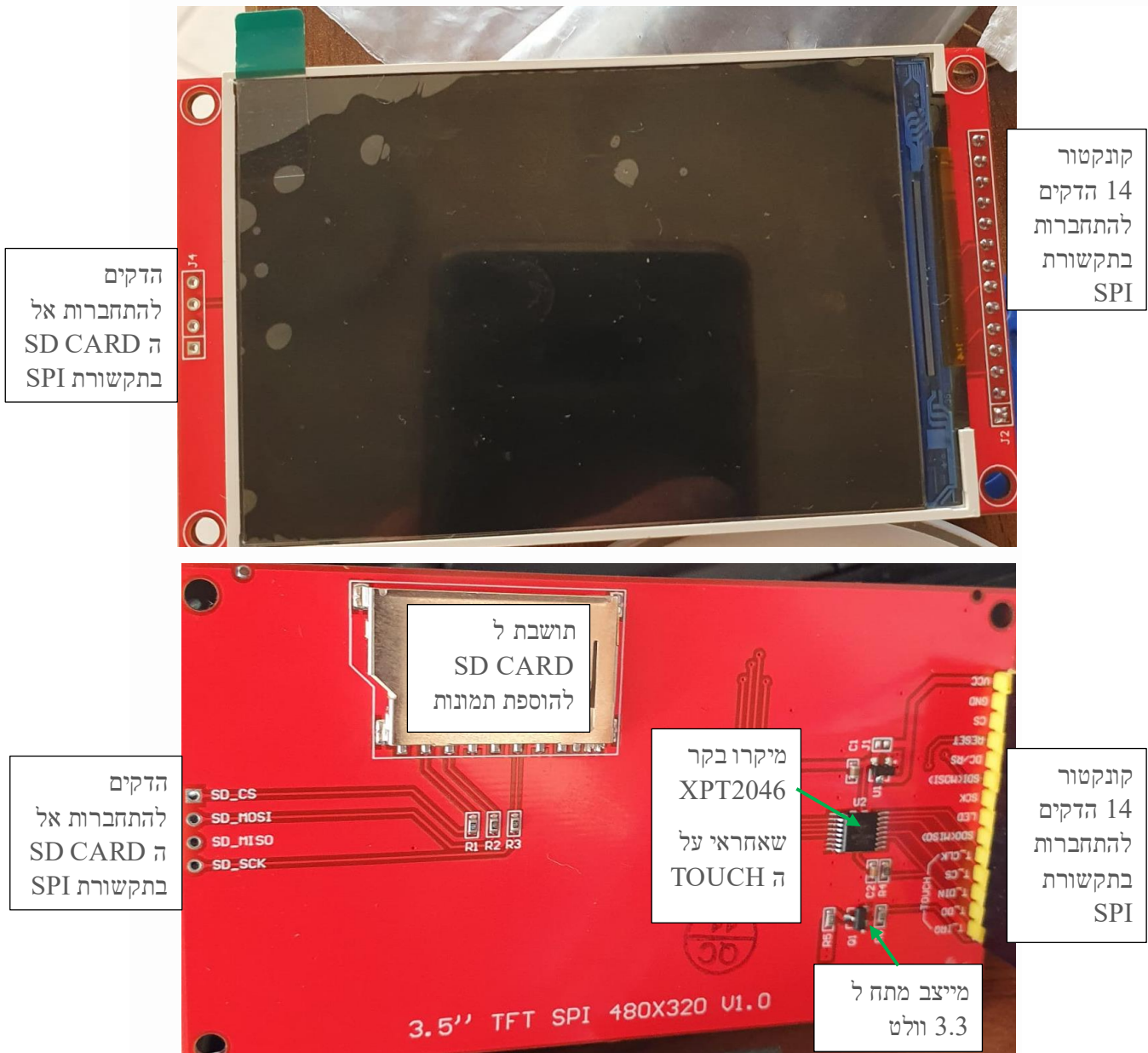


ה touch במסך תצוגת TFT

1. מבוא

תצוגת TFT בגודל 3.5 אינץ' נראית כך:

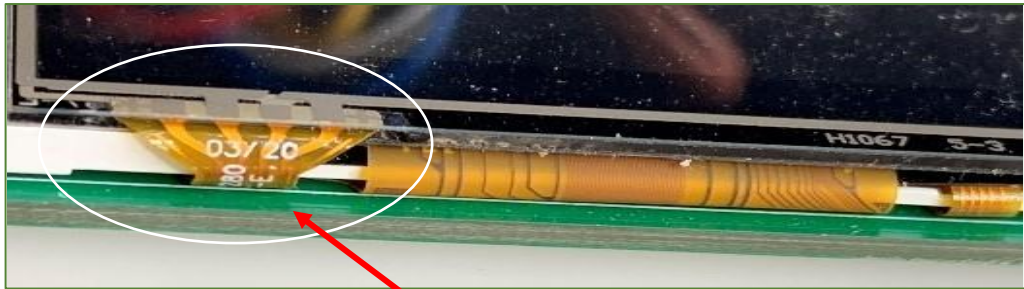


איור 1 : מראה התצוגה – באיור העליון מסך התצוגה ובאיור שמתחתיו צד המעגל המודפס עם הרכיבים שעליו.

בחלק העליון של האיור רואים את מסך התצוגה. הוא מגיע עם כיסוי הגנה מניילון דק למניעת שריטות וניתן לסלק אותו בקלות בעזרת הפס הדק הירוק הנראה בחלק השמאלי העליון של המסך.

תחילה עלינו לוודא שהמודול TFT שלנו תומך במגע. חלק מספקי TFT באינטרנט מפרסמים את המודולים שלהם כמו מסכי מגע, אבל למעשה הם לא. ישנם 3 סימנים פיזיים לכך שהמסך תומך ב touch - מגע :

א. האם יש ל TFT לוח מגע ? לוח המגע הוא כיסוי דק ושקוף המחובר למשטח העליון של הצג. הרמזים לנוכחותו הם ארבעה פסים עבים על כבל פלסטיק גמיש המחברים את לוח המגע למודול והם נראים באיור הבא :



תצוגה עם מסך מגע לצי המשיטה המסומן בעיגול



איור 2 : מסך TFT עם touch בחלק העליון של האיור וללא touch בחלק התחתון

בחלק העליון רואים את המשטח של הפלסטיק הגמיש שמראה את חיבורי ה touch . בחלק התחתון מופיע מסך ללא touch כי חסר לו משטח הפלסטיק הקשיח.

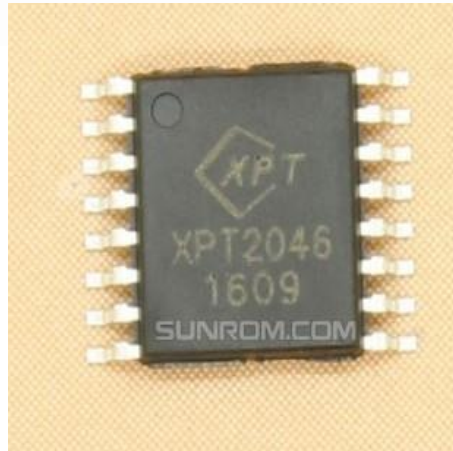
ב. האם יש ל TFT הדקים/פינים של מגע ? יש לספור את מספר הפינים בכל צד של המודול. למודולים עם הדקי/פיני מגע יש בדרך כלל 14 פינים בצד אחד, ו- 4 פינים בהמשך (עבור SD CARD). 14 הפינים מחולקים בצורה הבאה : 9 פינים של התצוגה ועוד 5 פינים של ה touch . הפינים של המגע מסומנים " T_CS ", " T_DIN ", " T_CLK ", " T_DO ", " T_IRQ "



איור 3 : 14 הדקי התצוגה. 9 הדקים של התצוגה ועוד 5 הדקים מצד שמאל השייכים ל touch .

ג. האם יש ל TFT בקר מגע ? בקר המגע הוא שבב/ג'וק קטן ומרובע המותקן על פני המשטח בגב המודול, הממוקם לעתים קרובות בסמוך לפינים של המגע. החלק הזה הוא הבקר XPT2046. ג'וקים תואמים יכולים להיקרא "RB2046". ג'וק

נוסף עם תפקיד זהה הוא ADS7843. הבקרים האלו נקראים TOUCH SCREEN CONTROLLER - בקרי מסך מגע. יש לוחות הכוללים רפידות עבור השבב, אך לא מותקן בהם שבב! השבב נראה באיור הבא:



איור 4 : ג'וק XPT2046

2. קצת על הג'וק XPT2046 שאחראי על ה touch

XPT2046 הוא בקר מסך מגע התנגדתי מצד אחד בתקשורת SPI אל הדקי תצוגת ה TFT בעזרת 5 הדקים (מתוך 14) שהראינו באיור קודם, ומצד שני אל 4 חוטים המתחברים להדקי ה X וה Y של משטח המגע. על תקשורת SPI ניתן למצוא בקישור :

<https://www.arikporat.com/wp-content/uploads/2022/12/serial-communication-comparison.pdf>

הוא משלב ממיר ADC של 12 סיביות עם דגימה בקצב של 125 kHz מסוג SAR (Successive Approximation Register – רגיסטר שערך מוצלח) .

הוא יכול לפעול גם ממתח ספק של 2.2 וולט (על מנת לחבר מיקרו בקרים של מתח נמוך) ותומך במתח ממשק קלט/פלט דיגיטלי מ- V1.5 ל- VCC.

זיהוי מיקום הלחיצה במסך מתבצעת על ידי ביצוע שתי המרות A/D.

בנוסף למיקום ה XPT2046 מודד גם את הלחץ שנגענו במסך המגע.

יש לו מתח ייחוס - VREF ברכיב עצמו של 2.5 וולט וניתן להשתמש בו לקריאת מתח חיצוני אנלוגי שנכניס מבחוץ, וגם למדידת טמפרטורה וניטור (מדידת מתח) סוללה עם היכולת למדוד מתח מ- 0 עד 5 וולט. יש לו גם חיישן טמפרטורה פנימי.

הג'וק זמין באריזה דקה QFN של 16 פינים (0.75 מ"מ גובה) ויש לו טווח טמפרטורות פעולה של -40° עד $+85^{\circ}$ מעלות צלסיוס.

Absolute Maximum Ratings – מקסימליים – ערכים נקובים מקובלים

| | |
|-----------------------------------|-----------------------|
| +VCC and IOVDD to GND | -0.3V to +6V |
| Analog Inputs to GND | -0.3V to +VCC + 0.3V |
| Digital Inputs to GND | -0.3V to IOVDD + 0.3V |
| Power Dissipation | . 250mW |
| Maximum Junction Temperature | +150°C |
| Operating Temperature Range | . -40°C to +85°C |
| Storage Temperature Range | -65°C to +150°C |
| Lead Temperature (soldering, 10s) | +300°C |

טבלה 1 : ערכים נקובים מקסימליים

מהטבלה רואים שמתח V_{cc} הוא עד 6 וולט ומתחי הכניסה הדיגיטאליים הם מ -0.3 וולט עד $V_{cc}+0.3$ וולט .

3. הדקי ה XPT2046

הטבלה הבאה מתארת את תפקיד ההדקים של הג'וק.

| Number | Pin Name | Descriptions |
|--------|-----------|--|
| 1 | VCC | Operating power supplies |
| 2 | GND | System ground level blocks |
| 3 | CS | Chip select input pin ("Low" enable) |
| 4 | RESET | This signal will reset the device and must be applied to properly initialize the chip. Signal is active low. |
| 5 | RS | This pin is used to select "Data or Command" . When RS = '1', data is selected. When RS = '0', command is selected |
| 6 | SDA(MOSI) | Low, Serial in/out signal. High, Serial input signal. |
| 7 | SCL | This pin is used serial interface clock |
| 8 | BL_EN | LCD backlight enable (is active high) |
| 9 | SDO(MISO) | Serial output signal |
| 10 | T_DCLK | External Clock Input. This clock runs the SAR conversion process and synchronizes serial data I/O. |
| 11 | T_CS | Chip Select Input. Controls conversion timing and enables the serial input/output register. |
| 12 | T_DIN | Serial Data Input. If CS is LOW, data is latched on rising edge of DCLK. |
| 13 | T_DOUT | Serial Data Output. Data is shifted on the falling edge of DCLK. This output is high impedance when CS is HIGH. |
| 14 | PENIRQ | Pen Interrupt. |

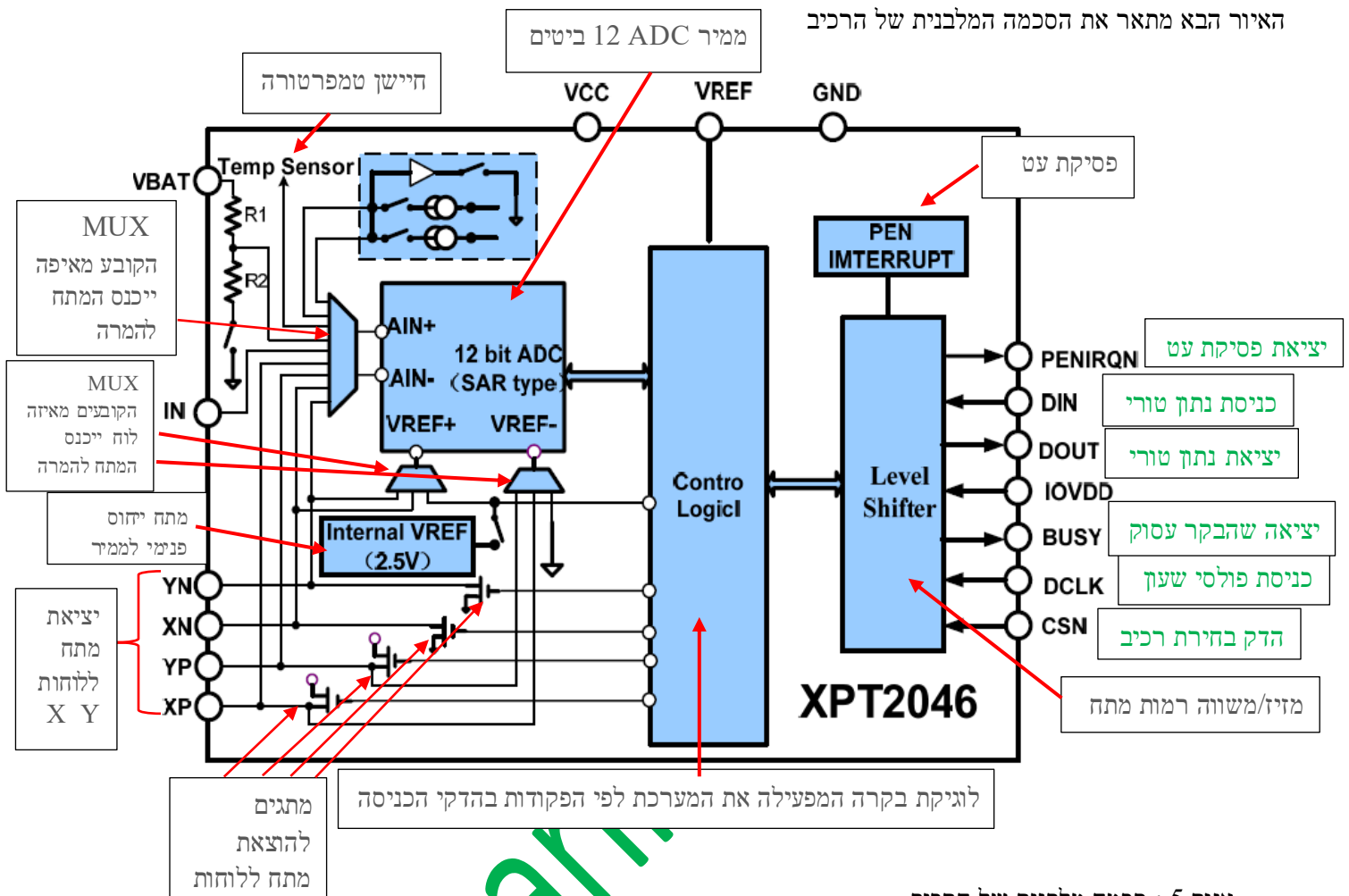
טבלה 2 : הדקי הרכיב מדפי היצרן

הסבר ההדקים של הטבלה נראה בטבלה הבאה :

| מספר ההדק | שם | תפקיד |
|-----------|-----------|--|
| 1 | Vcc | מתח ספק להפעלה |
| 2 | GND | אדמה |
| 3 | CS | כניסת Chip Select - בחירת רכיב של תקשורת ה SPI הטורית עם התצוגה . ההדק פעיל בנמוך. |
| 4 | RESET | אות איפוס הרכיב. בהתחלת העבודה יש לשים בהדק 0 ולהעלות ל 1 . (פעיל בנמוך). |
| 5 | RS | בעזרת הדק זה קובעים האם מכניסים פקודה (שמים 0) או נתן (שמים 1). |
| 6 | SDA(MOSI) | קו הנתן של תקשורת ה SPI מהאדוק לעבד. |
| 7 | SCL | קו השען של תקשורת ה SPI הטורית. |
| 8 | BL_EN | אפשרות התאורה האחורית של התצוגה Back Light Enable . פעיל בגבוה. |
| 9 | SDO(MISO) | קו הנתן הטורי מהעבד לאדוק בתקשורת ה SPI . |
| 10 | T_DCLK | כניסת שען חיצונית. פולסי השען מסנכרנים את תקשורת ה SPI הטורית של ה TOUCH וגם מפעילים את תהליך ההמרה של ה ADC . |
| 11 | T_CS | הדק Chip Select של ה touch . זהו הדק בחירת הרכיב של תקשורת ה SPI של ה touch . |
| 12 | T_DIN | כניסת הנתן הטורי בתקשורת ה SPI . אם T_CS=0 הנתן ננעל בעלייה של פולס השען. |
| 13 | T_DOUT | יציאת נתן טורי. הנתן מוזז בירידת השען ב T_DCLK . היציאה נמצאת בעכבה גבוהה אם T_CS בגבוה. |
| 14 | PENIRQ | הדק יציאה לפסיקת עט (לתצוגה מצורף פלסטיק דמו עט). נגיעה במסך בעזרת העט (או אצבע) גורמת ליציאת פסיקה למיקרו בקר חיצוני. הפסיקה פעילה בנמוך. |

טבלה 3 : הסבר הדקי הרכיב.

4. סכמה מלבנית

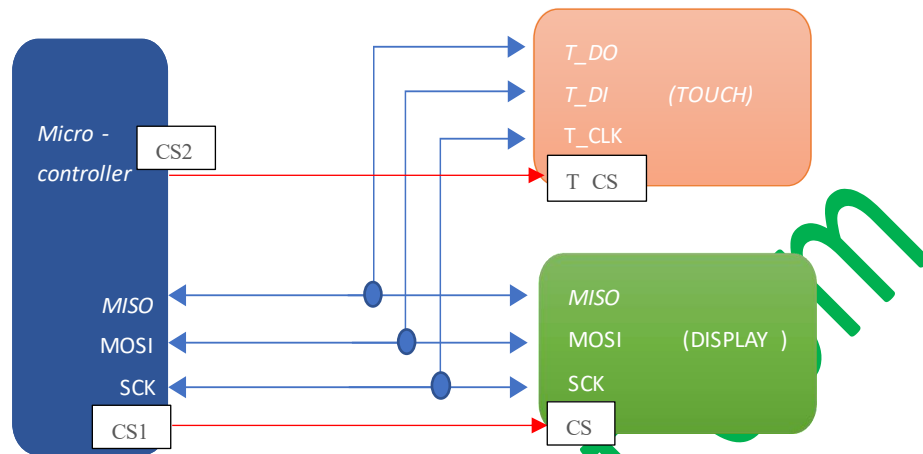


איור 5 : סכמה מלבנית של הרכיב

מצד שמאל באמצע אלו הדקי ה XPT2046 . 5 ההדקים המתחברים לקונקטור ה touch בתצוגת ה TFT מסומנים בצבע ירוק. ההדק BUSY מראה האם הסתיימה ההמרה (אנחנו לא משתמשים בה בתצוגת ה TFT) . במרכז רואים את XPT2046 הממיר מאנלוגי לדיגיטאלי - ADC - העובד בשיטת הקירוב הרציף (SAR Successive Approximation Register) או בעברית גם "רגיסטר שערך/ניחוש מתצלה" . הממיר של 12 ביטים. ברכיב מתח ייחוס פנימי של 2.5 וולט . בעזרת ה MUX שנמצא משמאל לממיר קובעים איזה מתח ייכנס להמרה. (האם מציר X או ציר Y או מחיישן הטמפרטורה הפנימי וכו'). מלבן ה Control Logic - בקרה לוגית הוא זה ששולט על כל מה שקורה ברכיב. הוא מבצע את כל הפעולות לפי מה ששלחנו לו בהדקים של תקשורת ה SPI . מלבן ה Level Shifter - מזיז רמה - שבצד ימין הוא משנה/מזיז את רמות המתח שנכנסות מצד ימין שלו לרמות מתח שמלבן הבקרה הלוגית עובד איתם.

5. חיווט התצוגה וה touch בתקשורת SPI

האיור הבא מתאר את חיבור תצוגת ה TFT וה touch שלה אל מיקרו בקר. שתיהן מתחברות בתקשורת SPI. יש להן 3 קווים משותפים שך SCK MISO MOSI והמבדיל ביניהן יהיה קו CS – בחירת רכיב. הרכיב שניתן לו 0 ב CS הוא זה שניצור איתו את התקשורת.

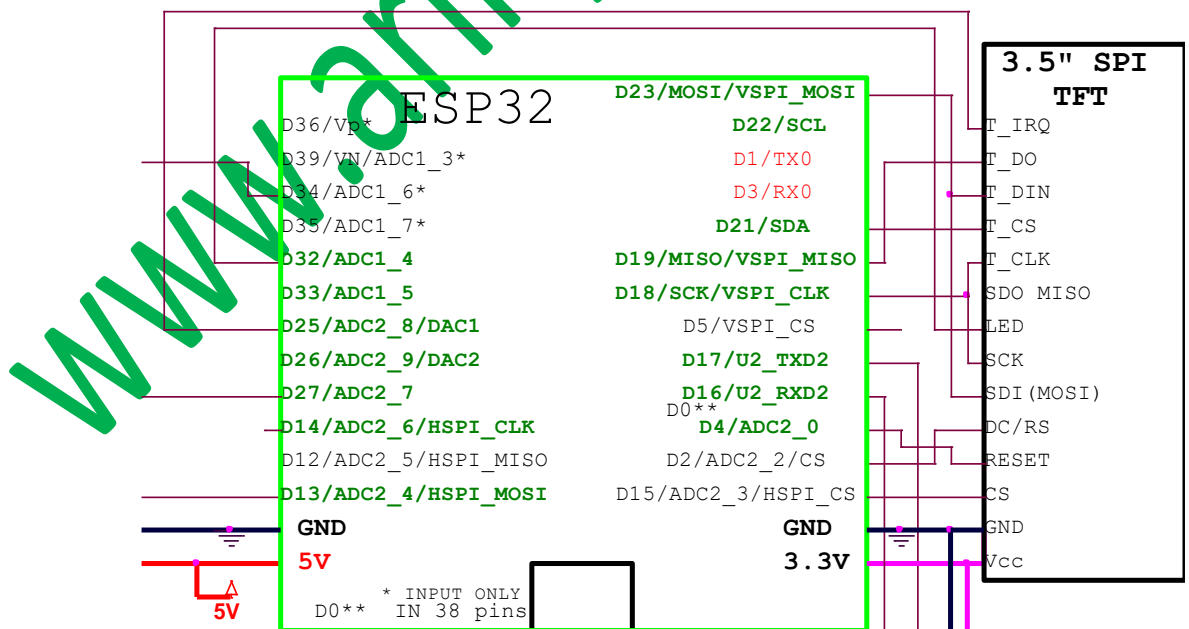


איור 6 : החיווט של התצוגה וה touch

ב touch של התצוגה יש שמות שונים במקצת להדקים :

“T_DO” הוא MISO, “T_DI” הוא MOSI, “T_CLK” הוא SCK

נחבר את התצוגה אל מיקרו בקר ESP32 לפי האיור הבא :



איור 7 : חיבור מיקרו בקר ESP32 אל תצוגת TFT עם TOUCH

לחיסכון בהדקים ניתן **לא לחבר** את הדק ה T_IRQ (להשאיר באוויר). אין צורך לחבר את הדק ה LED של התצוגה ניתן לחבר את ההדק ל 3.3V ואת הדק RESET ניתן לחבר לרגל EN של המיקרו בקר. כך חוסכים 3 הדקים !

6. תוכנה בסביבת העבודה של ארדואינו IDE .

לתמיכה ב-TFT נשתמש בחבילת התוכנה - "TFT_eSPI" של Bodmer . כדי להתקין אותה יש לעבור אל :
Arduino library manager (Sketch->Include Libraries->Manage Libraries)

לחפש "TFT_eSPI" ולבצע install .

ניתן גם למצוא את הקוד העדכני ביותר ב-GitHub בכתובת

https://github.com/Bodmer/TFT_eSPI

ולבצע התקנה לספריית הארדואינו.

ניכנס לספריית ה-TFT_esp_i ונפתח את הקובץ User_Setup.h . נעתיק את ההגדרות הבאות ונשמור את הקובץ.
אני השתמשתי ב-TFT עם דרייבר ILI9488 . יש לשים לב שכל שאר הדרייברים יהיו עם // לפני השורה שלהם (שורת הערה).
בהתחלה אני מראה דרך " ארוכה" ולאחריה דרך נוספת פשוטה ויעילה יותר.

```
//          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 1. Call up the right driver file and any options for it
//
// #####

// Define STM32 to invoke optimised processor support (only for STM32)
//#define STM32

// Defining the STM32 board allows the library to optimise the performance
// for UNO compatible "MCUfriend" style shields
//#define NUCLEO_64_TFT
//#define NUCLEO_144_TFT

// STM32 8 bit parallel only:
// If STM32 Port A or B pins 0-7 are used for 8 bit parallel data bus bits 0-7
// then this will improve rendering performance by a factor of ~8x
//#define STM_PORTA_DATA_BUS
```



```
//#define STM_PORTB_DATA_BUS

// Tell the library to use 8 bit parallel mode (otherwise SPI is assumed)
//#define TFT_PARALLEL_8_BIT

// Display type - only define if RPi display
//#define RPI_DISPLAY_TYPE // 20MHz maximum SPI

// Only define one driver, the other ones must be commented out
//#define ILI9341_DRIVER // Generic driver for common displays
//#define ILI9341_2_DRIVER // Alternative ILI9341 driver, see https://github.com/Bodmer/TFT_eSPI/issues/1172
//#define ST7735_DRIVER // Define additional parameters below for this display
//#define ILI9163_DRIVER // Define additional parameters below for this display
//#define S6D02A1_DRIVER
//#define RPI_ILI9486_DRIVER // 20MHz maximum SPI
//#define HX8357D_DRIVER
//#define ILI9481_DRIVER
//#define ILI9486_DRIVER
#define ILI9488_DRIVER // WARNING: Do not connect ILI9488 display SDO to MISO if other devices share the SPI bus
(TFT_SDO does NOT tristate when CS is high)
//#define ST7789_DRIVER // Full configuration option, define additional parameters below for this display
//#define ST7789_2_DRIVER // Minimal configuration option, define additional parameters below for this display
//#define R61581_DRIVER
//#define RM68140_DRIVER
//#define ST7796_DRIVER
//#define SSD1351_DRIVER
//#define SSD1963_480_DRIVER
//#define SSD1963_800_DRIVER
//#define SSD1963_800ALT_DRIVER
//#define ILI9225_DRIVER
//#define GC9A01_DRIVER

// Some displays support SPI reads via the MISO pin, other displays have a single
// bi-directional SDA pin and the library will try to read this via the MOSI line.
// To use the SDA line for reading data from the TFT uncomment the following line:

// #define TFT_SDA_READ // This option is for ESP32 ONLY, tested with ST7789 and GC9A01 display only

// For ST7735, ST7789 and ILI9341 ONLY, define the colour order IF the blue and red are swapped on your display
// Try ONE option at a time to find the correct colour order for your display
```

```
// #define TFT_RGB_ORDER TFT_RGB // Colour order Red-Green-Blue
// #define TFT_RGB_ORDER TFT_BGR // Colour order Blue-Green-Red

// For M5Stack ESP32 module with integrated ILI9341 display ONLY, remove // in line below

// #define M5STACK

// For ST7789, ST7735, ILI9163 and GC9A01 ONLY, define the pixel width and height in portrait orientation
// #define TFT_WIDTH 80
// #define TFT_WIDTH 128
// #define TFT_WIDTH 240 // ST7789 240 x 240 and 240 x 320
// #define TFT_HEIGHT 160
// #define TFT_HEIGHT 128
// #define TFT_HEIGHT 240 // ST7789 240 x 240
// #define TFT_HEIGHT 320 // ST7789 240 x 320
// #define TFT_HEIGHT 240 // GC9A01 240 x 240

// 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 stray 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_GREENTAB160x80 // For 160 x 80 display (BGR, inverted, 26 offset)
// #define ST7735_REDTAB
// #define ST7735_BLACKTAB
// #define ST7735_REDTAB160x80 // For 160 x 80 display with 24 pixel offset

// If colours are inverted (white shows as black) then uncomment one of the next
// 2 lines try both options, one of the options should correct the inversion.

// #define TFT_INVERSION_ON
// #define TFT_INVERSION_OFF
```

```
// #####  
//  
// Section 2. Define the pins that are used to interface with the display here  
//  
// #####  
  
// If a backlight control signal is available then define the TFT_BL pin in Section 2  
// below. The backlight will be turned ON when tft.begin() is called, but the library  
// needs to know if the LEDs are ON with the pin HIGH or LOW. If the LEDs are to be  
// driven with a PWM signal or turned OFF/ON then this must be handled by the user  
// sketch. e.g. with digitalWrite(TFT_BL, LOW);  
  
#define TFT_BL 32 // LED back-light control pin .  
#define TFT_BACKLIGHT_ON HIGH // Level to turn ON back-light (HIGH or LOW)  
  
// 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 labelled 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 as 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
```

```
//  
//  
// 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 TFT_BL PIN_D1 // LED back-light (only for ST7789 with backlight control pin)  
  
// #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. It is best not to connect MISO as some displays  
// do not tristate that line when chip select is high!  
// On NodeMCU 1.0 SD0=MISO, SD1=MOSI, CLK=SCLK to connect to TFT in overlap mode  
// On NodeMCU V3 S0=MISO, S1=MOSI, S2=SCLK  
// In ESP8266 overlap mode the following must be defined  
  
// #define TFT_SPI_OVERLAP  
  
// 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
```

```
// ##### 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
// for 9488 driver
#define TFT_MISO 19
#define TFT_MOSI 23
#define TFT_SCLK 18
#define TFT_CS 15 // 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 ESP32 Dev board (only tested with GC9A01 display)
// The hardware SPI can be mapped to any pins

//#define TFT_MOSI 15 // In some display driver board, it might be written as "SDA" and so on.
//#define TFT_SCLK 14
//#define TFT_CS 5 // 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 22 // 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

// 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 (required for M5Stack)

// ##### 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 for the STM32 and ESP32
// Example below is for ESP32 Parallel interface with UNO displays

// Tell the library to use 8 bit parallel mode (otherwise SPI is assumed)
//#define TFT_PARALLEL_8_BIT

// 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 - must 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 - must use a pin in the range 0-31
//#define TFT_RD 2 // Read strobe control pin

//#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

// ##### EDIT THE PINs BELOW TO SUIT YOUR STM32 SPI TFT SETUP #####

// The TFT can be connected to SPI port 1 or 2
//#define TFT_SPI_PORT 1 // SPI port 1 maximum clock rate is 55MHz
//#define TFT_MOSI PA7
//#define TFT_MISO PA6
//#define TFT_SCLK PA5

//#define TFT_SPI_PORT 2 // SPI port 2 maximum clock rate is 27MHz
//#define TFT_MOSI PB15
//#define TFT_MISO PB14
//#define TFT_SCLK PB13

// Can use Arduino pin references, arbitrary allocation, TFT_eSPI controls chip select
```



```
//#define TFT_CS D5 // Chip select control pin to TFT CS
//#define TFT_DC D6 // Data Command control pin to TFT DC (may be labelled RS = Register Select)
//#define TFT_RST D7 // Reset pin to TFT RST (or RESET)
// OR alternatively, we can use STM32 port reference names PXXn
//#define TFT_CS PE11 // Nucleo-F767ZI equivalent of D5
//#define TFT_DC PE9 // Nucleo-F767ZI equivalent of D6
//#define TFT_RST PF13 // Nucleo-F767ZI equivalent of D7

//#define TFT_RST -1 // Set TFT_RST to -1 if the display RESET is connected to processor reset
// Use an Arduino pin for initial testing as connecting to processor reset
// may not work (pulse too short at power up?)

// #####
//
// 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. 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.  
  
// #define SPI_FREQUENCY 1000000  
// #define SPI_FREQUENCY 5000000  
// #define SPI_FREQUENCY 10000000  
// #define SPI_FREQUENCY 20000000  
#define SPI_FREQUENCY 27000000  
// #define SPI_FREQUENCY 40000000  
// #define SPI_FREQUENCY 55000000 // STM32 SPI1 only (SPI2 maximum is 27MHz)  
// #define SPI_FREQUENCY 80000000  
  
// Optional reduced SPI frequency for reading TFT  
#define SPI_READ_FREQUENCY 20000000  
  
// The XPT2046 requires a lower SPI clock rate of 2.5MHz so we define that here:  
#define SPI_TOUCH_FREQUENCY 2500000  
  
// The ESP32 has 2 free SPI ports i.e. VSPI and HSPI, the VSPI is the default.  
// If the VSPI port is in use and pins are not accessible (e.g. TTGO T-Beam)  
// then uncomment the following line:  
// #define USE_HSPI_PORT  
  
// 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
```

הצעה נוספת פשוטה ויעילה:

כדאי להעתיק את הספרייה TFT_eSPI אל הספרייה שבה נמצאת התוכנית שאנחנו כותבים ולא לשנות את הקובץ המקורי .

לאחר התקנת הספרייה נשנה את הקובץ User_Setup.h שבספריית TFT_eSPI בצורה הבאה :

יש לכתוב קובץ User_Setup.h שיהיה בספרייה של התוכנית שלנו כך שיכלול את ההגדרות הבאות במקום כל ההגדרות האחרות:

```
#define ESP32
#define ILI9488_DRIVER
#define TFT_MISO 19
#define TFT_MOSI 23
#define TFT_SCLK 18
#define TFT_CS 15 // 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
#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
```

רשום את התוכנית הבאה :

```
#include "TFT_eSPI.h" // הספרייה TFT_esp של התוכנית TFT_esp
```

```
#define TITLE "www.arikporat.com" // ההודעה שתוצג
```

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

```
void setup()
```

```
{
```

```
tft.init();
```

```
tft.setRotation(1); // פספורט landscape שנראה תמונת נוף
```

```
tft.fillScreen(TFT_GREEN); // הרקע של התמונה יהיה ירוק
```

```
tft.setTextColor(TFT_RED); // הטקסט בצבע אדום
```

```
tft.drawString(TITLE,120,150,4); // הצג את ההודעה www.arikporat.com
```

```
}
```

```
void loop()  
{  
  
}
```

התמונה המתקבלת בסיום הרצת התוכנית נראית באיור הבא :



איור 8 : התמונה המתקבלת בהרצת התוכנית

אם רואים על ה TFT את ההודעה "www.arikporat.com" בצבע אדום על רקע ירוק , אפשר להמשיך ולעבור אל ה touch . אם התצוגה הפוכה, אפשר לסובב פיזית את המסך או לשנות את השורה `setRotation(1)` מ-1 ל-3. אם נרשום 2 נקבל תצוגת פספורט ולא נף.

7. התגובה לאירוע של נגיעה במסך

ה XPT2046 הוא בקר מגע התנגדתי. לוחות המגע מורכבות משתי יריעות שקופות עם ציפוי התנגדותי של כמה מאות אוהם) המופרדות על ידי מרווח אוויר קטן. בכל יריעה יש שתי אלקטרודות משני צידי היריעה. ליריעה אחת מחברים את האלקטרודות של X (X_p למתח חיובי ו X_m למתח שלילי) וליריעה השנייה אל לוחות Y (Y_p למתח חיובי ו Y_m למתח שלילי) . כשנוגעים בחלק התצוגה שתי היריעות מתכווצות והזרם עובר בין היריעות. ההתנגדות בין שתי אלקטרודות ה X וה Y משתנה בהתאם למיקום נקודת המגע. ממיר אנלוגי לדיגיטלי שבתוך ה XPT2046 מודד את מתחי האלקטרודות וממיר אותם לקואורדינטות עבור לוח X ולוח Y. הבקר קובע גם קואורדינטת Z, אשר מציינת את כמות הלחץ המופעל. ישנן ספריות עבור Arduino העובדות עם XPT2046. דוגמה טובה היא XPT2046_Touchscreen של פול סטופרנג על Github. עם זאת, מכיוון שהתמיכה במסך מגע מוכללת בתוך TFT_eSPI אין צורך להוסיף ספריות נוספות .

7.א בדיקה האם הייתה נגיעה במסך - הפונקציה touched()

הפונקציה בודקת האם יש נגיעה במסך לפי מדידה של לחץ הנגיעה במסך (מדובר במדידה שנקראת Z). היא מחזירה true (1) אם הייתה נגיעה במסך. אם לא אז היא מחזירה false (0).

```
bool touched() // הפונקציה מחזירה true אם יש נגיעה במסך מעל רמת לחץ סף שקבענו.
{
const int threshold = 500; // רמת סף של מדידת לחץ כדי שנגיעות קלות לא ישפיעו.
tft.getTouchRawZ() > threshold; // השוואה עם רמת הסף. אם הנגיעה מעל רמת הסף זה נגיעה מכוונת
}
```

7.ב בדיקה מהן קואורדינטות של נקודת המגע - הפונקציה checkForTouch()

בכל פעם שמהפונקציה touched() חוזר true נבצע שאילתה בבקר עבור הקואורדינטות (x,y) שבהן התרחש המגע. הפונקציה הבאה בודקת את נקודות המגע:

```
void checkForTouch()
{
short unsigned int x, y;
if (touched()) // האם המשתמש נגע במסך ?
{
tft.getTouch(&x,&y); // קבל את הקואורדינטות של איקס ו וואי
// הדפסה למסך של נקודת המגע.
// יש לרשום ב setup() את הפקודה
// Serial.begin(9600);
Serial.print("x = ");
Serial.print(x);
Serial.print(" y = ");
Serial.println(y);
markLocation(x,y); // נסמן נקודה/עיגול במסך במקום שנגענו
delay(300); // השהייה להתגבר על ניתור המגע
}
}
```

הפונקציה tft.getTouch() משמשת להחזרת קואורדינטות x,y של המגע. השהייה של 0.3 שנייה (300 אלפיות השנייה) היא כדי שלא נפעל שוב ושוב באותו אירוע מגע. הדבר נקרא debounce – ניתור מגעים.

להלן תוכנית/סקיצה מלאה לבדיקת מסך המגע. התוכנית מציירת עיגול צהוב קטן בכל מקום שבו נוגעים במסך.

```
#include <TFT_eSPI.h> // https://github.com/Bodmer/TFT\_eSPI
TFT_eSPI tft = TFT_eSPI(); // הגדרת אובייקט
//----- פונקציה הבודקת האם יש לחיצה במסך -----

bool touched() // הפונקציה מחזירה true אם יש נגיעה במסך מעל רמת לחץ סף שקבענו.
{
const int threshold = 500; // רמת סף של מדידת לחץ כדי שנגיעות קלות לא ישפיעו.
tft.getTouchRawZ() > threshold; // השוואה עם רמת הסף. אם הנגיעה מעל רמת הסף זה נגיעה מכוונת
}
```

```
//----- סימון נקודת המגע במסך -----  
void markLocation(int x, int y)  
{  
  tft.fillCircle(x,y,6,TFT_YELLOW); // ( נקודת המגע בצבע צהוב )  
}  
  
// ----- תוכנית הבדקת מהו המקש הנלחץ -----  
void touch()  
{  
  x=y=0;  
  if (touched())  
  { // did user touch the display?  
    tft.getTouch(&x,&y); // מתודה לקבלת נקודות המגע  
    // הדפסת נקודת המגע במוניטור הטורי  
    Serial.print("x = ");  
    Serial.print(x);  
    Serial.print(" y = ");  
    Serial.println(y);  
    markLocation(x,y); // סימון הנקודה במסך  
    delay(300); // השהיית ניתור מגעות  
  }  
}  
  
void setup()  
{  
  Serial.begin(9600); // אתחול הפורט הטורי לתקשורת עם מסך המחשב  
  tft.init(); // TFT אתחול תצוגת ה  
  tft.setRotation(3); // אתחול המסך לתמונת נוף ולא פספורט ( אפשר לרשום גם 1 )  
  tft.fillScreen(TFT_BLACK); // ניקוי המסך על ידי צביעת כל הפיקסלים בשחור  
}  
void loop()  
{  
  checkForTouch(); // קריאה לפונקציה הבדקת את נקודת המגע  
}
```


7.g. להצנים ואזורים

קבלת הקואורדינטות של נקודת המגע היא נהדרת אבל מה שאנחנו רוצים לעשות באופן מעשי הוא לקבוע אם נגעו באובייקט מסוים על המסך, כגון לחצן או פקד אחר. לשם כך אנו מגדירים "אזור" על המסך. זהו אזור שאם נלחץ עליו נגלה זאת ונבצע פעולה כלשהי. האיור הבא מתאר אזור במסך ה-TFT המתואר כמלבן עם קודקודים מעוגלים המשמש כאזור שעליו רוצים ללחוץ.



איור 9 : אזור בחירה

האזור הנבחר נמצא ב: $x_0=20, y_0=25$ (קודקוד שמאלי עליון), $width = 200$ - רוחב האזור, $height = 100$ גובה האזור. נתייחס לאזור המלבני הכתום/צהוב שרוחבו 200 פיקסלים וגובהו 100 פיקסלים, כשהפינה העליונה/השמאלית שלו נמצאת בקואורדינטות (20,25). כיצד אנו קובעים אם נקודת מגע נתונה נמצאת באזור? נקודה נמצאת באזור אם מתקיימים שני התנאים הבאים:

א. קואורדינטת X שלו היא בין X_0 ל- $(X_0 + \text{רוחב})$

ב. קואורדינטת Y שלו היא בין y_0 ל- $(y_0 + \text{גובה})$

כדי לכתוב תוכנית ל-2 התנאים האלו ניצור את האזור באמצעות סוג נתונים בשפת C שנקרא מבנה - structure.

```

Typedef struct {
int x; // נקודה x0 בקודקוד המלבן העליון בצד שמאל 20 בדוגמה שלנו
int y; // נקודה y0 בקודקוד המלבן בצד שמאל 25 בדוגמה שלנו
int w; // נקודה x בקודקוד הימני העליון x1=x0+width = 20+200=220
int h; // נקודה y בקודקוד הימני התחתון y1=y0+height = 25+100=125
} region; // שם המבנה הוא region
// נגדיר משתנה בשם rOrange ונאתחל אותו עם ערכי הקודקוד השמאלי הרוחב והגובה
region rOrange = { 20, 25, 200, 100};

```

כעת, כדי לקבוע אם נקודה נתונה נמצאת בתוך אזור זה, צור פונקציה in Region המקבלת אזור ונקודות x,y של נקודת המגע. הפונקציה מחזירה True אם הנקודה נמצאת בתוך האזור או false אם הנקודה לא בתוך האזור:

```

boolean inRegion (region b, int x, int y)
{
if ( x >= b.x && x <=(b.x+b.w) && y >= b.y && y <= (b.y+h)) // האם נקודת המגע בתוך האזור ?
    return true;
else
    return false;
}

```

כעת, נשנה את הפונקציה markLocation() המקורית שבעמודים קודמים, כך שצבע עיגול הסמן יהיה ירוק אם הוא נמצא בתוך האזור, ואדום אם הוא נמצא מחוץ לאזור :

```

void markLocation(int x, int y)
{
// נקרא לפונקציה הבדוקת האם הנקודה בתוך האזור ?
// האופרטור ? הוא סימן של if-else מקוצר . אם התנאי מתקיים מתבצע המשפט אחרי סימן השאלה. אם לא אז
// מתבצע המשפט שאחרי הנקודותיים
// TFT_GREEN עובר למשתנה color אם התנאי מתקיים. אם לא מועבר למשתנה TFT_RED
int color = inRegion(rOrange,x,y) ? TFT_GREEN:TFT_RED;
// אם כן הצבע ירוק. אם לא צבע באדום
tft.fillCircle(x,y,6,color); // פונקציה מצוירת עיגול ברדיוס 6 פיקסלים לפי הצבע ירוק או אדום
}

```

תוכנית המדגימה את כל הנאמר נמצאת בקישור :

[Touch-Control/touch_demo2/touch_demo2.ino at main · bhall66/Touch-Control · GitHub](https://github.com/bhall66/Touch-Control/blob/master/Touch-Control/touch_demo2.ino)

7.7 אזורים מרובים

מסך יכיל בדרך כלל יותר מלחצן או פקד אחד (מספר אזורים לבחירה בעזרת נגיעה) , ונצטרך לקבוע באיזה פקד נגענו. אם יש רק כמה אזורי מסך שיש לקחת בחשבון, הדרך הפשוטה ביותר היא לבדוק אותם בנפרד במתחם אם.. משפט אחר, כך:

```

void checkForTouch()
{
uint16_t x, y;
if (touched()) // פונקציה שבדוקת האם הייתה נגיעה במסך ?
{
tft.getTouch(&x,&y); // פונקציה המחזירה את קואורדינטות נקודת המגע
if (inRegion(region1,x,y)
    משפט/משפטים;
if (inRegion(region2,x,y)
    משפט/משפטים;
if (inRegion(region3,x,y)

```

```

משפט/משפטים;
delay(300);
}
}

```

כל אזור נבדק, ואם הנקודה נמצאת בתוך האזור, נבצע משפט או מספר משפטים (לא לשכוח סוגריים מסולסלים) או נקראת פונקציה המתאימה לאזור זה. אבל אם יש אזורים רבים לבדיקה, השימוש במערך עשוי להיות הגיוני יותר. להלן מבנה C לאתחול מערך אזורים, ופונקציה להצגת כל האזורים על המסך:

```

// מקרו לקבוע את מספר האיברים במערך
#define ELEMENTS(x) (sizeof(x) / sizeof(x[0])) // Macro to determine #elements in array
// הגדרה של מערך מבנים
region rScreen[] =
{
    // דוגמה למסך עם מספר אזורים
    {1,1,320,35}, // כותרת המערך
    {20,50,200,100}, // אזור הזמן
    {240,60,80,35}, // אזור אזור זמן
    {240,110,80,35}, // AM/PM אזור
    {1,180,140,40}, // אזור מצב שעות 1
    {180,180,140,40} // אזור מצב שעות 2
};

void fillRegion (int ID, int color) // פונקציה להצגת כל אזור במסך
{
    tft.fillRect(rScreen[ID].x,rScreen[ID].y, rScreen[ID].w, rScreen[ID].h, color);
}

void displayScreenRegions() // פונקציה להצגת כל האזורים במסך
{
    for (int i=0; i<ELEMENTS(rScreen); i++) // לכל אזור במערך
        fillRegion(i, TFT_BLUE); // קוראים לפונקציה בשורות שלמעלה לקבל צבע כחול
}

כדי לקבוע אם אירוע מגע נופל באחד מאזורים אלה, נבנה לולאה for שבודקת כל אזור במערך, ומחזירה את האינדקס של האזור המתאים למגע. אם הלולאה עברה על כל האפשרויות אז שום דבר לא תואם ואז מחזור -1 שאומר שאין התאמה.

int regionID(int x, int y)
{
    עבור כל אזור במערך
    for (int i=0; i<ELEMENTS(rScreen); i++)
        האם נקודת המגע באזור ?
        if inRegion(rScreen[i],x,y)
            כן . החזר את האינדקס במערך
            return i;
}

```

```
return -1; // החזר -1 כי נקודת המגע לא נמצאת באף אזור.  
}
```

כדי לטפל באירוע נגיעה, התקשר ל- regionID באמצעות קואורדינטות המגע ולאחר מכן צור משפט מקרה מתג לטיפול בכל אחד מהאזורים:

```
void handleTouchEvent (int x, int y)
```

```
{  
int ID = regionID(x,y); // קריאה לפונקציה בשורות הקודמות הבודקת האם הייתה נגיעה באחד האזורים  
switch(ID)  
{  
case 0: // הנגיעה באזור 0  
משפט/משפטים ;  
case 1: // הנגיעה באזור 1  
משפט/משפטים ;  
case 2: // הנגיעה באזור 2  
משפט/משפטים ;  
}
```

הקוד המלא להדגמת המגע השלישית והאחרונה נמצא ב-GitHub בקישור :

[Touch-Control/touch_demo3/touch_demo3.ino at main · bhall66/Touch-Control · GitHub](#)

www