



## ThaiEasyElec - 2.8 inch TFT Touch Shield

### English User Manual (V1.0)

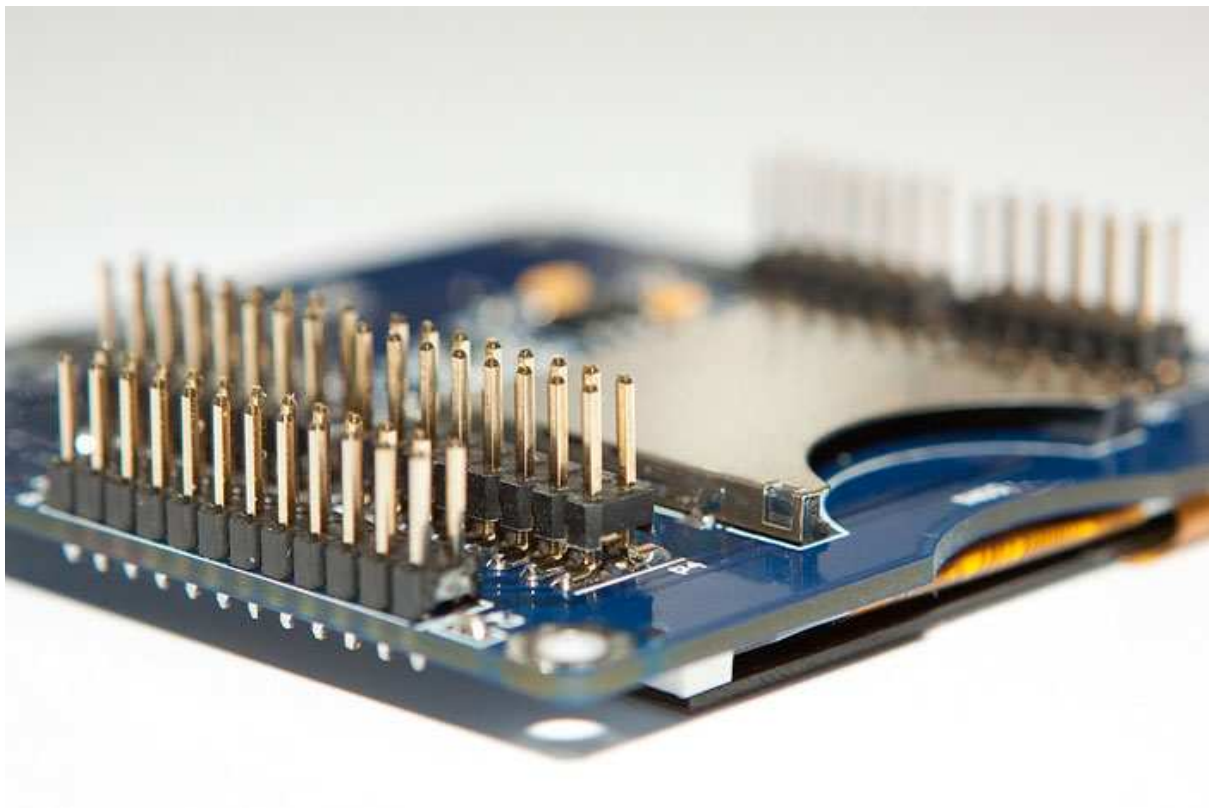
**Enable Your Design**  
**ThaiEasyElec.com**   
On-line Electronics Shop for Embedded System

#### Revision History

Version	Date	Changes
V1.0	14 DEC 2011	Original Version

## Contents

Introductions.....	2
Features .....	2
PIN Definitions .....	3
Principle of operation for 2.8 inch TFT Touch Shield .....	4
Operation for TFT LCD control (8-bit parallel interface).....	4
Access Register of ILI9328 with 8-bit parallel interface.....	5
How to write the value to register of ILI9328 in 8-bit parallel interface operation .....	6
Step by Step for write data to register .....	6
Operating for Touch controller with SPI interface.....	8
Hardware PINs layout .....	10
Arduino connectivity.....	11
Arduino programming.....	11
LCD graphic usage function in TFT_graphics.cpp.....	12
LCD touch screen usage function in TFT_touch.cpp.....	14
Appendix .....	15
Describing an example code “TFT2_8_Shield_Example.pde” .....	15



## Introductions

A 2.8-inch TFT Touch Shield is shield board for Arduino UNO/Arduino Mega with 2.8 inches TFT LCD and resistive touch screen attached. Included SD card socket enable reading picture file from SD card and displaying it easily. Support up to 320x240 pixels resolution and 65K colors.

TFT LCD controls via 8-bit parallel interfaces and touch screen controller (AD7843) controls via SPI interface. Both of them already designed the pin out for attaching with Arduino UNO/Arduino Mega platform.

## Features

- Design for Arduino UNO and Arduino Mega
- 2.8-inch TFT with 320 pixels x 240 pixels resolution and 65K colors (8-bit data parallel interface using 4 pins signal control)
- Resistive touch screen, using AD7843 touch screen controller (4-wire resistive touch screen)
- SD card support (SPI interface), you can store your picture (e.g. background or icon) in external storage
- Logic level using 74LVC245
- Onboard 3.3 volt regulator using NCP1117-3.3
- 5 volt power input (supply from Arduino UNO/Arduino Mega power supply pin)
- 2.8-inch TFT Touch Shield using Arduino UNO pins, Digital I/O pin 0-7 and 10-13 (Digital I/O pin 8 and 9 remaining) and Analog IN pin 0, 1, 2, and 4 (Analog IN pin 3 and 5 remaining)

## PIN Definitions

A 2.8-inch TFT Touch Shield has 24 PINs divided in 2 groups of usage. There are 18 signal PINs for control TFT LCD and 6 signal PINs for control touch screen controller.

### *Signal PINs for control TFT LCD 18 PINs*

Index	Pin Name	Description	Control Signal
1, 24	GND	Ground	-
2, 23	5V	Main voltage supply (+5V)	-
3	BL	Back light control signal	"0": Turn Off Back Light "1": Turn On Back Light
4	RST	Reset signal	"0": Reset LCD "1": No Reset
5	LCD_CS	LCD Chip Select signal	"0": Enable LCD "1": Disable LCD
6	RS	Register Select signal	"0": Data register "1": Control register
7	WR	Write Strobe signal	"0": Write Enable "1": Write Disable
8	RD	Read Strobe signal	"0": Read Enable "1": Read Disable
15-22	D0-D7	Data Bus 8-bit	"0": Low "1": High

### *Signal PINs for Touch Controller 6 PINs*

Index	Pin Name	Description	Control Signal
9	TC_CS	Touch Controller Chip Select	"0": Enable Touch Controller "1": Disable Touch Controller
10	SCLK	Serial Clock	-
11	MISO	Data Out of ADS7843	-
12	MOSI	Data In of ADS7843	-
13	BUSY	Busy Pin of ADS7843	Output "0": with Busy
14	PEN	Pen IRQ of ADS7843	Output "0": with Touch

## Principle of operation for 2.8 inch TFT Touch Shield

There are 2 parts of operation to use 2.8 inch TFT Touch Shield, TFT LCD control (8-bit parallel interface) and Touch controller (SPI Interface). You can see an Appendix for more programming description.

### Operation for TFT LCD control (8-bit parallel interface)

2.8-inch TFT Touch Shield has been designed to use 8-bit parallel mode for control. Then module will use data only 8 bits (PINS) interface. The PINS out from TFT LCD are DB10 to DB17 as schematic in figure 1.

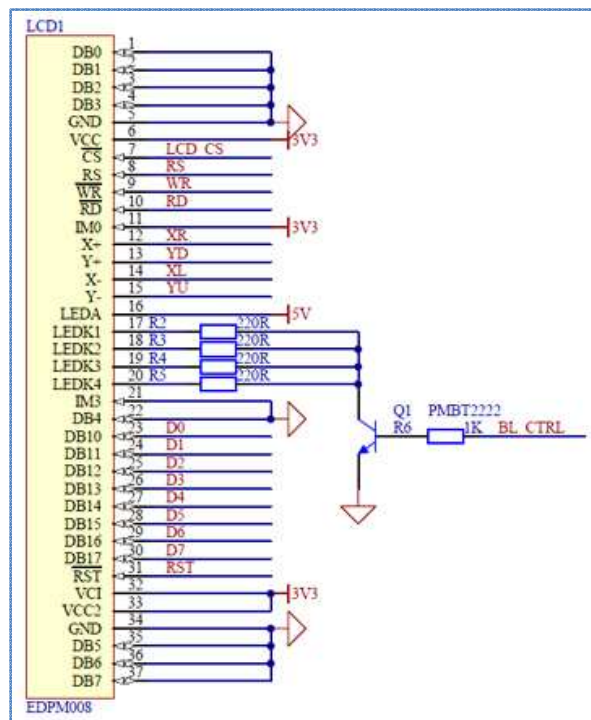


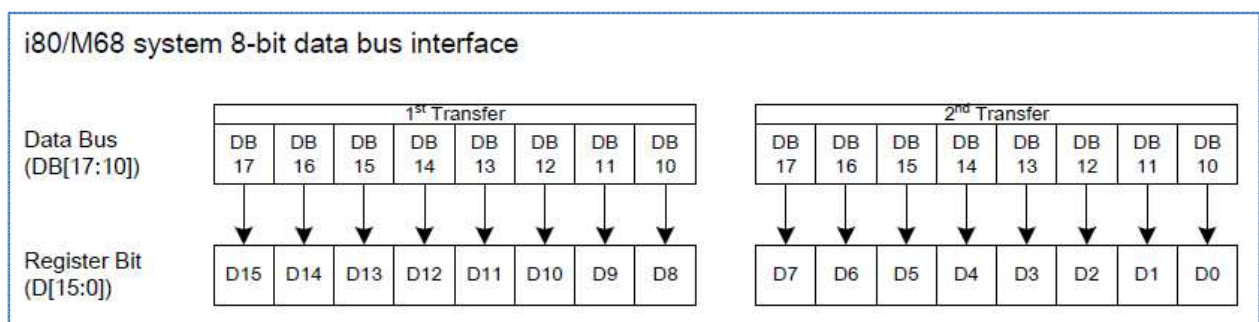
Figure 1: PINS out for TFT LCD control.

## Access Register of ILI9328 with 8-bit parallel interface

The TFT LCD use ILI9328 LCD Controller. If user needs to control the TFT LCD, you should understand the operation of how to control the ILI9328 first.

The operation and function of ILI9328 will start when receive the command from processor. For hardware design, the module will use 8-bit interface. The Index Register (IR) will store the value in register address which we will write the command or data. This operation will define by Register Selection (RS), Read/Write (RD/WR) and Data Bus (D0-D17)

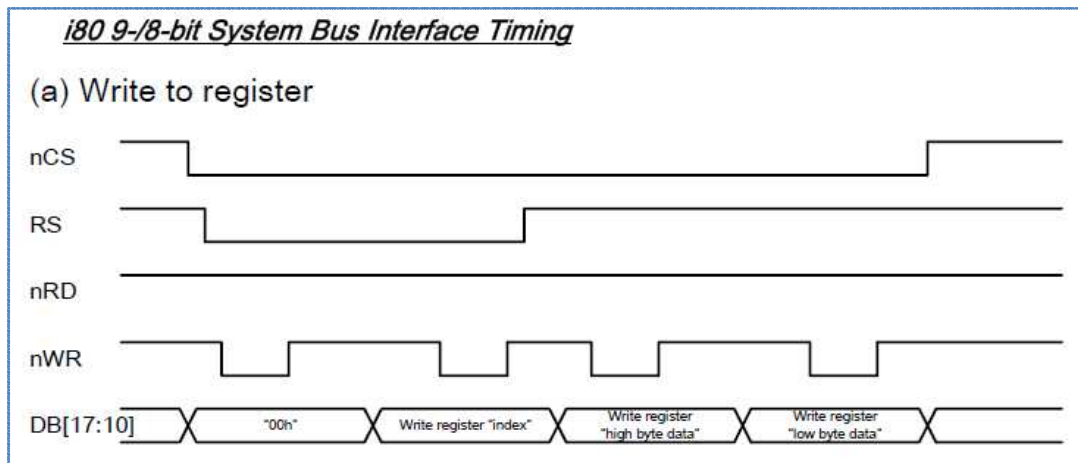
To define the value into 16-bit Register (D[15:0]) of ILI9328, you can follow the Data Transfer Format as figure 2.



**Figure 2: Data Transfer Format of ILI9328**

From figure 2, The TFT LCD Touch Screen we use 8-bit parallel interface operation which if we need to access register of ILI9328. User needs to send the data 2 times for access register in each operation, first sending is High Byte and second sending is Low Byte. The total is 16-bit sending operation.

## How to write the value to register of ILI9328 in 8-bit parallel interface operation



**Figure 3: Timing diagram for write to register**

Figure 3 show the timing diagram about the operation function to control sending command or data of ILI9328. There are 2 steps to access register as follow.

1. Step of register need to access (register address).
2. Step of data which we will write to the register.

From timing diagram, user needs to send the data 4 times (DB[17:10]). The first and second data is for access register to define register address that we need to connect. As the figure 3, the first data is 0x00 and the second data is register index. For third and second data, it is the operation to define the data for accessed register by sending high byte data and follow with low byte data.

### Step by Step for write data to register

From the timing diagram, PINs usage are shown below.

- Control Signal PINs: "LCD\_CS", "RS", "RD", and "WR"
- Data PIN: "D0" to "D7"

Following step use for writing data to ILI9328 register.

1. Set RD = "1"
2. Set LCD\_CD = "0" to enable LCD for writing data
3. Set RS = "0" to told ILI9328 receive this data as command
4. Set High Byte [D0-D7] = "0x00"
5. Set WR = "0" to start writing command to register

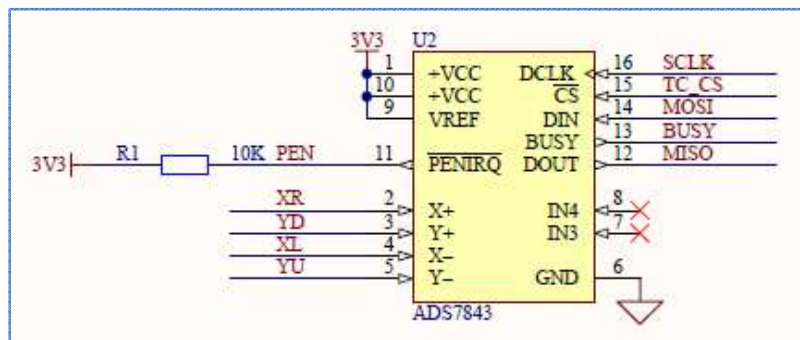
6. Set WR = "1" to stop writing after first byte sent
7. Set Low Byte [D0-D7] as register index which we need to communicate
8. Set WR = "0" to start writing register index to register
9. Set WR = "1" to stop writing after second byte sent
10. Set RS = "1" to told ILI9328 receive this data as data
11. Set High Byte [D0-D7] as high byte data which we need to store
12. Set WR = "0" to start writing high byte data to register
13. Set WR = "1" to stop writing after third byte sent
14. Set Low Byte [D0-D7] as low byte data which we need to store
15. Set WR = "0" to start writing low byte data to register
16. Set WR = "1" to stop writing after forth byte sent
17. Set LCD\_CD = "1" to disable LCD for finish data writing

Above steps demonstrated how to write data to ILI9328 register, then we can writing data to control TFT LCD



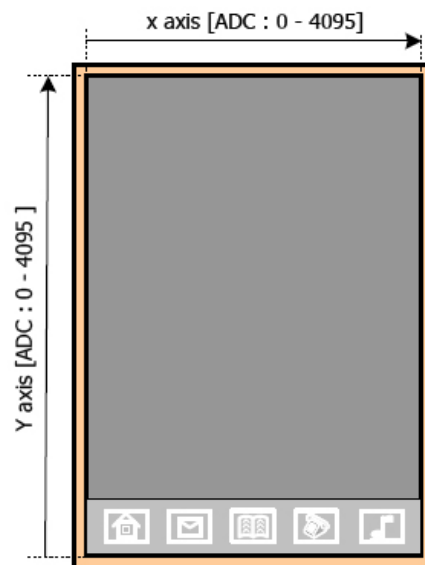
## Operating for Touch controller with SPI interface

Touch Screen designed to control using ADS7843 via SPI interface as schematic in figure 4.



**Figure 4: PINs out for Touch controller ADS7843**

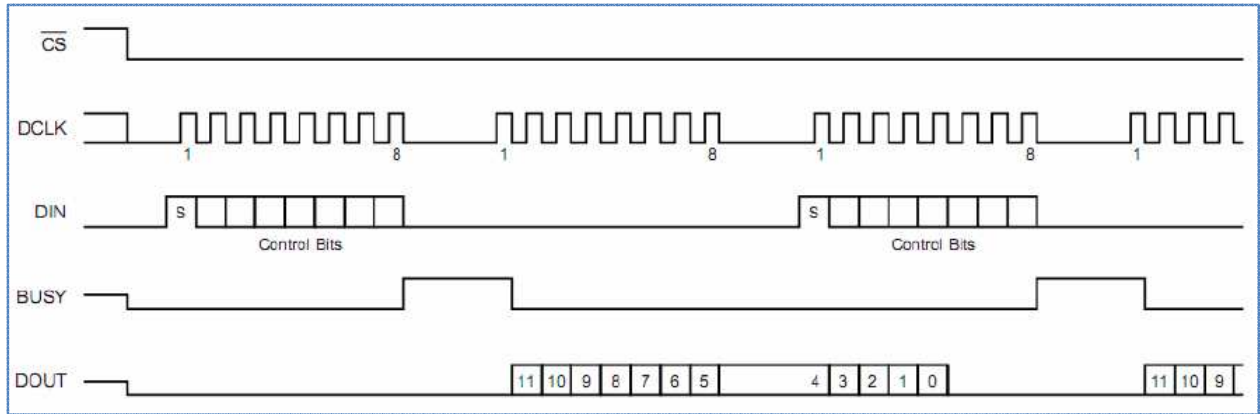
ADS7843 will convert analog signal from pin “X+”, Y+, X-, and Y- to digital signal with 12-bit sensitivity. When screen has been touched, pin “PENIRQ” will be “0”. So we can use it to check for touching, and then read digital output data. We need to calculate this data to find an actual position on LCD to control it.



**Figure 5: ADC value of horizontal (X-axis) and vertical (Y-axis)**

Reading ADC value from ADS7843, we need to send control byte to it every time before reading. More information about ADS7843 available in its product datasheet, useful control byte lists below.

- 0xD0: for reading ADC in X-axis
- 0x90: for reading ADC in Y-axis



**Figure 6: ADS7843 Timing diagram**

From timing diagram in figure 6, how to read ADC value from ADS7843 steps below.

1. Read "PENIRQ" status. If value is "0" mean that screen has been touched, we can start to read ADC value from ADS7843 using step 2 and later. If value is "1", there is no activity on touch screen.
2. Set TC\_CS = "0" to enable ADS7843
3. Send control byte "0xD0" via DIN pin to told ADS7843 get its X-axis ADC value
4. Send dummy data "0x00" via DIN pin to read ADC value from ADS7843, when each bit sent, ADS7843 will simultaneously shift first 7 bits of X-axis ADC value through DOUT pin are bit11-bit5: 0b0xxxxxxx
5. Send control byte "0x90" via DIN pin to told ADS7843 get its Y-axis ADC value, when each bit sent, ADS7843 will simultaneously shift last 5 bits of X-axis ADC value through DOUT pin are bit4-bit0: 0bxxxxx000
6. Send dummy data "0x00" via DIN pin to read ADC value from ADS7843, when each bit sent, ADS7843 will simultaneously shift first 7 bits of Y-axis ADC value through DOUT pin are bit11-bit5: 0b0xxxxxxx
7. Send dummy data "0x00" via DIN pin to read ADC value from ADS7843, when each bit sent, ADS7843 will simultaneously shift last 5 bits of Y-axis ADC value through DOUT pin are bit4-bit0: 0bxxxxx000
8. Set TC\_CS = "1" to disable ADS7843

Rearrangement ADC value for each axis by shift first 7 bits to the left side 5 times (5 bits) and shift last 5 bits to the right side 3 times (3 bits), then OR both value into 12-bit ADC value. Do these on both X-axis and Y-axis value.

## Hardware PINs layout

A 2.8-inch TFT Touch Shield has 24 PINs divided in 2 groups of usage. There are 18 signal PINs for control TFT LCD and 6 signal PINs for control touch screen controller and SD card for external storage. It was designed for Arduino UNO / Arduino Mega connectivity with 18 pins. Also designed for others microcontroller or others connectivity. All 24 pins located in figure 7.

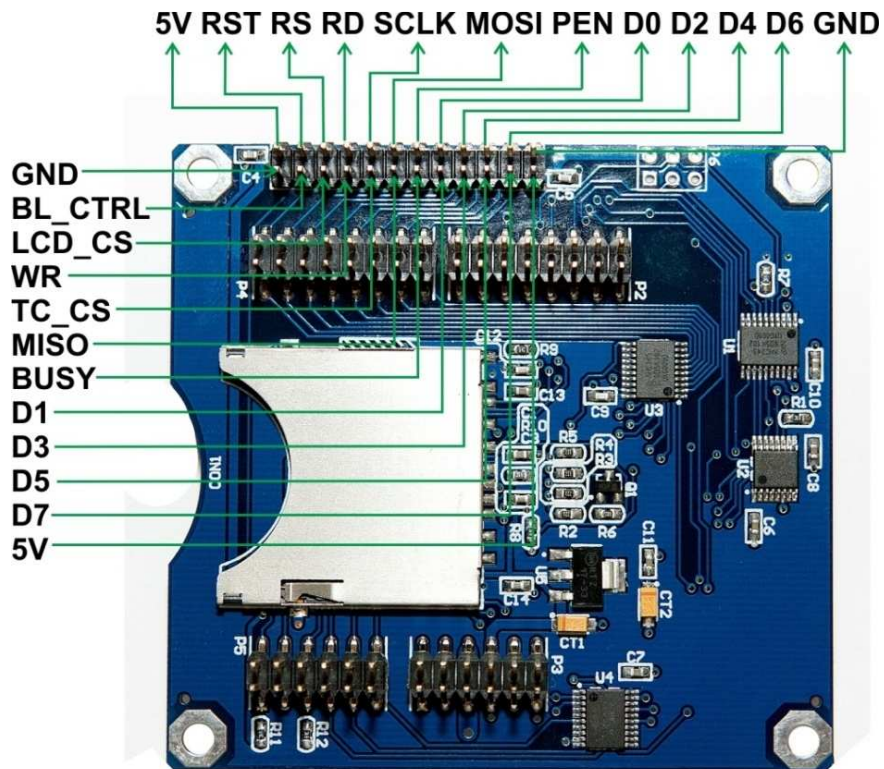


Figure 7: Hardware PINs layout

## Arduino connectivity

User needs 18 pins (out of 24 pins) of 2.8-inch TFT Touch Shield to connect to Arduino UNO / Arduino Mega. How to connect with Arduino shown in figure 8.

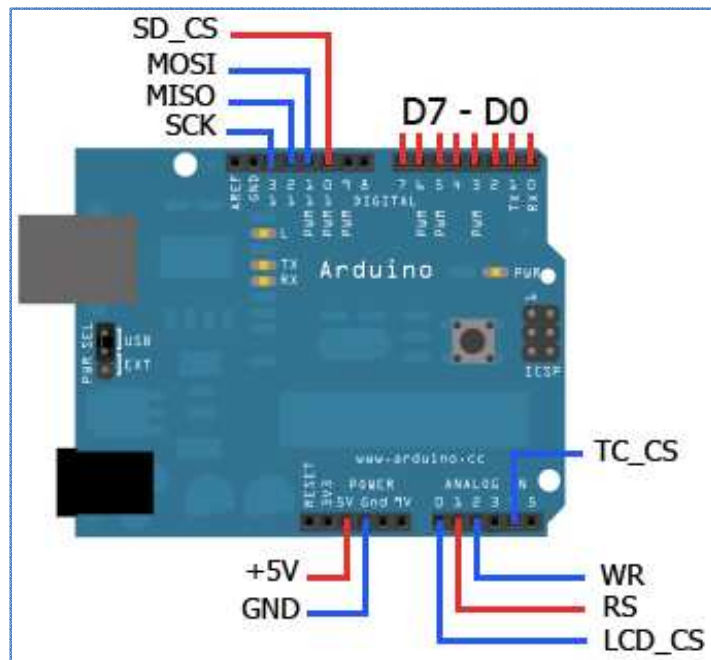


Figure 8: Arduino connectivity

## Arduino programming

ThaiEasyElec.com provided library for basic using LCD touch screen and SD card located in “TFT\_Library” directory. To use this library, you need to copy “TFT\_Library” directory to “libraries” directory under Arduino IDE directory

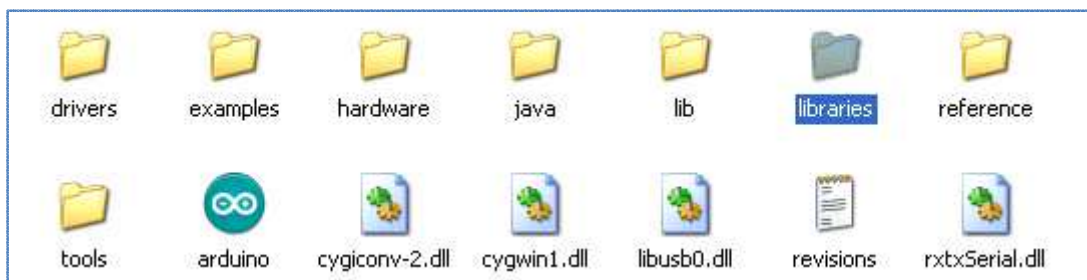


Figure 9: Location of “libraries” directory in Arduino IDE

Inside “TFT\_Library” directory stored necessary resources such as header files and source files, included Thai and English font that converted into C source code. Also you can include them to your source code if needed. There are many interesting files in library detailed below.

- TFT\_common\_rev\_a.h This file declaration each pin for Arduino.
- TFT\_driver\_rev\_a.cpp This file included many LCD low level function such as Command sending, Data sending, and LCD Initializing.
- TFT\_driver\_rev\_a.h This file defined many necessary values such as Basic color value and Commands.
- TFT\_graphics.cpp This file is important, it included all LCD function such as Initializing LCD for using, Drawing (Rectangle, Circle, and Line), Font using, and etc. For more about each function will describe in next chapter.
- TFT\_touch.cpp This file included with touch screen function. For more about each function will describe in next chapter.
- mmc.c This file use for communicating with SD card via SPI.
- pff.c This file included function for reading/writing file on FAT file system.

### LCD graphic usage function in TFT\_graphics.cpp

**void Graphics::initialize(uint8\_t orient)**

This function send a command to start using LCD, passing variable requires is value to define which XY-axis is vertically or horizontally aligned.

**void Graphics::SetArea(uint16\_t x1, uint16\_t y1, uint16\_t x2, uint16\_t y2)**

This function set a working area of LCD.

**void Graphics::SetCursor(uint16\_t x, uint16\_t y)**

This function set a starting point of drawing.

**void Graphics::ClearScreen(uint16\_t color)**

This function clear entire screen by fill them with specified color.

**void Graphics::FastSolidRect(uint16\_t x1, uint16\_t y1, uint16\_t x2, uint16\_t y2, uint16\_t color)**

This function quickly draws a solid rectangle filled with color by defining of start point, end point, and color.

**void Graphics::DrawPixel(uint16\_t x, uint16\_t y, uint16\_t color)**

This function draws a pixel with specified color.

**void Graphics::VerticalScroll(int16\_t y)**

This function scrolls entire screen in vertical.

**void Graphics::DrawLine(uint16\_t x1, uint16\_t y1, uint16\_t x2, uint16\_t y2, uint16\_t color)**

This function draws a line with color by defining of start point, end point, and color.

**void Graphics::DrawCircle(uint16\_t x, uint16\_t y, uint16\_t radius, uint16\_t color, uint8\_t fill)**

This function draws a circle with color plus option to fill it or not by defining of center point, radius, color, and fill.

**void Graphics::DrawRoundRect(uint16\_t x1, uint16\_t y1, uint16\_t x2, uint16\_t y2, uint16\_t radius, uint16\_t color, uint8\_t fill)**

This function draws a rectangle with color line and round corner plus option to fill it or not by defining of start point, end point, radius of corner, color, and fill.

**void Graphics::SetFontColor(uint16\_t color)**

This function set font color.

**void Graphics::SetBackColor(uint16\_t color)**

This function set font background color.

**void Graphics::SetOffset(uint16\_t x, uint16\_t y)**

This function set font offset.

**void Graphics::PrintStr(uint8\_t line, uint8\_t column, uint8\_t \*str)**

This function displays string by defining line, start column, and string.

**void Graphics::PrintCh(uint8\_t line, uint8\_t column, uint8\_t c)**

This function displays character by defining line, start column, and character.

**unsigned long Graphics::GetCharWidth(uint8\_t c)**

This function gets a width of specified character.

**unsigned long Graphics::GetStringWidth(uint8\_t \*str)**

This function gets a width of specified string.

**void Graphics::SetBold(uint8\_t on)**

This function sets a character to bold.

**uint8\_t Graphics::GetBold(void)**

This function gets a specified point has been bold or not.

**void Graphics::CfgLineHeight(uint8\_t line\_height)**

This function configures height of line.

```
void Graphics::CfgFont(uint8_t *_font, uint8_t width, uint8_t height_div_8, uint8_t gap)
```

This function configures English font by defining font, width, height (divide by 8), and character gap.

```
void Graphics::CfgExtFont(uint8_t *_font, uint8_t width, uint8_t height_div_8, uint8_t _tone_shift, uint8_t _font_ext_adj)
```

This function configures Thai font by defining font, width, height (divide by 8), tone mark and upper vowel height, and Thai and English character level adjustment.

```
void Graphics::ThaiUTF8_to_ASCII(uint8_t *ascii, uint8_t *utf8)
```

This function convert UTF-8 Thai character to buffer in ASCII, enable character drawing function display them properly.

### LCD touch screen usage function in TFT\_touch.cpp

```
void touchpanel::init(void)
```

This function initializes SPI communication with ADS7843 touch screen controller.

```
void touchpanel::get_matrix(void)
```

This function loads calibration data from EEPROM in AVR.

```
void touchpanel::save_matrix(void)
```

This function saves calibration data to EEPROM in AVR.

```
POINT* touchpanel::read_tp_point(void)
```

This function reads touched point, then store X and Y value in structure named POINT.

```
POINT* touchpanel::read_lcd_point(void)
```

This function comparing touched value with calibrated value to find actual point on LCD screen. Please notices, in the beginning of “TFT\_touch.cpp” declared a constructor coded below.

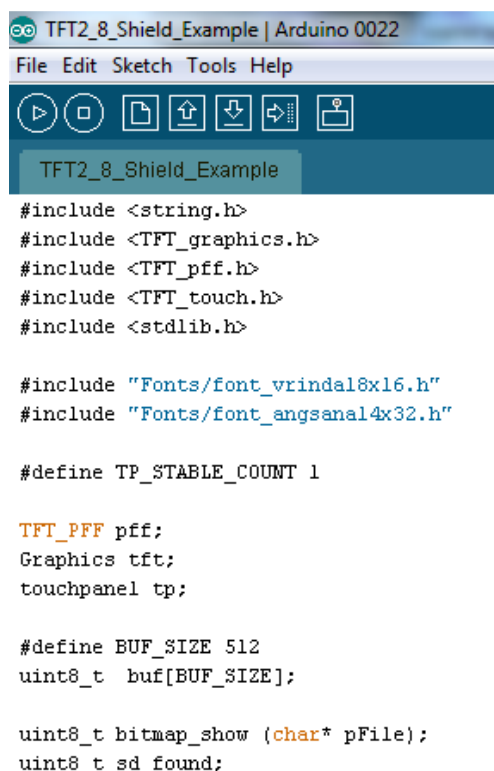
```
Touchpanel::touchpanel()  
{  
    get_matrix();  
}
```

When create an object of “touchpanel” class, it will automatically invoke get\_metrix() function.

## Appendix

### Describing an example code “TFT2\_8\_Shield\_Example.pde”

ThaiEasyelec.com has written example program to read picture file from SD card and display on LCD screen. You can download it from <http://www.thaieasyelec.net/archives/Manual/Example%20code.zip>, and extract it after download completed. Find and open “TFT2\_8\_Shield\_Example.pde” in Arduino IDE, upload it into your Arduino, and see how it works in this video (<http://www.youtube.com/watch?v=5zAb8GqaZoA>).

The image shows a screenshot of the Arduino IDE interface. The title bar reads 'TFT2\_8\_Shield\_Example | Arduino 0022'. The menu bar includes 'File', 'Edit', 'Sketch', 'Tools', and 'Help'. Below the menu bar is a toolbar with icons for running, stopping, saving, uploading, and other functions. The main text area displays the beginning of the source code for 'TFT2\_8\_Shield\_Example.pde'. The code includes several header files, defines constants, and declares variables for object creation.

```
TFT2_8_Shield_Example.pde
#include <string.h>
#include <TFT_graphics.h>
#include <TFT_pff.h>
#include <TFT_touch.h>
#include <stdlib.h>

#include "Fonts/font_vrindal8x16.h"
#include "Fonts/font_angsanal4x32.h"

#define TP_STABLE_COUNT 1

TFT_PFF pff;
Graphics tft;
touchpanel tp;

#define BUF_SIZE 512
uint8_t buf[BUF_SIZE];

uint8_t bitmap_show (char* pFile);
uint8_t sd_found;
```

Figure 10: Beginning of “TFT2\_8\_Shield\_Example.pde” source code

At the beginning of source code in figure 10, there are many included necessary header file to define variable for object creation from our library and declaration of 3 variables of created objects below.

```
TFT_PFF pff;
Graphics tft;
touchpanel tp;
```

They are objects of Filesystem for accessing file from SD card, Graphic on LCD, and touch screen.



```

TFT2_8_Shield_Example | Arduino 0022
File Edit Sketch Tools Help
TFT2_8_Shield_Example
int setCalibrationMatrix( POINT * displayPtr,
                        POINT * screenPtr,
                        MATRIX * matrixPtr)
{
    int retValue = 1 ;

    matrixPtr->Divider = ((screenPtr[0].x - screenPtr[2].x) * (screenPtr[1].y - screenPtr[2].y)) -
                        ((screenPtr[1].x - screenPtr[2].x) * (screenPtr[0].y - screenPtr[2].y)) ;

    if ( matrixPtr->Divider == 0 )
    {
        retValue = 0 ;
    }
    else
    {
        matrixPtr->An = ((displayPtr[0].x - displayPtr[2].x) * (screenPtr[1].y - screenPtr[2].y)) -
                        ((displayPtr[1].x - displayPtr[2].x) * (screenPtr[0].y - screenPtr[2].y)) ;

        matrixPtr->Bn = ((screenPtr[0].x - screenPtr[2].x) * (displayPtr[1].x - displayPtr[2].x)) -
                        ((displayPtr[0].x - displayPtr[2].x) * (screenPtr[1].x - screenPtr[2].x)) ;

        matrixPtr->Cn = (screenPtr[2].x * displayPtr[1].x - screenPtr[1].x * displayPtr[2].x) * screenPtr[0].y +
                        (screenPtr[0].x * displayPtr[2].x - screenPtr[2].x * displayPtr[0].x) * screenPtr[1].y +
                        (screenPtr[1].x * displayPtr[0].x - screenPtr[0].x * displayPtr[1].x) * screenPtr[2].y ;

        matrixPtr->Dn = ((displayPtr[0].y - displayPtr[2].y) * (screenPtr[1].y - screenPtr[2].y)) -
                        ((displayPtr[1].y - displayPtr[2].y) * (screenPtr[0].y - screenPtr[2].y)) ;

        matrixPtr->En = ((screenPtr[0].x - screenPtr[2].x) * (displayPtr[1].y - displayPtr[2].y)) -
                        ((displayPtr[0].y - displayPtr[2].y) * (screenPtr[1].x - screenPtr[2].x)) ;
    }
}

```

**Figure 11: Source code of “setCalibrationMatrix” function**

The “setCalibrationMatrix” function shown in figure 11 uses to calculate for position in calibration process, it uses horizontal screen formula calculation. A “tp\_calibrate” function shown in figure 12 uses in calibrating touch position of LCD screen.

```

void tp_calibrate()
{
    POINT lcd_screen_pts[3]={{28,18},{308,120}, {160,208}};
    POINT tp_pts[3];
    POINT* capture_pt;
    int16_t pre_tp_x=0, pre_tp_y=0;
}

```

**Figure 12: Source code of “tp\_calibrate” function**

Variable declaration for “tp\_calibrate” function have 3 POINTs are (28, 18), (308, 120), and (160, 208) use in calculation of calibration.

```

tft.initialize(LCD_HORIZONTAL);
tp.init();
tft.ClearScreen(BLACK);
// draw 3 points in turn //
for(uint8_t i = 0; i<3; i++)
{
    tft.DrawCircle(lcd_screen_pts[i].x, lcd_screen_pts[i].y, 5, BLUE,0);

    uint8_t j=0;

```

**Figure 13: Source code of calibration process**

At the beginning of calibration process in figure 13, there are 3 functions was called for horizontal adjustment, starting communication with touch screen, and clear entire screen to black, source code below.

```

tft.initialize(LCD_HORIZONTAL);
tp.init();
tft.ClearScreen(BLACK);

```

After that, prepare for calibration process source code below.

```

for(uint8_t i=0; i<3; i++)
{
    tft.DrawCircle(lcd_screen_pts[i].x, lcd_screen_pts[i].y, 5,
    BLUE, 0);
    uint8_t j=0;
    // loop to get stable tp point
    while(j<10)
    {
        delay(5);
        capture_pt = tp.read_tp_point();
        if(capture_pt->x!=0 && capture_pt->y!=0)
        {
            if(capture_pt->x == pre_tp_x && capture_pt->y ==
            pre_tp_y)
                j++;
            else
            {
                pre_tp_x = capture_pt->x;
                pre_tp_y = capture_pt->y;
            }
            if(j== 10) break;
        }
    }
    j=0;
    pre_tp_x = 0;
    pre_tp_y =0;
    tp_pts[i].x = capture_pt->x;
    tp_pts[i].y = capture_pt->y;
    tft.DrawCircle(lcd_screen_pts[i].x, lcd_screen_pts[i].y, 4,
    BLUE, 1);
    delay(500);
}

```

This process creates a blue transparent circle 5 pixels radius on calibration point to wait for touch activity on screen. While loop, `while(j<10)`, help to detect unexpected or bounce from touching using same method for de-bounce when press a switch. When touch values are stable, XY-axis values keep in “capture\_pt” structure then creates a blue solid circle over a transparent one indicated that already got touch values for this point.

There are 3 calibration points, although process above will loop 3 times, `for(uint8_t i=0; i<3; i++)`, in different defined position on (28, 18), (308, 120), and (160, 208). After that they have been calculated, then stored in EEPROM source code below.

```
if(setCalibrationMatrix(lcd_screen_pts, tp_pts, &tp.matrix))
{
    tp.save_matrix();
}
```

There is a necessary function in calibration process named “conv\_tp” source code in figure 14 below.

```
void conv_tp(POINT* tp)
{
    unsigned short buf;
    if (LCD::GetOrientation() == LCD_HORIZONTAL)
        return;
    buf = tp->x;
    tp->x = LCD::GetWidth() - tp->y - 1;
    tp->y = buf;
}
```

**Figure 14: Source code of “conv\_tp” function**

Because this calibration process uses horizontal point formula for horizontal screen then we need to convert to vertical point when we use vertical screen, this function will do this task.

In setup function, there are many declarations for screen shown in figure 15 below.

```
tft.initialize(LCD_VERTICAL);  
// tft.initialize(LCD_HORIZONTAL);  
tft.ClearScreen(BLACK);  
  
tft.SetFontColor(RED);  
tft.CfgFont(font_vrinda18x16,18,2,2);  
tft.CfgExtFont(font_angsana14x32,14,4,5,9);  
tft.CfgLineHeight(32);  
tft.SetBold(1);  
tft.SetOffset(0,0);
```

**Figure 15: Screen declaration setup function**

**tft.initialize(LCD\_VERTICAL);**

This function initialize LCD for using in vertical alignment, for horizontal alignment use **tft.initialize(LCD\_HORIZONTAL);**.

**tft.ClearScreen(BLACK);**

This function clears entire screen by filling with black.

**tft.SetFontColor(RED);**

This function sets font color to red.

**tft.CfgFont(font\_vrinda18x16,18,2,2);**

This function set English font to vrinda 18x16, width 18 pixels, height 2 (16 pixels/8), and gap 2 pixels.

**tft.CfgExtFont(font\_angsana14x32,14,4,5,9);**

This function set Thai font to angšana 14x32, width 14 pixels, height 4 (32 pixels/8, gap 5 pixels, and font level adjustment value to match English font level is 9.

**tft.CfgLineHeight(32);**

This function set line height to 32 pixels.

**tft.SetBold(1);**

This function set font to bold.

**tft.SetOffset(0,0);**

This function set offset on x=0 pixel and y=0 pixel.

```

sd_found = 0;
for (i=0;i<5;i++)
{
    if (disk_initialize() == 0)
    {
        pf_mount (&pf.fs);
        sd_found = 1;
        break;
    }
    delay(500);
}

if (!sd_found)
{
    tft.ThaiUTF8_to_ASCII(buf,(uint8_t *)"ไม่พบ SD Card");
    tft.PrintStr(2,1,buf);
    delay(5000);
    tp_calibrate();
}

```

**Figure 16: Source code for SD card checking**

SD card has been checking by procedure above in figure 16. If SD card inserted, it will break and exit to next function. If SD card not inserted, it will display “ไม่พบ SD card” (means “SD card not found”) for 5 seconds (5000 milliseconds). After that, it will start calibration process.

Please notice, if you need to display text string including with Thai character. You should use example below.

```

tft.ThaiUTF8_to_ASCII(buf,(uint8_t *)"ไม่พบ SD Card");
tft.PrintStr(2,1,buf);

```

It will convert Thai characters to ASCII like English characters and store them in array variable named “buf”. Then display it on 1<sup>st</sup> column on 2<sup>nd</sup> line.

The “get\_stable\_tp” function for getting specified position like loop use in calibration function in figure 17.

```
uint8_t get_stable_tp(POINT* tp)
{
    static POINT previous_tp;
    static uint8_t count=0;

    if(tp->x == 0 || tp->y == 0)
        return 0;

    if(tp->x == previous_tp.x && tp->y == previous_tp.y)
    {
        count++;
    }
    else
    {
        count = 0;
        previous_tp.x = tp->x;
        previous_tp.y = tp->y;
    }

    if(count >= TP_STABLE_COUNT)
    {
        count = 0;
        return 1;
    }

    return 0;
}
```

**Figure 17: Source code of “get\_stable\_tp” function**

Arduino essential standard “loop()” function source code in figure 18.

```
void loop()
{
    POINT *tp_dat;

    if (sd_found)
    {
        bitmap_show("pic1.bmp");
        delay(1000);
        bitmap_show("pic2.bmp");
        delay(1000);
        bitmap_show("pic3.bmp");
        delay(1000);
        bitmap_show("pic4.bmp");
        delay(1000);
    }

    tp_dat = tp.read_lcd_point();
    conv_tp(tp_dat);
    if (get_stable_tp(tp_dat) )
    { // Plot Pixel //
        //tft.DrawPixel (tp_dat->x, tp_dat->y, YELLOW);
        tft. DrawCircle(tp_dat->x,tp_dat->y, 3, YELLOW, 1);
    }
}
```

**Figure 18: Source code of Arduino essential standard “loop()” function**

There are 2 major parts. First one source code below, if SD card found, it will call “bitmap\_show” function to read bitmap image file from SD card and display on screen.

```
if (sd_found)
{
    bitmap_show("pic1.bmp");
    delay(1000);
    bitmap_show("pic2.bmp");
    delay(1000);
    bitmap_show("pic3.bmp");
    delay(1000);
    bitmap_show("pic4.bmp");
    delay(1000);
}
```

But if SD card not found it will skip above part to second one part, when we touch any position on screen. It will create solid yellow circle with radius of 3. If we drag on screen, it will create a solid yellow line on dragged path.

```
tp_dat = tp.read_lcd_point();
conv_tp(tp_dat);
if (get_stable_tp(tp_dat) )
{
    tft. DrawCircle(tp_dat->x,tp_dat->y, 3, YELLOW, 1);
}
```

The “bitmap\_show” function shown in figure 19, it starts reading header and info. of bitmap file from SD card to check file is usable.

```
res = pf_open (pFile);
if (res) return res;

res = pf_read (&BMPHeader, sizeof(BMPHeader), &cnt);
if (res) return res;

if (BMPHeader.bfType != 0x4d42) return 0xff;

res = pf_read (&BMPInfo, sizeof(BMPInfo), &cnt);
if (res) return res;

if (BMPInfo.biSize != 40) return 0xff;
if (BMPInfo.biPlanes != 1) return 0xff;
if (BMPInfo.biCompression != 0) return 0xff;
```

**Figure 19: Source code of “bitmap\_show” function for reading header and info.**

After that, it will specify an area for drawing that image, source code in figure 20.

```
sx = 0;
sy = 0;
ex = LCD::GetWidth() - 1;
ey = LCD::GetHeight() - 1;

if ((bitmap_width != LCD::GetWidth()) || (bitmap_height != LCD::GetHeight()))
{
    tft.ClearScreen(BLACK);
    if (LCD::GetWidth() > bitmap_width)
    {
        sx = (LCD::GetWidth() - bitmap_width) >> 1;
        ex = sx + bitmap_width - 1;
    }
    if (LCD::GetHeight() > bitmap_height)
    {
        sy = (LCD::GetHeight() - bitmap_height) >> 1;
        ey = sy + bitmap_height - 1;
    }
}
```

**Figure 20: Source code of “bitmap\_show” function for specify drawing area**



Finally, it reads image data and displays on screen, source code in figure 21 below.

```
pf_read ((void*) buf, z , &cnt);
if (Y)
    for (j=0;j<cnt;j+=3)
    {
        b = buf[j];
        g = buf[j+1];
        r = buf[j+2];
        color = (r << 8) & 0xF800;
        color |= (g << 3) & 0x07E0;
        color |= b >> 3;
        LCD::Write_DATA(color);
        Y -= 3;
        if (Y == 0)
            break;
    }
```

**Figure 21: Source code of “bitmap\_show” function for read and display image**