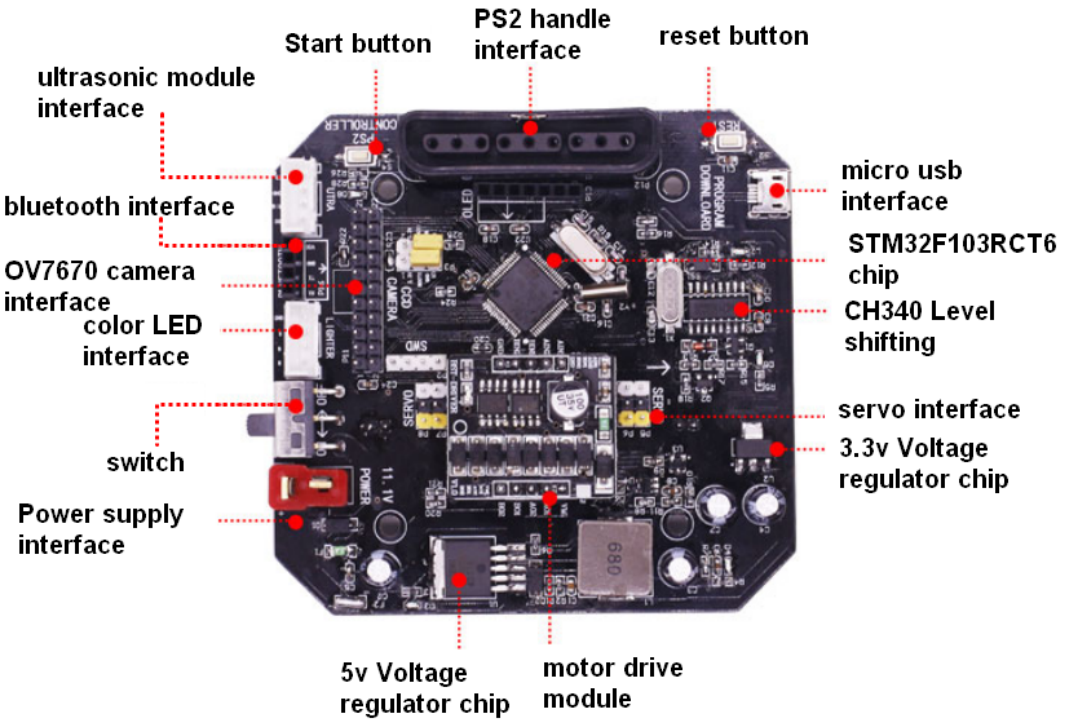
5.STM32 platform-------Battery voltage detection

1) Preparation



1-1 STM32 expansion board

1-2 STM32 smart car

2)Purpose of Experimental

When the car is powered on, press the start button next to the PS2 mark, you can see that battery voltage of car on the OLED.

3)Principle of experimental

The analog port of the STM32 can measure 0-3.3V voltage and return the value of 0-4065.

According to the principle of equal current: Vm/（R20+R21）= Vad/R21
According to STM32: Vad = (AD/4096)*3.3;
The following formula is obtained by the above two formulas:
Voltage = (Voltage / 4096) * 3.3 *（R20 + R21）/ R21

Let's talk about how to use the STM32 ADC.

The STM32 possess 1 to 3 ADCs (the STM32F101/102 series has only 1 ADC), and these ADCs can be used independently or in dual mode (increasing the sample rate). The STM32's ADC is a 12-bit successive approximation analog-to-digital converter. It has 18 channels for measuring 16 external and 2 internal sources. The A/D conversion of each channel can be performed in single, continuous, sweep or discontinuous mode. The results of the ADC can be stored in a 16-bit data register in left or right alignment. The analog watchdog feature allows the application to detect if the input voltage exceeds a user-defined high/low threshold.

The STM32F103RCT6 we chose includes three 12-bit ADCs. The maximum conversion rate of the STM32 ADC is 1Mhz, that is, the conversion time is 1us (this value is obtained at ADCCLK=14M and the sampling period is 1.5 ADC clocks). Do not let the ADC clock exceed 14M, otherwise the result will be degraded. .

The STM32 has a rule channel group of up to 16 conversions for its ADC and a maximum of 4 channels for the injection channel group. This chapter only describes how to use the single conversion mode of the rule channel.

The STM32's ADC performs only one conversion in single conversion mode, which can be initiated by the ADON bit of the ADC_CR2 register (for regular channels only) or by an external trigger (for regular and injection channels). The CONT bit is 0.

Taking the regular channel as an example, once the selected channel conversion is completed, the conversion result will be stored in the ADC_DR register, the EOC (end of conversion) flag will be set, and if EOCIE is set, an interrupt will be generated. The ADC will then stop until the next time it is started.

Next, let's take a look at the ADC registers we need to perform a single conversion of the regular channel. We start the process of using the ADC in the library function. Below we introduce the function of the library function to set the channel 1 using ADC1 for AD conversion. The library functions we use here are distributed in the stm32f10x_adc.c file and the stm32f10x_adc.h file.

The setup steps are as follows：
1）Turn on the PC port clock, ADC1 clock and set PC4 as an analog input. The ADC channel 14 of the STM32F103RCT6 is on PC4, so we first enable the PORTC clock and the ADC1 clock, then set PC14 to the analog input. use the RCC_APB2PeriphClockCmd

function to enable GPIOC and ADC clocks, using the GPIO_Init function to set the input mode of PC4.

Here we list the STM32 ADC channel and GPIO correspondence table:

	ADC1	ADC2	ADC3
channel 0	PA0	PA0	PA0
channel 1	PA1	PA1	PA1
channel 2	PA2	PA2	PA2
channel 3	PA3	PA3	PA3
channel 4	PA4	PA4	PF6
channel 5	PA5	PA5	PF7
channel 6	PA6	PA6	PF8
channel 7	PA7	PA7	PF9
channel 8	PB0	PB0	PF10
channel 9	PB1	PB1	
channel 10	PC0	PC0	PC0
channel 11	PC1	PC1	PC1
channel 12	PC2	PC2	PC2
channel 13	PC3	PC3	PC3
channel 14	PC4	PC4	
channel 15	PC5	PC5	
channel 16		Temperature sensor	
channel 17			

Figure 22.1.20 ADC channel and GPIO correspondence table

2）Reset ADC1 and set the ADC1 division factor.

After turning on the ADC1 clock, we will reset ADC1。When we reset all the registers of ADC1 to their default values. Then we can set the division factor of ADC1 through RCC_CFGR. The division factor should ensure that the ADC1 clock (ADCCLK) does not exceed 14Mhz. We set the division factor to 6, the clock is 72/6=12MHz, and the library function is implemented as follows:

RCC_ADCCLKConfig(RCC_PCLK2_Div6);

The method of resetting the ADC clock is:

ADC_DeInit(ADC1);

3) Initialize the ADC1 parameters, set the operating mode of ADC1 and information about the rule sequence.

After setting the division factor, we can start the mode configuration of ADC1. Setting the single conversion mode, trigger mode selection, data alignment, etc. are realized in this step. At the same time, we also need to set the relevant information of the ADC1 rule sequence. We only have one channel here, and it is a single conversion, so we need to set the number of channels in the rule sequence to 1.

void ADC_Init(ADC_TypeDef* ADCx, ADC_InitTypeDef* ADC_InitStruct)；

As you can see from the function definition, the first parameter is the specified ADC number. We take a look at the second parameter, just like other peripheral initialization, and also set the parameters by setting the value of the structure member variable.

```
typedef struct
{
uint32_t ADC_Mode;
FunctionalState ADC_ScanConvMode;
FunctionalState ADC_ContinuousConvMode;
uint32_t ADC_ExternalTrigConv;
uint32_t ADC_DataAlign;
uint8_t ADC_NbrOfChannel;
}ADC_InitTypeDef;
```

The parameter **ADC_Mode** is named to be the mode used to set the ADC.
There are many ADC modes, including independent mode, injection synchronous mode, etc. Here we choose independent mode, so the parameter is ADC_Mode_Independent.
The parameter **ADC_ScanConvMode** is used to set whether to enable the scan mode, because it is a single conversion, here we choose not to open the value DISABLE.
The parameter **ADC_ContinuousConvMode** is used to set whether to enable continuous conversion mode. Because it is a single conversion mode, we choose not to enable continuous conversion mode, DISABLE.
The parameter **ADC_ExternalTrigConv** is used to set the external event of the start rule conversion group conversion. Here we select the software trigger and select the value ADC_ExternalTrigConv_None.
The parameter **DataAlign** is used to set whether the ADC data alignment is left or right. Here we select the right alignment ADC_DataAlign_Right.
The parameter **ADC_NbrOfChannel** is used to set the length of the rule sequence. Here we are a single conversion, so the value is 1.

Let's take a look at our initialization example by following the explanation of each parameter:

```
ADC_InitTypeDef ADC_InitStructure;
ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
ADC_InitStructure.ADC_ScanConvMode = DISABLE;
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStructure.ADC_NbrOfChannel = 14;
ADC_Init(ADC1, &ADC_InitStructure);
```

4)Enable the ADC and calibrate
After setting up the above information, we enable the AD converter, perform reset calibration and AD calibration.
Note:These two steps are necessary! Failure to calibrate will result in very inaccurate results.
The way to enable the specified ADC is:

```
ADC_Cmd(ADC1, ENABLE);
```

The method to perform reset calibration is:

```
ADC_ResetCalibration(ADC1);
```

The way to perform ADC calibration is:

ADC_StartCalibration(ADC1);

We need to wait for the end of calibration after each calibration. We determine whether the calibration is over by obtaining the calibration status. Below we list the wait and end methods for reset calibration and AD calibration:

While(ADC_GetResetCalibrationStatus(ADC1));

While(ADC_GetCalibrationStatus(ADC1));

5）Read the ADC value.

Next, we need to do is set the channel in rule sequence 1, the sampling order, and the sampling period of the channel. Then start the ADC conversion, after the conversion is completed, the ADC conversion result value is read. The function of setting the rule sequence channel and the sampling period is:

void ADC_RegularChannelConfig(ADC_TypeDef* ADCx, uint8_t ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)；

Here is the first conversion in the rule sequence, and the sampling period is 239.5, so set it to:

ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5);

Method of open ADC conversion:

ADC_SoftwareStartConvCmd(ADC1, ENABLE);

ADC_GetConversionValue(ADC1);

In the AD conversion, we also need to get the status information of the AD conversion according to the flag of the status register.

FlagStatus ADC_GetFlagStatus(ADC_TypeDef* ADCx, uint8_t ADC_FLAG)

For example, we want to determine whether the conversion of ADC1d is over.

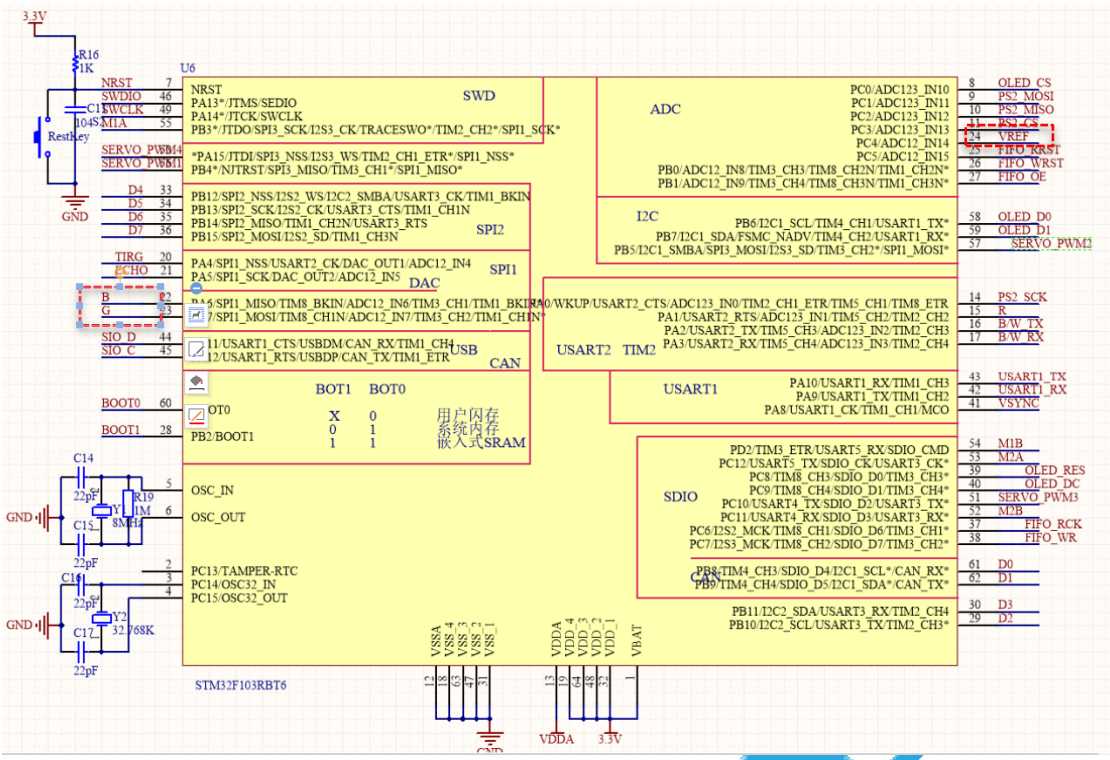
while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC));

In this chapter our reference voltage is set to 3.3V.

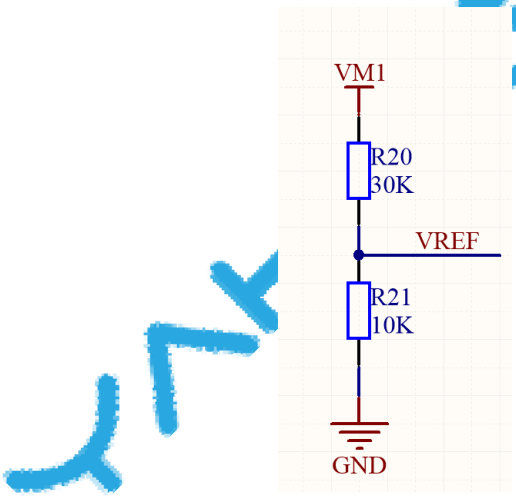
We can use the STM32 ADC1 to perform AD conversion operations by the above steps

2) Experimental Steps

4-1 About the schematic



4-1 STM32 main control board circuit diagram



4-2 Battery voltage detection circuit diagram

4-2 According to the circuit schematic:

VREF----PC4(STM32)

4-3 About the code

Please see the folder named Battery voltage detection in the code folder.