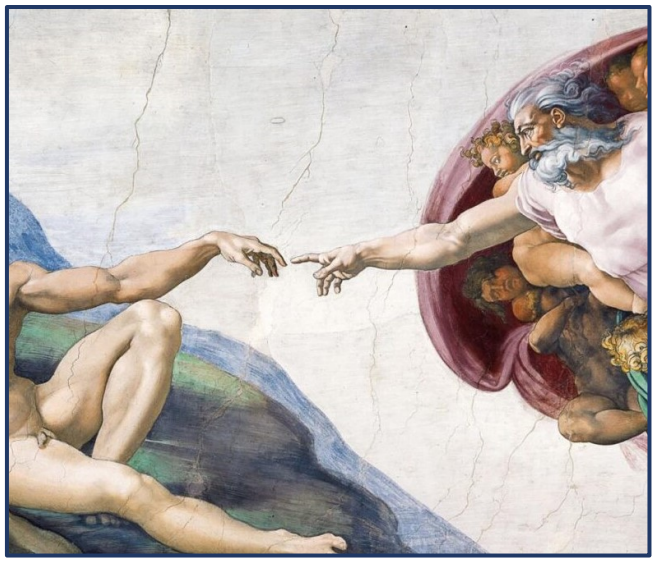
r

Touch Control

Add Touch Control to your Microcontroller Project

Bruce E. Hall, W8BH



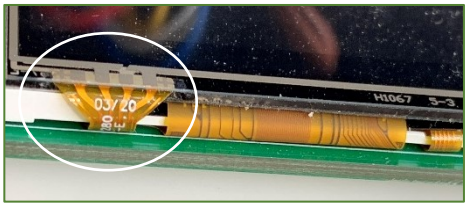
Introduction.

If you have an ILI9341 display module, and want to add touch control to your projects, this article is for you. First, determine that your module is touch-enabled. Many online vendors advertise their modules as touch-screens, but in fact they are not. *Caveat Emptor.*

There are 3 physical signs that your screen is touch-enabled:

1. **Does it have a touch panel?**

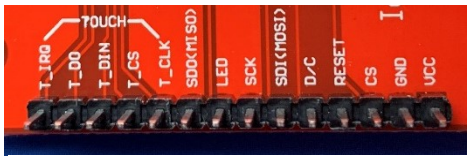
   The touch panel is a thin, transparent overlay that is bonded to the top surface of the display. The clues to its presence are four, thick traces on a flexible cable that connect the touch panel to your module. Carefully examine the flexible cables, found on one of the short sides of the display.



Display with flexible 4-wire touch cable

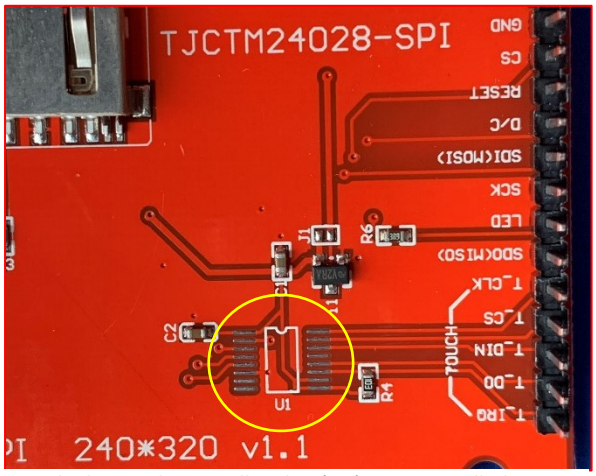
Display with no touch cable

2. **Does it have touch pins?**
   Count the number of pins on each side of the module. Modules with touch pins usually have 14 pins on one side, and 4 pins on the other. The touch pins are often labelled “T_IRQ”, “T_DO”, “T_DIN”, “T_CS”, and “T_CLK”.

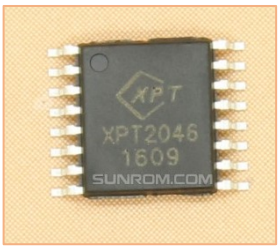


14 pins, with the first 5 labelled as touch

3. **Does it have a touch controller?**



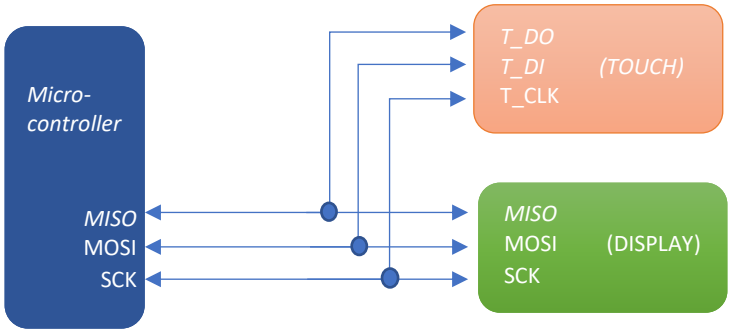
Oops! The touch controller ship (U1) is not installed



The touch controller is a small, square surface-mounted chip on the back of the module, often located adjacent to the touch pins. This part is the XPT2046. Clone chips will often use a similar part number, such as "RB2046". You might also encounter boards that have pads for the chip, but no chip installed.

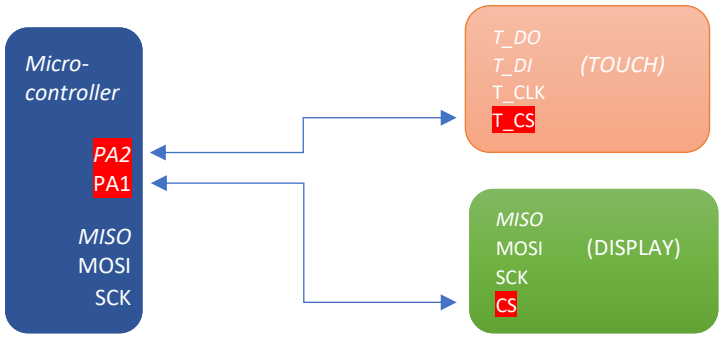
STEP1: WIRING

After confirming that you have a touch-screen, it is time to add it to your circuit. The XPT2046 touch controller uses SPI to communicate with your microcontroller, just like the display. So, we will add it to the existing three-wire SPI bus (MISO, MOSI, SCK). For the touch pins, these lines have slightly different names: MISO is "T_DO", MOSI is "T_DI", and SCK is "T_CLK":



The touch controller and the display are both peripheral devices. Only one peripheral device can be active on the SPI bus at a time. So how are the controlled? By adding an additional line for each device, called "chip select". The microcontroller enables only one of these lines at a line, and the peripheral will only read from or write to the bus when its corresponding chip select line is active:

The above description is true for all microcontrollers. For the rest of this tutorial, however, I will use the Blue Pill (STM32F103) microcontroller. For this micro, connect PA1 to the display CS, and PA2 to touch chip select "T_CS". MISO, MOSI and SCK connect to PA6, PA7, and PA5, respectively.



STEP 2: THE SOFTWARE

I assume that you are comfortable with the Arduino IDE and know how to program a Blue Pill microcontroller. The Blue Pill was initially supported in the Arduino IDE with a fantastic package written by Roger Clarke and hosted at dan.drown.org. I had great success using this software but Roger no longer supports his package. In the meantime, STMicroelectronics, the makers of the microcontroller in the Blue Pill, now support the Arduino environment and have created their own software package. To use it, copy the following URL into your Arduino Boards Manager list.

https://github.com/stm32duino/BoardManagerFiles/raw/master/STM32/package_stm_index.json

For TFT support I am using "TFT_eSPI" by Bodmer, version 2.2.14. To install it, go to the Arduino library manager (Sketch->Include Libaries->Manage Libraries), search for "TFT_eSPI", and install. You can also find the latest code on GitHub at https://github.com/Bodmer/TFT_eSPI

Once the TFT Library is installed, you will need to configure it by modifying the User_Setup.h file in your TFT_eSPI library directory. I'd prefer setting the configuration in my sketch, rather than modifying a file, but this is not a choice. Edit your User_Setup.h file to include the following DEFINEs:

```
#define STM32
#define ILI9341_DRIVER
#define TFT_SPI_PORT 1
#define TFT_MOSI PA7
#define TFT_MISO PA6
#define TFT_SCLK PA5
#define TOUCH_CS PA2
#define TFT_CS   PA1
#define TFT_DC   PA0
#define TFT_RST  -1
#define LOAD_GLCD
#define LOAD_FONT2
#define LOAD_FONT4
#define LOAD_FONT6
#define LOAD_FONT7
#define LOAD_FONT8
#define LOAD_GFXFF
#define SPI_FREQUENCY  40000000
#define SPI_READ_FREQUENCY  20000000
#define SPI_TOUCH_FREQUENCY  2500000
```

Next, configure the IDE for your Blue Pill. I am currently using IDE version 1.8.13.

a) Choose Tools-> Board -> STM32 boards (select from submenu) -> Generic STM32F1.
b) Tools -> Board -> Board Part Number -> Blue Pill F103CB (or C8 with 128k)
c) Upload method -> STM32CubeProgrammer (SWD)

For programming you will need an ST-LINK v2-compatible dongle, widely available on eBay and Amazon.

STEP 3: HELLO, WORLD

The following sketch will verify that your hardware is in working order, the STM32 package is correctly installed, the display library is correctly configured, and that you are able to upload code:

```
#include <TFT_eSPI.h>
#define TITLE "Hello, World!"

TFT_eSPI tft = TFT_eSPI();                          // display object
```

```
void setup() {
  tft.init();
  tft.setRotation(1);                          // portrait screen orientation
  tft.fillScreen(TFT_BLUE);                    // start with empty screen
  tft.setTextColor(TFT_YELLOW);                // yellow on blue text
  tft.drawString(TITLE,50,50,4);               // display text
}

void loop() {
}
```

If you see “Hello, World” on your display, you are ready to continue. If the display is upside-down, physically rotate the display or change the setRotation() parameter from 1 to 3.

STEP 4: RESPONDING TO TOUCH EVENTS

The XPT2046 is a *resistive* touch controller. The connected touch panel consists of two transparent, resistive-coated sheets, that are separated by a small air gap. When you touch the panel, the two sheets make contract, and current passes between the sheets. The resistance across the panel electrodes varies according to the position of the point of contact. Analog-to-digital converters in the XPT2046 measure the electrode voltages and convert them to an X,Y panel coordinate. The controller also determines a Z coordinate, which correlates with the amount of pressure applied.

There are many stand-alone Arduino libraries available for the XPT2046. A good example is Paul Stoffregen’s XPT2046_Touchscreen on Github. However, since touchscreen support is built into TFT_eSPI, no additional libraries are required here.

A simple way to check for a touch event is to monitor the pressure (Z) coordinate, which is returned by the function tft.getTouchRawZ. If the returned value is greater than some threshold, then a touch has occurred:

```
bool touched() {                               // true if user touched screen
  const int threshold = 500;                   // ignore light touches
  return tft.getTouchRawZ() > threshold;
}
```

Whenever touched() returns true, you can query the controller for the (x,y) coordinate where the touch occurred. Keep in mind that the analog signals are imprecise, and while the underlying code may average a few samples to improve accuracy, sometime the returns values are incorrect. You may wish to apply some simple error checking to the returned values. Consider the following:

```
void checkForTouch() {
  short unsigned int x, y;
  if (touched()) {                             // did user touch the display?
    tft.getTouch(&x,&y);                       // get touch coordinates
    markLocation(x,y);                         // do something with it!
    delay(300);                                // and wait (touch debouncer)
  }
}
```

The function tft.getTouch() is used to return the x,y coordinate of the touch. A small delay (300 milliseconds) is added so that we are not repeatedly acting on the same touch event.

The following is a complete sketch for testing your touch controller. The source is available on GitHub here. It places a small yellow circle wherever you touch the screen.

```
#include <TFT_eSPI.h>                          // https://github.com/Bodmer/TFT_eSPI
TFT_eSPI tft = TFT_eSPI();                     // display object

bool touched() {                               // true if user touched screen
  const int threshold = 500;                   // ignore light touches
  return tft.getTouchRawZ() > threshold;
}

void markLocation(int x, int y) {
  tft.fillCircle(x,y,6,TFT_YELLOW);            // and place a small circle there
}

void checkForTouch() {
  short unsigned int x, y;
  if (touched()) {                             // did user touch the display?
    tft.getTouch(&x,&y);                       // get touch coordinates
    markLocation(x,y);                         // show it on the screen
    delay(300);                                // and wait (touch deboucer)
  }
}

void setup() {
  tft.init();
  tft.setRotation(1);                          // portrait screen orientation
  tft.fillScreen(TFT_BLACK);                   // clear the screen
}

void loop() {
  checkForTouch();                             // test the touch function!
}
```

Load this sketch and touch the screen. Do you see small circles where you touched? If you do, your touch controller is working correctly. If not, verify your wiring and TFT_eSPI User_Setup.h file.

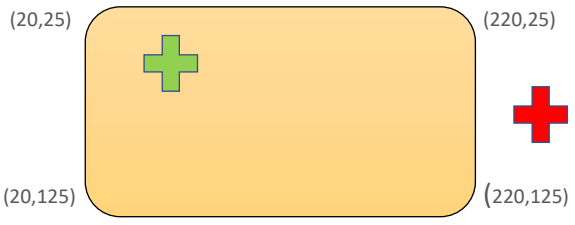
.

STEP 5: BUTTONS and REGIONS

Getting the coordinates of a touch press is great, but what we really want to do is determine if a particular object on the screen, such as a button or other control, was touched. To do this we define a "region" on the screen. If the point of contact is within the button's region, then the button was touched:

Region: X_0=20, Y_0=25, Width=200, Height=100

Touch Points: G at (50,40) and R at (240,80)



Consider the orange rectangular region above that is 200 pixels wide and 100 pixels tall, with its top/left corner at coordinate (20,25). How do we determine if a given touch point is in the region? A point is in the region if *both* of the following conditions are true:

a. its X coordinate is between X_0 and (X_0 + Width)
b. its Y coordinate is between Y_0 and (Y_0 + Height)

In the example above, touch point Green at (50,40) is in the region because 50 is between 20 and 220; and 40 is between 25 and125. Touch point Red is outside the region because its X coordinate, 240, is not between 20 and 220.

To put these ideas in code, let's create a region using a C data type called a struct. Then create the region, using the above coordinates:

```
typedef struct {
  int x;                                        // x position (left side of rectangle)
  int y;                                        // y position (top of rectangle)
  int w;                                        // width, such that right = x+w
  int h;                                        // height, such that bottom = y+h
} region;

region rOrange = {20,25,200,100};               // example region
```

Now, to determine if a given point is within this region, create a function inRegion() that accepts a region and a point, and returns true if the point is within the region:

```
boolean inRegion (region b, int x, int y) {     // true if regsion contains point
(x,y)
  if ((x < b.x ) || (x > (b.x + b.w)))          // x coordinate out of bounds?
    return false;                               // if so, leave
  if ((y < b.y ) || (y > (b.y + b.h)))          // y coordinate out of bounds?
    return false;                               // if so, leave
  return true;                                  // x & y both in bounds
}
```

The two highlighted lines correspond to the two conditions from above. The X coordinate is evaluated, then the Y. If either is out of bounds, the function returns false. Otherwise, the function returns true. The double parallel line symbol "||" is the OR operator, as in "if a *or* b are true".

Now, modify the original markLocation() routine, so that it the color of the marker circle is green if it is inside the region, and red if it outside:

```
void markLocation(int x, int y) {
  int color = (inRegion(rOrange,x,y))?          // is x,y within the region?
     TFT_GREEN:TFT_RED;                         // if yes, green circle. if no, red
circle
  tft.fillCircle(x,y,6,color);                  // place a small circle at x,y
}
```

The ternary operator "?" is used here, which returns the color green if inRegion() is true, and red if false. The complete TouchDemo2 sketch is on Github.

STEP 6: MULTIPLE REGIONS

A screen will typically contain more than one button or control, and we will need to determine which control was touched. If there are just a few screen regions to consider, the simplest way is to test them individually in a compound if..else statement, like this:

```
void checkForTouch() {
  uint16_t x, y;
  if (touched()) {                              // did user touch the display?
    tft.getTouch(&x,&y);                        // get touch coordinates
    if (inRegion(region1,x,y))                  // was time touched?
      touchedR1(x,y);
    else if (inRegion(region2,x,y))             // was location touched?
      touchedR2(x,y);
```

```
    else if (inRegion(region3,x,y))                // was AM/PM touched?
      touchedR3(x,y);
    delay(300);                                    // touch debouncer
  }
}
```

Each region is tested, and if the point is within the region, a function corresponding to that region is called. But if there are many regions to test, using an array might make more sense. Here is the C construct for initializing an array of regions, and a function to display all of the regions on the screen:

```
#define ELEMENTS(x) (sizeof(x) / sizeof(x[0]))     // Macro to determine #elements/array

region rScreen[] = {                               // Example multiple-region screen:
  {1,1,320,35},                                   // Title bar region
  {20,50,200,100},                                // Time region
  {240,60,80,35},                                 // Time Zone region
  {240,110,80,35},                                // AM/PM region
  {1,180,140,40},                                 // Clock status1 region
  {180,180,140,40}                                // Clock status2 region
};

void fillRegion (int ID, int color) {              // display a region on screen
  tft.fillRect(rScreen[ID].x,rScreen[ID].y,
    rScreen[ID].w, rScreen[ID].h, color);
}

void displayScreenRegions() {                      // display all regions on screen
  for (int i=0; i<ELEMENTS(rScreen); i++)          // For each region in the array
    fillRegion(i,TFT_BLUE);                        // Make it blue.
}
```

To determine if a touch event falls within one of these regions, construct a for-loop which tests each region in the array, and returns with the index of the one which matches the touch. If the for-loop completes, then nothing matched, so return -1 (no match):

```
int regionID(int x, int y) {
  for (int i=0; i<ELEMENTS(rScreen); i++)          // for each region in the array
    if (inRegion(rScreen[i],x,y))                  // is the point in this region?
      return i;                                    // yes, so return its index
  return -1;                                       // finished search, didn't find it.
}
```

To handle a touch event, call regionID with the touch coordinates, and then create a switch-case statement to handle each of the regions:

```
void handleTouchEvent (int x, int y) {
  int ID = regionID(x,y);                          // which region was pressed?
  switch (ID) {
    case 0: ;                                      // code for region #0
    case 1: ;                                      // code for region #1
    case 2: ;                                      // code for region #2
  }
}
```

The full code for this third and final touch demo is found on GitHub here.

73,

Bruce.