Driving 3.5inch 480x320 ILI9846 TFT Display using ESP32 Dev Board in 8 Bit Parallel Mode

Table of Contents

1.1 Introduction

The requirement was to produce a driver for a 3.5 inch ILI9486 based TFT display using a ESP32 development module. The UTFT driver library was used as a basis for the driver, the final result being a heavily modified version of the original library with functionality specifically for the ESP32-ILI9486 combination in 8-bit parallel mode. There was no requirement to provide any other compatibility with other boards or drivers.

Addition graphical functions were incorporated into this modified driver, further information on the driver is contained in section 1.5 of this document.

The software was written using the Arduino IDE version 1.8.12 (see section 3.1).

1.2 ESP Development Board

The ESP32 board used was the 38 pin version as shown below.

Figure 1: ESP32 Development board

This board will be used to interface and drive the TFT display in 8 bit parallel mode.

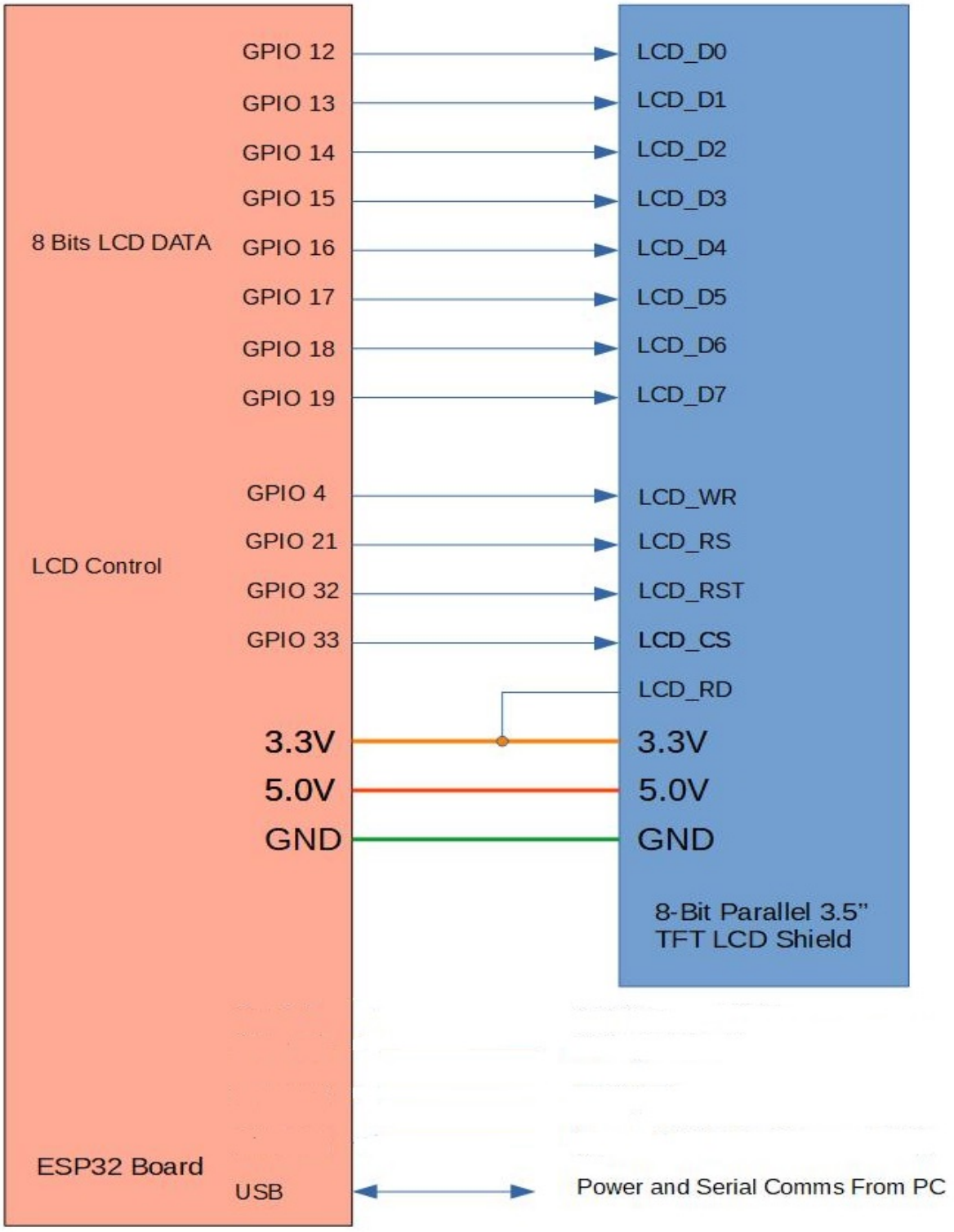


Figure 2: ESP32 to TFT Display Connections

1.3 8 Bit Parallel TFT ESP32 Interface Circuit

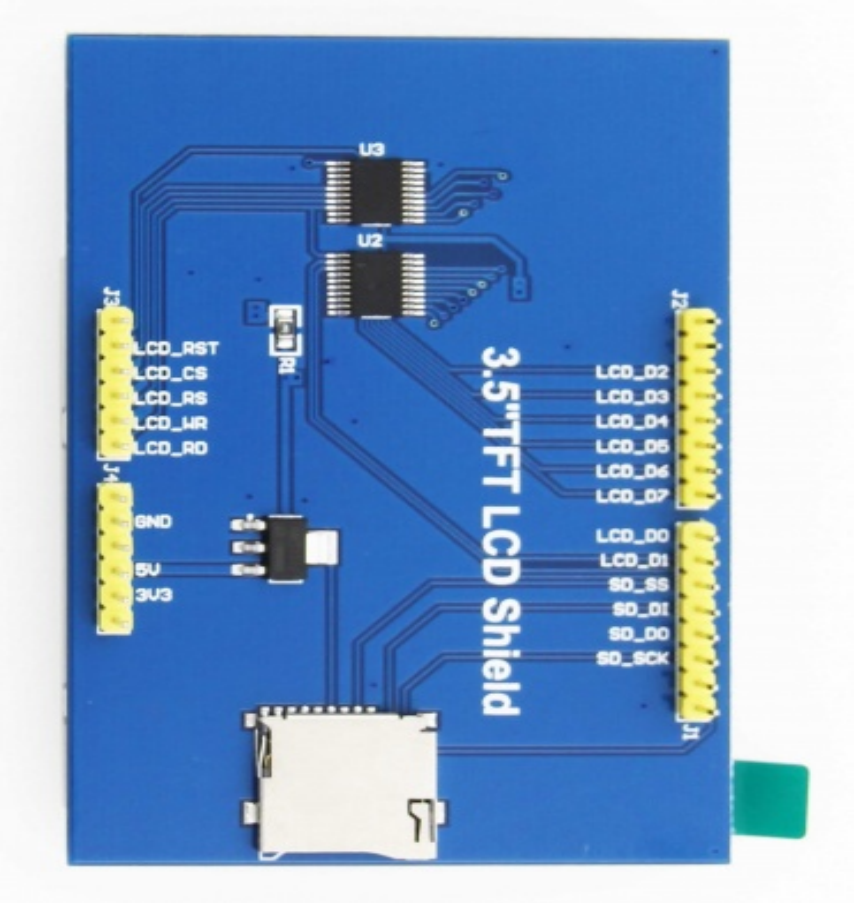


Figure 3: 3.5 inch ILI9846 480x320 TFT Display

Number	Pin Labe	Pin Description
1	LCD_RST	LCD bus reset signal: low reset
2	LCD_CS	LCD bus chip select signal: low enable
3	LCD_RS	LCD bus command / data selection signal low: command high: data
4	LCD_WR	LCD bus write signal
5	LCD_RD	LCD bus read signal
6	GND	Power ground
7	5V	5V power input
8	3V3	3.3V power input
9	LCD_D0	LCD 8-bit data Bit 0

10	LCD_D1	LCD 8-bit data Bit 1
11	LCD_D2	LCD 8-bit data Bit 2
12	LCD_D3	LCD 8-bit data Bit 3
13	LCD_D4	LCD 8-bit data Bit 4
14	LCD_D5	LCD 8-bit data Bit 5
15	LCD_D6	LCD 8-bit data Bit 6
16	LCD_D7	LCD 8-bit data Bit 7
17	SD_SS	SD card SPI bus chip select signal, low level enable
18	SD_DI	SD card SPI bus MOSI signal
19	SD_DO	SD card SPI bus MISO signal
20	SD_SCK	SD card SPI bus clock signal

ESP32 Interface GPIO Pins

N/C = Not Connected
N/A - Not Available either physically or used by other functions e.g SPI flash

ESP32 GPIO PIN	Connection To TFT Board
0	N/C
1	N/C
2	LCD_RD
3	UART 0 RXD
4	LCD_WR
5	N/C
6	X N/A
7	X N/A
8	X N/A
9	X N/A
10	X N/A
11	X N/A
12	LCD_D0
13	LCD_D1
14	LCD_D2
15	LCD_D3
16	LCD_D4
17	LCD_D5
18	LCD_D6

19	LCD_D7
21	LCD_RS
22	N/C
23	N/C
25	N/C
26	N/C
27	N/C
32	LCD_RST
33	LCD_CS
34	N/C
35	N/C
36(SP)	N/C

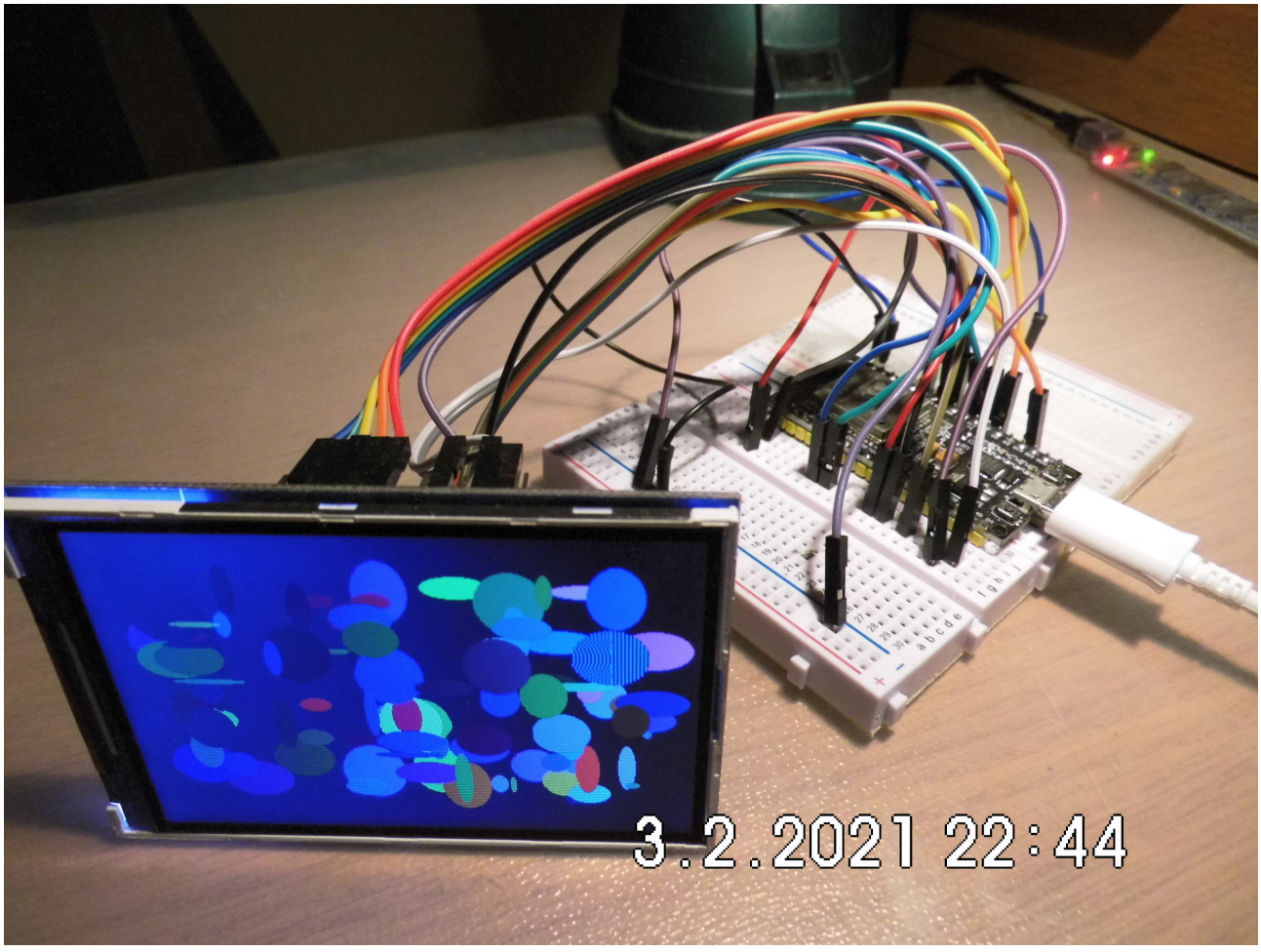


Figure 4: Circuit on breadboard

1.4 Driver Software

The driver UTFTESP32 library is an extensively modified version the UTFT library using ESP32 direct register access for controlling the ILI9846 based TFT display. As used the TFT display only requires write mode, so the read LCD_RD control line can be tied high (3.3V). The chip select line LCD_CS is always set to select (low 0V), so this line could also be tied low if required to free up the GPIO pin.

The driver uses direct ESP32 register access for speed, hence it is not compatible with other boards. Adduino methods such as pinMode and digitalWrite are only used in setting up. The **setxy.h** file in the tft_drivers folder is not used, **initlcd.h** is used for initialisation. ESP32 direct register control is used exclusively for graphics.

Addition functions are included to support fast drawing for graphics void UTFTESP32::FastDraw16 and void UTFTESP32::FastDraw8.

1.5 Additional Graphical Functions Added to UTFTESP32 Library

The original UTFT library was expended with the following graphical functions.

Improved circle fill

Replace original $r^2 = x^2 + y^2$ with Bresenham's algorithm.

Ellipse draw and fill

x and y are centre coordinates

a and b semi and major axes, if a=b then a circle

void UTFTESP32::drawEllipse(int x, int y, int a, int b)

void UTFTESP32::drawFilledEllipse(int x, int y, int a, int b)

Triangle draw and fill

x0 Vertex #0 x coordinate

y0 Vertex #0 y coordinate

x1 Vertex #1 x coordinate

y1 Vertex #1 y coordinate

x2 Vertex #2 x coordinate

y2 Vertex #2 y coordinate

colour 16-bit 5-6-5 Color to fill/draw with

void UTFTESP32::drawTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t colour)

void UTFTESP32::fillTriangle(int16_t x0, int16_t y0, int16_t x1, int16_t y1, int16_t x2, int16_t y2, uint16_t colour)

Scroll

Refer to ILI9846 data sheet

void UTFTESP32::vertScrollSetup(int16_t top_fixed_area, int16_t vert_scroll_start_addr, int16_t bottom_fixed_area)

void UTFTESP32::vertScroll(int16_t offset)

Four point Bezier curve

The four point Bezier curve is computational intensive process so the driver uses fixed point calculations for efficiency. A floating point version is also included but only as a comparison to demonstrate the speed difference with the fixed point function.

void UTFTESP32::bezierCurve(int x[4] , int y[4], uint16_t colour) - floating point version

void UTFTESP32::fp_bezierCurve(int x[4] , int y[4], uint16_t colour) - fixed point version

The fixed point version draws the curve about 24 times faster than the floating point version.

Bitmaps

There is a requirement for using a large number of 64x64 pixel icons in flash (PROGMEM) memory space. In general these as based on 24 bit colour bitmaps and are about 12kbytes in size. Given the limited memory of the ESP32 a large number of icons based on 24 bit bitmaps would be difficult, however an efficient solution was found by using 4-bit bitmaps (see section 2.1).

Support for 4 bit bitmap icons including background colour masking

Note this is only for landscape mode at the moment.

x,y start coordinates

sx, sy bitmap x,y, size in pixels

byte* data; data array containing 4 bit bitmap data generated by i**mage_convert1.exe** application

unsigned short int* table; 16 colour table containg 16 bit colours generated by **image_convert1.exe** application

bool mask; If true colour in table entry 16 (0f) normally white, will be replaced by current background colour.

void UTFTESP32::drawBitmap4(int x, int y, int sx, int sy,const byte* data, const unsigned short int* table, bool mask)

Support for 1 bit bitmap

Note this is only for landscape mode ot the moment.

x,y start coordinates

sx, sy bitmap x,y, size in pixels

byte* data; data array containing 1 bit bitmap data generated by image_convert.exe application

unsigned short int* table; The colour table only has two entrys generated by image_convert.exe application, but can be edited by hand.

unsigned short int* extcolour; If tableused is false this two colour table will be used instead. One entry can be set to background colour to provide masking effect.

bool tableused; If true colours from colour table will be used, if false colour extcolour table will be used instead.

void UTFTESP32::drawOneBitBitmap(int x, int y, int sx, int sy, const byte* data, const unsigned short int* table, unsigned short int* extcolour, bool tableused)

Additional Library Functions Supporting Graphical Methods

void UTFTESP32::FastDraw8(char colbyte, long pix) – Draws number of pixels given by pix value with colour given by single byte. This is useful to clear a screen to black fast.

void UTFTESP32::FastDraw16(char hibyte, char lobyte, long pix)– Draws number of pixels given by pix value with 16 bit colour given by hibyte lobyte.

int64_t inline UTFTESP32::imath_fixmul2(int32_t a, int32_t b) – Fixed point multiplier used in bezier curve calculations.

int64_t UTFTESP32::POW3(long a) - Calculate cubic; used in bezier curve calculations calls imath_fixmul2.

int64_t UTFTESP32::POW2(long a) Calculate square; used in bezier curve calculations calls imath_fixmul2.

2 Supporting Applications

Two windows applications have been produced to support the use icons generated from 4-bit bitmaps. These are **image_convert1.exe** and **palette4.exe**.

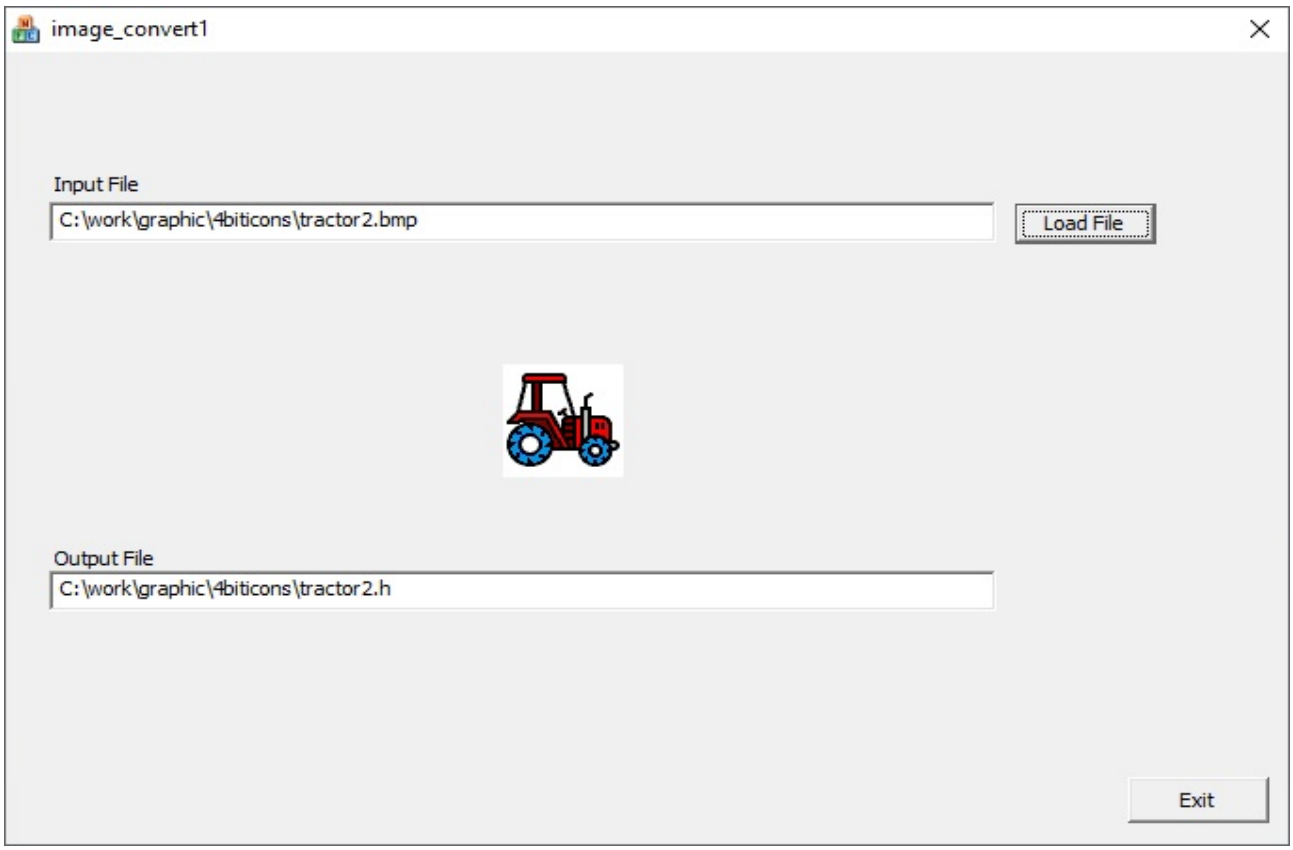
2.1 Application image_convert1.exe

It was found the most efficient 64x64 pixel icons for this applications were those generated from 4-bit bitmaps. The 4-bit bitmap has a 16 colour table which contains 24 bit colours. Each 4-bit nibble of data in the bitmap references a pixel colour in this table. It was found that the small (typically 64x64 pixel) icons generated using 24 bit colour bitmaps rarely contained more than 16 colours. Hence by editing the palette table in the 4-bit bitmap the same colours could be reproduced, generating a bitmap only 1/6th the size of the original 24 bit.

Example: below is the original 64x64 24 bit colour bitmap of a rain gauge size 12kB

Next a 64x64 4-bit bitmap whose palette table has been edited to produce the same 16 colours present in the 24 bit colour bitmap size 2.1kB;

To support the use of these 4-bit bimap icons the **image_convert1.exe** application converts the 4-bit bitmap data to a header file for use in the Arduino software. The original 24 bit colour table RRRRRRRR GGGGGGGG BBBBBBBB, is accurately converted to the nearest 16 bit value RRRRR GGGGGG BBBBB for use in the TFT display.



The conversion will only work with symmetrical 4-bit bitmap files. The header file produced contains the converted 16 bit colour table:

example table from for tractor icon header file

```
 const unsigned short int tractor2table[16] PROGMEM
={0x0000,0x8000,0x0400,0x8400,0x0010,0x8010,0x04DD,0x8410,0xC618,0xF800,0x07E0,0xFFE0,0x001F,0xF81F,0x07FF,0xFFFF};
```

followed by data of which a portion is shown here (data indicates colour index 0xf == 0xFFFF white).

```
const byte tractor2[0x0800] PROGMEM
={0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF
,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,
0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0
xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,  etc……...
```

In the Arduino code the header file is included and the data referenced;

```
#include "tractor2.h"
extern const unsigned short int tractor2table[16];
extern const byte tractor2[0x0800];
```

The bitmap is then generated using the new bitmap function in the modified UTFT library;

```
myGLCD.drawBitmap4(400, 0, 64, 64, tractor2, tractor2table, true);
```

From modified UTFTESP32 Library

void UTFTESP32::drawBitmap4(int x, int y, int sx, int sy, cont byte* data, const unsigned short int* table, bool mask)

Display 4-bit bitmap. Colours displayed are from table contained wthin bitmap

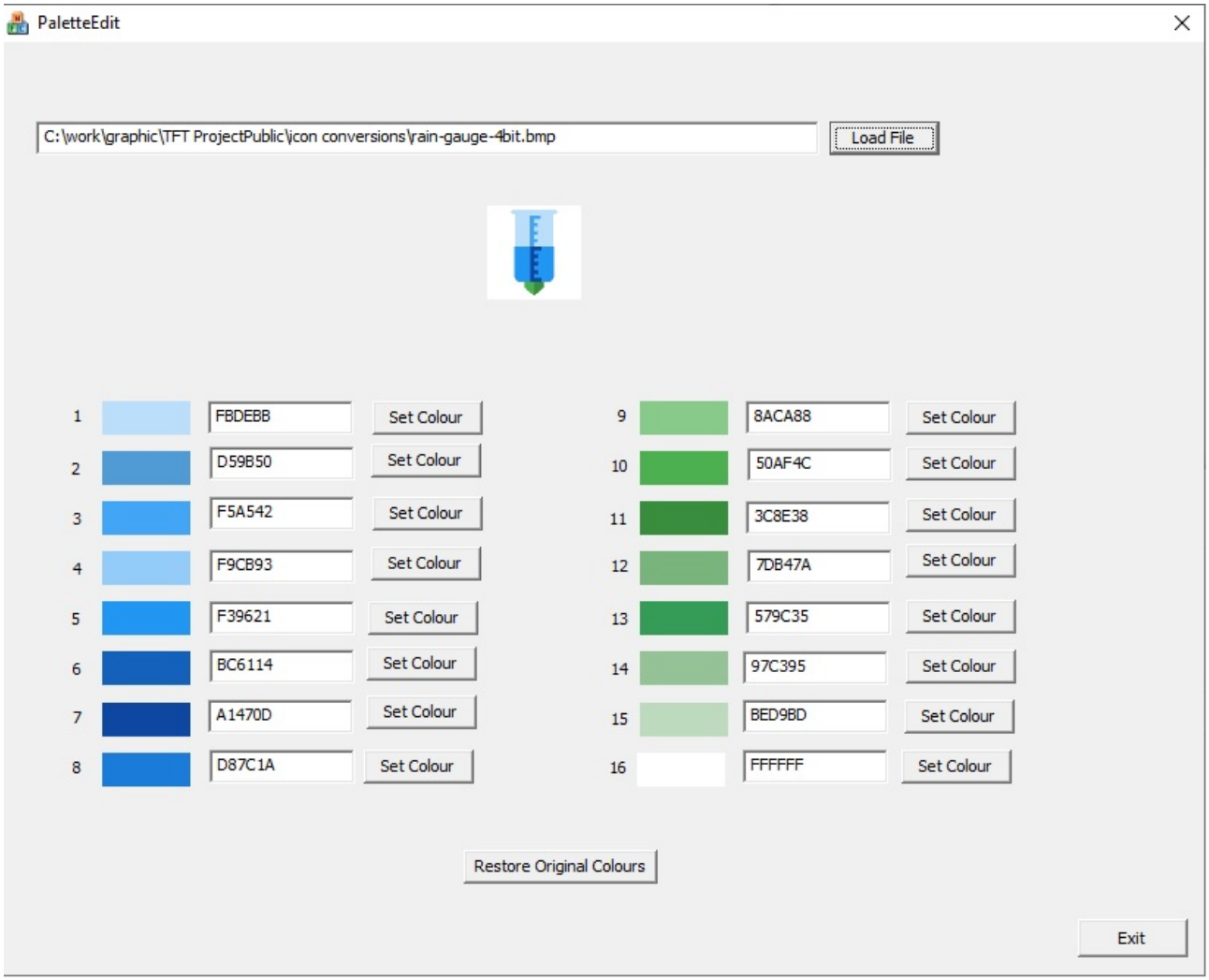
and converted to 16-bit colours for TFT display. If mask = true, the colour indicated

by table entry number 16, 0xf (usually white) will be replaced by current background colour.

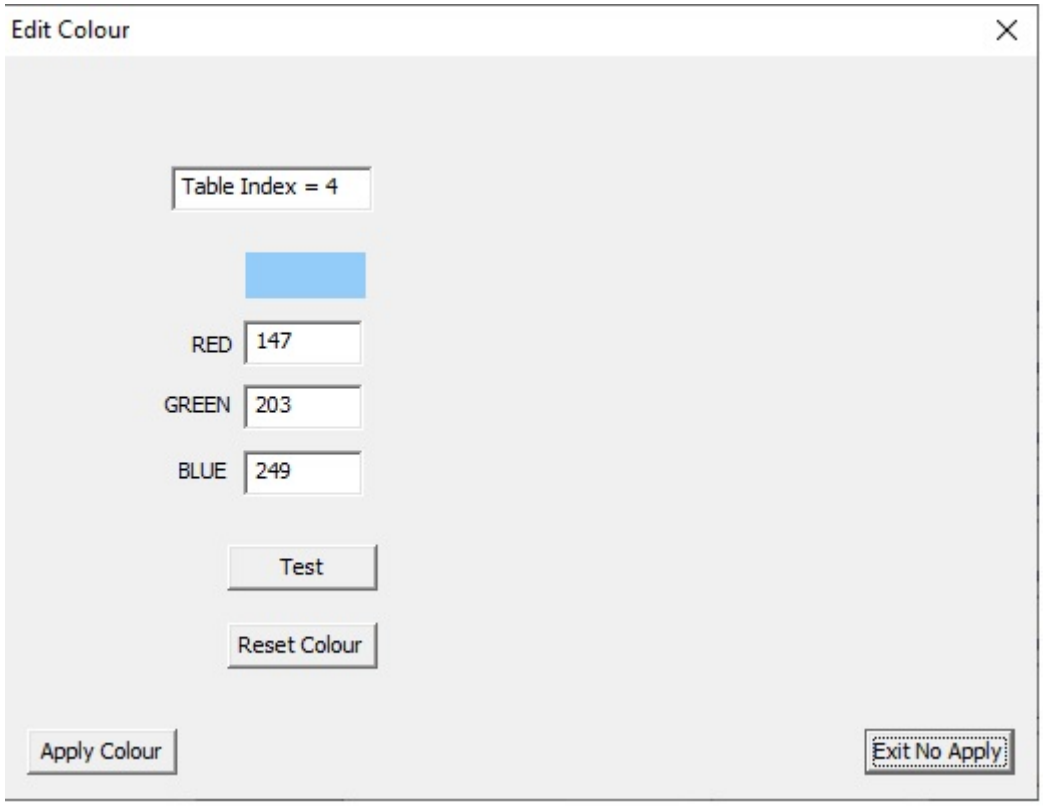
2.2 Application PaleteEdit.exe

It was found that most readily available tools such as Windows Paint do not provide much support for 4-bit bitmaps, and when used overwrite any modified palette table with a system default. To assist in generation of these bitmaps an application was produced to allow the editing of the 16 colour table in the 4-bit bitmap.

This example shows a 4-bit bitmap with an edited palette table which contains the same colours that were present in the original 24 bit bitmap. To edit an individual table entry the associated ‘Set Colour’ button is selected.



Here the colour can be modified to produce any colour from a 24 bit range.



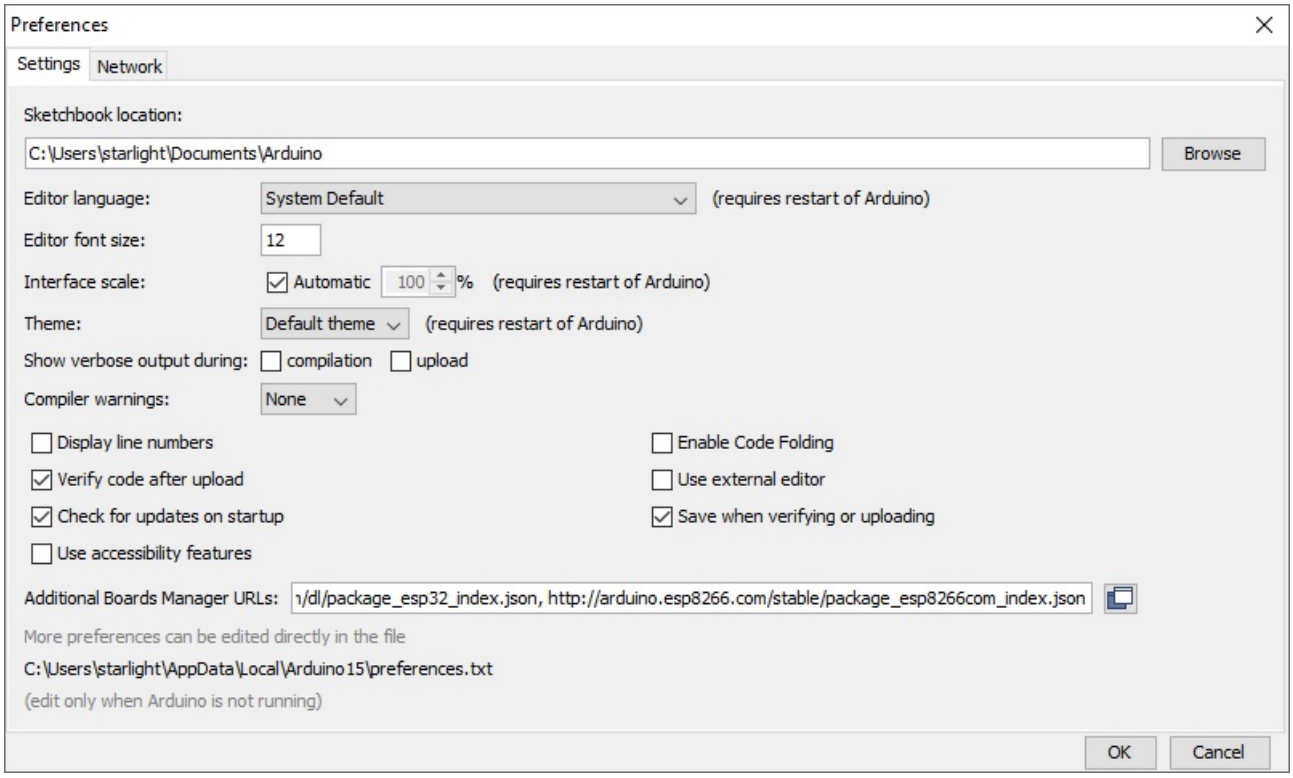
3 Arduino IDE

The version of the Arduino used was 1.8.12. To configure the IDE for ESP32 development requires the following setup. This assumes that the Arduino IDE has been installed.

3.1 Configure Setup Preferences

In the Arduino IDE select from File menu File → Preferences

Enter https://dl.espressif.com/dl/package_esp32_index.json into the “Additional Board Manager URLs” field as shown in the figure below, then click the “OK” button.



Note that this configuration also has the URL for the ESP8266 boards, separated by a comma

‘, http://arduino.esp8266.com/stable/package_esp8266com_index.json’.

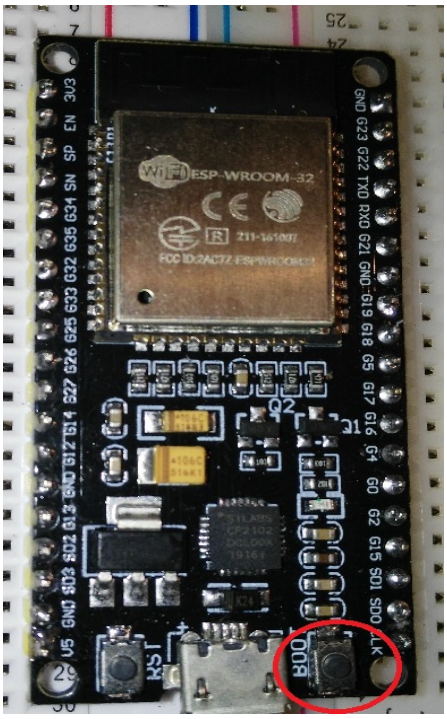
3.2 Instal software supporting ESP32 board

Next Open Board manager, select Tools → Board → Boards Manager

Search for ESP32, and press install button for the “ESP32 by Espressif Systems” shown below is after ESP32 installation.

3.3 Uploading Code using ESP32 Dev Board

When uploading a new software Sketch → Upload on the ESP32 development board it was found necessary to press the BOOT button (circled below) when the “Connecting…..” message appeared on the IDE until communication began, which on the board is indicated by rapid flashing on the led, and on the IDE download starts, after which the button can be released.



4 UTFTESP32 Library

The modified UTFTESP32 library can be installed by copying to the Adduino Library's folder which is typically in location “C:\Users\CompuerName\Documents\Arduino\libraries\”. If a previous UTFT library exist if should be removed before copying. Create folder ‘UTFTESP32’ and copy the new library. Hence new library will be at location “C:\Users\starlight\Documents\Arduino\libraries\UTFT”.