```
/*****************************************************
  Arduino TFT graphics library targeted at ESP8266
  and ESP32 based boards.

  This is a stand-alone library that contains the
  hardware driver, the graphics functions and the
  proportional fonts.

  The built-in fonts 4, 6, 7 and 8 are Run Length
  Encoded (RLE) to reduce the FLASH footprint.

  Last review/edit by Bodmer: 04/02/22
 *****************************************************/

// Stop fonts etc being loaded multiple times
#ifndef _TFT_eSPIH_
#define _TFT_eSPIH_

#define TFT_ESPI_VERSION "2.4.79"

// Bit level feature flags
// Bit 0 set: viewport capability
#define TFT_ESPI_FEATURES 1

/*****************************************************************************
**                          Section 1: Load required header files
*****************************************************************************/

//Standard support
#include <Arduino.h>
#include <Print.h>
#include <SPI.h>

/*****************************************************************************
**                          Section 2: Load library and processor specific header files
*****************************************************************************/
// Include header file that defines the fonts loaded, the TFT drivers
// available and the pins to be used, etc, etc
#ifdef CONFIG_TFT_eSPI_ESPIDF
  #include "TFT_config.h"
#endif

// New ESP8266 board package uses ARDUINO_ARCH_ESP8266
// old package defined ESP8266
#if defined (ESP8266)
  #ifndef ARDUINO_ARCH_ESP8266
    #define ARDUINO_ARCH_ESP8266
  #endif
#endif

// The following lines allow the user setup to be included in the sketch folder, see
// "Sketch_with_tft_setup" generic example.
#if !defined __has_include
  #if !defined(DISABLE_ALL_LIBRARY_WARNINGS)
    #warning Compiler does not support __has_include, so sketches cannot define the setup
  #endif
#else
  #if __has_include(<tft_setup.h>)
    // Include the sketch setup file
    #include <tft_setup.h>
    #ifndef USER_SETUP_LOADED
      // Prevent loading further setups
      #define USER_SETUP_LOADED
    #endif
  #endif
#endif

#include <User_Setup_Select.h>

// Handle FLASH based storage e.g. PROGMEM
#if defined(ARDUINO_ARCH_RP2040)
  #undef pgm_read_byte
  #define pgm_read_byte(addr)    (*(const unsigned char *)(addr))
  #undef pgm_read_word
  #define pgm_read_word(addr) ({ \
    typeof(addr) _addr = (addr); \
    *(const unsigned short *)(_addr); \
  })
  #undef pgm_read_dword
  #define pgm_read_dword(addr) ({ \
    typeof(addr) _addr = (addr); \
    *(const unsigned long *)(_addr); \
  })
#elif defined(__AVR__)
  #include <avr/pgmspace.h>
#elif defined(ARDUINO_ARCH_ESP8266) || defined(ESP32)
  #include <pgmspace.h>
#else
```

```
  #define PROGMEM
#endif

// Include the processor specific drivers
#if defined(CONFIG_IDF_TARGET_ESP32S3)
  #include "Processors/TFT_eSPI_ESP32_S3.h"
#elif defined(CONFIG_IDF_TARGET_ESP32C3)
  #include "Processors/TFT_eSPI_ESP32_C3.h"
#elif defined (ESP32)
  #include "Processors/TFT_eSPI_ESP32.h"
#elif defined (ARDUINO_ARCH_ESP8266)
  #include "Processors/TFT_eSPI_ESP8266.h"
#elif defined (STM32)
  #include "Processors/TFT_eSPI_STM32.h"
#elif defined(ARDUINO_ARCH_RP2040)
  #include "Processors/TFT_eSPI_RP2040.h"
#else
  #include "Processors/TFT_eSPI_Generic.h"
#endif

/*****************************************************************************
**                          Section 3: Interface setup
*****************************************************************************/
#ifndef TAB_COLOUR
  #define TAB_COLOUR 0
#endif

// If the SPI frequency is not defined, set a default
#ifndef SPI_FREQUENCY
  #define SPI_FREQUENCY  20000000
#endif

// If the SPI read frequency is not defined, set a default
#ifndef SPI_READ_FREQUENCY
  #define SPI_READ_FREQUENCY 10000000
#endif

// Some ST7789 boards do not work with Mode 0
#ifndef TFT_SPI_MODE
  #if defined(ST7789_DRIVER) || defined(ST7789_2_DRIVER)
    #define TFT_SPI_MODE SPI_MODE3
  #else
    #define TFT_SPI_MODE SPI_MODE0
  #endif
#endif

// If the XPT2046 SPI frequency is not defined, set a default
#ifndef SPI_TOUCH_FREQUENCY
  #define SPI_TOUCH_FREQUENCY  2500000
#endif

#ifndef SPI_BUSY_CHECK
  #define SPI_BUSY_CHECK
#endif

/*****************************************************************************
**                          Section 4: Setup fonts
*****************************************************************************/
// Use GLCD font in error case where user requests a smooth font file
// that does not exist (this is a temporary fix to stop ESP32 reboot)
#ifdef SMOOTH_FONT
  #ifndef LOAD_GLCD
    #define LOAD_GLCD
  #endif
#endif

// Only load the fonts defined in User_Setup.h (to save space)
// Set flag so RLE rendering code is optionally compiled
#ifdef LOAD_GLCD
  #include <Fonts/glcdfont.c>
#endif

#ifdef LOAD_FONT2
  #include <Fonts/Font16.h>
#endif

#ifdef LOAD_FONT4
  #include <Fonts/Font32rle.h>
  #define LOAD_RLE
#endif

#ifdef LOAD_FONT6
  #include <Fonts/Font64rle.h>
  #ifndef LOAD_RLE
    #define LOAD_RLE
  #endif
#endif
```

```
#ifdef LOAD_FONT7
  #include <Fonts/Font7srle.h>
  #ifndef LOAD_RLE
    #define LOAD_RLE
  #endif
#endif

#ifdef LOAD_FONT8
  #include <Fonts/Font72rle.h>
  #ifndef LOAD_RLE
    #define LOAD_RLE
  #endif
#elif defined LOAD_FONT8N // Optional narrower version
  #define LOAD_FONT8
  #include <Fonts/Font72x53rle.h>
  #ifndef LOAD_RLE
    #define LOAD_RLE
  #endif
#endif

#ifdef LOAD_GFXFF
  // We can include all the free fonts and they will only be built into
  // the sketch if they are used
  #include <Fonts/GFXFF/gfxfont.h>
  // Call up any user custom fonts
  #include <User_Setups/User_Custom_Fonts.h>
#endif // #ifdef LOAD_GFXFF

// Create a null default font in case some fonts not used (to prevent crash)
const  uint8_t widtbl_null[1] = {0};
PROGMEM const uint8_t chr_null[1] = {0};
PROGMEM const uint8_t* const chrtbl_null[1] = {chr_null};

// This is a structure to conveniently hold information on the default fonts
// Stores pointer to font character image address table, width table and height
typedef struct {
    const uint8_t *chartbl;
    const uint8_t *widthtbl;
    uint8_t height;
    uint8_t baseline;
    } fontinfo;

// Now fill the structure
const PROGMEM fontinfo fontdata [] = {
 #ifdef LOAD_GLCD
  { (const uint8_t *)font, widtbl_null, 0, 0 },
 #else
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },
 #endif
  // GLCD font (Font 1) does not have all parameters
  { (const uint8_t *)chrtbl_null, widtbl_null, 8, 7 },

 #ifdef LOAD_FONT2
  { (const uint8_t *)chrtbl_f16, widtbl_f16, chr_hgt_f16, baseline_f16},
 #else
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },
 #endif

  // Font 3 current unused
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },

 #ifdef LOAD_FONT4
  { (const uint8_t *)chrtbl_f32, widtbl_f32, chr_hgt_f32, baseline_f32},
 #else
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },
 #endif

  // Font 5 current unused
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },

 #ifdef LOAD_FONT6
  { (const uint8_t *)chrtbl_f64, widtbl_f64, chr_hgt_f64, baseline_f64},
 #else
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },
 #endif

 #ifdef LOAD_FONT7
  { (const uint8_t *)chrtbl_f7s, widtbl_f7s, chr_hgt_f7s, baseline_f7s},
 #else
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 },
 #endif

 #ifdef LOAD_FONT8
  { (const uint8_t *)chrtbl_f72, widtbl_f72, chr_hgt_f72, baseline_f72}
 #else
  { (const uint8_t *)chrtbl_null, widtbl_null, 0, 0 }
 #endif
};
```

```
/*****************************************************************************
**                          Section 5: Font datum enumeration
*****************************************************************************/
//These enumerate the text plotting alignment (reference datum point)
#define TL_DATUM 0 // Top left (default)
#define TC_DATUM 1 // Top centre
#define TR_DATUM 2 // Top right
#define ML_DATUM 3 // Middle left
#define CL_DATUM 3 // Centre left, same as above
#define MC_DATUM 4 // Middle centre
#define CC_DATUM 4 // Centre centre, same as above
#define MR_DATUM 5 // Middle right
#define CR_DATUM 5 // Centre right, same as above
#define BL_DATUM 6 // Bottom left
#define BC_DATUM 7 // Bottom centre
#define BR_DATUM 8 // Bottom right
#define L_BASELINE  9 // Left character baseline (Line the 'A' character would sit on)
#define C_BASELINE 10 // Centre character baseline
#define R_BASELINE 11 // Right character baseline

/*****************************************************************************
**                          Section 6: Colour enumeration
*****************************************************************************/
// Default color definitions
#define TFT_BLACK       0x0000      /*   0,   0,   0 */
#define TFT_NAVY        0x000F      /*   0,   0, 128 */
#define TFT_DARKGREEN   0x03E0      /*   0, 128,   0 */
#define TFT_DARKCYAN    0x03EF      /*   0, 128, 128 */
#define TFT_MAROON      0x7800      /* 128,   0,   0 */
#define TFT_PURPLE      0x780F      /* 128,   0, 128 */
#define TFT_OLIVE       0x7BE0      /* 128, 128,   0 */
#define TFT_LIGHTGREY   0xD69A      /* 211, 211, 211 */
#define TFT_DARKGREY    0x7BEF      /* 128, 128, 128 */
#define TFT_BLUE        0x001F      /*   0,   0, 255 */
#define TFT_GREEN       0x07E0      /*   0, 255,   0 */
#define TFT_CYAN        0x07FF      /*   0, 255, 255 */
#define TFT_RED         0xF800      /* 255,   0,   0 */
#define TFT_MAGENTA     0xF81F      /* 255,   0, 255 */
#define TFT_YELLOW      0xFFE0      /* 255, 255,   0 */
#define TFT_WHITE       0xFFFF      /* 255, 255, 255 */
#define TFT_ORANGE      0xFDA0      /* 255, 180,   0 */
#define TFT_GREENYELLOW 0xB7E0      /* 180, 255,   0 */
#define TFT_PINK        0xFE19      /* 255, 192, 203 */ //Lighter pink, was 0xFC9F
#define TFT_BROWN       0x9A60      /* 150,  75,   0 */
#define TFT_GOLD        0xFEA0      /* 255, 215,   0 */
#define TFT_SILVER      0xC618      /* 192, 192, 192 */
#define TFT_SKYBLUE     0x867D      /* 135, 206, 235 */
#define TFT_VIOLET      0x915C      /* 180,  46, 226 */

// Next is a special 16 bit colour value that encodes to 8 bits
// and will then decode back to the same 16 bit value.
// Convenient for 8 bit and 16 bit transparent sprites.
#define TFT_TRANSPARENT 0x0120 // This is actually a dark green

// Default palette for 4 bit colour sprites
static const uint16_t default_4bit_palette[] PROGMEM = {
  TFT_BLACK,    //  0  ^
  TFT_BROWN,    //  1  |
  TFT_RED,      //  2  |
  TFT_ORANGE,   //  3  |
  TFT_YELLOW,   //  4  Colours 0-9 follow the resistor colour code!
  TFT_GREEN,    //  5  |
  TFT_BLUE,     //  6  |
  TFT_PURPLE,   //  7  |
  TFT_DARKGREY, //  8  |
  TFT_WHITE,    //  9  v
  TFT_CYAN,     // 10  Blue+green mix
  TFT_MAGENTA,  // 11  Blue+red mix
  TFT_MAROON,   // 12  Darker red colour
  TFT_DARKGREEN,// 13  Darker green colour
  TFT_NAVY,     // 14  Darker blue colour
  TFT_PINK      // 15
};

/*****************************************************************************
**                          Section 7: Diagnostic support
*****************************************************************************/
// #define TFT_eSPI_DEBUG       // Switch on debug support serial messages   (not used yet)
// #define TFT_eSPI_FNx_DEBUG // Switch on debug support for function "x" (not used yet)

// This structure allows sketches to retrieve the user setup parameters at runtime
// by calling getSetup(), zero impact on code size unless used, mainly for diagnostics
typedef struct
{
String  version = TFT_ESPI_VERSION;
String  setup_info;  // Setup reference name available to use in a user setup
uint32_t setup_id;   // ID available to use in a user setup
```

```
  int32_t esp;            // Processor code
  uint8_t trans;          // SPI transaction support
  uint8_t serial;        // Serial (SPI) or parallel
  uint8_t   port;         // SPI port
  uint8_t overlap;       // ESP8266 overlap mode
  uint8_t interface;     // Interface type

  uint16_t tft_driver; // Hexadecimal code
  uint16_t tft_width;   // Rotation 0 width and height
  uint16_t tft_height;

  uint8_t r0_x_offset; // Display offsets, not all used yet
  uint8_t r0_y_offset;
  uint8_t r1_x_offset;
  uint8_t r1_y_offset;
  uint8_t r2_x_offset;
  uint8_t r2_y_offset;
  uint8_t r3_x_offset;
  uint8_t r3_y_offset;

  int8_t pin_tft_mosi; // SPI pins
  int8_t pin_tft_miso;
  int8_t pin_tft_clk;
  int8_t pin_tft_cs;

  int8_t pin_tft_dc;    // Control pins
  int8_t pin_tft_rd;
  int8_t pin_tft_wr;
  int8_t pin_tft_rst;

  int8_t pin_tft_d0;    // Parallel port pins
  int8_t pin_tft_d1;
  int8_t pin_tft_d2;
  int8_t pin_tft_d3;
  int8_t pin_tft_d4;
  int8_t pin_tft_d5;
  int8_t pin_tft_d6;
  int8_t pin_tft_d7;

  int8_t pin_tft_led;
  int8_t pin_tft_led_on;

  int8_t pin_tch_cs;    // Touch chip select pin

  int16_t tft_spi_freq;// TFT write SPI frequency
  int16_t tft_rd_freq; // TFT read   SPI frequency
  int16_t tch_spi_freq;// Touch controller read/write SPI frequency
} setup_t;

/***************************************************************************************
**                          Section 8: Class member and support functions
***************************************************************************************/
// Swap any type
template <typename T> static inline void
swap_coord(T& a, T& b) { T t = a; a = b; b = t; }

// Callback prototype for smooth font pixel colour read
typedef uint16_t (*getColorCallback)(uint16_t x, uint16_t y);

// Class functions and variables
class TFT_eSPI : public Print { friend class TFT_eSprite; // Sprite class has access to protected
members

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

  TFT_eSPI(int16_t _W = TFT_WIDTH, int16_t _H = TFT_HEIGHT);

  // init() and begin() are equivalent, begin() included for backwards compatibility
  // Sketch defined tab colour option is for ST7735 displays only
  void       init(uint8_t tc = TAB_COLOUR), begin(uint8_t tc = TAB_COLOUR);

  // These are virtual so the TFT_eSprite class can override them with sprite specific functions
  virtual void       drawPixel(int32_t x, int32_t y, uint32_t color),
                     drawChar(int32_t x, int32_t y, uint16_t c, uint32_t color, uint32_t bg, uint8_t size),
                     drawLine(int32_t xs, int32_t ys, int32_t xe, int32_t ye, uint32_t color),
                     drawFastVLine(int32_t x, int32_t y, int32_t h, uint32_t color),
                     drawFastHLine(int32_t x, int32_t y, int32_t w, uint32_t color),
                     fillRect(int32_t x, int32_t y, int32_t w, int32_t h, uint32_t color);

  virtual int16_t   drawChar(uint16_t uniCode, int32_t x, int32_t y, uint8_t font),
                     drawChar(uint16_t uniCode, int32_t x, int32_t y),
                     height(void),
                     width(void);

                     // Read the colour of a pixel at x,y and return value in 565 format
  virtual uint16_t readPixel(int32_t x, int32_t y);
```

```
  virtual void      setWindow(int32_t xs, int32_t ys, int32_t xe, int32_t ye);    // Note: start + end coordinates

                    // Push (aka write pixel) colours to the set window
  virtual void      pushColor(uint16_t color);

                    // These are non-inlined to enable override
  virtual void      begin_nin_write();
  virtual void      end_nin_write();

  void      setRotation(uint8_t r); // Set the display image orientation to 0, 1, 2 or 3
  uint8_t   getRotation(void);      // Read the current rotation

  void      invertDisplay(bool i);  // Tell TFT to invert all displayed colours


  // The TFT_eSprite class inherits the following functions (not all are useful to Sprite class
  void      setAddrWindow(int32_t xs, int32_t ys, int32_t w, int32_t h); // Note: start coordinates + width and height

  // Viewport commands, see "Viewport_Demo" sketch
  void      setViewport(int32_t x, int32_t y, int32_t w, int32_t h, bool vpDatum = true);
  bool      checkViewport(int32_t x, int32_t y, int32_t w, int32_t h);
  int32_t   getViewportX(void);
  int32_t   getViewportY(void);
  int32_t   getViewportWidth(void);
  int32_t   getViewportHeight(void);
  bool      getViewportDatum(void);
  void      frameViewport(uint16_t color, int32_t w);
  void      resetViewport(void);

            // Clip input window to viewport bounds, return false if whole area is out of bounds
  bool      clipAddrWindow(int32_t* x, int32_t* y, int32_t* w, int32_t* h);
            // Clip input window area to viewport bounds, return false if whole area is out of bounds
  bool      clipWindow(int32_t* xs, int32_t* ys, int32_t* xe, int32_t* ye);

            // Push (aka write pixel) colours to the TFT (use setAddrWindow() first)
  void      pushColor(uint16_t color, uint32_t len),   // Deprecated, use pushBlock()
            pushColors(uint16_t   *data, uint32_t len, bool swap = true), // With byte swap option
            pushColors(uint8_t   *data, uint32_t len); // Deprecated, use pushPixels()

            // Write a solid block of a single colour
  void      pushBlock(uint16_t color, uint32_t len);

            // Write a set of pixels stored in memory, use setSwapBytes(true/false) function to correct endianess
  void      pushPixels(const void * data_in, uint32_t len);

            // Support for half duplex (bi-directional SDA) SPI bus where MOSI must be switched to input
            #ifdef TFT_SDA_READ
              #if defined (TFT_eSPI_ENABLE_8_BIT_READ)
  uint8_t   tft_Read_8(void);       // Read 8 bit value from TFT command register
              #endif
  void      begin_SDA_Read(void); // Begin a read on a half duplex (bi-directional SDA) SPI bus - sets MOSI to input
  void      end_SDA_Read(void);    // Restore MOSI to output
            #endif

  // Graphics drawing
  void      fillScreen(uint32_t color),
            drawRect(int32_t x, int32_t y, int32_t w, int32_t h, uint32_t color),
            drawRoundRect(int32_t x, int32_t y, int32_t w, int32_t h, int32_t radius, uint32_t color),
            fillRoundRect(int32_t x, int32_t y, int32_t w, int32_t h, int32_t radius, uint32_t color);

  void      fillRectVGradient(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t color1, uint32_t color2);
  void      fillRectHGradient(int16_t x, int16_t y, int16_t w, int16_t h, uint32_t color1, uint32_t color2);

            // Draw a pixel blended with the pixel colour on the TFT or sprite, return blended colour
            // If bg_color is not included the background pixel colour will be read from TFT or sprite
  uint16_t drawPixel(int32_t x, int32_t y, uint32_t color, uint8_t alpha, uint32_t bg_color = 0x00FFFFFF);

            // Draw a small anti-aliased filled circle at ax,ay with radius r (uses drawWideLine)
            // If bg_color is not included the background pixel colour will be read from TFT or sprite
  void      drawSpot(float ax, float ay, float r, uint32_t fg_color, uint32_t bg_color = 0x00FFFFFF);

            // Draw an anti-aliased filled circle at x, y with radius r
            // If bg_color is not included the background pixel colour will be read from TFT or sprite
  void      fillSmoothCircle(int32_t x, int32_t y, int32_t r, uint32_t color, uint32_t bg_color = 0x00FFFFFF);

  void      fillSmoothRoundRect(int32_t x, int32_t y, int32_t w, int32_t h, int32_t radius, uint32_t color, uint32_t bg_color = 0x00FFFFFF);
```

```
          // Draw an anti-aliased wide line from ax,ay to bx,by width wd with radiused ends (radius is
wd/2)
          // If bg_color is not included the background pixel colour will be read from TFT or sprite
  void      drawWideLine(float ax, float ay, float bx, float by, float wd, uint32_t fg_color, uint32_t
bg_color = 0x00FFFFFF);

          // Draw an anti-aliased wide line from ax,ay to bx,by with different width at each end aw,
bw and with radiused ends
          // If bg_color is not included the background pixel colour will be read from TFT or sprite
  void      drawWedgeLine(float ax, float ay, float bx, float by, float aw, float bw, uint32_t fg_color,
uint32_t bg_color = 0x00FFFFFF);

  void      drawCircle(int32_t x, int32_t y, int32_t r, uint32_t color),
          drawCircleHelper(int32_t x, int32_t y, int32_t r, uint8_t cornername, uint32_t color),
          fillCircle(int32_t x, int32_t y, int32_t r, uint32_t color),
          fillCircleHelper(int32_t x, int32_t y, int32_t r, uint8_t cornername, int32_t delta, uint32_t
color),

          drawEllipse(int16_t x, int16_t y, int32_t rx, int32_t ry, uint16_t color),
          fillEllipse(int16_t x, int16_t y, int32_t rx, int32_t ry, uint16_t color),

          //                  Corner 1              Corner 2              Corner 3
          drawTriangle(int32_t x1,int32_t y1, int32_t x2,int32_t y2, int32_t x3,int32_t y3, uint32_t
color),
          fillTriangle(int32_t x1,int32_t y1, int32_t x2,int32_t y2, int32_t x3,int32_t y3, uint32_t color);

  // Image rendering
          // Swap the byte order for pushImage() and pushPixels() - corrects endianness
  void      setSwapBytes(bool swap);
  bool      getSwapBytes(void);

          // Draw bitmap
  void      drawBitmap( int16_t x, int16_t y, const uint8_t *bitmap, int16_t w, int16_t h, uint16_t
fgcolor),
          drawBitmap( int16_t x, int16_t y, const uint8_t *bitmap, int16_t w, int16_t h, uint16_t
fgcolor, uint16_t bgcolor),
          drawXBitmap(int16_t x, int16_t y, const uint8_t *bitmap, int16_t w, int16_t h, uint16_t
fgcolor),
          drawXBitmap(int16_t x, int16_t y, const uint8_t *bitmap, int16_t w, int16_t h, uint16_t
fgcolor, uint16_t bgcolor),
          setBitmapColor(uint16_t fgcolor, uint16_t bgcolor); // Define the 2 colours for 1bpp sprites

          // Set TFT pivot point (use when rendering rotated sprites)
  void      setPivot(int16_t x, int16_t y);
  int16_t   getPivotX(void), // Get pivot x
          getPivotY(void); // Get pivot y

          // The next functions can be used as a pair to copy screen blocks (or horizontal/vertical
lines) to another location
          // Read a block of pixels to a data buffer, buffer is 16 bit and the size must be at least w *
h
  void      readRect(int32_t x, int32_t y, int32_t w, int32_t h, uint16_t *data);
          // Write a block of pixels to the screen which have been read by readRect()
  void      pushRect(int32_t x, int32_t y, int32_t w, int32_t h, uint16_t *data);

          // These are used to render images or sprites stored in RAM arrays (used by Sprite class
for 16bpp Sprites)
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, uint16_t *data);
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, uint16_t *data, uint16_t
transparent);

          // These are used to render images stored in FLASH (PROGMEM)
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, const uint16_t *data, uint16_t
transparent);
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, const uint16_t *data);

          // These are used by Sprite class pushSprite() member function for 1, 4 and 8 bits per
pixel (bpp) colours
          // They are not intended to be used with user sketches (but could be)
          // Set bpp8 true for 8bpp sprites, false otherwise. The cmap pointer must be specified for
4bpp
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, uint8_t  *data, bool bpp8 = true,
uint16_t *cmap = nullptr);
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, uint8_t  *data, uint8_t  transparent,
bool bpp8 = true, uint16_t *cmap = nullptr);
          // FLASH version
  void      pushImage(int32_t x, int32_t y, int32_t w, int32_t h, const uint8_t *data, bool
bpp8,  uint16_t *cmap = nullptr);
          // This next function has been used successfully to dump the TFT screen to a PC for
documentation purposes
          // It reads a screen area and returns the 3 RGB 8 bit colour values of each pixel in the
buffer
          // Set w and h to 1 to read 1 pixel's colour. The data buffer must be at least w * h * 3 bytes
  void      readRectRGB(int32_t x, int32_t y, int32_t w, int32_t h, uint8_t *data);

  // Text rendering - value returned is the pixel width of the rendered text
  int16_t   drawNumber(long intNumber, int32_t x, int32_t y, uint8_t font), // Draw integer using
specified font number
```

```
        drawNumber(long intNumber, int32_t x, int32_t y),                // Draw integer using current font

        // Decimal is the number of decimal places to render
        // Use with setTextDatum() to position values on TFT, and setTextPadding() to blank old displayed values
        drawFloat(float floatNumber, uint8_t decimal, int32_t x, int32_t y, uint8_t font), // Draw float using specified font number
        drawFloat(float floatNumber, uint8_t decimal, int32_t x, int32_t y),             // Draw float using current font

        // Handle char arrays
        // Use with setTextDatum() to position string on TFT, and setTextPadding() to blank old displayed strings
        drawString(const char *string, int32_t x, int32_t y, uint8_t font),   // Draw string using specified font number
        drawString(const char *string, int32_t x, int32_t y),                // Draw string using current font
        drawString(const String& string, int32_t x, int32_t y, uint8_t font),// Draw string using specified font number
        drawString(const String& string, int32_t x, int32_t y),              // Draw string using current font

        drawCentreString(const char *string, int32_t x, int32_t y, uint8_t font),   // Deprecated, use setTextDatum() and drawString()
        drawRightString(const char *string, int32_t x, int32_t y, uint8_t font),    // Deprecated, use setTextDatum() and drawString()
        drawCentreString(const String& string, int32_t x, int32_t y, uint8_t font),// Deprecated, use setTextDatum() and drawString()
        drawRightString(const String& string, int32_t x, int32_t y, uint8_t font); // Deprecated, use setTextDatum() and drawString()

  // Text rendering and font handling support funtions
  void       setCursor(int16_t x, int16_t y),                  // Set cursor for tft.print()
             setCursor(int16_t x, int16_t y, uint8_t font);    // Set cursor and font number for tft.print()

  int16_t   getCursorX(void),                                  // Read current cursor x position (moves with tft.print())
             getCursorY(void);                                 // Read current cursor y position

  void       setTextColor(uint16_t color),                     // Set character (glyph) color only (background not over-written)
             setTextColor(uint16_t fgcolor, uint16_t bgcolor, bool bgfill = false),   // Set character (glyph) foreground and background colour, optional background fill for smooth fonts
             setTextSize(uint8_t size);                        // Set character size multiplier (this increases pixel size)

  void       setTextWrap(bool wrapX, bool wrapY = false);      // Turn on/off wrapping of text in TFT width and/or height

  void       setTextDatum(uint8_t datum);                      // Set text datum position (default is top left), see Section 6 above
  uint8_t   getTextDatum(void);

  void       setTextPadding(uint16_t x_width);                 // Set text padding (background blanking/over-write) width in pixels
  uint16_t getTextPadding(void);                               // Get text padding

#ifdef LOAD_GFXFF
  void       setFreeFont(const GFXfont *f = NULL),             // Select the GFX Free Font
             setTextFont(uint8_t font);                        // Set the font number to use in future
#else
  void       setFreeFont(uint8_t font),                        // Not used, historical fix to prevent an error
             setTextFont(uint8_t font);                        // Set the font number to use in future
#endif

  int16_t   textWidth(const char *string, uint8_t font),       // Returns pixel width of string in specified font
             textWidth(const char *string),                    // Returns pixel width of string in current font
             textWidth(const String& string, uint8_t font),    // As above for String types
             textWidth(const String& string),
             fontHeight(int16_t font),                         // Returns pixel height of string in specified font
             fontHeight(void);                                 // Returns pixel width of string in current font

             // Used by library and Smooth font class to extract Unicode point codes from a UTF8 encoded string
  uint16_t decodeUTF8(uint8_t *buf, uint16_t *index, uint16_t remaining),
             decodeUTF8(uint8_t c);

             // Support function to UTF8 decode and draw characters piped through print stream
  size_t    write(uint8_t);
  // size_t    write(const uint8_t *buf, size_t len);

             // Used by Smooth font class to fetch a pixel colour for the anti-aliasing
  void       setCallback(getColorCallback getCol);
```

```
  uint16_t fontsLoaded(void); // Each bit in returned value represents a font type that is loaded - used for debug/error handling only

  // Low level read/write
  void     spiwrite(uint8_t);          // legacy support only
#ifndef RM68120_DRIVER
  void     writecommand(uint8_t c);   // Send a command, function resets DC/RS high ready for data
#else
  void     writecommand(uint16_t c); // Send a command, function resets DC/RS high ready for data
  void     writeRegister(uint16_t c, uint8_t d); // Write data to 16 bit command register
#endif
  void     writedata(uint8_t d);       // Send data with DC/RS set high

  void     commandList(const uint8_t *addr); // Send a initialisation sequence to TFT stored in FLASH

  uint8_t   readcommand8( uint8_t cmd_function, uint8_t index = 0); // read 8 bits from TFT
  uint16_t readcommand16(uint8_t cmd_function, uint8_t index = 0); // read 16 bits from TFT
  uint32_t readcommand32(uint8_t cmd_function, uint8_t index = 0); // read 32 bits from TFT


  // Colour conversion
          // Convert 8 bit red, green and blue to 16 bits
  uint16_t color565(uint8_t red, uint8_t green, uint8_t blue);

          // Convert 8 bit colour to 16 bits
  uint16_t color8to16(uint8_t color332);
          // Convert 16 bit colour to 8 bits
  uint8_t   color16to8(uint16_t color565);

          // Convert 16 bit colour to/from 24 bit, R+G+B concatenated into LS 24 bits
  uint32_t color16to24(uint16_t color565);
  uint32_t color24to16(uint32_t color888);

          // Alpha blend 2 colours, see generic "alphaBlend_Test" example
          // alpha =   0 = 100% background colour
          // alpha = 255 = 100% foreground colour
  uint16_t alphaBlend(uint8_t alpha, uint16_t fgc, uint16_t bgc);
          // 16 bit colour alphaBlend with alpha dither (dither reduces colour banding)
  uint16_t alphaBlend(uint8_t alpha, uint16_t fgc, uint16_t bgc, uint8_t dither);
          // 24 bit colour alphaBlend with optional alpha dither
  uint32_t alphaBlend24(uint8_t alpha, uint32_t fgc, uint32_t bgc, uint8_t dither = 0);


  // DMA support functions - these are currently just for SPI writes when using the ESP32 or STM32 processors
  // DMA works also on RP2040 and PIO SPI, 8 bit parallel and 16 bit parallel
          // Bear in mind DMA will only be of benefit in particular circumstances and can be tricky
          // to manage by noobs. The functions have however been designed to be noob friendly and
          // avoid a few DMA behaviour "gotchas".
          //
          // At best you will get a 2x TFT rendering performance improvement when using DMA because
          // this library handles the SPI bus so efficiently during normal (non DMA) transfers. The best
          // performance improvement scenario is the DMA transfer time is exactly the same as the time it
          // takes for the processor to prepare the next image buffer and initiate another DMA transfer.
          //
          // DMA transfer to the TFT is done while the processor moves on to handle other tasks. Bear
          // this in mind and watch out for "gotchas" like the image buffer going out of scope as the
          // processor leaves a function or its content being changed while the DMA engine is reading it.
          //
          // The compiler MAY change the implied scope of a buffer which has been set aside by creating
          // an array. For example a buffer defined before a "for-next" loop may get de-allocated when
          // the loop ends. To avoid this use, for example, malloc() and free() to take control of when
          // the buffer space is available and ensure it is not released until DMA is complete.
          //
          // Clearly you should not modify a buffer that is being DMA'ed to the TFT until the DMA is over.
          // Use the dmaBusy() function to check this.   Use tft.startWrite() before invoking DMA so the
          // TFT chip select stays low. If you use tft.endWrite() before DMA is complete then the endWrite
          // function will wait for the DMA to complete, so this may defeat any DMA performance benefit.
          //
```

```
  bool      initDMA(bool ctrl_cs = false);   // Initialise the DMA engine and attach to SPI bus - typically used in setup()
                                             // Parameter "true" enables DMA engine control of TFT chip select (ESP32 only)
                                             // For ESP32 only, TFT reads will not work if parameter is true
  void      deInitDMA(void);     // De-initialise the DMA engine and detach from SPI bus - typically not used

            // Push an image to the TFT using DMA, buffer is optional and grabs (double buffers) a copy of the image
            // Use the buffer if the image data will get over-written or destroyed while DMA is in progress
            // If swapping colour bytes is defined, and the double buffer option is NOT used, then the bytes
            // in the original data image will be swapped by the function before DMA is initiated.
            // The function will wait for the last DMA to complete if it is called while a previous DMA is still
            // in progress, this simplifies the sketch and helps avoid "gotchas".
  void      pushImageDMA(int32_t x, int32_t y, int32_t w, int32_t h, uint16_t* data, uint16_t* buffer = nullptr);

#if defined (ESP32) // ESP32 only at the moment
            // For case where pointer is a const and the image data must not be modified (clipped or byte swapped)
  void      pushImageDMA(int32_t x, int32_t y, int32_t w, int32_t h, uint16_t const* data);
#endif
            // Push a block of pixels into a window set up using setAddrWindow()
  void      pushPixelsDMA(uint16_t* image, uint32_t len);

            // Check if the DMA is complete - use while(tft.dmaBusy); for a blocking wait
  bool      dmaBusy(void); // returns true if DMA is still in progress
  void      dmaWait(void); // wait until DMA is complete

  bool      DMA_Enabled = false;    // Flag for DMA enabled state
  uint8_t   spiBusyCheck = 0;       // Number of ESP32 transfer buffers to check

  // Bare metal functions
  void      startWrite(void);                       // Begin SPI transaction
  void      writeColor(uint16_t color, uint32_t len); // Deprecated, use pushBlock()
  void      endWrite(void);                         // End SPI transaction

  // Set/get an arbitrary library configuration attribute or option
  //        Use to switch ON/OFF capabilities such as UTF8 decoding - each attribute has a unique ID
  //        id = 0: reserved - may be used in future to reset all attributes to a default state
  //        id = 1: Turn on (a=true) or off (a=false) GLCD cp437 font character error correction
  //        id = 2: Turn on (a=true) or off (a=false) UTF8 decoding
  //        id = 3: Enable or disable use of ESP32 PSRAM (if available)
            #define CP437_SWITCH 1
            #define UTF8_SWITCH  2
            #define PSRAM_ENABLE 3
  void      setAttribute(uint8_t id = 0, uint8_t a = 0); // Set attribute value
  uint8_t   getAttribute(uint8_t id = 0);                // Get attribute value

            // Used for diagnostic sketch to see library setup adopted by compiler, see Section 7 above
  void      getSetup(setup_t& tft_settings); // Sketch provides the instance to populate
  bool      verifySetupID(uint32_t id);

  // Global variables
  static    SPIClass& getSPIinstance(void); // Get SPI class handle

  uint32_t textcolor, textbgcolor;           // Text foreground and background colours

  uint32_t bitmap_fg, bitmap_bg;             // Bitmap foreground (bit=1) and background (bit=0) colours

  uint8_t   textfont,   // Current selected font number
            textsize,   // Current font size multiplier
            textdatum, // Text reference datum
            rotation;   // Display rotation (0-3)

  uint8_t   decoderState = 0;    // UTF8 decoder state          - not for user access
  uint16_t decoderBuffer;        // Unicode code-point buffer - not for user access

 //------------------------------------ private ------------------------------------//
 private:
            // Legacy begin and end prototypes - deprecated TODO: delete
  void      spi_begin();
  void      spi_end();

  void      spi_begin_read();
  void      spi_end_read();

            // New begin and end prototypes
            // begin/end a TFT write transaction
            // For SPI bus the transmit clock rate is set
  inline void begin_tft_write() __attribute__((always_inline));
```

```
inline void end_tft_write()     __attribute__((always_inline));

        // begin/end a TFT read transaction
        // For SPI bus: begin lowers SPI clock rate, end reinstates transmit clock rate
inline void begin_tft_read()   __attribute__((always_inline));
inline void end_tft_read()     __attribute__((always_inline));

        // Initialise the data bus GPIO and hardware interfaces
void       initBus(void);

        // Temporary   library development function   TODO: remove need for this
void       pushSwapBytePixels(const void* data_in, uint32_t len);

        // Same as setAddrWindow but exits with CGRAM in read mode
void       readAddrWindow(int32_t xs, int32_t ys, int32_t w, int32_t h);

        // Byte read prototype
uint8_t   readByte(void);

        // GPIO parallel bus input/output direction control
void       busDir(uint32_t mask, uint8_t mode);

        // Single GPIO input/output direction control
void       gpioMode(uint8_t gpio, uint8_t mode);

        // Helper function: calculate distance of a point from a finite length line between two points
float     wedgeLineDistance(float pax, float pay, float bax, float bay, float dr);

        // Display variant settings
uint8_t   tabcolor,                     // ST7735 screen protector "tab" colour (now invalid)
        colstart = 0, rowstart = 0; // Screen display area to CGRAM area coordinate offsets

        // Port and pin masks for control signals (ESP826 only) - TODO: remove need for this
volatile uint32_t *dcport, *csport;
uint32_t cspinmask, dcpinmask, wrpinmask, sclkpinmask;

        #if defined(ESP32_PARALLEL)
        // Bit masks for ESP32 parallel bus interface
uint32_t xclr_mask, xdir_mask; // Port set/clear and direction control masks

        // Lookup table for ESP32 parallel bus interface uses 1kbyte RAM,
uint32_t xset_mask[256]; // Makes Sprite rendering test 33% faster, for slower macro equivalent
                         // see commented out #define set_mask(C) within TFT_eSPI_ESP32.h
        #endif
```

```
 //uint32_t lastColor = 0xFFFF; // Last colour - used to minimise bit shifting overhead

 getColorCallback getColor = nullptr; // Smooth font callback function pointer

 bool      locked, inTransaction, lockTransaction; // SPI transaction and mutex lock flags

//------------------------------------ protected ----------------------------------//
protected:

 //int32_t  win_xe, win_ye;           // Window end coords - not needed

 int32_t  _init_width, _init_height; // Display w/h as input, used by setRotation()
 int32_t  _width, _height;           // Display w/h as modified by current rotation
 int32_t  addr_row, addr_col;        // Window position - used to minimise window commands

 int16_t  _xPivot;   // TFT x pivot point coordinate for rotated Sprites
 int16_t  _yPivot;   // TFT x pivot point coordinate for rotated Sprites

 // Viewport variables
 int32_t  _vpX, _vpY, _vpW, _vpH;     // Note: x start, y start, x end + 1, y end + 1
 int32_t  _xDatum;
 int32_t  _yDatum;
 int32_t  _xWidth;
 int32_t  _yHeight;
 bool      _vpDatum;
 bool      _vpOoB;

 int32_t  cursor_x, cursor_y, padX;         // Text cursor x,y and padding setting
 int32_t  bg_cursor_x;                      // Background fill cursor
 int32_t  last_cursor_x;                    // Previous text cursor position when fill used

 uint32_t fontsloaded;                  // Bit field of fonts loaded

 uint8_t  glyph_ab,    // Smooth font glyph delta Y (height) above baseline
          glyph_bb;    // Smooth font glyph delta Y (height) below baseline

 bool      isDigits;    // adjust bounding box for numbers to reduce visual jiggling
 bool      textwrapX, textwrapY;   // If set, 'wrap' text at right and optionally bottom edge of display
 bool      _swapBytes; // Swap the byte order for TFT pushImage()

 bool      _booted;     // init() or begin() has already run once

                    // User sketch manages these via set/getAttribute()
```

```
  bool      _cp437;          // If set, use correct CP437 charset (default is ON)
  bool      _utf8;           // If set, use UTF-8 decoder in print stream 'write()' function (default ON)
  bool      _psram_enable; // Enable PSRAM use for library functions (TBD) and Sprites

  uint32_t _lastColor; // Buffered value of last colour used

  bool      _fillbg;     // Fill background flag (just for for smooth fonts at the moment)

#if defined (SSD1963_DRIVER)
  uint16_t Cswap;        // Swap buffer for SSD1963
  uint8_t r6, g6, b6;   // RGB buffer for SSD1963
#endif

#ifdef LOAD_GFXFF
  GFXfont   *gfxFont;
#endif

/***************************************************************************************
**                          Section 9: TFT_eSPI class conditional extensions
***************************************************************************************/
// Load the Touch extension
#ifdef TOUCH_CS
  #if defined (TFT_PARALLEL_8_BIT) || defined (RP2040_PIO_INTERFACE)
    #if !defined(DISABLE_ALL_LIBRARY_WARNINGS)
      #error >>>>------>> Touch functions not supported in 8/16 bit parallel mode or with RP2040
PIO.
    #endif
  #else
    #include "Extensions/Touch.h"          // Loaded if TOUCH_CS is defined by user
  #endif
#else
    #if !defined(DISABLE_ALL_LIBRARY_WARNINGS)
      #warning >>>>------>> TOUCH_CS pin not defined, TFT_eSPI touch functions will not be
available!
    #endif
#endif

// Load the Anti-aliased font extension
#ifdef SMOOTH_FONT
  #include "Extensions/Smooth_font.h"   // Loaded if SMOOTH_FONT is defined by user
#endif

}; // End of class TFT_eSPI
```

```
/***************************************************************************************
**                          Section 10: Additional extension classes
***************************************************************************************/
// Load the Button Class
#include "Extensions/Button.h"

// Load the Sprite Class
#include "Extensions/Sprite.h"

#endif // ends #ifndef _TFT_eSPIH_
```