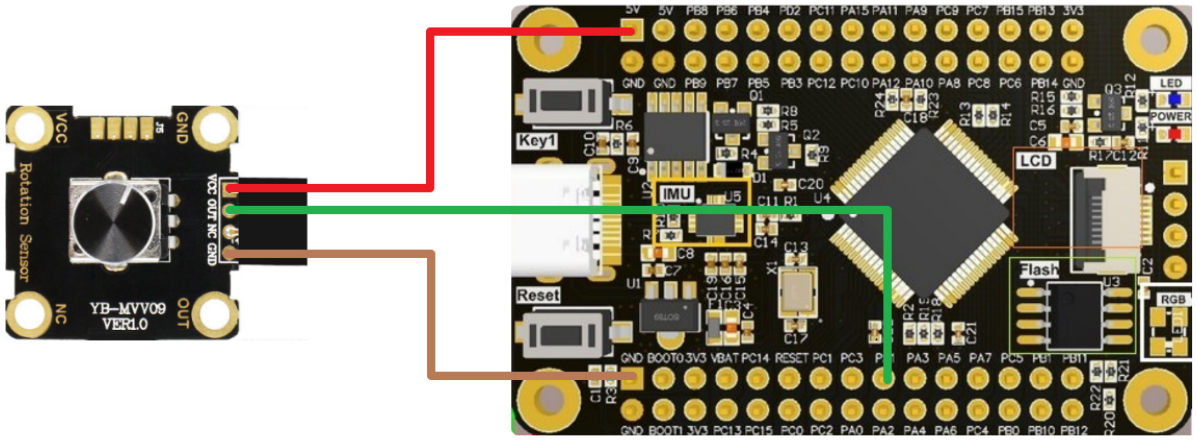
Potentiometer module: ADC single-channel conversion

Hardware wiring



Potentiometer module	STM32F103RCT6
VCC	5V/3.3V
OUT	PA1
NC	
GND	GND

Brief principle

Potentiometer (adjustable resistor) has three lead terminals and a resistance element whose resistance value can be adjusted according to a certain law of change;

It is usually composed of a resistor and a rotating or sliding system, that is, a moving contact moves on the resistor to obtain part of the voltage output;

The analog value range of the output of the OUT pin of this potentiometer is 0~4095.

Main code

main.c

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

int main(void)
```

```
{
    uint16_t ADC_Result;

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

    printf("Potentiometer analog value output!\n");

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

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;
}
```

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

#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

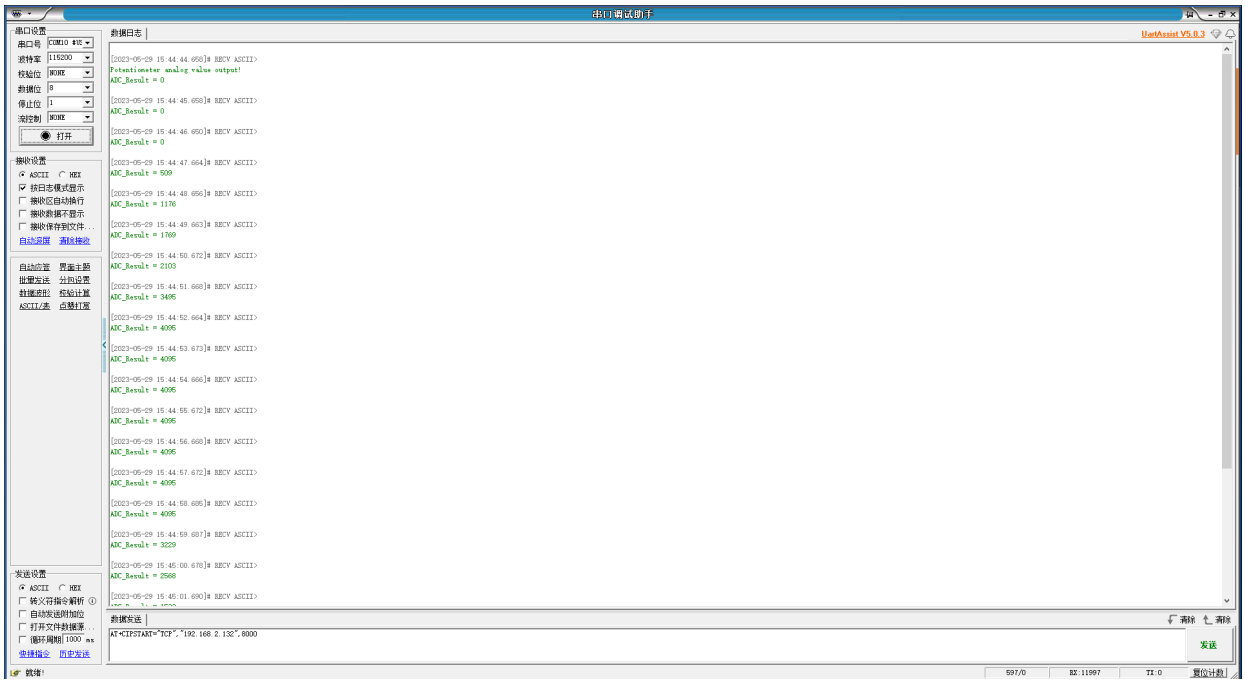
```
#ifndef __ADC_H__
#define __ADC_H__

#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.

At this time, slowly rotate the potentiometer, and the serial port will continuously print the analog value of the potentiometer at this time.

Note: The ADC converted values are printed every 1s as demonstrated here.


串口调试助手
串口设置
串口号 COM10 #3E
波特率 115200
校验位 NONE
数据位 8
停止位 1
流控制 NONE
打开
接收设置
ASCII HEX
按日志模式显示
接收区自动换行
接收数据不显示
接收保存到文件...
自动滚屏 清接收区
自动应答 界面主题
批量发送 分包设置
数据进制 校验计算
ASCII/走 点赞打赏
发送设置
ASCII HEX
转义符指令解析
自动发送附加位
打开文件数据源...
循环周期 1000 ms
快捷指令 历史发送
数据日志
UartAssist V5.0.3
[2023-05-29 15:44:44.650]# RECV ASCII>
Potentiometer analog value output!
ADC_Result = 0
[2023-05-29 15:44:45.658]# RECV ASCII>
ADC_Result = 0
[2023-05-29 15:44:46.650]# RECV ASCII>
ADC_Result = 0
[2023-05-29 15:44:47.664]# RECV ASCII>
ADC_Result = 509
[2023-05-29 15:44:48.656]# RECV ASCII>
ADC_Result = 1176
[2023-05-29 15:44:49.663]# RECV ASCII>
ADC_Result = 1769
[2023-05-29 15:44:50.672]# RECV ASCII>
ADC_Result = 2103
[2023-05-29 15:44:51.668]# RECV ASCII>
ADC_Result = 3495
[2023-05-29 15:44:52.664]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:53.673]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:54.666]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:55.672]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:56.668]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:57.672]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:58.695]# RECV ASCII>
ADC_Result = 4095
[2023-05-29 15:44:59.687]# RECV ASCII>
ADC_Result = 3229
[2023-05-29 15:45:00.670]# RECV ASCII>
ADC_Result = 2568
[2023-05-29 15:45:01.690]# RECV ASCII>
数据发送
清除 清除
AT+CIPSTART="TCP","192.168.2.132",8000
发送
就绪!
597/0
RX:11997
TX:0
复位计数