ETMS_ST7796S TnT Library
User Guide

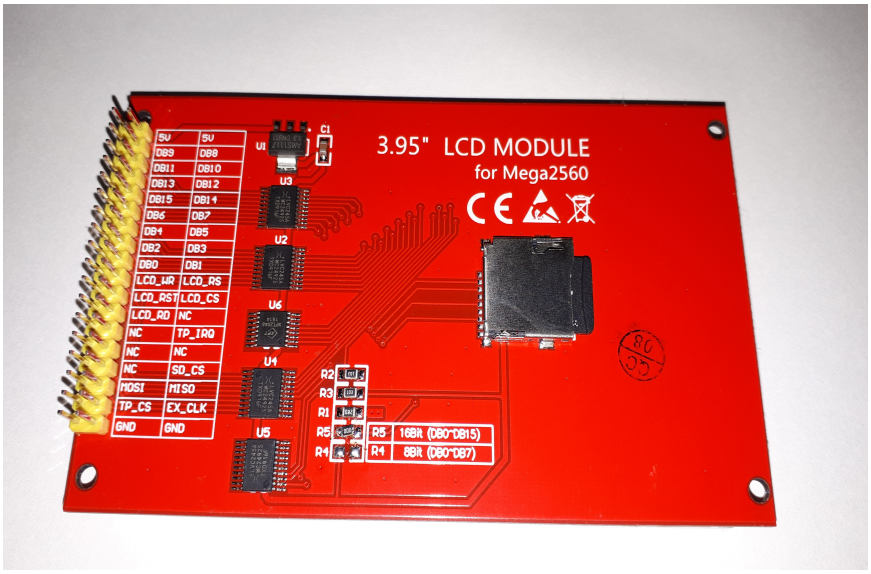
By Nash Ali

Version 1.00

Dec.,11, 2020.

INTRODUCTION

This library only supports this 3.95" colour tft display.



SECTION 1

ETMS_ST7796S TnT Library Commands

Initialize Display – Done once at beginning of program. - see examples folder.

Usage: init();
Prototypes supported:

init();

Display On – Turns on display, takes the display out of displayOff mode.

Usage: displayOn();
Prototypes supported:

displayOn();

Display Off – Turns off display, shuts down HV. Requires displayOn to resume.

Usage: displayOff();
Prototypes supported:

displayOff();

Set Brightness – Set the brightness value of the TFT – ranges from 0x00 to 0xFF.

Usage: setBrightness();
Prototypes supported:

setBrightness(uint8_t br);

Invert Display – Inverts the display.

Usage: invertDisplay();
Prototypes supported:

invertDisplay(uint8_t inv);

Set Address Window – sets the display window (CASET/RASET) for RAMWR operations.

This is the access to the frame buffer 480 x 320 x 18.

Usage: setAddressWindow(x-start,y-start,x-dest,y-dest);
Prototypes supported:

setAddressWindow(int16_t xorg, int16_t yorg,int16_t xdest, int16_t ydest);

Set Fill Window – fills the window with RGB888 colour for the length specified.

Usage: fillWindow(x-start,y-start,x-dest,y-dest);
Prototypes supported:

fillWindow(uint8_t red, uint8_t green, uint8_t blue, int length);

fillWindow(uint16_t colour, int length);

ETMS_ST7796S TnT Library Commands(Contd)

Clear Display – clears the frame buffer. Set all pixels to BLACK (0,0,0).

Usage: clrDisplay(); clrScr();

Prototypes supported:

clrDisplay();

clrScr();

Draw Pixel – draws a pixel of the specified colour and position.

Usage: drawPixel(x-coord, y-coord[,colour565][,red,green,blue]);

Prototypes supported:

drawPixel(uint16_t xp, uint16_t yp);

drawPixel(uint16_t xp, uint16_t yp, uint16_t colour565);

drawPixel(uint16_t xp, uint16_t yp, uint8_t red, uint8_t green, uint8_t blue);

Draw Horizontal Line – draws a line starting at xs,ys and moving to the right for the length in pixels.

Usage: drawHLine(x-start,y-start,length);
Prototypes supported:

drawHLine(uint16_t xs, uint16_t ys, uint16_t length);

Draw Vertical Line – draws a vertical line downwards in length in pixels.

Usage: drawVLine(x-start,y-start,length);
Prototypes supported:

drawVLine(uint16_t xs, uint16_t ys, uint16_t length);

Draw Rectangle – draws a box starting at xs,ys as the top-left corner and the width in pixels across and the height in pixels downward.

Usage: drawRect(x-start,y-start,width,height);
Prototypes supported:

drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height);

ETMS_ST7796S TnT Library Commands(Contd)

Print Character – print a single character from the selected font file, the x position is automatically incremented and at the end of the line there is roll-over to the next line by incrementing the y position and resetting the x position to the start of the line.

Usage: printChar(CHAR[,x-pos,ypos,[colour565][red,green,blue]]);
Prototypes supported:

printChar(uint8_t c);

printChar(uint8_t c, uint16_t xpos, uint16_t ypos, uint16_t colour);

printChar(uint8_t c, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue);

Print String – prints an string of characters in the specified colour. Can use 24 bit or 16 bit colour.

Usage: printString("CHAR",xpos,ypos);
Prototypes supported:

printString(char x);

printString(char x, uint16_t xpos, uint16_t ypos);

printString(char x, uint16_t xpos, uint16_t ypos, uint16_t colour);

printString(char x, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue);

printString(char *x);

printString(char *x, uint16_t xpos, uint16_t ypos, uint16_t colour);

printString(char *x, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue);

Print Number – prints a number in the specified format integer/float at the specified location xpos, ypos..

Usage: printNumI(val,xpos,ypos[,divider='.'][,digits][, spacer=' ']);

printNumF(num, xpos, ypos[, dec][, filler][, length]);

Prototypes supported:

printNumI(long val, uint16_t xpos, uint16_t ypos, [int length, char filler]);

printNumF(double num, uint16_t x, uint16_t y, byte dec=5, char divider='.', int length=0, char filler=' ');

Set Background Colour – sets the background colour in RGB565 format. The RGB prototype supports RGB888 to RGB565 conversion.

Usage:setBackColour(colour);

Prototypes supported:

setBackColour(uint16_t colour);

setBackColour(uint8_t red, uint8_t green, uint8_t blue);

ETMS_ST7796S TnT Library Commands(Contd)

Set Foreground Colour – set the foreground colour in RGB565 format. The RGB prototype supports RGB888 to RGB565 conversion.

Usage: setColour(colour);

Prototypes supported:

setColour(uint16_t colour);

setColour(uint8_t red, uint8_t green, uint8_t blue);

Set Font – sets the font to use for text operations. Your custom fonts can be installed in the driver's directory and delcared in your program. See example folder LibraryTest programs.

Usage: void setFont(char *font);
Prototypes supported:
setFont(SmallFont);

Get Font – gets the current font in use.

Usage: void setFont(char *font);
Prototypes supported:
uint8_t* getFont();

Get Font X Size – gets the current font x dimension.

Usage: getFontXsize();
Prototypes supported:
uint8_t getFontXsize();

Get Font Y Size – gets the current font y dimension.

Usage: getFontYsize();
Prototypes supported:
uint8_t getFontYsize();

Set Cursor – sets the x,y position for the next screen operation.

Usage: setCursor(xpos,ypos);
Prototypes supported:
setCursor(uint16_t xpos, uint16_t ypos);

Set Guage – set guage.(NEW) – concept stage.

Usage: setGuage(xpos,ypos,value,type);
Prototypes supported:

SECTION 2

Software description.

While the ST7796S controller hardware is configured for16-bit data, the software initialization sets up the colour palette for 565 encoding (65K colour), this allows the pixel transfers to the frame buffer to equal one pixel for each 16-bit word. LSB Data0 – Data4 is the 5 bit blue component, Data5 – Data 10 is the 6 bit green while Data11-Data15 MSB is the 5 bit red component.

Library Header:

```
/*
   Created:      06:23 PM 10/04/2020
   Last Updated: 12:05 PM 11/16/2020
   MIT License

   Copyright (c) 2020 Zulfikar Naushad(Nash) Ali

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.
  Uses the Arduino DUE ONLY!
  NOTES:
  This program tests the new ETMS library for the generic 3.95 parallel i/f ST7796S
  tft colour display controller.

    ETMS_ST7796S.h - library setup

    The #ifndef statement checks to see if the ETMS_ST7796S.h
    file isn't already defined. This is to stop double declarations
    of any identifiers within the library. It is paired with a
    #endif at the bottom of the header and this setup is known as
    an 'Include Guard'.
    The following def handles the ARM pcb.

*/

#ifndef __SAM3X8E__
   #define __SAM3X8E__
   #include "hardware/arm/HW_SAM3X8E.h"
#endif

#ifndef _etms_st7796s_h_
#define _etms_st7796s_h_

// st7796s registers
#define NOP     0x00       //  no operation
#define SWRESET 0x01       //  software reset
#define RDDID   0x04       //  read display ID
#define RDDSI   0x05
#define RDDST   0x09
```

```
#define RDDPM   0x0A       //  read display power mode
#define RDDMADCTL   0x0B
#define RDDCOLMOD   0x0C
#define RDDIM   0x0D       //  read display image mode
#define RDDSM   0x0E       //  read display signal mode
#define RDDSDR  0x0F       //  read display self-diagnostic result

#define SLPIN   0x10       //  sleep in
#define SLPOUT  0x11       //  sleep out
#define PTLON   0x12       //  partial display on
#define NORON   0x13       //  normal display on
#define INVOFF  0x20       //  display inversion off
#define INVON   0x21       //  display inversion on
#define DISPOFF 0X28       //  display off
#define DISPON  0x29       //  display on

#define CASET   0x2A       //  column address set
#define RASET   0x2B       //  row address set
#define RAMWR   0x2C       //  ram write
#define RAMRD   0x2E       //  ram read

#define PTLAR   0x30       //  partial area
#define VSCRDEF 0x33       //  vertical scroll definition
#define MADCTL  0x36       //  Memory data access control
#define VSCRSADD    0x37   //  vertical scroll start address of RAM
#define IDMOFF  0x38       //  idle mode off
#define IDMON   0x39       //  idle mode on
#define COLMOD  0x3A       //  interface pixel format register - colour data input format - RGB888 - RGB565.
#define RAMWRC  0x3C       //  write memory continue
#define RAMRDC  0x3E       //  read memory continue

#define WRDISBV 0x51       //  write display brightness
#define RDDISBV 0x52       //  read display brightness
#define WRCTRLD 0x53       //  write ctrl display
#define RDCTRLD 0x54       //  read ctrl display
#define WRCABC  0x55       //  write adaptive brightness control value
#define RDCABC  0x56       //  read adaptive brightness control value
#define WRCABCMB    0x5E   //  write CABC minimun brightness
#define RDCABCMB    0x5F   //  read CABC minimun brightness

#define RDFCHKSUM   0xAA   //  read first checksm
#define RDCFCS  0xAF       //  read checksum continue

#define IFMODE  0xB0       //  interface mode control
#define FRMCTR1 0xB1       //  frame rate control - normal mode / full colours
#define FRMCTR2 0xB2       //  frame rate control - in idle mode / 8 colours
#define FRMCTR3 0xB3       //  frame rate control - partial mode / full colours
#define DIC     0xB4       //  display inversion control
#define BPC     0xB5       //  blanking porch control
#define DFC     0xB6       //  display function control
#define EM      0xB7       //  entry mode set
#define PWR1    0xC0       //  power control 1
#define PWR2    0xC1       //  power control 2
#define PWR3    0xC2       //  power control 3
#define VCMPCTL 0xC5       //  vcomp control
#define VCMOFFSET   0xC6   //  vcomp offset register

#define NVMADW  0xD0       //  nvm address/data write
#define NVMBPROG    0xD1   //  nvm byte program
#define NVMSTATRD   0xD2   //  nvm status read
#define RDID4   0xD3       //  read ID4
#define RDID1   0xDA       //  read ID1
#define RDID2   0xDB       //  read ID2
#define RDID3   0xDC       //  read ID3

#define PGC     0xE0       //  positive gamma
#define NGC     0xE1       //  negative gamma
#define DGC1    0xE2       //  digital gamma 1
#define DGC2    0xE3       //  digital gamma 2
#define DOCA    0xE8       //  display output conrtol adjust
```

```
#define CSCON   0xF0        //  command set control
#define SPIRDCTRL   0xFB    //  spi read control
#define PROMACT 0xFE

//  masks
#define MADCTL_MY  0x80
#define MADCTL_MX  0x40
#define MADCTL_MV  0x20
#define MADCTL_ML  0x10
#define MADCTL_RGB 0x08
#define MADCTL_DISL 0x04

#define RGB888  0x07    //  24bit -  16M
#define RGB666  0x06    //  18bit -  262K
#define RGB565  0x05    //  16bit -  65K

#define BUS16BIT    0x50    //  16 bit bus
#define BUS18BIT    0x60    //  18 bit bus

#define ON  true
#define OFF false
#define PORTRAIT 0
#define LANDSCAPE 1
#define LEFT 0
#define RIGHT 9999
#define CENTER 9998
#define TOP 9997
#define MIDDLE  9996
#define BOTTOM  9995

//  standard colour defines ************************************************
//  565 colour defines
#define LTBLUE    0xB6DF
#define LTTEAL    0xBF5F
#define LTGREEN   0xBFF7
#define LTCYAN    0xC7FF
#define LTRED     0xFD34
#define LTMAGENTA 0xFD5F
#define LTYELLOW  0xFFF8
#define LTORANGE  0xFE73
#define LTPINK    0xFDDF
#define LTPURPLE  0xCCFF
#define LTGREY    0xE71C

#define BLUE      0x001F
#define TEAL      0x0438
#define GREEN     0x07E0
#define CYAN      0x07FF
#define RED       0xF800
#define MAGENTA   0xF81F
#define YELLOW    0xFFE0
#define ORANGE    0xFC00
#define PINK      0xF81F
#define PURPLE    0x8010
#define GREY      0xC618

#define WHITE     0xFFFF
#define BLACK     0x0000

#define DKBLUE    0x000D
#define DKTEAL    0x020C
#define DKGREEN   0x03E0
#define DKCYAN    0x03EF
#define DKRED     0x6000
#define DKMAGENTA 0x8008
#define DKYELLOW  0x8400
#define DKORANGE  0x8200
#define DKPINK    0x9009
#define DKPURPLE  0x4010
#define DKGREY    0x4A49
```

```
#define bitmapdatatype unsigned short*

/*
   The #include of Arduino.h gives this library access to the standard
   Arduino types and constants (HIGH, digitalWrite, etc.). It's
   unneccesary for sketches but required for libraries as they're not
   .ino (Arduino) files.
*/
#include "Arduino.h"

struct _current_font
{
   uint8_t* font;
   uint8_t x_size;
   uint8_t y_size;
   uint8_t offset;
   uint8_t numchars;
};

class ETMS_ST7796S{

   public:

      ETMS_ST7796S();

      //  Below are the functions for the class. These are the functions available in the library for the user to call.
      void init(boolean orientation = LANDSCAPE);
      void displayOn();
      void displayOff();
      void setBrightness(uint8_t br);
      uint8_t getID();
      uint16_t getHeight();
      uint16_t getDisplayYSize();
      uint16_t getWidth();
      uint16_t getDisplayXSize();
      uint16_t to565Colour(uint8_t red, uint8_t green, uint8_t blue); // convert RGB888
      void setPixel();
      void setPixel(uint16_t colour);
      void setPixel(uint8_t red, uint8_t green, uint8_t blue);
      void setCursor(uint16_t x, uint16_t y);
      void invertDisplay(uint8_t inv);
      void setTransparency(bool t);
      void setAddressWindow(uint16_t xorg, uint16_t yorg, uint16_t xdest, uint16_t ydest);
      void setXY(uint16_t xorg, uint16_t yorg, uint16_t xdest, uint16_t ydest);
      void clrXY();      //  resets the address window to maximum.
      void fillWindow(uint8_t red, uint8_t green, uint8_t blue, int length);
      void fillWindow(uint16_t colour, int length);
      void fastFill(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height);
      void clrDisplay();                                  //  writes 0s to the entire frame buffer.
      void clrScr();
      void fillScreen();
      void fillScreen(uint16_t colour);
      void fillScreen(uint8_t red, uint8_t green, uint8_t blue);

      void drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
      void drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t colour);
      void drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t red, uint8_t green, uint8_t blue);

      void drawPixel(uint16_t xpos, uint16_t ypos);
      void drawPixel(uint16_t xpos, uint16_t ypos, uint16_t colour);
      void drawPixel(uint16_t xp, uint16_t yp, uint8_t red, uint8_t green, uint8_t blue);

      void drawHLine(uint16_t xs,uint16_t ys ,uint16_t length);
      void drawHLine(uint16_t xs,uint16_t ys ,uint16_t length, uint16_t colour);
      void drawHLine(uint16_t xs,uint16_t ys ,uint16_t length, uint8_t red, uint8_t green, uint8_t blue);

      void drawVLine(uint16_t xs,uint16_t ys ,uint16_t length);
      void drawVLine(uint16_t xs,uint16_t ys ,uint16_t length, uint16_t colour);
      void drawVLine(uint16_t xs,uint16_t ys ,uint16_t length, uint8_t red, uint8_t green, uint8_t blue);
```

```
void drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height);
void drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour);
void drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blue);

void fillRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height);
void fillRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour);
void fillRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blue);

void drawRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height);
void drawRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour);
void drawRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blue);

void fillRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height);
void fillRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour);
void fillRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blue);

void drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
void drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t colour);
void drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2, uint8_t red, uint8_t green, uint8_t
blue);

void fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2);
void fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2, uint16_t colour);
void fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2, uint8_t red, uint8_t green, uint8_t b
lue);

void drawCircle(uint16_t xs, uint16_t ys, uint16_t radius);
void drawCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t colour);
void drawCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t lw, uint16_t colour);
void drawCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t lw, uint8_t red, uint8_t green, uint8_t blue);

void fillCircle(uint16_t xs, uint16_t ys, uint16_t radius);
void fillCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t colour);
void fillCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint8_t red, uint8_t green, uint8_t blue);

void drawArc(uint16_t x, uint16_t y, uint16_t r, uint16_t start, uint16_t stop, uint16_t colour);

void drawChar(byte c, uint16_t xpos, uint16_t ypos);
void drawChar(uint8_t c, uint16_t xpos, uint16_t ypos, uint16_t colour);
void drawChar(uint8_t c, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue);

void rotateChar(byte c, int x, int y, int pos, int deg);
void print(char *st, int x, int y, int deg=0);
void print(String st, int x, int y, int deg=0);

void printString(String x, uint16_t xpos, uint16_t ypos);
void printString(String x, uint16_t xpos, uint16_t ypos, uint16_t colour);
void printString(String x, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue);

void printString(String x, String xpos, uint16_t ypos);

void printNumI(long num, uint16_t x, uint16_t y, int length=0, char filler=' ');
void printNumF(double num, uint16_t x, uint16_t y, byte dec=5, char divider='.', int length=0, char filler=' ');

void drawBitMap(int x, int y, int sx, int sy, bitmapdatatype data, int scale=1);
void drawBitMap(int x, int y, int sx, int sy, bitmapdatatype data, int deg, int rox, int roy);

void setColour(uint16_t colour);
void setColour(uint8_t red, uint8_t green, uint8_t blue);

void setBackColour(uint16_t colour);
void setBackColour(uint8_t red, uint8_t green, uint8_t blue);
void setFgBgColour(uint16_t fcolour,uint16_t bcolour);
void setFont(uint8_t* font);

uint8_t* getFont();
uint8_t getFontXsize();
uint8_t getFontYsize();
```

```
  private:
      void SetOutputs();
      void Strobe_Write_Pin();
      void Strobe_Read_Pin();
      void Writ_Bus8(uint8_t val);
      void Writ_Bus16(uint16_t val);
      uint8_t Read_Bus8();
      uint16_t Read_Bus16();
      void Lcd_Dummy_Read();
      void Lcd_Write_Com(uint8_t com8);
      void Lcd_Write_Data8(uint8_t dat8);
      void Lcd_Write_Data16(uint16_t dat16);
      void Lcd_Write_Com_Data(uint8_t com8,uint8_t dat8);
      void Convert_Float(char *buf, double num, int width, byte prec);
      byte orient, fch, fcl, bch, bcl;
      long disp_x_size, disp_y_size;
      _current_font cfont;
      boolean  _TRANSPARENT = true;
      uint16_t _CurrentX, _CurrentY, _CurrentW, _CurrentH, _xStart, _yStart,  _colour565, _backcolour565;
};

//  The end wrapping of the #ifndef Include Guard
#endif
```

Library code:

```
/*
  ETMS_ST7796S.cpp
     Created:      03:06 PM 10/05/2020
   Last Updated: 12:01 PM 11/16/2020
   MIT License

   Copyright (c) 2020 Zulfikar Naushad(Nash) Ali

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and the associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in all
  copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
  SOFTWARE.

  NOTES:
    Uses the Arduino DUE ONLY! (has not been tested with any other product)
    This program tests the new ETMS library for the generic 3.95 parallel i/f ST7796S
    tft colour display controller in a 16-bit HW config. See docs.

  Commercial use of this library requires you to buy a license that
  will allow commercial use. This includes using the library,
  modified or not, as a tool to sell products.

  The license applies to all part of the library including the
  examples and tools supplied with the library.

  This will include the Header File so that the Source File has access
  to the function definitions in the ETMS_ST7796S library.
*/
```

```
#include "ETMS_ST7796S.h"
/*
  The #include of Arduino.h gives this library access to the standard
    Arduino types and constants (HIGH, digitalWrite, etc.). It's
  unneccesary for sketches but required for libraries as they're not
    .ino (Arduino) files.
*/
#include "Fonts.c"
#include "Arduino.h"
#define libver  100

// Pin shield control macros, this is like digitalWrite

#define PIN_LOW(port, pin)    (port)->PIO_CODR = (1<<(pin))
#define PIN_HIGH(port, pin)   (port)->PIO_SODR = (1<<(pin))
#define PIN_OUTPUT(port, pin) (port)->PIO_OER = (1<<(pin))

// configure masks for the control pins

#define ReadBitMask 0x100000;   //  PA20
#define WriteBitMask 0x80;      //  PC7
#define RegSelBitMask 0x40;     //  PC6
#define ChipSelBitMask 0x100;   //  PC8
#define ResetBitMask  0x200;    //  PC9

// configure macros for the 16 bit data bus @ pins 22 - 37

#define AMASK       ((1<<7)|(3<<14))        //PA7, PA14-PA15
#define BMASK       (1<<26)                 //PB26
#define CMASK       (31<<1)                 //PC1-PC5
#define DMASK       ((15<<0)|(1<<6)|(3<<9))   //PD0-PD3, PD6, PD9-PD10

#define write16(x)   { PIOA->PIO_CODR = AMASK; PIOB->PIO_CODR = BMASK; PIOC->PIO_CODR = CMASK; PIOD-
>PIO_CODR = DMASK; PIOA->PIO_SODR = (((x)&(1<<6))<<1)|(((x)&(3<<9))<<5); \
                PIOB->PIO_SODR = (((x)&(1<<8))<<18); PIOC->PIO_SODR = (((x)&(1<<0))<<5); PIOC->PIO_SODR = (((x)&(1
<<1))<<3); PIOC->PIO_SODR = (((x)&(1<<2))<<1); \
                PIOC->PIO_SODR = (((x)&(1<<3))>>1); PIOC->PIO_SODR = (((x)&(1<<4))>>3); PIOD->PIO_SODR = (((x)&(1<
<7))<<2)|(((x)&(1<<5))<<5)|(((x)&(15<<11))>>11)|(((x)&(1<<15))>>9);}

#define read16(dst) { dst=(0|((PIOC->PIO_PDSR & (1<<5))>>5)|((PIOC->PIO_PDSR & (1<<4))>>3)|((PIOC->PIO_PDSR & (1<<
3))>>1)|((PIOC->PIO_PDSR & (1<<2))<<1)|((PIOC->PIO_PDSR & (1<<1))<<3)|((PIOD->PIO_PDSR & (1<<10))>>5)\
              |((PIOA->PIO_PDSR & (1<<7))>>1)|((PIOD->PIO_PDSR & (1<<9))>>2)|((PIOB->PIO_PDSR & (1<<26))>>18)|
((PIOA->PIO_PDSR & (3<<14))>>5)|((PIOD->PIO_PDSR & (15<<0))<<11)|((PIOD->PIO_PDSR & (1<<6))<<9));}

#define write8(x)   write16(x & 0xFF)
#define read8(dst)  {read16(dst);uint8_t dst &= 0x0FF;}

#define _tft_width 480
#define _tft_height 320

//#define fontbyte(x) pgm_read_byte(&cfont.font[x])
#define swap(type, i, j) {type t = i; i = j; j = t;}
#define fontbyte(x) cfont.font[x]

#define pgm_read_word(data) *data
#define pgm_read_byte(data) *data


/*
  Source Code of the ETMS_ST7796S class for all constructors and functions that are part of this class.
  Constructor to set ports configuration.
*/

ETMS_ST7796S::ETMS_ST7796S() {
  SetOutputs();
}

void ETMS_ST7796S::SetOutputs(){
  PIOA->PIO_PER = 0x14C080;
```

```
  PIOA->PIO_OER = 0x14C080;
  PIOB->PIO_PER = 0x4000000;
  PIOB->PIO_OER = 0x4000000;
  PIOC->PIO_PER = 0x803FE;
  PIOC->PIO_OER = 0x803FE;
  PIOD->PIO_PER = 0x064F;
  PIOD->PIO_OER = 0x064F;
  PIOC->PIO_SODR = 0x3C0;       //  set the wr,reset,cd,cs pins high - inactive. port pins PC6 - PC9.
  PIOA->PIO_SODR = 0x100000;    //  set the rd pin high - inactive. port pin PA20.
}

void ETMS_ST7796S::Strobe_Write_Pin(){
  PIOC->PIO_CODR = WriteBitMask;   //  - Due port - PC07 mask
  PIOC->PIO_SODR = WriteBitMask;
}

void ETMS_ST7796S::Strobe_Read_Pin(){
  PIOA->PIO_CODR = ReadBitMask;   //  - Due port - PA20 mask
  PIOA->PIO_SODR = ReadBitMask;
}

void ETMS_ST7796S::Writ_Bus8(uint8_t val) {
  write8(val);                //  put 8-bit data/command on bus.
  Strobe_Write_Pin();
}

void ETMS_ST7796S::Writ_Bus16(uint16_t val) {
  write16(val);               //  put 16-bit data on bus.
  Strobe_Write_Pin();
}
/*
uint8_t ETMS_ST7796S::Read_Bus8(){
  PIOC->PIO_CODR |= ChipSelBitMask;   //  LCD_CS=0 pin D43 Due port A20
  Strobe_Read_Pin();
  read8(uint8_t dst);
  PIOC->PIO_SODR |= ChipSelBitMask;
}

uint16_t ETMS_ST7796S::Read_Bus16(){
  PIOC->PIO_CODR |= ChipSelBitMask;   //  LCD_CS=0 pin D43 Due port A20
  Strobe_Read_Pin();
  read16(uint16_t dst);
  PIOC->PIO_SODR |= ChipSelBitMask;
}
*/
void ETMS_ST7796S::Lcd_Dummy_Read() {
  PIOC->PIO_CODR |= ChipSelBitMask;   //  LCD_CS=0 pin D43 Due port A20
  Strobe_Read_Pin();
  PIOC->PIO_SODR |= ChipSelBitMask;
}

void ETMS_ST7796S::Lcd_Write_Com(uint8_t com8) {
  PIOC->PIO_CODR |= ChipSelBitMask;  //  LCD_CS low
  PIOC->PIO_CODR |= RegSelBitMask;   //  LCD_CD=0 pin D38 Due port C6 cd=0 command
  Writ_Bus8(com8);
  PIOC->PIO_SODR |= ChipSelBitMask;  // CS high
}

void ETMS_ST7796S::Lcd_Write_Data8(uint8_t dat8) {
  PIOC->PIO_CODR |= ChipSelBitMask;      //  LCD_CS low
  PIOC->PIO_SODR |= RegSelBitMask;       //  LCD_CD=1 pin D38 Due port PC6 cd=1 data
  Writ_Bus8(dat8);                       //  write to bus
  PIOC->PIO_SODR |= ChipSelBitMask;      //  LCD_CS high
}

void ETMS_ST7796S::Lcd_Write_Data16(uint16_t dat16) {
  PIOC->PIO_CODR |= ChipSelBitMask;      //  LCD_CS LOW
  PIOC->PIO_SODR |= RegSelBitMask;       //  LCD_CD=1 pin D38 Due port PC6 cd=1 data
  Writ_Bus16(dat16);                     //  write to bus D0-D15
  PIOC->PIO_SODR |= ChipSelBitMask;      //  LCD_CS HIGH
```

```
}

void ETMS_ST7796S::Convert_Float(char *buf, double num, int width, byte prec){
   char format[10];

   sprintf(format, "%%%i.%if", width, prec);
  Data access control default values top-left to bottom-right - 0x68 - UTFT 0x48 other
  Lcd_Write_Com_Data(COLMOD, RGB565);                          //  16-bit bus 16-bit 565colour  result - 1 transfer per
pixel. 153.6 kWORD per screen
  Lcd_Write_Com_Data(IFMODE, 0x80);                          //  interface control was 0x00

  Lcd_Write_Com_Data(CSCON, 0xC3);       //  set Command control on
  Lcd_Write_Com_Data(CSCON, 0x96);

  Lcd_Write_Com_Data(DFC, 0x20);
  Lcd_Write_Data8(0x02);

  Lcd_Write_Com(BPC);                    //  blanking porch control
  Lcd_Write_Data8(0x02);
  Lcd_Write_Data8(0x03);
  Lcd_Write_Data8(0x00);
  Lcd_Write_Data8(0x04);

  Lcd_Write_Com_Data(FRMCTR1, 0x80);
  Lcd_Write_Data8(0x10);

  Lcd_Write_Com_Data(DIC, 0x00);         //  display inversion control
  Lcd_Write_Com_Data(EM, 0xC6);          //  Entry mode
  Lcd_Write_Com_Data(VCMPCTL, 0x24);
  //Lcd_Write_Com_Data(0xE4, 0x31);      //  unknown command ??
   sprintf(buf, format, num);
}

void ETMS_ST7796S::Lcd_Write_Com_Data(uint8_t com8,uint8_t dat8) {
  Lcd_Write_Com(com8);
  Lcd_Write_Data8(dat8);
}

void ETMS_ST7796S::init(boolean orientation) {
  PIOC->PIO_CODR |= ResetBitMask;  //display RESET low
  delay(20);
  PIOC->PIO_SODR |= ResetBitMask;  //display RESET high
  delay(120);
  //    the chip initialization - sw reset.
  Lcd_Write_Com(SWRESET);
  delay(50);

  Lcd_Write_Data8(0x13);
  Lcd_Write_Data8(0x0C);
  Lcd_Write_Data8(0x0D);
  Lcd_Write_Data8(0x27);
  Lcd_Write_Data8(0x3B);
  Lcd_Write_Data8(0x44);
  Lcd_Write_Data8(0x4D);
  Lcd_Write_Data8(0x0B);
  Lcd_Write_Data8(0x17);
  Lcd_Write_Data8(0x17);
  Lcd_Write_Data8(0x1D);
  Lcd_Write_Data8(0x21);

  Lcd_Write_Com_Data(CSCON, 0x3C);   //  set command control off
  Lcd_Write_Com_Data(CSCON, 0x69);
  delay(120);             //  wait a sec (actually 120mSec)
  Lcd_Write_Com(NORON);   //  normal ON
```

```
 Lcd_Write_Com(SLPOUT);    //  sleep off
 Lcd_Write_Com(DISPON);    //  display on
 Lcd_Write_Com(SLPOUT);              //  sleep off
 delay(120);

 Lcd_Write_Com_Data(MADCTL, MADCTL_MV |MADCTL_MX|MADCTL_MY | MADCTL_RGB);   //  Memory
 Lcd_Write_Com(DOCA);
 Lcd_Write_Data8(0x40);
 Lcd_Write_Data8(0x8A);
 Lcd_Write_Data8(0x00);
 Lcd_Write_Data8(0x00);
 Lcd_Write_Data8(0x29);
 Lcd_Write_Data8(0x19);
 Lcd_Write_Data8(0xA5);
 Lcd_Write_Data8(0x33);

 Lcd_Write_Com(PWR1);
 Lcd_Write_Data8(0x80);
 Lcd_Write_Data8(0x51);

 //Lcd_Write_Com_Data(PWR2, 0x19);

 Lcd_Write_Com_Data(PWR3,0xA7);

 Lcd_Write_Com(PGC);   //  positive gamma
 Lcd_Write_Data8(0xF0);
 Lcd_Write_Data8(0x09);
 Lcd_Write_Data8(0x13);
 Lcd_Write_Data8(0x12);
 Lcd_Write_Data8(0x12);
 Lcd_Write_Data8(0x2B);
 Lcd_Write_Data8(0x3C);
 Lcd_Write_Data8(0x44);
 Lcd_Write_Data8(0x4B);
 Lcd_Write_Data8(0x1B);
 Lcd_Write_Data8(0x18);
 Lcd_Write_Data8(0x17);
 Lcd_Write_Data8(0x1D);
 Lcd_Write_Data8(0x21);

 Lcd_Write_Com(NGC);   //  negative gamma
 Lcd_Write_Data8(0xF0);
 Lcd_Write_Data8(0x09);
 //PIOC->PIO_SODR |= ChipSelBitMask;      //  CS HIGH
 orient = orientation;
 cfont.font = 0;
}

void ETMS_ST7796S::displayOn() {
   Lcd_Write_Com(SLPOUT);
   Lcd_Write_Com(DISPON);
}

void ETMS_ST7796S::displayOff() {
   Lcd_Write_Com(SLPIN);
   Lcd_Write_Com(DISPOFF);
}

void ETMS_ST7796S::setBrightness(uint8_t br) {
   Lcd_Write_Com_Data(WRDISBV, br);
}

uint8_t ETMS_ST7796S::getID(){

}
uint16_t ETMS_ST7796S::getHeight(){
return 320;
}
uint16_t ETMS_ST7796S::getDisplayYSize(){
return 320;
```

```
}
uint16_t ETMS_ST7796S::getWidth(){
return 480;
}
uint16_t ETMS_ST7796S::getDisplayXSize(){
return 480;
}
/*
  First set colour then write 16 bits
*/
uint16_t ETMS_ST7796S::to565Colour(uint8_t red, uint8_t green, uint8_t blue){
  uint16_t b = (blue >> 3) & 0x1f;
  uint16_t g = ((green >> 2) & 0x3f) << 5;
  uint16_t r = ((red >> 3) & 0x1f) << 11;
  _colour565 = (r | g | b);
  return _colour565;
}

// write a 16bit 565 colour pixel to frame buffer, if no parameters then uses last colour selected.
void ETMS_ST7796S::setPixel(){
  Lcd_Write_Data16(_colour565);      // rrrrrggg gggbbbbb
}
void ETMS_ST7796S::setPixel(uint16_t colour){
  Lcd_Write_Data16(colour);      // rrrrrggg gggbbbbb
}
void ETMS_ST7796S::setPixel(uint8_t red, uint8_t green, uint8_t blue){
  //_colour565 = ((red&248)<<8 | (green&252)<<3 | (blue&248)>>3);
  uint16_t pcolour565 = to565Colour(red,green,blue);
  Lcd_Write_Data16(pcolour565);      // rrrrrggg gggbbbbb
}

/*
  invert display - needs clarifying.
*/
void ETMS_ST7796S::invertDisplay(uint8_t inv) {
    Lcd_Write_Com_Data(DIC,inv);
    Lcd_Write_Com_Data(DIC,inv);
}
void ETMS_ST7796S::setTransparency(bool t) {
    _TRANSPARENT = t;
}
/*
  This is the same as the setxy routine found in most libraries. Sets the display address window boundaries. Normally
  portrait or landscape would affect this function, but only landscape is supported by this library (V1.0) for now.
*/
void ETMS_ST7796S::setAddressWindow(uint16_t x1, uint16_t y1, uint16_t w, uint16_t h) {

    uint16_t x2 = x1 + w;       //garranties x2 > x1
    uint16_t y2 = y1 + h;
  /*
    if (orient==LANDSCAPE)
    {
       swap(uint16_t, x1, y1);
       swap(uint16_t, x2, y2);
       y1=_tft_height-y1;
       y2=_tft_height-y2;
       swap(uint16_t, y1, y2);
    }
*/

    Lcd_Write_Com(CASET);
    Lcd_Write_Data8(x1>>8);
    Lcd_Write_Data8(x1);
    Lcd_Write_Data8(x2>>8);
    Lcd_Write_Data8(x2);
    Lcd_Write_Com(RASET);
    Lcd_Write_Data8(y1>>8);
    Lcd_Write_Data8(y1);
    Lcd_Write_Data8(y2>>8);
    Lcd_Write_Data8(y2);
```

```
   Lcd_Write_Com(RAMWR);
}
void ETMS_ST7796S::setXY(uint16_t x1, uint16_t y1, uint16_t w, uint16_t h){
   setAddressWindow( x1, y1, w, h);
}
void ETMS_ST7796S::clrXY()
{
   if (orient==LANDSCAPE)
      setAddressWindow(0,0,_tft_height-1,_tft_width-1);
   else
      setAddressWindow(0,0,_tft_width-1,_tft_height-1);
}

void ETMS_ST7796S::fillWindow(uint16_t colour, int length) {
    _colour565 = colour;
 Lcd_Write_Com(RAMWRC);
for (int i = 0; i <= length; i++) {
  Lcd_Write_Data16(_colour565);
  }
}
void ETMS_ST7796S::fastFill(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height){
   int length = width * height;
   Lcd_Write_Com(RAMWRC);
for (int i = 0; i <= length; i++) {
   Lcd_Write_Data16(_colour565);
  }
}
void ETMS_ST7796S::fillScreen(){
   int len = _tft_height*_tft_width;
   setAddressWindow(0, 0, _tft_width-1, _tft_height-1);
   fillWindow(_colour565, len);
}
void ETMS_ST7796S::fillScreen(uint16_t colour){
   int len = _tft_height*_tft_width;
   setAddressWindow(0, 0, _tft_width-1, _tft_height-1);
   fillWindow(colour, len);
}
void ETMS_ST7796S::fillScreen(uint8_t red, uint8_t green, uint8_t blue){
   _colour565 = to565Colour(red,green,blue);
   fillScreen(_colour565);
}
void ETMS_ST7796S::clrDisplay() {
   fillScreen(BLACK);
}
void ETMS_ST7796S::clrScr(){
   clrDisplay();
}

void ETMS_ST7796S::drawPixel(uint16_t xpos, uint16_t ypos) {
   // out of bounds check
   if ((xpos < 0) || (ypos < 0) || (xpos >= _tft_width) || (ypos >= _tft_height))
   return;
   // all good then draw a pixel with RGB565 here
   setAddressWindow(xpos, ypos, xpos, ypos);
   setPixel(_colour565);
}
void ETMS_ST7796S::drawPixel(uint16_t xpos, uint16_t ypos, uint16_t fcolour) {
   _colour565 = fcolour;
   drawPixel(xpos,ypos);
}
void ETMS_ST7796S::drawPixel(uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue) {
   _colour565 = to565Colour(red,green,blue);    //    to565Colour uses private variable - '_colour565'
   drawPixel(xpos,ypos);
}

/*
set foreground and background colours
*/
void ETMS_ST7796S::setColour(uint16_t colour){
  _colour565 = colour;
```

```
}
void ETMS_ST7796S::setColour(uint8_t red, uint8_t green, uint8_t blue){
 _colour565 = to565Colour(red, green, blue);
}

void ETMS_ST7796S::setBackColour(uint16_t colour){
  _backcolour565 = colour;
  _TRANSPARENT = false;
}
void ETMS_ST7796S::setBackColour(uint8_t red, uint8_t green, uint8_t blue){
  _backcolour565 = to565Colour(red, green, blue);
 _TRANSPARENT = false;
}
void ETMS_ST7796S::setFgBgColour(uint16_t fcolour,uint16_t bcolour){
  _colour565 = fcolour;
  _backcolour565 = bcolour;
  _TRANSPARENT = false;
}
/*
  Basic Graphic Primitives  ************************************************************
*/
// Draw a line..
void ETMS_ST7796S::drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2){
uint16_t steep = abs(y2 - y1) > abs(x2 - x1);
    if (steep)
    {
       swap(uint16_t, x1, y1);
       swap(uint16_t, x2, y2);
    }
    if (x1 > x2)
    {
       swap(uint16_t, x1, x2);
       swap(uint16_t, y1, y2);
    }

    int16_t dx, dy;
    dx = x2 - x1;
    dy = abs(y2 - y1);

    int16_t err = dx / 2;
    int16_t ystep;

    if (y1 < y2)
    {
       ystep = 1;
    }
    else
    {
       ystep = -1;
    }

    for (; x1<=x2; x1++)
    {
       if (steep)
       {
          drawPixel(y1, x1);
       }
       else
       {
          drawPixel(x1, y1);
       }
       err -= dy;
       if (err < 0)
       {
          y1 += ystep;
          err += dx;
       }
    }
}
void ETMS_ST7796S::drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t colour){
```

```
  _colour565 = colour;
  drawLine(x1, y1, x2, y2);
}
void ETMS_ST7796S::drawLine(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t red, uint8_t green, uint8_t blue){
  _colour565 = to565Colour( red, green, blue);
  drawLine(x1, y1, x2, y2);
}
//  Horizontal Line (FastLine method)
void ETMS_ST7796S::drawHLine(uint16_t xs,uint16_t ys ,uint16_t length) {
    //  draw line here,  do loop to add pixel to the x line
     for (int i = 0; i <= length; i++) {
    drawPixel(xs+i, ys, _colour565);
  }
}
void ETMS_ST7796S::drawHLine(uint16_t xs,uint16_t ys ,uint16_t length, uint16_t colour) {
    _colour565 = colour;
    drawHLine(xs,ys,length);
}
void ETMS_ST7796S::drawHLine(uint16_t xs,uint16_t ys ,uint16_t length, uint8_t red, uint8_t green, uint8_t blue) {
    _colour565 = to565Colour(red,green,blue);
    drawHLine(xs,ys,length);
}
//  Vertical Line (FastLine method)
void ETMS_ST7796S::drawVLine(uint16_t xs, uint16_t ys, uint16_t length) {
    //  draw line here, do loop
    for (int i = 0; i <= length; i++) {
    drawPixel(xs, ys+i, _colour565);
  }
}
void ETMS_ST7796S::drawVLine(uint16_t xs, uint16_t ys, uint16_t length, uint16_t colour) {
    _colour565 = colour;
    drawVLine(xs,ys,length);

}
void ETMS_ST7796S::drawVLine(uint16_t xs, uint16_t ys, uint16_t length, uint8_t red, uint8_t green, uint8_t blue) {
    _colour565 = to565Colour(red,green,blue);
    drawVLine(xs,ys,length);
}
/*
  Advanced Graphics.
  note: world coordinate system used - TOP-LEFT is origin (0,0).
  Let's start with the rectangles;open,filled,rounded.
  Then the Circles and Triangles.
  Asperations on fancy guages elements next...
*/
void ETMS_ST7796S::drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height){
    //xs = _tft_width - xs;
    drawHLine( xs, ys, width);
    drawHLine( xs, ys+height, width);
    drawVLine( xs, ys, height);
    drawVLine( xs+width, ys, height);
}
void ETMS_ST7796S::drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour){
    _colour565 = colour;
    drawRect( xs, ys, width, height);
}
void ETMS_ST7796S::drawRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blue){
    _colour565 = to565Colour( red, green, blue);
    drawRect( xs, ys, width, height);
}

void ETMS_ST7796S::fillRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height){
for (int j = 0; j <= height; j++){
    for (int i = 0; i <= width; i++) {
    drawPixel(xs+i, ys+j, _colour565);
    }
  }
}
void ETMS_ST7796S::fillRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour){
    _colour565 = colour;
```

```
    fillRect( xs, ys, width, height);
}
void ETMS_ST7796S::fillRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blue){
    _colour565 = to565Colour(red,green,blue);
    fillRect( xs, ys, width, height);
}

void ETMS_ST7796S::drawRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height){
if (xs>width)
    {
      swap(uint16_t, xs, width);
    }
    if (ys>height)
    {
      swap(uint16_t, ys, height);
    }
    if ((width-xs)>4 && (height-ys)>4)
    {
      drawPixel(xs+1,ys+1,_colour565);
      drawPixel(width-1,ys+1,_colour565);
      drawPixel(xs+1,height-1,_colour565);
      drawPixel(width-1,height-1,_colour565);
      drawHLine(xs+2, ys, width-xs-4,_colour565);
      drawHLine(xs+2, height, width-xs-4,_colour565);
      drawVLine(xs, ys+2, height-ys-4,_colour565);
      drawVLine(width, ys+2, height-ys-4,_colour565);
    }
}
void ETMS_ST7796S::drawRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour){
    _colour565 = colour;
    drawRoundRect( xs,  ys,  width,  height);
}
void ETMS_ST7796S::drawRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t b
lue){
    _colour565 = to565Colour(red,green,blue);
  drawRoundRect(xs, ys, width, height, _colour565);
}

void ETMS_ST7796S::fillRoundRect(uint16_t x1, uint16_t y1, uint16_t width, uint16_t height){
    uint16_t x2 = x1 + width;
    uint16_t y2 = y1 + height;

    if ((x2-x1)>4 && (y2-y1)>4)
    {
      for (int i=0; i<((y2-y1)/2)+1; i++)
      {
        switch(i)
        {
        case 0:
          drawHLine(x1+2, y1+i, x2-x1-4);
          drawHLine(x1+2, y2-i, x2-x1-4);
          break;
        case 1:
          drawHLine(x1+1, y1+i, x2-x1-2);
          drawHLine(x1+1, y2-i, x2-x1-2);
          break;
        default:
          drawHLine(x1, y1+i, x2-x1);
          drawHLine(x1, y2-i, x2-x1);
        }
      }
    }

}
void ETMS_ST7796S::fillRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint16_t colour){
    _colour565 = colour;
    fillRoundRect(xs,ys,width,height);
}
void ETMS_ST7796S::fillRoundRect(uint16_t xs, uint16_t ys, uint16_t width, uint16_t height, uint8_t red, uint8_t green, uint8_t blu
e){
```

```
    _colour565 = to565Colour(red,green,blue);
    fillRoundRect(xs,ys,width,height);
}

//  Circles

void ETMS_ST7796S::drawCircle(uint16_t xs, uint16_t ys, uint16_t radius)
{
    //xs = _tft_width - xs;
    int f = 1 - radius;
    int ddF_x = 1;
    int ddF_y = -2 * radius;
    int x1 = 0;
    int y1 = radius;

    setAddressWindow(xs, ys + radius, xs, ys + radius);
    Lcd_Write_Data16(_colour565);
    setAddressWindow(xs, ys - radius, xs, ys - radius);
    Lcd_Write_Data16(_colour565);
    setAddressWindow(xs + radius, ys, xs + radius, ys);
    Lcd_Write_Data16(_colour565);
    setAddressWindow(xs - radius, ys, xs - radius, ys);
    Lcd_Write_Data16(_colour565);

    while(x1 < y1)
    {
        if(f >= 0)
        {
            y1--;
            ddF_y += 2;
            f += ddF_y;
        }
        x1++;
        ddF_x += 2;
        f += ddF_x;
        setAddressWindow(xs + x1, ys + y1, xs + x1, ys + y1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs - x1, ys + y1, xs - x1, ys + y1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs + x1, ys - y1, xs + x1, ys - y1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs - x1, ys - y1, xs - x1, ys - y1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs + y1, ys + x1, xs + y1, ys + x1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs - y1, ys + x1, xs - y1, ys + x1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs + y1, ys - x1, xs + y1, ys - x1);
        Lcd_Write_Data16(_colour565);
        setAddressWindow(xs - y1, ys - x1, xs - y1, ys - x1);
        Lcd_Write_Data16(_colour565);
    }
    clrXY();
}
void ETMS_ST7796S::drawCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t colour){
    _colour565 = colour;
    drawCircle( xs, ys, radius);
}
void ETMS_ST7796S::drawCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t lw, uint8_t red, uint8_t green, uint8_t blue){
    _colour565 = to565Colour(red,green,blue);
    drawCircle( xs, ys, radius);
}

void ETMS_ST7796S::fillCircle(uint16_t xs, uint16_t ys, uint16_t radius){

  for(long y1=-radius; y1<=0; y1++)
      for(long x1=-radius; x1<=0; x1++)
          if(x1*x1+y1*y1 <= radius*radius)
          {
              drawHLine(xs+x1, ys+y1, 2*(-x1));
```

```
            drawHLine(xs+x1, ys-y1, 2*(-x1));
            break;
          }
}
void ETMS_ST7796S::fillCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint16_t colour){
_colour565 = colour;
 fillCircle(xs, ys, radius);
}
void ETMS_ST7796S::fillCircle(uint16_t xs, uint16_t ys, uint16_t radius, uint8_t red, uint8_t green, uint8_t blue){
  _colour565 = to565Colour(red,green,blue);
 fillCircle(xs, ys, radius);
}
//  Arc
void ETMS_ST7796S::drawArc(uint16_t x, uint16_t y, uint16_t r, uint16_t start, uint16_t stop, uint16_t colour){

}

//  Triangles

void ETMS_ST7796S::drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2){
   drawLine(x0, y0, x1, y1);
   drawLine(x1, y1, x2, y2);
   drawLine(x2, y2, x0, y0);
}
void ETMS_ST7796S::drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2, uint16_t colour){
   _colour565 = colour;
   drawTriangle(x0,y0,x1,y1,x2,y2);
}
void ETMS_ST7796S::drawTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1,uint16_t x2, uint16_t y2, uint8_t red, uint8_t
green, uint8_t blue){
   _colour565 = to565Colour(red,green,blue);
   drawTriangle(x0, y0,  x1,  y1, x2,  y2, _colour565);
}
// tbd
void ETMS_ST7796S::fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2) {

 uint16_t a, b, y, last;

 // Sort coordinates by Y order (y2 >= y1 >= y0)
 if (y0 > y1) {
  swap(uint16_t,y0, y1);
  swap(uint16_t,x0, x1);
 }
 if (y1 > y2) {
  swap(uint16_t,y2, y1);
  swap(uint16_t,x2, x1);
 }
 if (y0 > y1) {
  swap(uint16_t,y0, y1);
  swap(uint16_t,x0, x1);
 }

 //startWrite();
 if (y0 == y2) { // Handle awkward all-on-same-line case as its own thing
  a = b = x0;
  if (x1 < a)
   a = x1;
  else if (x1 > b)
   b = x1;
  if (x2 < a)
   a = x2;
  else if (x2 > b)
   b = x2;
  drawHLine(a, y0, b - a + 1, _colour565);
  //endWrite();
  return;
 }

 int16_t dx01 = x1 - x0, dy01 = y1 - y0, dx02 = x2 - x0, dy02 = y2 - y0,
       dx12 = x2 - x1, dy12 = y2 - y1;
```

```
  int32_t sa = 0, sb = 0;

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

  for (y = y0; y <= last; y++) {
    a = x0 + sa / dy01;
    b = x0 + sb / dy02;
    sa += dx01;
    sb += dx02;
    /* longhand:
    a = x0 + (x1 - x0) * (y - y0) / (y1 - y0);
    b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
    */
    if (a > b)
      swap(uint16_t,a, b);
    drawHLine(a, y, b - a + 1, _colour565);
  }

  // For lower part of triangle, find scanline crossings for segments
  // 0-2 and 1-2.  This loop is skipped if y1=y2.
  sa = (uint32_t)dx12 * (y - y1);
  sb = (uint32_t)dx02 * (y - y0);
  for (; y <= y2; y++) {
    a = x1 + sa / dy12;
    b = x0 + sb / dy02;
    sa += dx12;
    sb += dx02;
    /* longhand:
    a = x1 + (x2 - x1) * (y - y1) / (y2 - y1);
    b = x0 + (x2 - x0) * (y - y0) / (y2 - y0);
    */
    if (a > b)
      swap(uint16_t, a, b);
    drawHLine(a, y, b - a + 1, _colour565);
  }
  clrXY();
}
void ETMS_ST7796S::fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t colour) {
    _colour565 = colour;
    fillTriangle(x0,y0,x1,y1,x2,y2);
}
void ETMS_ST7796S::fillTriangle(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint8_t red, uint8_t g
reen, uint8_t blue) {
    _colour565 = to565Colour(red,green,blue);
    fillTriangle(x0,y0,x1,y1,x2,y2);
}

/*
  Everything Text...

  set cursor - 0,0 is top left!
*/
void ETMS_ST7796S::setCursor(uint16_t xpos, uint16_t ypos){
  _CurrentW = cfont.x_size-1;   //  start with these values for the initial font set.
  _CurrentH = cfont.y_size-1;
  _CurrentX = xpos;
  _CurrentY = ypos;
  setAddressWindow(_CurrentX, _CurrentY, _CurrentW, _CurrentH);
}
/*
  print a character from the currently selected font array
```

```
*/

void ETMS_ST7796S::drawChar(byte c, uint16_t x, uint16_t y){
  byte i,ch;
  word j;
  word temp;
  //x = _tft_width - x;
  if (!_TRANSPARENT)
  {
    if (orient==PORTRAIT)  //  is portrait orientaton
    {
      setAddressWindow(x,y,cfont.x_size-1,cfont.y_size-1);
      temp=((c-cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
      for(j=0;j<((cfont.x_size/8)*cfont.y_size);j++)
      {
        ch=pgm_read_byte(&cfont.font[temp]);
        for(i=0;i<8;i++)
        {
          if((ch&(1<<(7-i)))!=0)
          {
            setPixel(_colour565);
          }
          else
          {
            setPixel(_backcolour565);
          }
        }
        temp++;
      }
    }
    else   //  is landscape orientation and NOT transparent!
    {
      temp=((c-cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
      for(j=0;j<cfont.y_size;j++)
      {
        //setAddressWindow(x,y+(j/(cfont.x_size/8)),x+cfont.x_size-1,y+(j/(cfont.x_size/8)));

        for (int zz=0; zz<(cfont.x_size/8); zz++)
        {
          ch=pgm_read_byte(&cfont.font[temp+zz]);
          for(i=0;i<8;i++)
          {
            setAddressWindow(x+i+(zz*8),y+j,x+i+(zz*8)+1,y+j+1);
            if((ch&(1<<(7-i)))!=0)
            {
              setPixel(_colour565);
            }
            else
            {
              //setAddressWindow(x+i+(zz*8),y+j,x+i+(zz*8)+1,y+j+1);
              setPixel(_backcolour565);
            }
          }
        }
        temp+=(cfont.x_size/8);
      }
    }
    clrXY();
  }
  else       //  is transparent and LANDSCAPE
  {
    temp=((c-cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
    for(j=0;j<cfont.y_size;j++)
    {
      for (int zz=0; zz<(cfont.x_size/8); zz++)
      {
        ch=pgm_read_byte(&cfont.font[temp+zz]);
        for(i=0;i<8;i++)
        {
          if((ch&(1<<(7-i)))!=0)
```

```
                {
                  setAddressWindow(x+i+(zz*8),y+j,x+i+(zz*8)+1,y+j+1);
                  setPixel(_colour565);
                }

            }
          }
          temp+=(cfont.x_size/8);
        }
    }
    clrXY();
}
void ETMS_ST7796S::drawChar(uint8_t c, uint16_t xpos, uint16_t ypos, uint16_t colour){
  _colour565 = colour;
 drawChar(c,xpos,ypos);
}
void ETMS_ST7796S::drawChar(uint8_t c, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue){
_colour565 = to565Colour(red,green,blue);  //  converts RGB888 and loads to _colour565
drawChar(c,xpos,ypos);
}

void ETMS_ST7796S::rotateChar(byte c, int x, int y, int pos, int deg){
    byte i,j,ch;
    word temp;
    int newx,newy;
    double radian;
    radian=deg*0.0175;

    temp=((c-cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
    for(j=0;j<cfont.y_size;j++)
    {
      for (int zz=0; zz<(cfont.x_size/8); zz++)
      {
        ch=pgm_read_byte(&cfont.font[temp+zz]);
        for(i=0;i<8;i++)
        {
          newx=x+(((i+(zz*8)+(pos*cfont.x_size))*cos(radian))-((j)*sin(radian)));
          newy=y+(((j)*cos(radian))+((i+(zz*8)+(pos*cfont.x_size))*sin(radian)));

          setAddressWindow(newx,newy,newx+1,newy+1);

          if((ch&(1<<(7-i)))!=0)
          {
            setPixel(_colour565);
          }
          else
          {
            if (!_TRANSPARENT)
              setPixel(_backcolour565);
          }
        }
      }
      temp+=(cfont.x_size/8);
    }
    clrXY();
}
// This is to print numbers
void ETMS_ST7796S::print(char *st, int x, int y, int deg){
    int stl, i;
    stl = strlen(st);

    if (orient==PORTRAIT)
    {
    if (x==RIGHT)
      x=(disp_x_size+1)-(stl*cfont.x_size-1);
    if (x==CENTER)
      x=((disp_x_size+1)-(stl*cfont.x_size-1))/2;
    }
    else          //          *****          is landscape orientation
    {
```

```
  if (x==RIGHT)
    x=(disp_y_size+1)-(stl*cfont.x_size-1);
  if (x==CENTER)
    x=((disp_y_size+1)-(stl*cfont.x_size-1))/2;
  }

  for (i=0; i<stl; i++)
    if (deg==0)
      drawChar(*st++, x + (i*(cfont.x_size-1)), y);
    else
      rotateChar(*st++, x, y, i, deg);
}
void ETMS_ST7796S::print(String st, int x, int y, int deg)
{
  char buf[st.length()+1];
  st.toCharArray(buf, st.length()+1);
  print(buf, x, y, deg);
}

void ETMS_ST7796S::printString(String x, uint16_t xpos, uint16_t ypos){
  uint16_t csp = cfont.x_size;
  //  drawChar in a loop for the length of string x
  for (int i = 0; i <= x.length()-1; i++) {drawChar( x[i], xpos + (i * csp), ypos);}
}
void ETMS_ST7796S::printString(String x, uint16_t xpos, uint16_t ypos, uint16_t colour){
  _colour565 = colour;
  printString(x,xpos,ypos);
}
void ETMS_ST7796S::printString(String x, uint16_t xpos, uint16_t ypos, uint8_t red, uint8_t green, uint8_t blue){
  _colour565 = to565Colour(red,green,blue);
  printString(x,xpos,ypos);
}
//  Numbers
void ETMS_ST7796S::printNumI(long num, uint16_t x, uint16_t y, int length, char filler)
{
  char buf[25];
  char st[27];
  boolean neg=false;
  int c=0, f=0;

  if (num==0)
  {
    if (length!=0)
    {
      for (c=0; c<(length-1); c++)
        st[c]=filler;
      st[c]=48;
      st[c+1]=0;
    }
    else
    {
      st[0]=48;
      st[1]=0;
    }
  }
  else
  {
    if (num<0)
    {
      neg=true;
      num=-num;
    }

    while (num>0)
    {
      buf[c]=48+(num % 10);
      c++;
      num=(num-(num % 10))/10;
    }
    buf[c]=0;
```

```
    if (neg)
    {
      st[0]=45;
    }

    if (length>(c+neg))
    {
      for (int i=0; i<(length-c-neg); i++)
      {
        st[i+neg]=filler;
        f++;
      }
    }

    for (int i=0; i<c; i++)
    {
      st[i+neg+f]=buf[c-i-1];
    }
    st[c+neg+f]=0;

  }

  print(st,x,y);
}

void ETMS_ST7796S::printNumF(double num, uint16_t x, uint16_t y, byte dec, char divider, int length, char filler)
{
  char st[27];
  boolean neg=false;

  if (dec<1)
    dec=1;
  else if (dec>20)
    dec=20;

  if (num<0)
    neg = true;

  Convert_Float(st, num, length, dec);

  if (divider != '.')
  {
    for (int i=0; i<sizeof(st); i++)
      if (st[i]=='.')
        st[i]=divider;
  }

  if (filler != ' ')
  {
    if (neg)
    {
      st[0]='-';
      for (int i=1; i<sizeof(st); i++)
        if ((st[i]==' ') || (st[i]=='-'))
          st[i]=filler;
    }
    else
    {
      for (int i=0; i<sizeof(st); i++)
        if (st[i]==' ')
          st[i]=filler;
    }
  }

  print(st,x,y);
}
// Images
void ETMS_ST7796S::drawBitMap(int x, int y, int sx, int sy, bitmapdatatype data, int scale)
{
```

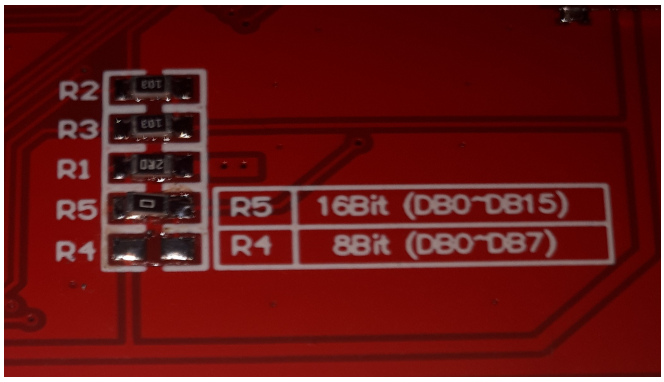
```
  unsigned int col;
  int tx, ty, tc, tsx, tsy;

  if (scale==1)
  {
    if (orient==PORTRAIT)
    {
      setXY(x, y, x+sx-1, y+sy-1);
      for (tc=0; tc<(sx*sy); tc++)
      {
        col=pgm_read_word(&data[tc]);
        setPixel(col);
        //LCD_Write_DATA(col>>8,col & 0xff);
      }
    }
    else    // is landscape
    {
      for (ty=0; ty<sy; ty++)
      {
        setXY(x, y+ty, x+sx-1, y+ty);
        for (tx=sx-1; tx>=0; tx--)
        {
          col=pgm_read_word(&data[(ty*sx)+tx]);
          setPixel(col);
          //LCD_Write_DATA(col>>8,col & 0xff);
        }
      }
    }
  }
  else
  {
    if (orient==PORTRAIT)
    {
      for (ty=0; ty<sy; ty++)
      {
        setXY(x, y+(ty*scale), x+((sx*scale)-1), y+(ty*scale)+scale);
        for (tsy=0; tsy<scale; tsy++)
          for (tx=0; tx<sx; tx++)
          {
            col=pgm_read_word(&data[(ty*sx)+tx]);
            for (tsx=0; tsx<scale; tsx++)
              setPixel(col);
              //LCD_Write_DATA(col>>8,col & 0xff);
          }
      }
    }
    else
    {
      for (ty=0; ty<sy; ty++)
      {
        for (tsy=0; tsy<scale; tsy++)
        {
          setXY(x, y+(ty*scale)+tsy, x+((sx*scale)-1), y+(ty*scale)+tsy);
          for (tx=sx-1; tx>=0; tx--)
          {
            col=pgm_read_word(&data[(ty*sx)+tx]);
            for (tsx=0; tsx<scale; tsx++)
            setPixel(col);
            //  LCD_Write_DATA(col>>8,col & 0xff);
          }
        }
      }
    }
  }
  clrXY();
}
void ETMS_ST7796S::drawBitMap(int x, int y, int sx, int sy, bitmapdatatype data, int deg, int rox, int roy)
{
  unsigned int col;
  int tx, ty, newx, newy;
```

```
    double radian;
    radian=deg*0.0175;

    if (deg==0)
      drawBitMap(x, y, sx, sy, data);
    else
    {
      for (ty=0; ty<sy; ty++)
        for (tx=0; tx<sx; tx++)
        {
          col=pgm_read_word(&data[(ty*sx)+tx]);
          newx=x+rox+(((tx-rox)*cos(radian))-((ty-roy)*sin(radian)));
          newy=y+roy+(((ty-roy)*cos(radian))+((tx-rox)*sin(radian)));
          setXY(newx, newy, newx, newy);
          setPixel(col);
          //LCD_Write_DATA(col>>8,col & 0xff);
        }
    }
    clrXY();
}

void ETMS_ST7796S::setFont(uint8_t* font){
    cfont.font = font;
    cfont.x_size=fontbyte(0);
    cfont.y_size=fontbyte(1);
    cfont.offset=fontbyte(2);
    cfont.numchars=fontbyte(3);
}

uint8_t* ETMS_ST7796S::getFont(){
    return cfont.font;
}

uint8_t ETMS_ST7796S::getFontXsize(){
    return cfont.x_size;
}

uint8_t ETMS_ST7796S::getFontYsize(){
    return cfont.y_size;
}
```

SECTION 3

Hardware Info.

This board uses LS245 octal bus transceivers to isolate and level shift the bus of the Mega2560 which operates at 5 volts from the onboard logic which operates at 3.3V. Now this is convenient because the Arduino Due has a maximum bus voltage of 3.3 volts. So you can safely plug this into a Due, the bus transceivers will level translate correctly.



In order for this library to work correctly, this board has to be configured to 16-bit mode. This is accomplished by removing jumper/0 ohm resistor from the R4 position and placing it at the R5 position.

Pin mapping for the Arduino Due and Mega2560.

SHIELD FUNCTION	DIGITAL PIN	16bit BUS	due	mega2560
	0		RX0	
	1		TX0	
	2		PB25	
	3		PC28	
	4		PA29/PC26	
	5		PC25	
	6		PC24	
	7		PC23	
	8		PC22	14C080
	9		PC21	
	10		PA28/PC26	4000000
	11		PD7	
	12		PD8	803FE
LED	13		B27	
	TX3		D4	064F
	RX3		D5	
	TX2		PA13	
	RX2		PA12	
	TX1		PA11	
	RX1		PA10	
	SDA		PB12	
	SCL		PB13	
DB8	22	b8	B26	PA0
DB9	23	b9	A14	PA1
DB10	24	b10	A15	PA2
DB11	25	b11	D0	PA3
DB12	26	b12	D1	PA4
DB13	27	b13	D2	PA5
DB14	28	b14	D3	PA6
DB15	29	b15	D6	PA7
DB7	30	b7	D9	PC7
DB6	31	b6	A7	PC6
DB5	32	b5	D10	PC5
DB4	33	b4	C1	PC4
DB3	34	b3	C2	PC3
DB2	35	b2	C3	PC2
DB1	36	b1	C4	PC1
DB0	37	b0	C5	PC0
RS	38		C6	PD7
WR	39		C7	PG2
CS	40		C8	PG1
RST	41		C9	PG0
NC	42		A19	PL7
RD	43		A20	PL6
TP_IRQ	44		C19	PL5
NC	45		C18	PL4
NC	46		C17	PL3
NC	47		C16	PL2
SD_CS	48		C15	PL1
NC	49		C14	PL0
MISO	50		C13	PB3
MOSI	51		C12	PB2
SPI_CLK	52		B21	PB1
TP_CS	53		B14	PB0
	A0		PA16	
	A1		PA24	
	A2		PA23	
	A3		PA22	
	A4		PA6	
	A5		PA4	
	A6		PA3	
	A7		PA22	
	A8		PB17	
	A9		PB18	
	A10		PB19	
	A11		PB20	
	DAC0		PB15	
	DAC1		PB16	
	CANRX		PA10	
	CANTX		PA0	
	SDA1		PA17	
	SCL1		PA18	
LED_RX			PC30	
LED_TX			PA21	
MISO			PA25	
MOSI			PA26	
SCLK			PA27	

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	16bit port	
D6	D3	D2	D1	D0	A15	A14	B26	D9	A7	D10	C1	C2	C3	C4	C5	DUE	
PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	MEGA	
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	APORT-HI	32-16
1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	APORT-LOW	15-0
0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	BPORT-HI	32-16
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BPORT-LOW	15-0
												1	0	0	0	CPORT-HI	32-16
0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	CPORT-LOW	15-0
																DPORT-HI	32-16
0	0	0	0	0	1	1	0	0	1	0	0	1	1	1	1	DPORT-LOW	15-0

36
PIN
HEADER

SECTION 4

Examples Description.

This page is intentionally blank.

SECTION 5

Introduction to Graphics.

This page is intentionally blank.

SECTION 6

Misc.

This page is intentionally blank.

DISCLAIMER

The information in this document is designed to provide helpful information on the operations of the above mentioned hardware and software components. Although all attempts to ensure that the information is accurate, it is presented "as is" without warranty of any kind. This is not meant to replace the applicable manufacturers instructions on the operation of the hardware or software in question. This software library has been tested in a limited capacity and will only operate on the tested components, including any modifications mentioned above. Some of the above procedures require a certain level of skill, if you should require assistance in this configuration issue, please contact someone with very good electronic/soldering skills to help. I will not be responsible for tort or damaged property, consequential or otherwise.

Permissions Coordinator:

email: nash4unow@hotmail.com