P.E.S COLLEGE OF ENGINEERING, MANDYA

(An Autonomous Institute under Visvesvaraya Technological University, Belagavi)

AN INTERNSHIP REPORT ON

"SPI–BASED TFT DISPLAY CONTROLLER"

Submitted in partial fulfillment of the
requirementFor the award of the

BACHELOR OF ENGINEERING DEGREE

Submitted by

KAUSHIK. C	**4PS21EC054**
KAVYA. K. S	**4PS21EC057**
JANYA. L. S	**4PS21EC048**
MANU. H. P	**4PS21EC167**

Under the guidance of

Ms. NISCHITHA. K
(Assistant Professor)

Department of Electronics and Communication Engineering
P.E.S. College of Engineering, Mandya.
2023-2024

P.E.S COLLEGE OF ENGINEERING

MANDYA-571401

(An Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

This is to certify that,

JANYA L. S	**4PS21EC048**
KAVYA K. S	**4PS21EC057**
KAUSHIK. C	**4PS21EC054**
MANU H. P	**4PS21EC167**

have successfully completed internship on "Embedded and Automotive" and completed work entitled **"SPI-based TFT Display Controller"** in partial fulfillment for the award of degree of **Bachelor of Engineering in Electronics and communication Engineering** of **P.E.S college of Engineering, Mandya, VTU Belagavi** during the year **2023- 2024**. The project has been approved as it satisfies the academic requirements in respect of project work prescribed for the degree **in Bachelor of Engineering.**

Signature of the guide
Nischitha K
Assistant Professor

Signature of the HOD
Dr .Punith Kumar M B
Professor & HOD

Dr .H M Nanjundaswamy
Principal
PES College of Engineering, Mandya

Project work viva-voice examination			
Sl.No	Examiners		Date
	Name	Signature	
1			
2			

P.E.S COLLEGE OF ENGINEERING

(An Autonomous Institute under VTU, Belagavi)

MANDYA-571401

Department of Electronics and Communication Engineering

DECLARATION

We KAUSHIK C, KAVYA K S, JANYA L S, MANU H P students of the 5th semester Bachelor of Engineering in Electronics & Communication, PESCE, Mandya, hereby declare that the project work presented in the dissertation entitled "SPI BASED TFT DISPLAY CONTROLLER" is an authentic record of the work that has been independently carried out by us and submitted in partial fulfilment of the requirements for the award of degree in semester Bachelor of Engineering in Electronics & Communication, affiliated to Visvesvaraya Technological University (VTU), Belagavi during the year 2022-2023.

The work contained in the thesis has not been submitted in part or full to any other university or institution or professional body for the award of any other degree or any fellowship.

Place: Mandya

Date: 26/03/2024

KAUSHIK C

KAVYA K S

JANYA L S

MANU H P

ABSTRACT

This report describes the work we did during our Skill Lync work, where we focused on the project called " TFT Over SPI Representation: Embedded Systems Approach". This project is designed to create and implement a TFT display using a Serial Peripheral Interface (SPI) in the system area. The report summarizes the theoretical concepts, practical applications, challenges encountered, and solutions developed during the project. Provides an overview of the learning experiences and skills gained during our time at Skill Lync. The report also highlights the importance of excellence in understanding and applying complex engineering concepts.

TABLE OF CONTENTS

CHAPTER 1

ABOUT THE COMPANY

1.1 SKILL-LYNC: AN OVERVIEW

Skill-Lync, founded in 2019, emerged as a response to the growing need for industry-relevant education in the engineering sector. The organization&#39; founders envisioned a platform that could equip students with practical skills and hands-on experience, ensuring their readiness for the demands of the modern workforce.

1.2 BRIEF HISTORY

Over the years, Skill-Lync has achieved significant milestones, including the expansion of its course catalogue to encompass a diverse range of engineering disciplines, including Electronics and Communication Engineering (ECE). This expansion marked a key turning point in Skill-Lync &#39; journey, solidifying its reputation as a leader in technical education.

1.3 OVERALL ORGANIZATION STRUCTURE:

Skill-Lync operates within a well-defined organizational structure designed to optimize efficiency and effectiveness. The company is structured into several key departments, each playing a vital role in the delivery of high-quality education and services to students. These departments include Course Development, responsible for designing and updating courses based on industry trends and feedback; Student Support, dedicated to providing guidance, resolving queries, and ensuring a seamless learning experience; Sales and Marketing, focused on promoting courses, reaching out to potential students, and establishing partnerships; Operations, managing day-to-day activities, logistics, and coordination between departments; and Research and Development, engaged in continuous improvement and innovation to enhance course content and stay at the forefront of industry advancements.

1.4 PRODUCTS AND SERVICES:

Skill-Lync offers a comprehensive suite of products and services tailored specifically for Electronics and Communication Engineering students. These include industry-driven courses covering a wide array of topics such as circuit design, signal processing, embedded systems, VLSI design, communication networks, and more. Additionally, the company provides hands-on projects and simulations that allow students to apply theoretical knowledge to practical scenarios, enhancing their problem-solving skills and practical understanding. Skill-Lync also offers internship and placement support, resume building, interview preparation, and customized corporate training programs, ensuring that students are well-equipped for success in their careers.

1.5 WORKFORCE SIZE AND FINANCIAL DETAILS:

Skill-Lync boasts a dedicated team of professionals working across various departments to support its operations and deliver exceptional services to students. While specific financial details such as turnover or operational costs are not publicly disclosed on the website, Skill-Lync&#39; success is evident from its growing student base, positive reputation, and strong industry partnerships. The organization&#39; workforce is comprised of experts in their respective fields, including educators, industry professionals, and support staff, all committed to providing students with a top-notch learning experience.

1.6 Operation of Different Departments:

- Course Development: Designs and updates courses based on industry trends, emerging technologies, and feedback from industry experts and students.

- Student Support: Provides personalized guidance, resolves queries, and ensures a smooth learning journey for students, offering continuous assistance throughout their course duration.

- Sales and Marketing: Strategically promotes courses, conducts outreach campaigns, and establishes collaborations with educational institutions, industry partners, and potential students.

- Operations: Manages logistical aspects, oversees administrative functions, and maintains effective communication and coordination between departments to ensure seamless operations.

- Research and Development: Engages in ongoing research initiatives, collaborates with industry partners, and explores innovative teaching methodologies and technologies to enhance the learning experience and course content.

CHAPTER 2

SCHEDULE OF TRAINING AND INTERNSHIP

Day	Topic	Learning Objectives	Category	Hrs
Monday 22/01/2024	Introduction to H/w and S/w used in the courses	Theoretical introduction on ARM on-chip peripherals	Delivery	1
	Different ways of Driver development	CMSIS Vs HAL VS Baremetal	Delivery	1
	Data Sheet Reference and Locating register Memory addresses	Introduction, Core Features, Version History, Data FlowModel, Registers	Delivery	2
	Understanding Memory MAP, MCU Clock Systems and Details/ Enabling and Disabling,	CPU Modes, Memory Organization, Interrupts, Pipelining, Addressing Modes, ARM Embedded C language Implementation, Exposure to an ARM7 CPU, Core BasedMicrocontroller	Delivery	2
Tuesday 23/01/2024	Resources and setting up IDE, ARM Peripheral Driver Development	Setup the tool chain for programming and understanding about the components in the target board	Hands on	1
	DIO HAL Programming	DIO - HAL Example - Debugging	Hands on	3
	GPIO Interrupt Programming	Introduction to Interrupts - GPIO Interrupt Configuration -HAL Example - Debugging	Hands on	2
Wednesday 24/01/2024	Introduction to CMSIS and DIO	Understanding CMSIS library - DIO Example	Hands on	2
	DIO Baremetal Programming	DIO - HAL & Baremetal Example - Debugging	Hands on	3
	Introduction to Timer and baremetal programming	Timer Driver Development Timer Driver Development using PWM, Input Capture, Output Compare	Delivery	1
Thursday 25/01/2024	HAL Example - Debugging	Timer development using HAL library for Delay	Hands on	1
		Timer development using HAL library for PWM	Hands on	2
		Timer development using HAL library for Input Capture and Output Capture	Hands on	3
Saturday 27/01/2024	DMA operation	Introduction to DMA Operation	Delivery	2
	ADC Programming	Introduction to ADC - ADC Driver Development (Polling, Interrupt, DMA) - HAL Example - Debugging	Hands on	4

CHAPTER 3
TASK PERFORMED

3.1 PROBLEM STATEMENT

Designing a SPI-BASED TFT DISPLAY controller

3.2 PROBLEM DESCRIPTION

In the field of embedded systems, display controllers play a crucial role in interfacing microcontrollers with display devices. One of the common protocols used for this purpose is the Serial Peripheral Interface (SPI) protocol. This project revolves around the design and implementation of a TFT (Thin-Film-Transistor) display controller using the SPI protocol.

The problem arises due to the need for a controller that can handle high-speed data transmission, accurately display the transmitted data on a TFT screen, and operate reliably under various conditions. The challenge lies in understanding the SPI protocol, the workings of TFT displays, and the interaction between the two. The goal is to apply this knowledge to design a practical solution that is efficient, reliable, and suitable for use in real-world embedded systems.

Therefore, the problem statement sets a clear goal for the project: to design and implement an SPI-based TFT display controller that meets the stated requirements. The success of the project can be evaluated based on whether the resulting display controller fulfills these requirements. This problem statement provides a direction for the project and serves as a guide for the work to be done. It also helps in evaluating the success of the project upon its completion.

3.3 DESIGN AND DEVELOPMENT

The project aimed to design and implement a Thin-Film-Transistor (TFT) display controller using the Serial Peripheral Interface (SPI) protocol. The controller was required to handle high-speed data transmission, accurately display the transmitted data on a TFT screen, and operate reliably under various conditions.

The design phase involved understanding the SPI protocol and the workings of TFT displays. The SPI protocol is a synchronous serial communication interface specification used for short-distance communication, primarily in embedded systems. The TFT technology is used for the liquid crystal display (LCD) panels to improve image qualities such as addressability and contrast.

A microcontroller was chosen to act as the master device in the SPI protocol, controlling the data flow by generating the clock signal and enabling the slave select lines as required. The TFT display acted as the slave device, receiving data from the microcontroller to generate the necessary display.

The development phase involved implementing the design using a combination of hardware components and software programming. The microcontroller was connected to the TFT display using the SPI bus, which included data lines, clock line, and slave select lines.

The software was developed in C, utilizing the SPI protocol for communication between the microcontroller and the display. The software controlled the data flow, sending pixel data to the display, and controlling the refresh rate to ensure a stable image was displayed.

The development process involved several iterations of coding, testing, debugging, and optimization to ensure the display controller operated reliably and met the project requirements.

The final phase of the project was testing and validation. The display controller was tested under various conditions to ensure it could handle high-speed data transmission and accurately display the transmitted data on the TFT screen. The tests included checking the clarity of the display, the stability of the images, and the reliability of the data transmission.

The display controller successfully passed all tests, validating the design and development process, and confirming that the project objectives had been met.

3.3.1 BLOCK DIAGRAM

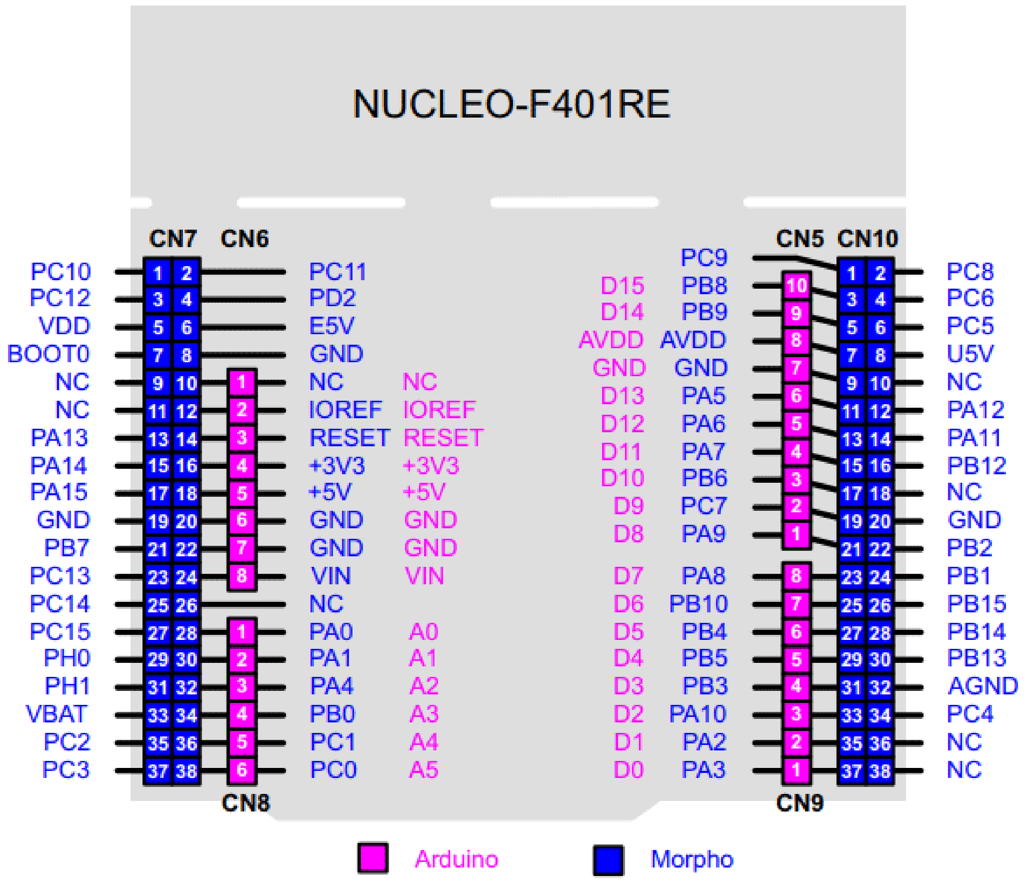

NUCLEO-F401RE
CN7
CN6
CN5
CN10
CN8
CN9
PC10 PC12 VDD BOOT0 NC NC PA13 PA14 PA15 GND PB7 PC13 PC14 PC15 PH0 PH1 VBAT PC2 PC3
PC11 PD2 E5V GND NC IOREF RESET +3V3 +5V GND GND VIN NC PA0 PA1 PA4 PB0 PC1 PC0
NC IOREF RESET +3V3 +5V GND GND VIN
A0 A1 A2 A3 A4 A5
D15 D14 AVDD GND D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
PC9 PB8 PB9 AVDD GND PA5 PA6 PA7 PB6 PC7 PA9 PA8 PB10 PB4 PB5 PB3 PA10 PA2 PA3
PC8 PC6 PC5 U5V NC PA12 PA11 PB12 NC GND PB2 PB1 PB15 PB14 PB13 AGND PC4 NC NC
Arduino
Morpho

FIGURE 3.1 Pin diagram of STM32 Nucleo-F401RE

The STM32 Nucleo-F401RE board is a development board that features the ARM Cortex M4 32-bit STM32F401RET6 microcontroller. The pinout of this board is similar to Arduino UNO, which makes it compatible with many Arduino shields.

Here's a brief explanation of the pin diagram:

Arduino Headers (CN5, CN6, CN8, CN9): These pins are female connector pins which exactly match the order and position of Arduino UNO pins1. They are divided into different categories:

Power Pins (CN6): These include IOREF, RESET, +3.3V, +5V, and GND.

Analog Pins and I2C (CN8): These include A0-A1 for measuring analog voltage and A4 (SDA) and A5 (SCL) for I2C communication.

Digital Pins and SPI (CN5): These include D8-D15 digital GPIO pins and D13 (SCK), D12 (MISO), D11 (MOSI), and D10 (CS) for SPI communication.

Digital Pins and USART (CN9): These include D0-D7 digital GPIO pins and D0 (Rx) and D1 (Tx) for USART communication.

STM32 Headers (CN7, CN10): These are male headers on either side of the board. They comprise of GPIO pins, Analog Pins, Timer Pins, and Power pins.

LEDs: There are three LEDs, where LD1 indicates USB communication, LD2 is programmable, and LD3 indicates power.

Push Buttons: There are two push buttons where one is user programmable, and the other is to reset the Microcontroller.

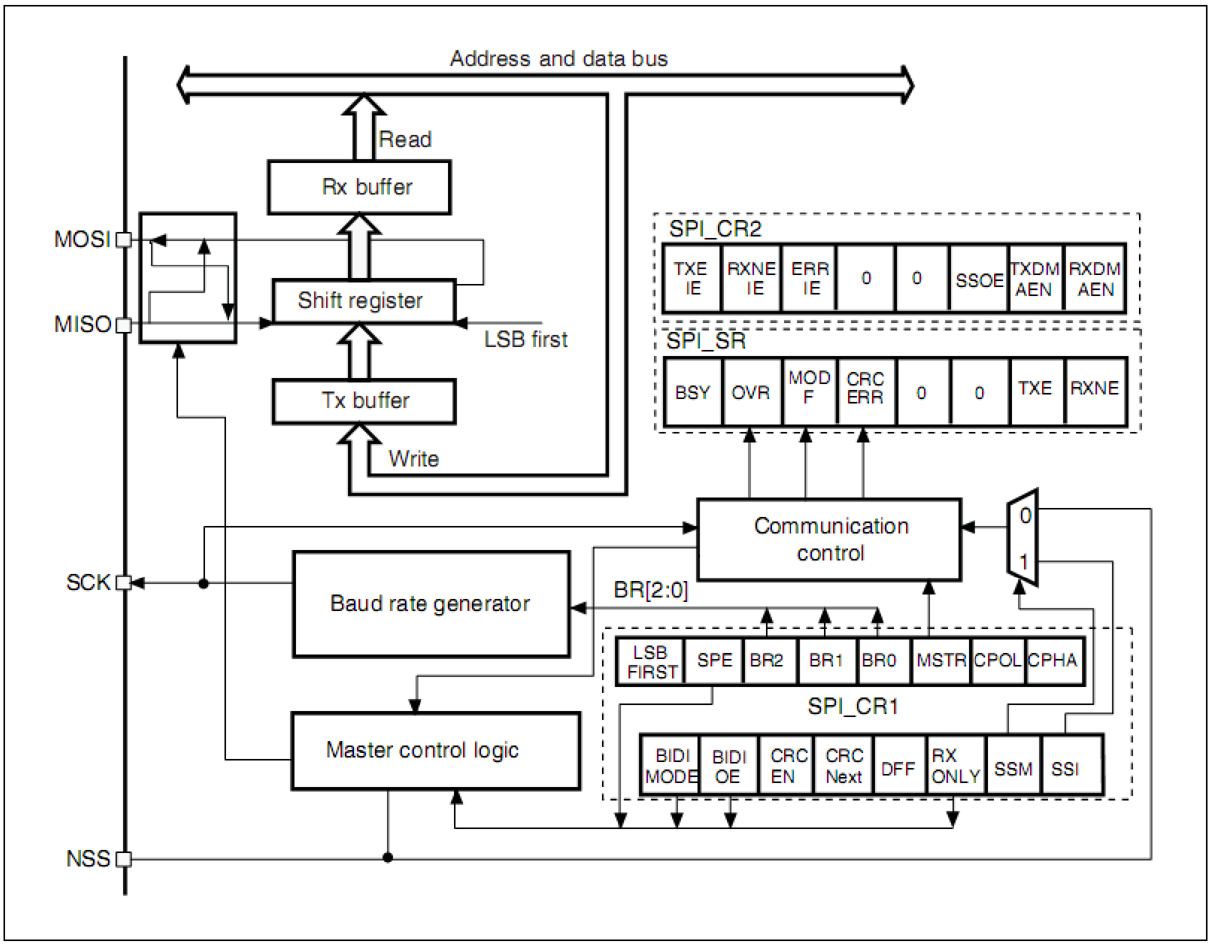


FIGURE 3.2 Block diagram of SPI controller.

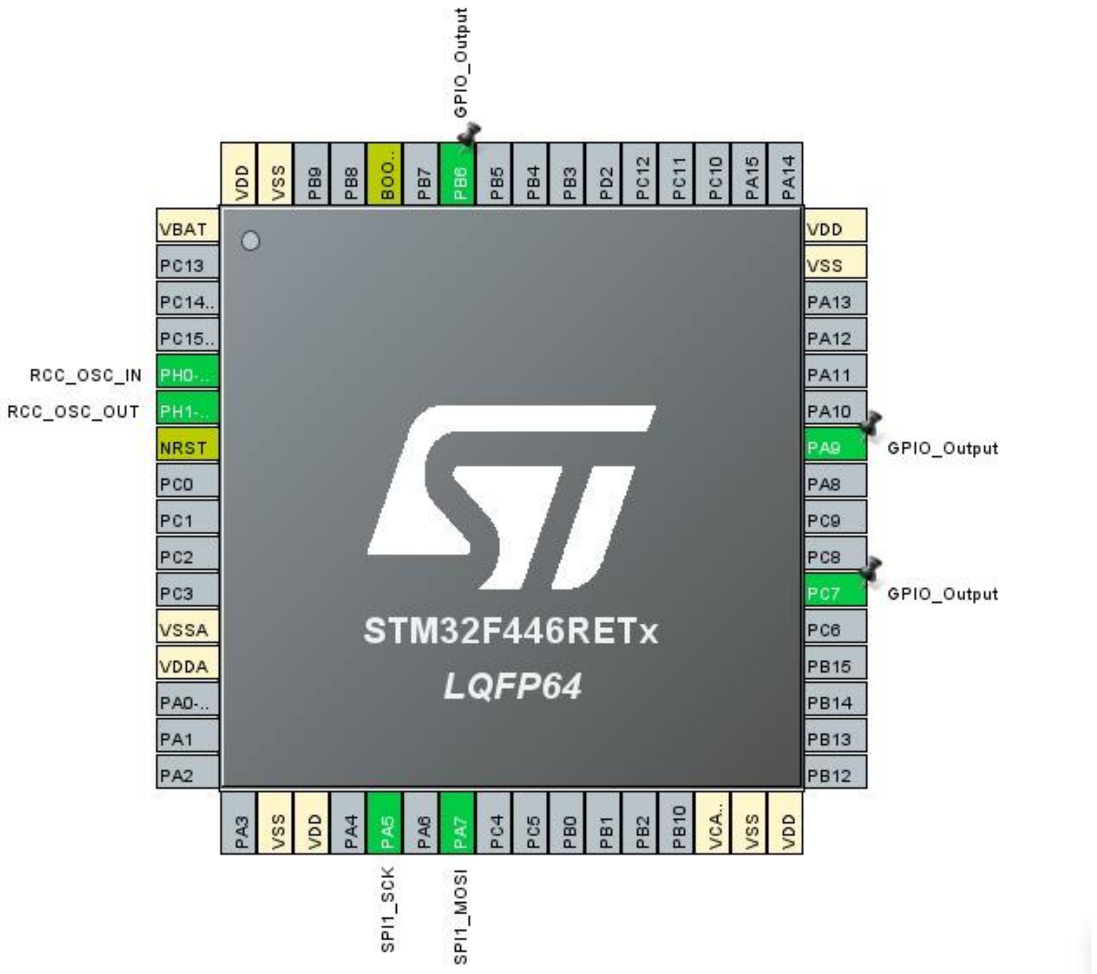


FIGURE 3.3 PINOUT AND CONFIGURATION

Here in the pinout and configuration we have selected GPIO_Output, PA9 [GPIO_OUTPUT], PC7[GPIO_OUTPUT].

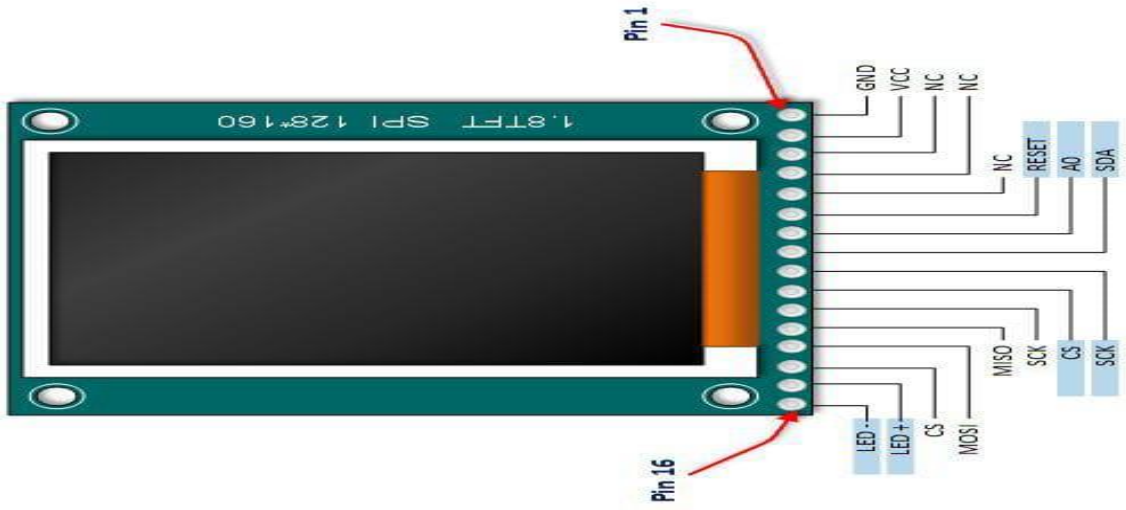


FIGURE 3.4 PIN DIAGRAM OF TFT DISPLAY

3.3.2 FLOW CHART

a flowchart for an SPI-based TFT Display Controller involves illustrating the sequence of steps and decisions involved in controlling the TFT display using the Serial Peripheral Interface (SPI). Here's a simplified representation of the flowchart:

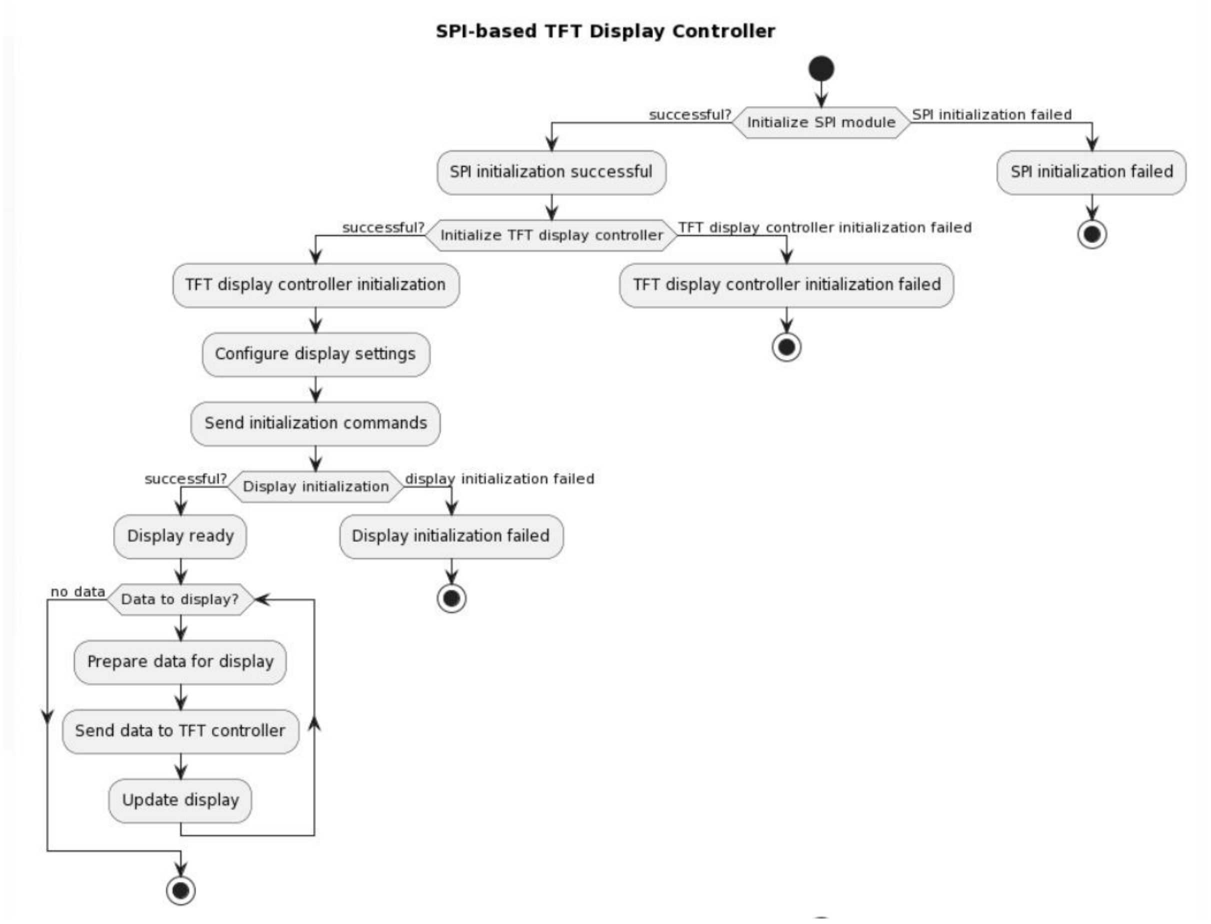

SPI-based TFT Display Controller
successful?
Initialize SPI module
SPI initialization failed
SPI initialization successful
SPI initialization failed
successful?
Initialize TFT display controller
TFT display controller initialization failed
TFT display controller initialization
TFT display controller initialization failed
Configure display settings
Send initialization commands
successful?
Display initialization
display initialization failed
Display ready
Display initialization failed
no data
Data to display?
Prepare data for display
Send data to TFT controller
Update display

FIGURE 3.5 FLOW CHART

1. Start: Begin the flowchart.

2. Initialize SPI: Initialize the SPI communication protocol for communication with the TFT display.

3. Initialize TFT Display: Set up the TFT display for communication and configuration.

4. Set Display Parameters: Define parameters such as resolution, color depth, orientation, etc., based on display requirements.

5. Display Content:

   - Draw Shapes/Text: Draw various shapes, text, images, etc., on the display.

- Update Display: Update the display content as required.

6. User Interaction:

   - Read Inputs: Check for any user inputs or sensor data.

   - Process Inputs: Process the inputs received, if any.

7. Update Display: Update the display based on the processed inputs or sensor data.

8. Check for Errors:

   - Communication Errors: Check for any errors in SPI communication.

   - Display Errors: Check for any errors related to the TFT display.

9. Error Handling:

   - Retry Mechanism: Implement a retry mechanism for failed SPI transactions.

   - Error Messages: Display error messages on the TFT display if any errors occur.

10. End: End of the flowchart.

3.3.3 Assumptions

- The SPI protocol is suitable for the data transmission requirements of the project.
- The microcontroller and the TFT display are operating within their specified voltage and temperature ranges.
- The software running on the microcontroller is reliable and free of bugs that could affect the data transmission or display.
- The hardware connections between the microcontroller and the TFT display are secure and reliable.
- The TFT display has a suitable refresh rate for the intended application.

3.3.4 Constraints

- The microcontroller has a limited number of SPI ports, which may limit the number of devices it can communicate with simultaneously.
- The physical size and pin configuration of the microcontroller and the TFT display may impose constraints on the design of the hardware setup.
- The power consumption of the microcontroller and the TFT display could be a concern, especially in battery-powered applications.
- The microcontroller may have limited processing power and memory, which could affect the complexity of the software that can be run on it.
- The viewing angle and brightness of the TFT display may be limited, which could affect the visibility of the display under certain conditions.

3.4 COMPONENTS AND TOOL USAGE

To develop an SPI-based TFT Display Controller using STM32CubeIDE software, you'll need the following components and tools:

1. STM32 microcontroller: Choose an STM32 microcontroller that supports SPI communication and has sufficient GPIO pins to connect to the TFT display. Examples include STM32F4, STM32F7, or STM32H7 series.

2. TFT Display: Select a TFT display module that supports SPI communication and has a suitable resolution for your application. Make sure it comes with a compatible driver chip.

3. Breadboard or PCB: Depending on your project requirements, you can either prototype the circuit on a breadboard or design a custom PCB for a more permanent solution.

4. Jumper wires: Use jumper wires to connect the STM32 microcontroller to the TFT display module and other peripheral components.

5. Power supply: Provide a stable power supply to the STM32 microcontroller and TFT display module.

6. STM32CubeIDE software: Download and install STM32CubeIDE, which is an integrated development environment for STM32 microcontrollers. It includes tools for code development, debugging, and firmware deployment.

7. STM32CubeMX: This is a graphical tool that allows you to configure STM32 microcontroller peripherals and generate initialization code. Use STM32CubeMX to configure the SPI peripheral and GPIO pins for communication with the TFT display.

8. TFT display library: Depending on the specific TFT display module you're using, you may need a display library or driver code to interface with the display. Some displays come with pre-written libraries, while others may require you to write your own driver code.

9. Code editor: Use the built-in code editor in STM32CubeIDE to write, debug, and compile your firmware code. You can write code in C or C++ using the STM32CubeIDE editor.

10. Debugger: STM32CubeIDE includes a built-in debugger that allows you to debug your firmware code and monitor variables, registers, and memory contents in real-time.

By leveraging these components and tools, you can efficiently develop an SPI-based TFT Display Controller using STM32CubeIDE software.

3.5 INDIVIDUAL AND TEAM CONTRIBUTION

Sl No.	Member Name	Work Done
1.	Janya L S	Information Collection
2.	Kavya K S	PPT(Power Point Presentation)
3.	Kaushik C	Project Report
4.	Manu H P	Programming using software

TABLE 1: INDIVIDUAL AND TEAM CONTRIBUTION

3.6 PROJECT PLANNING AND EXECUTION

Date	Work Done
23-03-2024	Information gatherings, Detail study of Problem Statement
24-03-2024	Software Installation,Programming using ST32Cube IDE
25-03-2024	PPT, Project Report

TABLE 2 : PROJECT PLANNING AND EXECUTION

CHAPTER 4

OUTCOMES

4.1 TECHNICAL OUTCOMES

1. Technical Contributions:

- Contributed to the development of firmware for the SPI-based TFT Display Controller using STM32CubeIDE software.

- Designed and configured the SPI communication protocol using STM32CubeMX tool.

- Implemented efficient algorithms and optimized memory usage to enhance the performance and responsiveness of the display controller.

- Participated in hardware testing and debugging sessions to identify and resolve issues promptly.

2. Achievements and Results:

- Successfully optimized firmware code to improve the efficiency of the display controller, resulting in a smoother user experience.

- Actively collaborated with senior engineers to meet project milestones and deliverables, ensuring the timely completion of tasks.

- Played a key role in troubleshooting and resolving technical challenges, contributing to the successful implementation of the display controller.

4.2 NON-TECHNICAL OUTCOMES:

1. Verbal and Written Communication:

- Enhanced verbal communication skills through regular team meetings and presentations, effectively conveying technical concepts to team members and stakeholders.

- Improved written communication skills by documenting project requirements, design specifications, and test results with clarity and precision.

2. Personality Development:

- Developed a proactive and solution-oriented approach to problem-solving, demonstrating resilience and adaptability in challenging situations.

- Cultivated a collaborative and supportive team spirit, fostering positive working relationships with colleagues and peers.

3. Time Management:

- Learned to prioritize tasks effectively and manage time efficiently to meet project deadlines and objectives.

- Developed the ability to balance multiple responsibilities and allocate resources judiciously to maximize productivity and outcomes.

4. Resource Utilization Skills:

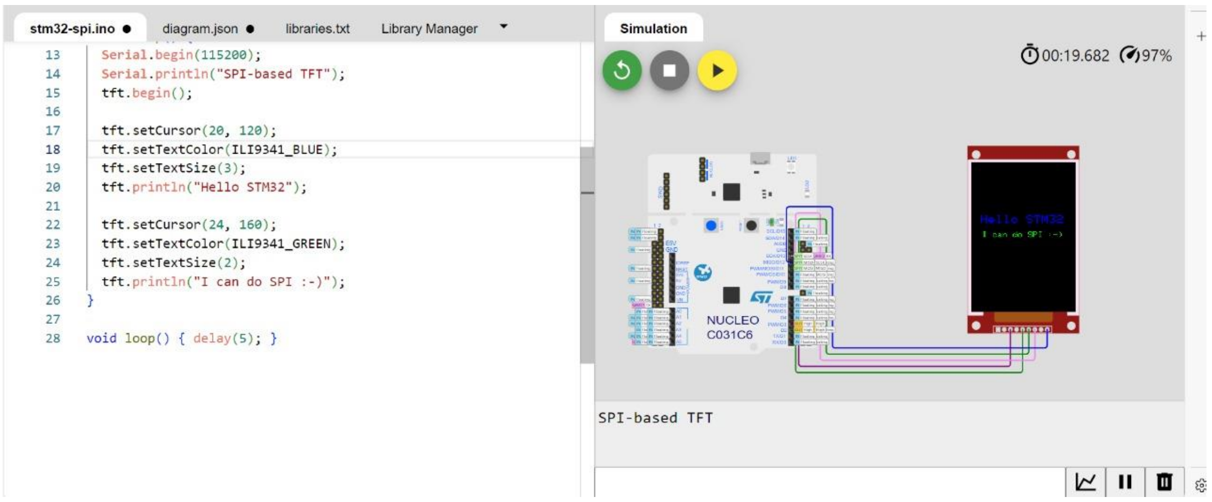
- Acquired proficiency in utilizing software tools such as STM32CubeMX and STM32CubeIDE for embedded systems development, optimizing resource utilization and enhancing project efficiency.

- Leveraged available resources effectively to overcome technical obstacles and achieve project goals, demonstrating resourcefulness and innovation.

Overall, the internship experience on the SPI-based TFT Display Controller project using STM32CubeIDE software provided valuable technical insights and personal growth opportunities, equipping me with essential skills and competencies for future endeavors in embedded systems development.

CHAPTER 5

RESULT



```
  Serial.begin(115200);
  Serial.println("SPI-based TFT");
  tft.begin();

  tft.setCursor(20, 120);
  tft.setTextColor(ILI9341_BLUE);
  tft.setTextSize(3);
  tft.println("Hello STM32");

  tft.setCursor(24, 160);
  tft.setTextColor(ILI9341_GREEN);
  tft.setTextSize(2);
  tft.println("I can do SPI :-)");
}

void loop() { delay(5); }
```

```
SPI-based TFT
```

FIGURE 3.5

CHAPTER 6

REFERENCE AND APPENDIX

6.1 REFERENCE

1. https://controllerstech.com/st7735-1-8-tft-display-with-stm32/

2. https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf

3. https://www.st.com/resource/en/reference_manual/dm00119316-stm32h742-stm32h743753-stm32h750stm32h753-advanced-armbased-32bit-mcus-stmicroelectronics.pdf

4. https://learn.adafruit.com/adafruit-2-8-tft-touch-shield-v2/overview

6.2 APPENDIX

Code used in STM32Cube IDE:

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
***
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
***
  * @attention
  *
  * Copyright (c) 2024 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
```

```
******************************************************************************
***
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "ST7735.h"
#include "GFX_FUNCTIONS.h"
/* USER CODE END Includes */
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables ---------------------------------------------------------*/
SPI_HandleTypeDef hspi1;
/* USER CODE BEGIN PV */
/* USER CODE END PV */
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
/* USER CODE END 0 */
```

```
int main(void)
{
  HAL_Init();
  /* USER CODE BEGIN Init */
  /* USER CODE END Init */
  /* Configure the system clock */
  SystemClock_Config();
  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */
  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  /* USER CODE BEGIN 2 */
  ST7735_Init(0);
  fillScreen(BLACK);
  testAll();
  /* USER CODE END 2 */
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
            ST7735_SetRotation(0);
            ST7735_WriteString(0, 0, "HELLO", Font_11x18, RED, BLACK);
            HAL_Delay(1000);
            fillScreen(BLACK);
            ST7735_SetRotation(1);
            ST7735_WriteString(0, 0, "WORLD", Font_11x18, GREEN, BLACK);
            HAL_Delay(1000);
            fillScreen (BLACK);
            ST7735_SetRotation(2);
            ST7735_WriteString(0, 0, "FROM", Font_11x18, BLUE, BLACK);
            HAL_Delay(1000);
            fillScreen(BLACK);
            ST7735_SetRotation(3);
```

```
          ST7735_WriteString(0, 0, "ControllersTech", Font_16x26, YELLOW, BLACK);
          HAL_Delay(1000);
          fillScreen(BLACK);
  }
  /* USER CODE END 3 */
}
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE
3);
  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 4;
  RCC_OscInitStruct.PLL.PLLN = 50;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 2;
  RCC_OscInitStruct.PLL.PLLR = 2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
```

```
                         |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
}
static void MX_SPI1_Init(void)
{
  /* USER CODE BEGIN SPI1_Init 0 */
  /* USER CODE END SPI1_Init 0 */
  /* USER CODE BEGIN SPI1_Init 1 */
  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_1LINE;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_32;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */
  /* USER CODE END SPI1_Init 2 */
}
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_9, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
  /*Configure GPIO pin : PC7 */
  GPIO_InitStruct.Pin = GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  /*Configure GPIO pin : PA9 */
  GPIO_InitStruct.Pin = GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
  /*Configure GPIO pin : PB6 */
  GPIO_InitStruct.Pin = GPIO_PIN_6;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

```
/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}
/* USER CODE BEGIN 4 */
/* USER CODE END 4 */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```