wasInTransaction = inTransaction;

  if (inTransaction) { inTransaction= false; end\_tft\_write();}

  uint16\_t color = 0;

  begin\_tft\_read(); // Sets CS low

  readAddrWindow(x0, y0, 1, 1);

  #ifdef TFT\_SDA\_READ

    begin\_SDA\_Read();

  #endif

  // Dummy read to throw away don't care value

  tft\_Read\_8();

  //#if !defined (ILI9488\_DRIVER)

    #if defined (ST7796\_DRIVER)

      // Read the 2 bytes

      color = ((tft\_Read\_8()) << 8) | (tft\_Read\_8());

    #else

      // Read the 3 RGB bytes, colour is actually only in the top 6 bits of each byte

      // as the TFT stores colours as 18 bits

      uint8\_t r = tft\_Read\_8();

      uint8\_t g = tft\_Read\_8();

      uint8\_t b = tft\_Read\_8();

      color = color565(r, g, b);

    #endif

/\*

  #else

    // The 6 colour bits are in MS 6 bits of each byte, but the ILI9488 needs an extra clock pulse

    // so bits appear shifted right 1 bit, so mask the middle 6 bits then shift 1 place left

    uint8\_t r = (tft\_Read\_8()&0x7E)<<1;

    uint8\_t g = (tft\_Read\_8()&0x7E)<<1;

    uint8\_t b = (tft\_Read\_8()&0x7E)<<1;

    color = color565(r, g, b);

  #endif

\*/

  CS\_H;

  #ifdef TFT\_SDA\_READ

    end\_SDA\_Read();

  #endif

  end\_tft\_read();

  // Reinstate the transaction if one was in progress

  if(wasInTransaction) { begin\_tft\_write(); inTransaction = true; }

  return color;

#endif

}

void TFT\_eSPI::setCallback(getColorCallback getCol)

{

  getColor = getCol;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           read rectangle (for SPI Interface II i.e. IM [3:0] = "1101")

\*\* Description:             Read 565 pixel colours from a defined area

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::readRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data)

{

  PI\_CLIP ;

#if defined(TFT\_PARALLEL\_8\_BIT) || defined(RP2040\_PIO\_INTERFACE)

  CS\_L;

  readAddrWindow(x, y, dw, dh);

  data += dx + dy \* w;

  // Set masked pins D0- D7 to input

  busDir(GPIO\_DIR\_MASK, INPUT);

  #if defined (ILI9341\_DRIVER)  || defined(ILI9341\_2\_DRIVER) || defined (ILI9488\_DRIVER) // Read 3 bytes

    // Dummy read to throw away don't care value

    readByte();

    // Fetch the 24 bit RGB value

    while (dh--) {

      int32\_t lw = dw;

      uint16\_t\* line = data;

      while (lw--) {

        // Assemble the RGB 16 bit colour

        uint16\_t rgb = ((readByte() & 0xF8) << 8) | ((readByte() & 0xFC) << 3) | (readByte() >> 3);

        // Swapped byte order for compatibility with pushRect()

        \*line++ = (rgb<<8) | (rgb>>8);

      }

      data += w;

    }

  #elif  defined (SSD1963\_DRIVER)

    // Fetch the 18 bit BRG pixels

    while (dh--) {

      int32\_t lw = dw;

      uint16\_t\* line = data;

      while (lw--) {

        uint16\_t bgr = ((readByte() & 0xF8) >> 3);; // CS\_L adds a small delay

        bgr |= ((readByte() & 0xFC) << 3);

        bgr |= (readByte() << 8);

        // Swap Red and Blue (could check MADCTL setting to see if this is needed)

        uint16\_t rgb = (bgr>>11) | (bgr<<11) | (bgr & 0x7E0);

        // Swapped byte order for compatibility with pushRect()

        \*line++ = (rgb<<8) | (rgb>>8);

      }

      data += w;

    }

  #else // ILI9481 reads as 16 bits

    // Dummy read to throw away don't care value

    readByte();

    // Fetch the 16 bit BRG pixels

    while (dh--) {

      int32\_t lw = dw;

      uint16\_t\* line = data;

      while (lw--) {

      #ifdef ILI9486\_DRIVER

        // Read the RGB 16 bit colour

        \*line++ = readByte() | (readByte() << 8);

      #else

        // Read the BRG 16 bit colour

        uint16\_t bgr = (readByte() << 8) | readByte();

        // Swap Red and Blue (could check MADCTL setting to see if this is needed)

        uint16\_t rgb = (bgr>>11) | (bgr<<11) | (bgr & 0x7E0);

        // Swapped byte order for compatibility with pushRect()

        \*line++ = (rgb<<8) | (rgb>>8);

      #endif

      }

      data += w;

    }

  #endif

  CS\_H;

  // Set masked pins D0- D7 to output

  busDir(GPIO\_DIR\_MASK, OUTPUT);

#else // SPI interface

  // This function can get called after a begin\_tft\_write

  // so a transaction may be in progress

  bool wasInTransaction = inTransaction;

  if (inTransaction) { inTransaction= false; end\_tft\_write();}

  uint16\_t color = 0;

  begin\_tft\_read();

  readAddrWindow(x, y, dw, dh);

  data += dx + dy \* w;

  #ifdef TFT\_SDA\_READ

    begin\_SDA\_Read();

  #endif

  // Dummy read to throw away don't care value

  tft\_Read\_8();

  // Read window pixel 24 bit RGB values

  while (dh--) {

    int32\_t lw = dw;

    uint16\_t\* line = data;

    while (lw--) {

  #if !defined (ILI9488\_DRIVER)

    #if defined (ST7796\_DRIVER)

      // Read the 2 bytes

      color = ((tft\_Read\_8()) << 8) | (tft\_Read\_8());

    #else

      // Read the 3 RGB bytes, colour is actually only in the top 6 bits of each byte

      // as the TFT stores colours as 18 bits

      uint8\_t r = tft\_Read\_8();

      uint8\_t g = tft\_Read\_8();

      uint8\_t b = tft\_Read\_8();

      color = color565(r, g, b);

    #endif

  #else

      // The 6 colour bits are in MS 6 bits of each byte but we do not include the extra clock pulse

      // so we use a trick and mask the middle 6 bits of the byte, then only shift 1 place left

      uint8\_t r = (tft\_Read\_8()&0x7E)<<1;

      uint8\_t g = (tft\_Read\_8()&0x7E)<<1;

      uint8\_t b = (tft\_Read\_8()&0x7E)<<1;

      color = color565(r, g, b);

  #endif

      // Swapped colour byte order for compatibility with pushRect()

      \*line++ = color << 8 | color >> 8;

    }

    data += w;

  }

  //CS\_H;

  #ifdef TFT\_SDA\_READ

    end\_SDA\_Read();

  #endif

  end\_tft\_read();

  // Reinstate the transaction if one was in progress

  if(wasInTransaction) { begin\_tft\_write(); inTransaction = true; }

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           push rectangle

\*\* Description:             push 565 pixel colours into a defined area

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data)

{

  bool swap = \_swapBytes; \_swapBytes = false;

  pushImage(x, y, w, h, data);

  \_swapBytes = swap;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage

\*\* Description:             plot 16 bit colour sprite or image onto TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data)

{

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  setWindow(x, y, x + dw - 1, y + dh - 1);

  data += dx + dy \* w;

  // Check if whole image can be pushed

  if (dw == w) pushPixels(data, dw \* dh);

  else {

    // Push line segments to crop image

    while (dh--)

    {

      pushPixels(data, dw);

      data += w;

    }

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage

\*\* Description:             plot 16 bit sprite or image with 1 colour being transparent

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint16\_t \*data, uint16\_t transp)

{

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  data += dx + dy \* w;

  uint16\_t  lineBuf[dw]; // Use buffer to minimise setWindow call count

  // The little endian transp color must be byte swapped if the image is big endian

  if (!\_swapBytes) transp = transp >> 8 | transp << 8;

  while (dh--)

  {

    int32\_t len = dw;

    uint16\_t\* ptr = data;

    int32\_t px = x, sx = x;

    bool move = true;

    uint16\_t np = 0;

    while (len--)

    {

      if (transp != \*ptr)

      {

        if (move) { move = false; sx = px; }

        lineBuf[np] = \*ptr;

        np++;

      }

      else

      {

        move = true;

        if (np)

        {

          setWindow(sx, y, sx + np - 1, y);

          pushPixels((uint16\_t\*)lineBuf, np);

          np = 0;

        }

      }

      px++;

      ptr++;

    }

    if (np) { setWindow(sx, y, sx + np - 1, y); pushPixels((uint16\_t\*)lineBuf, np); }

    y++;

    data += w;

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage - for FLASH (PROGMEM) stored images

\*\* Description:             plot 16 bit image

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, const uint16\_t \*data)

{

  // Requires 32 bit aligned access, so use PROGMEM 16 bit word functions

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  data += dx + dy \* w;

  uint16\_t  buffer[dw];

  setWindow(x, y, x + dw - 1, y + dh - 1);

  // Fill and send line buffers to TFT

  for (int32\_t i = 0; i < dh; i++) {

    for (int32\_t j = 0; j < dw; j++) {

      buffer[j] = pgm\_read\_word(&data[i \* w + j]);

    }

    pushPixels(buffer, dw);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage - for FLASH (PROGMEM) stored images

\*\* Description:             plot 16 bit image with 1 colour being transparent

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, const uint16\_t \*data, uint16\_t transp)

{

  // Requires 32 bit aligned access, so use PROGMEM 16 bit word functions

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  data += dx + dy \* w;

  uint16\_t  lineBuf[dw];

  // The little endian transp color must be byte swapped if the image is big endian

  if (!\_swapBytes) transp = transp >> 8 | transp << 8;

  while (dh--) {

    int32\_t len = dw;

    uint16\_t\* ptr = (uint16\_t\*)data;

    int32\_t px = x, sx = x;

    bool move = true;

    uint16\_t np = 0;

    while (len--) {

      uint16\_t color = pgm\_read\_word(ptr);

      if (transp != color) {

        if (move) { move = false; sx = px; }

        lineBuf[np] = color;

        np++;

      }

      else {

        move = true;

        if (np) {

          setWindow(sx, y, sx + np - 1, y);

          pushPixels(lineBuf, np);

          np = 0;

        }

      }

      px++;

      ptr++;

    }

    if (np) { setWindow(sx, y, sx + np - 1, y); pushPixels(lineBuf, np); }

    y++;

    data += w;

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage

\*\* Description:             plot 8 bit or 4 bit or 1 bit image or sprite using a line buffer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, const uint8\_t \*data, bool bpp8,  uint16\_t \*cmap)

{

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  bool swap = \_swapBytes;

  setWindow(x, y, x + dw - 1, y + dh - 1); // Sets CS low and sent RAMWR

  // Line buffer makes plotting faster

  uint16\_t  lineBuf[dw];

  if (bpp8)

  {

    \_swapBytes = false;

    uint8\_t  blue[] = {0, 11, 21, 31}; // blue 2 to 5 bit colour lookup table

    \_lastColor = -1; // Set to illegal value

    // Used to store last shifted colour

    uint8\_t msbColor = 0;

    uint8\_t lsbColor = 0;

    data += dx + dy \* w;

    while (dh--) {

      uint32\_t len = dw;

      uint8\_t\* ptr = (uint8\_t\*)data;

      uint8\_t\* linePtr = (uint8\_t\*)lineBuf;

      while(len--) {

        uint32\_t color = pgm\_read\_byte(ptr++);

        // Shifts are slow so check if colour has changed first

        if (color != \_lastColor) {

          //          =====Green=====     ===============Red==============

          msbColor = (color & 0x1C)>>2 | (color & 0xC0)>>3 | (color & 0xE0);

          //          =====Green=====    =======Blue======

          lsbColor = (color & 0x1C)<<3 | blue[color & 0x03];

          \_lastColor = color;

        }

       \*linePtr++ = msbColor;

       \*linePtr++ = lsbColor;

      }

      pushPixels(lineBuf, dw);

      data += w;

    }

    \_swapBytes = swap; // Restore old value

  }

  else if (cmap != nullptr) // Must be 4bpp

  {

    \_swapBytes = true;

    w = (w+1) & 0xFFFE;   // if this is a sprite, w will already be even; this does no harm.

    bool splitFirst = (dx & 0x01) != 0; // split first means we have to push a single px from the left of the sprite / image

    if (splitFirst) {

      data += ((dx - 1 + dy \* w) >> 1);

    }

    else {

      data += ((dx + dy \* w) >> 1);

    }

    while (dh--) {

      uint32\_t len = dw;

      uint8\_t \* ptr = (uint8\_t\*)data;

      uint16\_t \*linePtr = lineBuf;

      uint8\_t colors; // two colors in one byte

      uint16\_t index;

      if (splitFirst) {

        colors = pgm\_read\_byte(ptr);

        index = (colors & 0x0F);

        \*linePtr++ = cmap[index];

        len--;

        ptr++;

      }

      while (len--)

      {

        colors = pgm\_read\_byte(ptr);

        index = ((colors & 0xF0) >> 4) & 0x0F;

        \*linePtr++ = cmap[index];

        if (len--)

        {

          index = colors & 0x0F;

          \*linePtr++ = cmap[index];

        } else {

          break;  // nothing to do here

        }

        ptr++;

      }

      pushPixels(lineBuf, dw);

      data += (w >> 1);

    }

    \_swapBytes = swap; // Restore old value

  }

  else // Must be 1bpp

  {

    \_swapBytes = false;

    uint8\_t \* ptr = (uint8\_t\*)data;

    uint32\_t ww =  (w+7)>>3; // Width of source image line in bytes

    for (int32\_t yp = dy;  yp < dy + dh; yp++)

    {

      uint8\_t\* linePtr = (uint8\_t\*)lineBuf;

      for (int32\_t xp = dx; xp < dx + dw; xp++)

      {

        uint16\_t col = (pgm\_read\_byte(ptr + (xp>>3)) & (0x80 >> (xp & 0x7)) );

        if (col) {\*linePtr++ = bitmap\_fg>>8; \*linePtr++ = (uint8\_t) bitmap\_fg;}

        else     {\*linePtr++ = bitmap\_bg>>8; \*linePtr++ = (uint8\_t) bitmap\_bg;}

      }

      ptr += ww;

      pushPixels(lineBuf, dw);

    }

  }

  \_swapBytes = swap; // Restore old value

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage

\*\* Description:             plot 8 bit or 4 bit or 1 bit image or sprite using a line buffer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint8\_t \*data, bool bpp8,  uint16\_t \*cmap)

{

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  bool swap = \_swapBytes;

  setWindow(x, y, x + dw - 1, y + dh - 1); // Sets CS low and sent RAMWR

  // Line buffer makes plotting faster

  uint16\_t  lineBuf[dw];

  if (bpp8)

  {

    \_swapBytes = false;

    uint8\_t  blue[] = {0, 11, 21, 31}; // blue 2 to 5 bit colour lookup table

    \_lastColor = -1; // Set to illegal value

    // Used to store last shifted colour

    uint8\_t msbColor = 0;

    uint8\_t lsbColor = 0;

    data += dx + dy \* w;

    while (dh--) {

      uint32\_t len = dw;

      uint8\_t\* ptr = data;

      uint8\_t\* linePtr = (uint8\_t\*)lineBuf;

      while(len--) {

        uint32\_t color = \*ptr++;

        // Shifts are slow so check if colour has changed first

        if (color != \_lastColor) {

          //          =====Green=====     ===============Red==============

          msbColor = (color & 0x1C)>>2 | (color & 0xC0)>>3 | (color & 0xE0);

          //          =====Green=====    =======Blue======

          lsbColor = (color & 0x1C)<<3 | blue[color & 0x03];

          \_lastColor = color;

        }

       \*linePtr++ = msbColor;

       \*linePtr++ = lsbColor;

      }

      pushPixels(lineBuf, dw);

      data += w;

    }

    \_swapBytes = swap; // Restore old value

  }

  else if (cmap != nullptr) // Must be 4bpp

  {

    \_swapBytes = true;

    w = (w+1) & 0xFFFE;   // if this is a sprite, w will already be even; this does no harm.

    bool splitFirst = (dx & 0x01) != 0; // split first means we have to push a single px from the left of the sprite / image

    if (splitFirst) {

      data += ((dx - 1 + dy \* w) >> 1);

    }

    else {

      data += ((dx + dy \* w) >> 1);

    }

    while (dh--) {

      uint32\_t len = dw;

      uint8\_t \* ptr = data;

      uint16\_t \*linePtr = lineBuf;

      uint8\_t colors; // two colors in one byte

      uint16\_t index;

      if (splitFirst) {

        colors = \*ptr;

        index = (colors & 0x0F);

        \*linePtr++ = cmap[index];

        len--;

        ptr++;

      }

      while (len--)

      {

        colors = \*ptr;

        index = ((colors & 0xF0) >> 4) & 0x0F;

        \*linePtr++ = cmap[index];

        if (len--)

        {

          index = colors & 0x0F;

          \*linePtr++ = cmap[index];

        } else {

          break;  // nothing to do here

        }

        ptr++;

      }

      pushPixels(lineBuf, dw);

      data += (w >> 1);

    }

    \_swapBytes = swap; // Restore old value

  }

  else // Must be 1bpp

  {

    \_swapBytes = false;

    uint32\_t ww =  (w+7)>>3; // Width of source image line in bytes

    for (int32\_t yp = dy;  yp < dy + dh; yp++)

    {

      uint8\_t\* linePtr = (uint8\_t\*)lineBuf;

      for (int32\_t xp = dx; xp < dx + dw; xp++)

      {

        uint16\_t col = (data[(xp>>3)] & (0x80 >> (xp & 0x7)) );

        if (col) {\*linePtr++ = bitmap\_fg>>8; \*linePtr++ = (uint8\_t) bitmap\_fg;}

        else     {\*linePtr++ = bitmap\_bg>>8; \*linePtr++ = (uint8\_t) bitmap\_bg;}

      }

      data += ww;

      pushPixels(lineBuf, dw);

    }

  }

  \_swapBytes = swap; // Restore old value

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushImage

\*\* Description:             plot 8 or 4 or 1 bit image or sprite with a transparent colour

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushImage(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint8\_t \*data, uint8\_t transp, bool bpp8, uint16\_t \*cmap)

{

  PI\_CLIP;

  begin\_tft\_write();

  inTransaction = true;

  bool swap = \_swapBytes;

  // Line buffer makes plotting faster

  uint16\_t  lineBuf[dw];

  if (bpp8) { // 8 bits per pixel

    \_swapBytes = false;

    data += dx + dy \* w;

    uint8\_t  blue[] = {0, 11, 21, 31}; // blue 2 to 5 bit colour lookup table

    \_lastColor = -1; // Set to illegal value

    // Used to store last shifted colour

    uint8\_t msbColor = 0;

    uint8\_t lsbColor = 0;

    while (dh--) {

      int32\_t len = dw;

      uint8\_t\* ptr = data;

      uint8\_t\* linePtr = (uint8\_t\*)lineBuf;

      int32\_t px = x, sx = x;

      bool move = true;

      uint16\_t np = 0;

      while (len--) {

        if (transp != \*ptr) {

          if (move) { move = false; sx = px; }

          uint8\_t color = \*ptr;

          // Shifts are slow so check if colour has changed first

          if (color != \_lastColor) {

            //          =====Green=====     ===============Red==============

            msbColor = (color & 0x1C)>>2 | (color & 0xC0)>>3 | (color & 0xE0);

            //          =====Green=====    =======Blue======

            lsbColor = (color & 0x1C)<<3 | blue[color & 0x03];

            \_lastColor = color;

          }

          \*linePtr++ = msbColor;

          \*linePtr++ = lsbColor;

          np++;

        }

        else {

          move = true;

          if (np) {

            setWindow(sx, y, sx + np - 1, y);

            pushPixels(lineBuf, np);

            linePtr = (uint8\_t\*)lineBuf;

            np = 0;

          }

        }

        px++;

        ptr++;

      }

      if (np) { setWindow(sx, y, sx + np - 1, y); pushPixels(lineBuf, np); }

      y++;

      data += w;

    }

  }

  else if (cmap != nullptr) // 4bpp with color map

  {

    \_swapBytes = true;

    w = (w+1) & 0xFFFE; // here we try to recreate iwidth from dwidth.

    bool splitFirst = ((dx & 0x01) != 0);

    if (splitFirst) {

      data += ((dx - 1 + dy \* w) >> 1);

    }

    else {

      data += ((dx + dy \* w) >> 1);

    }

    while (dh--) {

      uint32\_t len = dw;

      uint8\_t \* ptr = data;

      int32\_t px = x, sx = x;

      bool move = true;

      uint16\_t np = 0;

      uint8\_t index;  // index into cmap.

      if (splitFirst) {

        index = (\*ptr & 0x0F);  // odd = bits 3 .. 0

        if (index != transp) {

          move = false; sx = px;

          lineBuf[np] = cmap[index];

          np++;

        }

        px++; ptr++;

        len--;

      }

      while (len--)

      {

        uint8\_t color = \*ptr;

        // find the actual color you care about.  There will be two pixels here!

        // but we may only want one at the end of the row

        uint16\_t index = ((color & 0xF0) >> 4) & 0x0F;  // high bits are the even numbers

        if (index != transp) {

          if (move) {

            move = false; sx = px;

          }

          lineBuf[np] = cmap[index];

          np++; // added a pixel

        }

        else {

          move = true;

          if (np) {

            setWindow(sx, y, sx + np - 1, y);

            pushPixels(lineBuf, np);

            np = 0;

          }

        }

        px++;

        if (len--)

        {

          index = color & 0x0F; // the odd number is 3 .. 0

          if (index != transp) {

            if (move) {

              move = false; sx = px;

             }

            lineBuf[np] = cmap[index];

            np++;

          }

          else {

            move = true;

            if (np) {

              setWindow(sx, y, sx + np - 1, y);

              pushPixels(lineBuf, np);

              np = 0;

            }

          }

          px++;

        }

        else {

          break;  // we are done with this row.

        }

        ptr++;  // we only increment ptr once in the loop (deliberate)

      }

      if (np) {

        setWindow(sx, y, sx + np - 1, y);

        pushPixels(lineBuf, np);

        np = 0;

      }

      data += (w>>1);

      y++;

    }

  }

  else { // 1 bit per pixel

    \_swapBytes = false;

    uint32\_t ww =  (w+7)>>3; // Width of source image line in bytes

    uint16\_t np = 0;

    for (int32\_t yp = dy;  yp < dy + dh; yp++)

    {

      int32\_t px = x, sx = x;

      bool move = true;

      for (int32\_t xp = dx; xp < dx + dw; xp++)

      {

        if (data[(xp>>3)] & (0x80 >> (xp & 0x7))) {

          if (move) {

            move = false;

            sx = px;

          }

          np++;

        }

        else {

          move = true;

          if (np) {

            setWindow(sx, y, sx + np - 1, y);

            pushBlock(bitmap\_fg, np);

            np = 0;

          }

        }

        px++;

      }

      if (np) { setWindow(sx, y, sx + np - 1, y); pushBlock(bitmap\_fg, np); np = 0; }

      y++;

      data += ww;

    }

  }

  \_swapBytes = swap; // Restore old value

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setSwapBytes

\*\* Description:             Used by 16 bit pushImage() to swap byte order in colours

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setSwapBytes(bool swap)

{

  \_swapBytes = swap;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getSwapBytes

\*\* Description:             Return the swap byte order for colours

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool TFT\_eSPI::getSwapBytes(void)

{

  return \_swapBytes;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           read rectangle (for SPI Interface II i.e. IM [3:0] = "1101")

\*\* Description:             Read RGB pixel colours from a defined area

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// If w and h are 1, then 1 pixel is read, \*data array size must be 3 bytes per pixel

void  TFT\_eSPI::readRectRGB(int32\_t x0, int32\_t y0, int32\_t w, int32\_t h, uint8\_t \*data)

{

#if defined(TFT\_PARALLEL\_8\_BIT) || defined(RP2040\_PIO\_INTERFACE)

  uint32\_t len = w \* h;

  uint8\_t\* buf565 = data + len;

  readRect(x0, y0, w, h, (uint16\_t\*)buf565);

  while (len--) {

    uint16\_t pixel565 = (\*buf565++)<<8;

    pixel565 |= \*buf565++;

    uint8\_t red   = (pixel565 & 0xF800) >> 8; red   |= red   >> 5;

    uint8\_t green = (pixel565 & 0x07E0) >> 3; green |= green >> 6;

    uint8\_t blue  = (pixel565 & 0x001F) << 3; blue  |= blue  >> 5;

    \*data++ = red;

    \*data++ = green;

    \*data++ = blue;

  }

#else  // Not TFT\_PARALLEL\_8\_BIT

  begin\_tft\_read();

  readAddrWindow(x0, y0, w, h); // Sets CS low

  #ifdef TFT\_SDA\_READ

    begin\_SDA\_Read();

  #endif

  // Dummy read to throw away don't care value

  tft\_Read\_8();

  // Read window pixel 24 bit RGB values, buffer must be set in sketch to 3 \* w \* h

  uint32\_t len = w \* h;

  while (len--) {

  #if !defined (ILI9488\_DRIVER)

    // Read the 3 RGB bytes, colour is actually only in the top 6 bits of each byte

    // as the TFT stores colours as 18 bits

    \*data++ = tft\_Read\_8();

    \*data++ = tft\_Read\_8();

    \*data++ = tft\_Read\_8();

  #else

    // The 6 colour bits are in MS 6 bits of each byte, but the ILI9488 needs an extra clock pulse

    // so bits appear shifted right 1 bit, so mask the middle 6 bits then shift 1 place left

    \*data++ = (tft\_Read\_8()&0x7E)<<1;

    \*data++ = (tft\_Read\_8()&0x7E)<<1;

    \*data++ = (tft\_Read\_8()&0x7E)<<1;

  #endif

  }

  CS\_H;

  #ifdef TFT\_SDA\_READ

    end\_SDA\_Read();

  #endif

  end\_tft\_read();

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawCircle

\*\* Description:             Draw a circle outline

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Optimised midpoint circle algorithm

void TFT\_eSPI::drawCircle(int32\_t x0, int32\_t y0, int32\_t r, uint32\_t color)

{

  if ( r <= 0 ) return;

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

    int32\_t f     = 1 - r;

    int32\_t ddF\_y = -2 \* r;

    int32\_t ddF\_x = 1;

    int32\_t xs    = -1;

    int32\_t xe    = 0;

    int32\_t len   = 0;

    bool first = true;

    do {

      while (f < 0) {

        ++xe;

        f += (ddF\_x += 2);

      }

      f += (ddF\_y += 2);

      if (xe-xs>1) {

        if (first) {

          len = 2\*(xe - xs)-1;

          drawFastHLine(x0 - xe, y0 + r, len, color);

          drawFastHLine(x0 - xe, y0 - r, len, color);

          drawFastVLine(x0 + r, y0 - xe, len, color);

          drawFastVLine(x0 - r, y0 - xe, len, color);

          first = false;

        }

        else {

          len = xe - xs++;

          drawFastHLine(x0 - xe, y0 + r, len, color);

          drawFastHLine(x0 - xe, y0 - r, len, color);

          drawFastHLine(x0 + xs, y0 - r, len, color);

          drawFastHLine(x0 + xs, y0 + r, len, color);

          drawFastVLine(x0 + r, y0 + xs, len, color);

          drawFastVLine(x0 + r, y0 - xe, len, color);

          drawFastVLine(x0 - r, y0 - xe, len, color);

          drawFastVLine(x0 - r, y0 + xs, len, color);

        }

      }

      else {

        ++xs;

        drawPixel(x0 - xe, y0 + r, color);

        drawPixel(x0 - xe, y0 - r, color);

        drawPixel(x0 + xs, y0 - r, color);

        drawPixel(x0 + xs, y0 + r, color);

        drawPixel(x0 + r, y0 + xs, color);

        drawPixel(x0 + r, y0 - xe, color);

        drawPixel(x0 - r, y0 - xe, color);

        drawPixel(x0 - r, y0 + xs, color);

      }

      xs = xe;

    } while (xe < --r);

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawCircleHelper

\*\* Description:             Support function for drawRoundRect()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawCircleHelper( int32\_t x0, int32\_t y0, int32\_t rr, uint8\_t cornername, uint32\_t color)

{

  if (rr <= 0) return;

  int32\_t f     = 1 - rr;

  int32\_t ddF\_x = 1;

  int32\_t ddF\_y = -2 \* rr;

  int32\_t xe    = 0;

  int32\_t xs    = 0;

  int32\_t len   = 0;

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  while (xe < rr--)

  {

    while (f < 0) {

      ++xe;

      f += (ddF\_x += 2);

    }

    f += (ddF\_y += 2);

    if (xe-xs==1) {

      if (cornername & 0x1) { // left top

        drawPixel(x0 - xe, y0 - rr, color);

        drawPixel(x0 - rr, y0 - xe, color);

      }

      if (cornername & 0x2) { // right top

        drawPixel(x0 + rr    , y0 - xe, color);

        drawPixel(x0 + xs + 1, y0 - rr, color);

      }

      if (cornername & 0x4) { // right bottom

        drawPixel(x0 + xs + 1, y0 + rr    , color);

        drawPixel(x0 + rr, y0 + xs + 1, color);

      }

      if (cornername & 0x8) { // left bottom

        drawPixel(x0 - rr, y0 + xs + 1, color);

        drawPixel(x0 - xe, y0 + rr    , color);

      }

    }

    else {

      len = xe - xs++;

      if (cornername & 0x1) { // left top

        drawFastHLine(x0 - xe, y0 - rr, len, color);

        drawFastVLine(x0 - rr, y0 - xe, len, color);

      }

      if (cornername & 0x2) { // right top

        drawFastVLine(x0 + rr, y0 - xe, len, color);

        drawFastHLine(x0 + xs, y0 - rr, len, color);

      }

      if (cornername & 0x4) { // right bottom

        drawFastHLine(x0 + xs, y0 + rr, len, color);

        drawFastVLine(x0 + rr, y0 + xs, len, color);

      }

      if (cornername & 0x8) { // left bottom

        drawFastVLine(x0 - rr, y0 + xs, len, color);

        drawFastHLine(x0 - xe, y0 + rr, len, color);

      }

    }

    xs = xe;

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillCircle

\*\* Description:             draw a filled circle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Optimised midpoint circle algorithm, changed to horizontal lines (faster in sprites)

// Improved algorithm avoids repetition of lines

void TFT\_eSPI::fillCircle(int32\_t x0, int32\_t y0, int32\_t r, uint32\_t color)

{

  int32\_t  x  = 0;

  int32\_t  dx = 1;

  int32\_t  dy = r+r;

  int32\_t  p  = -(r>>1);

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  drawFastHLine(x0 - r, y0, dy+1, color);

  while(x<r){

    if(p>=0) {

      drawFastHLine(x0 - x, y0 + r, dx, color);

      drawFastHLine(x0 - x, y0 - r, dx, color);

      dy-=2;

      p-=dy;

      r--;

    }

    dx+=2;

    p+=dx;

    x++;

    drawFastHLine(x0 - r, y0 + x, dy+1, color);

    drawFastHLine(x0 - r, y0 - x, dy+1, color);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillCircleHelper

\*\* Description:             Support function for fillRoundRect()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Support drawing roundrects, changed to horizontal lines (faster in sprites)

void TFT\_eSPI::fillCircleHelper(int32\_t x0, int32\_t y0, int32\_t r, uint8\_t cornername, int32\_t delta, uint32\_t color)

{

  int32\_t f     = 1 - r;

  int32\_t ddF\_x = 1;

  int32\_t ddF\_y = -r - r;

  int32\_t y     = 0;

  delta++;

  while (y < r) {

    if (f >= 0) {

      if (cornername & 0x1) drawFastHLine(x0 - y, y0 + r, y + y + delta, color);

      if (cornername & 0x2) drawFastHLine(x0 - y, y0 - r, y + y + delta, color);

      r--;

      ddF\_y += 2;

      f     += ddF\_y;

    }

    y++;

    ddF\_x += 2;

    f     += ddF\_x;

    if (cornername & 0x1) drawFastHLine(x0 - r, y0 + y, r + r + delta, color);

    if (cornername & 0x2) drawFastHLine(x0 - r, y0 - y, r + r + delta, color);

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawEllipse

\*\* Description:             Draw a ellipse outline

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawEllipse(int16\_t x0, int16\_t y0, int32\_t rx, int32\_t ry, uint16\_t color)

{

  if (rx<2) return;

  if (ry<2) return;

  int32\_t x, y;

  int32\_t rx2 = rx \* rx;

  int32\_t ry2 = ry \* ry;

  int32\_t fx2 = 4 \* rx2;

  int32\_t fy2 = 4 \* ry2;

  int32\_t s;

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  for (x = 0, y = ry, s = 2\*ry2+rx2\*(1-2\*ry); ry2\*x <= rx2\*y; x++) {

    // These are ordered to minimise coordinate changes in x or y

    // drawPixel can then send fewer bounding box commands

    drawPixel(x0 + x, y0 + y, color);

    drawPixel(x0 - x, y0 + y, color);

    drawPixel(x0 - x, y0 - y, color);

    drawPixel(x0 + x, y0 - y, color);

    if (s >= 0) {

      s += fx2 \* (1 - y);

      y--;

    }

    s += ry2 \* ((4 \* x) + 6);

  }

  for (x = rx, y = 0, s = 2\*rx2+ry2\*(1-2\*rx); rx2\*y <= ry2\*x; y++) {

    // These are ordered to minimise coordinate changes in x or y

    // drawPixel can then send fewer bounding box commands

    drawPixel(x0 + x, y0 + y, color);

    drawPixel(x0 - x, y0 + y, color);

    drawPixel(x0 - x, y0 - y, color);

    drawPixel(x0 + x, y0 - y, color);

    if (s >= 0)

    {

      s += fy2 \* (1 - x);

      x--;

    }

    s += rx2 \* ((4 \* y) + 6);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillEllipse

\*\* Description:             draw a filled ellipse

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillEllipse(int16\_t x0, int16\_t y0, int32\_t rx, int32\_t ry, uint16\_t color)

{

  if (rx<2) return;

  if (ry<2) return;

  int32\_t x, y;

  int32\_t rx2 = rx \* rx;

  int32\_t ry2 = ry \* ry;

  int32\_t fx2 = 4 \* rx2;

  int32\_t fy2 = 4 \* ry2;

  int32\_t s;

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  for (x = 0, y = ry, s = 2\*ry2+rx2\*(1-2\*ry); ry2\*x <= rx2\*y; x++) {

    drawFastHLine(x0 - x, y0 - y, x + x + 1, color);

    drawFastHLine(x0 - x, y0 + y, x + x + 1, color);

    if (s >= 0) {

      s += fx2 \* (1 - y);

      y--;

    }

    s += ry2 \* ((4 \* x) + 6);

  }

  for (x = rx, y = 0, s = 2\*rx2+ry2\*(1-2\*rx); rx2\*y <= ry2\*x; y++) {

    drawFastHLine(x0 - x, y0 - y, x + x + 1, color);

    drawFastHLine(x0 - x, y0 + y, x + x + 1, color);

    if (s >= 0) {

      s += fy2 \* (1 - x);

      x--;

    }

    s += rx2 \* ((4 \* y) + 6);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillScreen

\*\* Description:             Clear the screen to defined colour

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillScreen(uint32\_t color)

{

  fillRect(0, 0, \_width, \_height, color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawRect

\*\* Description:             Draw a rectangle outline

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Draw a rectangle

void TFT\_eSPI::drawRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint32\_t color)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  drawFastHLine(x, y, w, color);

  drawFastHLine(x, y + h - 1, w, color);

  // Avoid drawing corner pixels twice

  drawFastVLine(x, y+1, h-2, color);

  drawFastVLine(x + w - 1, y+1, h-2, color);

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawRoundRect

\*\* Description:             Draw a rounded corner rectangle outline

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Draw a rounded rectangle

void TFT\_eSPI::drawRoundRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, int32\_t r, uint32\_t color)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  // smarter version

  drawFastHLine(x + r  , y    , w - r - r, color); // Top

  drawFastHLine(x + r  , y + h - 1, w - r - r, color); // Bottom

  drawFastVLine(x    , y + r  , h - r - r, color); // Left

  drawFastVLine(x + w - 1, y + r  , h - r - 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);

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillRoundRect

\*\* Description:             Draw a rounded corner filled rectangle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Fill a rounded rectangle, changed to horizontal lines (faster in sprites)

void TFT\_eSPI::fillRoundRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, int32\_t r, uint32\_t color)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  // smarter version

  fillRect(x, y + r, w, h - r - r, color);

  // draw four corners

  fillCircleHelper(x + r, y + h - r - 1, r, 1, w - r - r - 1, color);

  fillCircleHelper(x + r    , y + r, r, 2, w - r - r - 1, color);

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawTriangle

\*\* Description:             Draw a triangle outline using 3 arbitrary points

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Draw a triangle

void TFT\_eSPI::drawTriangle(int32\_t x0, int32\_t y0, int32\_t x1, int32\_t y1, int32\_t x2, int32\_t y2, uint32\_t color)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  drawLine(x0, y0, x1, y1, color);

  drawLine(x1, y1, x2, y2, color);

  drawLine(x2, y2, x0, y0, color);

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillTriangle

\*\* Description:             Draw a filled triangle using 3 arbitrary points

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Fill a triangle - original Adafruit function works well and code footprint is small

void TFT\_eSPI::fillTriangle ( int32\_t x0, int32\_t y0, int32\_t x1, int32\_t y1, int32\_t x2, int32\_t y2, uint32\_t color)

{

  int32\_t a, b, y, last;

  // Sort coordinates by Y order (y2 >= y1 >= y0)

  if (y0 > y1) {

    swap\_coord(y0, y1); swap\_coord(x0, x1);

  }

  if (y1 > y2) {

    swap\_coord(y2, y1); swap\_coord(x2, x1);

  }

  if (y0 > y1) {

    swap\_coord(y0, y1); swap\_coord(x0, x1);

  }

  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;

    drawFastHLine(a, y0, b - a + 1, color);

    return;

  }

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  int32\_t

  dx01 = x1 - x0,

  dy01 = y1 - y0,

  dx02 = x2 - x0,

  dy02 = y2 - y0,

  dx12 = x2 - x1,

  dy12 = y2 - y1,

  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;

    if (a > b) swap\_coord(a, b);

    drawFastHLine(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 = dx12 \* (y - y1);

  sb = dx02 \* (y - y0);

  for (; y <= y2; y++) {

    a   = x1 + sa / dy12;

    b   = x0 + sb / dy02;

    sa += dx12;

    sb += dx02;

    if (a > b) swap\_coord(a, b);

    drawFastHLine(a, y, b - a + 1, color);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawBitmap

\*\* Description:             Draw an image stored in an array on the TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawBitmap(int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t color)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  int32\_t i, j, byteWidth = (w + 7) / 8;

  for (j = 0; j < h; j++) {

    for (i = 0; i < w; i++ ) {

      if (pgm\_read\_byte(bitmap + j \* byteWidth + i / 8) & (128 >> (i & 7))) {

        drawPixel(x + i, y + j, color);

      }

    }

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawBitmap

\*\* Description:             Draw an image stored in an array on the TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawBitmap(int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t fgcolor, uint16\_t bgcolor)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  int32\_t i, j, byteWidth = (w + 7) / 8;

  for (j = 0; j < h; j++) {

    for (i = 0; i < w; i++ ) {

      if (pgm\_read\_byte(bitmap + j \* byteWidth + i / 8) & (128 >> (i & 7)))

           drawPixel(x + i, y + j, fgcolor);

      else drawPixel(x + i, y + j, bgcolor);

    }

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawXBitmap

\*\* Description:             Draw an image stored in an XBM array onto the TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawXBitmap(int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t color)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  int32\_t i, j, byteWidth = (w + 7) / 8;

  for (j = 0; j < h; j++) {

    for (i = 0; i < w; i++ ) {

      if (pgm\_read\_byte(bitmap + j \* byteWidth + i / 8) & (1 << (i & 7))) {

        drawPixel(x + i, y + j, color);

      }

    }

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawXBitmap

\*\* Description:             Draw an XBM image with foreground and background colors

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawXBitmap(int16\_t x, int16\_t y, const uint8\_t \*bitmap, int16\_t w, int16\_t h, uint16\_t color, uint16\_t bgcolor)

{

  //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  int32\_t i, j, byteWidth = (w + 7) / 8;

  for (j = 0; j < h; j++) {

    for (i = 0; i < w; i++ ) {

      if (pgm\_read\_byte(bitmap + j \* byteWidth + i / 8) & (1 << (i & 7)))

           drawPixel(x + i, y + j,   color);

      else drawPixel(x + i, y + j, bgcolor);

    }

  }

  inTransaction = lockTransaction;

  end\_tft\_write();              // Does nothing if Sprite class uses this function

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setCursor

\*\* Description:             Set the text cursor x,y position

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setCursor(int16\_t x, int16\_t y)

{

  cursor\_x = x;

  cursor\_y = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setCursor

\*\* Description:             Set the text cursor x,y position and font

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setCursor(int16\_t x, int16\_t y, uint8\_t font)

{

  textfont = font;

  cursor\_x = x;

  cursor\_y = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getCursorX

\*\* Description:             Get the text cursor x position

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::getCursorX(void)

{

  return cursor\_x;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getCursorY

\*\* Description:             Get the text cursor y position

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::getCursorY(void)

{

  return cursor\_y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextSize

\*\* Description:             Set the text size multiplier

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextSize(uint8\_t s)

{

  if (s>7) s = 7; // Limit the maximum size multiplier so byte variables can be used for rendering

  textsize = (s > 0) ? s : 1; // Don't allow font size 0

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextColor

\*\* Description:             Set the font foreground colour (background is transparent)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextColor(uint16\_t c)

{

  // For 'transparent' background, we'll set the bg

  // to the same as fg instead of using a flag

  textcolor = textbgcolor = c;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextColor

\*\* Description:             Set the font foreground and background colour

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Smooth fonts use the background colour for anti-aliasing and by default the

// background is not filled. If bgfill = true, then a smooth font background fill will

// be used.

void TFT\_eSPI::setTextColor(uint16\_t c, uint16\_t b, bool bgfill)

{

  textcolor   = c;

  textbgcolor = b;

  \_fillbg     = bgfill;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setPivot

\*\* Description:             Set the pivot point on the TFT

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*x\*/

void TFT\_eSPI::setPivot(int16\_t x, int16\_t y)

{

  \_xPivot = x;

  \_yPivot = y;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getPivotX

\*\* Description:             Get the x pivot position

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::getPivotX(void)

{

  return \_xPivot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getPivotY

\*\* Description:             Get the y pivot position

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::getPivotY(void)

{

  return \_yPivot;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setBitmapColor

\*\* Description:             Set the foreground foreground and background colour

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setBitmapColor(uint16\_t c, uint16\_t b)

{

  if (c == b) b = ~c;

  bitmap\_fg = c;

  bitmap\_bg = b;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextWrap

\*\* Description:             Define if text should wrap at end of line

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextWrap(bool wrapX, bool wrapY)

{

  textwrapX = wrapX;

  textwrapY = wrapY;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextDatum

\*\* Description:             Set the text position reference datum

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextDatum(uint8\_t d)

{

  textdatum = d;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextPadding

\*\* Description:             Define padding width (aids erasing old text and numbers)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextPadding(uint16\_t x\_width)

{

  padX = x\_width;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextPadding

\*\* Description:             Define padding width (aids erasing old text and numbers)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t TFT\_eSPI::getTextPadding(void)

{

  return padX;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getRotation

\*\* Description:             Return the rotation value (as used by setRotation())

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t TFT\_eSPI::getRotation(void)

{

  return rotation;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getTextDatum

\*\* Description:             Return the text datum value (as used by setTextDatum())

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t TFT\_eSPI::getTextDatum(void)

{

  return textdatum;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           width

\*\* Description:             Return the pixel width of display (per current rotation)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Return the size of the display (per current rotation)

int16\_t TFT\_eSPI::width(void)

{

  if (\_vpDatum) return \_xWidth;

  return \_width;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           height

\*\* Description:             Return the pixel height of display (per current rotation)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::height(void)

{

  if (\_vpDatum) return \_yHeight;

  return \_height;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           textWidth

\*\* Description:             Return the width in pixels of a string in a given font

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::textWidth(const String& string)

{

  int16\_t len = string.length() + 2;

  char buffer[len];

  string.toCharArray(buffer, len);

  return textWidth(buffer, textfont);

}

int16\_t TFT\_eSPI::textWidth(const String& string, uint8\_t font)

{

  int16\_t len = string.length() + 2;

  char buffer[len];

  string.toCharArray(buffer, len);

  return textWidth(buffer, font);

}

int16\_t TFT\_eSPI::textWidth(const char \*string)

{

  return textWidth(string, textfont);

}

int16\_t TFT\_eSPI::textWidth(const char \*string, uint8\_t font)

{

  int32\_t str\_width = 0;

  uint16\_t uniCode  = 0;

#ifdef SMOOTH\_FONT

  if(fontLoaded) {

    while (\*string) {

      uniCode = decodeUTF8(\*string++);

      if (uniCode) {

        if (uniCode == 0x20) str\_width += gFont.spaceWidth;

        else {

          uint16\_t gNum = 0;

          bool found = getUnicodeIndex(uniCode, &gNum);

          if (found) {

            if(str\_width == 0 && gdX[gNum] < 0) str\_width -= gdX[gNum];

            if (\*string || isDigits) str\_width += gxAdvance[gNum];

            else str\_width += (gdX[gNum] + gWidth[gNum]);

          }

          else str\_width += gFont.spaceWidth + 1;

        }

      }

    }

    isDigits = false;

    return str\_width;

  }

#endif

  if (font>1 && font<9) {

    char \*widthtable = (char \*)pgm\_read\_dword( &(fontdata[font].widthtbl ) ) - 32; //subtract the 32 outside the loop

    while (\*string) {

      uniCode = \*(string++);

      if (uniCode > 31 && uniCode < 128)

      str\_width += pgm\_read\_byte( widthtable + uniCode); // Normally we need to subtract 32 from uniCode

      else str\_width += pgm\_read\_byte( widthtable + 32); // Set illegal character = space width

    }

  }

  else {

#ifdef LOAD\_GFXFF

    if(gfxFont) { // New font

      while (\*string) {

        uniCode = decodeUTF8(\*string++);

        if ((uniCode >= pgm\_read\_word(&gfxFont->first)) && (uniCode <= pgm\_read\_word(&gfxFont->last ))) {

          uniCode -= pgm\_read\_word(&gfxFont->first);

          GFXglyph \*glyph  = &(((GFXglyph \*)pgm\_read\_dword(&gfxFont->glyph))[uniCode]);

          // If this is not the  last character or is a digit then use xAdvance

          if (\*string  || isDigits) str\_width += pgm\_read\_byte(&glyph->xAdvance);

          // Else use the offset plus width since this can be bigger than xAdvance

          else str\_width += ((int8\_t)pgm\_read\_byte(&glyph->xOffset) + pgm\_read\_byte(&glyph->width));

        }

      }

    }

    else

#endif

    {

#ifdef LOAD\_GLCD

      while (\*string++) str\_width += 6;

#endif

    }

  }

  isDigits = false;

  return str\_width \* textsize;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fontsLoaded

\*\* Description:             return an encoded 16 bit value showing the fonts loaded

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Returns a value showing which fonts are loaded (bit N set =  Font N loaded)

uint16\_t TFT\_eSPI::fontsLoaded(void)

{

  return fontsloaded;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fontHeight

\*\* Description:             return the height of a font (yAdvance for free fonts)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::fontHeight(int16\_t font)

{

#ifdef SMOOTH\_FONT

  if(fontLoaded) return gFont.yAdvance;

#endif

#ifdef LOAD\_GFXFF

  if (font==1) {

    if(gfxFont) { // New font

      return pgm\_read\_byte(&gfxFont->yAdvance) \* textsize;

    }

  }

#endif

  return pgm\_read\_byte( &fontdata[font].height ) \* textsize;

}

int16\_t TFT\_eSPI::fontHeight(void)

{

  return fontHeight(textfont);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawChar

\*\* Description:             draw a single character in the GLCD or GFXFF font

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawChar(int32\_t x, int32\_t y, uint16\_t c, uint32\_t color, uint32\_t bg, uint8\_t size)

{

  if (\_vpOoB) return;

  if (c < 32) return;

#ifdef LOAD\_GLCD

//>>>>>>>>>>>>>>>>>>

  #ifdef LOAD\_GFXFF

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

  #endif

//>>>>>>>>>>>>>>>>>>

  int32\_t xd = x + \_xDatum;

  int32\_t yd = y + \_yDatum;

  if ((xd >= \_vpW)                 || // Clip right

     ( yd >= \_vpH)                 || // Clip bottom

     ((xd + 6 \* size - 1) < \_vpX)  || // Clip left

     ((yd + 8 \* size - 1) < \_vpY))    // Clip top

    return;

  bool fillbg = (bg != color);

  bool clip = xd < \_vpX || xd + 6  \* textsize >= \_vpW || yd < \_vpY || yd + 8 \* textsize >= \_vpH;

  if ((size==1) && fillbg && !clip) {

    uint8\_t column[6];

    uint8\_t mask = 0x1;

    begin\_tft\_write();

    setWindow(xd, yd, xd+5, yd+7);

    for (int8\_t i = 0; i < 5; i++ ) column[i] = pgm\_read\_byte(font + (c \* 5) + i);

    column[5] = 0;

    for (int8\_t j = 0; j < 8; j++) {

      for (int8\_t k = 0; k < 5; k++ ) {

        if (column[k] & mask) {tft\_Write\_16(color);}

        else {tft\_Write\_16(bg);}

      }

      mask <<= 1;

      tft\_Write\_16(bg);

    }

    end\_tft\_write();

  }

  else {

    //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

    inTransaction = true;

    for (int8\_t i = 0; i < 6; i++ ) {

      uint8\_t line;

      if (i == 5)

        line = 0x0;

      else

        line = pgm\_read\_byte(font + (c \* 5) + i);

      if (size == 1 && !fillbg) { // default size

        for (int8\_t j = 0; j < 8; j++) {

          if (line & 0x1) drawPixel(x + i, y + j, color);

          line >>= 1;

        }

      }

      else {  // big size or clipped

        for (int8\_t j = 0; j < 8; j++) {

          if (line & 0x1) fillRect(x + (i \* size), y + (j \* size), size, size, color);

          else if (fillbg) fillRect(x + i \* size, y + j \* size, size, size, bg);

          line >>= 1;

        }

      }

    }

    inTransaction = lockTransaction;

    end\_tft\_write();              // Does nothing if Sprite class uses this function

  }

//>>>>>>>>>>>>>>>>>>>>>>>>>>>

  #ifdef LOAD\_GFXFF

  } else { // Custom font

  #endif

//>>>>>>>>>>>>>>>>>>>>>>>>>>>

#endif // LOAD\_GLCD

#ifdef LOAD\_GFXFF

    // Filter out bad characters not present in font

    if ((c >= pgm\_read\_word(&gfxFont->first)) && (c <= pgm\_read\_word(&gfxFont->last ))) {

      //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

      inTransaction = true;

//>>>>>>>>>>>>>>>>>>>>>>>>>>>

      c -= pgm\_read\_word(&gfxFont->first);

      GFXglyph \*glyph  = &(((GFXglyph \*)pgm\_read\_dword(&gfxFont->glyph))[c]);

      uint8\_t  \*bitmap = (uint8\_t \*)pgm\_read\_dword(&gfxFont->bitmap);

      uint32\_t bo = pgm\_read\_word(&glyph->bitmapOffset);

      uint8\_t  w  = pgm\_read\_byte(&glyph->width),

               h  = 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);

      uint8\_t  xx, yy, bits=0, bit=0;

      int16\_t  xo16 = 0, yo16 = 0;

      if(size > 1) {

        xo16 = xo;

        yo16 = yo;

      }

      // GFXFF rendering speed up

      uint16\_t hpc = 0; // Horizontal foreground pixel count

      for(yy=0; yy<h; yy++) {

        for(xx=0; xx<w; xx++) {

          if(bit == 0) {

            bits = pgm\_read\_byte(&bitmap[bo++]);

            bit  = 0x80;

          }

          if(bits & bit) hpc++;

          else {

           if (hpc) {

              if(size == 1) drawFastHLine(x+xo+xx-hpc, y+yo+yy, hpc, color);

              else fillRect(x+(xo16+xx-hpc)\*size, y+(yo16+yy)\*size, size\*hpc, size, color);

              hpc=0;

            }

          }

          bit >>= 1;

        }

        // Draw pixels for this line as we are about to increment yy

        if (hpc) {

          if(size == 1) drawFastHLine(x+xo+xx-hpc, y+yo+yy, hpc, color);

          else fillRect(x+(xo16+xx-hpc)\*size, y+(yo16+yy)\*size, size\*hpc, size, color);

          hpc=0;

        }

      }

      inTransaction = lockTransaction;

      end\_tft\_write();              // Does nothing if Sprite class uses this function

    }

#endif

#ifdef LOAD\_GLCD

  #ifdef LOAD\_GFXFF

  } // End classic vs custom font

  #endif

#else

  #ifndef LOAD\_GFXFF

    // Avoid warnings if fonts are disabled

    x = x;

    y = y;

    color = color;

    bg = bg;

    size = size;

  #endif

#endif

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setAddrWindow

\*\* Description:             define an area to receive a stream of pixels

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Chip select is high at the end of this function

void TFT\_eSPI::setAddrWindow(int32\_t x0, int32\_t y0, int32\_t w, int32\_t h)

{

  begin\_tft\_write();

  setWindow(x0, y0, x0 + w - 1, y0 + h - 1);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setWindow

\*\* Description:             define an area to receive a stream of pixels

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Chip select stays low, call begin\_tft\_write first. Use setAddrWindow() from sketches

void TFT\_eSPI::setWindow(int32\_t x0, int32\_t y0, int32\_t x1, int32\_t y1)

{

  //begin\_tft\_write(); // Must be called before setWindow

  addr\_row = 0xFFFF;

  addr\_col = 0xFFFF;

#if defined (ILI9225\_DRIVER)

  if (rotation & 0x01) { swap\_coord(x0, y0); swap\_coord(x1, y1); }

  SPI\_BUSY\_CHECK;

  DC\_C; tft\_Write\_8(TFT\_CASET1);

  DC\_D; tft\_Write\_16(x0);

  DC\_C; tft\_Write\_8(TFT\_CASET2);

  DC\_D; tft\_Write\_16(x1);

  DC\_C; tft\_Write\_8(TFT\_PASET1);

  DC\_D; tft\_Write\_16(y0);

  DC\_C; tft\_Write\_8(TFT\_PASET2);

  DC\_D; tft\_Write\_16(y1);

  DC\_C; tft\_Write\_8(TFT\_RAM\_ADDR1);

  DC\_D; tft\_Write\_16(x0);

  DC\_C; tft\_Write\_8(TFT\_RAM\_ADDR2);

  DC\_D; tft\_Write\_16(y0);

  // write to RAM

  DC\_C; tft\_Write\_8(TFT\_RAMWR);

  DC\_D;

  // Temporary solution is to include the RP2040 code here

  #if (defined(ARDUINO\_ARCH\_RP2040)  || defined (ARDUINO\_ARCH\_MBED)) && !defined(RP2040\_PIO\_INTERFACE)

    // For ILI9225 and RP2040 the slower Arduino SPI transfer calls were used, so need to swap back to 16 bit mode

    while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

    hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (16 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

  #endif

#elif defined (SSD1351\_DRIVER)

  if (rotation & 1) {

    swap\_coord(x0, y0);

    swap\_coord(x1, y1);

  }

  SPI\_BUSY\_CHECK;

  DC\_C; tft\_Write\_8(TFT\_CASET);

  DC\_D; tft\_Write\_16(x1 | (x0 << 8));

  DC\_C; tft\_Write\_8(TFT\_PASET);

  DC\_D; tft\_Write\_16(y1 | (y0 << 8));

  DC\_C; tft\_Write\_8(TFT\_RAMWR);

  DC\_D;

#else

  #if defined (SSD1963\_DRIVER)

    if ((rotation & 0x1) == 0) { swap\_coord(x0, y0); swap\_coord(x1, y1); }

  #endif

  #ifdef CGRAM\_OFFSET

    x0+=colstart;

    x1+=colstart;

    y0+=rowstart;

    y1+=rowstart;

  #endif

  // Temporary solution is to include the RP2040 optimised code here

  #if (defined(ARDUINO\_ARCH\_RP2040)  || defined (ARDUINO\_ARCH\_MBED))

    #if !defined(RP2040\_PIO\_INTERFACE)

      // Use hardware SPI port, this code does not swap from 8 to 16 bit

      // to avoid the spi\_set\_format() call overhead

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_C;

      #if !defined (SPI\_18BIT\_DRIVER)

        #if  defined (RPI\_DISPLAY\_TYPE) // RPi TFT type always needs 16 bit transfers

          hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (16 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

        #else

          hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (8 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

        #endif

      #endif

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_CASET;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_D;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x0>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x0;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x1>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x1;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_C;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_PASET;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_D;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y0>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y0;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y1>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y1;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_C;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_RAMWR;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      #if !defined (SPI\_18BIT\_DRIVER)

        hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (16 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

      #endif

      DC\_D;

    #else

      // This is for the RP2040 and PIO interface (SPI or parallel)

      WAIT\_FOR\_STALL;

      tft\_pio->sm[pio\_sm].instr = pio\_instr\_addr;

      TX\_FIFO = TFT\_CASET;

      TX\_FIFO = (x0<<16) | x1;

      TX\_FIFO = TFT\_PASET;

      TX\_FIFO = (y0<<16) | y1;

      TX\_FIFO = TFT\_RAMWR;

    #endif

  #else

    SPI\_BUSY\_CHECK;

    DC\_C; tft\_Write\_8(TFT\_CASET);

    DC\_D; tft\_Write\_32C(x0, x1);

    DC\_C; tft\_Write\_8(TFT\_PASET);

    DC\_D; tft\_Write\_32C(y0, y1);

    DC\_C; tft\_Write\_8(TFT\_RAMWR);

    DC\_D;

  #endif // RP2040 SPI

#endif

  //end\_tft\_write(); // Must be called after setWindow

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           readAddrWindow

\*\* Description:             define an area to read a stream of pixels

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::readAddrWindow(int32\_t xs, int32\_t ys, int32\_t w, int32\_t h)

{

  //begin\_tft\_write(); // Must be called before readAddrWindow or CS set low

  int32\_t xe = xs + w - 1;

  int32\_t ye = ys + h - 1;

  addr\_col = 0xFFFF;

  addr\_row = 0xFFFF;

#if defined (SSD1963\_DRIVER)

  if ((rotation & 0x1) == 0) { swap\_coord(xs, ys); swap\_coord(xe, ye); }

#endif

#ifdef CGRAM\_OFFSET

  xs += colstart;

  xe += colstart;

  ys += rowstart;

  ye += rowstart;

#endif

  // Temporary solution is to include the RP2040 optimised code here

#if (defined(ARDUINO\_ARCH\_RP2040)  || defined (ARDUINO\_ARCH\_MBED)) && !defined(RP2040\_PIO\_INTERFACE)

  // Use hardware SPI port, this code does not swap from 8 to 16 bit

  // to avoid the spi\_set\_format() call overhead

  while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  DC\_C;

  hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (8 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_CASET;

  while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  DC\_D;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)xs>>8;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)xs;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)xe>>8;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)xe;

  while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  DC\_C;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_PASET;

  while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  DC\_D;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)ys>>8;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)ys;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)ye>>8;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)ye;

  while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  DC\_C;

  spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_RAMRD;

  while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  DC\_D;

  // Flush the rx buffer and reset overflow flag

  while (spi\_is\_readable(SPI\_X)) (void)spi\_get\_hw(SPI\_X)->dr;

  spi\_get\_hw(SPI\_X)->icr = SPI\_SSPICR\_RORIC\_BITS;

#else

  // Column addr set

  DC\_C; tft\_Write\_8(TFT\_CASET);

  DC\_D; tft\_Write\_32C(xs, xe);

  // Row addr set

  DC\_C; tft\_Write\_8(TFT\_PASET);

  DC\_D; tft\_Write\_32C(ys, ye);

  // Read CGRAM command

  DC\_C; tft\_Write\_8(TFT\_RAMRD);

  DC\_D;

#endif // RP2040 SPI

  //end\_tft\_write(); // Must be called after readAddrWindow or CS set high

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawPixel

\*\* Description:             push a single pixel at an arbitrary position

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawPixel(int32\_t x, int32\_t y, uint32\_t color)

{

  if (\_vpOoB) return;

  x+= \_xDatum;

  y+= \_yDatum;

  // Range checking

  if ((x < \_vpX) || (y < \_vpY) ||(x >= \_vpW) || (y >= \_vpH)) return;

#ifdef CGRAM\_OFFSET

  x+=colstart;

  y+=rowstart;

#endif

#if (defined (MULTI\_TFT\_SUPPORT) || defined (GC9A01\_DRIVER)) && !defined (ILI9225\_DRIVER)

  addr\_row = 0xFFFF;

  addr\_col = 0xFFFF;

#endif

  begin\_tft\_write();

#if defined (ILI9225\_DRIVER)

  if (rotation & 0x01) { swap\_coord(x, y); }

  SPI\_BUSY\_CHECK;

  // Set window to full screen to optimise sequential pixel rendering

  if (addr\_row != 0x9225) {

    addr\_row = 0x9225; // addr\_row used for flag

    DC\_C; tft\_Write\_8(TFT\_CASET1);

    DC\_D; tft\_Write\_16(0);

    DC\_C; tft\_Write\_8(TFT\_CASET2);

    DC\_D; tft\_Write\_16(175);

    DC\_C; tft\_Write\_8(TFT\_PASET1);

    DC\_D; tft\_Write\_16(0);

    DC\_C; tft\_Write\_8(TFT\_PASET2);

    DC\_D; tft\_Write\_16(219);

  }

  // Define pixel coordinate

  DC\_C; tft\_Write\_8(TFT\_RAM\_ADDR1);

  DC\_D; tft\_Write\_16(x);

  DC\_C; tft\_Write\_8(TFT\_RAM\_ADDR2);

  DC\_D; tft\_Write\_16(y);

  // write to RAM

  DC\_C; tft\_Write\_8(TFT\_RAMWR);

  #if defined(TFT\_PARALLEL\_8\_BIT) || defined(TFT\_PARALLEL\_16\_BIT) || !defined(ESP32)

    DC\_D; tft\_Write\_16(color);

  #else

    DC\_D; tft\_Write\_16N(color);

  #endif

// Temporary solution is to include the RP2040 optimised code here

#elif (defined (ARDUINO\_ARCH\_RP2040) || defined (ARDUINO\_ARCH\_MBED)) && !defined (SSD1351\_DRIVER)

  #if defined (SSD1963\_DRIVER)

    if ((rotation & 0x1) == 0) { swap\_coord(x, y); }

  #endif

  #if !defined(RP2040\_PIO\_INTERFACE)

    while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

    #if  defined (RPI\_DISPLAY\_TYPE) // RPi TFT type always needs 16 bit transfers

      hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (16 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

    #else

      hw\_write\_masked(&spi\_get\_hw(SPI\_X)->cr0, (8 - 1) << SPI\_SSPCR0\_DSS\_LSB, SPI\_SSPCR0\_DSS\_BITS);

    #endif

    if (addr\_col != x) {

      DC\_C;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_CASET;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS){};

      DC\_D;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)x;

      addr\_col = x;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

    }

    if (addr\_row != y) {

      DC\_C;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_PASET;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_D;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y>>8;

      spi\_get\_hw(SPI\_X)->dr = (uint32\_t)y;

      addr\_row = y;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

    }

    DC\_C;

    spi\_get\_hw(SPI\_X)->dr = (uint32\_t)TFT\_RAMWR;

    #if defined (SPI\_18BIT\_DRIVER) // SPI 18 bit colour

      uint8\_t r = (color & 0xF800)>>8;

      uint8\_t g = (color & 0x07E0)>>3;

      uint8\_t b = (color & 0x001F)<<3;

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_D;

      tft\_Write\_8N(r); tft\_Write\_8N(g); tft\_Write\_8N(b);

    #else

      while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

      DC\_D;

      #if  defined (RPI\_DISPLAY\_TYPE) // RPi TFT type always needs 16 bit transfers

        spi\_get\_hw(SPI\_X)->dr = (uint32\_t)color;

      #else

        spi\_get\_hw(SPI\_X)->dr = (uint32\_t)color>>8;

        spi\_get\_hw(SPI\_X)->dr = (uint32\_t)color;

      #endif

    #endif

    while (spi\_get\_hw(SPI\_X)->sr & SPI\_SSPSR\_BSY\_BITS) {};

  #else

    // This is for the RP2040 and PIO interface (SPI or parallel)

    WAIT\_FOR\_STALL;

    tft\_pio->sm[pio\_sm].instr = pio\_instr\_addr;

    TX\_FIFO = TFT\_CASET;

    TX\_FIFO = (x<<16) | x;

    TX\_FIFO = TFT\_PASET;

    TX\_FIFO = (y<<16) | y;

    TX\_FIFO = TFT\_RAMWR;

    //DC set high by PIO

    #if  defined (SPI\_18BIT\_DRIVER)

      TX\_FIFO = ((color & 0xF800)<<8) | ((color & 0x07E0)<<5) | ((color & 0x001F)<<3);

    #else

      TX\_FIFO = color;

    #endif

  #endif

#else

  #if defined (SSD1963\_DRIVER)

    if ((rotation & 0x1) == 0) { swap\_coord(x, y); }

  #endif

    SPI\_BUSY\_CHECK;

  #if defined (SSD1351\_DRIVER)

    if (rotation & 0x1) { swap\_coord(x, y); }

    // No need to send x if it has not changed (speeds things up)

    if (addr\_col != x) {

      DC\_C; tft\_Write\_8(TFT\_CASET);

      DC\_D; tft\_Write\_16(x | (x << 8));

      addr\_col = x;

    }

    // No need to send y if it has not changed (speeds things up)

    if (addr\_row != y) {

      DC\_C; tft\_Write\_8(TFT\_PASET);

      DC\_D; tft\_Write\_16(y | (y << 8));

      addr\_row = y;

    }

  #else

    // No need to send x if it has not changed (speeds things up)

    if (addr\_col != x) {

      DC\_C; tft\_Write\_8(TFT\_CASET);

      DC\_D; tft\_Write\_32D(x);

      addr\_col = x;

    }

    // No need to send y if it has not changed (speeds things up)

    if (addr\_row != y) {

      DC\_C; tft\_Write\_8(TFT\_PASET);

      DC\_D; tft\_Write\_32D(y);

      addr\_row = y;

    }

  #endif

  DC\_C; tft\_Write\_8(TFT\_RAMWR);

  #if defined(TFT\_PARALLEL\_8\_BIT) || defined(TFT\_PARALLEL\_16\_BIT) || !defined(ESP32)

    DC\_D; tft\_Write\_16(color);

  #else

    DC\_D; tft\_Write\_16N(color);

  #endif

#endif

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushColor

\*\* Description:             push a single pixel

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushColor(uint16\_t color)

{

  begin\_tft\_write();

  SPI\_BUSY\_CHECK;

  tft\_Write\_16N(color);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushColor

\*\* Description:             push a single colour to "len" pixels

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushColor(uint16\_t color, uint32\_t len)

{

  begin\_tft\_write();

  pushBlock(color, len);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           startWrite

\*\* Description:             begin transaction with CS low, MUST later call endWrite

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::startWrite(void)

{

  begin\_tft\_write();

  lockTransaction = true; // Lock transaction for all sequentially run sketch functions

  inTransaction = true;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           endWrite

\*\* Description:             end transaction with CS high

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::endWrite(void)

{

  lockTransaction = false; // Release sketch induced transaction lock

  inTransaction = false;

  DMA\_BUSY\_CHECK;          // Safety check - user code should have checked this!

  end\_tft\_write();         // Release SPI bus

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           writeColor (use startWrite() and endWrite() before & after)

\*\* Description:             raw write of "len" pixels avoiding transaction check

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::writeColor(uint16\_t color, uint32\_t len)

{

  pushBlock(color, len);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushColors

\*\* Description:             push an array of pixels for 16 bit raw image drawing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Assumed that setAddrWindow() has previously been called

// len is number of bytes, not pixels

void TFT\_eSPI::pushColors(uint8\_t \*data, uint32\_t len)

{

  begin\_tft\_write();

  pushPixels(data, len>>1);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           pushColors

\*\* Description:             push an array of pixels, for image drawing

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::pushColors(uint16\_t \*data, uint32\_t len, bool swap)

{

  begin\_tft\_write();

  if (swap) {swap = \_swapBytes; \_swapBytes = true; }

  pushPixels(data, len);

  \_swapBytes = swap; // Restore old value

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawLine

\*\* Description:             draw a line between 2 arbitrary points

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Bresenham's algorithm - thx wikipedia - speed enhanced by Bodmer to use

// an efficient FastH/V Line draw routine for line segments of 2 pixels or more

void TFT\_eSPI::drawLine(int32\_t x0, int32\_t y0, int32\_t x1, int32\_t y1, uint32\_t color)

{

  if (\_vpOoB) return;

  //begin\_tft\_write();       // Sprite class can use this function, avoiding begin\_tft\_write()

  inTransaction = true;

  //x+= \_xDatum;             // Not added here, added by drawPixel & drawFastXLine

  //y+= \_yDatum;

  bool steep = abs(y1 - y0) > abs(x1 - x0);

  if (steep) {

    swap\_coord(x0, y0);

    swap\_coord(x1, y1);

  }

  if (x0 > x1) {

    swap\_coord(x0, x1);

    swap\_coord(y0, y1);

  }

  int32\_t dx = x1 - x0, dy = abs(y1 - y0);;

  int32\_t err = dx >> 1, ystep = -1, xs = x0, dlen = 0;

  if (y0 < y1) ystep = 1;

  // Split into steep and not steep for FastH/V separation

  if (steep) {

    for (; x0 <= x1; x0++) {

      dlen++;

      err -= dy;

      if (err < 0) {

        if (dlen == 1) drawPixel(y0, xs, color);

        else drawFastVLine(y0, xs, dlen, color);

        dlen = 0;

        y0 += ystep; xs = x0 + 1;

        err += dx;

      }

    }

    if (dlen) drawFastVLine(y0, xs, dlen, color);

  }

  else

  {

    for (; x0 <= x1; x0++) {

      dlen++;

      err -= dy;

      if (err < 0) {

        if (dlen == 1) drawPixel(xs, y0, color);

        else drawFastHLine(xs, y0, dlen, color);

        dlen = 0;

        y0 += ystep; xs = x0 + 1;

        err += dx;

      }

    }

    if (dlen) drawFastHLine(xs, y0, dlen, color);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Description:  Constants for anti-aliased line drawing on TFT and in Sprites

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

constexpr float PixelAlphaGain   = 255.0;

constexpr float LoAlphaTheshold  = 1.0/32.0;

constexpr float HiAlphaTheshold  = 1.0 - LoAlphaTheshold;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawPixel (alpha blended)

\*\* Description:             Draw a pixel blended with the screen or bg pixel colour

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t TFT\_eSPI::drawPixel(int32\_t x, int32\_t y, uint32\_t color, uint8\_t alpha, uint32\_t bg\_color)

{

  if (bg\_color == 0x00FFFFFF) bg\_color = readPixel(x, y);

  color = alphaBlend(alpha, color, bg\_color);

  drawPixel(x, y, color);

  return color;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillSmoothCircle

\*\* Description:             Draw a filled anti-aliased circle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillSmoothCircle(int32\_t x, int32\_t y, int32\_t r, uint32\_t color, uint32\_t bg\_color)

{

  if (r <= 0) return;

  inTransaction = true;

  drawFastHLine(x - r, y, 2 \* r + 1, color);

  int32\_t xs = 1;

  int32\_t cx = 0;

  int32\_t r1 = r \* r;

  r++;

  int32\_t r2 = r \* r;

  for (int32\_t cy = r - 1; cy > 0; cy--)

  {

    int32\_t dy2 = (r - cy) \* (r - cy);

    for (cx = xs; cx < r; cx++)

    {

      int32\_t hyp2 = (r - cx) \* (r - cx) + dy2;

      if (hyp2 <= r1) break;

      if (hyp2 >= r2) continue;

      float alphaf = (float)r - sqrtf(hyp2);

      if (alphaf > HiAlphaTheshold) break;

      xs = cx;

      if (alphaf < LoAlphaTheshold) continue;

      uint8\_t alpha = alphaf \* 255;

      if (bg\_color == 0x00FFFFFF) {

        drawPixel(x + cx - r, y + cy - r, color, alpha, bg\_color);

        drawPixel(x - cx + r, y + cy - r, color, alpha, bg\_color);

        drawPixel(x - cx + r, y - cy + r, color, alpha, bg\_color);

        drawPixel(x + cx - r, y - cy + r, color, alpha, bg\_color);

      }

      else {

        uint16\_t pcol = drawPixel(x + cx - r, y + cy - r, color, alpha, bg\_color);

        drawPixel(x - cx + r, y + cy - r, pcol);

        drawPixel(x - cx + r, y - cy + r, pcol);

        drawPixel(x + cx - r, y - cy + r, pcol);

      }

    }

    drawFastHLine(x + cx - r, y + cy - r, 2 \* (r - cx) + 1, color);

    drawFastHLine(x + cx - r, y - cy + r, 2 \* (r - cx) + 1, color);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillSmoothRoundRect

\*\* Description:             Draw a filled anti-aliased rounded corner rectangle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillSmoothRoundRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, int32\_t r, uint32\_t color, uint32\_t bg\_color)

{

  inTransaction = true;

  int32\_t xs = 0;

  int32\_t cx = 0;

  // Limit radius to half width or height

  if (r < 0) r = 0;

  if (r > w/2) r = w/2;

  if (r > h/2) r = h/2;

  y += r;

  h -= 2\*r;

  fillRect(x, y, w, h, color);

  h--;

  x += r;

  w -= 2\*r+1;

  int32\_t r1 = r \* r;

  r++;

  int32\_t r2 = r \* r;

  for (int32\_t cy = r - 1; cy > 0; cy--)

  {

    int32\_t dy2 = (r - cy) \* (r - cy);

    for (cx = xs; cx < r; cx++)

    {

      int32\_t hyp2 = (r - cx) \* (r - cx) + dy2;

      if (hyp2 <= r1) break;

      if (hyp2 >= r2) continue;

      float alphaf = (float)r - sqrtf(hyp2);

      if (alphaf > HiAlphaTheshold) break;

      xs = cx;

      if (alphaf < LoAlphaTheshold) continue;

      uint8\_t alpha = alphaf \* 255;

      drawPixel(x + cx - r, y + cy - r, color, alpha, bg\_color);

      drawPixel(x - cx + r + w, y + cy - r, color, alpha, bg\_color);

      drawPixel(x - cx + r + w, y - cy + r + h, color, alpha, bg\_color);

      drawPixel(x + cx - r, y - cy + r + h, color, alpha, bg\_color);

    }

    drawFastHLine(x + cx - r, y + cy - r, 2 \* (r - cx) + 1 + w, color);

    drawFastHLine(x + cx - r, y - cy + r + h, 2 \* (r - cx) + 1 + w, color);

  }

  inTransaction = lockTransaction;

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawSpot - maths intensive, so for small filled circles

\*\* Description:             Draw an anti-aliased filled circle at ax,ay with radius r

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawSpot(float ax, float ay, float r, uint32\_t fg\_color, uint32\_t bg\_color)

{

  // Filled circle can be created by the wide line function with zero line length

  drawWedgeLine( ax, ay, ax, ay, r, r, fg\_color, bg\_color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawWideLine - background colour specified or pixel read

\*\* Description:             draw an anti-aliased line with rounded ends, width wd

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawWideLine(float ax, float ay, float bx, float by, float wd, uint32\_t fg\_color, uint32\_t bg\_color)

{

  drawWedgeLine( ax, ay, bx, by, wd/2.0, wd/2.0, fg\_color, bg\_color);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawWedgeLine - background colour specified or pixel read

\*\* Description:             draw an anti-aliased line with different width radiused ends

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawWedgeLine(float ax, float ay, float bx, float by, float ar, float br, uint32\_t fg\_color, uint32\_t bg\_color)

{

  if ( (ar < 0.0) || (br < 0.0) )return;

  if ( (abs(ax - bx) < 0.01f) && (abs(ay - by) < 0.01f) ) bx += 0.01f;  // Avoid divide by zero

  // Find line bounding box

  int32\_t x0 = (int32\_t)floorf(fminf(ax-ar, bx-br));

  int32\_t x1 = (int32\_t) ceilf(fmaxf(ax+ar, bx+br));

  int32\_t y0 = (int32\_t)floorf(fminf(ay-ar, by-br));

  int32\_t y1 = (int32\_t) ceilf(fmaxf(ay+ar, by+br));

  if (!clipWindow(&x0, &y0, &x1, &y1)) return;

  // Establish x start and y start

  int32\_t ys = ay;

  if ((ax-ar)>(bx-br)) ys = by;

  float rdt = ar - br; // Radius delta

  float alpha = 1.0f;

  ar += 0.5;

  uint16\_t bg = bg\_color;

  float xpax, ypay, bax = bx - ax, bay = by - ay;

  begin\_nin\_write();

  inTransaction = true;

  int32\_t xs = x0;

  // Scan bounding box from ys down, calculate pixel intensity from distance to line

  for (int32\_t yp = ys; yp <= y1; yp++) {

    bool swin = true;  // Flag to start new window area

    bool endX = false; // Flag to skip pixels

    ypay = yp - ay;

    for (int32\_t xp = xs; xp <= x1; xp++) {

      if (endX) if (alpha <= LoAlphaTheshold) break;  // Skip right side

      xpax = xp - ax;

      alpha = ar - wedgeLineDistance(xpax, ypay, bax, bay, rdt);

      if (alpha <= LoAlphaTheshold ) continue;

      // Track edge to minimise calculations

      if (!endX) { endX = true; xs = xp; }

      if (alpha > HiAlphaTheshold) {

        if (swin) { setWindow(xp, yp, width()-1, yp); swin = false; }

        pushColor(fg\_color);

        continue;

      }

      //Blend color with background and plot

      if (bg\_color == 0x00FFFFFF) {

        bg = readPixel(xp, yp); swin = true;

      }

      if (swin) { setWindow(xp, yp, width()-1, yp); swin = false; }

      pushColor(alphaBlend((uint8\_t)(alpha \* PixelAlphaGain), fg\_color, bg));

    }

  }

  // Reset x start to left side of box

  xs = x0;

  // Scan bounding box from ys-1 up, calculate pixel intensity from distance to line

  for (int32\_t yp = ys-1; yp >= y0; yp--) {

    bool swin = true;  // Flag to start new window area

    bool endX = false; // Flag to skip pixels

    ypay = yp - ay;

    for (int32\_t xp = xs; xp <= x1; xp++) {

      if (endX) if (alpha <= LoAlphaTheshold) break;  // Skip right side of drawn line

      xpax = xp - ax;

      alpha = ar - wedgeLineDistance(xpax, ypay, bax, bay, rdt);

      if (alpha <= LoAlphaTheshold ) continue;

      // Track line boundary

      if (!endX) { endX = true; xs = xp; }

      if (alpha > HiAlphaTheshold) {

        if (swin) { setWindow(xp, yp, width()-1, yp); swin = false; }

        pushColor(fg\_color);

        continue;

      }

      //Blend color with background and plot

      if (bg\_color == 0x00FFFFFF) {

        bg = readPixel(xp, yp); swin = true;

      }

      if (swin) { setWindow(xp, yp, width()-1, yp); swin = false; }

      pushColor(alphaBlend((uint8\_t)(alpha \* PixelAlphaGain), fg\_color, bg));

    }

  }

  inTransaction = lockTransaction;

  end\_nin\_write();

}

// Calculate distance of px,py to closest part of line

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           lineDistance - private helper function for drawWedgeLine

\*\* Description:             returns distance of px,py to closest part of a to b wedge

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

inline float TFT\_eSPI::wedgeLineDistance(float xpax, float ypay, float bax, float bay, float dr)

{

  float h = fmaxf(fminf((xpax \* bax + ypay \* bay) / (bax \* bax + bay \* bay), 1.0f), 0.0f);

  float dx = xpax - bax \* h, dy = ypay - bay \* h;

  return sqrtf(dx \* dx + dy \* dy) + h \* dr;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawFastVLine

\*\* Description:             draw a vertical line

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawFastVLine(int32\_t x, int32\_t y, int32\_t h, uint32\_t color)

{

  if (\_vpOoB) return;

  x+= \_xDatum;

  y+= \_yDatum;

  // Clipping

  if ((x < \_vpX) || (x >= \_vpW) || (y >= \_vpH)) return;

  if (y < \_vpY) { h += y - \_vpY; y = \_vpY; }

  if ((y + h) > \_vpH) h = \_vpH - y;

  if (h < 1) return;

  begin\_tft\_write();

  setWindow(x, y, x, y + h - 1);

  pushBlock(color, h);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawFastHLine

\*\* Description:             draw a horizontal line

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::drawFastHLine(int32\_t x, int32\_t y, int32\_t w, uint32\_t color)

{

  if (\_vpOoB) return;

  x+= \_xDatum;

  y+= \_yDatum;

  // Clipping

  if ((y < \_vpY) || (x >= \_vpW) || (y >= \_vpH)) return;

  if (x < \_vpX) { w += x - \_vpX; x = \_vpX; }

  if ((x + w) > \_vpW) w = \_vpW - x;

  if (w < 1) return;

  begin\_tft\_write();

  setWindow(x, y, x + w - 1, y);

  pushBlock(color, w);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillRect

\*\* Description:             draw a filled rectangle

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillRect(int32\_t x, int32\_t y, int32\_t w, int32\_t h, uint32\_t color)

{

  if (\_vpOoB) return;

  x+= \_xDatum;

  y+= \_yDatum;

  // Clipping

  if ((x >= \_vpW) || (y >= \_vpH)) return;

  if (x < \_vpX) { w += x - \_vpX; x = \_vpX; }

  if (y < \_vpY) { h += y - \_vpY; y = \_vpY; }

  if ((x + w) > \_vpW) w = \_vpW - x;

  if ((y + h) > \_vpH) h = \_vpH - y;

  if ((w < 1) || (h < 1)) return;

  //Serial.print(" \_xDatum=");Serial.print( \_xDatum);Serial.print(", \_yDatum=");Serial.print( \_yDatum);

  //Serial.print(", \_xWidth=");Serial.print(\_xWidth);Serial.print(", \_yHeight=");Serial.println(\_yHeight);

  //Serial.print(" \_vpX=");Serial.print( \_vpX);Serial.print(", \_vpY=");Serial.print( \_vpY);

  //Serial.print(", \_vpW=");Serial.print(\_vpW);Serial.print(", \_vpH=");Serial.println(\_vpH);

  //Serial.print(" x=");Serial.print( y);Serial.print(", y=");Serial.print( y);

  //Serial.print(", w=");Serial.print(w);Serial.print(", h=");Serial.println(h);

  begin\_tft\_write();

  setWindow(x, y, x + w - 1, y + h - 1);

  pushBlock(color, w \* h);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillRectVGradient

\*\* Description:             draw a filled rectangle with a vertical colour gradient

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillRectVGradient(int16\_t x, int16\_t y, int16\_t w, int16\_t h, uint32\_t color1, uint32\_t color2)

{

  if (\_vpOoB) return;

  x+= \_xDatum;

  y+= \_yDatum;

  // Clipping

  if ((x >= \_vpW) || (y >= \_vpH)) return;

  if (x < \_vpX) { w += x - \_vpX; x = \_vpX; }

  if (y < \_vpY) { h += y - \_vpY; y = \_vpY; }

  if ((x + w) > \_vpW) w = \_vpW - x;

  if ((y + h) > \_vpH) h = \_vpH - y;

  if ((w < 1) || (h < 1)) return;

  begin\_nin\_write();

  float delta = -255.0/h;

  float alpha = 255.0;

  uint32\_t color = color1;

  while (h--) {

    drawFastHLine(x, y++, w, color);

    alpha += delta;

    color = alphaBlend((uint8\_t)alpha, color1, color2);

  }

  end\_nin\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           fillRectHGradient

\*\* Description:             draw a filled rectangle with a horizontal colour gradient

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::fillRectHGradient(int16\_t x, int16\_t y, int16\_t w, int16\_t h, uint32\_t color1, uint32\_t color2)

{

  if (\_vpOoB) return;

  x+= \_xDatum;

  y+= \_yDatum;

  // Clipping

  if ((x >= \_vpW) || (y >= \_vpH)) return;

  if (x < \_vpX) { w += x - \_vpX; x = \_vpX; }

  if (y < \_vpY) { h += y - \_vpY; y = \_vpY; }

  if ((x + w) > \_vpW) w = \_vpW - x;

  if ((y + h) > \_vpH) h = \_vpH - y;

  if ((w < 1) || (h < 1)) return;

  begin\_nin\_write();

  float delta = -255.0/w;

  float alpha = 255.0;

  uint32\_t color = color1;

  while (w--) {

    drawFastVLine(x++, y, h, color);

    alpha += delta;

    color = alphaBlend((uint8\_t)alpha, color1, color2);

  }

  end\_nin\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           color565

\*\* Description:             convert three 8 bit RGB levels to a 16 bit colour value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t TFT\_eSPI::color565(uint8\_t r, uint8\_t g, uint8\_t b)

{

  return ((r & 0xF8) << 8) | ((g & 0xFC) << 3) | (b >> 3);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           color16to8

\*\* Description:             convert 16 bit colour to an 8 bit 332 RGB colour value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t TFT\_eSPI::color16to8(uint16\_t c)

{

  return ((c & 0xE000)>>8) | ((c & 0x0700)>>6) | ((c & 0x0018)>>3);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           color8to16

\*\* Description:             convert 8 bit colour to a 16 bit 565 colour value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint16\_t TFT\_eSPI::color8to16(uint8\_t color)

{

  uint8\_t  blue[] = {0, 11, 21, 31}; // blue 2 to 5 bit colour lookup table

  uint16\_t color16 = 0;

  //        =====Green=====     ===============Red==============

  color16  = (color & 0x1C)<<6 | (color & 0xC0)<<5 | (color & 0xE0)<<8;

  //        =====Green=====    =======Blue======

  color16 |= (color & 0x1C)<<3 | blue[color & 0x03];

  return color16;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           color16to24

\*\* Description:             convert 16 bit colour to a 24 bit 888 colour value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t TFT\_eSPI::color16to24(uint16\_t color565)

{

  uint8\_t r = (color565 >> 8) & 0xF8; r |= (r >> 5);

  uint8\_t g = (color565 >> 3) & 0xFC; g |= (g >> 6);

  uint8\_t b = (color565 << 3) & 0xF8; b |= (b >> 5);

  return ((uint32\_t)r << 16) | ((uint32\_t)g << 8) | ((uint32\_t)b << 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           color24to16

\*\* Description:             convert 24 bit colour to a 16 bit 565 colour value

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint32\_t TFT\_eSPI::color24to16(uint32\_t color888)

{

  uint16\_t r = (color888 >> 8) & 0xF800;

  uint16\_t g = (color888 >> 5) & 0x07E0;

  uint16\_t b = (color888 >> 3) & 0x001F;

  return (r | g | b);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           invertDisplay

\*\* Description:             invert the display colours i = 1 invert, i = 0 normal

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::invertDisplay(bool i)

{

  begin\_tft\_write();

  // Send the command twice as otherwise it does not always work!

  writecommand(i ? TFT\_INVON : TFT\_INVOFF);

  writecommand(i ? TFT\_INVON : TFT\_INVOFF);

  end\_tft\_write();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setAttribute

\*\* Description:             Sets a control parameter of an attribute

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setAttribute(uint8\_t attr\_id, uint8\_t param) {

    switch (attr\_id) {

            break;

        case CP437\_SWITCH:

            \_cp437 = param;

            break;

        case UTF8\_SWITCH:

            \_utf8  = param;

            decoderState = 0;

            break;

        case PSRAM\_ENABLE:

#if defined (ESP32) && defined (CONFIG\_SPIRAM\_SUPPORT)

            if (psramFound()) \_psram\_enable = param; // Enable the use of PSRAM (if available)

            else

#endif

            \_psram\_enable = false;

            break;

        //case 4: // TBD future feature control

        //    \_tbd = param;

        //    break;

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getAttribute

\*\* Description:             Get value of an attribute (control parameter)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

uint8\_t TFT\_eSPI::getAttribute(uint8\_t attr\_id) {

    switch (attr\_id) {

        case CP437\_SWITCH: // ON/OFF control of full CP437 character set

            return \_cp437;

        case UTF8\_SWITCH: // ON/OFF control of UTF-8 decoding

            return \_utf8;

        case PSRAM\_ENABLE:

            return \_psram\_enable;

        //case 3: // TBD future feature control

        //    return \_tbd;

        //    break;

    }

    return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           decodeUTF8

\*\* Description:             Serial UTF-8 decoder with fall-back to extended ASCII

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*x\*/

uint16\_t TFT\_eSPI::decodeUTF8(uint8\_t c)

{

  if (!\_utf8) return c;

  // 7 bit Unicode Code Point

  if ((c & 0x80) == 0x00) {

    decoderState = 0;

    return c;

  }

  if (decoderState == 0) {

    // 11 bit Unicode Code Point

    if ((c & 0xE0) == 0xC0) {

      decoderBuffer = ((c & 0x1F)<<6);

      decoderState = 1;

      return 0;

    }

    // 16 bit Unicode Code Point

    if ((c & 0xF0) == 0xE0) {

      decoderBuffer = ((c & 0x0F)<<12);

      decoderState = 2;

      return 0;

    }

    // 21 bit Unicode  Code Point not supported so fall-back to extended ASCII

    // if ((c & 0xF8) == 0xF0) return c;

  }

  else {

    if (decoderState == 2) {

      decoderBuffer |= ((c & 0x3F)<<6);

      decoderState--;

      return 0;

    }

    else {

      decoderBuffer |= (c & 0x3F);

      decoderState = 0;

      return decoderBuffer;

    }

  }

  decoderState = 0;

  return c; // fall-back to extended ASCII

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           decodeUTF8

\*\* Description:             Line buffer UTF-8 decoder with fall-back to extended ASCII

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*x\*/

uint16\_t TFT\_eSPI::decodeUTF8(uint8\_t \*buf, uint16\_t \*index, uint16\_t remaining)

{

  uint16\_t c = buf[(\*index)++];

  //Serial.print("Byte from string = 0x"); Serial.println(c, HEX);

  if (!\_utf8) return c;

  // 7 bit Unicode

  if ((c & 0x80) == 0x00) return c;

  // 11 bit Unicode

  if (((c & 0xE0) == 0xC0) && (remaining > 1))

    return ((c & 0x1F)<<6) | (buf[(\*index)++]&0x3F);

  // 16 bit Unicode

  if (((c & 0xF0) == 0xE0) && (remaining > 2)) {

    c = ((c & 0x0F)<<12) | ((buf[(\*index)++]&0x3F)<<6);

    return  c | ((buf[(\*index)++]&0x3F));

  }

  // 21 bit Unicode not supported so fall-back to extended ASCII

  // if ((c & 0xF8) == 0xF0) return c;

  return c; // fall-back to extended ASCII

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           alphaBlend

\*\* Description:             Blend 16bit foreground and background

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*x\*/

uint16\_t TFT\_eSPI::alphaBlend(uint8\_t alpha, uint16\_t fgc, uint16\_t bgc)

{

  // For speed use fixed point maths and rounding to permit a power of 2 division

  uint16\_t fgR = ((fgc >> 10) & 0x3E) + 1;

  uint16\_t fgG = ((fgc >>  4) & 0x7E) + 1;

  uint16\_t fgB = ((fgc <<  1) & 0x3E) + 1;

  uint16\_t bgR = ((bgc >> 10) & 0x3E) + 1;

  uint16\_t bgG = ((bgc >>  4) & 0x7E) + 1;

  uint16\_t bgB = ((bgc <<  1) & 0x3E) + 1;

  // Shift right 1 to drop rounding bit and shift right 8 to divide by 256

  uint16\_t r = (((fgR \* alpha) + (bgR \* (255 - alpha))) >> 9);

  uint16\_t g = (((fgG \* alpha) + (bgG \* (255 - alpha))) >> 9);

  uint16\_t b = (((fgB \* alpha) + (bgB \* (255 - alpha))) >> 9);

  // Combine RGB565 colours into 16 bits

  //return ((r&0x18) << 11) | ((g&0x30) << 5) | ((b&0x18) << 0); // 2 bit greyscale

  //return ((r&0x1E) << 11) | ((g&0x3C) << 5) | ((b&0x1E) << 0); // 4 bit greyscale

  return (r << 11) | (g << 5) | (b << 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           alphaBlend

\*\* Description:             Blend 16bit foreground and background with dither

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*x\*/

uint16\_t TFT\_eSPI::alphaBlend(uint8\_t alpha, uint16\_t fgc, uint16\_t bgc, uint8\_t dither)

{

  if (dither) {

    int16\_t alphaDither = (int16\_t)alpha - dither + random(2\*dither+1); // +/-4 randomised

    alpha = (uint8\_t)alphaDither;

    if (alphaDither <  0) alpha = 0;

    if (alphaDither >255) alpha = 255;

  }

  return alphaBlend(alpha, fgc, bgc);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           alphaBlend

\*\* Description:             Blend 24bit foreground and background with optional dither

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*x\*/

uint32\_t TFT\_eSPI::alphaBlend24(uint8\_t alpha, uint32\_t fgc, uint32\_t bgc, uint8\_t dither)

{

  if (dither) {

    int16\_t alphaDither = (int16\_t)alpha - dither + random(2\*dither+1); // +/-dither randomised

    alpha = (uint8\_t)alphaDither;

    if (alphaDither <  0) alpha = 0;

    if (alphaDither >255) alpha = 255;

  }

  // For speed use fixed point maths and rounding to permit a power of 2 division

  uint16\_t fgR = ((fgc >> 15) & 0x1FE) + 1;

  uint16\_t fgG = ((fgc >>  7) & 0x1FE) + 1;

  uint16\_t fgB = ((fgc <<  1) & 0x1FE) + 1;

  uint16\_t bgR = ((bgc >> 15) & 0x1FE) + 1;

  uint16\_t bgG = ((bgc >>  7) & 0x1FE) + 1;

  uint16\_t bgB = ((bgc <<  1) & 0x1FE) + 1;

  // Shift right 1 to drop rounding bit and shift right 8 to divide by 256

  uint16\_t r = (((fgR \* alpha) + (bgR \* (255 - alpha))) >> 9);

  uint16\_t g = (((fgG \* alpha) + (bgG \* (255 - alpha))) >> 9);

  uint16\_t b = (((fgB \* alpha) + (bgB \* (255 - alpha))) >> 9);

  // Combine RGB colours into 24 bits

  return (r << 16) | (g << 8) | (b << 0);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           write

\*\* Description:             draw characters piped through serial stream

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* // Not all processors support buffered write

#ifndef ARDUINO\_ARCH\_ESP8266 // Avoid ESP8266 board package bug

size\_t TFT\_eSPI::write(const uint8\_t \*buf, size\_t len)

{

  inTransaction = true;

  uint8\_t \*lbuf = (uint8\_t \*)buf;

  while(\*lbuf !=0 && len--) write(\*lbuf++);

  inTransaction = lockTransaction;

  end\_tft\_write();

  return 1;

}

#endif

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           write

\*\* Description:             draw characters piped through serial stream

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

size\_t TFT\_eSPI::write(uint8\_t utf8)

{

  if (\_vpOoB) return 1;

  uint16\_t uniCode = decodeUTF8(utf8);

  if (!uniCode) return 1;

  if (utf8 == '\r') return 1;

#ifdef SMOOTH\_FONT

  if(fontLoaded) {

    if (uniCode < 32 && utf8 != '\n') return 1;

    drawGlyph(uniCode);

    return 1;

  }

#endif

  if (uniCode == '\n') uniCode+=22; // Make it a valid space character to stop errors

  else if (uniCode < 32) return 1;

  uint16\_t cwidth = 0;

  uint16\_t cheight = 0;

//vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv DEBUG vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

  //Serial.print((uint8\_t) uniCode); // Debug line sends all printed TFT text to serial port

  //Serial.println(uniCode, HEX); // Debug line sends all printed TFT text to serial port

  //delay(5);                     // Debug optional wait for serial port to flush through

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ DEBUG ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#ifdef LOAD\_GFXFF

  if(!gfxFont) {

#endif

//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#ifdef LOAD\_FONT2

  if (textfont == 2) {

    if (uniCode > 127) return 1;

    cwidth = pgm\_read\_byte(widtbl\_f16 + uniCode-32);

    cheight = chr\_hgt\_f16;

    // Font 2 is rendered in whole byte widths so we must allow for this

    cwidth = (cwidth + 6) / 8;  // Width in whole bytes for font 2, should be + 7 but must allow for font width change

    cwidth = cwidth \* 8;        // Width converted back to pixels

  }

  #ifdef LOAD\_RLE

  else

  #endif

#endif

#ifdef LOAD\_RLE

  {

    if ((textfont>2) && (textfont<9)) {

      if (uniCode > 127) return 1;

      // Uses the fontinfo struct array to avoid lots of 'if' or 'switch' statements

      cwidth = pgm\_read\_byte( (uint8\_t \*)pgm\_read\_dword( &(fontdata[textfont].widthtbl ) ) + uniCode-32 );

      cheight= pgm\_read\_byte( &fontdata[textfont].height );

    }

  }

#endif

#ifdef LOAD\_GLCD

  if (textfont==1) {

      cwidth =  6;

      cheight = 8;

  }

#else

  if (textfont==1) return 1;

#endif

  cheight = cheight \* textsize;

  if (utf8 == '\n') {

    cursor\_y += cheight;

    cursor\_x  = 0;

  }

  else {

    if (textwrapX && (cursor\_x + cwidth \* textsize > width())) {

      cursor\_y += cheight;

      cursor\_x = 0;

    }

    if (textwrapY && (cursor\_y >= (int32\_t) height())) cursor\_y = 0;

    cursor\_x += drawChar(uniCode, cursor\_x, cursor\_y, textfont);

  }

//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

#ifdef LOAD\_GFXFF

  } // Custom GFX font

  else {

    if(utf8 == '\n') {

      cursor\_x  = 0;

      cursor\_y += (int16\_t)textsize \* (uint8\_t)pgm\_read\_byte(&gfxFont->yAdvance);

    } else {

      if (uniCode > pgm\_read\_word(&gfxFont->last )) return 1;

      if (uniCode < pgm\_read\_word(&gfxFont->first)) return 1;

      uint16\_t   c2    = uniCode - pgm\_read\_word(&gfxFont->first);

      GFXglyph \*glyph = &(((GFXglyph \*)pgm\_read\_dword(&gfxFont->glyph))[c2]);

      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);

        if(textwrapX && ((cursor\_x + textsize \* (xo + w)) > width())) {

          // Drawing character would go off right edge; wrap to new line

          cursor\_x  = 0;

          cursor\_y += (int16\_t)textsize \* (uint8\_t)pgm\_read\_byte(&gfxFont->yAdvance);

        }

        if (textwrapY && (cursor\_y >= (int32\_t) height())) cursor\_y = 0;

        drawChar(cursor\_x, cursor\_y, uniCode, textcolor, textbgcolor, textsize);

      }

      cursor\_x += pgm\_read\_byte(&glyph->xAdvance) \* (int16\_t)textsize;

    }

  }

#endif // LOAD\_GFXFF

//<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

  return 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawChar

\*\* Description:             draw a Unicode glyph onto the screen

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

  // TODO: Rationalise with TFT\_eSprite

  // Any UTF-8 decoding must be done before calling drawChar()

int16\_t TFT\_eSPI::drawChar(uint16\_t uniCode, int32\_t x, int32\_t y)

{

  return drawChar(uniCode, x, y, textfont);

}

  // Any UTF-8 decoding must be done before calling drawChar()

int16\_t TFT\_eSPI::drawChar(uint16\_t uniCode, int32\_t x, int32\_t y, uint8\_t font)

{

  if (\_vpOoB || !uniCode) return 0;

  if (font==1) {

#ifdef LOAD\_GLCD

  #ifndef LOAD\_GFXFF

    drawChar(x, y, uniCode, textcolor, textbgcolor, textsize);

    return 6 \* textsize;

  #endif

#else

  #ifndef LOAD\_GFXFF

    return 0;

  #endif

#endif

#ifdef LOAD\_GFXFF

    drawChar(x, y, uniCode, textcolor, textbgcolor, textsize);

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

    #ifdef LOAD\_GLCD

      return 6 \* textsize;

    #else

      return 0;

    #endif

    }

    else {

      if((uniCode >= pgm\_read\_word(&gfxFont->first)) && (uniCode <= pgm\_read\_word(&gfxFont->last) )) {

        uint16\_t   c2    = uniCode - pgm\_read\_word(&gfxFont->first);

        GFXglyph \*glyph = &(((GFXglyph \*)pgm\_read\_dword(&gfxFont->glyph))[c2]);

        return pgm\_read\_byte(&glyph->xAdvance) \* textsize;

      }

      else {

        return 0;

      }

    }

#endif

  }

  if ((font>1) && (font<9) && ((uniCode < 32) || (uniCode > 127))) return 0;

  int32\_t width  = 0;

  int32\_t height = 0;

  uint32\_t flash\_address = 0;

  uniCode -= 32;

#ifdef LOAD\_FONT2

  if (font == 2) {

    flash\_address = pgm\_read\_dword(&chrtbl\_f16[uniCode]);

    width = pgm\_read\_byte(widtbl\_f16 + uniCode);

    height = chr\_hgt\_f16;

  }

  #ifdef LOAD\_RLE

  else

  #endif

#endif

#ifdef LOAD\_RLE

  {

    if ((font>2) && (font<9)) {

      flash\_address = pgm\_read\_dword( (const void\*)(pgm\_read\_dword( &(fontdata[font].chartbl ) ) + uniCode\*sizeof(void \*)) );

      width = pgm\_read\_byte( (uint8\_t \*)pgm\_read\_dword( &(fontdata[font].widthtbl ) ) + uniCode );

      height= pgm\_read\_byte( &fontdata[font].height );

    }

  }

#endif

  int32\_t xd = x + \_xDatum;

  int32\_t yd = y + \_yDatum;

  if ((xd + width \* textsize < \_vpX || xd >= \_vpW) && (yd + height \* textsize < \_vpY || yd >= \_vpH)) return width \* textsize ;

  int32\_t w = width;

  int32\_t pX      = 0;

  int32\_t pY      = y;

  uint8\_t line = 0;

  bool clip = xd < \_vpX || xd + width  \* textsize >= \_vpW || yd < \_vpY || yd + height \* textsize >= \_vpH;

#ifdef LOAD\_FONT2 // chop out code if we do not need it

  if (font == 2) {

    w = w + 6; // Should be + 7 but we need to compensate for width increment

    w = w / 8;

    if (textcolor == textbgcolor || textsize != 1 || clip) {

      //begin\_tft\_write();          // Sprite class can use this function, avoiding begin\_tft\_write()

      inTransaction = true;

      for (int32\_t i = 0; i < height; i++) {

        if (textcolor != textbgcolor) fillRect(x, pY, width \* textsize, textsize, textbgcolor);

        for (int32\_t k = 0; k < w; k++) {

          line = pgm\_read\_byte((uint8\_t \*)flash\_address + w \* i + k);

          if (line) {

            if (textsize == 1) {

              pX = x + k \* 8;

              if (line & 0x80) drawPixel(pX, pY, textcolor);

              if (line & 0x40) drawPixel(pX + 1, pY, textcolor);

              if (line & 0x20) drawPixel(pX + 2, pY, textcolor);

              if (line & 0x10) drawPixel(pX + 3, pY, textcolor);

              if (line & 0x08) drawPixel(pX + 4, pY, textcolor);

              if (line & 0x04) drawPixel(pX + 5, pY, textcolor);

              if (line & 0x02) drawPixel(pX + 6, pY, textcolor);

              if (line & 0x01) drawPixel(pX + 7, pY, textcolor);

            }

            else {

              pX = x + k \* 8 \* textsize;

              if (line & 0x80) fillRect(pX, pY, textsize, textsize, textcolor);

              if (line & 0x40) fillRect(pX + textsize, pY, textsize, textsize, textcolor);

              if (line & 0x20) fillRect(pX + 2 \* textsize, pY, textsize, textsize, textcolor);

              if (line & 0x10) fillRect(pX + 3 \* textsize, pY, textsize, textsize, textcolor);

              if (line & 0x08) fillRect(pX + 4 \* textsize, pY, textsize, textsize, textcolor);

              if (line & 0x04) fillRect(pX + 5 \* textsize, pY, textsize, textsize, textcolor);

              if (line & 0x02) fillRect(pX + 6 \* textsize, pY, textsize, textsize, textcolor);

              if (line & 0x01) fillRect(pX + 7 \* textsize, pY, textsize, textsize, textcolor);

            }

          }

        }

        pY += textsize;

      }

      inTransaction = lockTransaction;

      end\_tft\_write();

    }

    else { // Faster drawing of characters and background using block write

      begin\_tft\_write();

      setWindow(xd, yd, xd + width - 1, yd + height - 1);

      uint8\_t mask;

      for (int32\_t i = 0; i < height; i++) {

        pX = width;

        for (int32\_t k = 0; k < w; k++) {

          line = pgm\_read\_byte((uint8\_t \*) (flash\_address + w \* i + k) );

          mask = 0x80;

          while (mask && pX) {

            if (line & mask) {tft\_Write\_16(textcolor);}

            else {tft\_Write\_16(textbgcolor);}

            pX--;

            mask = mask >> 1;

          }

        }

        if (pX) {tft\_Write\_16(textbgcolor);}

      }

      end\_tft\_write();

    }

  }

  #ifdef LOAD\_RLE

  else

  #endif

#endif  //FONT2

#ifdef LOAD\_RLE  //674 bytes of code

  // Font is not 2 and hence is RLE encoded

  {

    begin\_tft\_write();

    inTransaction = true;

    w \*= height; // Now w is total number of pixels in the character

    if (textcolor == textbgcolor && !clip) {

      int32\_t px = 0, py = pY; // To hold character block start and end column and row values

      int32\_t pc = 0; // Pixel count

      uint8\_t np = textsize \* textsize; // Number of pixels in a drawn pixel

      uint8\_t tnp = 0; // Temporary copy of np for while loop

      uint8\_t ts = textsize - 1; // Temporary copy of textsize

      // 16 bit pixel count so maximum font size is equivalent to 180x180 pixels in area

      // w is total number of pixels to plot to fill character block

      while (pc < w) {

        line = pgm\_read\_byte((uint8\_t \*)flash\_address);

        flash\_address++;

        if (line & 0x80) {

          line &= 0x7F;

          line++;

          if (ts) {

            px = xd + textsize \* (pc % width); // Keep these px and py calculations outside the loop as they are slow

            py = yd + textsize \* (pc / width);

          }

          else {

            px = xd + pc % width; // Keep these px and py calculations outside the loop as they are slow

            py = yd + pc / width;

          }

          while (line--) { // In this case the while(line--) is faster

            pc++; // This is faster than putting pc+=line before while()?

            setWindow(px, py, px + ts, py + ts);

            if (ts) {

              tnp = np;

              while (tnp--) {tft\_Write\_16(textcolor);}

            }

            else {tft\_Write\_16(textcolor);}

            px += textsize;

            if (px >= (xd + width \* textsize)) {

              px = xd;

              py += textsize;

            }

          }

        }

        else {

          line++;

          pc += line;

        }

      }

    }

    else {

      // Text colour != background and textsize = 1 and character is within viewport area

      // so use faster drawing of characters and background using block write

      if (textcolor != textbgcolor && textsize == 1 && !clip)

      {

        setWindow(xd, yd, xd + width - 1, yd + height - 1);

        // Maximum font size is equivalent to 180x180 pixels in area

        while (w > 0) {

          line = pgm\_read\_byte((uint8\_t \*)flash\_address++); // 8 bytes smaller when incrementing here

          if (line & 0x80) {

            line &= 0x7F;

            line++; w -= line;

            pushBlock(textcolor,line);

          }

          else {

            line++; w -= line;

            pushBlock(textbgcolor,line);

          }

        }

      }

      else

      {

        int32\_t px = 0, py = 0;  // To hold character pixel coords

        int32\_t tx = 0, ty = 0;  // To hold character TFT pixel coords

        int32\_t pc = 0;          // Pixel count

        int32\_t pl = 0;          // Pixel line length

        uint16\_t pcol = 0;       // Pixel color

        bool     pf = true;      // Flag for plotting

        while (pc < w) {

          line = pgm\_read\_byte((uint8\_t \*)flash\_address);

          flash\_address++;

          if (line & 0x80) { pcol = textcolor; line &= 0x7F; pf = true;}

          else { pcol = textbgcolor; if (textcolor == textbgcolor) pf = false;}

          line++;

          px = pc % width;

          tx = x + textsize \* px;

          py = pc / width;

          ty = y + textsize \* py;

          pl = 0;

          pc += line;

          while (line--) {

            pl++;

            if ((px+pl) >= width) {

              if (pf) fillRect(tx, ty, pl \* textsize, textsize, pcol);

              pl = 0;

              px = 0;

              tx = x;

              py ++;

              ty += textsize;

            }

          }

          if (pl && pf) fillRect(tx, ty, pl \* textsize, textsize, pcol);

        }

      }

    }

    inTransaction = lockTransaction;

    end\_tft\_write();

  }

  // End of RLE font rendering

#endif

#if !defined (LOAD\_FONT2) && !defined (LOAD\_RLE)

  // Stop warnings

  flash\_address = flash\_address;

  w = w;

  pX = pX;

  pY = pY;

  line = line;

  clip = clip;

#endif

  return width \* textsize;    // x +

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawString (with or without user defined font)

\*\* Description :            draw string with padding if it is defined

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Without font number, uses font set by setTextFont()

int16\_t TFT\_eSPI::drawString(const String& string, int32\_t poX, int32\_t poY)

{

  int16\_t len = string.length() + 2;

  char buffer[len];

  string.toCharArray(buffer, len);

  return drawString(buffer, poX, poY, textfont);

}

// With font number

int16\_t TFT\_eSPI::drawString(const String& string, int32\_t poX, int32\_t poY, uint8\_t font)

{

  int16\_t len = string.length() + 2;

  char buffer[len];

  string.toCharArray(buffer, len);

  return drawString(buffer, poX, poY, font);

}

// Without font number, uses font set by setTextFont()

int16\_t TFT\_eSPI::drawString(const char \*string, int32\_t poX, int32\_t poY)

{

  return drawString(string, poX, poY, textfont);

}

// With font number. Note: font number is over-ridden if a smooth font is loaded

int16\_t TFT\_eSPI::drawString(const char \*string, int32\_t poX, int32\_t poY, uint8\_t font)

{

  int16\_t sumX = 0;

  uint8\_t padding = 1, baseline = 0;

  uint16\_t cwidth = textWidth(string, font); // Find the pixel width of the string in the font

  uint16\_t cheight = 8 \* textsize;

#ifdef LOAD\_GFXFF

  #ifdef SMOOTH\_FONT

    bool freeFont = (font == 1 && gfxFont && !fontLoaded);

  #else

    bool freeFont = (font == 1 && gfxFont);

  #endif

  if (freeFont) {

    cheight = glyph\_ab \* textsize;

    poY += cheight; // Adjust for baseline datum of free fonts

    baseline = cheight;

    padding =101; // Different padding method used for Free Fonts

    // We need to make an adjustment for the bottom of the string (eg 'y' character)

    if ((textdatum == BL\_DATUM) || (textdatum == BC\_DATUM) || (textdatum == BR\_DATUM)) {

      cheight += glyph\_bb \* textsize;

    }

  }

#endif

  // If it is not font 1 (GLCD or free font) get the baseline and pixel height of the font

#ifdef SMOOTH\_FONT

  if(fontLoaded) {

    baseline = gFont.maxAscent;

    cheight  = fontHeight();

  }

  else

#endif

  if (font!=1) {

    baseline = pgm\_read\_byte( &fontdata[font].baseline ) \* textsize;

    cheight = fontHeight(font);

  }

  if (textdatum || padX) {

    switch(textdatum) {

      case TC\_DATUM:

        poX -= cwidth/2;

        padding += 1;

        break;

      case TR\_DATUM:

        poX -= cwidth;

        padding += 2;

        break;

      case ML\_DATUM:

        poY -= cheight/2;

        //padding += 0;

        break;

      case MC\_DATUM:

        poX -= cwidth/2;

        poY -= cheight/2;

        padding += 1;

        break;

      case MR\_DATUM:

        poX -= cwidth;

        poY -= cheight/2;

        padding += 2;

        break;

      case BL\_DATUM:

        poY -= cheight;

        //padding += 0;

        break;

      case BC\_DATUM:

        poX -= cwidth/2;

        poY -= cheight;

        padding += 1;

        break;

      case BR\_DATUM:

        poX -= cwidth;

        poY -= cheight;

        padding += 2;

        break;

      case L\_BASELINE:

        poY -= baseline;

        //padding += 0;

        break;

      case C\_BASELINE:

        poX -= cwidth/2;

        poY -= baseline;

        padding += 1;

        break;

      case R\_BASELINE:

        poX -= cwidth;

        poY -= baseline;

        padding += 2;

        break;

    }

  }

  int8\_t xo = 0;

#ifdef LOAD\_GFXFF

  if (freeFont && (textcolor!=textbgcolor)) {

      cheight = (glyph\_ab + glyph\_bb) \* textsize;

      // Get the offset for the first character only to allow for negative offsets

      uint16\_t c2 = 0;

      uint16\_t len = strlen(string);

      uint16\_t n = 0;

      while (n < len && c2 == 0) c2 = decodeUTF8((uint8\_t\*)string, &n, len - n);

      if((c2 >= pgm\_read\_word(&gfxFont->first)) && (c2 <= pgm\_read\_word(&gfxFont->last) )) {

        c2 -= pgm\_read\_word(&gfxFont->first);

        GFXglyph \*glyph = &(((GFXglyph \*)pgm\_read\_dword(&gfxFont->glyph))[c2]);

        xo = pgm\_read\_byte(&glyph->xOffset) \* textsize;

        // Adjust for negative xOffset

        if (xo > 0) xo = 0;

        else cwidth -= xo;

        // Add 1 pixel of padding all round

        //cheight +=2;

        //fillRect(poX+xo-1, poY - 1 - glyph\_ab \* textsize, cwidth+2, cheight, textbgcolor);

        fillRect(poX+xo, poY - glyph\_ab \* textsize, cwidth, cheight, textbgcolor);

      }

      padding -=100;

    }

#endif

  uint16\_t len = strlen(string);

  uint16\_t n = 0;

#ifdef SMOOTH\_FONT

  if(fontLoaded) {

    setCursor(poX, poY);

    bool fillbg = \_fillbg;

    // If padding is requested then fill the text background

    if (padX && !\_fillbg) \_fillbg = true;

    while (n < len) {

      uint16\_t uniCode = decodeUTF8((uint8\_t\*)string, &n, len - n);

      drawGlyph(uniCode);

    }

    \_fillbg = fillbg; // restore state

    sumX += cwidth;

    //fontFile.close();

  }

  else

#endif

  {

    while (n < len) {

      uint16\_t uniCode = decodeUTF8((uint8\_t\*)string, &n, len - n);

      sumX += drawChar(uniCode, poX+sumX, poY, font);

    }

  }

//vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv DEBUG vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

// Switch on debugging for the padding areas

//#define PADDING\_DEBUG

#ifndef PADDING\_DEBUG

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ DEBUG ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  if((padX>cwidth) && (textcolor!=textbgcolor)) {

    int16\_t padXc = poX+cwidth+xo;

#ifdef LOAD\_GFXFF

    if (freeFont) {

      poX +=xo; // Adjust for negative offset start character

      poY -= glyph\_ab \* textsize;

      sumX += poX;

    }

#endif

    switch(padding) {

      case 1:

        fillRect(padXc,poY,padX-cwidth,cheight, textbgcolor);

        break;

      case 2:

        fillRect(padXc,poY,(padX-cwidth)>>1,cheight, textbgcolor);

        padXc = poX - ((padX-cwidth)>>1);

        fillRect(padXc,poY,(padX-cwidth)>>1,cheight, textbgcolor);

        break;

      case 3:

        if (padXc>padX) padXc = padX;

        fillRect(poX + cwidth - padXc,poY,padXc-cwidth,cheight, textbgcolor);

        break;

    }

  }

#else

//vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv DEBUG vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv

// This is debug code to show text (green box) and blanked (white box) areas

// It shows that the padding areas are being correctly sized and positioned

  if((padX>sumX) && (textcolor!=textbgcolor)) {

    int16\_t padXc = poX+sumX; // Maximum left side padding

#ifdef LOAD\_GFXFF

    if ((font == 1) && (gfxFont)) poY -= glyph\_ab;

#endif

    drawRect(poX,poY,sumX,cheight, TFT\_GREEN);

    switch(padding) {

      case 1:

        drawRect(padXc,poY,padX-sumX,cheight, TFT\_WHITE);

        break;

      case 2:

        drawRect(padXc,poY,(padX-sumX)>>1, cheight, TFT\_WHITE);

        padXc = (padX-sumX)>>1;

        drawRect(poX - padXc,poY,(padX-sumX)>>1,cheight, TFT\_WHITE);

        break;

      case 3:

        if (padXc>padX) padXc = padX;

        drawRect(poX + sumX - padXc,poY,padXc-sumX,cheight, TFT\_WHITE);

        break;

    }

  }

#endif

//^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ DEBUG ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

return sumX;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawCentreString (deprecated, use setTextDatum())

\*\* Descriptions:            draw string centred on dX

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::drawCentreString(const String& string, int32\_t dX, int32\_t poY, uint8\_t font)

{

  int16\_t len = string.length() + 2;

  char buffer[len];

  string.toCharArray(buffer, len);

  return drawCentreString(buffer, dX, poY, font);

}

int16\_t TFT\_eSPI::drawCentreString(const char \*string, int32\_t dX, int32\_t poY, uint8\_t font)

{

  uint8\_t tempdatum = textdatum;

  int32\_t sumX = 0;

  textdatum = TC\_DATUM;

  sumX = drawString(string, dX, poY, font);

  textdatum = tempdatum;

  return sumX;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawRightString (deprecated, use setTextDatum())

\*\* Descriptions:            draw string right justified to dX

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::drawRightString(const String& string, int32\_t dX, int32\_t poY, uint8\_t font)

{

  int16\_t len = string.length() + 2;

  char buffer[len];

  string.toCharArray(buffer, len);

  return drawRightString(buffer, dX, poY, font);

}

int16\_t TFT\_eSPI::drawRightString(const char \*string, int32\_t dX, int32\_t poY, uint8\_t font)

{

  uint8\_t tempdatum = textdatum;

  int16\_t sumX = 0;

  textdatum = TR\_DATUM;

  sumX = drawString(string, dX, poY, font);

  textdatum = tempdatum;

  return sumX;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawNumber

\*\* Description:             draw a long integer

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int16\_t TFT\_eSPI::drawNumber(long long\_num, int32\_t poX, int32\_t poY)

{

  isDigits = true; // Eliminate jiggle in monospaced fonts

  char str[12];

  ltoa(long\_num, str, 10);

  return drawString(str, poX, poY, textfont);

}

int16\_t TFT\_eSPI::drawNumber(long long\_num, int32\_t poX, int32\_t poY, uint8\_t font)

{

  isDigits = true; // Eliminate jiggle in monospaced fonts

  char str[12];

  ltoa(long\_num, str, 10);

  return drawString(str, poX, poY, font);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           drawFloat

\*\* Descriptions:            drawFloat, prints 7 non zero digits maximum

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Assemble and print a string, this permits alignment relative to a datum

// looks complicated but much more compact and actually faster than using print class

int16\_t TFT\_eSPI::drawFloat(float floatNumber, uint8\_t dp, int32\_t poX, int32\_t poY)

{

  return drawFloat(floatNumber, dp, poX, poY, textfont);

}

int16\_t TFT\_eSPI::drawFloat(float floatNumber, uint8\_t dp, int32\_t poX, int32\_t poY, uint8\_t font)

{

  isDigits = true;

  char str[14];               // Array to contain decimal string

  uint8\_t ptr = 0;            // Initialise pointer for array

  int8\_t  digits = 1;         // Count the digits to avoid array overflow

  float rounding = 0.5;       // Round up down delta

  bool negative = false;

  if (dp > 7) dp = 7; // Limit the size of decimal portion

  // Adjust the rounding value

  for (uint8\_t i = 0; i < dp; ++i) rounding /= 10.0;

  if (floatNumber < -rounding) {   // add sign, avoid adding - sign to 0.0!

    str[ptr++] = '-'; // Negative number

    str[ptr] = 0; // Put a null in the array as a precaution

    digits = 0;   // Set digits to 0 to compensate so pointer value can be used later

    floatNumber = -floatNumber; // Make positive

    negative = true;

  }

  floatNumber += rounding; // Round up or down

  if (dp == 0) {

    if (negative) floatNumber = -floatNumber;

    return drawNumber((long)floatNumber, poX, poY, font);

  }

  // For error put ... in string and return (all TFT\_eSPI library fonts contain . character)

  if (floatNumber >= 2147483647) {

    strcpy(str, "...");

    return drawString(str, poX, poY, font);

  }

  // No chance of overflow from here on

  // Get integer part

  uint32\_t temp = (uint32\_t)floatNumber;

  // Put integer part into array

  ltoa(temp, str + ptr, 10);

  // Find out where the null is to get the digit count loaded

  while ((uint8\_t)str[ptr] != 0) ptr++; // Move the pointer along

  digits += ptr;                  // Count the digits

  str[ptr++] = '.'; // Add decimal point

  str[ptr] = '0';   // Add a dummy zero

  str[ptr + 1] = 0; // Add a null but don't increment pointer so it can be overwritten

  // Get the decimal portion

  floatNumber = floatNumber - temp;

  // Get decimal digits one by one and put in array

  // Limit digit count so we don't get a false sense of resolution

  uint8\_t i = 0;

  while ((i < dp) && (digits < 9)) { // while (i < dp) for no limit but array size must be increased

    i++;

    floatNumber \*= 10;       // for the next decimal

    temp = floatNumber;      // get the decimal

    ltoa(temp, str + ptr, 10);

    ptr++; digits++;         // Increment pointer and digits count

    floatNumber -= temp;     // Remove that digit

  }

  // Finally we can plot the string and return pixel length

  return drawString(str, poX, poY, font);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setFreeFont

\*\* Descriptions:            Sets the GFX free font to use

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifdef LOAD\_GFXFF

void TFT\_eSPI::setFreeFont(const GFXfont \*f)

{

  if (f == nullptr) { // Fix issue #400 (ESP32 crash)

    setTextFont(1); // Use GLCD font

    return;

  }

  textfont = 1;

  gfxFont = (GFXfont \*)f;

  glyph\_ab = 0;

  glyph\_bb = 0;

  uint16\_t numChars = pgm\_read\_word(&gfxFont->last) - pgm\_read\_word(&gfxFont->first);

  // Find the biggest above and below baseline offsets

  for (uint16\_t c = 0; c < numChars; c++) {

    GFXglyph \*glyph1  = &(((GFXglyph \*)pgm\_read\_dword(&gfxFont->glyph))[c]);

    int8\_t ab = -pgm\_read\_byte(&glyph1->yOffset);

    if (ab > glyph\_ab) glyph\_ab = ab;

    int8\_t bb = pgm\_read\_byte(&glyph1->height) - ab;

    if (bb > glyph\_bb) glyph\_bb = bb;

  }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextFont

\*\* Description:             Set the font for the print stream

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextFont(uint8\_t f)

{

  textfont = (f > 0) ? f : 1; // Don't allow font 0

  gfxFont = NULL;

}

#else

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setFreeFont

\*\* Descriptions:            Sets the GFX free font to use

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Alternative to setTextFont() so we don't need two different named functions

void TFT\_eSPI::setFreeFont(uint8\_t font)

{

  setTextFont(font);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           setTextFont

\*\* Description:             Set the font for the print stream

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::setTextFont(uint8\_t f)

{

  textfont = (f > 0) ? f : 1; // Don't allow font 0

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getSPIinstance

\*\* Description:             Get the instance of the SPI class

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#if !defined (TFT\_PARALLEL\_8\_BIT) && ! defined (RP2040\_PIO\_INTERFACE)

SPIClass& TFT\_eSPI::getSPIinstance(void)

{

  return spi;

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           verifySetupID

\*\* Description:             Compare the ID if USER\_SETUP\_ID defined in user setup file

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

bool TFT\_eSPI::verifySetupID(uint32\_t id)

{

#if defined (USER\_SETUP\_ID)

  if (USER\_SETUP\_ID == id) return true;

#else

  id = id; // Avoid warning

#endif

  return false;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\* Function name:           getSetup

\*\* Description:             Get the setup details for diagnostic and sketch access

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void TFT\_eSPI::getSetup(setup\_t &tft\_settings)

{

// tft\_settings.version is set in header file

#if defined (USER\_SETUP\_INFO)

  tft\_settings.setup\_info = USER\_SETUP\_INFO;

#else

  tft\_settings.setup\_info = "NA";

#endif

#if defined (USER\_SETUP\_ID)

  tft\_settings.setup\_id = USER\_SETUP\_ID;

#else

  tft\_settings.setup\_id = 0;

#endif

#if defined (PROCESSOR\_ID)

  tft\_settings.esp = PROCESSOR\_ID;

#else

  tft\_settings.esp = -1;

#endif

#if defined (SUPPORT\_TRANSACTIONS)

  tft\_settings.trans = true;

#else

  tft\_settings.trans = false;

#endif

#if defined (TFT\_PARALLEL\_8\_BIT) || defined(TFT\_PARALLEL\_16\_BIT)

  tft\_settings.serial = false;

  tft\_settings.tft\_spi\_freq = 0;

#else

  tft\_settings.serial = true;

  tft\_settings.tft\_spi\_freq = SPI\_FREQUENCY/100000;

  #ifdef SPI\_READ\_FREQUENCY

    tft\_settings.tft\_rd\_freq = SPI\_READ\_FREQUENCY/100000;

  #endif

  #ifdef TFT\_SPI\_PORT

    tft\_settings.port = TFT\_SPI\_PORT;

  #else

    tft\_settings.port = 255;

  #endif

  #ifdef RP2040\_PIO\_SPI

    tft\_settings.interface = 0x10;

  #else

    tft\_settings.interface = 0x0;

  #endif

#endif

#if defined(TFT\_SPI\_OVERLAP)

  tft\_settings.overlap = true;

#else

  tft\_settings.overlap = false;

#endif

  tft\_settings.tft\_driver = TFT\_DRIVER;

  tft\_settings.tft\_width  = \_init\_width;

  tft\_settings.tft\_height = \_init\_height;

#ifdef CGRAM\_OFFSET

  tft\_settings.r0\_x\_offset = colstart;

  tft\_settings.r0\_y\_offset = rowstart;

  tft\_settings.r1\_x\_offset = 0;

  tft\_settings.r1\_y\_offset = 0;

  tft\_settings.r2\_x\_offset = 0;

  tft\_settings.r2\_y\_offset = 0;

  tft\_settings.r3\_x\_offset = 0;

  tft\_settings.r3\_y\_offset = 0;

#else

  tft\_settings.r0\_x\_offset = 0;

  tft\_settings.r0\_y\_offset = 0;

  tft\_settings.r1\_x\_offset = 0;

  tft\_settings.r1\_y\_offset = 0;

  tft\_settings.r2\_x\_offset = 0;

  tft\_settings.r2\_y\_offset = 0;

  tft\_settings.r3\_x\_offset = 0;

  tft\_settings.r3\_y\_offset = 0;

#endif

#if defined (TFT\_MOSI)

  tft\_settings.pin\_tft\_mosi = TFT\_MOSI;

#else

  tft\_settings.pin\_tft\_mosi = -1;

#endif

#if defined (TFT\_MISO)

  tft\_settings.pin\_tft\_miso = TFT\_MISO;

#else

  tft\_settings.pin\_tft\_miso = -1;

#endif

#if defined (TFT\_SCLK)

  tft\_settings.pin\_tft\_clk  = TFT\_SCLK;

#else

  tft\_settings.pin\_tft\_clk  = -1;

#endif

#if defined (TFT\_CS)

  tft\_settings.pin\_tft\_cs   = TFT\_CS;

#else

  tft\_settings.pin\_tft\_cs   = -1;

#endif

#if defined (TFT\_DC)

  tft\_settings.pin\_tft\_dc  = TFT\_DC;

#else

  tft\_settings.pin\_tft\_dc  = -1;

#endif

#if defined (TFT\_RD)

  tft\_settings.pin\_tft\_rd  = TFT\_RD;

#else

  tft\_settings.pin\_tft\_rd  = -1;

#endif

#if defined (TFT\_WR)

  tft\_settings.pin\_tft\_wr  = TFT\_WR;

#else

  tft\_settings.pin\_tft\_wr  = -1;

#endif

#if defined (TFT\_RST)

  tft\_settings.pin\_tft\_rst = TFT\_RST;

#else

  tft\_settings.pin\_tft\_rst = -1;

#endif

#if defined (TFT\_PARALLEL\_8\_BIT) || defined(TFT\_PARALLEL\_16\_BIT)

  tft\_settings.pin\_tft\_d0 = TFT\_D0;

  tft\_settings.pin\_tft\_d1 = TFT\_D1;

  tft\_settings.pin\_tft\_d2 = TFT\_D2;

  tft\_settings.pin\_tft\_d3 = TFT\_D3;

  tft\_settings.pin\_tft\_d4 = TFT\_D4;

  tft\_settings.pin\_tft\_d5 = TFT\_D5;

  tft\_settings.pin\_tft\_d6 = TFT\_D6;

  tft\_settings.pin\_tft\_d7 = TFT\_D7;

#else

  tft\_settings.pin\_tft\_d0 = -1;

  tft\_settings.pin\_tft\_d1 = -1;

  tft\_settings.pin\_tft\_d2 = -1;

  tft\_settings.pin\_tft\_d3 = -1;

  tft\_settings.pin\_tft\_d4 = -1;

  tft\_settings.pin\_tft\_d5 = -1;

  tft\_settings.pin\_tft\_d6 = -1;

  tft\_settings.pin\_tft\_d7 = -1;

#endif

#if defined (TFT\_BL)

  tft\_settings.pin\_tft\_led = TFT\_BL;

#endif

#if defined (TFT\_BACKLIGHT\_ON)

  tft\_settings.pin\_tft\_led\_on = TFT\_BACKLIGHT\_ON;

#endif

#if defined (TOUCH\_CS)

  tft\_settings.pin\_tch\_cs   = TOUCH\_CS;

  tft\_settings.tch\_spi\_freq = SPI\_TOUCH\_FREQUENCY/100000;

#else

  tft\_settings.pin\_tch\_cs   = -1;

  tft\_settings.tch\_spi\_freq = 0;

#endif

}

////////////////////////////////////////////////////////////////////////////////////////

#ifdef TOUCH\_CS

  #include "Extensions/Touch.cpp"

#endif

#include "Extensions/Button.cpp"

#include "Extensions/Sprite.cpp"

#ifdef SMOOTH\_FONT

  #include "Extensions/Smooth\_font.cpp"

#endif

#ifdef AA\_GRAPHICS

  #include "Extensions/AA\_graphics.cpp"  // Loaded if SMOOTH\_FONT is defined by user

#endif

////////////////////////////////////////////////////////////////////////////////////////