|  |  |
| --- | --- |
| Adafruit TouchScreen Documentation | Abstract  Documentation on how to use Adafruit’s 2.8 capacitive touch screen. Wiring and software explanation included.  Bryant Gonzaga & Victor Tellez |

Table of Contents

[1 Introduction 2](#_Toc508840707)

[2 Definitions 2](#_Toc508840708)

[3 Section A: Hardware 2](#_Toc508840709)

[3.1 Pin Out 2](#_Toc508840710)

[3.2 Wiring 3](#_Toc508840711)

[3.3 Interrupt Pin 3](#_Toc508840712)

[4 Section B: Software 3](#_Toc508840713)

[4.1 High Level Overview of Directories 3](#_Toc508840714)

[4.2 AdafruitTFTSPIDriver 4](#_Toc508840715)

[4.2.1 AVRPin 5](#_Toc508840716)

[4.2.2 SPI Driver 5](#_Toc508840717)

[4.2.3 Adafruit Provided Constants 6](#_Toc508840718)

[4.2.4 AdafruitTFTSPIDriver 6](#_Toc508840719)

[4.2.5 AdafruitTFTButton 12](#_Toc508840720)

[4.3 AdafruitTFTI2CDriver 12](#_Toc508840721)

[4.3.1 I2C Driver 13](#_Toc508840722)

[4.3.2 AdafruitTSPoint 14](#_Toc508840723)

[4.3.3 AdafruitTFTI2CDriver 14](#_Toc508840724)

[4.4 Using the TFT Drivers 14](#_Toc508840725)

[5 Section C: Example Projects 15](#_Toc508840726)

[5.1 AdafruitTFTLibTest 16](#_Toc508840727)

[5.2 AdafruitTFTKeyPad 16](#_Toc508840728)

[5.3 AdafruitTFTPaint 16](#_Toc508840729)

# Introduction

This document contains information on Adafruit’s 2.8" TFT LCD with Capacitive Touch Breakout Board w/MicroSD Socket. It has information regarding the hardware and the software library, adapted to work with AVR’s 8-bit microcontroller.

The document is partitioned into three sections. The first section describes the hardware and how to wire the TFT. The following section explains the software needed to control the TFT. Finally, the last section provides some examples on how to use the software along with example wiring with the ATmega324.

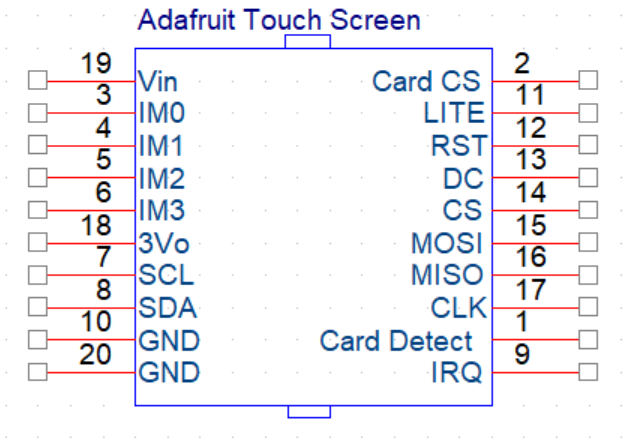
# Acronyms

|  |  |
| --- | --- |
| TFT | Thin Film Transistor. Throughout this document the touch screen module will be referred to as “the TFT.” |
| MOSI | Master Out Slave In |
| MISO | Master In Slave Out |
| sclk | Slave Clock |
| LCD | Liquid Crystal Display |
| SPI | Serial Peripheral Interface; communication used to send data to TFT |
| I2C | Inter-Integrated Circuit Bus; communication used to read from the capacitive touch controller on the TFT |

# 

# Section A: Hardware

Adafruit’s TFT provides two modes of operation; it provides an 8-bit parallel and an SPI mode. This document only describes how to use the SPI mode.



## Pin Out

|  |  |  |
| --- | --- | --- |
| # | Name | Description |
| 1 | Card Detect | Grounded when SD card is inserted |
| 2 | Card CS | Used to select SD card |
| 3 | IM0 | Interface set control pin |
| 4 | IM1 | Interface set control pin |
| 5 | IM2 | Interface set control pin |
| 6 | IM3 | Interface set control pin |
| 7 | SCL | I2C slave clock pin |
| 8 | SDA | I2C data pin |
| 9 | IRQ | Interrupt request pin. When touched pin is pulled low |
| 10 | GND | Ground pin |
| 11 | Lite | Back light control. By default, it is pulled high |
| 12 | RST | Reset pin; active low |
| 13 | DC | Data or Command pin; logic 1 indicates data |
| 14 | CS | Chip select pin; active low |
| 15 | MOSI | SPI Master Out Slave In pin |
| 16 | MISO | SPI Master In Slave Out pin |
| 17 | CLK | SPI slave clock pin |
| 18 | 3Vo | 3-volt output pin |
| 19 | Vin | Input voltage; 5 V |
| 20 | GND | Ground pin |

## Wiring

There are 4 pins needed for the SPI mode:

* Pin 17 – CLK
* Pin 15 – MOSI
* Pin 14 – CS
* Pin 13 – DC

The microcontroller is the master SPI and the TFT is the slave SPI in this case.

There are 2 pins needed for the capacitive touch controller that uses I2C:

* Pin 7 – SCL
* Pin 8 – SDA

IM1, IM2, and IM3 should all be pulled up to 3.3 V (**NOT 5 V**). This will indicate to the TFT that it should operate in the SPI mode. Jumpers on the back of the board can be soldered together to accomplish this. If users wish to not solder they can simply tie them to the 3Vo pin of the board.

## Interrupt Pin

Pin 9, the IRQ pin, is the interrupt pin. When the TFT is touched, this pin is pulled low at an average frequency of 77 Hz (for as long as the screen is touched); this pin operates on 3.3 V.

When using the interrupt pin with a 5 V system make sure to use logic shifters or that the microcontroller will read the 3.3 V as a logic 1.

# 

# Section B: Software

## High Level Overview of Directories

There are 5 sub-directories within the Adafruit\_TFT\_Lib folder.

1. AdafruitTFTSPIDriver
2. AdafruitTFTI2CDriver
3. AdafruitTFTKeyPad
4. AdafruitTFTLibTest
5. AdafruitTFTPaint

Directories AdafruitTFTSPIDriver and AdafruitTFTI2CDriver are part of the library used to control the touch screen. Directories AdafruitTFTKeyPad, AdafruitTFTLibTest, and AdafruitTFTPaint are all example programs.

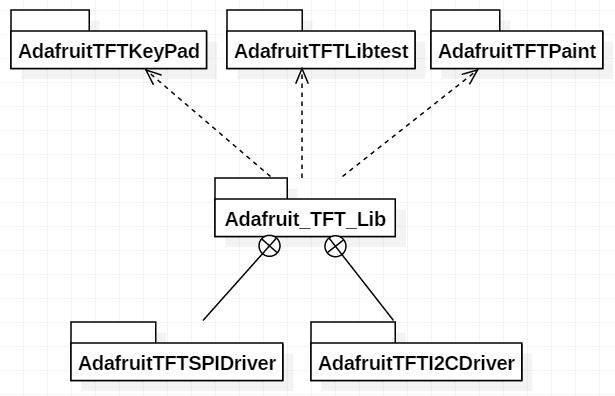


Figure 1: Directory Overview

## AdafruitTFTSPIDriver

Figure 2 shows all the .h and .c files that can be found inside AdafruitTFTSPIDriver.

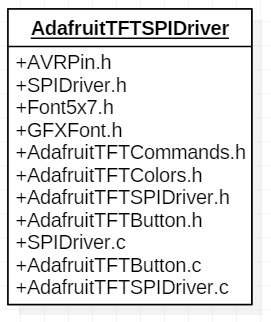


Figure 2: AdafruitTFTSPIDriver Directory

This folder contains all files needed to communicate with the TFT via SPI. With this directory alone, users can send data to the TFT. Users can draw shapes and text of assorted colors on the screen.

**NOTE:** this folder does NOT contain any support to receive data from the TFT. If users wish to receive data regarding whether the TFT was touch and where it was touched, then they will have to use the AdafruitTFTI2CDriver directory (explained in detail later).

### AVRPin

AVRPin is a simple structure used to store information in regards to a specific Port on an AVR 8-bit microcontroller. The AVRPin has three uint8\_t pointers. One for each of the associated registers of a Port (DDRx, PORTx, PINx). The AVRPin also contains an 8-bit mask. This structure can be used to specify a specific pin on the microcontroller. This structure can be found in “AVRPin.h”.

### SPI Driver

This is the lowest level code that is hardware dependent. SPIDriver.h contains functions to initialize and use the SPI hardware on the microcontroller.

/\*\*

\* Initializes the SPI hardware to operate in the Master mode, to

\* send the most significant bit first, and to have slave clock

\* frequency of SOME Hz.

\* Initializes the ss, sclk, and mosi pins as outputs

\* and the miso pin as input. It also drives the mosi and sclk pins

\* low and the ss pin high.

\* @param ss Slave Select. Pointer to AVRPin struct

\* @param sclk Slave Clock. Pointer to AVRPin struct

\* @param mosi Master Out Slave In. Pointer to AVRPin struct

\* @param miso Master In Slave Out. Pointer to AVRPin struct

\* @see AVRPin

\*/

void spiMasterInit(AVRPin\* ss, AVRPin\* sclk, AVRPin\* mosi, AVRPin\* miso);

/\*\*

\* Initializes the SPI hardware to operate in the Slave mode.

\* Firstly, it initializes the ss, sclk, and mosi pins as inputs and

\* the miso pin as output.

\* @param ss Slave Select. Pointer to AVRPin struct

\* @param sclk Slave Clock. Pointer to AVRPin struct

\* @param mosi Master Out Slave In. Pointer to AVRPin struct

\* @param miso Master In Slave Out. Pointer to AVRPin struct

\* @see AVRPin

\*/

void spiSlaveInit(AVRPin\* ss, AVRPin\* sclk, AVRPin\* mosi, AVRPin\* miso);

/\*\*

\* Empty for now.

\*/

void spiSetClkPrescalar(*uint8\_t* prescalar);

/\*\*

\* Initializes the SPI transaction by driving the slave select pin

\* low.

\* @param ss AVRPin structure for the slave select pin.

\*/

void spiStartTransmission(AVRPin\* ss);

/\*\*

\* Transmits a single byte of data through hardware SPI.

\* Will return once data has been transmitted

\* @param data byte of data to be transmitted.

\*/

void spiMasterTransmit(*uint8\_t* data);

/\*\*

\* Transmits a two bytes of data through hardware SPI.

\* Will return once data has been transmitted

\* @param data 16-bit data to be transmitted.

\*/

void spiMasterTransmit16(*uint16\_t* data);

/\*\*

\* Transmits a four bytes of data through hardware SPI.

\* Will return once data has been transmitted

\* @param data 32-bit data to be transmitted.

\*/

void spiMasterTransmit32(*uint32\_t* data);

/\*\*

\* Ends the SPI transaction by driving the slave select pin high.

\* @param ss AVRPin structure for the slave select pin.

\*/

void spiEndTransmission(AVRPin\* ss);

/\*\*

\* Used to read data from a master through SPI.

\* @return a byte of data received from master.

\*/

*uint8\_t* spiSlaveReceive();

### Adafruit Provided Constants

The following files contain constants provided by Adafruit.

* Font5x7.h – Contains the bit patterns for default font.
* GFXFont.h – Structure that describes a custom font.
* AdafruitTFTCommands.h – Commands for the TFT that can be sent through SPI.
* AdafruitTFTColors.h – Color definitions.

### 

### AdafruitTFTSPIDriver

AdafruitTFTSPIDriver.h contains all the functions needed to control the TFT. This is hardware independent. It assumes a working SPI Driver for the specific microcontroller. This file also contains TFTVars structure.

/\*\*

\* Contains all the variables associated with the touchscreen.

\*/

typedef struct TFTVars

{

*uint16\_t* width; /\*\*< Screen width according to rotation. >\*/

*uint16\_t* height; /\*\*< Screen height according to rotation. >\*/

*uint16\_t* cursor\_x; /\*\*< Cursor x-position used in some methods

to specify where to display. >\*/

*uint16\_t* cursor\_y; /\*\*< Cursor y-position used in some methods

to specify where to display. >\*/

*uint8\_t* rotation; /\*\*< Current orientation of the screen >\*/

*uint8\_t* textSize; /\*\*< Multiplier for the font >\*/

*uint16\_t* textColor; /\*\*< Color of text >\*/

*uint16\_t* textBGColor;

bool wrap; /\*\*< If set, 'wrap' text at right edge of

display >\*/

bool cp437; /\*\*< If set, use correct CP437 char set

(default is off) >\*/

GFXfont \*gfxFont; /\*\*< Pointer to a GFXfont structure for

custom fonts. >\*/

AVRPin\* cs; /\*\*< Chip Select. >\*/

AVRPin\* dc; /\*\*< Data or Command. Low indicates command

and high indicates data. >\*/

AVRPin\* rst; /\*\*< Reset. TFT has a active low reset. >\*/

AVRPin\* mosi; /\*\*< Master Out Slave In SPI Pin. >\*/

AVRPin\* miso; /\*\*< Master In Slave Out SPI Pin. >\*/

AVRPin\* sclk; /\*\*< Slave Clock SPI Pin. >\*/

} TFTVars;

The following functions are all located in the “AdafruitTFTSPIDriver.h”. They can be split up in the following way:

#### Initialization and Control

The TFT should be initialized by calling the initTFT(…) function. Make sure to initialize the appropriate variables in TFTVars. The minimal initialization is shown later under section 4.4.

The orientation of the screen is determined by the rotation variable. This variable can be modified using the setRotationTFT(…); this function also updates other necessary variables. Rotation settings are as follows:

Rotation 0: Rotation: 2

x y 0

0 y x

Rotation: 1 Rotation: 3

0 x y

y x 0

When using the library to draw text, shapes, or buttons onto screen, it assumes that x grows to the right and y grows downward. When getting a point from the TFT, the point returned is given in terms of rotation 2.

/\*\*

\* Initializes the SPI hardware with SUCH PARAMETERS. As well as

\* initiating the touchscreen.

\* @param var Pointer to TFTVars structure that contains the

\* current variables used for the touchscreen.

\*/

void initTFT(TFTVars\* var);

/\*\*

\* Sends a command to the touchscreen through SPI.

\* @param cmd 8-bit command

\* @param var pointer to TFTVars structure.

\*/

void writeCommandTFT(*uint8\_t* cmd, TFTVars\* var);

/\*\*

\* Sets the rotation of the screen. The screen can have four

\* different rotations. Updates the width and height according to

\* rotation.

\* Rotation 0 has 0 for x and y at the bottom left corner. Where x

\* is height and y is width. Each subsequent rotation turns the axes

\* clock wise. Where rotation 1 has the 0 at the upper left corner

\* and the x-axis as width and y-axis as height.

\* @param r rotation can be from 0 to 3.

\* @param var pointer to TFTVars structure

\*/

void setRotationTFT(*uint8\_t* r, TFTVars\* var);

/\*\*

\* Invert display according to the boolean value of <code> i </code>

\* @param i boolean value; true to invert display.

\* @param var pointer to TFTVars structure.

\*/

void invertDisplay(bool i, TFTVars\* var);

/\*\*

\* Fills the whole screen with the specified color

\* @param color Color the screen is to be filled with.

\* @param var Pointer to TFTVars structure.

\*/

void fillScreenTFT(*uint16\_t* color, TFTVars\* var);

#### Lines

/\*\*

\* Draws a vertical line on the screen of length <code> h </code>

\* starting form ( <code> x </code>, <code> y </code>).

\* @param x x starting position.

\* @param y y starting position.

\* @param h Height of the line.

\* @param color Color of line.

\* @param var Pointer to TFTVars structure.

\*/

void drawVLineTFT(*int16\_t* x, *int16\_t* y, *int16\_t* h, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Draws a horizontal line on the screen of length <code> w </code>

\* starting form ( <code> x </code>, <code> y </code>).

\* @param x x starting position.

\* @param y y starting position.

\* @param w Width of the line.

\* @param color Color of line.

\* @param var Pointer to TFTVars structure.

\*/

void drawHLineTFT(*int16\_t* x, *int16\_t* y, *int16\_t* w, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Draws a line on the screen of from (x0, y0) to (x1, y1)

\* @param x0 x starting position.

\* @param y0 y starting position.

\* @param x1 x ending position.

\* @param y1 y ending position.

\* @param color Color of line.

\* @param var Pointer to TFTVars structure.

\*/

void drawLineTFT(*int16\_t* x0, *int16\_t* y0, *int16\_t* x1, *int16\_t* y1, *uint16\_t* color, TFTVars\* var);

#### Rectangles

/\*\*

\* Draws a filled rectangle with dimensions <code> w </code> and

\* <code> h </code>.

\* @param x x starting position

\* @param y y starting position

\* @param w Width of rectangle

\* @param h Height of rectangle

\* @param color Color of rectangle

\* @param var Pointer to TFTVars structure

\*/

void fillRect(*int16\_t* x, *int16\_t* y, *int16\_t* w, *int16\_t* h, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Draws a hollow rectangle with dimensions <code> w </code> and

\* <code> h </code>.

\* @param x x starting position

\* @param y y starting position

\* @param w Width of rectangle

\* @param h Height of rectangle

\* @param color Color of rectangle

\* @param var Pointer to TFTVars structure

\*/

void drawRect(*int16\_t* x, *int16\_t* y, *int16\_t* w, *int16\_t* h, *uint16\_t* color, TFTVars\* var);

#### Circles

/\*\*

\* Draws a hollow circle with radius <code> r </code>.

\* @param x0 x center position.

\* @param y0 y center position.

\* @param r Circle radius.

\* @param color Circle color

\* @param var Pointer to TFTVars structure

\*/

void drawCircle(*int16\_t* x0, *int16\_t* y0, *int16\_t* r, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Draws a filled circle with radius <code> r </code>

\* @param x x center position.

\* @param y y center position.

\* @param r Circle radius.

\* @param color Circle color

\* @param var Pointer to TFTVars structure

\*/

void fillCircle(*int16\_t* x, *int16\_t* y, *int16\_t* r, *uint16\_t* color, TFTVars\* var);

#### Round Rectangles

/\*\*

\* Draws a hollow rectangle with dimensions <code> w </code> and

\* <code> h </code> and rounded corners.

\* @param x x starting position

\* @param y y starting position

\* @param w Width of rectangle

\* @param h Height of rectangle

\* @param radius How rounded out corners should be

\* @param color Color of rectangle

\* @param var Pointer to TFTVars structure

\*/

void drawRoundRect(*int16\_t* x, *int16\_t* y, *int16\_t* w, *int16\_t* h, *int16\_t* radius, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Helper method to help with drawing circles. Used for round

\* rectangles.

\*/

void drawCircleHelper(*int16\_t* x0, *int16\_t* y0, *int16\_t* r, *uint8\_t* cornername, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Draws a filled rectangle with dimensions <code> w </code> and

\* <code> h </code> and rounded corners.

\* @param x x starting position

\* @param y y starting position

\* @param w Width of rectangle

\* @param h Height of rectangle

\* @param radius How rounded out corners should be

\* @param color Color of rectangle

\* @param var Pointer to TFTVars structure

\*/

void fillRoundRect(*int16\_t* x, *int16\_t* y, *int16\_t* w, *int16\_t* h, *int16\_t* radius, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Helper method to help with drawing circles. Used for round

\* rectangles.

\*/

void fillCircleHelper(*int16\_t* x0, *int16\_t* y0, *int16\_t* r, *uint8\_t* cornername, *int16\_t* delta, *uint16\_t* color, TFTVars\* var);

#### Triangles

/\*\*

\* Draws a hollow triangle with the three specified corners.

\* @param x0 x-position of corner 0.

\* @param y0 y-position of corner 0.

\* @param x1 x-position of corner 1.

\* @param y1 y-position of corner 1.

\* @param x2 x-position of corner 2.

\* @param y2 y-position of corner 2.

\* @param color Color of triangle.

\* @param var Pointer to TFTVars structure.

\*/

void drawTriangle(*int16\_t* x0, *int16\_t* y0, *int16\_t* x1, *int16\_t* y1, *int16\_t* x2, *int16\_t* y2, *uint16\_t* color, TFTVars\* var);

/\*\*

\* Draws a filled triangle with the three specified corners.

\* @param x0 x-position of corner 0.

\* @param y0 y-position of corner 0.

\* @param x1 x-position of corner 1.

\* @param y1 y-position of corner 1.

\* @param x2 x-position of corner 2.

\* @param y2 y-position of corner 2.

\* @param color Color of triangle.

\* @param var Pointer to TFTVars structure.

\*/

void fillTriangle(*int16\_t* x0, *int16\_t* y0, *int16\_t* x1, *int16\_t* y1, *int16\_t* x2, *int16\_t* y2, *uint16\_t* color, TFTVars\* var);

#### Text

/\*\*

\* Draws the specified character onto the screen

\* @param x x coordinate for where to place character.

\* @param y y coordinate for where to place character.

\* @param c Character to be place. ASCII char.

\* @param color Color of character.

\* @param bg Background color for character.

\* @param size Size of character to be displayed on screen.

\* @param var Pointer to TFTVars data structure.

\*/

void drawChar(*int16\_t* x, *int16\_t* y, unsigned char c, *uint16\_t* color, *uint16\_t* bg, *uint8\_t* size, TFTVars\* var);

/\*\*

\* Draws the specified character onto the screen. This method uses

\* the values stored in the TFTVars structure to format the character

\* accordingly. Users should pass a pointer that points to a

\* structure with already set values for <code> cursor\_x </code>,

\* <code> cursor\_y </code>, <code> textSize </code>, <code> textColor

\* </code>, and <code> textBGColor </code>. Furthermore, this

\* function advances the cursor one character after its been written.

\* @param c Character to be place. ASCII char.

\* @param var Pointer to TFTVars data structure.

\*/

void write(*uint8\_t* c, TFTVars\* var);

/\*\*

\* Used by both the PROGMEM- and RAM-resident

\* getTextBounds() functions.

\*/

void charBounds(char c, *int16\_t* \*x, *int16\_t* \*y, *int16\_t* \*minx, *int16\_t* \*miny, *int16\_t* \*maxx, *int16\_t* \*maxy, TFTVars\* var);

/\*\*

\* Pass string and a cursor position, returns UL corner and W, H.

\*/

void getTextBounds(char \*string, *int16\_t* x, *int16\_t* y, *int16\_t* \*x1, *int16\_t* \*y1, *uint16\_t* \*w, *uint16\_t* \*h, TFTVars\* var);

### 

### AdafruitTFTButton

The “AdafruitTFTButton.h” file contains support for drawing buttons on the screen. The TSButtonVars structure should be initialized with the desired parameters for a given button. The drawButtonTFT(…) is used to actually draw the button on the screen. The buttonContainsPointTFT(…) function can be used when a user touches the screen to test whether the user touch a given button.

/\*\*

\* A collection of variables needed to create a button.

\*/

typedef struct TSButtonVars

{

*int16\_t* x; /\*\*< x-coordinate of button >\*/

*int16\_t* y; /\*\*< y-coordinate of button >\*/

*uint16\_t* w; /\*\*< Button width. >\*/

*uint16\_t* h; /\*\*< Button height >\*/

*uint8\_t* size; /\*\* Size of button \*/

*uint16\_t* outlineColor; /\*\*< Outline of button >\*/

*uint16\_t* fillColor; /\*\*< Button color >\*/

*uint16\_t* textColor; /\*\*< Color of the button label >\*/

char label; /\*\*< Single ASCII character to be

displayed on button >\*/

} TSButtonVars;

/\*\*

\* Draws the button specified by the TSButtonVars structure.

\* @param button Pointer to button variable used to draw button.

\* @param tftVars Pointer to TFT variables structure.

\*/

void drawButtonTFT(TSButtonVars\* button, TFTVars\* tftVars);

/\*\*

\* Is used to check if a point is within a given button

\* @param x x-position.

\* @param y y-position.

\* @param button Pointer to button in question.

\* @return True if (x, y) are within the button's bounds.

\*/

bool buttonContainsPointTFT(*int16\_t* x, *int16\_t* y, TSButtonVars\* button);

## AdafruitTFTI2CDriver

This directory contains all files needed to communicate with the TFT via I2C. This driver can be used to get data regarding where the user last touched the TFT.

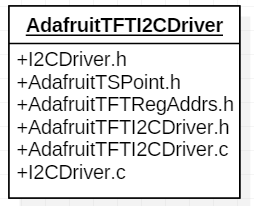


Figure 3: AdafruitTFTI2CDriver Directory

### I2C Driver

The “I2CDriver.h” contains the lowest level code that is hardware dependent. The following functions are declared in this header file:

/\*\*

\* Sends a start condition.

\* @return true if slave acknowledges start condition.

\*/

bool i2cStartCondition();

/\*\*

\* Send address of slave and write bit.

\* @param addr 7-bit slave address that is right aligned

\* @return true if slave acknowledges call.

\*/

bool i2cBeginWrite(*uint8\_t* addr);

/\*\*

\* Send address of slave and read bit

\* @param addr 7-bit slave address that is right aligned

\* @return true if slave acknowledges call.

\*/

bool i2cBeginRead(*uint8\_t* addr);

/\*\*

\* Transmits a byte of data. Start Condition, Slave Address, and

\* Register Address should have all been sent before sending data.

\* This method is also used to send register address.

\* @param data Byte of data or 8-bit register address

\* @return true if data was send and acknowledged by slave.

\*/

bool i2cTransmit(*uint8\_t* data);

/\*\*

\* Reads a specified number of bytes from the slave.

\* @param dataAddr Pointer to where the read data is to be

\* stored.

\* @param numOfBytes Number of bytes to be read from slave.

\* @return true if data was read successfully.

\*/

bool i2cReadData(*uint8\_t*\* dataAddr, *uint8\_t* numOfBytes);

/\*\*

\* Stop Condition is used to end transmission

\*/

void i2cStopCondition();

### 

### AdafruitTSPoint

The “adafruitTFPoint.h” file contains the TS\_Point struct. This is a simple structure with three variables (x, y, and z). Each is a 16-bit number. Together they represent where the TFT was touched.

### AdafruitTFTI2CDriver

The “AdafruitTFTI2CDriver.h” contains all high level functions to communicate wit the TFT via I2C. Before talking with the TFT via I2C the initCPTTS(…) function should be called.

/\*\*

\* Initialize capacitive touchscreen with specified threshold

\* @param thresh Threshold for capacitive touchscreen

\* @return True if touchscreen is initialized properly.

\*/

bool initCPTTS(*uint8\_t* thresh);

/\*\*

\* @return True if the screen was touched. False otherwise.

\*/

bool touched();

/\*\*

\* @return The last touched point in form of TS\_Point structure.

\*/

TS\_Point getPoint(TSVars\* var);

The getPoint(…) function can be called after the microcontroller receives an interrupt from the TFT. This function will return the last touched point. The touch() function can be used to poll the TFT.

## 

## Using the TFT Drivers

To get the full functionality of the TFT, users should include both the AdafruitTFTSPIDriver and the AdafruitTFTI2CDriver. The following sample code shows how users should go about initializing the TFT.

//INITIALIZE PORTS FOR TFT

AVRPin cs;

cs.DDRx = &DDRB;

cs.PORTx = &PORTB;

cs.PINx = &PINB;

cs.mask = 0x10;

tftVars.cs = &cs;

AVRPin dc;

dc.DDRx = &DDRB;

dc.PORTx = &PORTB;

dc.PINx = &PINB;

dc.mask = 0x08;

tftVars.dc = &dc;

AVRPin rst;

rst.DDRx = &DDRB;

rst.PORTx = &PORTB;

rst.PINx = &PINB;

rst.mask = 0x10;

tftVars.rst = &rst;

AVRPin sclk;

sclk.DDRx = &DDRB;

sclk.PORTx = &PORTB;

sclk.PINx = &PINB;

sclk.mask = 0x80;

tftVars.sclk = &sclk;

AVRPin mosi;

mosi.DDRx = &DDRB;

mosi.PORTx = &PORTB;

mosi.PINx = &PINB;

mosi.mask = 0x20;

tftVars.mosi = &mosi;

AVRPin miso;

miso.DDRx = &DDRB;

miso.PORTx = &PORTB;

miso.PINx = &PINB;

miso.mask = 0x40;

tftVars.miso = &miso;

//INITIALIZE TOUCH SCREEN

initTFT(&tftVars);

initCPTTS(FT6206\_DEFAULT\_THRESSHOLD);

fillScreenTFT(ILI9341\_BLACK, &tftVars);

This is the bare minimum that must be done before drawing anything on the screen or trying to read any data from the screen.

# Section C: Example Projects

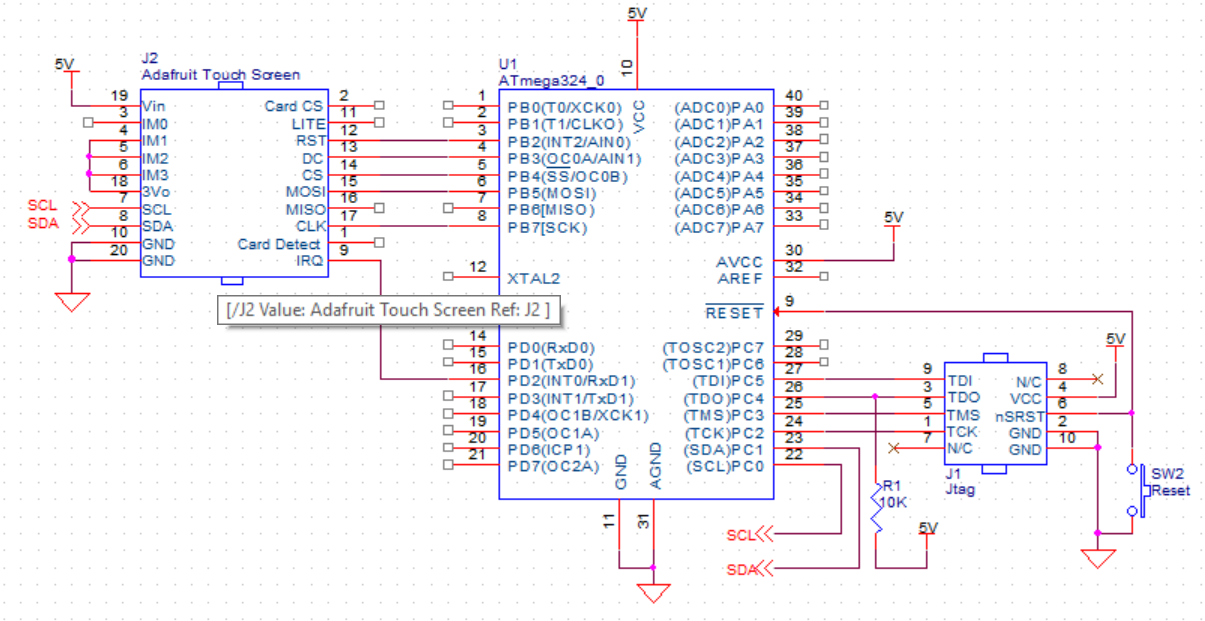


Figure 4: Sample Schematic

The above schematic shows the circuit that was used to test the following test projects.

## AdafruitTFTLibTest

This is a test project that initializes the TFT and then goes through all functions found in the AdafruitTFTSPIDriver. At the end of the main function the program just prints out the same text and while running through all the rotations. Inspect the TFTLibTest.c for more details on exactly how the tests are done. For this project to work only the SPI Driver is needed.

## AdafruitTFTKeyPad

This test program creates sixteen buttons and arranges them on the screen. When a button is pressed the label is printed out on the screen. For this project to work both the SPI and the I2C library are needed. Please refer to the TFTKeyPad.c file to fully see how the program is implemented.

## AdafruitTFTPaint

This test program creates a simplified version of paint. It has colors an assortment of colors on one side of the screen that can be selected. When the user touches the screen, the selected color is printing on the screen. Like a blackboard where different colors can be chosen to draw on the screen. For this project to work both the SPI and the I2C library are needed.

Tutorial for the Sample Keypad Project