# 摇杆模块

## 1.模块介绍

摇杆模块是一种常用于遥控车、无人机、船模等设备的控制源，通常有两个供电引脚，VCC和GND,一般有三个信号输出引脚，VRx（X轴），VRy(Y轴)和SW(Z轴)。其中，VRx和VRy连接内部两个电位器，输出的数据就是电压值，电压范围在0-VCC之间，VRx,VRy的值从0-4095分别代表左-右，上-下，摇杆回中时值为2048，内部结构如下图所示需要连接到MCU的模拟IO口，通过ADC读取模拟值。SW是数字信号，仅有高低电平两种状态，需要连接到MCU的通用GPIO上，判断引脚电平状态。

**2.程序实现**

要采集VRx和VRy两个模拟量，我们需要使用ADC来采集。配置好ADC后，获取数据有两种方式，一种方式是定时去获取1次，另一种是使用DMA,将ADC数据实时传输到一个全局变量中。这里我们使用DMA方式，这种方式比较节省CPU资源，采样速率也可以更高。

首先是初始化GPIO

    GPIO\_InitTypeDef GPIO\_InitStructure={0};

    RCC\_APB2PeriphClockCmd(JOYSTICK\_XY\_CLK | JOYSTICK\_Z\_CLK, ENABLE );

    GPIO\_InitStructure.GPIO\_Pin = JOYSTICK\_X\_PIN|JOYSTICK\_Z\_PIN;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN; //GPIO模式为模拟输入

    GPIO\_Init(JOYSTICK\_XY\_PORT, &GPIO\_InitStructure);

    GPIO\_InitStructure.GPIO\_Pin = JOYSTICK\_Z\_PIN;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;

    GPIO\_Init(JOYSTICK\_Z\_PORT, &GPIO\_InitStructure);

采集两个模拟量的引脚需要有ADC功能，并配置成模拟输入模式，Z轴引脚只需要普通GPIO，使用方式与按键开关相同。

接着是初始化ADC

ADC\_InitTypeDef  ADC\_InitStructure={0};

    RCC\_APB2PeriphClockCmd(JOYSTICK\_ADC\_CLK, ENABLE);

    RCC\_ADCCLKConfig(RCC\_PCLK2\_Div8);

    ADC\_DeInit(JOYSTICK\_ADC);

    ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;  //配置ADC为独立模式

    ADC\_InitStructure.ADC\_ScanConvMode = ENABLE;        //多通道模式下开启扫描模式

    ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE;  //设置开启连续转换模式

    ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None; //设置转换不是由外部触发启动，软件触发启动

    ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right; //设置ADC数据右对齐

    ADC\_InitStructure.ADC\_NbrOfChannel = 2;                //规则转换的ADC通道的数目

    ADC\_Init(JOYSTICK\_ADC, &ADC\_InitStructure);                    //根据ADC\_InitStructure中指定的参数初始化ADC1寄存器

    ADC\_Cmd(JOYSTICK\_ADC, ENABLE);      //使能ADC1

    ADC\_ResetCalibration(JOYSTICK\_ADC); //重置ADC1校准寄存器。

    while(ADC\_GetResetCalibrationStatus(JOYSTICK\_ADC)); //等待复位校准结束

    ADC\_StartCalibration(JOYSTICK\_ADC); //开启AD校准

    while(ADC\_GetCalibrationStatus(JOYSTICK\_ADC));      //等待校准结束

然后是初始化DMA

    DMA\_InitTypeDef DMA\_InitStructure={0};

    RCC\_AHBPeriphClockCmd( JOYSTICK\_DMA\_CLK, ENABLE ); //使能开启DMA时钟

    DMA\_DeInit(JOYSTICK\_DMA\_CHANNEL); //复位DMA控制器

    DMA\_InitStructure.DMA\_PeripheralBaseAddr = (u32)&ADC1->RDATAR;  //配置外设地址为ADC数据寄存器地址

    DMA\_InitStructure.DMA\_MemoryBaseAddr = (u32)gJoystickValue; //配置存储器地址为读取ADC值地址,全局变量

    DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralSRC;              //配置数据源为外设，即DMA传输方式为外设到存储器

    DMA\_InitStructure.DMA\_BufferSize = 2;                      //设置DMA数据缓冲区大小，此处设置为2

    DMA\_InitStructure.DMA\_PeripheralInc = DMA\_PeripheralInc\_Disable;//设置DMA外设递增模式关闭

    DMA\_InitStructure.DMA\_MemoryInc = DMA\_MemoryInc\_Enable;         //设置DMA存储器递增模式开启

    DMA\_InitStructure.DMA\_PeripheralDataSize = DMA\_PeripheralDataSize\_HalfWord; //设置外设数据大小为半字，即两个字节

    DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord;         //设置存储器数据大小为半字，即两个字节

    DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Circular;     //设置DMA模式为循环传输模式

    DMA\_InitStructure.DMA\_Priority = DMA\_Priority\_High; //设置DMA传输通道优先级为高，当使用一 DMA通道时，优先级设置不影响

    DMA\_InitStructure.DMA\_M2M = DMA\_M2M\_Disable;        //因为此DMA传输方式为外设到存储器，因此禁用存储器到存储器传输方式

    DMA\_Init( JOYSTICK\_DMA\_CHANNEL, &DMA\_InitStructure );      //初始化DMA

    DMA\_Cmd(JOYSTICK\_DMA\_CHANNEL , ENABLE);  //使能DMA

    //开启DMA传输

    ADC\_RegularChannelConfig(JOYSTICK\_ADC, JOYSTICK\_ADC\_XCHANNEL, 1, ADC\_SampleTime\_55Cycles5);

    ADC\_RegularChannelConfig(JOYSTICK\_ADC, JOYSTICK\_ADC\_YCHANNEL, 2, ADC\_SampleTime\_55Cycles5);

    ADC\_DMACmd(JOYSTICK\_ADC, ENABLE);

    ADC\_SoftwareStartConvCmd(JOYSTICK\_ADC, ENABLE);//使用软件触发

需要注意的是如果要一直采集数据，需要使能循环模式，DMA传输完成一次数据后，会自动重新开始下一次传输。gJoystickValue是用来保存采集数据的全局变量。

完整代码如下所示：

**Joystick.h**

#ifndef \_\_JOYSTICK\_H

#define \_\_JOYSTICK\_H

#include "debug.h"

#include "user.h"

#define JOYSTICK\_X\_PIN     GPIO\_Pin\_0

#define JOYSTICK\_Y\_PIN     GPIO\_Pin\_1

#define JOYSTICK\_Z\_PIN     GPIO\_Pin\_13

#define JOYSTICK\_XY\_PORT   GPIOA

#define JOYSTICK\_Z\_PORT    GPIOC

#define JOYSTICK\_XY\_CLK         RCC\_APB2Periph\_GPIOA

#define JOYSTICK\_Z\_CLK          RCC\_APB2Periph\_GPIOC

#define JOYSTICK\_ADC            ADC1

#define JOYSTICK\_ADC\_CLK        RCC\_APB2Periph\_ADC1

#define JOYSTICK\_ADC\_XCHANNEL   ADC\_Channel\_0

#define JOYSTICK\_ADC\_YCHANNEL   ADC\_Channel\_1

#define JOYSTICK\_DMA            DMA1

#define JOYSTICK\_DMA\_CHANNEL    DMA1\_Channel1

#define JOYSTICK\_DMA\_CLK        RCC\_AHBPeriph\_DMA1

void Joystick\_Init(void);

#endif

**Joystick.c**

#include "Joystick.h"

void Joystick\_Init(void)

{

    //GPIO初始化

    GPIO\_InitTypeDef GPIO\_InitStructure={0};

    RCC\_APB2PeriphClockCmd(JOYSTICK\_XY\_CLK | JOYSTICK\_Z\_CLK, ENABLE );

    GPIO\_InitStructure.GPIO\_Pin = JOYSTICK\_X\_PIN|JOYSTICK\_Z\_PIN;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN; //GPIO模式为模拟输入

    GPIO\_Init(JOYSTICK\_XY\_PORT, &GPIO\_InitStructure);

    GPIO\_InitStructure.GPIO\_Pin = JOYSTICK\_Z\_PIN;

    GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;

    GPIO\_Init(JOYSTICK\_Z\_PORT, &GPIO\_InitStructure);

    //ADC初始化

    ADC\_InitTypeDef  ADC\_InitStructure={0};

    RCC\_APB2PeriphClockCmd(JOYSTICK\_ADC\_CLK, ENABLE);

    RCC\_ADCCLKConfig(RCC\_PCLK2\_Div8);

    ADC\_DeInit(JOYSTICK\_ADC);

    ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;  //配置ADC为独立模式

    ADC\_InitStructure.ADC\_ScanConvMode = ENABLE;        //多通道模式下开启扫描模式

    ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE;  //设置开启连续转换模式

    ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None; //设置转换不是由外部触发启动，软件触发启动

    ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right; //设置ADC数据右对齐

    ADC\_InitStructure.ADC\_NbrOfChannel = 2;                //规则转换的ADC通道的数目

    ADC\_Init(JOYSTICK\_ADC, &ADC\_InitStructure);                    //根据ADC\_InitStructure中指定的参数初始化ADC1寄存器

    ADC\_Cmd(JOYSTICK\_ADC, ENABLE);      //使能ADC1

    ADC\_ResetCalibration(JOYSTICK\_ADC); //重置ADC1校准寄存器。

    while(ADC\_GetResetCalibrationStatus(JOYSTICK\_ADC)); //等待复位校准结束

    ADC\_StartCalibration(JOYSTICK\_ADC); //开启AD校准

    while(ADC\_GetCalibrationStatus(JOYSTICK\_ADC));      //等待校准结束

    //DMA初始化

    DMA\_InitTypeDef DMA\_InitStructure={0};

    RCC\_AHBPeriphClockCmd( JOYSTICK\_DMA\_CLK, ENABLE ); //使能开启DMA时钟

    DMA\_DeInit(JOYSTICK\_DMA\_CHANNEL); //复位DMA控制器

    DMA\_InitStructure.DMA\_PeripheralBaseAddr = (u32)&ADC1->RDATAR;  //配置外设地址为ADC数据寄存器地址

    DMA\_InitStructure.DMA\_MemoryBaseAddr = (u32)gJoystickValue; //配置存储器地址为读取ADC值地址,全局变量

    DMA\_InitStructure.DMA\_DIR = DMA\_DIR\_PeripheralSRC;              //配置数据源为外设，即DMA传输方式为外设到存储器

    DMA\_InitStructure.DMA\_BufferSize = 2;                      //设置DMA数据缓冲区大小，此处设置为2

    DMA\_InitStructure.DMA\_PeripheralInc = DMA\_PeripheralInc\_Disable;//设置DMA外设递增模式关闭

    DMA\_InitStructure.DMA\_MemoryInc = DMA\_MemoryInc\_Enable;         //设置DMA存储器递增模式开启

    DMA\_InitStructure.DMA\_PeripheralDataSize = DMA\_PeripheralDataSize\_HalfWord; //设置外设数据大小为半字，即两个字节

    DMA\_InitStructure.DMA\_MemoryDataSize = DMA\_MemoryDataSize\_HalfWord;         //设置存储器数据大小为半字，即两个字节

    DMA\_InitStructure.DMA\_Mode = DMA\_Mode\_Circular;     //设置DMA模式为循环传输模式

    DMA\_InitStructure.DMA\_Priority = DMA\_Priority\_High; //设置DMA传输通道优先级为高，当使用一 DMA通道时，优先级设置不影响

    DMA\_InitStructure.DMA\_M2M = DMA\_M2M\_Disable;        //因为此DMA传输方式为外设到存储器，因此禁用存储器到存储器传输方式

    DMA\_Init( JOYSTICK\_DMA\_CHANNEL, &DMA\_InitStructure );      //初始化DMA

    DMA\_Cmd(JOYSTICK\_DMA\_CHANNEL , ENABLE);  //使能DMA

    //开启DMA传输

    ADC\_RegularChannelConfig(JOYSTICK\_ADC, JOYSTICK\_ADC\_XCHANNEL, 1, ADC\_SampleTime\_55Cycles5);

    ADC\_RegularChannelConfig(JOYSTICK\_ADC, JOYSTICK\_ADC\_YCHANNEL, 2, ADC\_SampleTime\_55Cycles5);

    ADC\_DMACmd(JOYSTICK\_ADC, ENABLE);

    ADC\_SoftwareStartConvCmd(JOYSTICK\_ADC, ENABLE);//使用软件触发

}

测试程序

void JOYSTICK\_TEST(void)

{

    printf("x方向值为：%d;",gJoystickValue[0]);

    printf("y方向值为：%d;",gJoystickValue[1]);

    printf("Z值为%d;",GPIO\_ReadOutputDataBit(JOYSTICK\_Z\_PORT, JOYSTICK\_Z\_PIN));

    Delay\_Ms(100);

}

结果如下图所示：

