Lesson 4 ADC Detects Voltage and Realizes Low-voltage Alarm

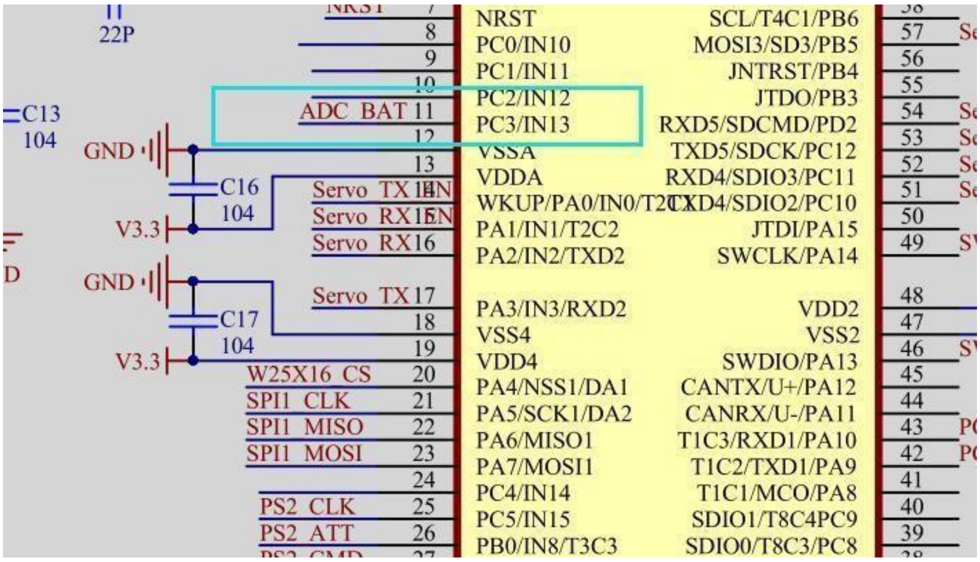
1. Project Purpose

Use ADC to examine the battery voltage and realize the buzzer to make low-voltage alarm.

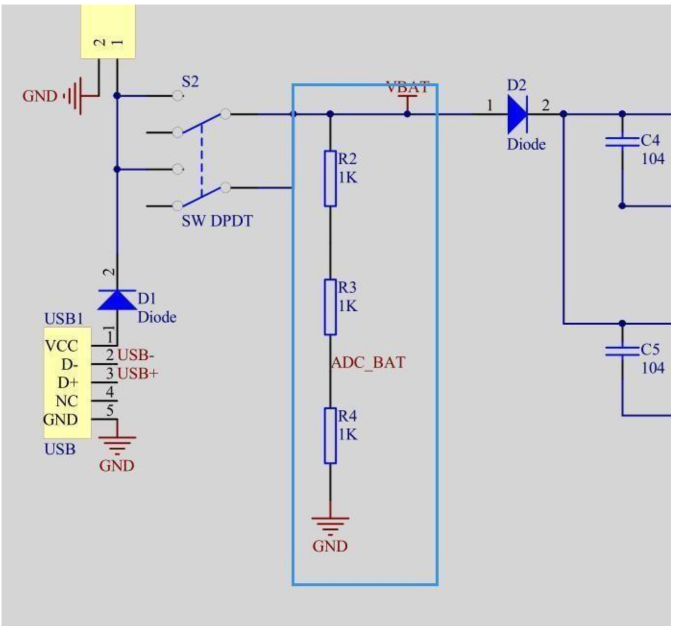
2. Project Principle

ADC (A/D converter) is short for analog-digital converter. In microcontroller application system, the input analog voltage signal is often converted into the digital signal that can be recognized by microcontroller, and the technology converting the continuously changing analogue signal into digital signal is called A/D conversion technology.

In practice, A/D can be connected between the input signal and the microcontroller to complete A/D conversion, or you can also choose to use a microcontroller with built-in A/D converter. Our controller has built-in A/D converter. When the analog signal is imported into the controller, it can be converted into the digital signal, and then process with the numerical analysis to calculate the voltage value.



By checking the schematic diagram, we can know that the battery voltage is separated into PC3 pin of STM32 through three 1k resistant so that the voltage detected by ACD is required to multiple 3 and then get the real battery voltage.



3. Program Analyst

1) ADC is an independent peripheral, and its clock needs to be turned on before using ADC, and it can be turned off when it is idle to reduce power consumption.
2) The yellow box in the figure below is the clock code for configuring the ADC and the red box is the configuration of I/O port to be collected by the ADC. The collected battery voltage is connected to the PC3 I/O port of STM32 so we need to configure PC3 to analog input mode to perform analog-to-digital conversion.
3) The green box is the parameter configuration of ADC. In STM32, ADC supports multiple working modes such as single conversion, continuous conversion, scan conversion. There we use one ADC and configure to single-channel single conversion. The start of the conversion is controlled by software. The black box is the code for calibrating ADC.

```c
void InitADC(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC |RCC_APB2Periph_ADC1    , ENABLE);     //Enable ADC1 channel clock

    RCC_ADCCLKConfig(RCC_PCLK2_Div6);   //72M/6=12, the largest time of ADC cannot more than 14M.
    //PA0/1/2/3 As an analog channel input pin
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;//|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;       //Analog input pin
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    ADC_DeInit(ADC1);  //Reset all registers of peripheral ADC1 to default values

    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;  //ADC work mode: ADC1 and ADC2 works under independent mode.
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;   //Analog-to-digital conversion works in single-channel mode
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //Analog-to-digital conversion works in single conversion mode
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None; //Conversion is started by software instead of the external trigger.
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;  //ADC data right align
    ADC_InitStructure.ADC_NbrOfChannel = 1; //Number of ADC channels for regular conversion in sequence
    ADC_Init(ADC1, &ADC_InitStructure); //Initial the registers of peripherals ADCx according to parameter specified in ADC_InitStruct


    ADC_Cmd(ADC1, ENABLE);  //Enable the specified ADC1

    ADC_ResetCalibration(ADC1); //Reset the calibration register of specified ADC1.

    while(ADC_GetResetCalibrationStatus(ADC1)); //Get the status of ADC1 reset calibration register, If it is setting status, wait.

    ADC_StartCalibration(ADC1);     //Start specifying the calibration status of ADC1

    while(ADC_GetCalibrationStatus(ADC1));      //Get the calibration program of the specified ADC1. If it is setting status, wait.

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);     //Enable software conversion start function of ADC1
}
```

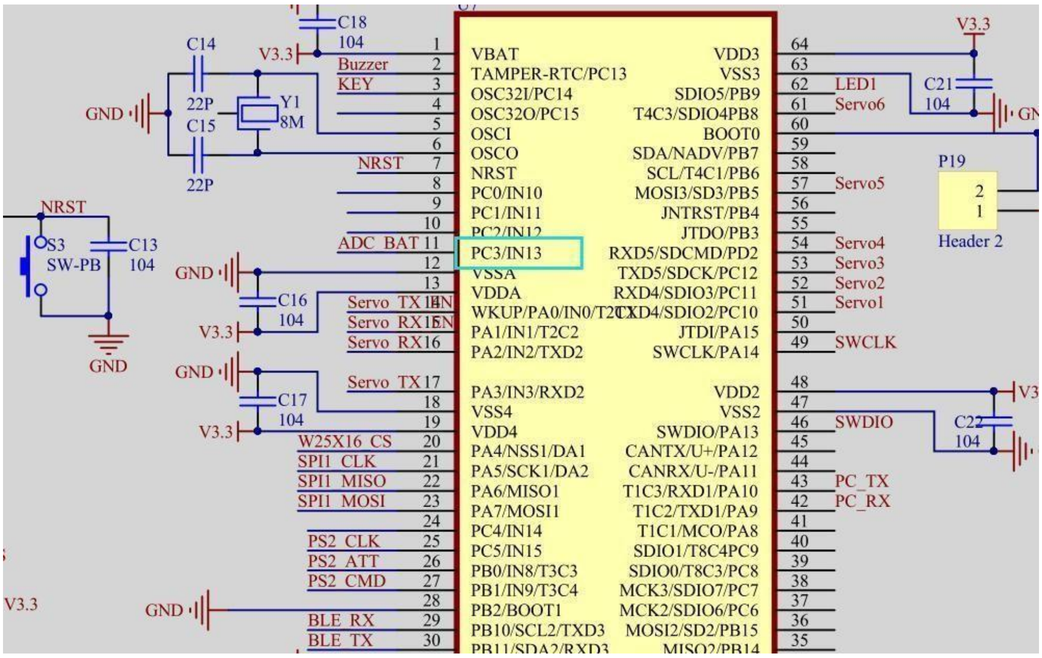
4) After configuring the ADC, we can call the function below to detect the battery voltage.

```c
void CheckBatteryVoltage(void)
{
    uint8 i;
    uint32 v = 0;
    for(i = 0;i < 8;i++)
    {
        v += GetADCResult(ADC_BAT);
    }
    v >>= 3;

    v = v * 2475 / 1024;//adc / 4096 * 3300 * 3(3 means that it is amplified 3 times, because the resistor divides the voltage when collecting the voltage)
    BatteryVoltage = v;

}
```

5) In the code shown in the figure above, the GetADCResult function is called 8 times continuously. GetADCReslt is the function for obtaining ADC sampling value. This function has a parameter which is the channel number used for conversing the channel. Through the manual or the schematic diagram, you can know that the ADC channel of the battery voltage sampling corresponding to the I/O port is channel 13.



6) The return value obtained is summed and then shifted to the right by 3 bits. Shifting 3 bits to the right is equivalent to dividing by 8, which means that the value is averaged 8 times and then the value is converted to the corresponding voltage.

7) The ADC of STM32 is 12 bits, that is, when the measured voltage is 3.3V, the sampling value of ADC is 4096. 4096/3300 = ADC / XmV so the voltage should be ADC / 4096 * 3300mV. Because the sampled voltage has been divided to one-third of the original, it needs to be multiplied by 3. The final battery voltage should be ADC sampling value/ 4096 * 3300 * 3.

8) Next, let’s look at the GetADCResult function. When the ADC conversion is completed, the converted value will be returned.

```
uint16 GetADCResult(BYTE ch)
{
    //Set the regular group channels of the specified ADC, set their conversion order and sampling time
    ADC_RegularChannelConfig(ADC1, ch, 1, ADC_SampleTime_239Cycles5);   //ADC1, ADC channel 3, the regular sampling sequence value is 1, the sampling time is 239.5 cycles

    ADC_SoftwareStartConvCmd(ADC1, ENABLE);    //Enable specified ADC1 software conversion start function

    while(!ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC)); //Wait for the conversion to end

    return ADC_GetConversionValue(ADC1);    //Return the latest conversion result of ADC1 rule group
}
```

9) In the 100us interrupt of timer 2, the code can be viewed, as shown in the blue box below. The function of GetBatteryVoltage is to obtain the battery voltage. If all the detected voltage values are less than 6.4 in 5s, the buzzer will be started to alarm.

```

            Buzzer();
            if(++time >= 10)
            {
                time = 0;
                gSystemTickCount++;
                Ps2TimeCount++;
                if(GetBatteryVoltage() < 5500)//Alarm when less than 5.5V
                {
                    timeBattery++;
                    if(timeBattery > 5000)//last 5 seconds
                        BuzzerState = 1;
                    }
                }
                else
                {
                    timeBattery = 0;
                    if(manual == FALSE)
                    {
                        BuzzerState = 0;
                    }
                }
            }
        }
    }
```

10) GetBatteryVoltage returns the value of global variables BatteryVoltage directly.

```
uint16 GetBatteryVoltage(void)
{//Voltage mV
    return BatteryVoltage;
}
```

11) CheckBatteryVoltage is called every 500ms in TaskTimerHandle so as to update the battery voltage data. TaskTimerHandle is called in each main loop in TaskRun.