Voltage detection

This tutorial demonstrates: obtaining the real-time voltage of the battery pack through **ADC (ADC3_IN5)**, and printing the battery voltage information through **serial port (UASRT1)**.

1. Software-Hardware

- **STM32F103CubeIDE**
- **STM32 Robot Development Board**

  ADC, USART: chip internal peripherals

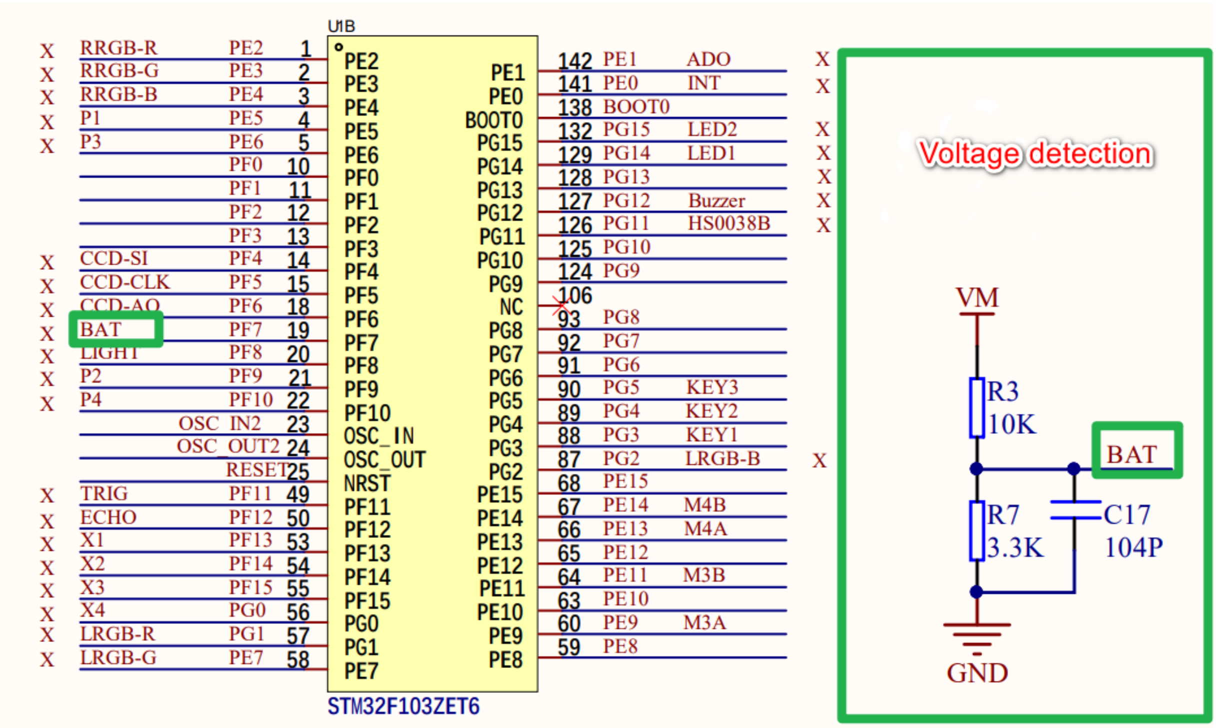
  7.4V battery pack: external

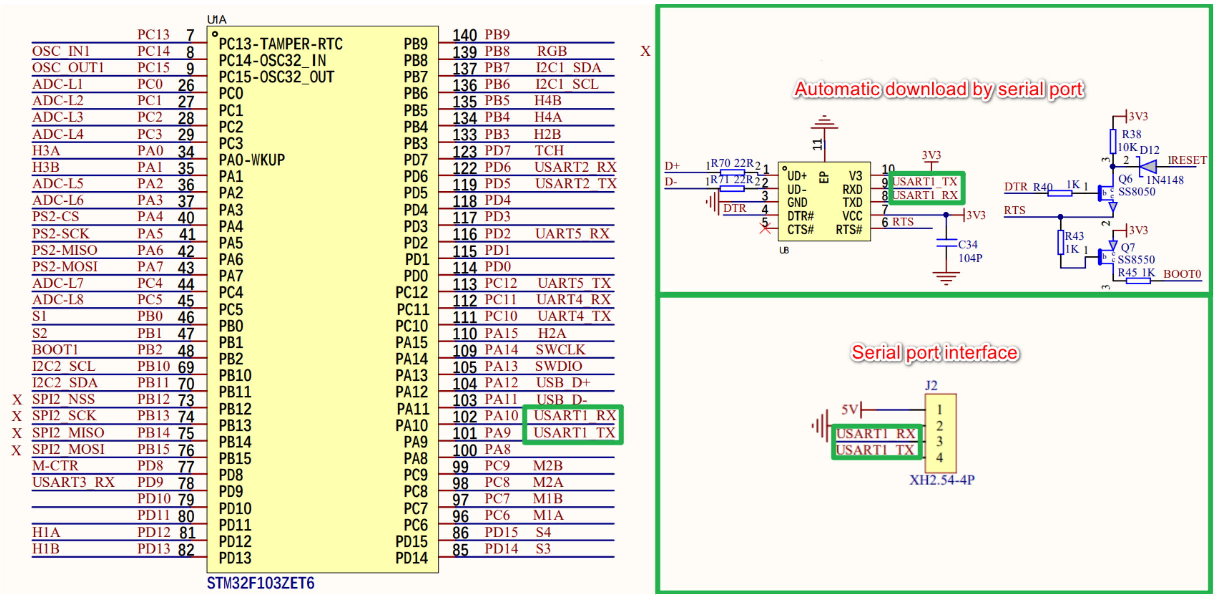
- **Type-C data cable or ST-Link**

  Download programs or simulate the development board

2. Brief principle

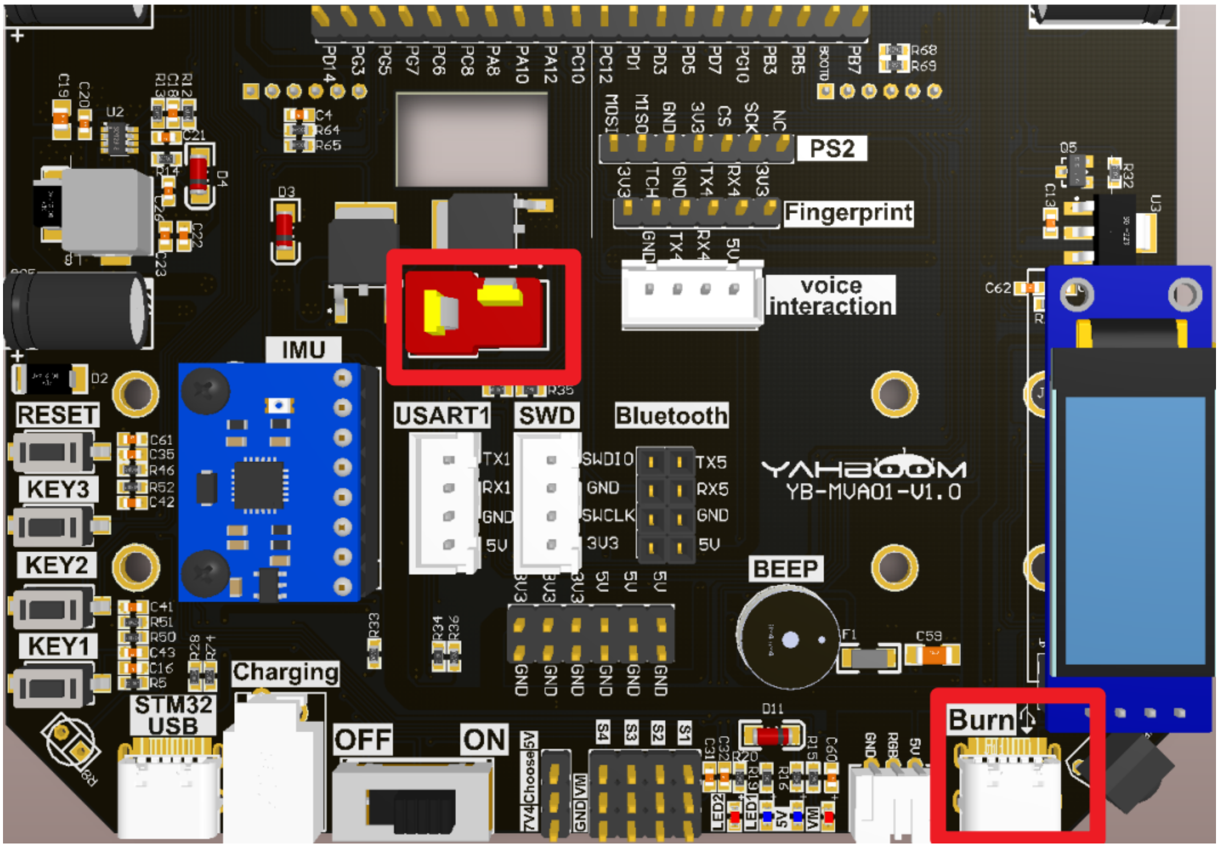
1. Hardware schematic diagram

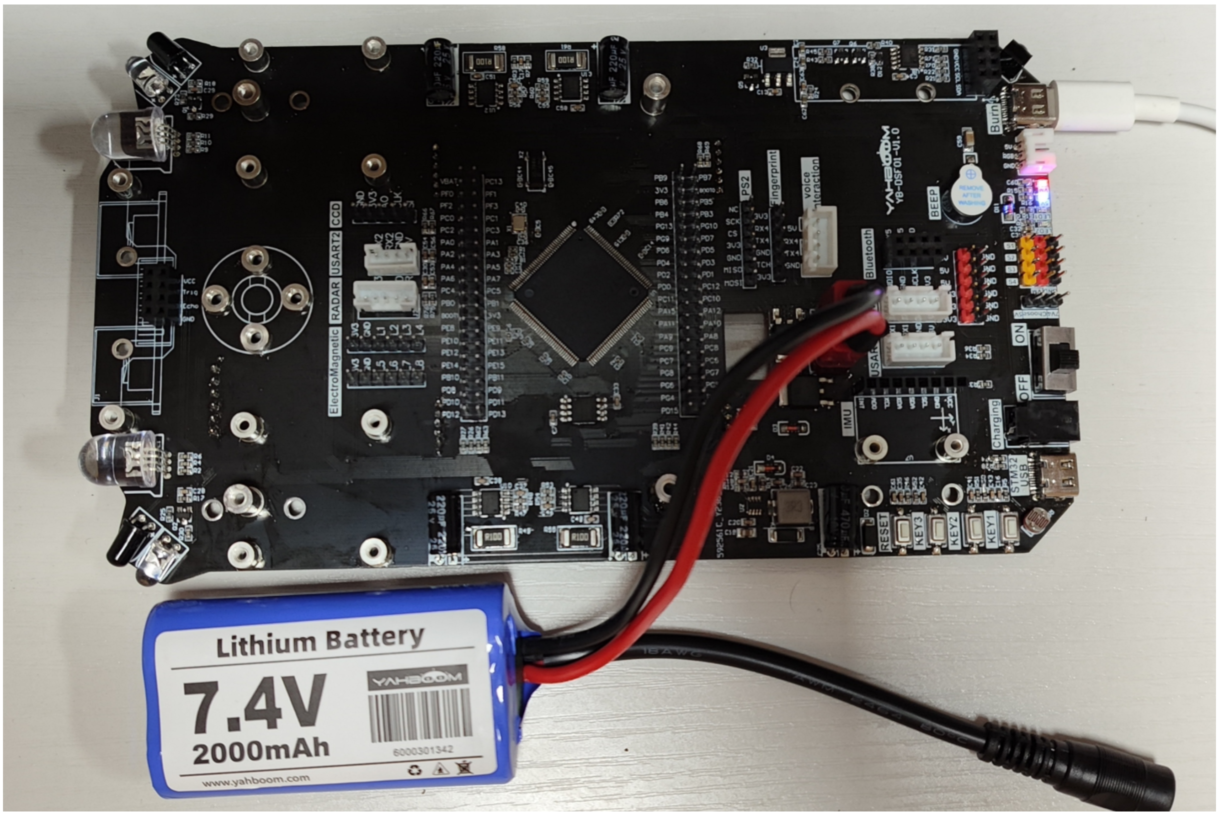




2. Physical connection diagram

When conducting this experiment, an external 7.4V battery pack is required.





3. Control principle

Obtain the converted value of the battery BAT pin through a single ADC, convert the value into actual voltage information and print it out through the serial port.

- **ADC**

STM32F103ZET6 has three built-in 12-bit analog-to-digital converters. Each ADC controller has up to 18 channels and can measure 16 external and 2 internal signal sources.

- **ADC Channel**

  Only the ADC channels used in this tutorial are selected here.

Foot position						Name of discipline	Types	I/O levels	Main function (after reset)	Optional reuse functionality	
BGA144	BGA100	WLCSP64	LQFP64	LQFP100	LQFP144					Reuse by default	Redefining functionality
F2	-	-	-	-	19	PF7	I/O		PF7	ADC3_IN5/FSMC_NREG	

- **ADC conversion value**

The ADC of STM32F103ZET6 is a 12-bit successive approximation analog-to-digital converter with 12-bit resolution;

$$ADC\ value\ range = 0 - 2^{12} = 0 - 4095$$

The ADC converted value can be stored in the 16-bit data register in a left-justified or right-justified manner.

- **ADC Conversion Mode**

A/D conversion of each channel of the ADC can be performed in single, continuous, scan or discontinuous modes.

model	function
Single conversion mode	ADC only performs one conversion

model	function
continuous conversion mode	As soon as the previous ADC conversion is completed, another conversion is started.
Scan mode	Used to scan a group of ADC channels
Intermittent mode	Group conversion of multiple ADC channels until the entire sequence is converted

```
Rule channel: perform channel conversion in a certain order
Injection channel: The injection channel can interrupt the conversion of the regular channel to execute the injection channel. After the injection channel is completed, the regular channel conversion can be continued.
```

- **ADC conversion time**

The input clock of the ADC must not exceed 14MHz, which is generated by dividing PCLK2.

$$Total\ conversion\ time = T_{CONV} = sampling\ time + 12.5\ clock\ cycles$$

Conversion time of this tutorial: ADC clock frequency 12MHz, sampling time 239.5 clock cycles

$$Total\ conversion\ time = T_{CONV} = sampling\ time + 12.5 clock\ cycles = (239.5 + 12.5) * \frac{1}{12000000} = 21us$$

- **Actual voltage conversion**: ADC internal reference voltage 3.3V

$$V_{BAT} = \frac{Value_{ADC\ converted\ value} * (3.3)}{4096}$$

Referring to the hardware schematic diagram, we can see: **Principle of equal current**

$$I_{Current} = \frac{V_{BAT}}{R7} = \frac{V_M}{R3 + R7} \Longrightarrow \frac{V_{BAT}}{3.3} = \frac{V_M}{10 + 3.3} \Longrightarrow V_M = \frac{V_{BAT} * (10 + 3.3)}{3.3}$$

$$Thatis, actualvoltage = V_M = \frac{V_{BAT} * (10 + 3.3)}{3.3}$$

3. Project configuration

Project configuration: Prompt configuration options during STM32CubeIDE project configuration

1. Description

Omitted project configuration part: **New project, chip selection, project configuration, SYS of pin configuration, RCC configuration, clock configuration and project configuration** content

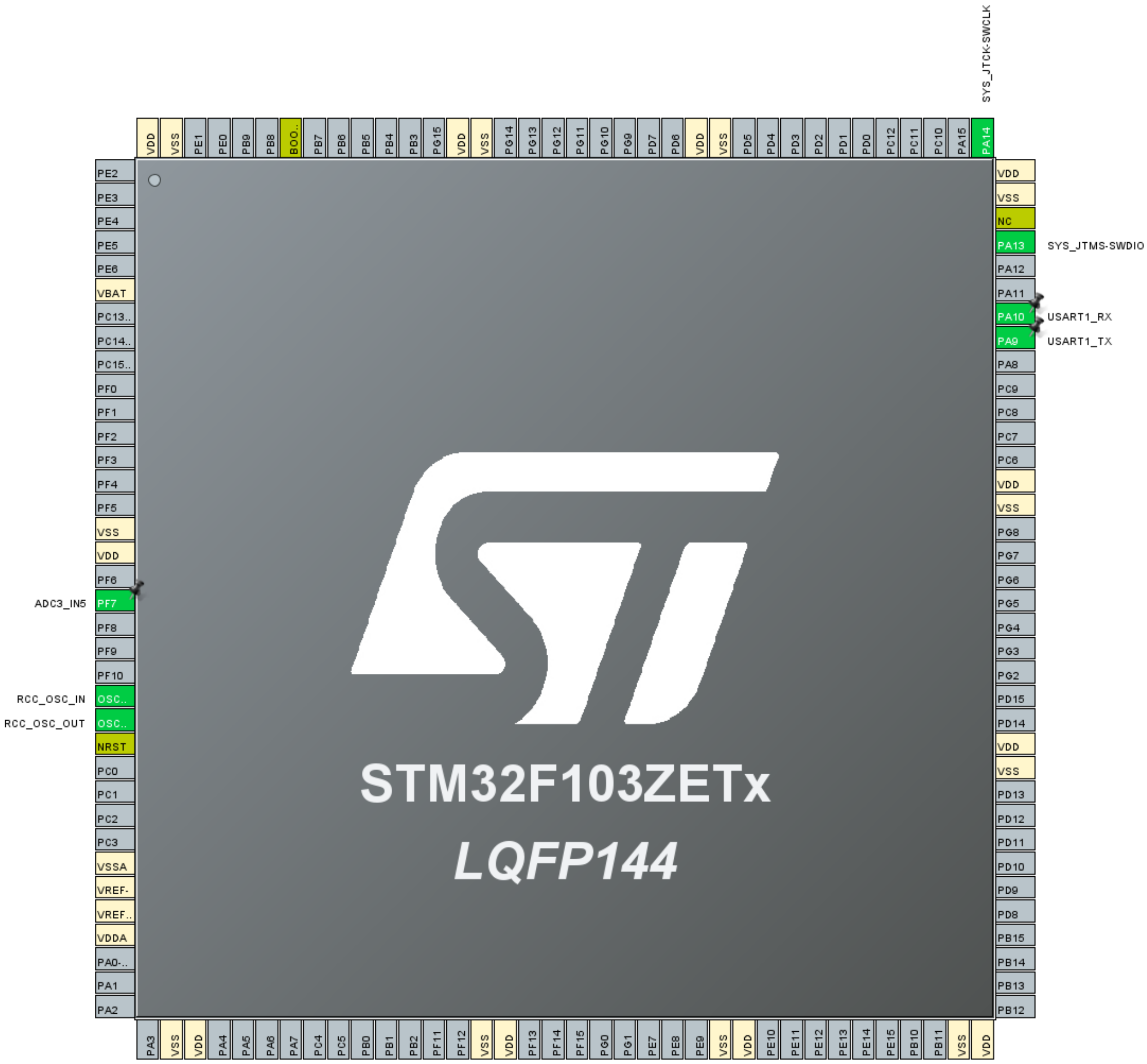
The project configuration part that is not omitted is the key point that needs to be configured in this tutorial.

```
Please refer to [2. Development environment construction and use: STM32CubeIDE installation and use] to understand how to configure the omitted parts of the project.
```

2. Pin configuration

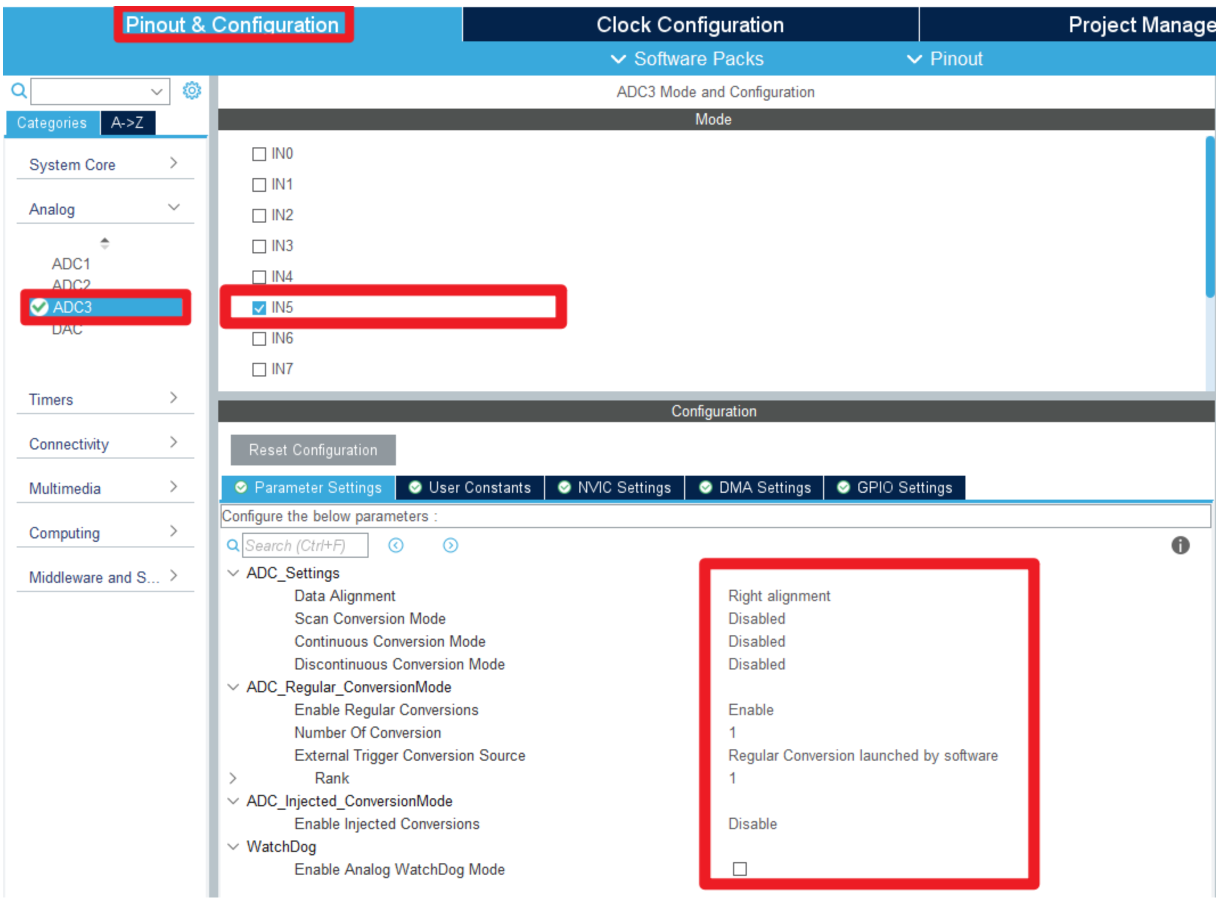
- **Configure specified pin function**

You can directly select the corresponding pin number in the pin view, and the corresponding options will appear when you left-click the mouse.



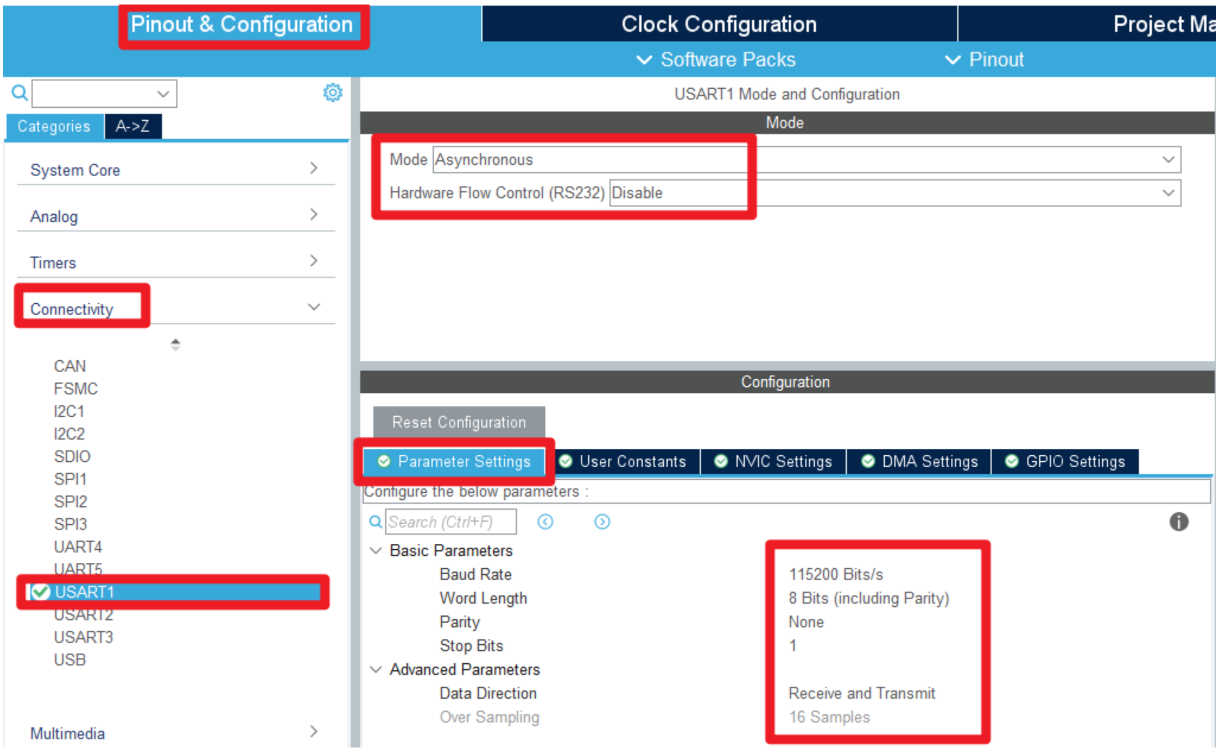
SYS_JTCK-SWCLK
SYS_JTMS-SWDIO
USART1_RX
USART1_TX
ADC3_IN5
RCC_OSC_IN
RCC_OSC_OUT
STM32F103ZETx
LQFP144

- **ADC**

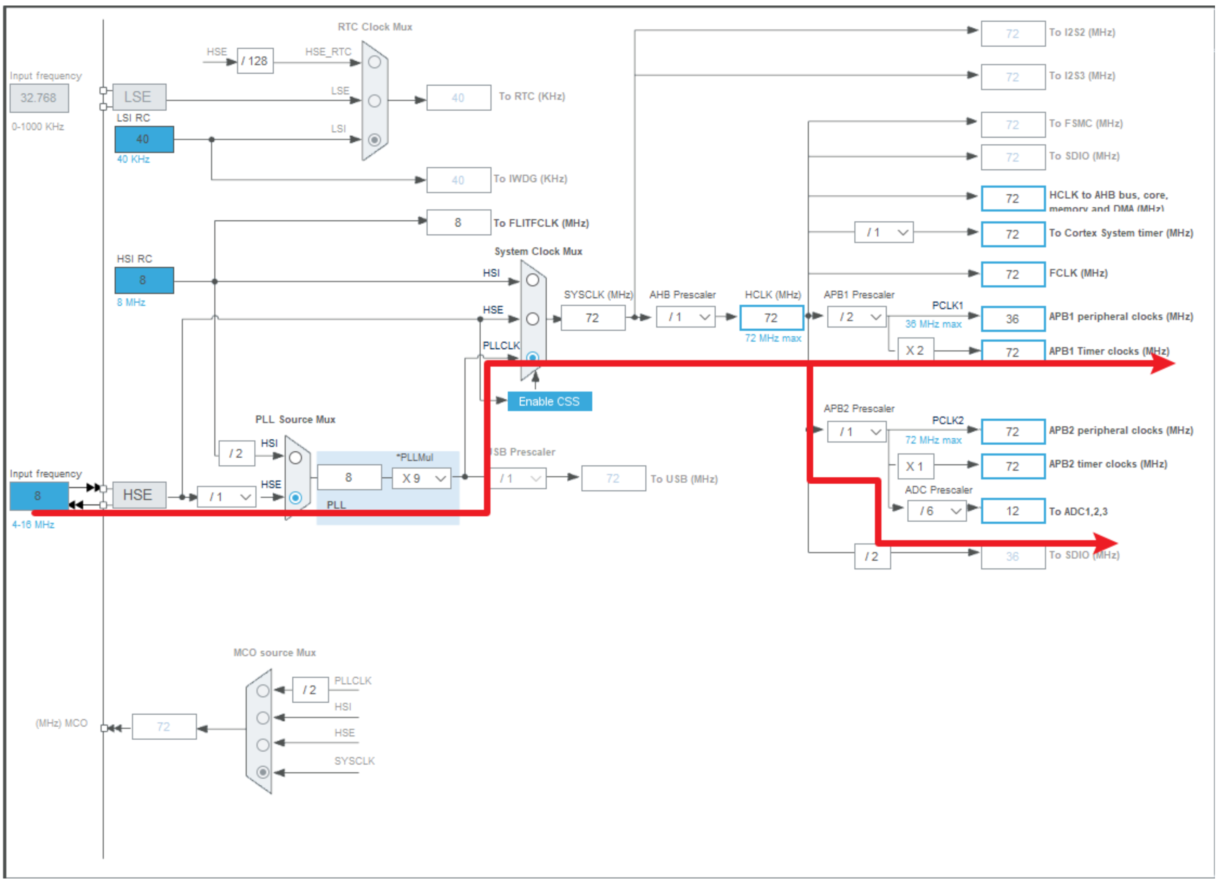


Pinout & Configuration
Clock Configuration
Project Manage
Software Packs
Pinout
ADC3 Mode and Configuration
Mode
Categories
A->Z
System Core
Analog
ADC1
ADC2
ADC3
DAC
Timers
Connectivity
Multimedia
Computing
Middleware and S...
IN0
IN1
IN2
IN3
IN4
IN5
IN6
IN7
Configuration
Reset Configuration
Parameter Settings
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Configure the below parameters :
Search (Ctrl+F)
ADC_Settings
Data Alignment
Right alignment
Scan Conversion Mode
Disabled
Continuous Conversion Mode
Disabled
Discontinuous Conversion Mode
Disabled
ADC_Regular_ConversionMode
Enable Regular Conversions
Enable
Number Of Conversion
1
External Trigger Conversion Source
Regular Conversion launched by software
Rank
1
ADC_Injected_ConversionMode
Enable Injected Conversions
Disable
WatchDog
Enable Analog WatchDog Mode

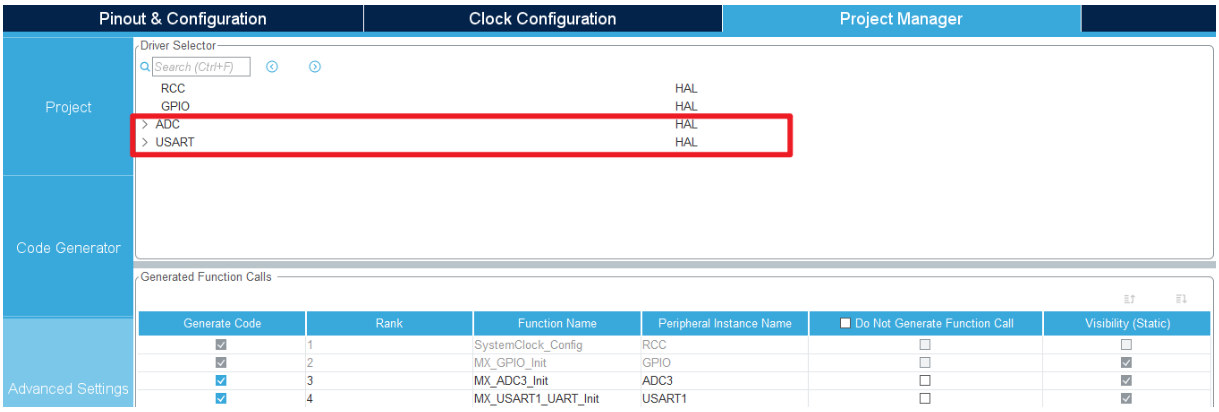
- **USART**



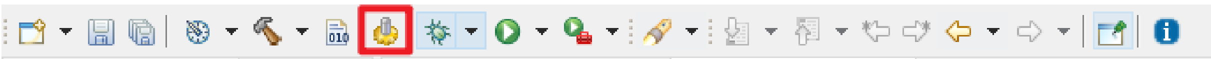
- **Clock Configuration**



- **Advanced Settings**



- **Generate code**



4. Main functions

It mainly introduces the functional code written by the user. For detailed code, you can open the project file provided by us yourself and enter the Bsp folder to view the source code.

1. User function

> **Function: Redirect print**

```
#include "stdio.h"

#ifdef __GNUC__
  #define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
  #define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif

PUTCHAR_PROTOTYPE {
  HAL_UART_Transmit(&USART1, (uint8_t *)&ch, 1, 0xFFFF);
  return ch;
}
```

> **Function: Get_BAT**

Function prototype	uint16_t Get_BAT(uint32_t ch)
Function description	Obtain battery ADC conversion raw data
Input parameters	**ch**: channel
Return value	ADC converted value

> **Function: Adc_Get_Average**

Function prototype	uint16_t Adc_Get_Average(uint32_t ch, uint8_t times)
Function description	Average value of multiple ADC conversion values
Input parameter 1	**ch**: channel
Input parameter 2	**times**: average times
Return value	ADC converted value

> **Function: Adc_Get_Measure_Volotage**

Function prototype	float Adc_Get_Measure_Volotage(void)
Function description	Get original voltage value
Input parameters	None
Return value	Original voltage value

> **Function: Adc_Get_Battery_Volotage**

Function prototype	float Adc_Get_Battery_Volotage(void)
Function description	Obtain the actual battery voltage before voltage division
Input parameters	None
Return value	Actual battery voltage

2. LL library function analysis

The HAL library functions that have been introduced in the previous tutorial will not be introduced again in the tutorial!

```
If you want to find the HAL library and LL library function analysis involved in the
entire tutorial, you can view the documents in the folder [8. STM32 Manual: STM32F1_HAL
Library and LL Library_User Manual]
```

> **Function: HAL_ADC_Init**

Function prototype	HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc)
Function description	Initialize ADC parameters
Input parameters	**hadc**: ADC handle address
Return value	**HAL status value**: HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT

> **Function: HAL_ADC_MspInit**

Function prototype	void HAL_ADC_MspInit(ADC_HandleTypeDef* hadc)
Function description	**Initialize the peripheral clock, GPIO and NVIC of the ADC**
Input parameters	**hadc**: ADC handle address
Return value	None

> **Function: HAL_ADC_MspDeInit**

Function prototype	void HAL_ADC_MspDeInit(ADC_HandleTypeDef* hadc)
Function description	**Cancel the initialization of ADC peripheral clock, GPIO and NVIC**
Input parameters	**hadc**: ADC handle address
Return value	None

Function: HAL_ADC_ConfigChannel

Function prototype	HAL_StatusTypeDef HAL_ADC_ConfigChannel (ADC_HandleTypeDef* hadc, ADC_ChannelConfTypeDef* sConfig)
Function description	Configure ADC specific channels
Input parameter 1	**hadc**: ADC handle address
Input parameter 2	**sConfig**: Configure ADC channel parameters: conversion channel, conversion sequence, sampling period
Return value	**HAL status value**: HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT

Function: HAL_ADC_Start

Function prototype	HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc)
Function description	Enable ADC rule group conversion
Input parameters	**hadc**: ADC handle address
Return value	**HAL status value**: HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT

Function: HAL_ADC_PollForConversion

Function prototype	HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout)
Function description	Wait for ADC rule group conversion to complete
Input parameter 1	**hadc**: ADC handle address
Input parameter 2	**Timeout**: timeout time
Return value	**HAL status value**: HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT

Function: HAL_ADC_GetValue

Function prototype	uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)
Function description	Get ADC conversion value
Input parameters	**hadc**: ADC handle address
Return value	ADC conversion value

Function: HAL_UART_Init

Function prototype	HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart)
Function description	**Initialize serial port parameters**
Input parameters	**huart**: serial port handle address

Function prototype	HAL_StatusTypeDef HAL_UART_Init(UART_HandleTypeDef *huart)
Return value	**HAL status value**: HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT

> **Function: HAL_UART_MspInit**

Function prototype	void HAL_UART_MspInit(UART_HandleTypeDef *huart)
Function description	**Initialize the peripheral clock, GPIO and NVIC of the serial port**
Input parameters	**huart**: serial port handle address
Return value	None

> **Function: HAL_UART_MspDeInit**

Function prototype	void HAL_UART_MspInit(UART_HandleTypeDef *huart)
Function description	**Cancel the initialization of serial port peripheral clock, GPIO and NVIC**
Input parameters	**huart**: serial port handle address
Return value	None

> **Function: HAL_UART_Transmit**

Function prototype	HAL_StatusTypeDef HAL_UART_Transmit (UART_HandleTypeDef *huart, const uint8_t *pData, uint16_t Size, uint32_t Timeout)
Function description	Send data in **polling mode**
Input parameter 1	**huart**: serial port handle address
Input parameter 2	**pData**: The first address of the data buffer to be sent
Input parameter 3	**Size**: Number of data bytes sent
Input parameter 4	**Timeout**: timeout time, in milliseconds
Return value	**HAL status value**: HAL_OK, HAL_ERROR, HAL_BUSY, HAL_TIMEOUT

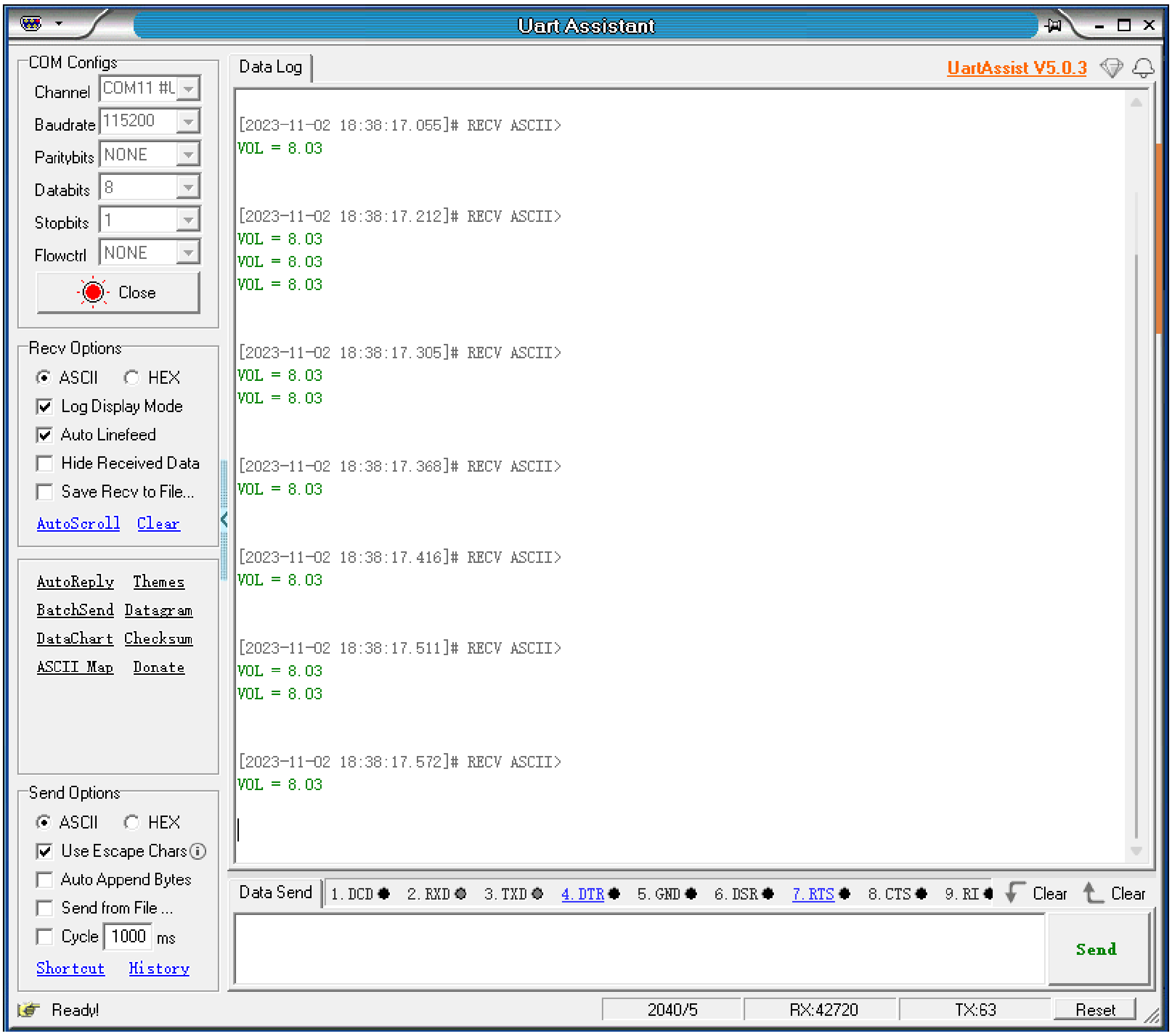
5. Experimental phenomena

After successfully downloading the program, press the RESET button on the development board to open the serial port debugging assistant to observe the phenomenon!

```
For program download, please refer to [2. Development environment construction and use: program download and simulation]
```

Phenomenon:

The serial port debugging assistant will print the real-time voltage information of the battery pack (the maximum voltage of the 7.4V battery pack can reach 8.4V).


Uart Assistant
COM Configs
Channel COM11 #L
Baudrate 115200
Paritybits NONE
Databits 8
Stopbits 1
Flowctrl NONE
Close
Recv Options
ASCII HEX
Log Display Mode
Auto Linefeed
Hide Received Data
Save Recv to File...
AutoScroll Clear
AutoReply Themes
BatchSend Datagram
DataChart Checksum
ASCII Map Donate
Send Options
ASCII HEX
Use Escape Chars
Auto Append Bytes
Send from File ...
Cycle 1000 ms
Shortcut History
Data Log
UartAssist V5.0.3
[2023-11-02 18:38:17.055]# RECV ASCII>
VOL = 8.03
[2023-11-02 18:38:17.212]# RECV ASCII>
VOL = 8.03
VOL = 8.03
VOL = 8.03
[2023-11-02 18:38:17.305]# RECV ASCII>
VOL = 8.03
VOL = 8.03
[2023-11-02 18:38:17.368]# RECV ASCII>
VOL = 8.03
[2023-11-02 18:38:17.416]# RECV ASCII>
VOL = 8.03
[2023-11-02 18:38:17.511]# RECV ASCII>
VOL = 8.03
VOL = 8.03
[2023-11-02 18:38:17.572]# RECV ASCII>
VOL = 8.03
Data Send
1.DCD 2.RXD 3.TXD 4.DTR 5.GND 6.DSR 7.RTS 8.CTS 9.RI
Clear Clear
Send
Ready!
2040/5
RX:42720
TX:63
Reset