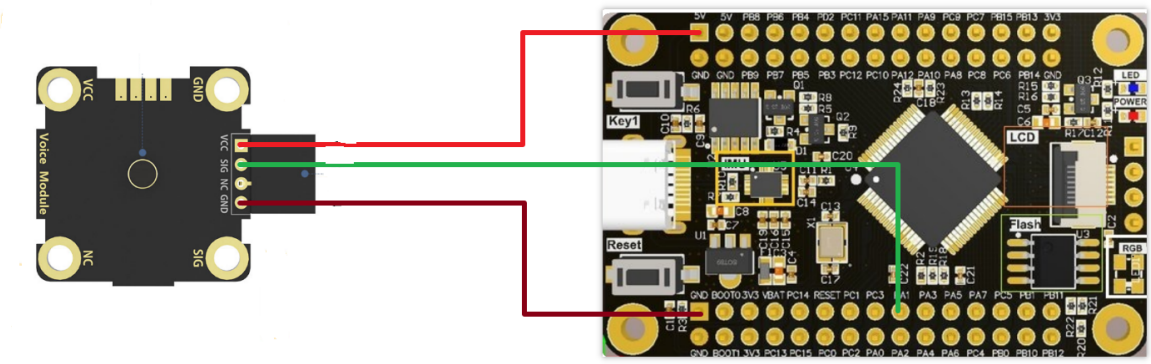
Sound module: ADC single-channel conversion

Hardware wiring



Sound module	STM32F103RCT6
VCC	5V/3.3V
SIG	PA1
NC	
GND	GND

Brief principle

The sound sensor module can detect the sound magnitude of the surrounding environment and its changes.

The microphone on the sound sensor module converts the audio signal into an electrical signal (analog).

Main code

main.c

```
#include "stm32f10x.h"
#include "SysTick.h"
#include "UART.h"
#include "ADC.h"

int main(void)
{
```

```
    uint16_t ADC_Result;
    float ADC_Value;

    SysTick_Init();//滴答定时器初始化
    UART1_Init();//UART1初始化
    ADC1_Init();//ADC1初始化

    printf("Ambient sound measurement!\n");//打印环境声音测量!

    while(1)
    {
        ADC_Result = ADC1_Result();//获取ADC转换数据
        printf("ADC_Result = %d\r\n",ADC_Result);//打印ADC转换数据

        //将声音大小转换成0-100等级
        ADC_Value = ADC_Result * 100 / 4095;//数据转换
        printf("ADC1_Value = %0.2f\r\n",ADC_Value);//打印音量对应等级
        Delay_us(100000);//延时100ms 时间越短 串口打印的数据更快，更实时
    }
}
```

SysTick.c

```
#include "SysTick.h"

unsigned int Delay_Num;

void SysTick_Init(void)//滴答定时器初始化
{
    while(SysTick_Config(72));//设置重装载值 72 对应延时函数为微秒级
    //若将重装载值设置为72000 对应延时函数为毫秒级
    SysTick->CTRL &= ~(1 << 0);//定时器初始化后关闭，使用再开启
}

void Delay_us(unsigned int NCount)//微秒级延时函数
{
    Delay_Num = NCount;
    SysTick->CTRL |= (1 << 0);//开启定时器
    while(Delay_Num);
    SysTick->CTRL &= ~(1 << 0);//定时器初始化后关闭，使用再开启
}

void SysTick_Handler(void)
{
    if(Delay_Num != 0)
    {
        Delay_Num--;
    }
}
```

SysTick.h

```
#ifndef __SYSTICK_H__
#define __SYSTICK_H__

#include "stm32f10x.h"

void SysTick_Init(void);//滴答定时器初始化
void Delay_us(unsigned int NCount);//微秒级延时函数

#endif
```

UART.c

```
#include "UART.h"

void UART1_Init(void)//UART1初始化
{
    USART_InitTypeDef USART_InitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;

    /* Enable GPIO clock */
    /* 使能GPIOA AFIO时钟 TXD(PA9) RXD(PA10) */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO, ENABLE);

    /* Enable USART1 Clock */
    /* 使能串口1 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 , ENABLE);

    /* Configure USART1 Rx as input floating */
    /* 配置RXD引脚 PA10 浮空输入模式 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure USART1 Tx as alternate function push-pull */
    /* 配置TXD引脚 PA9 复用推挽输出模式 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* USART1 configured as follow:
    - BaudRate = 115200 baud
    - Word Length = 8 Bits
    - One Stop Bit
    - No parity
    - Hardware flow control disabled (RTS and CTS signals)
    - Receive and transmit enabled
    */
```

```
    USART_InitStructure.USART_BaudRate = 115200;//波特率设置
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//8位数据位
    USART_InitStructure.USART_StopBits = USART_StopBits_1;//1位停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验位
    USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;//无需硬件流控
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;//全双工 即发送也接
受
    USART_Init(USART1, &USART_InitStructure);

    /* Enable USART1 Receive and Transmit interrupts */
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
    USART_ITConfig(USART1, USART_IT_TXE, ENABLE);

    /* Enable the USART1 */
    /* 使能USART1 */
    USART_Cmd(USART1, ENABLE);
}

void NVIC_UART1_Init(void)//UART1 NVIC配置
{
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Configure the NVIC Preemption Priority Bits */
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_0);

    /* Enable the USART1 Interrupt */
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);
}


void USART_SendString(USART_TypeDef* USARTx, char *pt)//给指定串口发送字符串
{
    while(*pt)
    {
        while(USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
        USART_SendData(USARTx,*pt);
        while(USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET);
        pt++;
    }
}

int fputc(int c, FILE *pt)//printf重定向
{
    USART_TypeDef* USARTx = USART1;
    while(USART_GetFlagStatus(USARTx, USART_FLAG_TXE) == RESET);
    USART_SendData(USARTx, c);
    while(USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET);
    return 0;
```

```
}

void USART1_IRQHandler(void)
{
    unsigned char ch;
    while(USART_GetFlagStatus(USART1, USART_FLAG_RXNE) == SET)//接收到数据
    {
        ch = USART_ReceiveData(USART1);
        printf("%c\n",ch);
    }
}
```

UART.h

```
#ifndef __UART_H__
#define __UART_H__

#include "stm32f10x.h"
#include "stdio.h"

void UART1_Init(void);//UART1初始化
void USART_SendString(USART_TypeDef* USARTx, char *pt);//给指定串口发送字符串
int fputc(int c, FILE *pt);//printf重定向
void NVIC_UART1_Init(void);//UART1 NVIC配置

#endif
```

ADC.c

```
#include "ADC.h"

void ADC1_Init(void)//ADC初始化
{
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;

    /* ADCCLK = PCLK2/6 */
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);

    /* Enable ADC1, and GPIOA clocks */
    /* 使能ADC1 GPIOA时钟 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1| RCC_APB2Periph_GPIOA, ENABLE);

    /* Configure PA1 (ADC Channel) as analog inputs */
    /* 配置ADC 通道1 PA1 模拟输入引脚 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* ADC1 configuration */
```

```
    /* ADC1 配置 */
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;//独立模式
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;//扫描模式
    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;//连续转换
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;//软件触发
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//数据右对齐
    ADC_InitStructure.ADC_NbrOfChannel = 1;//单通道
    ADC_Init(ADC1, &ADC_InitStructure);

    /* ADC1 regular channels configuration */
    /* ADC1 规则通道配置 */
    ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_55Cycles5);

    /* Enable ADC1 */
    /* 使能 ADC1 */
    ADC_Cmd(ADC1, ENABLE);

    /* Enable ADC1 reset calibration register */
    /* 使能ADC1复位校准寄存器 */
    ADC_ResetCalibration(ADC1);
    /* Check the end of ADC1 reset calibration register */
    /* 检查ADC1复位校准寄存器的末端 */
    while(ADC_GetResetCalibrationStatus(ADC1));

    /* Start ADC1 calibration */
    /* 启动 ADC1 校准 */
    ADC_StartCalibration(ADC1);
    /* Check the end of ADC1 calibration */
    /* 检查ADC1校准结束 */
    while(ADC_GetCalibrationStatus(ADC1));

    /* Start ADC1 Software Conversion */
    /* 启动 ADC1 软件转换 */
    ADC_SoftwareStartConvCmd(ADC1, ENABLE);
}

uint16_t ADC1_Result(void)//获取ADC转换数据
{
    while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
    return (ADC_GetConversionValue(ADC1));
}
```

ADC.h

```
#ifndef __ADC_H__
#define __ADC_H__

#include "stm32f10x.h"

void ADC1_Init(void);//ADC1初始化
uint16_t ADC1_Result(void);//获取ADC转换数据

#endif
```

Phenomenon

After downloading the program, press the Reset key once, and the downloaded program will run.

The serial debugging assistant will print the current environment volume every 0.1 seconds.

Note: If you find that the sound size is inconsistent with the data printed by the serial port, it may be that the delay function in the while loop is too long (there are comments in the code).