A Summer Internship on

Implementation drivers of TFT Display ILI9341 using STM32G071 microcontroller

Submitted to the Department of Electronics Engineering in Partial Fulfilment for the Requirements for the Degree of

Bachelor of Technology (Electronics

and Communication)

by

Om Lukhi

(U21EC057)

(BTech.V(EC), vth Semester)

Guide

Dr. Pinalkumar J Engineer

Assistant Professor, DoECE

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SARDAR VALLABHBHAI NATIONAL INSTITUTE OF TECHNOLOGY

JULY-2023

Sardar Vallabhbhai National Institute of Technology

Surat-395 007, Gujarat, India

Department of Electronics Engineering

Certificate

This is to certify that the Summer Training Report entitled "Implementation drivers of TFT Display ILI9341" is presented and submitted by canadidate Om Lukhi, bearing roll no U21EC057, of B.Tech. III, 5th semester in the partial fulfillment of the requirement for the award of B.Tech. Degree in Electronics and Communication Engineering for academic year 2024-2025.

He has successfully and satisfactorily completed her/his Summer Training Presentation Exam in all respects. We certify that the work is comprehensive, complete and fit for evalution.

Dr. Pinalkumar J Engineer

Assistant professor and Summer Internship Guide

Name of Examiners Signature with date

1. _____ _____

2. _____ _____

Dr.Rasika N. Dhavse
Head and Associate Professor Seal of Department DoECE, SVNIT (August 2022)

Abstract

This internship abstract highlights the experience of interfacing a TFT Display ILI9341 using both the Arduino ATmega328 and the STM32G071 microcontroller platforms. The project involved the integration of hardware and software components to successfully communicate with the display and showcase graphical content. Through hands-on experimentation, programming, and troubleshooting, the intern gained valuable insights into the intricacies of display communication protocols, microcontroller configurations, and graphical rendering techniques. This internship provided a comprehensive learning opportunity in the realm of embedded systems, fostering proficiency in both hardware and software aspects of microcontroller-based projects. The intern's accomplishments include successful display initialization, data transmission, and graphical rendering, demonstrating a deepened understanding of display technologies and microcontroller capabilities. This experience not only enhanced technical skills but also honed problem-solving abilities and the capacity to adapt to different hardware environments, ultimately contributing to a holistic understanding of embedded systems development.

Acknowledgement

I would like to express my special thanks of gratitude to Dr. Pinalkumar J Engineer, Assistant Professor, DoECE,SVNIT for giving me this opportunity and providing guidance for the internship. I would also like to extend my gratitude to Mr. Aksh Patel and Mitul patel for his guidance throughout the internship. I would also like to thanks Departement of Electronics and Communication for giving the lad facilites and all the required hardware required during the internship.

Om Lukhi
Sardar Vallabhbhai National Institute of Technology
Surat

August 10,2023

Contents

List of Figures

Chapter 1

Introduction

1.1 Problem statement

Implementation of drivers of TFT Display ILI9341, using mnicrocontrollers 1.Arduino atmega328p and 2. STM32G071 .

The internship project aims to address the challenge of effectively interfacing a TFT Display ILI9341 with two distinct microcontroller platforms: the Arduino ATmega328 and the STM32G071. The key problem is to establish seamless communication between the microcontrollers and the display, enabling the rendering of graphical content on the screen. This involves deciphering the intricacies of the display's communication protocol, configuring the microcontrollers to transmit data in accordance with the protocol, and implementing code to render visually appealing graphics.

1.2 Components used in interfacing

- STM32G071 microcontroller
- TFT Display ILI9341
- Arduino
- Connecting wires, breadboard

Chapter 2

TFT Display ILI9341

Introduction

Basic six components of TFT Display:

1. TFT Pixel
2. TFT Pixel Schematic
3. TFT Panels (Also known as TFT Glass)
4. TFT Gate Drivers
5. TFT Source Drivers
6. Gamma
7. Multiplexing Gate and Source Drivers

2.1 TFT Pixel

The fundamental element in a TFT display is the liquid crystal. These elements have the property that the crystals will align from horizontal (which blocks the light) to vertical (which lets most of the light through) based on the electric field applied to them. Basically, we shine light through the liquid crystal, which blocks some or all of the light, the remainder of the white light then goes through a color filter to make red, green, or blue. It works like this:

1.We use an array of LEDs to shine white light from the back of the screen towards the front (your eyes).

2.Into a diffuser (to spread out the light and make it even)

3.We control the orientation of the crystals using a voltage to apply an electric field to the crystals

4.The white light from the back (often called the backlight) will shine through the liquid crystal elements. The amount of light coming out will depend on the orientation of the crystals.

5.The white light coming out of the crystal will then go into a red, green or blue color filter making it red, green or blue (RGB)

6.The light from three RGB filters will combine in your eye into a color based on the amount of red, green and blue (purple in the case below).

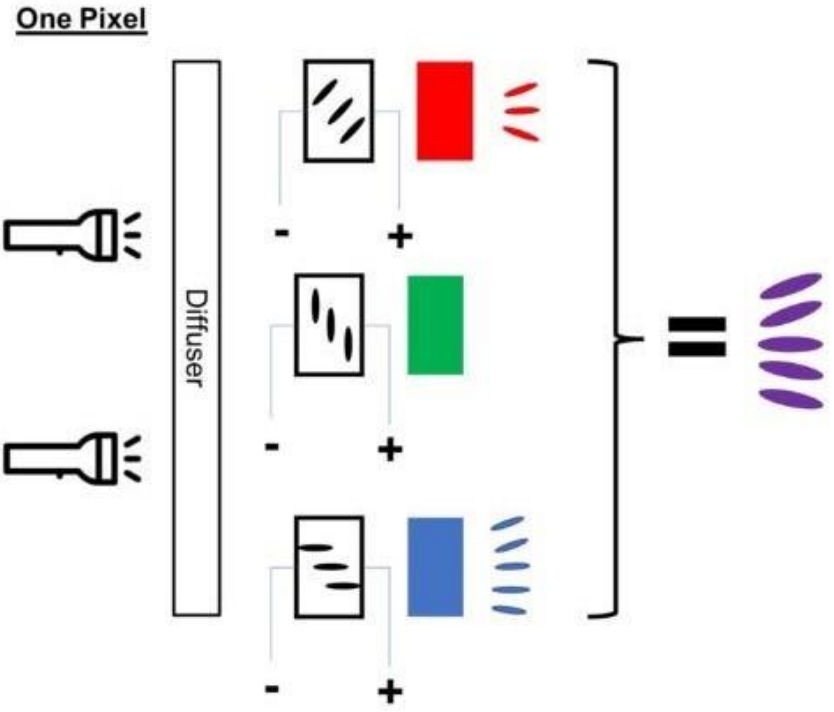


Figure 2.1: One Pixel

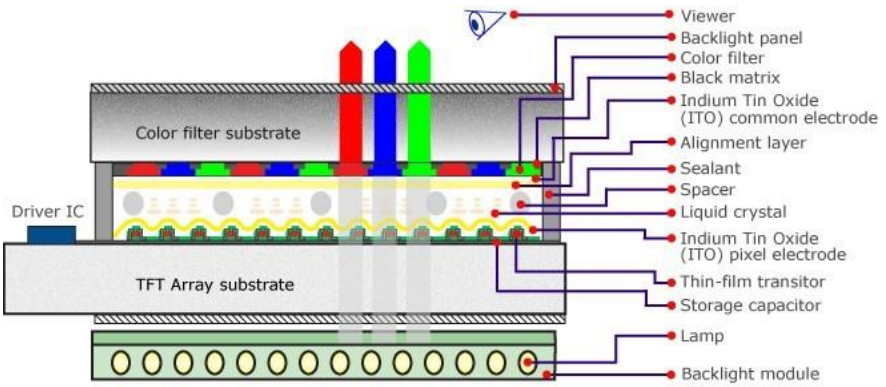


Figure 2.2: structure of TFT LCD

2.2 Pixel Schemetic

What does the schematic for one element in a pixel look like? And where is the T(transistor) in the TFT? The three letter acronym TFT stands for a thin film transistor that is physically on the top of the LCD matrix right next to each liquid crystal element. Here is a schematic model for one element in the array. C-LC represents the capacitance of the liquid crystal. CS is a storage capacitor that is used to hold the electric field across the liquid crystal when the

transistor is OFF. To apply a voltage across the LC you just turn on the gate and apply the correct voltage to the column commonly known as the source.

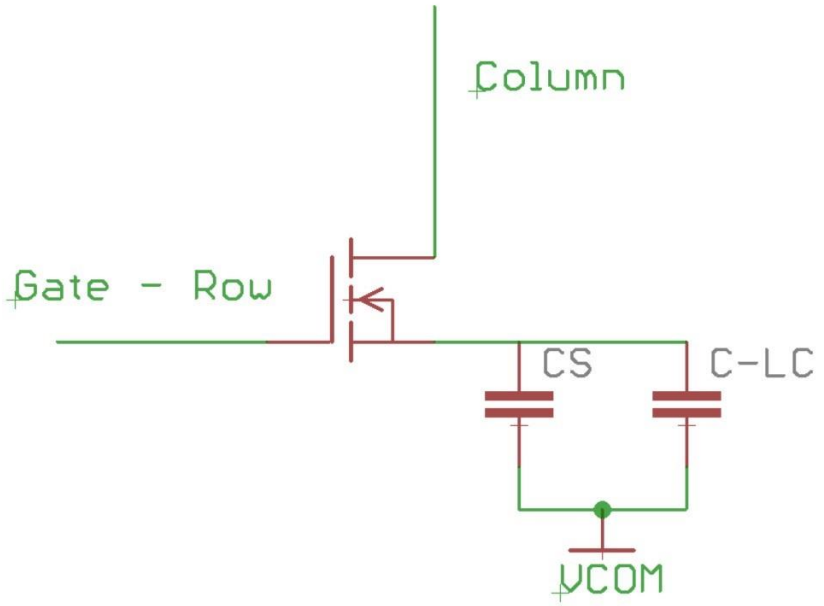


Figure 2.3: pixel schemetic

It is noticed that the “back” terminal of the two capacitors is called “VCOM” and is physically on the other side of the liquid crystal matrix from the TFT. All of the liquid crystal backsides in the display are connected to the same VCOM. A bit of painfulness in this system is that the CS capacitor leaks, which means that the LCD changes state which means that each pixel must be updated, properly called refreshed, on a regular basis.

This architecture means that every pixel in the display will require a red, green and blue element. And, you will need to control the voltage on all of the elements (which will be quite a lot on a screen of any size)

2.3 TFT Panel

Each pixel has three three thin film transistors, three capacitors, three color filters (red, green and blue) and that we need to control the voltage on the source/drain of each transistor in order to cause the right amount of light to come through the liquid crystal. How do we do that? The first step is to arrange all of the pixels in a matrix. Each row of matrix has all of the gates connected together. And each column of the matrix has all of the

sources tied together. In order to address a specific pixel RGB element, you turn on the correct row and then apply a voltage to the correct column at the right time.

If we have been thinking about this system you might have done a little bit of math and figured out that we are going to need an absolute boatload of source and gate driver signals. And you would be right! For example, a 4.3 screen with 480×272 will require 480x272x3 elements which are probably organized into 480 rows by 816 columns. This would require a chip with at least 480+816=1296 pins, that is a lot. It turns out that for small

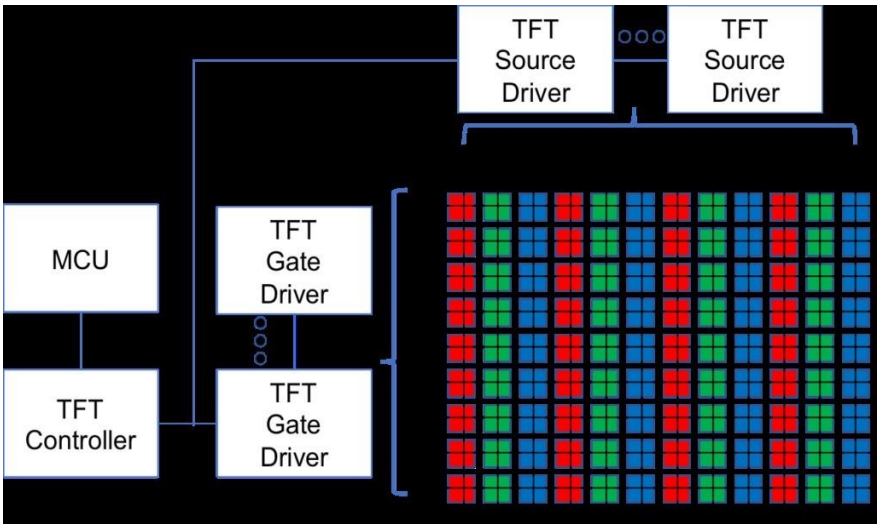


Figure 2.4: tft panel

screens ¡=3.5 there are chips with enough pins to do the job. But, for larger screens, it requires multiple chips to do the job. The "..." in the picture above shows the driver chips being cascaded. The next thing to know is that "TFT Glass" usually has the driver chip(s) embedded into the screen at the edge (we can see that in the picture from Innolux above).

2.4 TFT Gate Drivers

We must put a quite high voltage source 20v and drain voltage-10V across the liquid crystal at the right time to get it to do its thing. In order to pass that source voltage, the gate must be turned on at the right time to the right voltage, this is the purpose of the Gate Driver IC. The gate driver is conceptually simple and. We can see that it is basically a shift register,

with one element per gate. You shift in a "1" and then clock it through the entire shift register which will have the effect of applying a 1 to each gate.

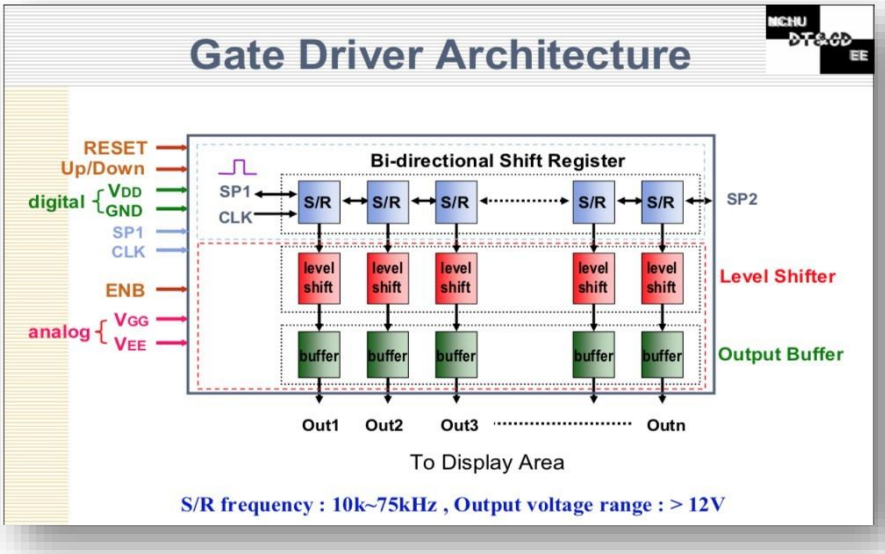


Figure 2.5: gate drivers

However, a 3.3v logic 1 is not anywhere high enough to drive the gate so that it can pass the much higher source voltage. So, you need to level shifter and a buffer to get the “right” voltage.

2.5 TFT Source Drivers

In its most basic form, the TFT source driver is responsible for taking an 8-bit digital input value representing the value of an individual LCD element and turning it into a voltage, the driving the voltage. We could conceptually have one DAC per column in the panel. But this would have at least two problems:

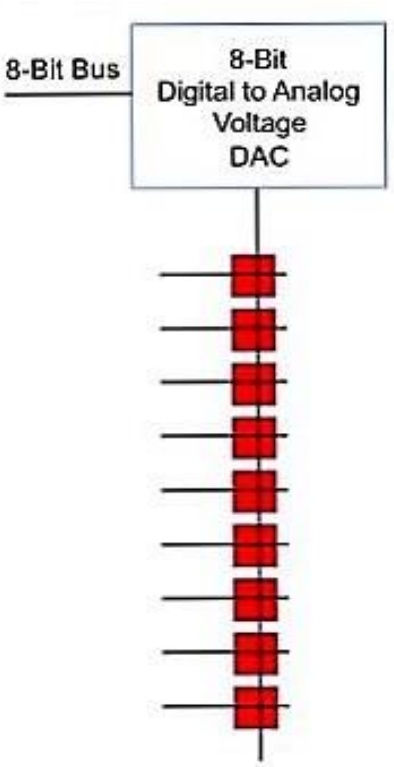


Figure 2.6: source drivers

1. The DACs are big circuits and this would make for giant source driverchips.
2. You would need to “save” all of the digital values for an entire row sothat when you turned on the row, you could turn on all of the DACs on at the same time.

You could conceptually also have one DAC for all of the columns, but this would have a bunch of problems including:

1. The DAC would have to be strong enough to drive all of the columns
2. You would need 3x the number of row drivers to effectively de-mux thecolumn
3. You would need 1 pin on the source driver per column in the panel (foran 800×600 lcd that would be 600×3 = 1800 pins).

In reality there is some compromise of chip size, number of pins and time that is made by multiplexing pins, columns and rows. For example, many of the small screens appear to have 1 column driver for all of the reds, 1 driver for the blues and one for greens.

2.6 Gamma Correction

The last issue in TFT LCD drivers is called Gamma Correction or more simply Gamma. Gamma is an intensity adjustment factor. For any given digital intensity input, you will need a non-linear translation to a voltage output on the source. For example a doubling of digital input (so that a pixel appears twice as bright) you will not double but instead will have some non-linear translation of the output voltage.

There appear to be a bunch of reason why you need Gamma Correction including at least:

1. Your eye perceives light intensity in a non-linear way
2. The LCD panel responds differently based on the input
3. The intensity variance is dependent on the color

The good news is that this gamma correction is built into the display drivers. From my reading, this is sometimes done with digital processing, and sometimes done with an analog circuit. But in general, it appears to be tuned and programmed into the driver by the panel vendor for these smaller display.

Chapter 3

Interfacing TFT Display using Arduino

3.1 Why should we interface it with Arduino?

Arduino provides a predefined libraries of ILI9341 display, GFX and SPI, so it is a comparatively very easy task to drive tft display using Arduino instead of other microcontrollers or FPGA Board.

So we use Arduino to check that Hardware is properly working or not, or to test Hardware before start implementing driver in STM32 Microcontroller.
The Adafruit GFX library with the Adafruit ILI9341 library is a popular choice for interfacing with the ILI9341 TFT display on Arduino boards. This library provides a wide range of graphical functions and enables easy communication with the ILI9341 display controller.

3.2 Pinouts and circuit diagram

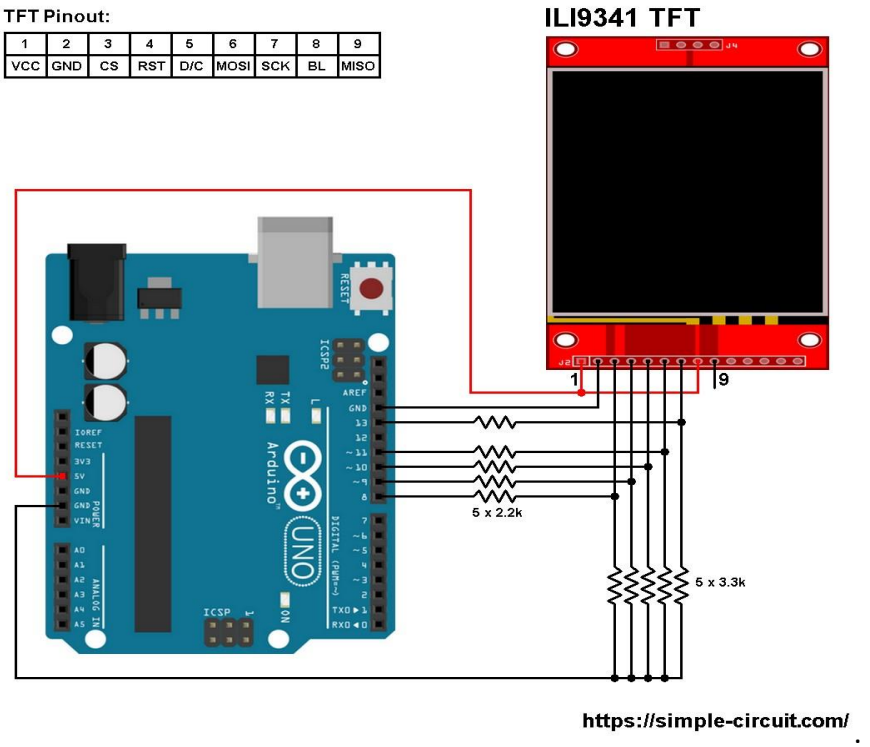


Figure 3.1: pinout and circuit diagram

3.3 code

The Adafruit ILI9341 library, along with the Adafruit GFX library, provides various functions that we can use to create different patterns on an ILI9341 TFT display using an Arduino board. The ILI9341 library handles the communication with the display driver, while the GFX library provides graphics primitives and functions to draw shapes, text, and more on the display.

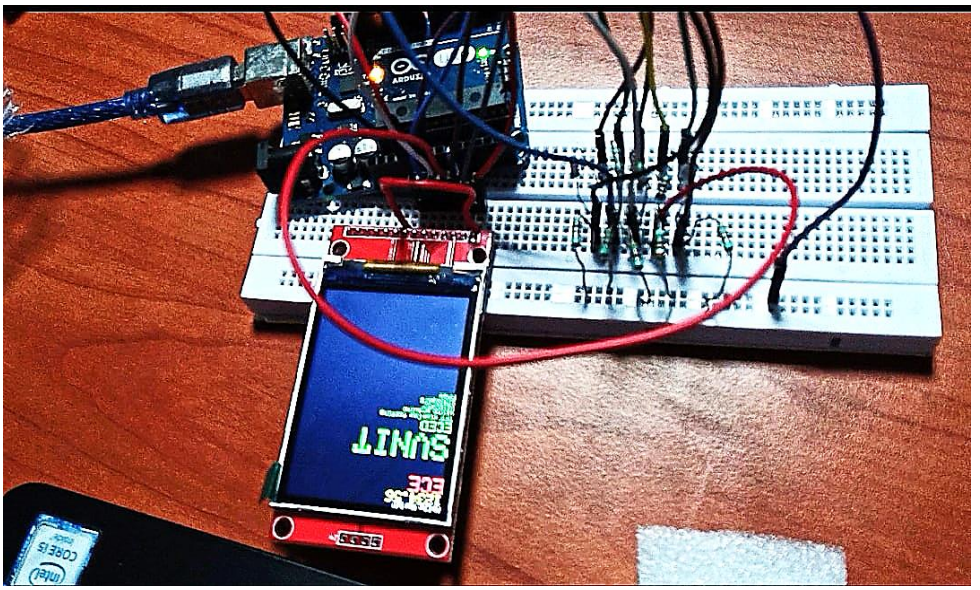
Here are some functions from these libraries that I used to create different patterns:

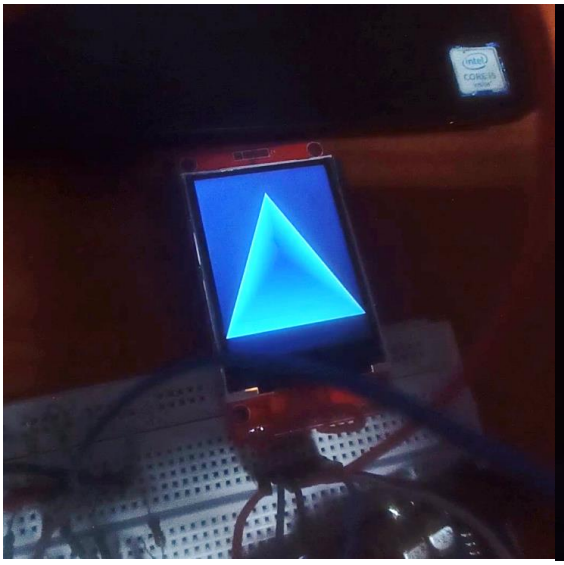
1. **Basic Shapes:** The Adafruit GFX library provides functions to draw basic shapes such as lines, rectangles, circles, triangles, and more. You can use these functions to create various patterns by combining different shapes.
   Examples:
   - `drawPixel(x, y, color)`
   - `drawLine(x0, y0, x1, y1, color)`
   - `drawRect(x, y, width, height, color)`
   - `drawCircle(x, y, radius, color)`
   - `drawTriangle(x0, y0, x1, y1, x2, y2, color)`
2. **Filled Shapes:** Similar to basic shapes, I also drawn filled versions of these shapes to create solid patterns.
   Examples:
   - `fillRect(x, y, width, height, color)`
   - `fillCircle(x, y, radius, color)`
   - `fillTriangle(x0, y0, x1, y1, x2, y2, color)`
3. **Text:** I used the Adafruit GFX library to display text in different fonts and sizes, which can be useful for creating pattern-like effects using textual elements.
   Example:
   - `setTextSize(size)`
   - `setFont(font)`
   - `setCursor(x, y)`
   - `print("Your text here")`
4. **Bitmap Images:** I created more intricate patterns by displaying bitmap images stored in arrays.
   Example:
   - `drawBitmap(x, y, bitmap, width, height, color)`
5. **Custom Patterns:** By combining various shapes, lines, text, and images, you can create your own custom patterns and designs.

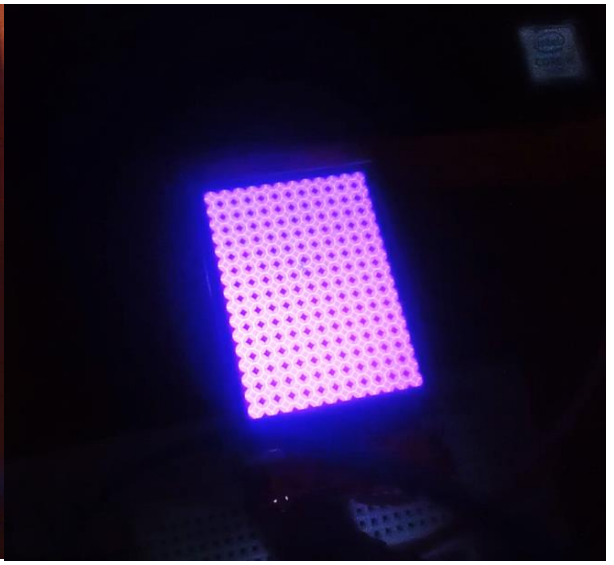
Github Link for whole code:

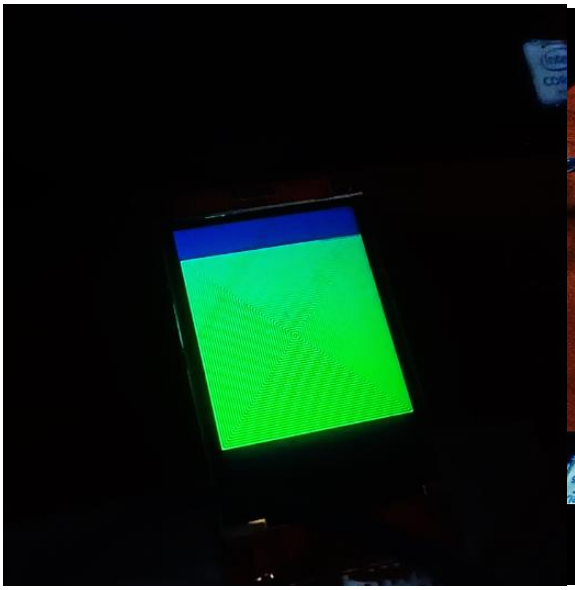
https://github.com/om01lukhi/legendary-enigma/blob/STM32G071_ILI9341_tft/sketch_may24a.ino

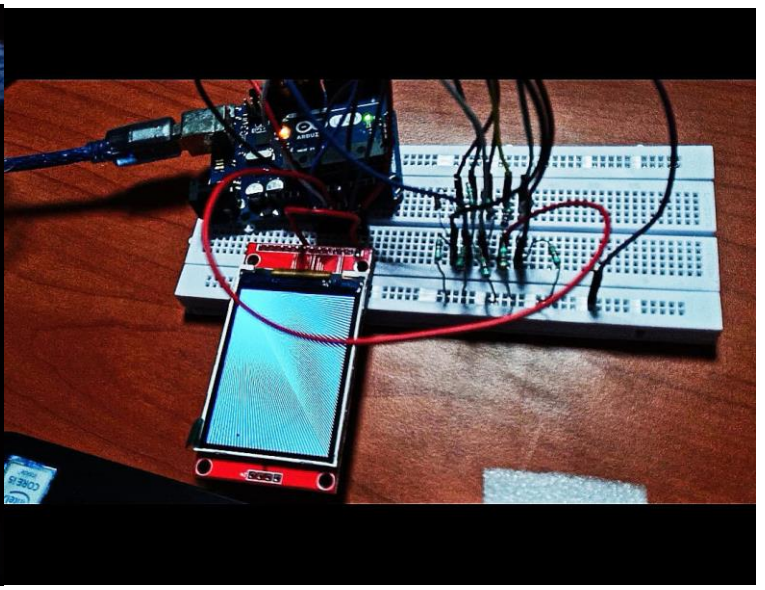
3.4 Output











Chapter 4

STM32G071 Microcontroller

4.1 Introduction

The STM32G071 is a member of the STM32G0 series of microcontrollers developed by STMicroelectronics. These microcontrollers are part of the ARM Cortex-M0+ based STM32 family, which are known for their efficiency, low power consumption, and performance in various embedded applications. The STM32G071 microcontroller is particularly designed to cater to applications that require a good balance between performance and energy efficiency.

4.2 Features of STM32G071

The STM32G071 microcontroller offers a range of features that make it suitable for a wide variety of embedded applications. Here are some key features:

- **ARM Cortex-M0+ Core:** The STM32G071 is powered by the ARM Cortex-M0+ processor, which provides efficient processing capabilities with a good balance between performance and power consumption.
- **Clock and Power Management:** The microcontroller offers various clock sources and power modes to optimize energy consumption. This includes multiple low-power modes to achieve extended battery life in battery-powered applications.
- **Memory:** It typically includes Flash memory for program storage and SRAM for data storage. The exact memory sizes can vary based on the specific variant of the microcontroller.
- **Peripherals and Communication Interfaces:**
- **Timers:** It features multiple general-purpose and advanced timers, which are essential for various timing and control tasks.
- **UART, SPI, I2C:** These communication interfaces allow the microcontroller to communicate with other devices or modules in the system.
- **Analog-to-Digital Converter (ADC):** The built-in ADC allows the microcontroller to convert analog signals into digital data for processing.
- **GPIO Pins:** The STM32G071 provides a number of General Purpose Input/Output pins, which can be configured for various purposes, including digital I/O, interrupts, and more.
- **Security Features:** Some variants of the STM32G071 come with hardware security features such as a True Random Number Generator (TRNG) and read-out protection to enhance the security of sensitive data.
- **Rich Development Ecosystem:** STMicroelectronics provides a comprehensive development ecosystem including STM32Cube software package, HAL/LL drivers, and various development tools to ease the development process.

- **Operating Voltage Range:** The microcontroller typically operates over a wide voltage range, making it suitable for various supply voltage conditions.
- **Packages:** The STM32G071 is available in different package options, including various sizes of LQFP, UFQFPN, and WLCSP packages, allowing flexibility in board design.
- **Temperature Range:** The microcontroller is designed to operate over a specific temperature range, making it suitable for various environmental conditions.

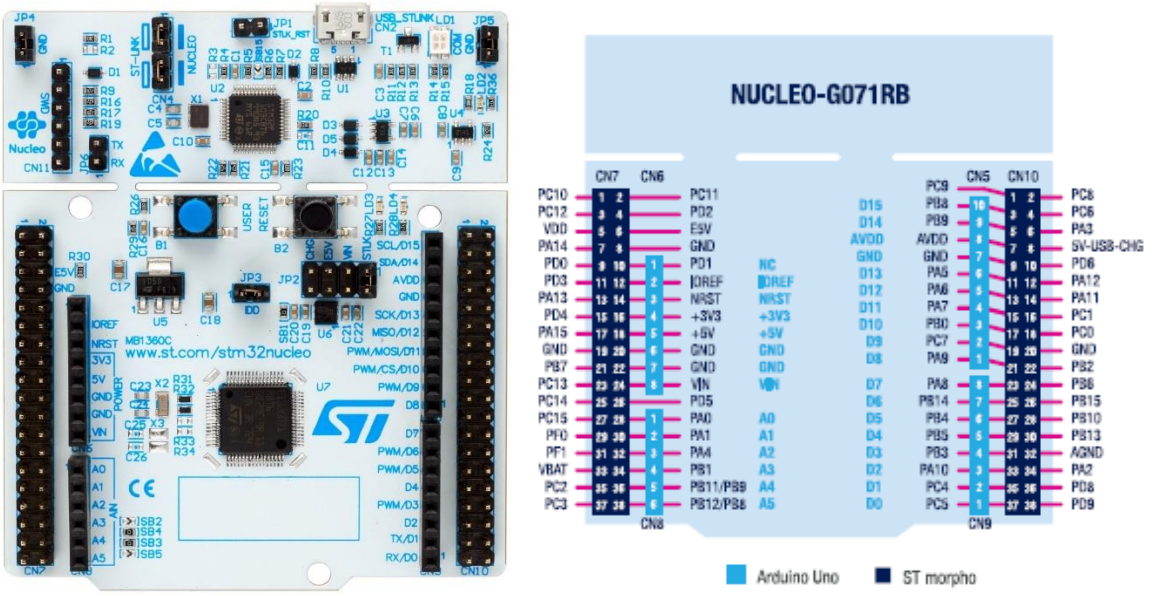


Figure 4.1: Pinout of STM32G071 MCU

4.3 Serial Peripheral Interface

We have used SPI protocol for communication between STM32G071 MCU and TFT Display ILI9341

Serial peripheral interface(SPI) is a synchronous serial communication protocol that provides full – duplex communication at very high speeds. Serial Peripheral Interface (SPI) is a master – slave type protocol that provides a simple and low cost interface between a microcontroller and its peripherals.

SPI uses a dedicated clock signal to synchronise master and slave. The clock signal must be supplied by the Master to the slave (or all the slaves in case of multiple slave setup). There are two types of triggering mechanisms on the clock signal that are used to intimate the receiver about the data: Edge Triggering and Level Triggering.The most commonly used triggering is edge triggering and there are two types: rising edge (low to high transition on the clock) and falling edge (high to low transition).

4.4 Modes of SPI

Another pair of parameters called clock polarity (CPOL) and clock phase (CPHA) determine the edges of the clock signal on which the data are driven and sampled. That means, in addition to setting the clock frequency, the master must also configure the clock polarity (CPOL) and phase (CPHA) with

respect to the data. Since the clock serves as synchronization of the data communication, there are four possible modes that can be used in an SPI protocol, based on this CPOL and CPHA.

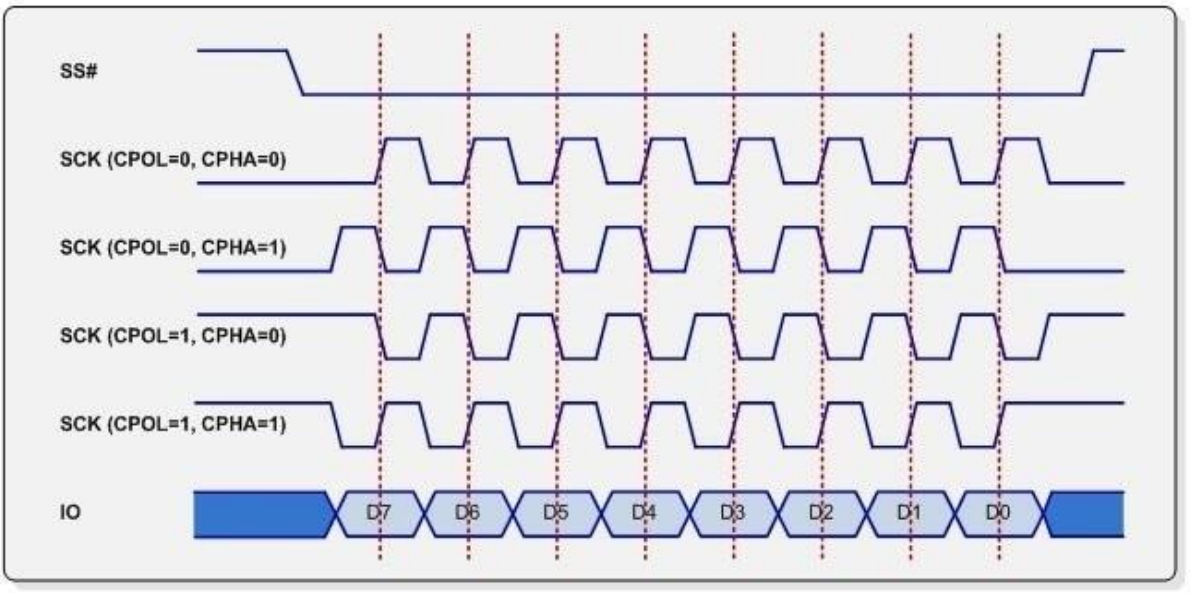


Figure 4.2: Modes of SPI

- Mode 0 : Mode 0 occurs when Clock Polarity is LOW and Clock Phase is 0 (CPOL = 0 and CPHA = 0). During Mode 0, data transmission occurs during rising edge of the clock.
- Mode 1: Mode 1 occurs when Clock Polarity is LOW and Clock Phase is 1 (CPOL = 0 and CPHA = 1). During Mode 1, data transmission occurs during falling edge of the clock.
- Mode 2: Mode 2 occurs when Clock Polarity is HIGH and Clock Phase is 0 (CPOL = 1 and CPHA = 0). During Mode 2, data transmission occurs during rising edge of the clock.
- Mode 3: Mode 3 occurs when Clock Polarity is HIGH and Clock Phase is 1 (CPOL = 1 and CPHA = 1). During Mode 3, data transmission occurs during rising edge of the clock.

4.5 SPI signals and working

SPI (Serial Peripheral Interface) is a synchronous serial communication protocol commonly used to interface microcontrollers with peripheral devices, such as sensors, memory chips, and display modules like the ILI9341 TFT display. It allows data to be exchanged between the microcontroller (master) and the peripheral device (slave) over a few wires.

In the context of interfacing the ILI9341 TFT display with the STM32G071 microcontroller using SPI, the following components are involved:

1. Master (STM32G071):The STM32G071 microcontroller acts as themaster device in the SPI communication. It controls the data exchange with the ILI9341 TFT display by generating clock signals and sending data on the MOSI (Master Output Slave Input) line.

2. Slave (ILI9341 TFT Display):The ILI9341 TFT display module acts asthe slave device in the SPI communication. It responds to the master's commands and data and sends data back to the master on the MISO (Master Input Slave Output) line (usually not used in TFT displays as they are write-only).

3. SPI Bus (Serial Bus): The SPI bus consists of four signal lines.
   SCK (Serial Clock): The master generates clock pulses to synchronize data transmission.

   MOSI (Master Output Slave Input): The master sends data to the slave on this line.

   MISO (Master Input Slave Output): The slave sends data to the master on this line (not used in write-only devices like the ILI9341).

   CS (Chip Select): This line is used to select the slave device with which the master wants to communicate. When the CS line is low, the slave is selected, and communication can occur.

SPI Communication Process:

1. The master initiates communication by pulling the CS line low, selectingthe ILI9341 TFT display.

2. The master sends a command or data to the display on the MOSI line,bit by bit, while also providing clock pulses on the SCK line.

3. The ILI9341 TFT display receives the data and interprets it as either acommand or pixel color information.

4. After sending all the required data, the master pulls the CS line high todeselect the ILI9341 TFT display, ending the communication.

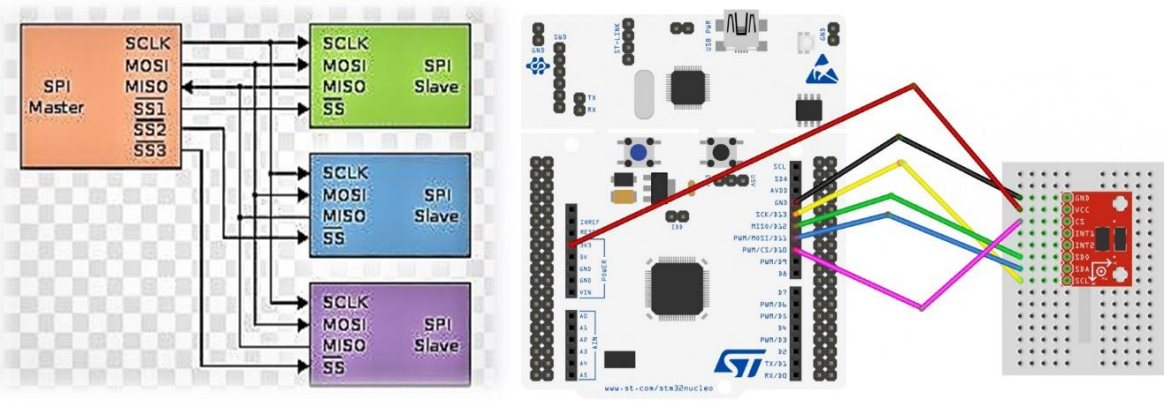


Figure 4.3: Master-Slave connection

Chapter 5

Pinouts and codes:

5.1 Pinout

1. VCC = 3.3V
2. GND = Gnd
3. CS = PB0
4. RST = PC7
5. DC = PA9
6. MOSI = PB5
7. SCK = PB3
8. LED = VCC
9. MISO = PB4

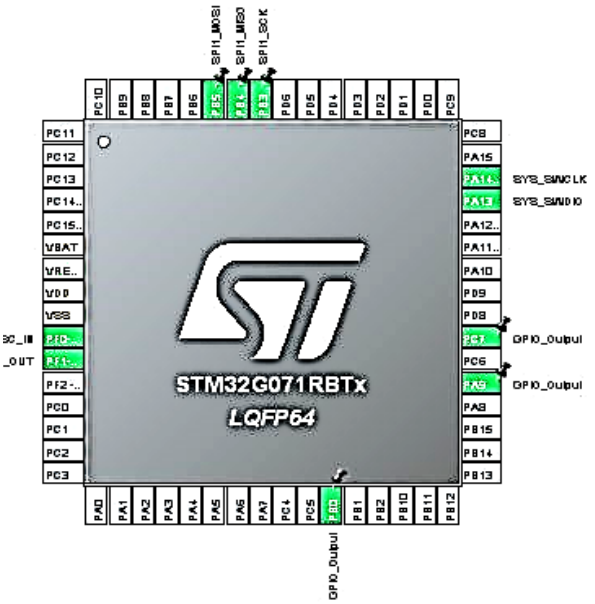


Figure 5.1: STM32 cube ide pinouts

f

f

5.2 Development environment setup

1. STM32Cube IDE Installation: Download and install STM32Cube IDE, a comprehensive development environment provided by STMicroelectronics for STM32 microcontrollers.

2. STM32CubeMX Configuration:

- Open STM32CubeMX and create a new project for the STM32G071 microcontroller.
- Configure the system clock and peripheral settings using the graphical interface.
- Enable the necessary system clock, GPIO pins for the SPI communication and control lines (such as data/command selection and reset) using the graphical interface.
- Configure the SPI peripheral for communication with the ILI9341 display.

PB0, PC7 and PA9 as GPIO output

PB5 and PB3 as MOSI and SCK respectively (we also can use PA7,PA2 as MOSI pin and PA5,PA1 as SCK)

5.3 ILI9341 Initialization

1. Initialization Sequence Overview: The initialization sequence for the ILI9341 involves sending a series of commands and data in a specific order to configure various display parameters. The sequence typically includes steps like software reset, power control, memory access control, and gamma correction.

2. Implementing Initialization:

Here's how I implemented the initialization sequence for the ILI9341 TFT display using the STM32G071 microcontroller:

```
#include "ILI9341_STM32_Driver.h"
volatile uint16_t LCD_HEIGHT = ILI9341_SCREEN_HEIGHT;
volatile uint16_t LCD_WIDTH        = ILI9341_SCREEN_WIDTH;
void HAL_SPI_TxCpltCallback(SPI_HandleTypeDef *hspi)
{
 /* Deselect when Tx Complete */
 if(hspi == HSPI_INSTANCE)
 {
 HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);
 }
}
static void ILI9341_SPI_Tx(uint8_t data)
{
while(!__HAL_SPI_GET_FLAG(HSPI_INSTANCE, SPI_FLAG_TXE));
HAL_SPI_Transmit_DMA(HSPI_INSTANCE, &data, 1);
//HAL_SPI_Transmit(HSPI_INSTANCE, &data, 1, 10);
}
static void ILI9341_SPI_TxBuffer(uint8_t *buffer, uint16_t len)
{
while(!__HAL_SPI_GET_FLAG(HSPI_INSTANCE, SPI_FLAG_TXE));
HAL_SPI_Transmit_DMA(HSPI_INSTANCE, buffer, len);
//HAL_SPI_Transmit(HSPI_INSTANCE, buffer, len, 10);
}
void ILI9341_WriteCommand(uint8_t cmd)
{
HAL_GPIO_WritePin(LCD_DC_PORT, LCD_DC_PIN, GPIO_PIN_RESET);  //command
```

```
HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);   //select
ILI9341_SPI_Tx(cmd);
//HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);    //deselect
}
void ILI9341_WriteData(uint8_t data)
{
HAL_GPIO_WritePin(LCD_DC_PORT, LCD_DC_PIN, GPIO_PIN_SET);     //data
HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);   //select
ILI9341_SPI_Tx(data);
//HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);    //deselect
}
void ILI9341_WriteBuffer(uint8_t *buffer, uint16_t len)
{
HAL_GPIO_WritePin(LCD_DC_PORT, LCD_DC_PIN, GPIO_PIN_SET);     //data
HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);   //select
ILI9341_SPI_TxBuffer(buffer, len);
//HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);    //deselect
}
void ILI9341_SetAddress(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)
{
uint8_t buffer[4];
buffer[0] = x1 >> 8;
buffer[1] = x1;
buffer[2] = x2 >> 8;
buffer[3] = x2;
ILI9341_WriteCommand(0x2A);
ILI9341_WriteBuffer(buffer, sizeof(buffer));

buffer[0] = y1 >> 8;
buffer[1] = y1;
buffer[2] = y2 >> 8;
buffer[3] = y2;
```

```
ILI9341_WriteCommand(0x2B);
ILI9341_WriteBuffer(buffer, sizeof(buffer));
ILI9341_WriteCommand(0x2C);
}
void ILI9341_Reset(void)
{
HAL_GPIO_WritePin(LCD_RST_PORT, LCD_RST_PIN, GPIO_PIN_RESET);        //Disable
HAL_Delay(10);
HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_RESET);          //Select
HAL_Delay(10);
HAL_GPIO_WritePin(LCD_RST_PORT, LCD_RST_PIN, GPIO_PIN_SET);          //Enable
HAL_GPIO_WritePin(LCD_CS_PORT, LCD_CS_PIN, GPIO_PIN_SET);            //Deselect
}
void ILI9341_Enable(void)
{
HAL_GPIO_WritePin(LCD_RST_PORT, LCD_RST_PIN, GPIO_PIN_SET);          //Enable
}

void ILI9341_Init(void)
{
HAL_Delay(50);
ILI9341_Enable();
HAL_Delay(50);
ILI9341_Reset();
//SOFTWARE RESET
ILI9341_WriteCommand(0x01);
//POWER CONTROL A
ILI9341_WriteCommand(0xCB);
ILI9341_WriteData(0x39);
ILI9341_WriteData(0x2C);
ILI9341_WriteData(0x00);
ILI9341_WriteData(0x34);
ILI9341_WriteData(0x02); //….. more data and codes for Power control B
```

```
//DRIVER TIMING CONTROL A

ILI9341_WriteCommand(0xE8);
ILI9341_WriteData(0x85);
ILI9341_WriteData(0x00);
…….more

//POWER ON SEQUENCE CONTROL
   ILI9341_WriteCommand(0xED);
ILI9341_WriteData(0x64);
ILI9341_WriteData(0x03);
……
//PUMP RATIO CONTROL
ILI9341_WriteCommand(0xF7);
ILI9341_WriteData(0x20);

//POWER CONTROL,VRH[5:0]
ILI9341_WriteCommand(0xC0);
ILI9341_WriteData(0x10);

//POWER CONTROL,SAP[2:0];BT[3:0]
ILI9341_WriteCommand(0xC1);
ILI9341_WriteData(0x10);

//VCM CONTROL
ILI9341_WriteCommand(0xC5);
ILI9341_WriteData(0x3E);
ILI9341_WriteData(0x28);

//VCM CONTROL 2
ILI9341_WriteCommand(0xC7);
ILI9341_WriteData(0x86);
```

```
//MEMORY ACCESS CONTROL
ILI9341_WriteCommand(0x36);
HAL_Delay(50);
ILI9341_WriteData(0x48);

//PIXEL FORMAT
HAL_Delay(50);
ILI9341_WriteCommand(0x3A);
HAL_Delay(50);
ILI9341_WriteData(0x55);

//FRAME RATIO CONTROL, STANDARD RGB COLOR
ILI9341_WriteCommand(0xB1);
ILI9341_WriteData(0x00);
ILI9341_WriteData(0x18);

//DISPLAY FUNCTION CONTROL
ILI9341_WriteCommand(0xB6);
ILI9341_WriteData(0x08);
ILI9341_WriteData(0x82);
……

//3GAMMA FUNCTION DISABLE
ILI9341_WriteCommand(0xF2);
ILI9341_WriteData(0x00);

//GAMMA CURVE SELECTED
ILI9341_WriteCommand(0x26);
ILI9341_WriteData(0x01);

//POSITIVE GAMMA CORRECTION
ILI9341_WriteCommand(0xE0);
```

```
ILI9341_WriteData(0x0F); // …….

//NEGATIVE GAMMA CORRECTION
ILI9341_WriteCommand(0xE1);
ILI9341_WriteData(0x00);
ILI9341_WriteData(0x0E);
……

//EXIT SLEEP
ILI9341_WriteCommand(0x11);
HAL_Delay(500);

//TURN ON DISPLAY
ILI9341_WriteCommand(0x29);
ILI9341_WriteData(0x0F);
```

1. All complete data and commands for completion of particular task is given to code with the help of data sheet of TFT Display ILI9341. And Many other very important functions like setRotation, drawcolor, drawcolorBurst, setAddress, fillScreen, drawPixel, drawHline, drawVline, etc are implemented using datasheet and this whole code of drivers for tft display is given in following link:

https://github.com/om01lukhi/legendary-enigma/blob/main/ILI9341_STM32_Driver.h

https://github.com/om01lukhi/legendary-enigma/blob/main/ILI9341_STM32_Driver.c

2. ILI9341_GFX.c Library: The ILI9341_GFX.c library facilitates drawing basic shapes and handling graphical operations on the ILI9341 TFT display. Its key features include:

Drawing basic shapes: Lines, rectangles, circles, triangles, etc.

Filling shapes with colors.

Setting text and background colors.

Specifying drawing coordinates and sizes.

https://github.com/om01lukhi/legendary-enigma/blob/main/ILI9341_GFX.h

https://github.com/om01lukhi/legendary-enigma/blob/main/ILI9341_GFX.c

3. ILI9341_FONTS.c Library: The ILI9341_FONTS.c library enhances the user interface by enabling text rendering with various fonts on the ILI9341 TFT display. It offers:

Support for different font sizes and types.

Font rendering functions for printing text on the display.

Handling text alignment and positioning.

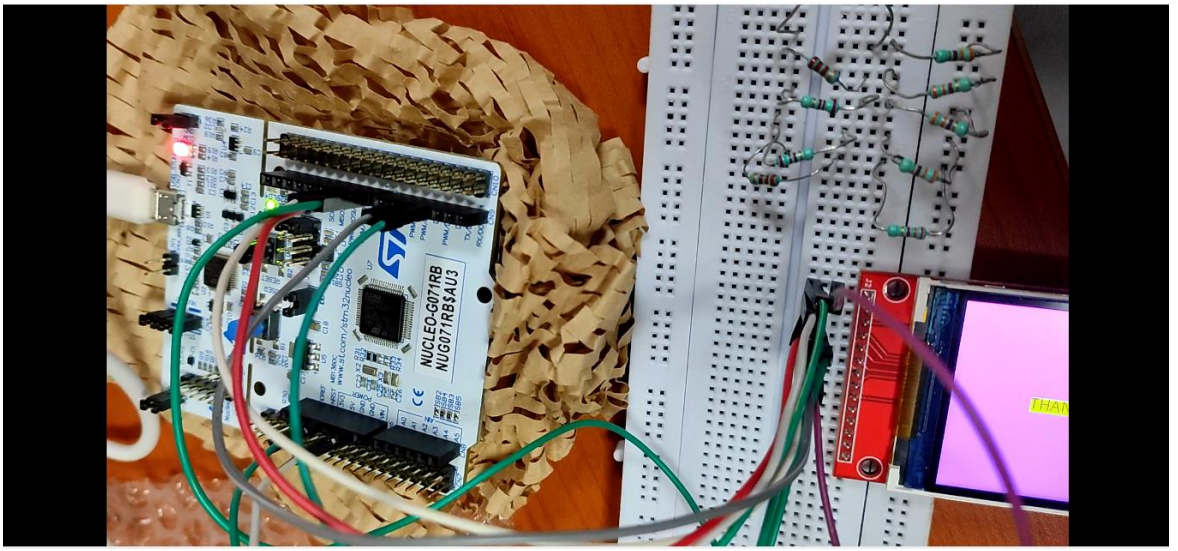
https://github.com/om01lukhi/legendary-enigma/blob/main/fonts.h

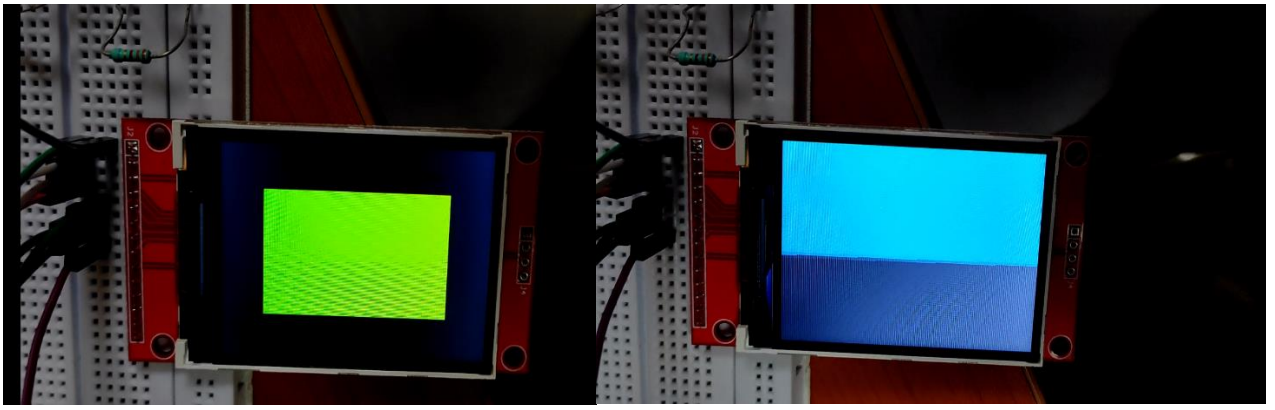
https://github.com/om01lukhi/legendary-enigma/blob/main/fonts.c

At last these libraries of GFX, driver and fonts were used in main file, which was able to complete several tasks like initializing the display, loading fonts, and utilizing various graphical functions.

https://github.com/om01lukhi/legendary-enigma/blob/main/main.c

5.4 Output and Conclusion





5.5 Learning and future probabilities

This project served as an educational opportunity to understand the importance of display drivers in embedded systems and microcontroller-based projects. By delving into the complexities of graphics rendering and font management, we gained insights into creating visually engaging user interfaces. As a result, we are now equipped to explore more advanced features, such as touch screen integration, color calibration, and interactive graphical applications.
In conclusion, the successful implementation of ILI9341 TFT display drivers using both the Arduino platform and the STM32G071 microcontroller highlights the versatility and adaptability of these technologies in realizing graphical interfaces for a wide range of applications. The knowledge gained from this project can be harnessed to enhance future projects and ventures in the realm of embedded systems and human-machine interaction.

References

1. https://iotexpert.com/embedded-graphics-tft-displays-drivers/
2. https://simple-circuit.com/interfacing-arduino-ili9341-tft-display/
3. https://www.micropeta.com/video37