devacron.com

# Add an ST7735 TFT display to an ESP32 - Devacron.com

*by admin*

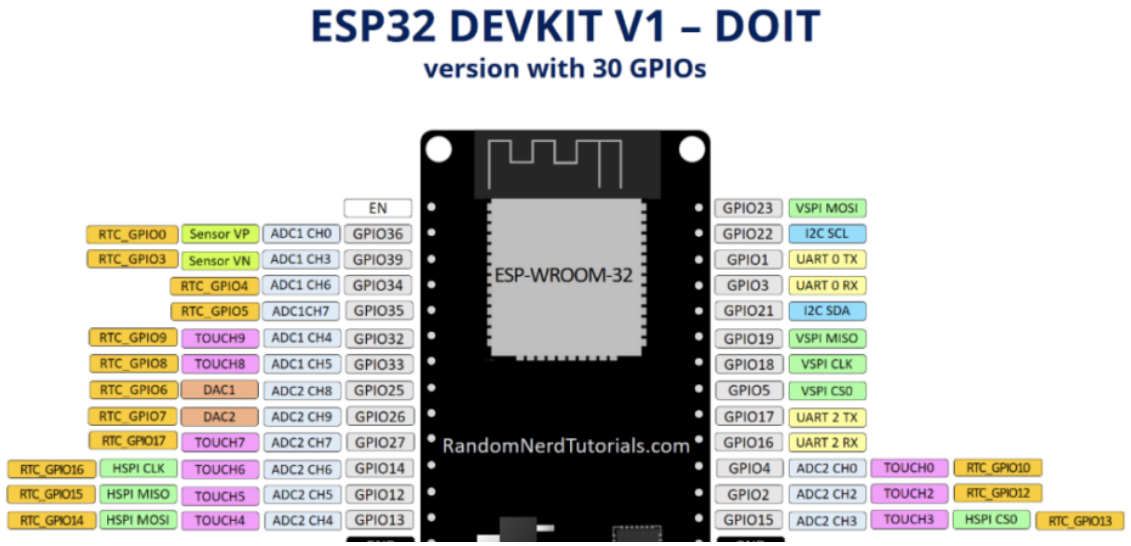
18–22 minutes

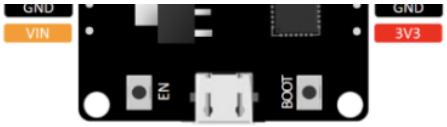
---

Hello,

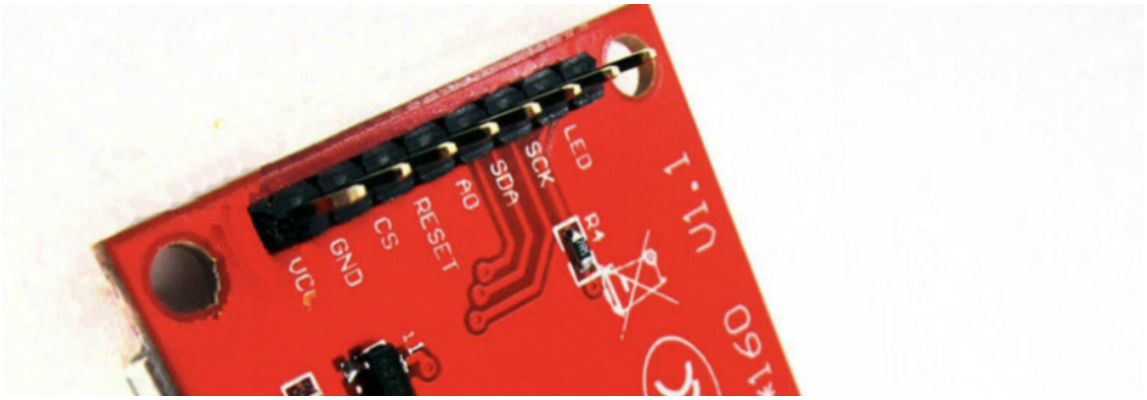
I wanted to try these ST7735 inexpensive displays that can be found all over the internet, so I ordered a couple for a few euros each. My quick research showed that a number of libraries support them and it turns out that you can display anything you want. Of course, we are not talking about playing modern games on it or watching 4k videos. These are just simple displays that can be really helpful to any project.

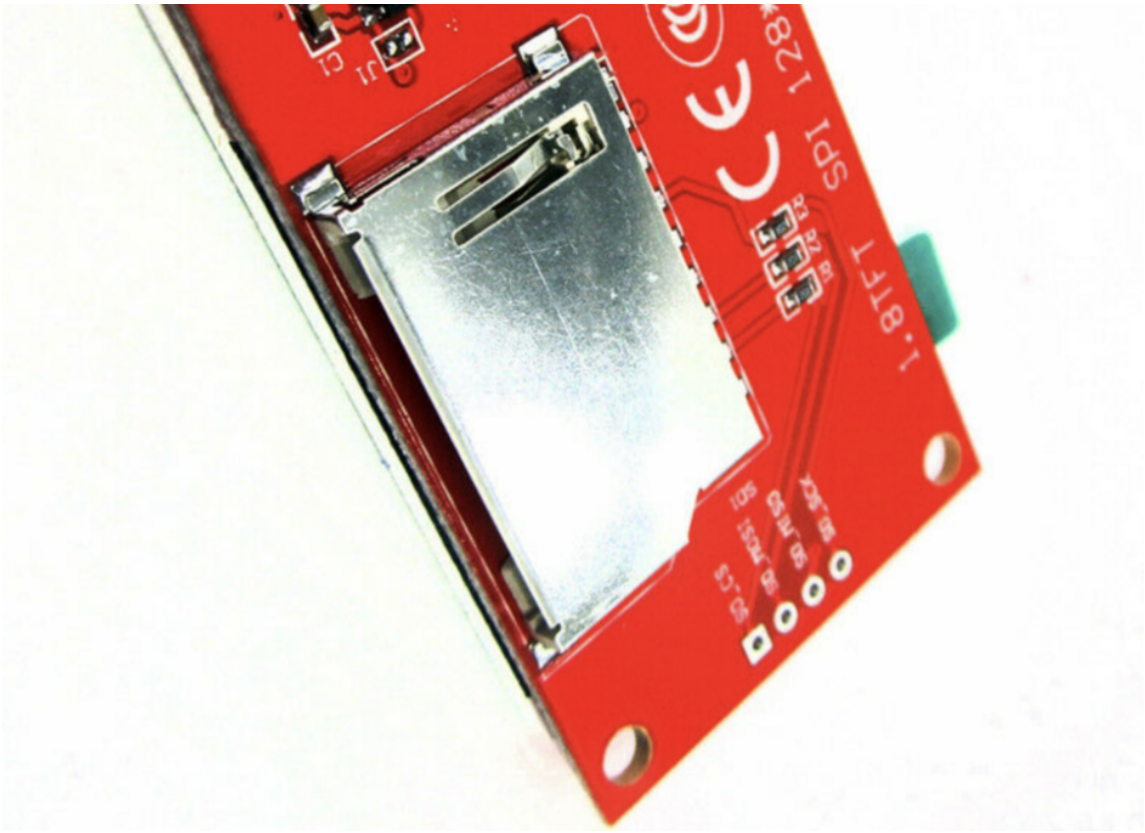
I used an older version of ESP32, the DEVKITV1, and actually the smaller version with the 30 pins. If you have a different one please try to find the correct pinout because they differ. Here is the pinout diagram from mine






And here is how the TFT looks. As you see it also has a port for an SD card if you want to use e.g. for reading images from it. In my case, I didn't connect it.






The resolution is 128×160 pixels and this is something you need to be aware of since you may need to configure it in your code.

The connections are:

| TFT | ESP32 |
|---|---|
| 5V | VIN |
| GND | GND |
| LED | VIN |
| SCL (SPI Clock) | GPIO18 VSPI SCK (SPI Clock) |
| SDA (SPI Data) | GPIO23 VSPI MOSI (MOSI, Data to Screen) |
| RS (Register Select) | GPIO2 (labelled as DC in Adafruit code) |
| RST (Screen reset) | GPIO4 |

| TFT | ESP32 |
| --- | --- |
| CS (Chip Select) | GPIO5 VSPI SS (Slave Select/ Chip Select) |

Once you have the connections ready next step is to install the TFT library in your Arduino IDE. Go to Tools – > Manage Libraries and then search for TFT_eSPI and click install. Alternatively, crab the lib from here.

Next step is to configure the pins in the file User_Setup.h of the library. In your favourite text editor open the file and change the pins in lines 124,125,126. Here how mine looks:

```
//                                   USER DEFINED
SETTINGS
//   Set driver type, fonts to be loaded, pins
used and SPI control method etc
//
//   See the User_Setup_Select.h file if you
wish to be able to define multiple
//   setups and then easily select which setup
file is used by the compiler.
//
//   If this file is edited correctly then all
the library example sketches should
//   run without the need to make any more
changes for a particular hardware setup!
//   Note that some sketches are designed for a
particular TFT pixel width/height

//
##############################################
//
```

```
// Section 0. Call up the right driver file and
any options for it
//
//
##############################################

// Only define one driver, the other ones must
be commented out
//#define ILI9341_DRIVER
#define ST7735_DRIVER
//#define ILI9163_DRIVER
//#define S6D02A1_DRIVER
//#define RPI_ILI9486_DRIVER // 20MHz maximum
SPI
//#define HX8357D_DRIVER
//#define ILI9481_DRIVER
//#define ILI9488_DRIVER
//#define ST7789_DRIVER

// For M5Stack ESP32 module with integrated
display ONLY, remove // in line below
//#define M5STACK

// For ST7735  and ILI9163 ONLY, define the
pixel width and height in portrait orientation
#define TFT_WIDTH  128
#define TFT_HEIGHT 160
//#define TFT_HEIGHT 128

// For ST7735 ONLY, define the type of display,
originally this was based on the
// colour of the tab on the screen protector
```

```
film but this is not always true, so try
// out the different options below if the
screen does not display graphics correctly,
// e.g. colours wrong, mirror images, or tray
pixels at the edges.
// Comment out ALL BUT ONE of these options for
a ST7735 display driver, save this
// this User_Setup file, then rebuild and
upload the sketch to the board again:

//#define ST7735_INITB
#define ST7735_GREENTAB
//#define ST7735_GREENTAB2
//#define ST7735_GREENTAB3
//#define ST7735_GREENTAB128 // For 128 x 128
display
//#define ST7735_REDTAB
//#define ST7735_BLACKTAB

//
##############################################
//
// Section 1. Define the pins that are used to
interface with the display here
//
//
##############################################

// We must use hardware SPI, a minimum of 3
GPIO pins is needed.
// Typical setup for ESP8266 NodeMCU ESP-12 is
:
```

```
//
// Display SDO/MISO  to NodeMCU pin D6 (or
leave disconnected if not reading TFT)
// Display LED       to NodeMCU pin VIN (or 5V,
see below)
// Display SCK       to NodeMCU pin D5
// Display SDI/MOSI  to NodeMCU pin D7
// Display DC (RS/AO)to NodeMCU pin D3
// Display RESET     to NodeMCU pin D4 (or RST,
see below)
// Display CS        to NodeMCU pin D8 (or GND,
see below)
// Display GND       to NodeMCU pin GND (0V)
// Display VCC       to NodeMCU 5V or 3.3V
//
// The TFT RESET pin can be connected to the
NodeMCU RST pin or 3.3V to free up a control
pin
//
// The DC (Data Command) pin may be labeled AO
or RS (Register Select)
//
// With some displays such as the ILI9341 the
TFT CS pin can be connected to GND if no more
// SPI devices (e.g. an SD Card) are connected,
in this case comment out the #define TFT_CS
// line below so it is NOT defined. Other
displays such at the ST7735 require the TFT CS
pin
// to be toggled during setup, so in these
cases the TFT_CS line must be defined and
connected.
```

```
//
// The NodeMCU D0 pin can be used for RST
//
// See Section 2. below if DC or CS is
connected to D0
//
// Note: only some versions of the NodeMCU
provide the USB 5V on the VIN pin
// If 5V is not available at a pin you can use
3.3V but backlight brightness
// will be lower.


// ###### EDIT THE PIN NUMBERS IN THE LINES
FOLLOWING TO SUIT YOUR ESP8266 SETUP ######

// For NodeMCU - use pin numbers in the form
PIN_Dx where Dx is the NodeMCU pin designation
//#define TFT_CS   PIN_D8  // Chip select
control pin D8
//#define TFT_DC   PIN_D3  // Data Command
control pin
//#define TFT_RST  PIN_D4  // Reset pin (could
connect to NodeMCU RST, see next line)
//#define TFT_RST  -1  // Set TFT_RST to -1 if
the display RESET is connected to NodeMCU RST
or 3.3V

//#define TOUCH_CS PIN_D2     // Chip select
pin (T_CS) of touch screen

//#define TFT_WR PIN_D2       // Write strobe
```

```
for modified Raspberry Pi TFT only


// ######  FOR ESP8266 OVERLAP MODE EDIT THE
PIN NUMBERS IN THE FOLLOWING LINES  ######

// Overlap mode shares the ESP8266 FLASH SPI
bus with the TFT so has a performance impact
// but saves pins for other functions.
// Use NodeMCU SD0=MISO, SD1=MOSI, CLK=SCLK to
connect to TFT in overlap mode

// In ESP8266 overlap mode the TFT chip select
MUST connect to pin D3
//#define TFT_CS   PIN_D3
//#define TFT_DC   PIN_D5  // Data Command
control pin
//#define TFT_RST  PIN_D4  // Reset pin (could
connect to NodeMCU RST, see next line)
//#define TFT_RST  -1  // Set TFT_RST to -1 if
the display RESET is connected to NodeMCU RST
or 3.3V

// In ESP8266 overlap mode the following must
be defined
//#define TFT_SPI_OVERLAP

// ###### EDIT THE PIN NUMBERS IN THE LINES
FOLLOWING TO SUIT YOUR ESP32 SETUP   ######

// For ESP32 Dev board (only tested with
ILI9341 display)
```

```
// The hardware SPI can be mapped to any pins

//#define TFT_MISO 19
//#define TFT_MOSI 23
//#define TFT_SCLK 18
#define TFT_CS    5  // Chip select control pin
#define TFT_DC    2  // Data Command control
pin
#define TFT_RST   4  // Reset pin (could
connect to RST pin)
//#define TFT_RST  -1  // Set TFT_RST to -1 if
display RESET is connected to ESP32 board RST

// For the M5Stack module use these #define
lines
//#define TFT_MISO 19
//#define TFT_MOSI 23
//#define TFT_SCLK 18
//#define TFT_CS   14  // Chip select control
pin
//#define TFT_DC   27  // Data Command control
pin
//#define TFT_RST  33  // Reset pin (could
connect to Arduino RESET pin)
//#define TFT_BL   32  // LED back-light

//#define TOUCH_CS 21     // Chip select pin
(T_CS) of touch screen

//#define TFT_WR 22    // Write strobe for
modified Raspberry Pi TFT only
```

```
// ######       EDIT THE PINs BELOW TO SUIT
YOUR ESP32 PARALLEL TFT SETUP        ######

// The library supports 8 bit parallel TFTs
with the ESP32, the pin
// selection below is compatible with ESP32
boards in UNO format.
// Wemos D32 boards need to be modified, see
diagram in Tools folder.
// Only ILI9481 and ILI9341 based displays have
been tested!

// Parallel bus is only supported on ESP32
// Uncomment line below to use ESP32 Parallel
interface instead of SPI

//#define ESP32_PARALLEL

// The ESP32 and TFT the pins used for testing
are:
//#define TFT_CS   33  // Chip select control
pin (library pulls permanently low
//#define TFT_DC   15  // Data Command control
pin - use a pin in the range 0-31
//#define TFT_RST  32  // Reset pin, toggles on
startup

//#define TFT_WR    4  // Write strobe control
pin - use a pin in the range 0-31
//#define TFT_RD    2  // Read strobe control
pin  - use a pin in the range 0-31
```

```
//#define TFT_D0   12  // Must use pins in the
range 0-31 for the data bus
//#define TFT_D1   13  // so a single register
write sets/clears all bits.
//#define TFT_D2   26  // Pins can be randomly
assigned, this does not affect
//#define TFT_D3   25  // TFT screen update
performance.
//#define TFT_D4   17
//#define TFT_D5   16
//#define TFT_D6   27
//#define TFT_D7   14

//
##############################################
//
// Section 2. Define the way the DC and/or CS
lines are driven (ESP8266 only)
//
//
##############################################

// Normally the library uses direct register
access for the DC and CS lines for speed
// If D0 (GPIO16) is used for CS or DC then a
different slower method must be used
// Uncomment one line if D0 is used for DC or
CS
// DC on D0 = 6% performance penalty at 40MHz
SPI running graphics test
// CS on D0 = 2% performance penalty at 40MHz
SPI running graphics test
```

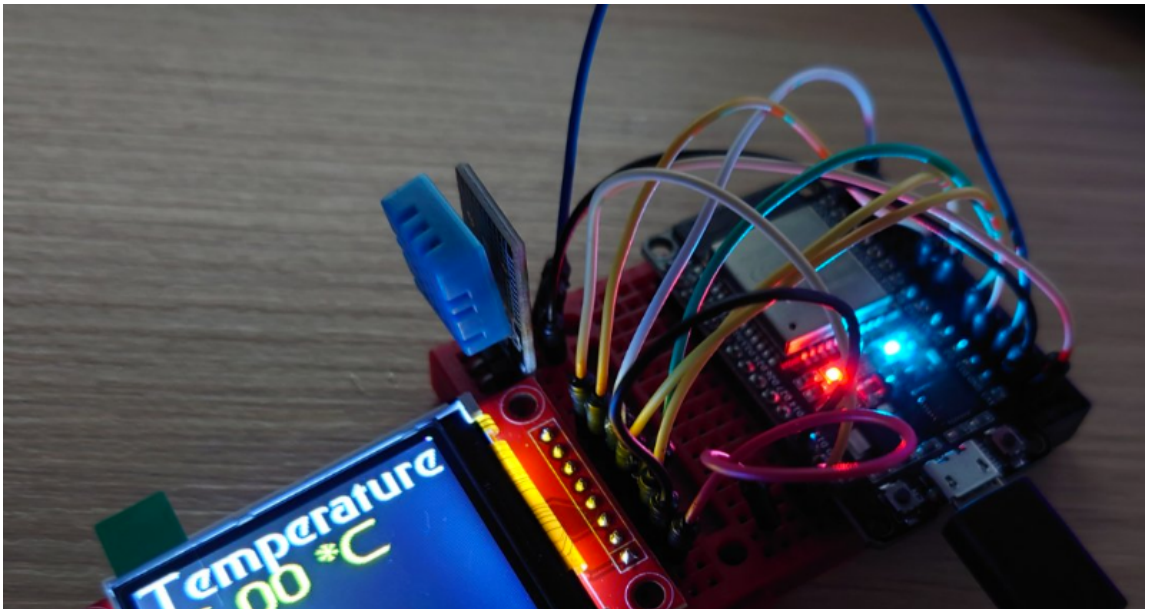
```
// #define D0_USED_FOR_DC
// #define D0_USED_FOR_CS

//
##################################################
//
// Section 3. Define the fonts that are to be
used here
//
//
##################################################

// Comment out the #defines below with // to
stop that font being loaded
// The ESP8366 and ESP32 have plenty of memory
so commenting out fonts is not
// normally necessary. If all fonts are loaded
the extra FLASH space required is
// about 17Kbytes. To save FLASH space only
enable the fonts you need!

//#define LOAD_GLCD   // Font 1. Original
Adafruit 8 pixel font needs ~1820 bytes in
FLASH
//#define LOAD_FONT2  // Font 2. Small 16 pixel
high font, needs ~3534 bytes in FLASH, 96
characters
//#define LOAD_FONT4  // Font 4. Medium 26
pixel high font, needs ~5848 bytes in FLASH, 96
characters
//#define LOAD_FONT6  // Font 6. Large 48 pixel
```

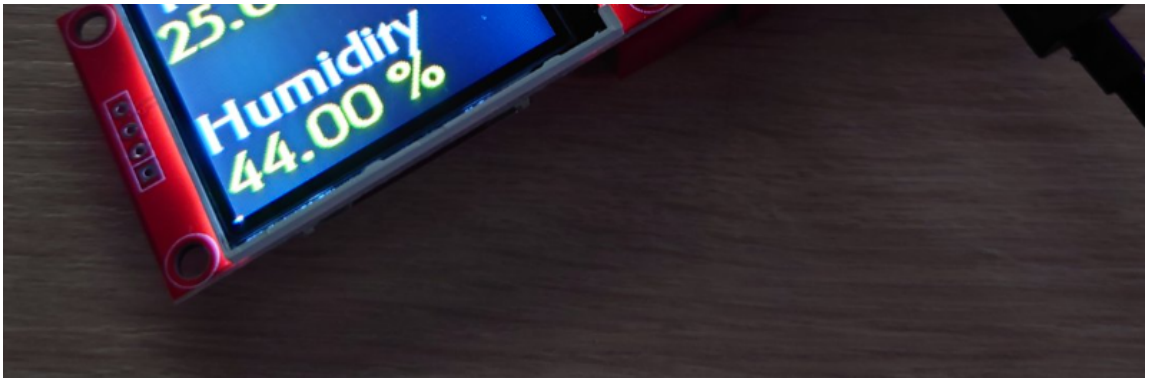
```
font, needs ~2666 bytes in FLASH, only
characters 1234567890:-.apm
#define LOAD_FONT7  // Font 7. 7 segment 48
pixel font, needs ~2438 bytes in FLASH, only
characters 1234567890:-.
#define LOAD_FONT8  // Font 8. Large 75 pixel
font needs ~3256 bytes in FLASH, only
characters 1234567890:-.
//#define LOAD_FONT8N // Font 8. Alternative to
Font 8 above, slightly narrower, so 3 digits
fit a 160 pixel TFT
#define LOAD_GFXFF  // FreeFonts. Include
access to the 48 Adafruit_GFX free fonts FF1 to
FF48 and custom fonts

// Comment out the #define below to stop the
SPIFFS filing system and smooth font code being
loaded
// this will save ~20kbytes of FLASH
#define SMOOTH_FONT

//
##############################################
//
// Section 4. Not used
//
//
##############################################



//
##############################################
```

```
//
// Section 5. Other options
//
//
##########################################

// Define the SPI clock frequency, this affects
the graphics rendering speed. Too
// fast and the TFT driver will not keep up and
display corruption appears.
// With an ILI9341 display 40MHz works OK,
80MHz sometimes fails
// With a ST7735 display more than 27MHz may
not work (spurious pixels and lines)
// With an ILI9163 display 27 MHz works OK.
// The RPi typically only works at 20MHz
maximum.

// #define SPI_FREQUENCY   1000000
// #define SPI_FREQUENCY   5000000
// #define SPI_FREQUENCY  10000000
// #define SPI_FREQUENCY  20000000
#define SPI_FREQUENCY  27000000 // Actually
sets it to 26.67MHz = 80/3
// #define SPI_FREQUENCY  40000000 // Maximum
to use SPIFFS
// #define SPI_FREQUENCY  80000000

// The XPT2046 requires a lower SPI clock rate
of 2.5MHz so we define that here:
#define SPI_TOUCH_FREQUENCY  2500000
```

```
// Comment out the following #define if "SPI
Transactions" do not need to be
// supported. When commented out the code size
will be smaller and sketches will
// run slightly faster, so leave it commented
out unless you need it!

// Transaction support is needed to work with
SD library but not needed with TFT_SdFat
// Transaction support is required if other SPI
devices are connected.

// Transactions are automatically enabled by
the library for an ESP32 (to use HAL mutex)
// so changing it here has no effect

//#define SUPPORT_TRANSACTIONS
```

After this, you can pick any of the examples from the library to upload to your ESP32 microcontroller. Some of them are really nice. For testing, I connected a DHT11 temperature/humidity sensor and I displayed the readings in the ST7735. Sweet!






And here is my code if you want to copy it.

```
/*
  Sketch to demonstrate using the print class
with smooth fonts,
  the Smooth fonts are stored in a FLASH
program memory array.

  Sketch is written for a 240 x 320 display

  New font files in the .vlw format can be
created using the Processing
  sketch in the library Tools folder. The
Processing sketch can convert
  TrueType fonts in *.ttf or *.otf files.

  The library supports 16 bit unicode
characters:
  https://en.wikipedia.org/wiki/Unicode_font

  The characters supported are in the in the
Basic Multilingual Plane:
  https://en.wikipedia.org/wiki/
Plane_(Unicode)#Basic_Multilingual_Plane

  Make sure all the display driver and pin
```

```
connections are correct by
  editing the User_Setup.h file in the TFT_eSPI
library folder.
*/

//  The font is stored in an array within a
sketch tab.

//  A processing sketch to create new fonts can
be found in the Tools folder of TFT_eSPI
//  https://github.com/Bodmer/TFT_eSPI/tree/
master/Tools/Create_Smooth_Font/Create_font

#include "Final_Frontier_28.h"

// Graphics and font library
#include <TFT_eSPI.h>
#include <SPI.h>

// Sensor libs
#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>

#define DHTPIN            15         // Pin
which is connected to the DHT sensor.
#define DHTTYPE           DHT11     // DHT 11

DHT_Unified dht(DHTPIN, DHTTYPE);

uint32_t delayMS;
```

```
TFT_eSPI tft = TFT_eSPI();  // Invoke library

//
----------------------------------------------
// Setup
//
----------------------------------------------
void setup(void) {
  Serial.begin(115200); // Used for messages

  tft.init();
  tft.setRotation(1);

  dht.begin();
    sensor_t sensor;
  dht.temperature().getSensor(&sensor);

Serial.println("------------------------------
  Serial.println("Temperature");
  Serial.print  ("Sensor:       ");
Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   ");
Serial.println(sensor.version);
  Serial.print  ("Unique ID:    ");
Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    ");
Serial.print(sensor.max_value);
Serial.println(" *C");
  Serial.print  ("Min Value:    ");
Serial.print(sensor.min_value);
Serial.println(" *C");
  Serial.print  ("Resolution:   ");
```

```
Serial.print(sensor.resolution);
Serial.println(" *C");

Serial.println("------------------------------
  // Print humidity sensor details.
  dht.humidity().getSensor(&sensor);

Serial.println("------------------------------
  Serial.println("Humidity");
  Serial.print  ("Sensor:       ");
Serial.println(sensor.name);
  Serial.print  ("Driver Ver:   ");
Serial.println(sensor.version);
  Serial.print  ("Unique ID:    ");
Serial.println(sensor.sensor_id);
  Serial.print  ("Max Value:    ");
Serial.print(sensor.max_value);
Serial.println("%");
  Serial.print  ("Min Value:    ");
Serial.print(sensor.min_value);
Serial.println("%");
  Serial.print  ("Resolution:   ");
Serial.print(sensor.resolution);
Serial.println("%");

Serial.println("------------------------------
  // Set delay between sensor readings based on
sensor details.
  delayMS = sensor.min_delay / 1000;
}

//
```

```
//------------------------------------------------
// Main loop
//
//------------------------------------------------
void loop() {
  // Wrap test at right and bottom of screen
  tft.setTextWrap(true, true);

  // Font and background colour, background
colour is used for anti-alias blending
  tft.setTextColor(TFT_WHITE, TFT_BLACK);

  // Load the font
  tft.loadFont(Final_Frontier_28);

  // Display all characters of the font
//  tft.showFont(2000);

  // Set "cursor" at top left corner of display
(0,0)
  // (cursor will move to next line
automatically during printing with
'tft.println'
  //  or stay on the line is there is room for
the text with tft.print)
  tft.setCursor(0, 0);

  // Set the font colour to be white with a
black background, set text size multiplier to 1
  tft.setTextColor(TFT_WHITE, TFT_BLACK);
```

```
    // Delay between measurements.
  delay(delayMS);
  // Get temperature event and print its value.
  sensors_event_t event;
  dht.temperature().getEvent(&event);
  tft.fillScreen(TFT_BLACK);
  if (isnan(event.temperature)) {
    Serial.println("Error reading
temperature!");
  }
  else {
    Serial.print("Temperature: ");
    Serial.print(event.temperature);
    Serial.println(" *C");
    tft.setTextSize(2);
    tft.setTextColor(TFT_WHITE, TFT_BLACK);
    tft.println("Temperature");
    tft.setTextColor(TFT_YELLOW, TFT_BLACK);
    tft.print(event.temperature);
    tft.print(" *C");
  }
  // Get humidity event and print its value.
  dht.humidity().getEvent(&event);
  if (isnan(event.relative_humidity)) {
    Serial.println("Error reading humidity!");
  }
  else {
    Serial.print("Humidity: ");
    Serial.print(event.relative_humidity);
    Serial.println("%");
    tft.println(" ");
    tft.println(" ");
```

```
    tft.setTextSize(2);
    tft.setTextColor(TFT_WHITE, TFT_BLACK);
    tft.println("Humidity");
    tft.setTextColor(TFT_YELLOW, TFT_BLACK);
    tft.print(event.relative_humidity);
    tft.print(" %");
  }

  // Unload the font to recover used RAM
  tft.unloadFont();

  delay(5000);
}
```

The code is not 100% as I need to find a better way to update the screen instead of filling the whole screen with black background before an update. There is documentation about it here https://learn.adafruit.com/adafruit-gfx-graphics-library/graphics-primitives I need to go through it.

Also as you can see I am using a custom font for the text so you will need to include the Final_Frontier_28.h file. It will probably be easier to change the code from the example Print_Smooth_Font. This is what I did any.

Enjoy!

Post Views: 29,028