General purpose data acquisition system using the STM32 microcontroller and FreeRTOS

LM118 – Bachelor of Engineering in Electronic and Computer Engineering

Final Year Project

Project Report

Qinyuan Liu
20137095

Dr Ian Grout
24/Mar/205

Abstract

The rapid advancement of embedded systems has significantly increased the demand for efficient, scalable, and real-time data acquisition solutions, particularly in industrial automation, robotics, and control systems. This project focuses on developing a general-purpose data acquisition system using the STM32 microcontroller and FreeRTOS, integrating 16×16 analog I/O and 16×16 digital I/O with synchronized data transfer.

The hardware design includes PCB layout, component selection, and power optimization, featuring MCP3008 ADCs, MCP4822 DACs, and digital I/O expansion via 74HC165/74HC595 shift registers, all communicating through SPI and GPIO multiplexing techniques.

On the software side, the system is designed with RTOS-based task scheduling, synchronized communication, and real-time data acquisition.

Future work will concentrate on firmware implementation and system optimization, addressing key challenges such as reducing ADC sampling jitter, optimizing task prioritization, and refining serial data transfer protocols. The final system aims to deliver real-time monitoring and control capabilities, applicable to battery management, industrial vibration analysis, and automation systems.

Declaration

This interim report is presented in part fulfilment of the requirements for the LM118 Bachelor of Engineering in Electronic and Computer Engineering **Bachelors Project**.

It is entirely my own work and has not been submitted to any other University or Higher Education Institution or for any other academic award within the University of Limerick.

Where there has been made use of work of other people it has been fully acknowledged and referenced.

Name	**Qinyuan Liu**
Signature	
Date	28/ OCT/ 2024

Table of Contents

Table of Contents

List of Figures and Drawings

List of Figures:

List of Drawings:

1. **General Data Acquisition System Block Diagram (Appendix D drawing 1)**

   - Title: General Data Acquisition System Block Diagram
   - Author: Qinyuan Liu

2. **RTOS Task Priority Diagram (Appendix D drawing 2)**

   - Title: RTOS Task Priority Diagram
   - Author: Qinyuan Liu

3. **ADC Module Schematic (Appendix D drawing 3)**

   - Title: ADC_Module
   - Author: Qinyuan Liu

4. **DAC Module Schematic (Appendix D drawing 4)**

- Title: DAC_and_CS_Module
- Author: Qinyuan Liu

5. Digital Input Module Schematic (Appendix D drawing 5)

- Title: Digital_Input_Module
- Author: Qinyuan Liu

6. Digital Output Module Schematic (Appendix D drawing 6)

- Title: Digital_Output_Module
- Author: Qinyuan Liu

List of Equations

This project has not used any mathematical Equations at this stage.

Sample equation:

Equation 1 Sample Equation - Fourier Transform

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt$$

List of Tables

N/A

Chapter 1 Introduction

RTOS (Real-time Operating System) has a wide range of critical applications in embedded system development, from robot control in Mars traction to real-time data scheduling in digital signal processing (DSP). Its core value provides an operating platform that can accurately control the execution time of tasks. By introducing mechanisms such as task priority, time slice scheduling and interrupt management, RTOS can enable traditional ARM AVR architecture chips to complete good and precise control of multiple tasks under resource budget. Its biggest advantage is the strict guarantee of **time determinism (time determinism)**.

This project aims to build a general data acquisition system based on the STM32 microcontroller platform. RTOS is a substitute in the system architecture. It not only provides time-controlled scheduling support for various tasks, but also realizes the scheduling and control of multi-channel ADC, DAC and digital I/O modules through priority division and task synchronization mechanism; at the same time, the synchronous data interaction realized by connecting the serial communication module with the PC lays the foundation for the real-time data transmission and remote control of the system.

A low-speed, general-purpose data acquisition system prototype platform. The enhanced structure designed in this project has strong equipment and high standardization. It has the potential for practical expansion and deployment in many fields such as industrial automation, civil engineering, mechanical equipment status monitoring, vehicle electronic control, and safety protection systems. Through the flexible call of the RTOS architecture, the system can quickly equip different data acquisition needs and build a bridge between teaching and industrial applications.

For example, in the battery management system (BMS) scenario, the system can be used to realize the status monitoring and adjustment of multiple batteries. ADC colour can be used to collect the real-time voltage of each battery cell, and DAC can output precise control voltage to adjust the charging circuit; digital I/O is used for simple module status display and error indication; serial port communication is good with the host computer software to realize data transmission, exception handling and status algorithm decision-making. Each module runs well to form a good local control system with maintainability and scalability.

For example, in industrial vibration detection applications, rotating mechanical equipment such as generators and thermal power steam turbines are in high-load operation all year round, and stable and continuous vibration monitoring is extremely required. The operating frequency of such devices is usually lower than 1kHz, which matches the sampling frequency of this system. The ADC module in the project can collect large-scale signal vibrations in real time, transmit them to the PLC control system through SPI bus exchange or serial communication, and realize high-frequency sampling, data analysis and alarm response of system status in combination with RTOS.

System implementation phase division and development process description

In order to ensure that the project can achieve the maximum degree of functional implementation and prototype verification within limited resources and time, the development process of this system was divided into three progressive stages at the beginning of the project, namely

Breadboard prototype verification stage (Proof-of-Concept Prototype)

NUCLEO interface PCB prototype development stage (Prototype PCB)

System-on-Chip Integration Stage (Chip-on-Board, CoB Design)

Each stage revolves around the main line of "functional suction verification-structural stability realization-exploration of engineering productization possibilities", and is dynamically adjusted according to actual conditions during the development process. The following is a detailed description of each stage:

Phase 1: Proof-of-Concept

The focus of novice development is on the minimum viable system (MVS) of the system's fast logic and functions. The hardware circuit is built by breadboard to verify the following:

Functional and serial correctness of digital I/O control timing: Test whether the drive logic, signal synchronization and SPI clock edge control of 74HC595 (parallel to serial) and 74HC165 (parallel to serial) shift registers meet the design requirements;

Communication sound absorption verification between STM32 and peripherals: Use the SPI module on the NUCLEO-L476RG board to drive the mainstream control signal lines (SPI_CLK, MOSI, MISO, LATCH, CLR, OE), and test the GPIO behavior through the HAL layer configuration;

RTOS basic function access and task response verification: Focus on testing task scheduling, SysTick timer configuration, basic task switching mechanism and delay function use, and evaluate the aperture of future multi-task concurrent operation.

Although this stage provides a clear direction for the concept verification of the system, the breadboard has serious limitations in terms of connection reliability, electromagnetic interference control, layout space, etc., especially in the 16x16 digital channel wiring, signal interference, cold soldering, crosstalk and other problems often occur. Therefore, the project quickly entered the second stage and constructed the circuit into a PCB prototype board.

Phase 2: NUCLEO interface PCB prototype development (Prototype PCB)

This stage is the core achievement stage of this project, and the actual production and testing tasks are mainly concentrated on the dock.

The goal is to complete the interface of the system from experimental circuit to standardized engineering design based on the full verification of the logic of the first stage. The goals to be achieved include:

Standardized interface and wiring: Standardize all SPI interfaces and digital I/O control lines into 2.54mm interfaces, clearly distinguish between main control signals and module signals, and ensure the stability and reusability of debugging;

Power/ground layout optimization: Configure bypass capacitors and current limiting resistors to enhance the circuit's anti-interference ability and prevent power-on transient breakdown of peripheral chips;

Priority design principle: Arrange digital input and output modules into disassembled functional areas to enhance the maintainability of system upgrades;

SPI Daisy Chain expansion verification: Expand 74HC595 and 165 mechanisms through daisy chain mode to achieve pin multiplexing and reduce the control pins required for STM32;

Digital input and output modules are fully implemented and tested successfully: This stage successfully implemented and tested digital input (DI) and output (DO) modules, the system runs stably, the communication is accurate, and the logic is correct, providing reliable support for the subsequent expansion and solidification of analog (ADC/DAC) modules.

The completion of this phase has completed the main technical achievements of the deliverable and demonstrable projects in the FYP phase, meeting the basic engineering requirements of the "design-implementation-test" closed loop.

Phase 3: Chip-on-Board (CoB Design)

The long-term concept of the project involved in this phase is a natural extension of the further "productization" of the system. This phase aims to:

Strip the development board: directly solder the STM32 core chip to the mainboard to achieve complete configuration of peripheral logic such as independent power supply, crystal oscillator, reset, and debugging;

Full system integration: all modules and power supply systems, power management, USB communication interfaces, debug ports, etc. are laid out on the same PCB;

Industrial layout considerations: assemble DIN-Rail mounting holes, standard power/signal plug positions, chassis installation and batch deployment foundation when laying out the board.

Due to the serious time constraints of the FYP project and the system, the current stage is only at the system concept and circuit design pre-research, and has not yet been installed. If there is an opportunity to extend it to an advanced project or cooperate with enterprises in the future, the potential for engineering implementation will be realized.

The source of the entire project can be found at:

https://github.com/Qinyuan72/FYP/tree/master

[1]

Figure 24. NUCLEO-L476RG

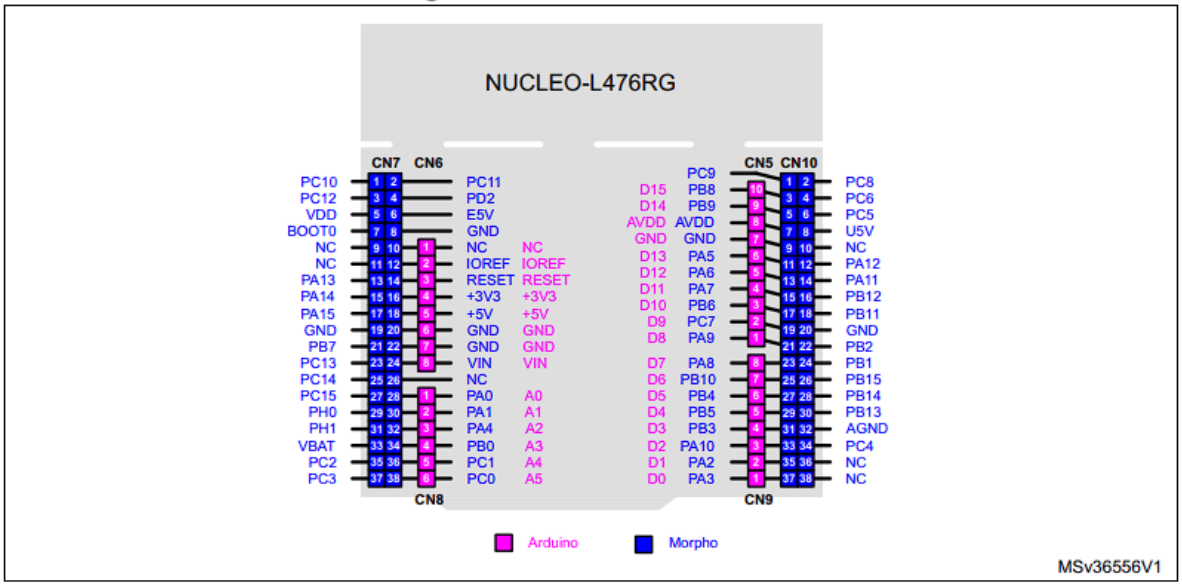


Figure 1 NUCLEO-L476RG pin layout

Chapter 2 Background

Due to limited processing power, early embedded systems usually use bare-metal programming to achieve task control. However, with the increase in application complexity, such as multi-channel sensor acquisition, network communication, interactive control, etc., bare-metal systems gradually cannot meet development needs. At this time, RTOS came into being. It provides a good software structure and runtime guarantee for embedded systems by providing task scheduler, interrupt management, inter-task communication mechanism (such as message queue, semaphore), timer control, resource mutual exclusion and synchronization mechanism.

In the late 1980s, with the increase in microprocessor processing power, military industry, aviation and other industries with extremely high requirements for real-time performance took the lead in promoting the development of RTOS. Among them, RTEMS (Real-Time Executive for Missile Systems), as one of the early mature RTOS, was originally funded and developed by the US Department of Defense and was widely used in missile systems and avionics equipment[6]. Its early version even supported the programming language promoted by the US military at that time, Ada language, which was a very advanced and representative embedded real-time system platform at that time [7]. Today, RTEMS has evolved into an open source RTOS for multi-processor systems and is still widely used in NASA and ESA's space missions, showing its advantages in stability and long-term maintainability.

In the 1990s, with the rise of the Internet and the popularization of 32-bit processors (such as ARM, PowerPC, MIPS, etc.), the complexity of embedded systems has further increased. A large number of embedded application scenarios for network, communication, audio and video processing have put forward higher requirements for RTOS. At this time, commercial RTOS platforms such as VxWorks and QNX have risen rapidly. VxWorks is an embedded RTOS developed by Wind River. With its excellent real-time performance and extensive industrial support, it has been widely used in aerospace, industrial control, medical equipment and other fields. For example, the main control systems of the US "Mars Exploration Rovers" Curiosity and InSight both use VxWorks as the underlying system, which is enough to show its influence in the field of high reliability. QNX, with its microkernel architecture and modular process communication design, is widely used in in-vehicle infotainment systems (IVI) and

autonomous driving assistance systems (ADAS), becoming an important part of the software architecture of modern smart cars. [6]

Application scenarios and advantages of RTOS

Today, RTOS has been widely used in aerospace, defense industry, automotive electronics, industrial control, communication systems, consumer electronics and other fields. Its core advantages include [4]:

1. Time determinism: RTOS can ensure that tasks are completed within strict time limits, thereby meeting real-time requirements.
2. Multi-task scheduling capability: supports priority scheduling mechanism, suitable for collaborative execution of complex tasks.
3. Interrupt and event response mechanism: has the ability to respond quickly to external interrupts to ensure the system's handling of emergencies.
4. Resource management and synchronization mechanism: provides means such as semaphores, mutexes, event groups, etc. to ensure resource sharing security in a multi-task environment.
5. Customizability and portability: adapts to a variety of processor architectures, has good modularity and customizability, and is suitable for resource-constrained devices.

Project Background and RTOS Selection

This project aims to design a general-purpose data acquisition system based on STM32. Its core task is to collect and control external analog and digital signals at a high speed at a scanning frequency of 1kHz. The system hardware structure includes 16 ADC analog inputs, 16 DAC analog outputs, 16x16 digital I/O interfaces, and data communication and command interaction are required through the USART serial port. Such a complex system task places high demands on the stability, real-time performance and multi-task scheduling capabilities of the system.

To this end, we chose to use the CMSIS-RTOS V2 real-time operating system interface provided in the STM32CubeIDE integrated development environment as the software basic platform for this project. CMSIS-RTOS V2 is a general RTOS API specification officially defined by ARM [8]. It is compatible with multiple mainstream RTOS (such as Keil RTX5, FreeRTOS, etc.) and is optimized for the ARM Cortex-M architecture. It has the characteristics of unified standards, good portability, and high IDE integration. STM32CubeIDE has natively integrated CMSIS-RTOS V2. Developers can directly create kernel objects such as threads,

timers, and semaphores through graphical configuration tools, greatly improving development efficiency and system maintainability.

Specific role of RTOS in this project

1. Solve the sampling "jitter" problem (Jitter)

In digital signal processing (DSP) and data acquisition, the consistency of sampling intervals is extremely critical. Traditional bare metal systems often cause sampling time instability due to interrupt interference or main loop blocking, resulting in "jitter" (Jitter), which in turn affects the spectrum analysis and control accuracy of the signal. RTOS can strictly control the periodicity of task operation through system timers and periodic thread scheduling mechanisms, thereby effectively alleviating the jitter problem. For example, in CMSIS-RTOS, using osTimer or osDelayUntil functions can achieve millisecond-level periodic task management, ensuring that ADC/DAC sampling tasks are always executed with a period of 1ms, improving the stability of system data processing.

2. Dealing with the complexity of system tasks

This project involves the coordination and scheduling of multiple concurrent tasks, such as:

- ADC acquisition and preprocessing tasks;
- DAC output signal adjustment tasks
- Digital IO scanning and response tasks
- Serial port command reception and analysis
- Upper computer communication data packaging and sending tasks;
- System status monitoring and exception handling tasks.

Such complex functional requirements will be extremely complex and difficult to maintain if implemented on a bare metal system. The multi-thread management mechanism provided by RTOS can decompose system functions into multiple independent tasks, each with different priorities. For example, the ADC acquisition task can be set as a high-priority real-time task, while the serial port reception processing can be set as a medium-priority task, and the system status monitoring can be set as a low-priority background task. Through the coordination of task priorities and scheduling strategies, the functions of various parts of the system can be coordinated and run in an orderly manner.

In addition, the inter-task communication mechanism of RTOS (such as queues, semaphores, and event flags) also greatly simplifies the data synchronization and control flow design between tasks. For example, we can transfer sampled data to the processing thread through the message queue in the ADC acquisition thread, and wake up the data parsing thread through the event flag in the serial port receiving thread, thereby avoiding complex global variables and polling logic.

3. Interrupt and resource mutual exclusion processing capabilities

In actual systems, ADC/DAC control, interrupt response, serial port reception, etc. often involve shared resources (such as buffers, control flags, etc.). The mutex lock and critical section protection mechanism provided by RTOS can ensure that multiple tasks or interrupt service programs will not compete or have data confusion when accessing shared resources. In addition, the interrupt nesting and priority management mechanism supported by RTOS also enables the system to respond to multi-source interrupts with higher efficiency.

Experimental platform and development environment

This project uses the STM32-Nucleo development board as the experimental platform. The main control chip adopts the ARM Cortex-M4 architecture, integrates high-precision ADC, rich I/O interfaces and serial communication modules, which is very suitable for the functional requirements of this project. The development environment uses STM32CubeIDE, which integrates code editor, compiler, debugger, graphical peripheral configuration tool (STM32CubeMX), RTOS middleware support, etc., providing a complete tool chain for embedded project development.

In addition, in order to cultivate students' ability in hardware system design, the project will also try to introduce PCB design and proofing process. We will use Target 3001 electronic drawing tool for schematic design and PCB wiring, and finally output standard Gerber files and complete small batch PCB production. This process not only helps students understand the complete process from system design to physical implementation, but also lays a good foundation for subsequent productization.

As an important supporting platform for modern embedded systems, real-time operating system has evolved from the initial task scheduler to a multi-functional integrated platform

including network communication, security mechanism, graphical interface, etc. In this project, by reasonably selecting and configuring CMSIS-RTOS V2, efficient task management, system scheduling and data acquisition functions are realized. The introduction of RTOS not only improves the reliability and scalability of the system, but also provides students with a real case platform for understanding industrial-grade embedded system development.

In the future, as the project scale expands and the control algorithm deepens, we can further explore the advanced features of RTOS, such as soft real-time and hard real-time scheduling strategies, multi-core processor support, security authentication interfaces, etc., and gradually build an embedded development capability system for complex control systems.

Chapter 3 Technical details - Software

STM32 Background and Hardware Configuration

Introduction to STM32 Cube:

The STM32Cube IDE (Cube IDE) is a complete development platform, Cube IDE offers intuitive hardware configuration in the 'ICO' interface and generates a skeleton code using the ICO configuration for the device. The Cube IDE used a C-based scripting language, offering HAL (hardware abstraction layers) to simplify hardware interaction by providing an abstraction layer between the application and microcontroller peripherals, which greatly reduced the complexity of the software design of this project

Introduction to STM32 Cube Interface:

Introduction to the STM32CubeMX configuration interface (taking STM32CubeIDE as an example)

STM32CubeMX is a graphical configuration tool provided by STMicroelectronics to simplify the initialization configuration process of the STM32 series microcontrollers. It is integrated in STM32CubeIDE and supports the completion of chip pin configuration, clock system settings, middleware enablement, and initialization code generation in a graphical manner. The following are the basic components and instructions for use of the interface:

1) **Main view structure (center)**
    a) The central area of the screenshot shows the chip package view (Package View), that is, the LQFP64 pin arrangement diagram of the STM32L476RGT6. Each pin can be assigned a function by clicking, such as configuring it as GPIO, USART, SPI, ADC, TIM, etc. Different colors represent different functional states, such as green represents enabled peripheral functions (such as PA2/PA3 for USART_TX/RX respectively), yellow represents multiplexing functions, and gray is unused.
2) **Functional classification menu on the left**

The functional classification menu on the left covers:

a) System Core: system core configuration, such as clock (RCC), interrupt controller (NVIC), power management (PWR)
b) Analog: analog functions, such as ADC, DAC;
c) Timers: timer configuration, such as basic timer, PWM output;
d) Connectivity: communication interface, such as USART, I2C, SPI;
e) Middleware and Software Packs: such as USB, middleware support, FreeRTOS, etc.

3) **Top navigation bar**

a) The four tabs at the top, "Pinout & Configuration", "Clock Configuration", "Project Manager", and "Tools", correspond to:

b) Pinout & Configuration: main pin configuration interface;

c) Clock Configuration: graphical configuration interface of the system clock tree;

d) Project Manager: project path, code structure, tool chain selection and other settings;

e) Tools: advanced tool and plug-in management.

4) **Function area on the right (hidden and expandable)**

a) Click "System View" in the upper right corner to switch to the system function diagram; you can also manually zoom in, zoom out or drag the chip diagram for easy operation.

5) **Project structure manager (upper left corner)**

a) "Project Explorer" displays all projects and their source file structures in the current workspace. You can quickly open the .ioc configuration file or the main code entry main.c for programming and debugging.

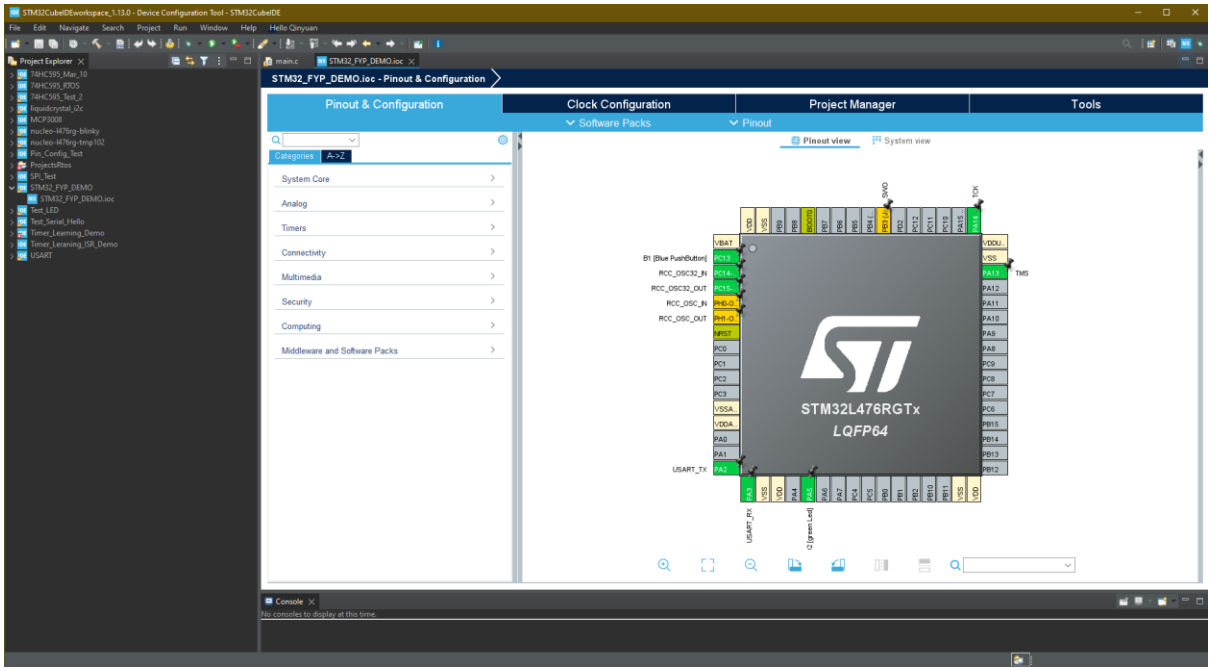


Figure 2 Cude IDE Layout

The following is a detailed introduction of how to configure the Cude IDE environment:

Download STM32CubeIDE from the ST website and have the ST-Link hardware programmer launcher/debugger. The downloaded software installation package will automatically install and update the required instruments for you.

https://www.st.com/en/development-tools/stm32cubeide.html

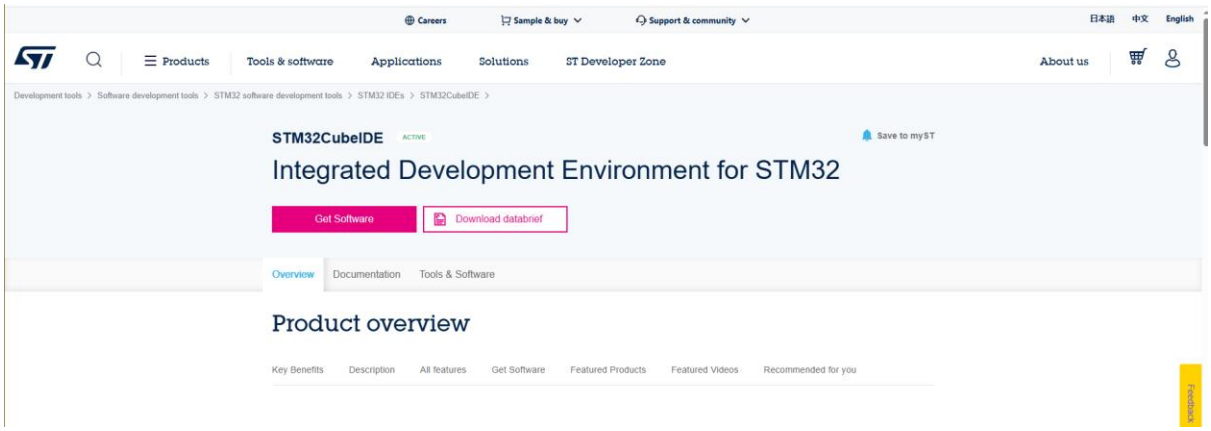


Figure 3 STM32 Cude IDE Download Page

On the left select "Start new STM32 project". You will see a target selector for choosing your STM32 chip or development board. Click "Next" to continue.

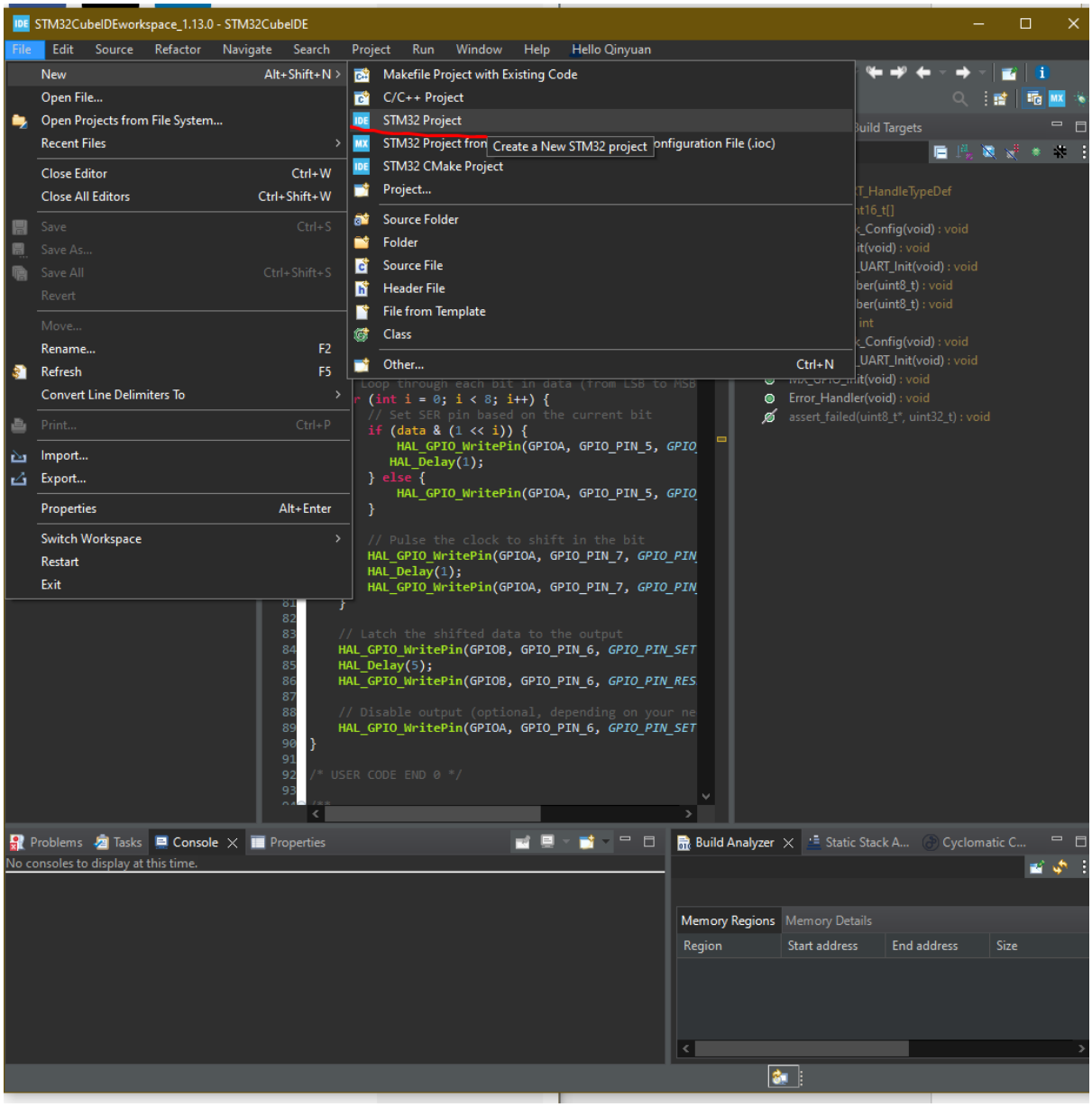


Figure 4 Cude IDE Start Config 1

In the interface that pops up automatically, switch to the Board Select Tab. Find the search bar on the left and search for Necleo-L47RG. You will find the corresponding development board in the Board-list on the right. Click this development board and click Next in the lower right

corner.

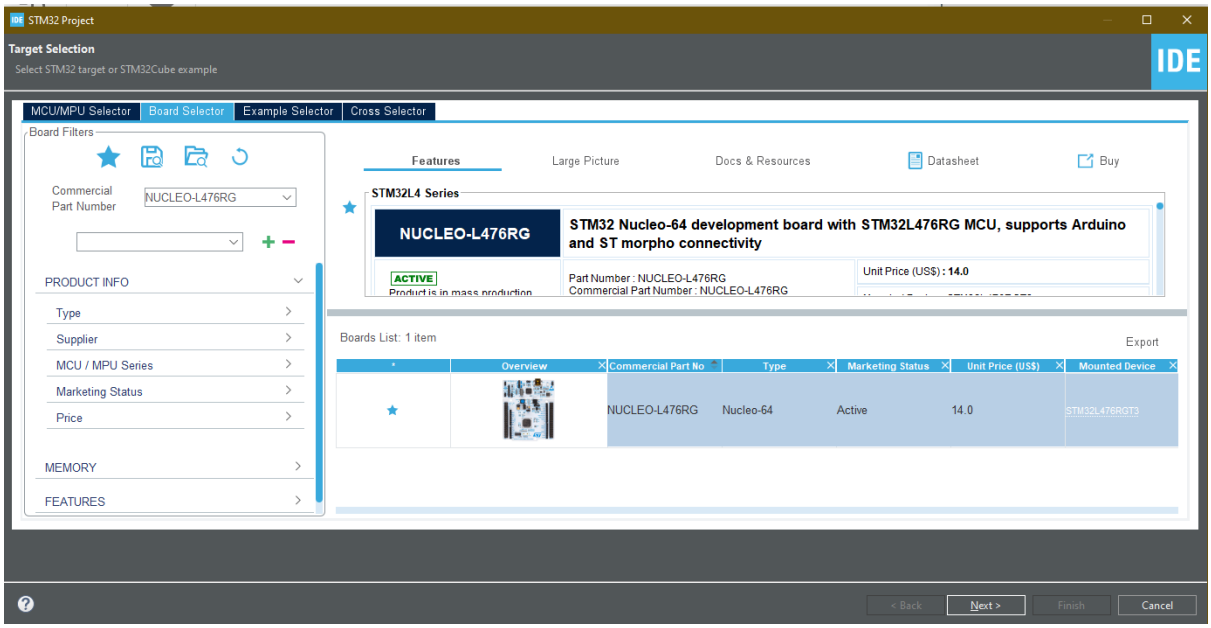


Figure 5 Stm32 Development Board Selection Menu

You'll be asked to input a project name next. Choose the default parameters and then click Finish.

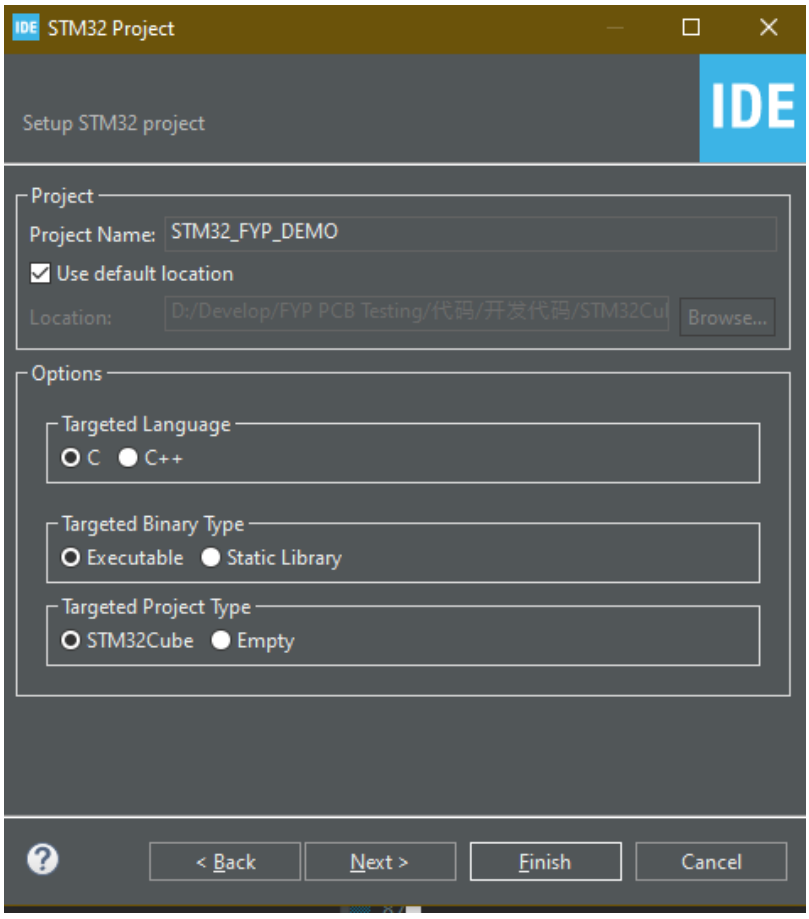


Figure 6 Stm32 Project configeration tab

The process continues to the STM32 CubeMX graphical interface. Here you can easily assign pins and various functions to the chip.

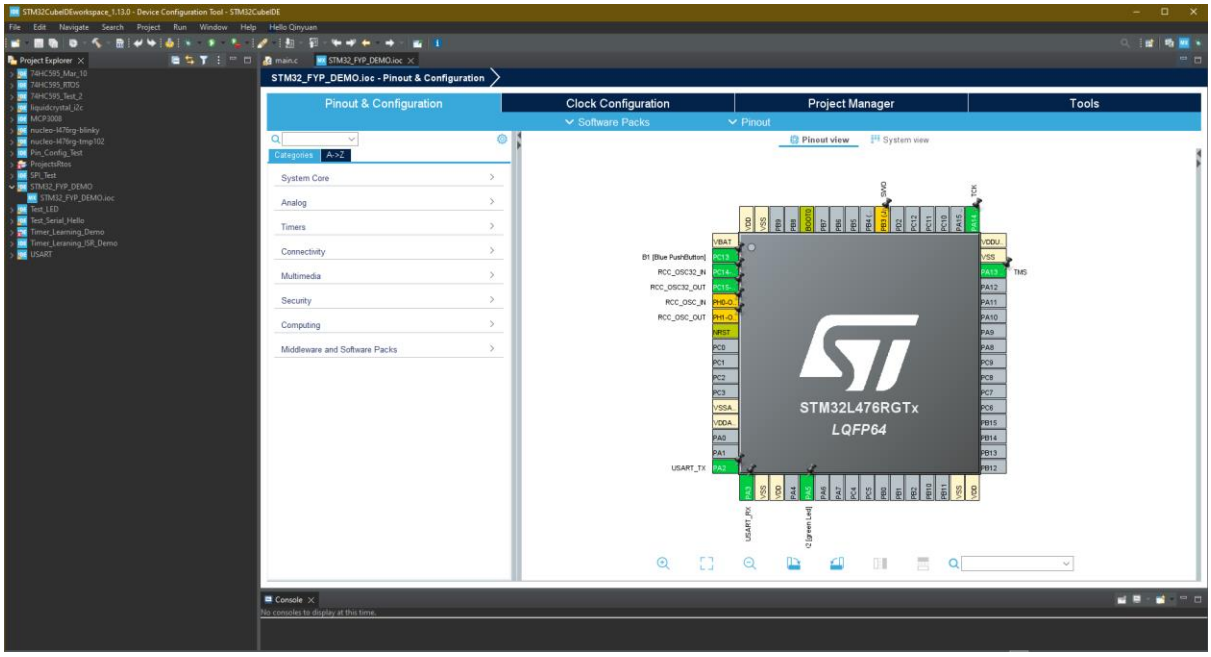


Figure 7 Cude IDE finish configuration

STM32 and RTOS Integration:

In this project, we used the integrated development environment provided by STM32CubeIDE, which has the CMSIS-RTOS V2 real-time operating system interface embedded. This RTOS is a standardized API specification officially launched by ARM, designed for processors based on the Cortex-M architecture, with good portability and IDE compatibility. STM32CubeIDE has very complete support for this interface, allowing developers to directly create RTOS tasks, timers, semaphores and other kernel objects through graphical configuration tools, and automatically generate framework code.

In this project, the task priority configuration of RTOS adopts a two-tier priority system, which divides tasks into high priority and ordinary priority, to achieve fast scheduling and resource preemption of critical tasks (such as data acquisition and communication interrupt response), and at the same time, low priority operation of non-critical tasks (such as status monitoring and LED display) to ensure the overall responsiveness and real-time performance of the system.

From a grammatical perspective, the API design of CMSIS-RTOS V2 is relatively intuitive and easy to use. Through the RTOS skeleton code automatically generated by STM32CubeIDE, developers can insert user task definitions and create functions in the /* USER CODE BEGIN

*/ area of the main.c file. The syntax structure is similar to the HAL library interrupt function (ISR) style, which is easy to understand and maintain. For example, developers can use the osThreadNew() function to create a task and use the task function and parameters such as stack size and priority as input to start a user-defined thread.

The code snippet below shows a simple RTOS task creation and LED flashing function implementation, where the StartDefaultTask task calls HAL_GPIO_TogglePin() at a fixed period to achieve LED flashing. In this example, developers only need to add task creation statements in the MX_FREERTOS_Init() function and implement specific logic in the task function body to achieve basic multi-tasking operations.

At the same time, this article also attaches the corresponding CubeMX configuration interface screenshot, showing the RTOS task allocation, system scheduling settings and pin mapping layout, so that readers can more intuitively understand the relationship between code and hardware configuration. Through this graphical configuration and code linkage mechanism, this project significantly improved the efficiency of RTOS system development, reduced the probability of errors, and laid a solid foundation for subsequent function expansion and system stability testing.

The following is a simple example of building a small flashing light using CMSIS-RTOS V2 for introduction purposes:

```
/* 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"
#include "cmsis_os.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
```

```
/* 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 ---------------------------------------------------------*/
/* Definitions for defaultTask */
osThreadId_t defaultTaskHandle;
const osThreadAttr_t defaultTask_attributes = {
  .name = "defaultTask",
  .stack_size = 128 * 4,
  .priority = (osPriority_t) osPriorityNormal,
};
/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
void StartDefaultTask(void *argument);

/* USER CODE BEGIN PFP */
void vTaskLed_0 (void *pvPrameters);

void vTaskLed_1 (void *pvPrameters);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
```

```
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();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Init scheduler */
osKernelInitialize();

/* USER CODE BEGIN RTOS_MUTEX */
/* add mutexes, ... */
/* USER CODE END RTOS_MUTEX */

/* USER CODE BEGIN RTOS_SEMAPHORES */
/* add semaphores, ... */
/* USER CODE END RTOS_SEMAPHORES */

/* USER CODE BEGIN RTOS_TIMERS */
/* start timers, add new ones, ... */
/* USER CODE END RTOS_TIMERS */

/* USER CODE BEGIN RTOS_QUEUES */
/* add queues, ... */
/* USER CODE END RTOS_QUEUES */

/* Create the thread(s) */
/* creation of defaultTask */
defaultTaskHandle = osThreadNew(StartDefaultTask, NULL, &defaultTask_attributes);

/* USER CODE BEGIN RTOS_THREADS */
/* add threads, ... */
//Create task to manipulate LED_0
 if((xTaskCreate(vTaskLed_0, "Task LED
0", configMINIMAL_STACK_SIZE, NULL, 1, NULL)) != pdTRUE)
 {

 }

 //Create task to manipulate LED_1
 if((xTaskCreate(vTaskLed_1, "Task LED
1", configMINIMAL_STACK_SIZE, NULL, 1, NULL)) != pdTRUE)
 {

 }
 /* USER CODE END RTOS_THREADS */

 /* USER CODE BEGIN RTOS_EVENTS */
```

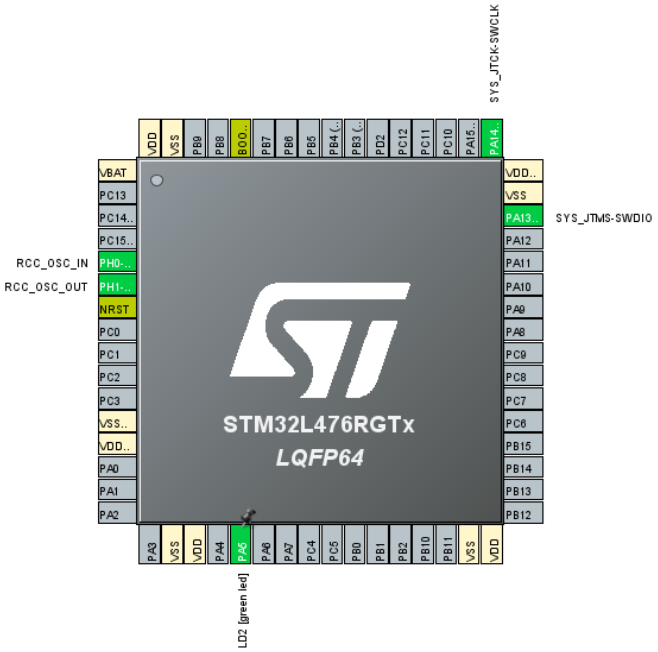
```
  /* add events, ... */
  /* USER CODE END RTOS_EVENTS */

  /* Start scheduler */
  osKernelStart();

  /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
  RCC_OscInitStruct.MSIState = RCC_MSI_ON;
  RCC_OscInitStruct.MSICalibrationValue = 0;
  RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
  RCC_OscInitStruct.PLL.PLLM = 1;
  RCC_OscInitStruct.PLL.PLLN = 40;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  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_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
/* USER CODE BEGIN MX_GPIO_Init_1 */
/* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : LD2_Pin */
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);

/* USER CODE BEGIN MX_GPIO_Init_2 */
/* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */
void vTaskLed_0 (void *pvParameters)
{
    //Variable Declaration

    //Infinite Loop
    for (;;)
    {
        HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_5);

        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
    //Delete this task if break out the loop
    vTaskDelete(NULL);
}
void vTaskLed_1 (void *pvParameters)
{
    //Variable Declaration

    //Infinite Loop
    for (;;)
    {
```

```
        //HAL_GPIO_TogglePin(User_LED_GPIO_Port, LED_1_Pin);

        //vTaskDelay(1000 / portTick_PERIODS_MS);
    }
    //Delete this task if break out the loop
    vTaskDelete(NULL);
}
/* USER CODE END 4 */

/* USER CODE BEGIN Header_StartDefaultTask */
/**
  * @brief  Function implementing the defaultTask thread.
  * @param  argument: Not used
  * @retval None
  */
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void *argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    osDelay(1);
  }
  /* USER CODE END 5 */
}

/**
  * @brief  Period elapsed callback in non blocking mode
  * @note   This function is called  when TIM6 interrupt took place, inside
  * HAL_TIM_IRQHandler(). It makes a direct call to HAL_IncTick() to increment
  * a global variable "uwTick" used as application time base.
  * @param  htim : TIM handle
  * @retval None
  */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  /* USER CODE BEGIN Callback 0 */

  /* USER CODE END Callback 0 */
  if (htim->Instance == TIM6) {
    HAL_IncTick();
  }
  /* USER CODE BEGIN Callback 1 */

  /* USER CODE END Callback 1 */
}

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
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
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
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,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Figure 8 ICO pin configuration of in the Cube IDE for the RTOS LED toggling demo



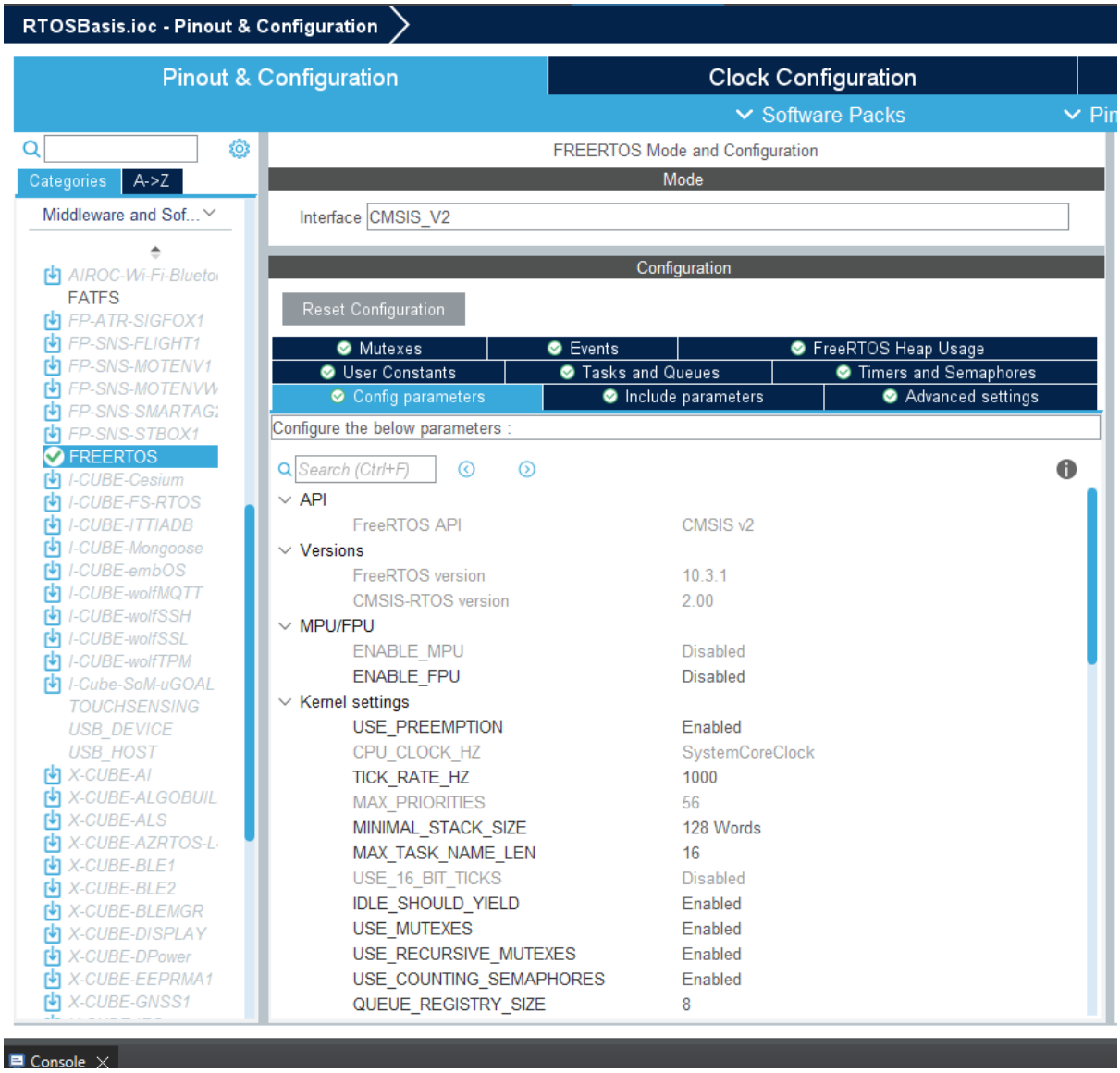


Figure 9 RTOS configuration in the Cube IDE

During the configuration of FreeRTOS, developers can enter the "Pinout & Configuration" page through the graphical .ioc project configuration file, and select "Middleware and Software Packs" > "FREERTOS" in the left navigation bar to enter the configuration interface. In this interface, the system automatically enables the CMSIS-RTOS V2 interface as the RTOS API layer selected for this project. CMSIS-RTOS V2 is a standardized interface officially launched by ARM. It has cross-platform, portability and strong integration, and is widely used in Cortex-M series processors

In the main configuration window, developers can set the FreeRTOS core parameters in detail. First, in the API and version information area, you can see that the current FreeRTOS version used is 10.3.1 and the CMSIS-RTOS interface version is 2.00. Then there are the options for

MPU/FPU support. In this project, due to limited system resources, MPU (memory protection unit) and FPU (floating point processor) related functions are not enabled.

The core configuration area is Kernel Settings, which contains a series of key parameters:

USE_PREEMPTION is enabled, which means that the system adopts a preemptive scheduling mechanism, which can automatically switch tasks according to task priorities to ensure timely response of high-priority tasks;

TICK_RATE_HZ is set to 1000, which means that the system beat is 1ms, which meets the design requirement of 1kHz ADC/DAC sampling rate in this project;

MAX_PRIORITIES is set to 56. Although this project actually adopts a dual priority structure (high priority is used for sampling and control, and low priority is used for communication and maintenance), retaining a higher configuration will help system expansion;

MINIMAL_STACK_SIZE is 128 Words, which is the minimum stack space allocated for each task, ensuring that each task has enough memory resources when running in multiple threads;

USE_MUTEXES, USE_RECURSIVE_MUTEXES and USE_COUNTING_SEMAPHORES to support mutual exclusion control and inter-task communication in a multi-tasking environment;

QUEUE_REGISTRY_SIZE is set to 8 to register up to 8 RTOS queue objects during debugging.

In the actual configuration process, developers will first set these core parameters, and then enter the upper submodules (such as Tasks and Queues, Timers and Semaphores, Events, etc.) to configure the task name, priority, stack size, scheduling cycle, etc. one by one, and finally STM32CubeIDE automatically generates initialization code (including freertos.c, freertos.h, task function framework, etc.).

Propused implementation 1: Task Priority in RTOS:

*The full **Task Priority** Diagram drawing is shown in Appendicitis C: 2. **RTOS Task Priority Diagram***

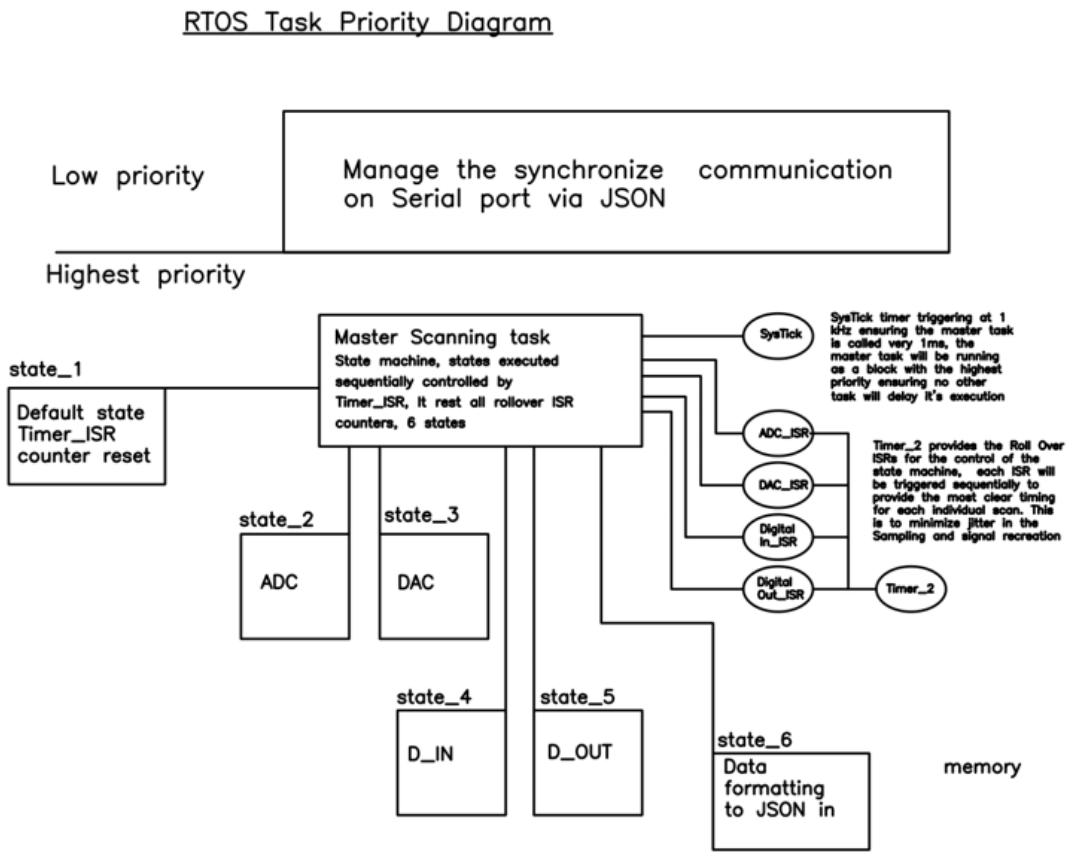


Figure 10 . RTOS Task Priority Diagram

A correct and well-evaluated Task priority setup is critical in an RTOS application, a miss-prioritized task can be critical in the success of a mission. For instance, the Mars lander that was introduced in the background section had the communication task settled in a way that would interfere with the data acquisition task, leaving the lander with a mission-critical problem that must be addressed remotely. [5]

In our application, a 2-level priority is defined: the highest priority Master Scanning task, and the low priority data management and Serial communication task.

Low-Priority Tasks

In the low-priority tasks, it'll have 2 functionalities, it will collect the JSON format from the Master Scanning task and verify its integrity if the data is not corrupted. The data will be sent to the Queue for the communication task, the communication will be managing the serial

communication with the Python code on the PC, providing information for the Python script for data visualization. As well as collecting data from the user input from the Python script and setting up the JSON data in its input/ configuration section for user input purposes. The control data from user input will be set to the **fixed memory location** for the Master scanning Task's direct access. Additionally, for a specific application like the battery management system, a local algorithm could be implemented on this level overriding user input, to achieve an 'Offline' close-loop control system.

High-Priority Tasks: Master Scanning Task

The high-priority task level ensures that the task running on this level will not be interrupted in execution, ensuring timing accuracy to the clock ticking level. Master Scanning Task is controlled by the SysTick timer in 1kHZ frequency, meaning it's invoked every 1 millisecond, The Master Scanning Task is essentially a **Finite State Machine (FSM)**, the state of the Master Scanning task FSM is controlled by an internal flag and 4 timers_ISR interrupts. When SysTick is invoked, the FSM will be set to the default state which will rest all the ISR_timer reset the scanning status flags and wait for the first ISR to trigger the FSM to move to the ADC_scanning state, the ADC_scanning state will scan all 2 ADC chips, and save the data into dedicated fixed memory locations. A setup with a similar structure to the ADC is configured for the rest of the chip scanning task. After state 5 (Digital_Out), state 6 data formatting will read the data from the memory location of the privies scanning task and format the data into a JSON string in the dedicated memory location for the low-priority task to access.

Purpose of Configuring the Master Scanning Task as an FSM controlled by ISR

The purpose of configuring the master task as a timer rollover ISR is to provide accurate timing for the Data acquisition system, especially in minimizing jitter in the analog IO, meanwhile provides scalability for explanation as the timer rollover can be easily extended to accommodate more data channels (additional chips).

Prototype RTOS implementation for 74HC595:

```
#include "main.h"
#include "cmsis_os.h"

/* Private variables ---------------------------------------------------------*/
```

```c
UART_HandleTypeDef huart2;

/* Define task handles */
osThreadId_t displayTaskHandle;

/* GPIO Pin Definitions for SN74HC595 */
#define SER_PIN     GPIO_PIN_5 // Serial Data Input
#define SRCLK_PIN   GPIO_PIN_7 // Shift Register Clock
#define RCLK_PIN    GPIO_PIN_6 // Storage Register Clock (Latch)
#define OE_PIN      GPIO_PIN_6 // Output Enable (active low)
#define SER_PORT    GPIOA
#define RCLK_PORT   GPIOB
#define OE_PORT     GPIOA

/* Array for displaying numbers 0 to 9 on the 7-segment display */
uint16_t number[10] = {
    0b00000011, 0b10011111, 0b00100101, 0b00001101,
    0b10011001, 0b01001001, 0b01000001, 0b00011011,
    0b00000001, 0b00001001
};

/* Function prototypes */
void SN74HC595_Init(void);
void SN74HC595_SendData(uint8_t data);
void vTaskSN74HC595(void *argument);
void SystemClock_Config(void);
void MX_USART2_UART_Init(void);
void MX_GPIO_Init(void);

/**
  * @brief  Main function to initialize RTOS and tasks.
  * @retval int
  */
int main(void)
{
    /* Initialize HAL library */
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_USART2_UART_Init();

    /* Initialize SN74HC595 */
    SN74HC595_Init();

    /* Initialize FreeRTOS */
    osKernelInitialize();

    /* Create the SN74HC595 display task */
    const osThreadAttr_t displayTask_attributes = {
        .name = "DisplayTask",
        .stack_size = 128 * 4,
        .priority = (osPriority_t) osPriorityNormal,
    };
    displayTaskHandle = osThreadNew(vTaskSN74HC595, NULL,
&displayTask_attributes);

    /* Start FreeRTOS scheduler */
    osKernelStart();
```

```c
    /* Infinite loop (we should never reach here) */
    while (1) {}

    return 0;
}

/**
  * @brief SN74HC595 initialization.
  */
void SN74HC595_Init(void)
{
    HAL_GPIO_WritePin(OE_PORT, OE_PIN, GPIO_PIN_SET); // Disable output initially
}

/**
  * @brief Send data to SN74HC595 shift register.
  * @param data: 8-bit data to be sent
  */
void SN74HC595_SendData(uint8_t data)
{
    // Enable output (active low)
    HAL_GPIO_WritePin(OE_PORT, OE_PIN, GPIO_PIN_RESET);

    // Shift each bit into the shift register
    for (int i = 0; i < 8; i++)
    {
        // Set SER pin based on the current bit
        HAL_GPIO_WritePin(SER_PORT, SER_PIN, (data & (1 << i)) ? GPIO_PIN_SET :
GPIO_PIN_RESET);

        // Pulse SRCLK to shift in the bit
        HAL_GPIO_WritePin(SER_PORT, SRCLK_PIN, GPIO_PIN_SET);
        HAL_GPIO_WritePin(SER_PORT, SRCLK_PIN, GPIO_PIN_RESET);
    }

    // Latch the data into the storage register
    HAL_GPIO_WritePin(RCLK_PORT, RCLK_PIN, GPIO_PIN_SET);
    HAL_GPIO_WritePin(RCLK_PORT, RCLK_PIN, GPIO_PIN_RESET);

    // Disable output if required
    HAL_GPIO_WritePin(OE_PORT, OE_PIN, GPIO_PIN_SET);
}

/**
  * @brief  Task to manage the SN74HC595 shift register display.
  * @param  argument: Not used
  */
void vTaskSN74HC595(void *argument)
{
    uint8_t cnt = 0;

    /* Infinite loop to update display */
    for (;;)
    {
        /* Send current count value to 74HC595 */
        SN74HC595_SendData(number[cnt]);

        /* Increment counter and wrap around after 9 */
        cnt = (cnt + 1) % 10;
```

```c
        /* Delay for a while (replace HAL_Delay with FreeRTOS delay) */
        vTaskDelay(pdMS_TO_TICKS(500));
    }
}

/**
  * @brief  Configure the system clock.
  */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Configure the main internal regulator output voltage
    */
    if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the RCC Oscillators according to the specified parameters
    * in the RCC_OscInitTypeDef structure.
    */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_MSI;
    RCC_OscInitStruct.MSIState = RCC_MSI_ON;
    RCC_OscInitStruct.MSICalibrationValue = 0;
    RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
    RCC_OscInitStruct.PLL.PLLM = 1;
    RCC_OscInitStruct.PLL.PLLN = 40;
    RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
    RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
    RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }

    /** Initializes the CPU, AHB and APB buses clocks
    */
    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_DIV1;
    RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

    if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
    {
        Error_Handler();
    }
}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
```

```c
  */
void MX_USART2_UART_Init(void)
{
    huart2.Instance = USART2;
    huart2.Init.BaudRate = 115200;
    huart2.Init.WordLength = UART_WORDLENGTH_8B;
    huart2.Init.StopBits = UART_STOPBITS_1;
    huart2.Init.Parity = UART_PARITY_NONE;
    huart2.Init.Mode = UART_MODE_TX_RX;
    huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart2.Init.OverSampling = UART_OVERSAMPLING_16;
    huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
    huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
    if (HAL_UART_Init(&huart2) != HAL_OK)
    {
        Error_Handler();
    }
}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();

    /*Configure GPIO pins : SER_PIN, SRCLK_PIN, OE_PIN */
    GPIO_InitStruct.Pin = SER_PIN | SRCLK_PIN | OE_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(SER_PORT, &GPIO_InitStruct);

    /*Configure GPIO pin : RCLK_PIN */
    GPIO_InitStruct.Pin = RCLK_PIN;
    HAL_GPIO_Init(RCLK_PORT, &GPIO_InitStruct);
}

/**
  * @brief Error Handler
  */
void Error_Handler(void)
{
    __disable_irq();
    while (1) {}
}
```

This program is based on the STM32L4 microcontroller platform and is developed using the STM32CubeIDE environment. The code implements the function of controlling the serial shift register SN74HC595 through GPIO to drive the seven-segment digital tube to display

numbers, and introduces CMSIS-RTOS V2 (FreeRTOS) to implement task scheduling and improve the scalability and real-time performance of the system. In the main program, a task named vTaskSN74HC595 is created through osThreadNew(), which displays the numbers 0-9 in a 500 millisecond cycle.

In terms of hardware connection, the SER, SRCLK, and RCLK of the SN74HC595 are connected to the three pins of GPIOA and GPIOB respectively; OE (output enable) is also controlled by GPIO to achieve soft control of the register output. The data transmission function SN74HC595_SendData(uint8_t data) will serially transmit 8-bit data to the shift register in sequence by shifting, and output the data to the seven-segment digital tube by latching the clock. In the task, a number[10] array is used to store the seven-segment display codes from 0 to 9 for loop calls.

The code follows the basic architecture of FreeRTOS, and the initialization sequence is clear. First, the system clock and UART are configured, then the GPIO is initialized, and then the FreeRTOS kernel and task scheduler are started. vTaskDelay() is used inside the task to implement non-blocking delay, avoiding the system freezing problem caused by using HAL_Delay, thereby ensuring that the system can subsequently expand other concurrent tasks (such as ADC reading, communication sending, etc.).

Singal out put:

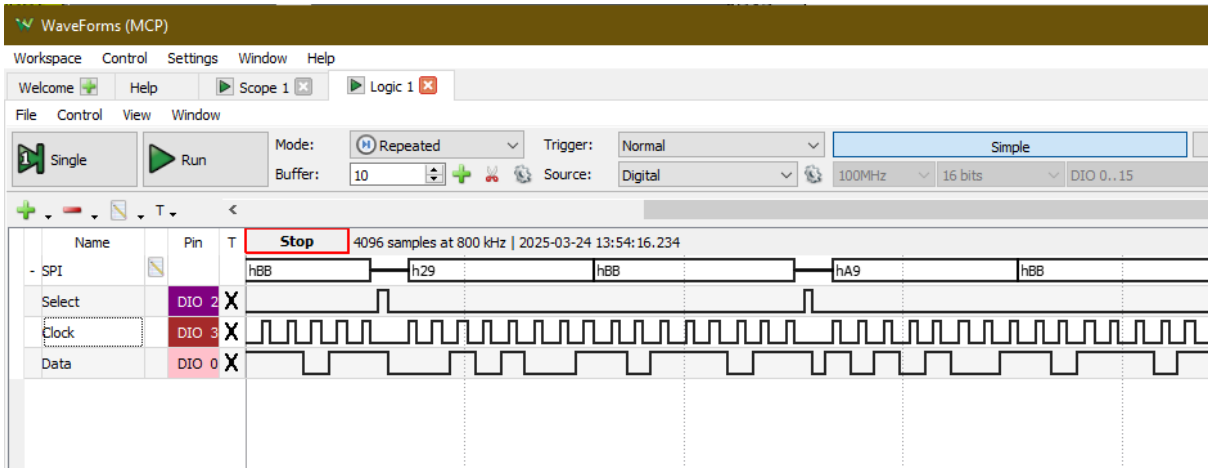


Figure 11 Singal out put for controling 74HC595

The core control logic of the chip revolves around three key pins: DS (serial data input), SHCP (shift clock input) and STCP (latch clock input). In actual operation, whenever SHCP

receives a rising edge pulse, the current level state on the DS pin will be sampled and shifted into the lowest bit of the shift register, and the original register content will be shifted right by one bit. After eight consecutive shift operations, a complete byte of data is loaded into the chip's internal shift register. However, the data at this time has not yet been transferred to the parallel output terminals Q0 to Q7. Only when the STCP pin receives a rising edge signal will the content of the shift register be latched into the output register and drive the external circuit. It is worth noting that the output is valid only when the output enable terminal OE is low, and the reset terminal MR low level will clear the register, so it is usually connected to a high level to maintain normal working state.

Chapter 4 Technical details – Hardware

Refers to the drawing in Appendicitis C drawing 1, 3, 4, 5, 6; Appendix E: Header Table

The hardware development has been completed, and the corresponding drawings have been produced and delivered for PCB design. In the following chapters, we will introduce the design of each module.

Digital Output Module:

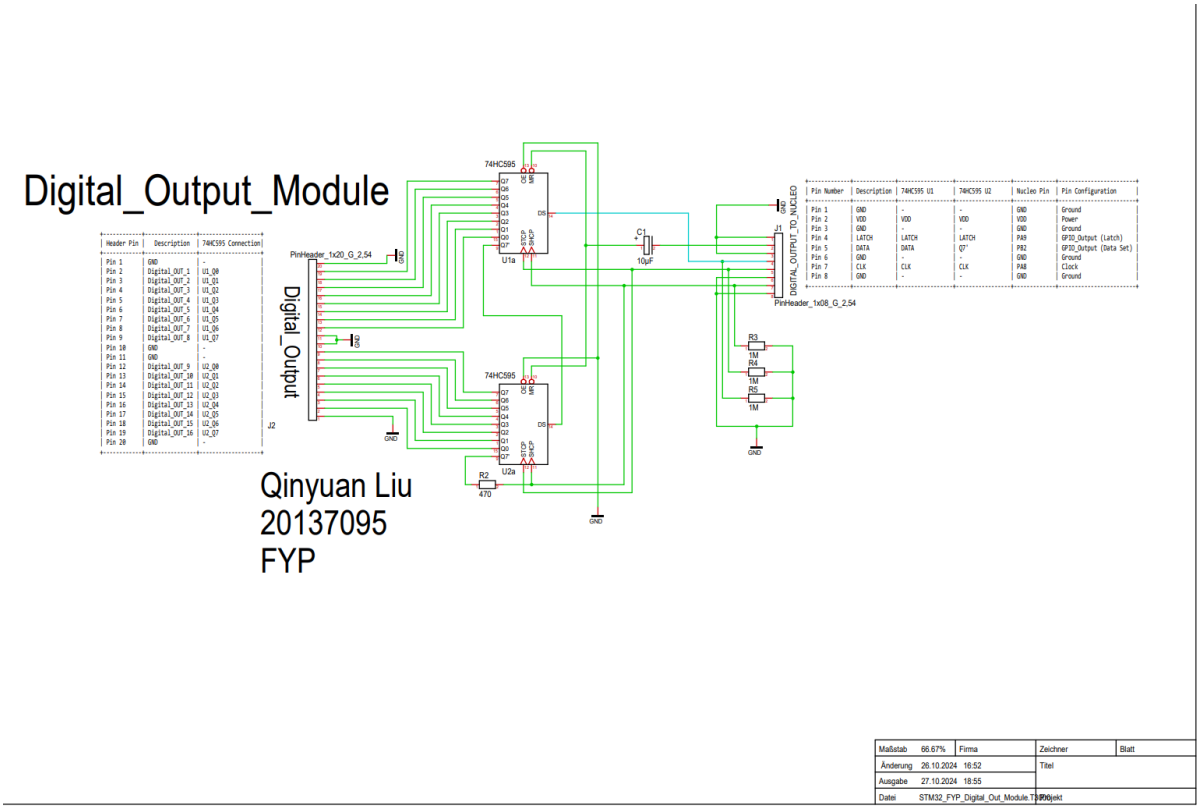


Figure 12 Schematics of Digital OutPut Module

Figure Digital_Output_Module shows the detailed schematic diagram of the digital output module in this project. The module realizes the extended control of 16 digital outputs based on two 74HC595 serial-to-parallel shift registers and is the core component in the digital I/O subsystem of the entire system. The module has the advantages of low cost, high integration and good scalability, and is suitable for embedded control systems that require a large number of digital output channels.

Module Functional Structure

The core components are two 74HC595 chips connected in series, numbered U1a and U2a. They are controlled by serial input (DS), shift clock (SHCP, usually marked as CLK) and latch clock (STCP, usually marked as LATCH) to convert the serial data of the MCU into 8-bit parallel output. The two chips are connected in a daisy chain, that is, the Q7' output of the first chip is connected to the DS input of the second chip, realizing 16-bit cascade control.

The output ports are uniformly led out through the PinHeader_1x20_G_2.54 interface (J2), and the naming method is such as U1_Q0, U2_Q3, etc., corresponding to the parallel output pins (Q0-Q7) of each 74HC595. This structure supports external digital loads such as relays, LEDs, and switch quantity control

The table in the upper right corner defines the signal allocation relationship of the control interface J1

LATCH (STCP): Corresponding to the latch signal line, used to output the data in the shift register to Q0~Q7. Connected to the PB9 pin of the Nucleo board.

DATA (DS): Serial data input, connected to PB2.

CLK (SHCP): Shift clock signal, the data input rhythm is controlled by the MCU, corresponding to PA8 of the Nucleo board.

GND, VDD: Provide power supply and reference ground for all chips.

A 10μF electrolytic capacitor (C1) is also introduced in the figure as a power filter to suppress the spike interference of the VDD voltage and ensure the stable operation of the shift register. At the same time, a 470Ω resistor (R2) and several 1MΩ pull-up resistors (R3~R5) are connected in series in the output channel for basic current limiting and level stabilization.

Expandability: This design supports multi-chip cascading. Just connect the Q7' of the second 74HC595 to the DS of the next one to achieve control of 24, 32 or even higher channels.

Strong compatibility: All signal pins are led out through standard 2.54mm spacing pin headers, which is convenient for subsequent welding or connection to the motherboard, breadboard or test equipment

Simple control method: Only three control lines (LATCH, DATA, CLK) are needed to achieve 16-bit data control, which greatly saves the IO resources of the MCU

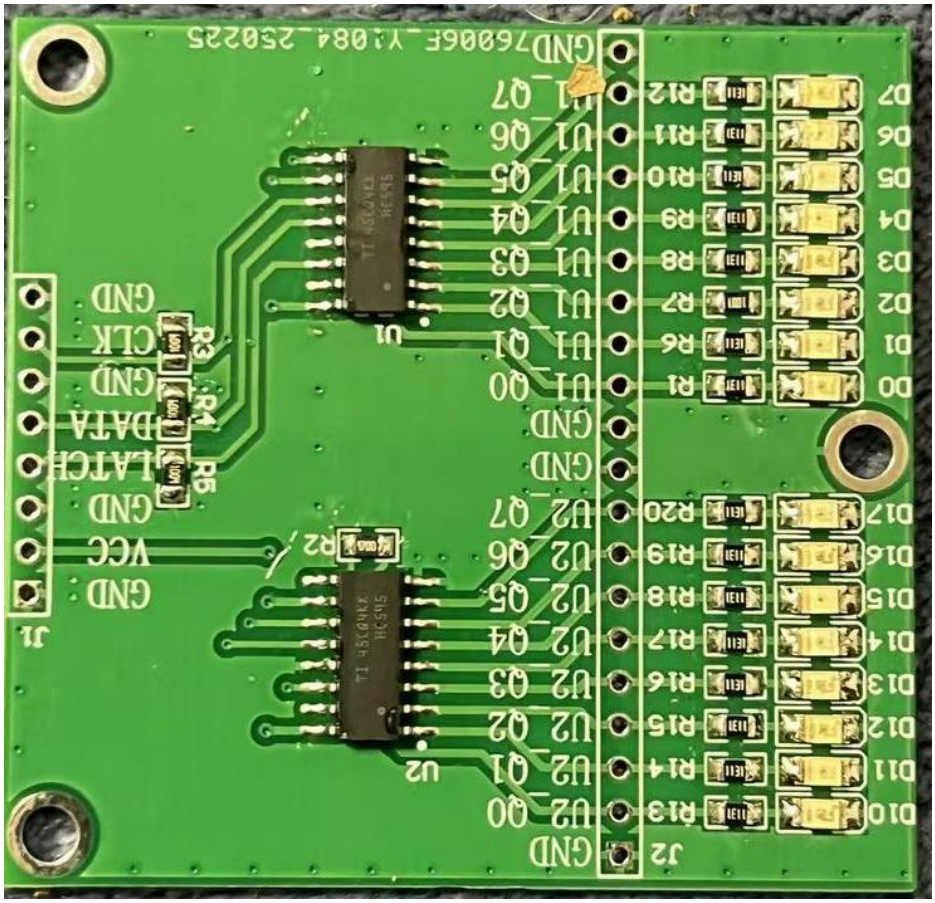


Figure 13 Picture of the PCB prototype - Digital Output

Digital Input Module：

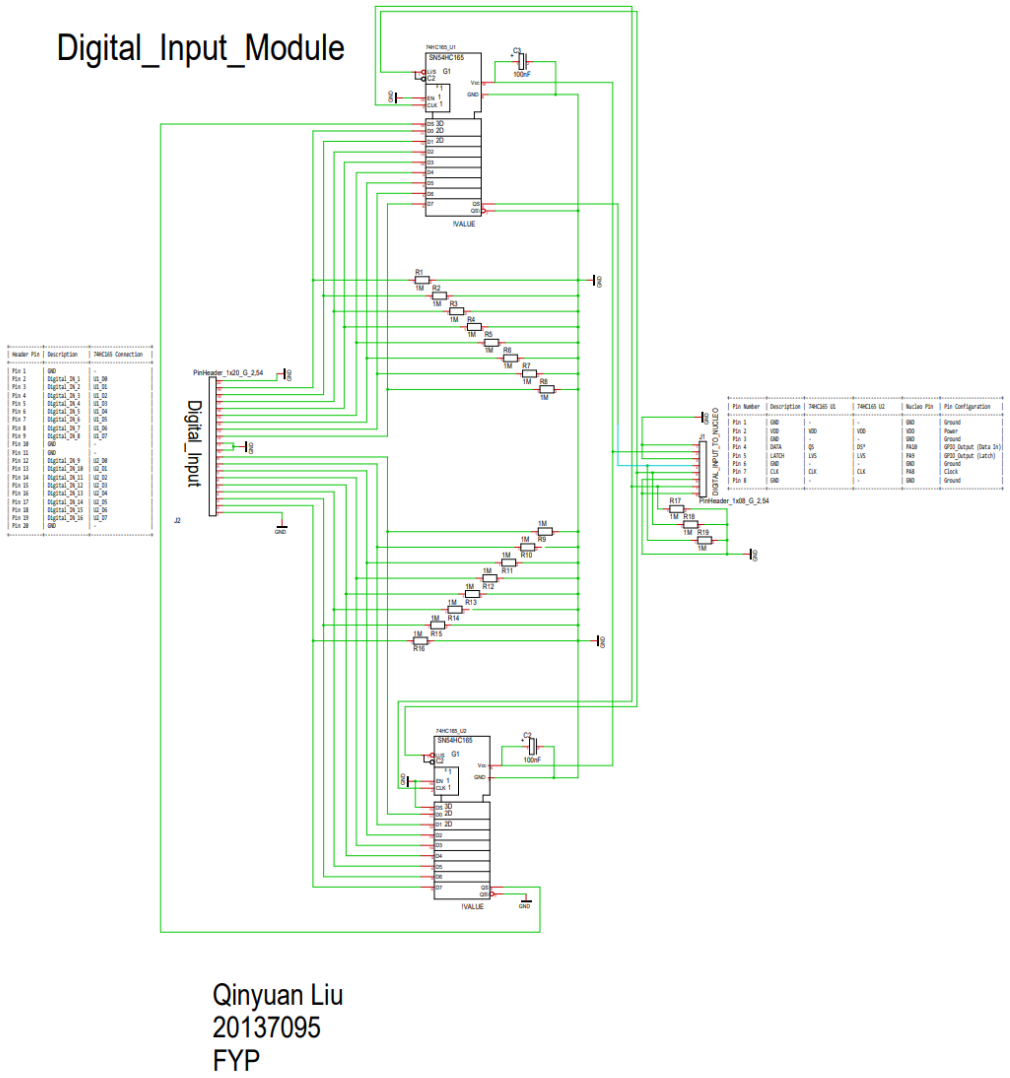


Figure 14 Schematics of Digital Input1 Module

Figure Digital_Input_Module shows the complete schematic structure of the digital input module in this project. This module realizes the acquisition of 16 digital input signals through two 74HC165 parallel-to-serial shift register chips. It is an input subsystem supporting the digital output module and is designed for high-channel signal reading. The module supports stable sampling at low-speed scanning frequency and is suitable for state acquisition of buttons, sensor triggers, mechanical switches, etc.

In the schematic diagram, the core components are two 74HC165 parallel-to-serial shift registers, marked as U1a and U2a, which are responsible for collecting data from external parallel inputs and sending them serially to the main control MCU. The 16 input signals are led out through the PinHeader_1x20_G_2.54 interface (J2), numbered as U1_Q0~U2_Q7, corresponding to the 16 input channels one by one. Each input is connected in series with a current limiting resistor and a pull-down resistor to ensure stable level in the high-impedance input state.

U1a and U2a are connected in series in a daisy chain, and the QH output of the second chip is connected to the SER input of the first chip, so that all collected data are transmitted in sequence

Signal control logic

This module mainly uses three control signals to communicate with the STM32 master:

CLK (clock signal): input sampling synchronization pulse, shifted once per rising edge

SH_LD (parallel trigger): low level trigger to merge input data into the register, and then shift output.

QH (serial output): connected to the SPI_MISO interface of the MCU, used to read the 16-bit input status

The interface table in the upper right corner lists the connection relationship of each signal line, indicating the electrical connection between the module and the NUCLEO-L476RG development board, including:

PB4 (SH/LD): control latch trigger;

PB3 (CLK): clock input;

PA6 (QH): receive serial output data

The power supply and ground are uniformly provided by VDD and GND.

Signal anti-shake and electrical characteristics design

A 10kΩ pull-down resistor (R1~R16) is connected in series to the front stage of all input channels to ensure that the default state is low level to avoid signal jitter or misjudgment caused by input suspension. A 100nF decoupling capacitor is also added to the upper right corner of the module for power supply filtering to suppress the power supply noise that may be generated when the logic chip is working, further improving the stability of the system

High channel count integration: 16 inputs are achieved through two 74HC165s, greatly reducing MCU IO occupancy

Standard interface layout: The signal line is led out through a 2.54mm pin header for easy system integration;

Good compatibility: It can be connected to digital signal sources such as switches, Hall sensors, infrared detection, etc.

Clear structure and strong scalability: Supports cascade design, which is convenient for the future expansion of more input bits.

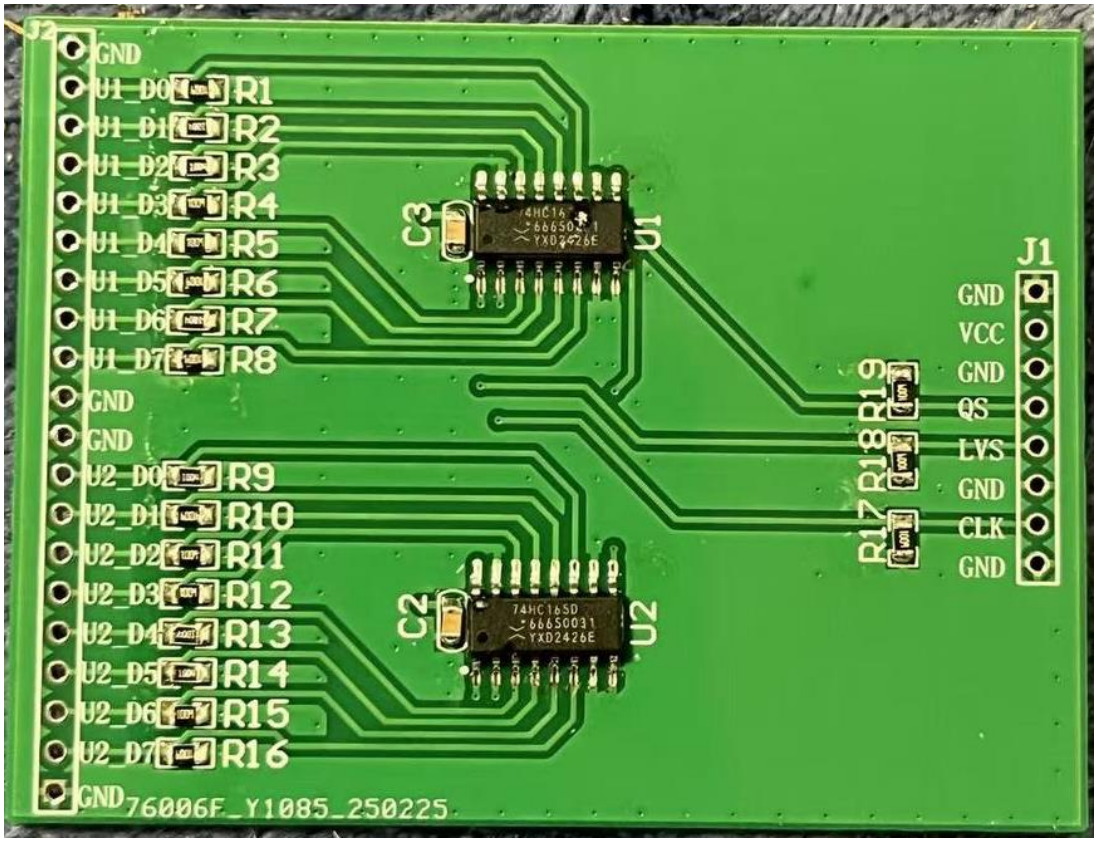


Figure 15 Picture of the PCB prototype - Digital Input

DAC Analog Output:

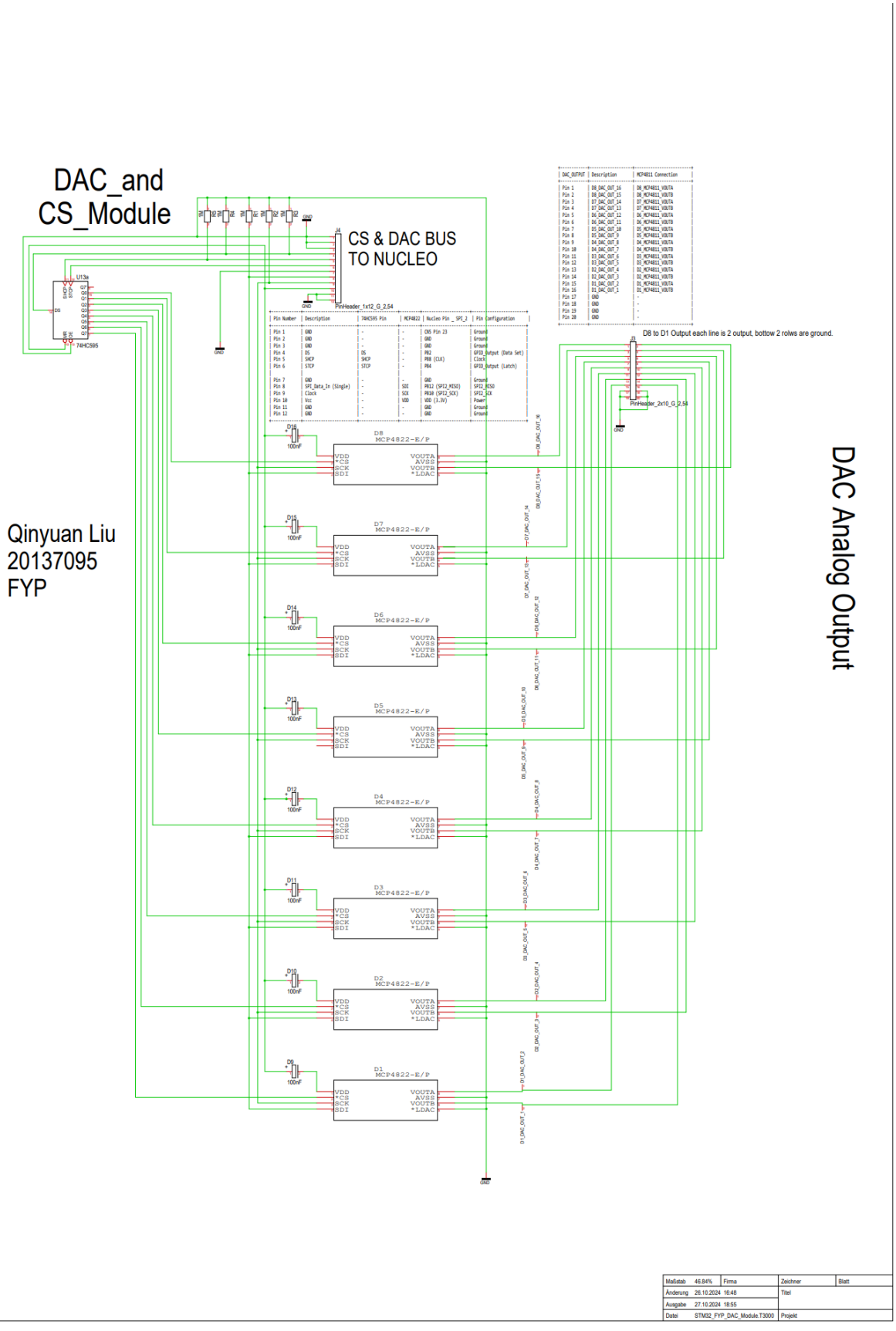


Figure 16 DAC Analog Output

The drawing "DAC_and_CS_Module" shows the complete design schematic of the analog signal output part of this project. This module realizes 16 high-precision analog signal outputs through eight MCP4822 dual-channel DAC chips, and uses SPI bus communication and chip select control expansion mechanism to realize multi-channel DAC control. This part is the key component of the entire data acquisition and control system, responsible for converting MCU digital signals into analog quantities (0~4.096V) required by external devices

1. Module core architecture

The eight MCP4822-E/P chips (U3~U10) arranged vertically in the figure are the analog signal output units of this system. Each chip has two 12-bit DAC output channels (VOUTA/VOUTB), providing a total of 16 analog outputs, which are concentrated on the right interface PinHeader_2x10_G_2.54 output, marked as DAC Analog Output.

All MCP4822 chips share the following bus signals

- SCK (SPI clock)
- SDI (SPI data input)
- LDAC (update latch signal)

The chip select signal CS is provided by the 74HC595 serial-to-shift register (U1a) in the upper left corner, and the CS pin of each DAC chip is controlled through 8 parallel output pins to achieve multi-device SPI expansion

2. Communication and control mechanism

SPI bus sharing:

> MCP4822 supports standard SPI communication protocol. The SCK and SDI signals of all chips are connected to the MCU SPI bus, and the wiring is simplified by sharing the clock and data lines.

Chip select (CS) expansion design:

> Since the actual GPIO resources of STM32 are limited and insufficient to independently control all 8 DACs, a 74HC595 serial-to-parallel shift register is used in the design for chip select control:

The MCU sends a serial control byte to the 74HC595;

It latches it as an 8-bit parallel chip select signal at the rising edge of RCLK;

Each bit controls the CS level of an MCP4822 chip;

Through the dynamic switching of the chip select signal, different DAC chips can be accessed one by one.

This design greatly saves GPIO usage and has good scalability and flexibility

III. Power supply and signal integrity design

The VDD and REF pins of each MCP4822 chip are connected to the system 5V power supply, and the output voltage range is 0~4.096V. Each chip's VOUTA/B is connected to the ground with a 0.1μF decoupling capacitor to filter out high-frequency interference from power supply and digital signals, ensuring the purity of the analog output signal. The DAC output pins are finally brought out through the interface below for sampling by other modules or peripherals.

The power supply of 74HC595 is also connected to 5V, and its three control lines (SHCP, STCP, DS) are connected to STM32 GPIO to realize its shift and latch control and assist in completing the expansion of multiple CS chip select signals.

This figure uses a clear naming standard in the wiring design. Each signal line is named by function (such as DAC_CS_1, DAC_SDI, etc.), which helps to clarify the logic and read the software layer. At the same time, a clear mark "CS & DAC BUS TO NUCLEO" is used on the top of the module to indicate the data interaction path between the master control and the DAC module.

Although there are labelling errors in the wiring table in the upper right corner of the figure, it will not affect the actual circuit understanding and development. The real connection relationship has been clearly shown through the schematic wiring.

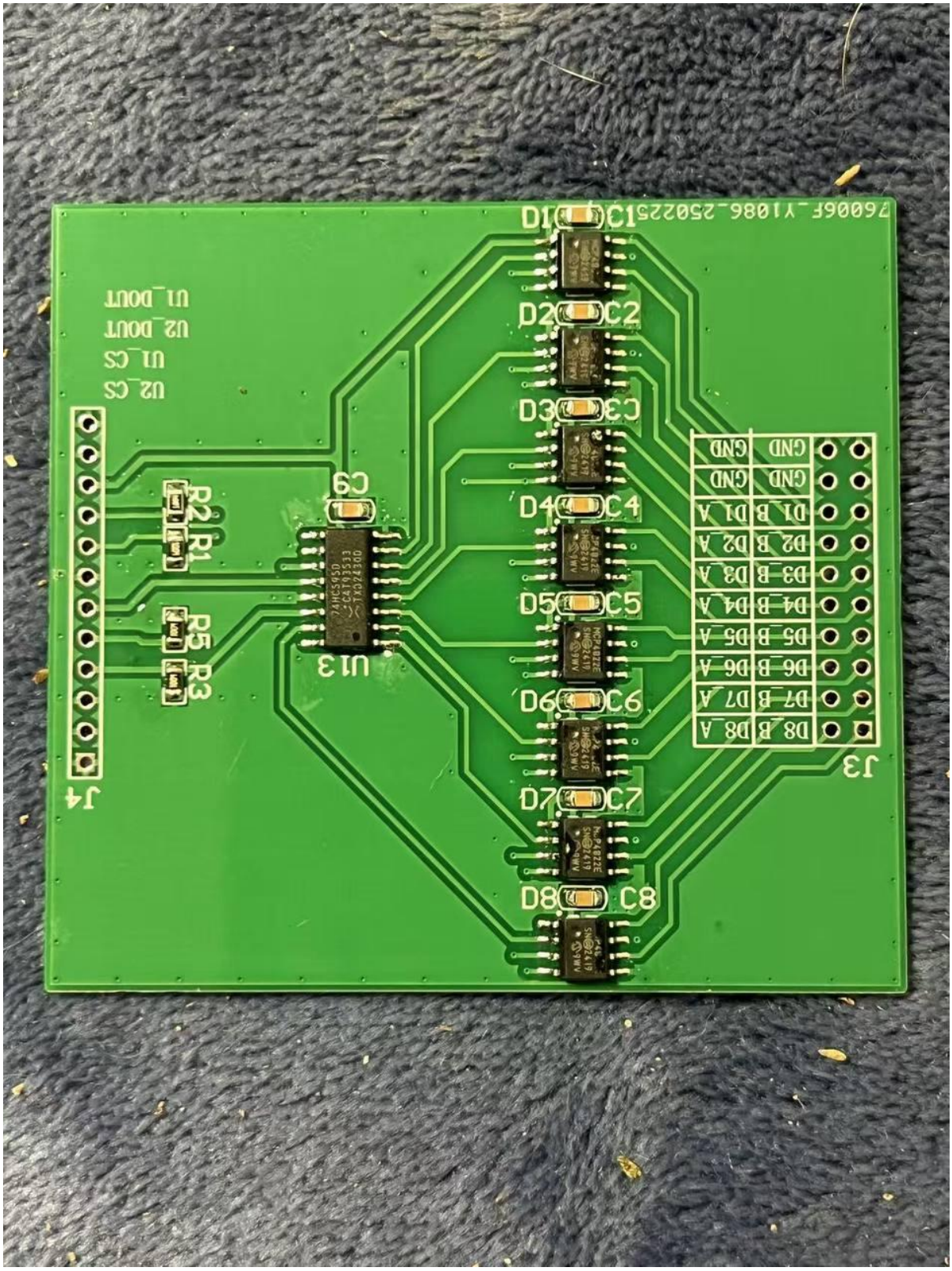


Figure 17 Picture of the PCB prototype DAC Output

ADC Analog Input:

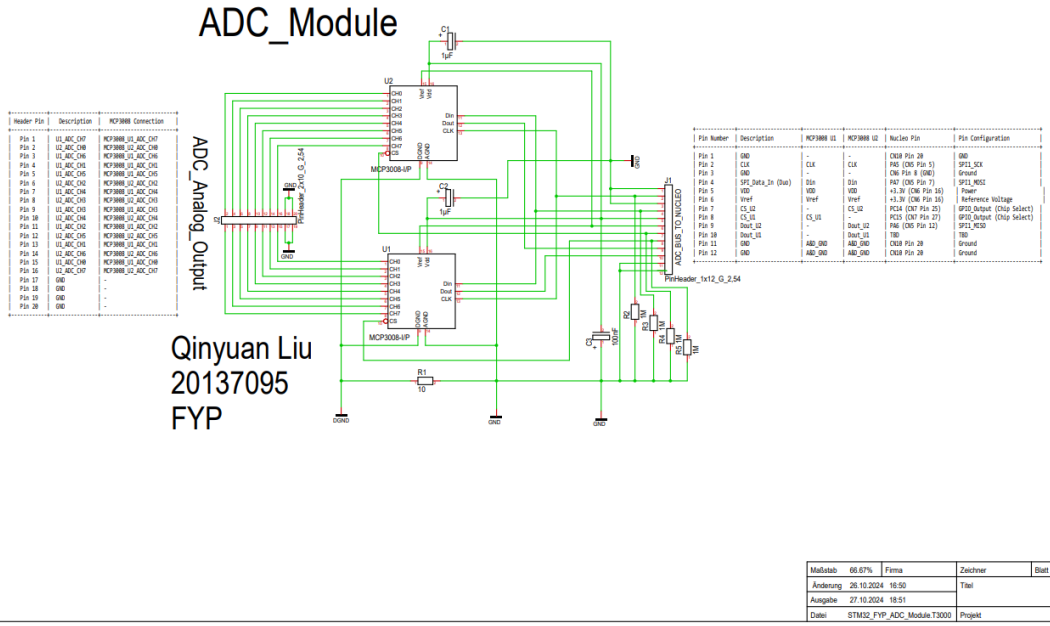


Figure 18 Schematics of ADC Module

This module is the analog signal acquisition module (Analog-to-Digital Conversion Module) in this project. Its main function is to convert external analog signals into digital signals and send them to the main control chip STM32 through the SPI interface for subsequent processing. Two MCP3008-I/P analog-to-digital converters (ADCs) are used in the figure. Each chip supports 8 single-ended inputs, and the overall 16-channel analog input acquisition function is realized.

I. Module Function and Overview

The overall structure of the drawing is clear and divided into three major areas:

The left side is the analog signal input interface (ADC_Analog_Output), which provides 16 single-ended input channels;

The middle part is two MCP3008 chips, named U1 and U2 respectively;

The right side is the SPI bus communication and control signal interface, which communicates with the STM32 main control through the pin header PinHeader_1x10_G_2.54.

2. Introduction to MCP3008 chip

MCP3008 is a widely used 10-bit SPI interface ADC chip with the following features:

Supports 8 single-ended inputs or 4 differential inputs;

Communication protocol is compatible with standard SPI;

Inside contains sample-and-hold circuit (Sample-and-Hold);

The power supply voltage is 5V, and the analog input range is 0~VREF (the default is 5V).

In this design, U1 and U2 are responsible for collecting 8 analog signals (AIN0~AIN7) respectively, and share the SPI bus for communication.

3. Communication and control logic

The SPI bus includes the following signal lines:

CLK (clock) → Control data sampling rate;

MISO (data output) → Transmit digital signals from ADC to master control;

MOSI (data input) → STM32 sends control commands to ADC;

CS (chip select) → Low level is valid, activate the corresponding ADC.

In the interface area in the upper right corner of the drawing, each MCP3008 has an independent chip select line (CS1, CS2), which is controlled by the STM32 GPIO pin. The remaining SPI bus signals are shared by U1 and U2 to achieve serial communication of multiple devices

By pulling down CS1 or CS2, U1 or U2 can be selected for data interaction. Under the action of clock CLK, STM32 sends control words (channel selection + start bit, etc.) to ADC. After ADC collects the voltage of the specified channel, it returns the result to the master through MISO

4. Power supply and reference voltage design

The power supply pins (VDD, VREF) of the two MCP3008s are connected to the system +5V power supply, and are grounded and filtered through 0.1μF decoupling capacitors (C1, C2) to suppress power supply ripple interference and improve ADC accuracy and stability. The AGND and DGND pins of the chip are grounded uniformly to ensure the reference consistency between analog and digital signals.

The PinHeader_2x10_G_2.54 on the left provides 16 analog input channel interfaces for connecting external signal sources. Each channel is connected to the AINx pins of U1 and U2 through internal wiring to achieve 0~5V voltage sampling. The interface is clearly arranged and named in a standardized manner, which is convenient for host computer identification and subsequent testing.

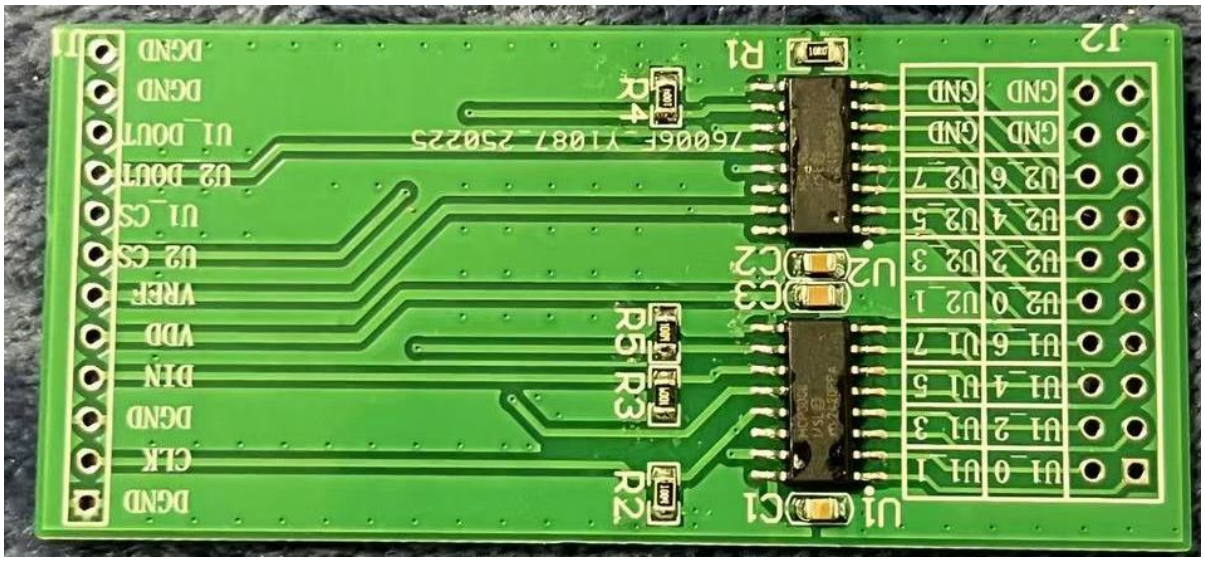


Figure 19 PCB Prototype of the ADC Module

Proposed implementation:

1. Data Acquisition System Modules: (Master Scanning FSM controlled)

In the current design. 4 module is developed for the data acquisition system.

1. **DAC module** (MCP3008 * 2) -- **Appendix D drawing 4**
2. **ADC module** (MCP4822 * 8 and 74HC595 as CS) -- **Appendix D drawing 3**
3. **Digital Input** (74HC165 * 2) -- **Appendix D drawing 5**
4. **Digital Output** (74HC595 * 2) -- **Appendix D drawing 6**

The Overview of the Prototype Hardware Configuration:

Appendix D, Drawing 1: *General Data Acquisition System Block Diagram*, illustrates the layout of the project hardware. The project consists of four custom-designed modules to achieve 16 × 16 analog I/O and 16 × 16 digital I/O. The selection of chips was made in consultation with Dr. Grout, the project supervisor, and the schematics were thoroughly reviewed and iterated from weeks 4 to 7.

The components have arrived, and all modules will be tested on a breadboard during weeks 8 and 9 for further refinement. The pin configuration details are provided in Appendix E: *Header Table*.

Chapter 5 Conclusions and future work

This project aims to build a general data acquisition system based on the STM32 platform and FreeRTOS operating system, with the expansion capability of 16×16 digital IO and 16×16 analog IO, and can collect and control data for all channels at a frequency of 1kHz, and communicate with the host computer in real time through the serial port. During the entire project development cycle, we have successfully completed all the work from system design, prototype verification, module testing to final integration, and achieved the expected goals in terms of functionality, stability and scalability.

First, we completed the overall architecture design of the system, including hardware topology, functional module division and task scheduling framework construction. In the early stage, we determined to use the SPI bus to connect the MCP3008 and MCP4822 chips based on demand analysis, and at the same time used shift registers and multiplexers to reduce the number of IO pins required for STM32. This hardware design has been rigorously verified through schematic modeling, providing a solid foundation for subsequent PCB design.

Next, we built the first prototype system on the breadboard and tested the analog input acquisition, analog output control and digital signal reading and writing one by one. By cooperating with the STM32 Nucleo board, we realized the basic ADC and DAC channel communication test, and through the FreeRTOS multi-task scheduling mechanism, we realized the system-level task division, such as parallel execution of tasks such as data acquisition, communication transmission, and status indication. Based on the 1kHz timer interrupt, we built a three-level scheduling architecture to ensure that the system has good real-time performance and responsiveness.

During the testing phase, we used Analog Discovery Kit MK2 to carefully measure and record the signal waveform, SPI communication timing, and system response delay. All hardware modules and firmware logic have undergone multiple rounds of iterative optimization to ensure that they can still run stably in an interference environment. At the same time, we wrote a serial communication host computer tool based on Python to achieve

real-time control and data visualization of all channels, and verified the reliability of the system under high-load operation.

After completing all functional verifications in the breadboard stage, we entered the PCB design stage. We used Torgit PCB (Beta Layout) software to complete the layout and routing of the system, and carefully optimized the power distribution, signal integrity and EMI suppression. After the PCB proofing was completed, we performed welding, assembly and functional regression tests to confirm that its functions were consistent with the prototype, the structure was more compact, and the performance was more stable.

Finally, we conducted a comprehensive test on the entire system, including the maintenance of IO scanning frequency, temperature rise control under long-term operation, module communication stability and the implementation of power redundancy strategy. The project results were excellent and all design indicators were achieved. It is worth mentioning that this project has accumulated a lot of experience in PCB design and multi-chip communication collaboration, especially in the high-frequency SPI communication between STM32 and multiple peripherals, successfully avoiding common problems such as bus conflicts and timing errors.

Through the implementation of this project, we not only completed a data acquisition and control platform with practical application value, but also laid a solid foundation for the development of subsequent more complex systems. In the future expansion direction, the system can be expanded into a multi-functional platform suitable for industrial control, robot sensor integration, or scientific experiments by replacing the interface module or upgrading the main control chip.

Chapter 6 References

[1] STMicroelectronics, "NUCLEO-XXXXCX, NUCLEO-XXXXRX, NUCLEO-XXXXRX-P, NUCLEO-XXXXRX-Q: STM32 Nucleo-64 boards," Data brief, Rev 19, June 2024. Available: https://www.st.com

[2] IntechHouse, "Real-Time Operating System in Embedded Systems," IntechHouse Blog, June 14, 2023. [Online]. Available: https://intechhouse.com/blog/real-time-operating-system-im-embedded-systems/. [Accessed: 20-Oct-2024].

[3] RTEMS Project. *RTEMS 5.1 Documentation* - Doxygen Documentation. https://ftp.rtems.org/pub/rtems/releases/5/5.1/docs/html/doxygen/RTEMSPreface.html (Accessed October 28, 2024).

[4] ARM CMSIS Project. *CMSIS-RTOS V2 API Documentation*. https://arm-software.github.io/CMSIS_5/RTOS2/html/index.html (Accessed October 28, 2024).

[5] K. Wahome, "The first bug on Mars: OS scheduling, priority inversion, and the Mars Pathfinder," *Medium*, 16-Aug-2020. [Online]. Available: https://kwahome.medium.com/the-first-bug-on-mars-os-scheduling-priority-inversion-and-the-mars-pathfinder-53586a631525. [Accessed: 28-Oct-2024].

[6] RTEMS Project, RTEMS User Manual, Release 7.e8e6f12, 8th March 2025. [Online]. Available: https://www.rtems.org/docs.

[7] AdaCore, AdaCore's GNAT Pro Available for Wind River VxWorks 7, press release, Jul. 21, 2015. [Online]. Available: [AdaCore Website or News Archive URL].

[8] ARM Ltd., CMSIS-RTOS2 Version 2.1.3: Real-Time Operating System API and RTX Reference Implementation, May 2, 2022. [Online]. Available: https://arm-software.github.io/CMSIS_5/RTOS2/html/index.html.

Additional Acknowledgment

Special thanks to my dad, who provided valuable insight into engineering drawings and introduced me to the world of engineering. A heartfelt thank you to my mom for her immense support during my injury while I was working on this project. I am also deeply grateful to all my friends who helped take care of me during this difficult time. Finally, I extend my gratitude to the dep of ECE for their comprehensive support throughout this project.

Appendices

Appendix A shows the Drawings in its Print export form.

Appendix A: Drawing

1. **General Data Acquisition System Block Diagram**

   Title: General Data Acquisition System Block Diagram
   Author: Qinyuan Liu
   Institution: University of Limerick
   Student ID: 20137095
   Supervisor: Dr. Ian Grout
   Time: 27/10/2024

2. **RTOS Task Priority Diagram**

   Title: RTOS Task Priority Diagram
   Author: Qinyuan Liu
   Institution: University of Limerick
   Student ID: 20137095
   Supervisor: Dr. Ian Grout
   Time: 27/10/2024

3. **ADC Module Schematic**

   Title: ADC_Module
   Author: Qinyuan Liu
   Institution: University of Limerick
   Student ID: 20137095
   Date: 27/10/2024

4. **DAC Module Schematic**

   Title: DAC_and_CS_Module
   Author: Qinyuan Liu
   Institution: University of Limerick
   Student ID: 20137095
   Date: 27/10/2024

5. **Digital Input Module Schematic**

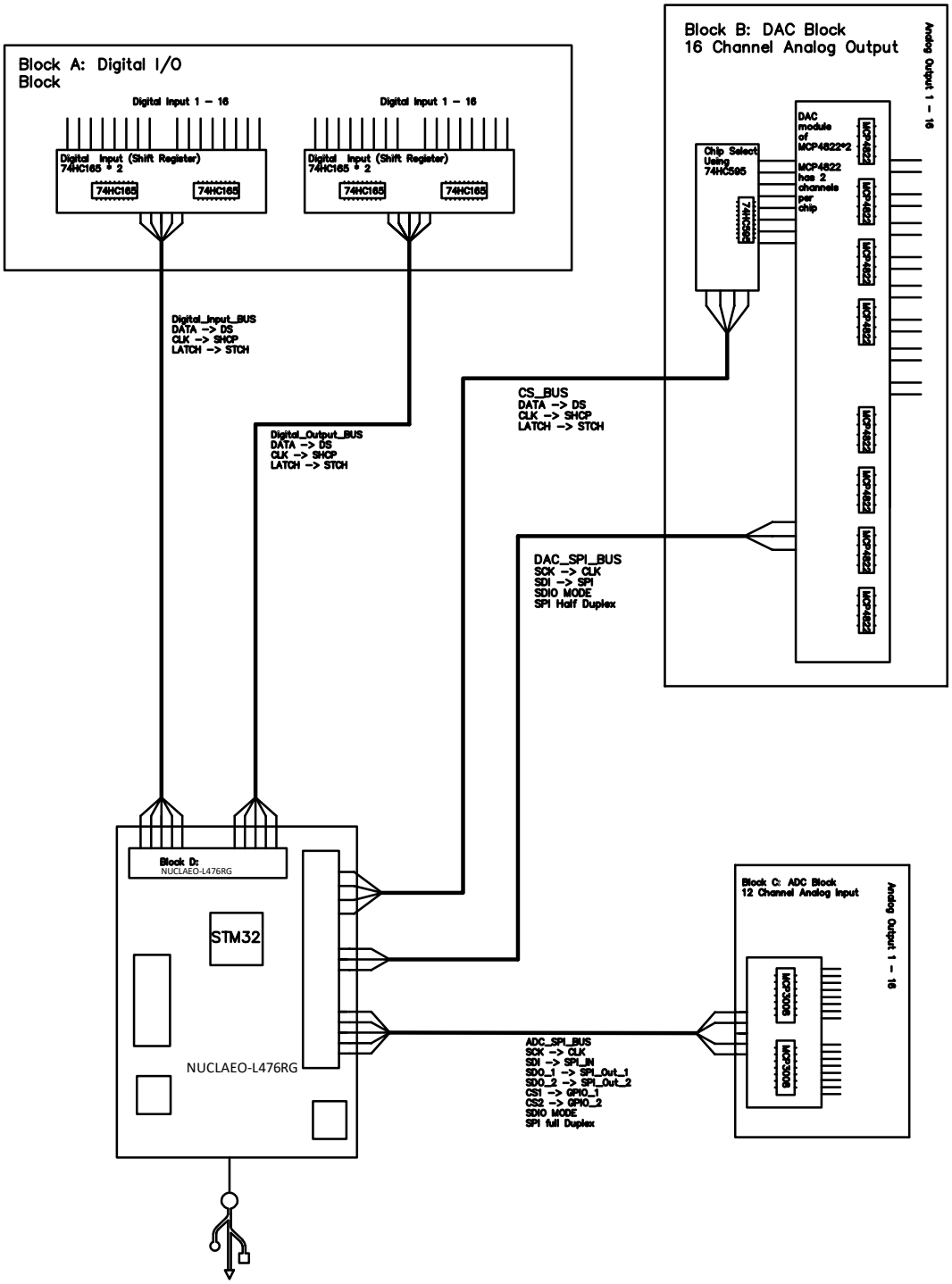
   Title: Digital_Input_Module
   Author: Qinyuan Liu
   Institution: University of Limerick
   Student ID: 20137095
   Date: 27/10/2024

6. **Digital Output Module Schematic**

   Title: Digital_Output_Module
   Author: Qinyuan Liu
   Institution: University of Limerick
   Student ID: 20137095
   Date: 27/10/2024

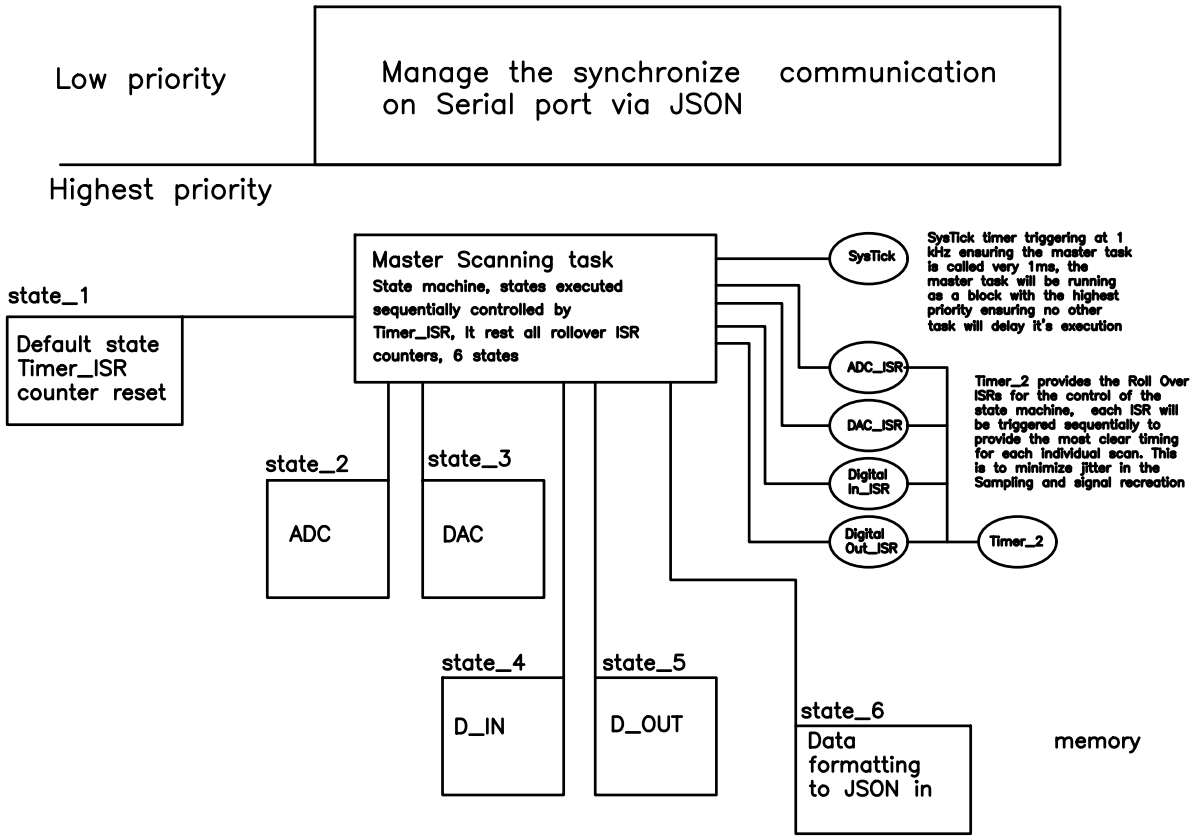
Link to all the Drawings:

https://github.com/Qinyuan72/FYP/tree/master/Drawings



Title: General Data Acquisition System Block Diagram
Author: Qinyuan Liu
Institution: University of Limerick
Student ID: 20137095
Supervisor: Dr. Ian Grout
Time: 27/10/2024

RTOS Task Priority Diagram



General Notes

This is the task Priority Diagram of the project

No.	Revision/Issue	Date

Author: Qinyuan Liu
Supervisor: Dr. Ian Grout

General Purpose Data Acquisition System Using STM32 and FreeRTOS

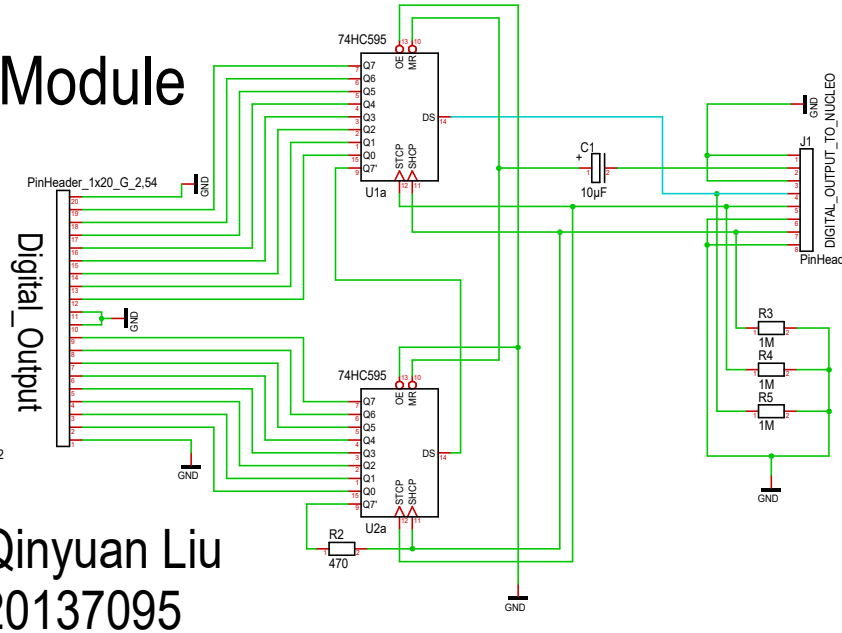
Date: 27.10.2024

Sheet: 01

Date: N/A

Digital_Output_Module

Header Pin	Description	74HC595 Connection
Pin 1	GND	-
Pin 2	Digital_OUT_1	U1_Q0
Pin 3	Digital_OUT_2	U1_Q1
Pin 4	Digital_OUT_3	U1_Q2
Pin 5	Digital_OUT_4	U1_Q3
Pin 6	Digital_OUT_5	U1_Q4
Pin 7	Digital_OUT_6	U1_Q5
Pin 8	Digital_OUT_7	U1_Q6
Pin 9	Digital_OUT_8	U1_Q7
Pin 10	GND	-
Pin 11	GND	-
Pin 12	Digital_OUT_9	U2_Q0
Pin 13	Digital_OUT_10	U2_Q1
Pin 14	Digital_OUT_11	U2_Q2
Pin 15	Digital_OUT_12	U2_Q3
Pin 16	Digital_OUT_13	U2_Q4
Pin 17	Digital_OUT_14	U2_Q5
Pin 18	Digital_OUT_15	U2_Q6
Pin 19	Digital_OUT_16	U2_Q7
Pin 20	GND	-



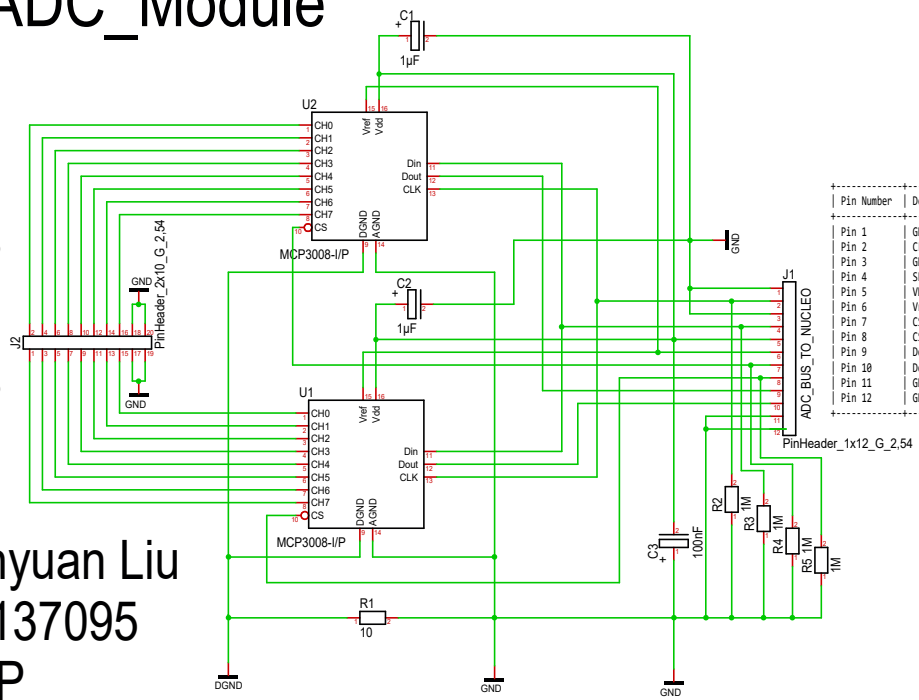
Pin Number	Description	74HC595 U1	74HC595 U2	Nucleo Pin	Pin Configuration
Pin 1	GND	-	-	GND	Ground
Pin 2	VDD	VDD	VDD	VDD	Power
Pin 3	GND	-	-	GND	Ground
Pin 4	LATCH	LATCH	LATCH	PA9	GPIO_Output (Latch)
Pin 5	DATA	DATA	Q7'	PB2	GPIO_Output (Data Set)
Pin 6	GND	-	-	GND	Ground
Pin 7	CLK	CLK	CLK	PA8	Clock
Pin 8	GND	-	-	GND	Ground

Qinyuan Liu
20137095
FYP

Maßstab	66.67%	Firma	Zeichner	Blatt
Änderung	26.10.2024 16:52		Titel	
Ausgabe	27.10.2024 18:55			
Datei	STM32_FYP_Digital_Out_Module.T3000		Projekt	

ADC_Module

ADC_Analog_Output



Header Pin	Description	MCP3008 Connection
Pin 1	U1_ADC_CH7	MCP3008_U1_ADC_CH7
Pin 2	U2_ADC_CH0	MCP3008_U2_ADC_CH0
Pin 3	U1_ADC_CH6	MCP3008_U1_ADC_CH6
Pin 4	U1_ADC_CH1	MCP3008_U1_ADC_CH1
Pin 5	U1_ADC_CH5	MCP3008_U1_ADC_CH5
Pin 6	U2_ADC_CH2	MCP3008_U2_ADC_CH2
Pin 7	U1_ADC_CH4	MCP3008_U1_ADC_CH4
Pin 8	U2_ADC_CH3	MCP3008_U2_ADC_CH3
Pin 9	U1_ADC_CH3	MCP3008_U1_ADC_CH3
Pin 10	U2_ADC_CH4	MCP3008_U2_ADC_CH4
Pin 11	U1_ADC_CH2	MCP3008_U1_ADC_CH2
Pin 12	U2_ADC_CH5	MCP3008_U2_ADC_CH5
Pin 13	U1_ADC_CH1	MCP3008_U1_ADC_CH1
Pin 14	U2_ADC_CH6	MCP3008_U2_ADC_CH6
Pin 15	U1_ADC_CH0	MCP3008_U1_ADC_CH0
Pin 16	U2_ADC_CH7	MCP3008_U2_ADC_CH7
Pin 17	GND	-
Pin 18	GND	-
Pin 19	GND	-
Pin 20	GND	-

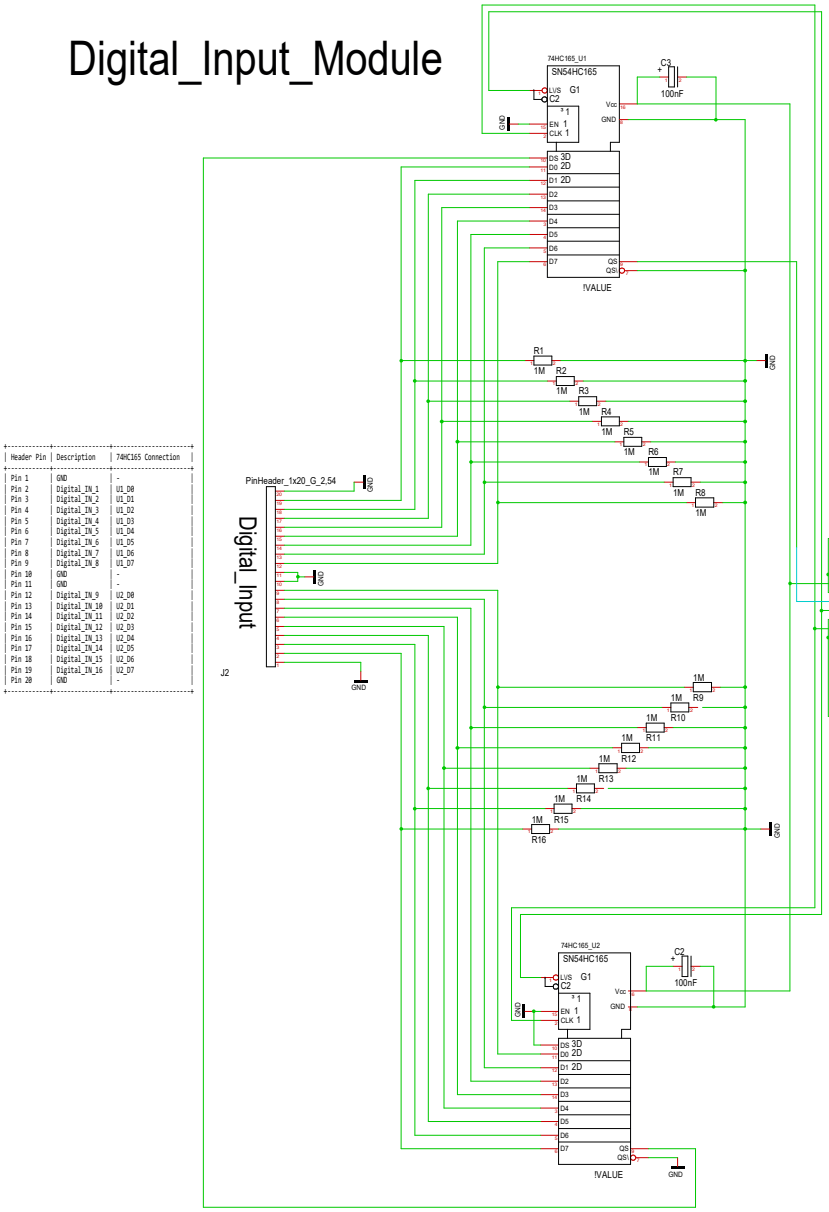
Pin Number	Description	MCP3008 U1	MCP3008 U2	Nucleo Pin	Pin Configuration
Pin 1	GND	-	-	CN10 Pin 20	GND
Pin 2	CLK	CLK	CLK	PA5 (CN5 Pin 5)	SPI1_SCK
Pin 3	GND	-	-	CN6 Pin 8 (GND)	Ground
Pin 4	SPI_Data_In (Duo)	Din	Din	PA7 (CN5 Pin 7)	SPI1_MOSI
Pin 5	VDD	VDD	VDD	+3.3V (CN6 Pin 16)	Power
Pin 6	Vref	Vref	Vref	+3.3V (CN6 Pin 16)	Reference Voltage
Pin 7	CS_U2	-	CS_U2	PC14 (CN7 Pin 25)	GPIO_Output (Chip Select)
Pin 8	CS_U1	CS_U1	-	PC15 (CN7 Pin 27)	GPIO_Output (Chip Select)
Pin 9	Dout_U2	-	Dout_U2	PA6 (CN5 Pin 12)	SPI1_MISO
Pin 10	Dout_U1	-	Dout_U1	TBD	TBD
Pin 11	GND	A&D_GND	A&D_GND	CN10 Pin 20	Ground
Pin 12	GND	A&D_GND	A&D_GND	CN10 Pin 20	Ground

Qinyuan Liu
20137095

FYP

Maßstab	66.67%	Firma	Zeichner	Blatt
Änderung	26.10.2024 16:50		Titel	
Ausgabe	27.10.2024 18:51			
Datei	STM32_FYP_ADC_Module.T3000		Projekt	

Digital_Input_Module



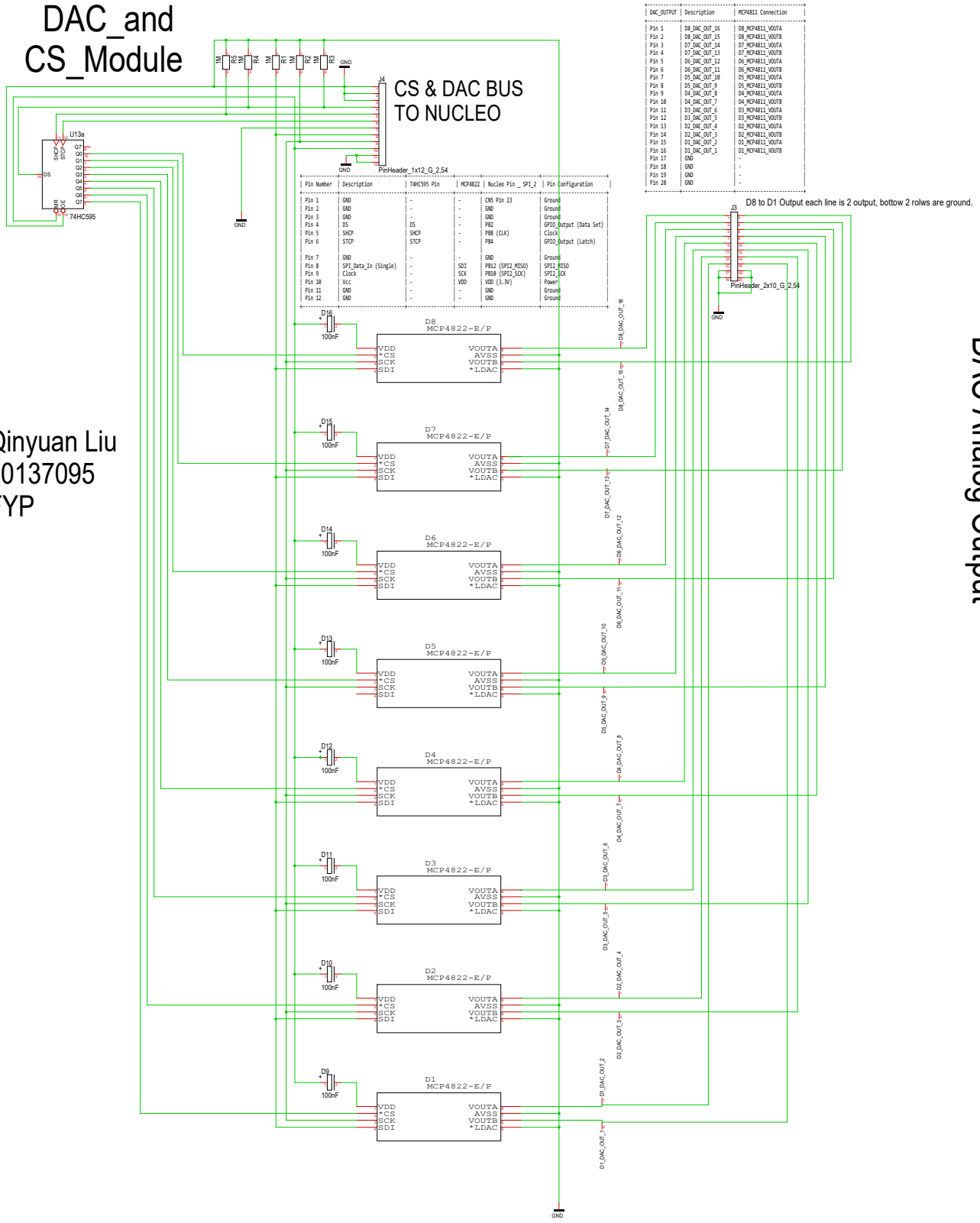
Header Pin	Description	74HC165 Connection
Pin 1	GND	-
Pin 2	Digital_IN_1	U1_D0
Pin 3	Digital_IN_2	U1_D1
Pin 4	Digital_IN_3	U1_D2
Pin 5	Digital_IN_4	U1_D3
Pin 6	Digital_IN_5	U1_D4
Pin 7	Digital_IN_6	U1_D5
Pin 8	Digital_IN_7	U1_D6
Pin 9	Digital_IN_8	U1_D7
Pin 10	GND	-
Pin 11	GND	-
Pin 12	Digital_IN_9	U2_D0
Pin 13	Digital_IN_10	U2_D1
Pin 14	Digital_IN_11	U2_D2
Pin 15	Digital_IN_12	U2_D3
Pin 16	Digital_IN_13	U2_D4
Pin 17	Digital_IN_14	U2_D5
Pin 18	Digital_IN_15	U2_D6
Pin 19	Digital_IN_16	U2_D7
Pin 20	GND	-

Pin Number	Description	74HC165 U1	74HC165 U2	Nucleo Pin	Pin Configuration
Pin 1	GND	-	-	GND	Ground
Pin 2	VDD	VDD	VDD	VDD	Power
Pin 3	GND	-	-	GND	Ground
Pin 4	DATA	Q5	D5*	PA10	GPIO_Output (Data In)
Pin 5	LATCH	LVS	LVS	PA9	GPIO_Output (Latch)
Pin 6	GND	-	-	GND	Ground
Pin 7	CLK	CLK	CLK	PA8	Clock
Pin 8	GND	-	-	GND	Ground

Qinyuan Liu
20137095
FYP

Maßstab 46.84%	Firma	Zeichner	Blatt
Änderung 27.10.2024 18:53		Titel	
Ausgabe 27.10.2024 18:54			
Datei STM32_FYP_Digital_In_Module .T3000		Objekt	

DAC_and CS_Module



Pin Number	Description	74HC595 Pin	MCP4822	Nucleo Pin _ SPI_2	Pin Configuration
Pin 1	GND	-	-	CN5 Pin 23	Ground
Pin 2	GND	-	-	GND	Ground
Pin 3	GND	-	-	GND	Ground
Pin 4	DS	DS	-	PB2	GPIO_Output (Data Set)
Pin 5	SHCP	SHCP	-	PB8 (CLK)	Clock
Pin 6	STCP	STCP	-	PB4	GPIO_Output (Latch)
Pin 7	GND	-	-	GND	Ground
Pin 8	SPI_Data_In (Single)	-	SDI	PB12 (SPI2_MISO)	SPI2_MISO
Pin 9	Clock	-	SCK	PB10 (SPI2_SCK)	SPI2_SCK
Pin 10	Vcc	-	VDD	VDD (3.3V)	Power
Pin 11	GND	-	-	GND	Ground
Pin 12	GND	-	-	GND	Ground

DAC_OUTPUT	Description	MCP4811 Connection
Pin 1	D8_DAC_OUT_16	D8_MCP4811_VOUTA
Pin 2	D8_DAC_OUT_15	D8_MCP4811_VOUTB
Pin 3	D7_DAC_OUT_14	D7_MCP4811_VOUTA
Pin 4	D7_DAC_OUT_13	D7_MCP4811_VOUTB
Pin 5	D6_DAC_OUT_12	D6_MCP4811_VOUTA
Pin 6	D6_DAC_OUT_11	D6_MCP4811_VOUTB
Pin 7	D5_DAC_OUT_10	D5_MCP4811_VOUTA
Pin 8	D5_DAC_OUT_9	D5_MCP4811_VOUTB
Pin 9	D4_DAC_OUT_8	D4_MCP4811_VOUTA
Pin 10	D4_DAC_OUT_7	D4_MCP4811_VOUTB
Pin 11	D3_DAC_OUT_6	D3_MCP4811_VOUTA
Pin 12	D3_DAC_OUT_5	D3_MCP4811_VOUTB
Pin 13	D2_DAC_OUT_4	D2_MCP4811_VOUTA
Pin 14	D2_DAC_OUT_3	D2_MCP4811_VOUTB
Pin 15	D1_DAC_OUT_2	D1_MCP4811_VOUTA
Pin 16	D1_DAC_OUT_1	D1_MCP4811_VOUTB
Pin 17	GND	-
Pin 18	GND	-
Pin 19	GND	-
Pin 20	GND	-

Qinyuan Liu
20137095
FYP

DAC Analog Output

Maßstab 46.84%	Firma	Zeichner	Blatt
Änderung 26.10.2024 16:48		Titel	
Ausgabe 27.10.2024 18:55			
Datei STM32_FYP_DAC_Module.T3000		Projekt	