MINIGPA103 USBHID评估板入门学习手册基于APM32SDK

第一部分、硬件概述

1.1 实物概图

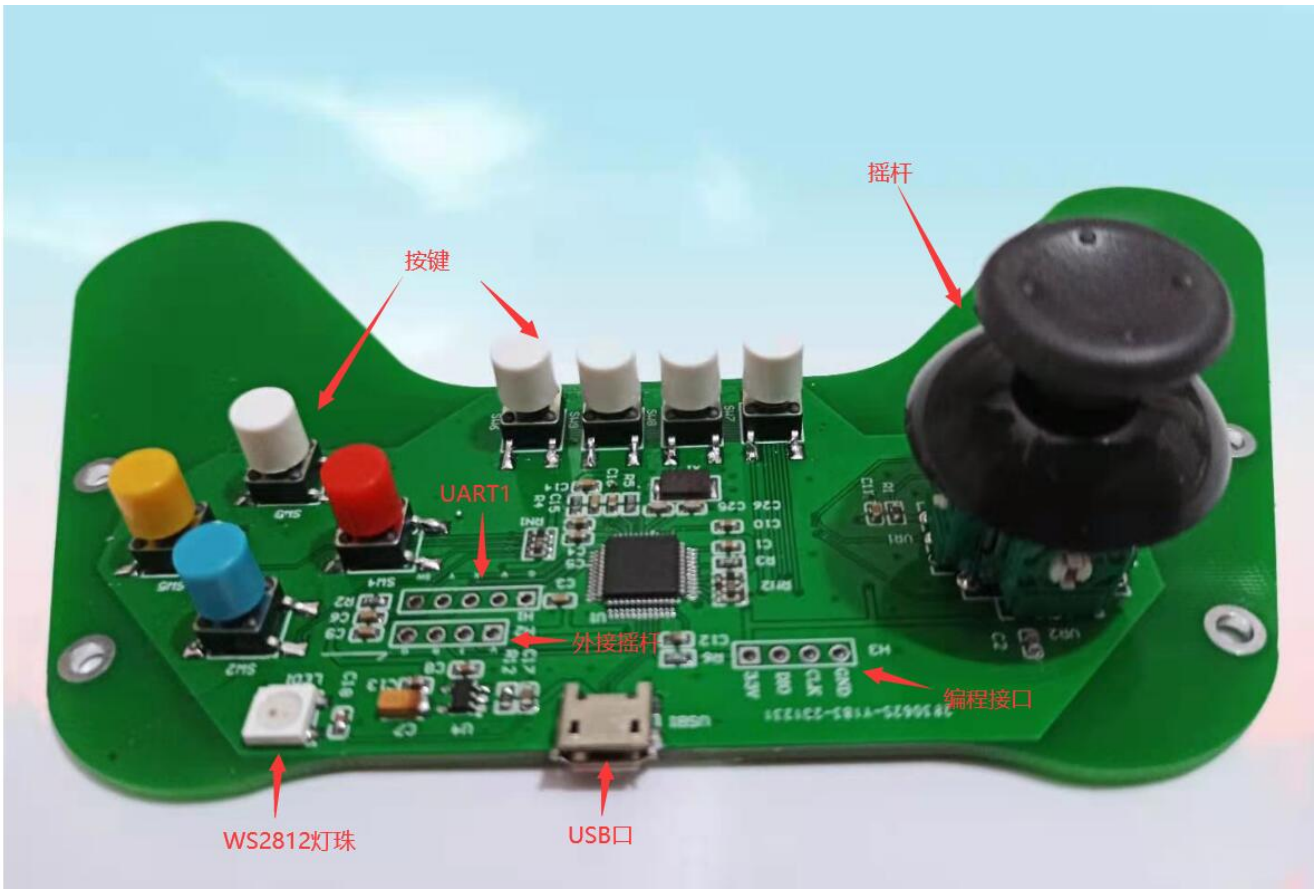


图1.1Gamepad实物概图

如图1.1所示Gamepad评估板配置了8个6*6轻触按键，一个摇杆（Joystick），搭载一颗WS2812B灯珠，并将UART1串口，编程接口（SWD），外接Joystick接口，microUSB接口引出;

1.2 Gamepad原理图

Gamepad原理图如图1.2所示，如看不清可打开Doc目录下的PDF文档查阅

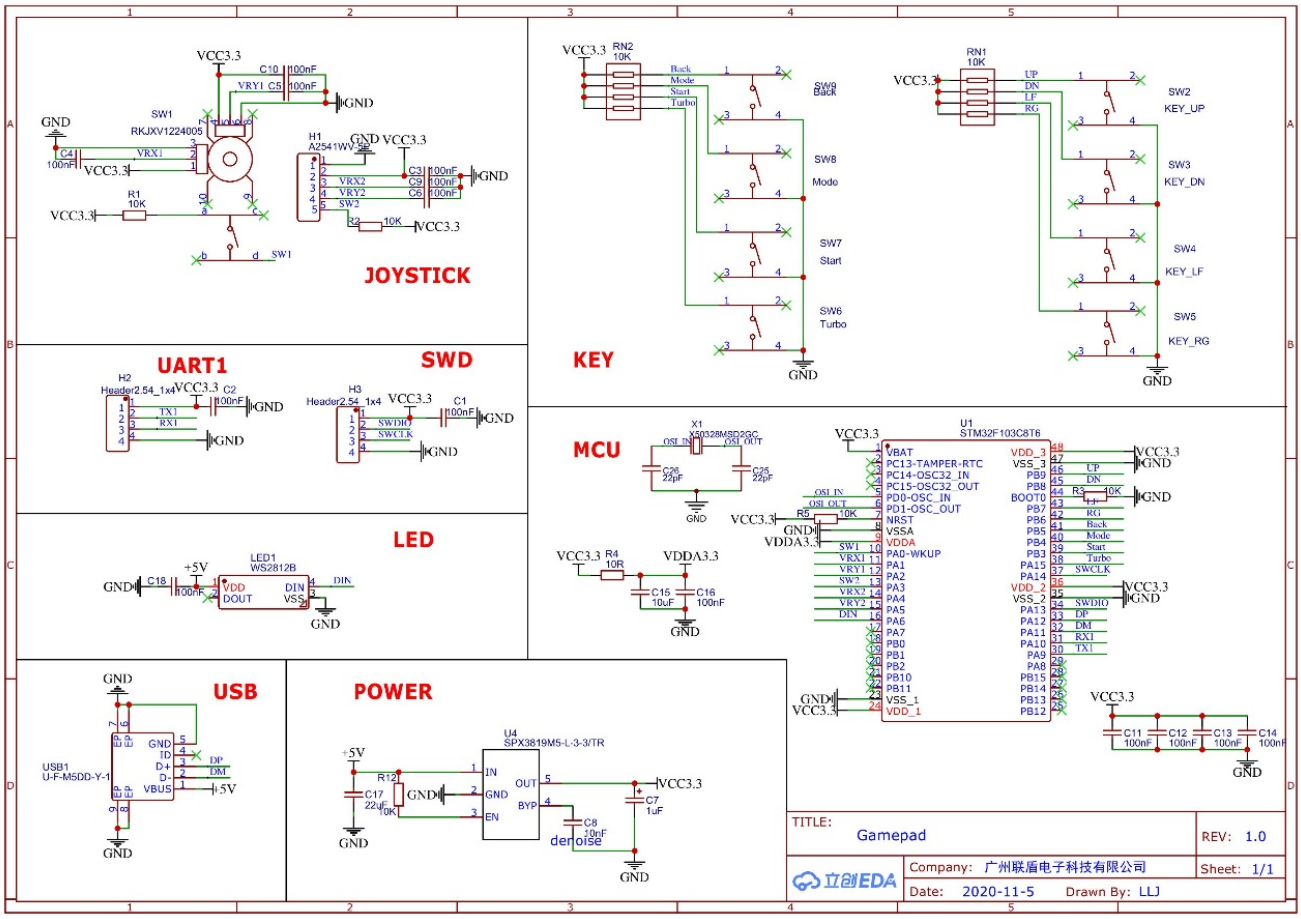


图1.2 Gamepad原理图

第二部分、软件工具

2.1 软件概述

在 /Software 目录下是常用的工具软件：

1. Dt2_4：配置USB设备Report描述符的工具；
2. USBHID调试助手/呀呀USB： USB调试工具，相当于串口调试助手功能；
3. BUSHound：总线调试工具；
4. USBlyzer：一款专业的USB协议分析软件
5. MDK:常用编译器；
6. STM32CubeMX：代码生成工具；

第三部分、实战训练

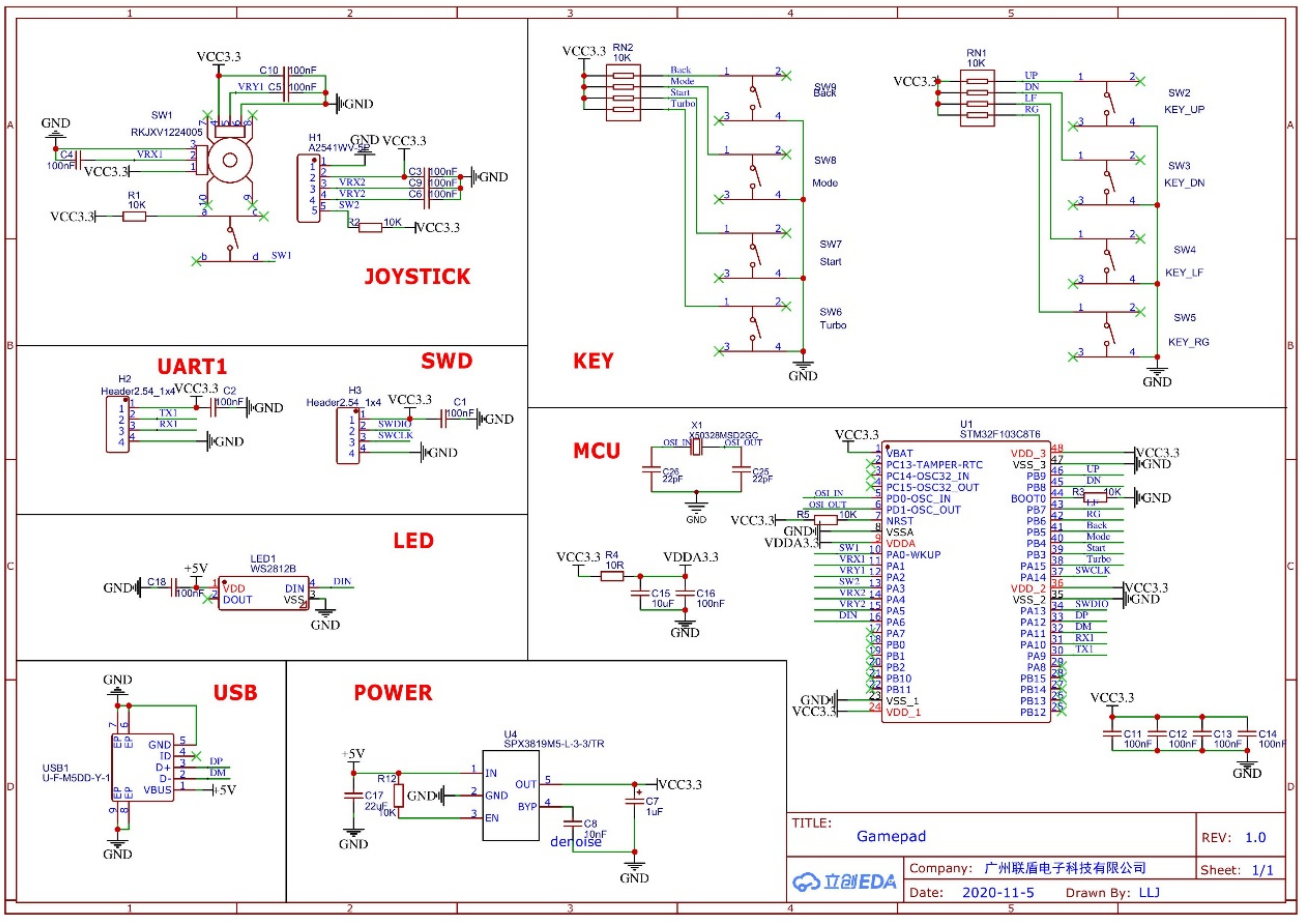
3.1 实例Eg1_KeyTest

我们想要测试一下按键的按下功能，主要是测试使用APM32F10x_SDK的库函数开发测试GPIO 输入模式的使用；

3.1.1硬件设计

如下图是我们评估板的原理图，原理图是基于STM32F103的，不过APM32F103软硬件上完全兼容STM32F103，所以这里我们直接使用原理图，可以看到SW1~SW9分别有以下对应：

```
#define         SW1         GPIO_ReadInputBit(GPIOA,GPIO_PIN_0)
#define         SW2         GPIO_ReadInputBit(GPIOB,GPIO_PIN_9)
#define         SW3         GPIO_ReadInputBit(GPIOB,GPIO_PIN_8)
#define         SW4         GPIO_ReadInputBit(GPIOB,GPIO_PIN_7)
#define         SW5         GPIO_ReadInputBit(GPIOB,GPIO_PIN_6)
#define         SW6         GPIO_ReadInputBit(GPIOA,GPIO_PIN_15)
#define         SW7         GPIO_ReadInputBit(GPIOB,GPIO_PIN_3)
#define         SW8         GPIO_ReadInputBit(GPIOB,GPIO_PIN_4)
#define         SW9         GPIO_ReadInputBit(GPIOB,GPIO_PIN_5)
```



我们只要配置8个GPIO作为输入去检测按键信号；

3.1.2 软件设计

首先关于新建工程，我们直接使用官方的Examples下面的GPIO例子，将输出改成输入模式，初始化代码如下：

```
/*!
 * @brief       Board_KeyGPIOInit
 *
 * @param       None
 *
 * @retval      None
 */
void Board_KeyGPIOInit(void)
{
```

```
    GPIO_Config_T gpioConfigStruct;


    RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_GPIOA|RCM_APB2_PERIPH_GPIOB|RCM_APB2_PERIPH_AFIO);
    //PB3复位后是JTDO功能，这里需要禁用JTAG以实现PB3作为上拉输入模式 ，并且AFIO时钟也要使能
    GPIO_ConfigPinRemap(GPIO_REMAP_SWJ_JTAGDISABLE);

    gpioConfigStruct.mode = GPIO_MODE_IN_PU;
    gpioConfigStruct.pin = GPIO_PIN_3|GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6
                          |GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9;
    gpioConfigStruct.speed = GPIO_SPEED_50MHz;

    GPIO_Config(GPIOB, &gpioConfigStruct);

    gpioConfigStruct.pin = GPIO_PIN_15;
    GPIO_Config(GPIOA, &gpioConfigStruct);

    gpioConfigStruct.mode = GPIO_MODE_IN_PD;
    gpioConfigStruct.pin = GPIO_PIN_0;
    GPIO_Config(GPIOA, &gpioConfigStruct);
}
```

需要留意的是PB3复位后是JTDO功能（这点参考规格书可以确认），这里需要禁用JTAG以实现PB3作为上拉输入模式；由于Up主一开始也没有留意，直接配置，然后发现PB7一直处于按下，低电平状态，后来才通过查看规格书中引脚图得知PB3复位后是JTDO功能；这个故事告诉我们一个道理：数据手册要经常看；

另外我们发现RCM_APB2_PERIPH_AFIO时钟也需要使能，GPIO_ConfigPinRemap这个函数我们也是只禁用了JTAG；

初始化完成之后我们需要写一个测试程序以测试按键是否按下；

```
void Board_ButtonScan(void)
{
    if(SW1==BIT_SET)
    {
        printf("SW1 Down\r\n");
    }
    if(SW2==BIT_RESET)
    {
        printf("SW2 Down\r\n");
    }
    if(SW3==BIT_RESET)
    {
        printf("SW3 Down\r\n");
    }
    if(SW4==BIT_RESET)
    {
        printf("SW4 Down\r\n");
    }
    if(SW5==BIT_RESET)
    {
        printf("SW5 Down\r\n");
    }
    if(SW6==BIT_RESET)
    {
        printf("SW6 Down\r\n");
    }
```

```
    if(SW7==BIT_RESET)
    {
        printf("SW7 Down\r\n");
    }
    if(SW8==BIT_RESET)
    {
        printf("SW8 Down\r\n");
    }
    if(SW9==BIT_RESET)
    {
        printf("SW9 Down\r\n");
    }
}
```

最后在main函数中的while循环里调用并延迟一会；

```
/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
int main(void)
{
    Board_KeyGPIOInit();
    Board_UartPrintInit();
    while (1)
    {
        Board_ButtonScan();
        Delay();

    }
}

/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
void Delay(void)
{
    volatile uint32_t delay = 0xfffff;

    while(delay--);
}
```

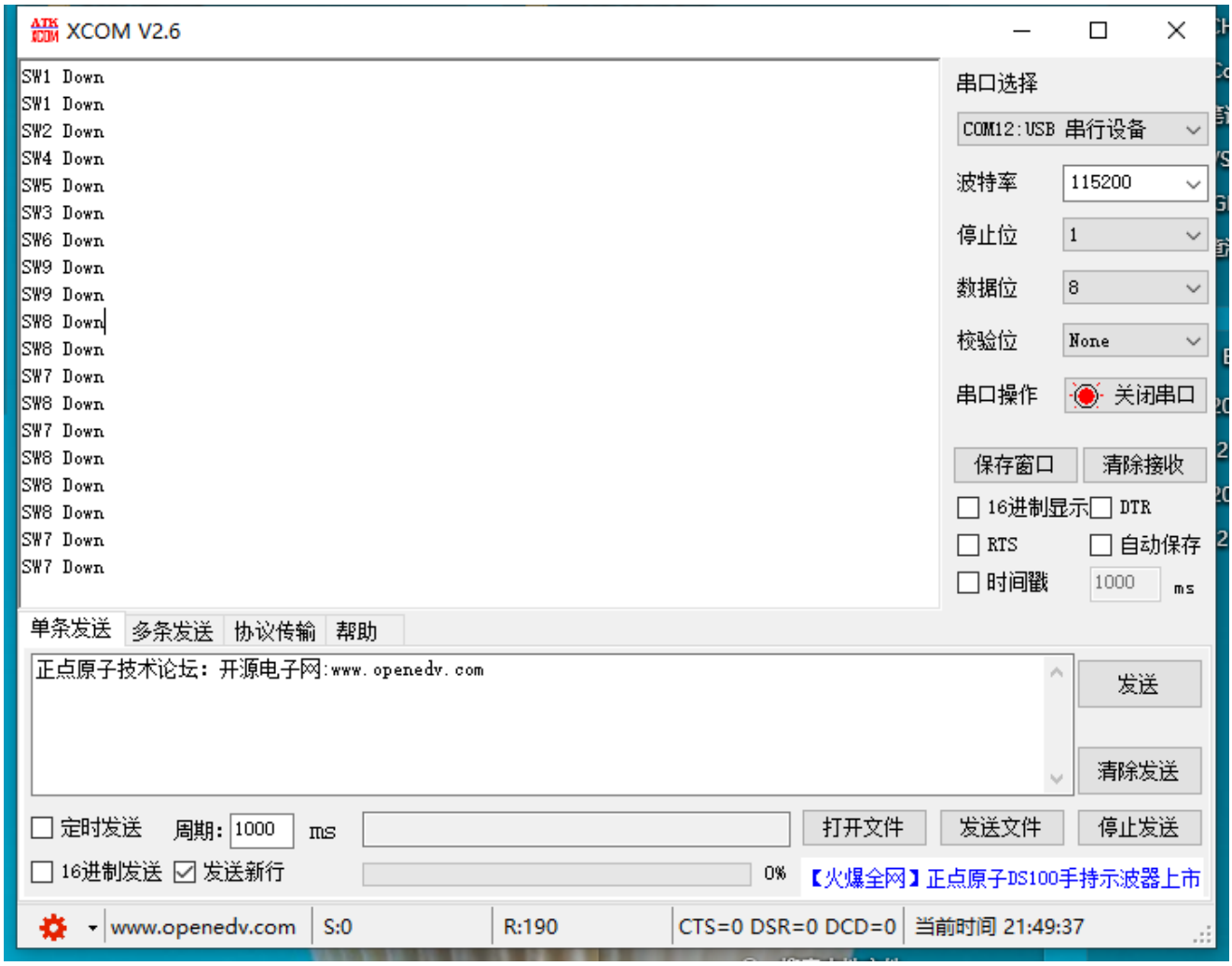
3.1.3 下载验证

我们通过我们自有的仿真器模块WCH-link（DAP模式）把程序下载进去即可，仿真器需要选择CMSIS-DAP Debugger；

这里用到的wch-link，我们是在这里购买的：https://item.taobao.com/item.htm?id=671288574690

wch-link支持DAP(ARM)和RV（wch RISC-V）模式，并且支持Usb转TTL串口；

我们把我们评估板上的H1的GND和TX分别接到wch-link的GND和RX；打开串口调试助手可以看到如下现象：



3.1.4 入门视频

本节的入门视频链接如下：

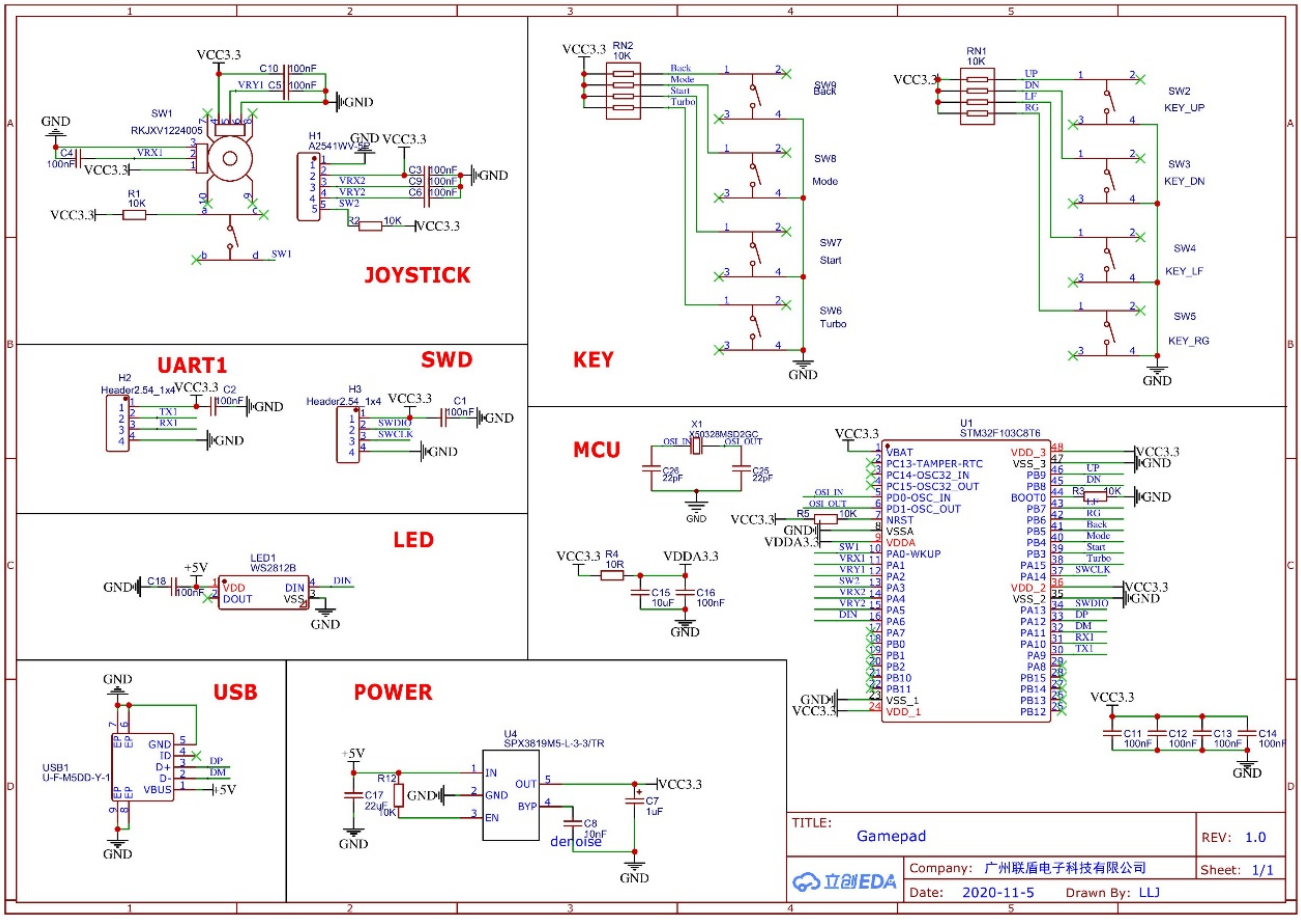
【MINIGPA103 USBHID评估板】重新出发基于APM32F103CxT6实现评估板KEY按键输入测试 摇杆鼠标游戏手柄键盘设备*哔哩哔哩*bilibili

3.2 实例Eg2_JtickAdc

本节我们将通过配置ADC DMA模式去对摇杆电位器的电压进行采样；

3.2.1硬件设计

如下图是我们评估板的原理图，可以看到VRX1和VRY1分别有以下对应PA1和PA2，这是ADC的ch1和ch2；



3.2.2 软件设计

在上一节的基础上，我们初始ADC和DMA，初始化代码如下：

```
/*!
 * @brief      Board_KeyGPIOInit
 *
 * @param       None
 *
 * @retval      None
 */
void Board_JoystickADCInit(void)
{
    GPIO_Config_T configStruct;
    ADC_Config_T ADC_configStruct;
    DMA_Config_T DMA_ConfigStruct;


    RCM_ConfigADCCLK(RCM_PCLK2_DIV_6); /* 6分频 72/6=12MHZ ADCCLK不能超过14MHZ*/
    RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_GPIOA); /* 使能GPIO时钟 */
    RCM_EnableAHBPeriphClock(RCM_AHB_PERIPH_DMA1);      /* 使能DMA1时钟 */
```

```
    RCM_EnableAPB2PeriphClock(RCM_APB2_PERIPH_ADC1);   /* 使能ADC1时钟 */

    NVIC_EnableIRQRequest(ADC1_2_IRQn, 0, 0);

    configStruct.pin = GPIO_PIN_1|GPIO_PIN_2;
    configStruct.mode = GPIO_MODE_ANALOG;
    GPIO_Config(GPIOA, &configStruct);

    DMA_Reset(DMA1_Channel1); /* 复位DMA1通道1 */

    DMA_ConfigStruct.peripheralBaseAddr = ADC1_DR_Address;          /* DMA通道外设基地址 */
    DMA_ConfigStruct.memoryBaseAddr = (uint32_t)dma_buffer;         /* DMA通道ADC数据存储器 */
    DMA_ConfigStruct.dir = DMA_DIR_PERIPHERAL_SRC;                  /* 指定外设为源地址 */
    DMA_ConfigStruct.bufferSize = 2;                                /* DMA缓冲区大小（根据ADC采集通
道数量修改） */
    DMA_ConfigStruct.peripheralInc = DMA_PERIPHERAL_INC_DISABLE;    /* 当前外设寄存器地址不变（即不
自增） */
    DMA_ConfigStruct.memoryInc = DMA_MEMORY_INC_ENABLE;             /*  当前存储器地址：Disable不
变，Enable递增（用于多通道采集） */
    DMA_ConfigStruct.peripheralDataSize = DMA_PERIPHERAL_DATA_SIZE_HALFWORD; /* 外设数据宽度16位
*/
    DMA_ConfigStruct.memoryDataSize = DMA_MEMORY_DATA_SIZE_HALFWORD;         /* 存储器数据宽度16位
*/
    DMA_ConfigStruct.loopMode = DMA_MODE_CIRCULAR; /* DMA通道操作模式位环形缓冲模式 */
    DMA_ConfigStruct.priority = DMA_PRIORITY_HIGH; /* DMA通道优先级高 */
    DMA_ConfigStruct.M2M = DMA_M2MEN_DISABLE;      /* 禁止DMA通道存储器到存储器传输 */
    DMA_Config(DMA1_Channel1, &DMA_ConfigStruct);

    //DMA_EnableInterrupt(DMA1_Channel1, DMA_INT_TC);
    DMA_Enable(DMA1_Channel1);



    ADC_Reset(ADC1); /* 复位ADC1 */
    /** ADC1 Configuration */
    ADC_configStruct.mode = ADC_MODE_INDEPENDENT;               /* ADC1工作在独立模式 */
    ADC_configStruct.scanConvMode = ENABLE;                     /* 使能扫描 */
    ADC_configStruct.continuosConvMode = ENABLE;                /* 使能ADC连续转换模式 轮询方式使用
*/
//  ADC_configStruct.continuosConvMode = DISABLE;               /* 不使能ADC连续转换模式 中断方式使
用*/
    ADC_configStruct.externalTrigConv = ADC_EXT_TRIG_CONV_None; /* 软件控制转换 */
    ADC_configStruct.dataAlign = ADC_DATA_ALIGN_RIGHT;          /* 转换数据右对齐 */
    ADC_configStruct.nbrOfChannel = 2;                          /* 顺序进行规则转换的ADC通道的数目
*/
    ADC_Config(ADC1, &ADC_configStruct);                        /* 初始化ADC1寄存器 */

    /* 设置指定ADC的规则组通道，设置它们的转化顺序和采样时间 */
    ADC_ConfigRegularChannel(ADC1, ADC_CHANNEL_1, 1, ADC_SAMPLETIME_13CYCLES5); /* ADC1选择通道10
采样顺序1 采样时间13.5个周期 */
    ADC_ConfigRegularChannel(ADC1, ADC_CHANNEL_2, 2, ADC_SAMPLETIME_13CYCLES5); /* ADC1选择通道11
采样顺序2 采样时间13.5个周期 */


//  ADC_EnableInterrupt(ADC1, ADC_INT_EOC); /* 使能ADC转换完成中断 */
    ADC_EnableDMA(ADC1); /* 使能ADC的DMA支持 */
    ADC_Enable(ADC1);    /* 使能ADC1 */
```

```
    ADC_ResetCalibration(ADC1);                 /* 复位ADC1的校准寄存器 */
    while(ADC_ReadResetCalibrationStatus(ADC1)); /* 等待ADC1复位校准完成 */
    ADC_StartCalibration(ADC1);                 /* 开始ADC1校准 */
    while(ADC_ReadCalibrationStartFlag(ADC1));   /* 等待ADC1校准完成 */

    ADC_EnableSoftwareStartConv(ADC1); /* 启动ADC1转换 */



}
```

配置完成之后我们需要写一个测试程序以测试ADC DMA的采样；

```
/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
int main(void)
{
    Board_Init();
    while (1)
    {
        /* 以下采用轮询方式等待转换完成 */
        while(!ADC_ReadStatusFlag(ADC1, ADC_FLAG_EOC)); /* 使用此行代码必须使能连续转换模式 */
        printf("ADC1采样数据:\r\n");
        for (uint8_t i = 0; i < 2; i++) {
            printf("ADC_CHANNEL_%d:%d\r\n", i, dma_buffer[i]);
        }
        printf("\r\n");
        Delay();

    }
```

最后在main函数中的while循环里调用并延迟一会；

```
/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
int main(void)
{
    Board_KeyGPIOInit();
    Board_UartPrintInit();
    while (1)
    {
        Board_ButtonScan();
```

```
        Delay();

    }
}

/*!
 * @brief       Main program
 *
 * @param       None
 *
 * @retval      None
 *
 */
void Delay(void)
{
    volatile uint32_t delay = 0xfffff;

    while(delay--);
}
```

3.2.3 下载验证

请参考视频；

3.2.4 入门视频

本节的入门视频链接如下：

【MINIGPA103 USBHID评估板】基于APM32F103CxT6实现评估板ADC DMA对摇杆电位器进行采样测试 摇杆鼠标游戏手柄键盘设备*哔哩哔哩*bilibili

3.3 实例Eg3_USB_HID_Joystick

前两节我们把基本的外设以及调试OK，现在我们开始USB的学习，本节需要具备一定的USB设备开发知识；关于Usb的学习，这里推荐两个学习视频和一个学习网站：

1. USB技术应用与开发：

   https://www.bilibili.com/video/BV1sy4y1n7d9/?spm_id_from=333.33.header_right.fav_list.click&vd_source=2bbde87de845d5220b1d8ba075c12fb0

2. CherryUSB设备协议栈教程：

   https://www.bilibili.com/video/BV1Ef4y1t73d/?spm_id_from=333.33.header_right.fav_list.click&vd_source=2bbde87de845d5220b1d8ba075c12fb0

3. USB中文网：

   https://www.usbzh.com/

我们主要做USB HID开发，一般我们需要了解一些标准请求，还有HID类的请求；其中标准请求主要是主机获取设备描述符、配置描述符、接口描述符、端点描述符、字符串描述符的过程，如果是HID，还有HID描述符的过程，以及报表描述符的过程；

3.2.1硬件设计

请参考原理图；

3.2.2 软件设计

这一节，主要是USB的代码；主要对USBD_InitParam_T这个USB初始化参数结构体的初始化

```
/*!
 * @brief       HID mouse init
 *
 * @param       None
 *
 * @retval      None
 */
void HidMouse_Init(void)
{
    USBD_InitParam_T usbParam;

    Get_SerialNum();

    USBD_InitParamStructInit(&usbParam);

    usbParam.classReqHandler = USBD_ClassHandler;
    usbParam.stdReqExceptionHandler = HidMouse_ReportDescriptor;

    usbParam.resetHandler = HidMouse_Reset;
    usbParam.inEpHandler = HidMouse_EPHandler;
    usbParam.pDeviceDesc = (USBD_Descriptor_T *)&g_deviceDescriptor;
    usbParam.pConfigurationDesc = (USBD_Descriptor_T *)&g_configDescriptor;


    usbParam.pStringDesc = (USBD_Descriptor_T *)g_stringDescriptor;
    usbParam.pStdReqCallback = &s_stdCallback;

    USBD_Init(&usbParam);
}
```

首先是USBD_ClassHandler，我们不做任何修改

```
/*!
 * @brief       USB HID Class request handler
 *
 * @param       reqData : point to USBD_DevReqData_T structure
 *
 * @retval      None
 */
void USBD_ClassHandler(USBD_DevReqData_T* reqData)
{
    switch (reqData->byte.bRequest)
    {
        case HID_CLASS_REQ_SET_IDLE:
            s_hidIdleState = reqData->byte.wValue[1];
            USBD_CtrlInData(NULL, 0);
            break;
```

```
        case HID_CLASS_REQ_GET_IDLE:
            USBD_CtrlInData(&s_hidIdleState, 1);
            break;

        case HID_CLASS_REQ_SET_PROTOCOL:
            s_hidProtocol = reqData->byte.wValue[0];
            USBD_CtrlInData(NULL, 0);
            break;

        case HID_CLASS_REQ_GET_PROTOCOL:
            USBD_CtrlInData(&s_hidProtocol, 1);
            break;

        default:
            break;
    }
}
```

接着是HidMouse_ReportDescriptor，主要是对获取HID描述符与报表描述符

```
/*!
 * @brief       Standard request Report HID Descriptor
 *
 * @param       reqData:    Standard request data
 *
 * @retval      None
 */
void HidMouse_ReportDescriptor(USBD_DevReqData_T *reqData)
{
    uint8_t len;

    if((reqData->byte.bRequest == USBD_GET_DESCRIPTOR) &&
        (reqData->byte.bmRequestType.bit.recipient == USBD_RECIPIENT_INTERFACE) &&
        (reqData->byte.bmRequestType.bit.type == USBD_REQ_TYPE_STANDARD))
    {
        if(reqData->byte.wValue[1] == 0x21)
        {
            len = USB_MIN(reqData->byte.wLength[0], 9);
            USBD_CtrlInData((uint8_t *)&g_configDescriptor.pDesc[0x12], len);
        }
        else if(reqData->byte.wValue[1] == 0x22)
        {
            len = USB_MIN(reqData->byte.wLength[0], g_ReportDescriptor.size);
            USBD_CtrlInData((uint8_t *)g_ReportDescriptor.pDesc, len);
        }
    }
    else
    {
        USBD_SetEPTxRxStatus(USBD_EP_0, USBD_EP_STATUS_STALL, USBD_EP_STATUS_STALL);
    }
}
```

再有，HidMouse_Reset和HidMouse_EPHandler，前者是配置打开端点1，后者是清除USB缓存；

```
/*!
```

```
 * @brief       Reset
 *
 * @param       None
 *
 * @retval      None
 */
void HidMouse_Reset(void)
{
    USBD_EPConfig_T epConfig;

    s_usbConfigStatus = 0;

    /* Endpoint 1 IN */
    epConfig.epNum = USBD_EP_1;
    epConfig.epType = USBD_EP_TYPE_INTERRUPT;
    epConfig.epBufAddr = USB_EP1_TX_ADDR;
    epConfig.maxPackSize = 4;
    epConfig.epStatus = USBD_EP_STATUS_NAK;
    USBD_OpenInEP(&epConfig);

    USBD_SetEPRxStatus(USBD_EP_1, USBD_EP_STATUS_DISABLE);
}
/*!
 * @brief       Endpoint handler
 *
 * @param       ep:     Endpoint number
 *
 * @param       dir:    Direction.0: Out; 1: In
 *
 * @retval      None
 */
void HidMouse_EPHandler(uint8_t ep)
{
    s_statusEP = 1;
}
```

还有最关键的g_deviceDescriptor与g_configDescriptor，以及g_stringDescriptor，是获取设备描述符，配置描述符以及字符串描述符；

```
/* Device descriptor */
USBD_Descriptor_T g_deviceDescriptor = {s_hidMouseDeviceDescriptor,
HID_MOUSE_DEVICE_DESCRIPTOR_SIZE};
/* Config descriptor */
USBD_Descriptor_T g_configDescriptor = {s_hidMouseConfigDescriptor,
USB_CUSTOM_HID_CONFIG_DESC_SIZ};
/* String descriptor */
USBD_Descriptor_T g_stringDescriptor[SRTING_DESC_NUM] =
{
    {s_hidMouseLandIDString, HID_MOUSE_LANGID_STRING_SIZE},
    {s_hidMouseVendorString, HID_MOUSE_VENDOR_STRING_SIZE},
    {s_hidMouseProductString, HID_MOUSE_PRODUCT_STRING_SIZE},
    {s_hidMouseSerialString, HID_MOUSE_SERIAL_STRING_SIZE}
};
```

这些描述符的获取比较重要，亲直接参考我们的视频讲解；

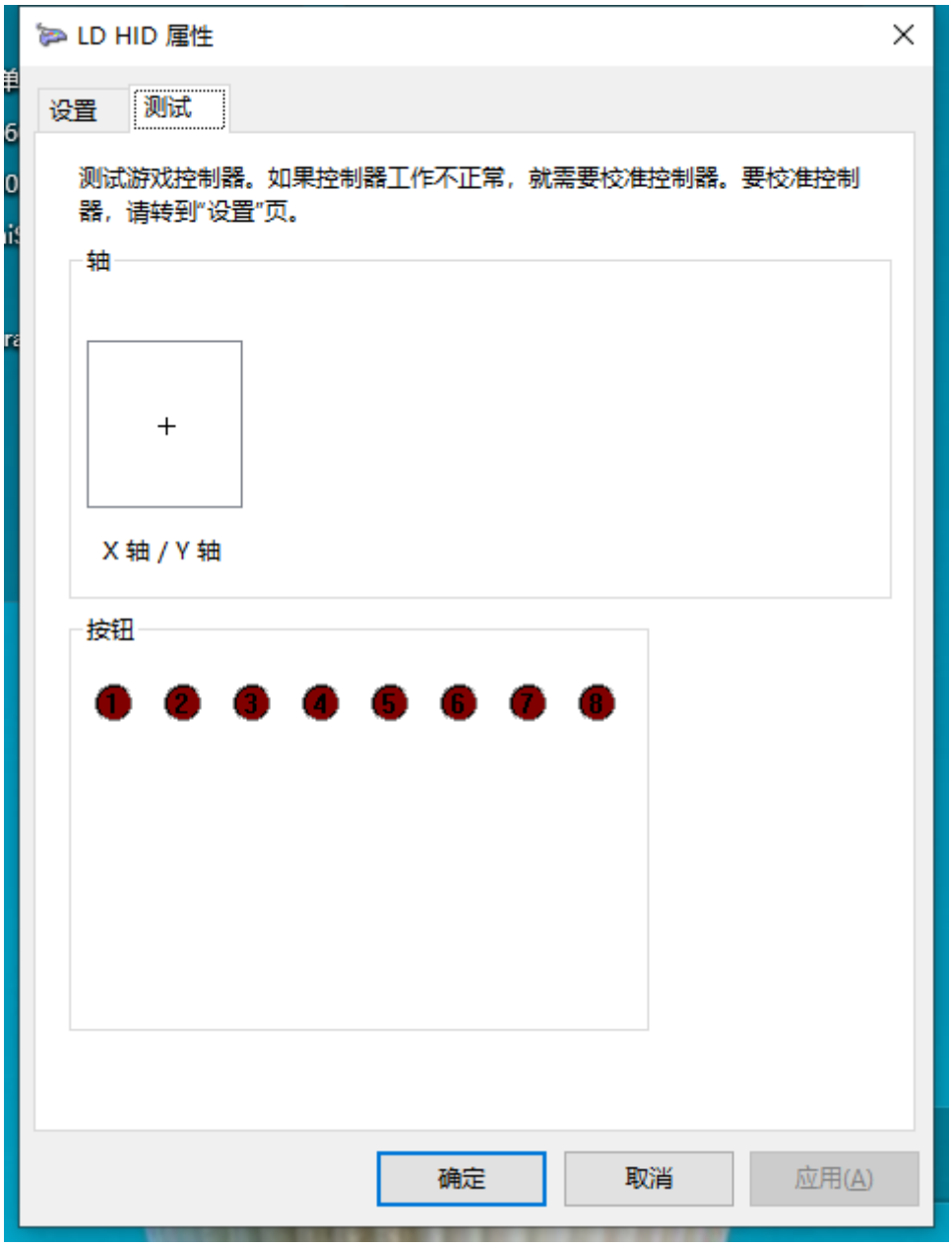
最后是s_stdCallback，是USB初始完成的标志；

```
/*!
 * @brief       Standard request set configuration call back
 *
 * @param       None
 *
 * @retval      None
 */
void HidMouse_SetConfigCallBack(void)
{
    s_usbConfigStatus = 1;
}
/** @defgroup USB_HID_Mouse_Variables Variables
  @{
  */

USBD_StdReqCallback_T s_stdCallback =
{
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    HidMouse_SetConfigCallBack,
    NULL,
    NULL,
    NULL,
    NULL,
};
```

3.2.3 下载验证

我们通过我们自有的仿真器模块WCH-link（DAP模式）把程序下载进去即可，可以得到一个Joystick；



3.2.4 入门视频

本节的入门视频链接如下：

https://www.bilibili.com/video/BV1VP411J7QE/?spm_id_from=333.880.my_history.page.click&vd_source=2bbde87de845d5220b1d8ba075c12fb0