

STM32F401RE como HID

por L. Lindgren e J. Falcão

Introdução

Esta Application Note tem o objetivo de apresentar um exemplo prático do uso de HID no STM32F401RE. Será apresentado um exemplo utilizando uma placa NUCLEO-F401RE e um módulo joystick. Os códigos deste projeto podem ser encontrados no repositório <https://github.com/joaomoreno/f/STM32F401-Joystick-Shield>

Índice

Introdução	1
Visão Geral	2
Descrição do Firmware	2
Descritores	2
Configuração	3
Biblioteca	3
Handle necessário	3
Envio de dados	3
Descrição do Hardware	3
Setup de hardware	3
Reproduzindo o Exemplo	4
STM32CubeMX	4
SW4STM32	5
Executando o Exemplo	5

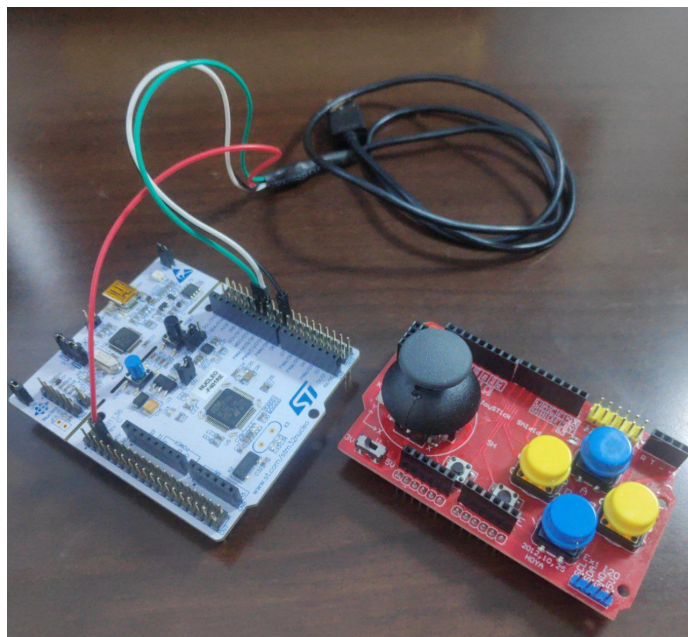


Figura 1: Configuração de exemplo.

Visão Geral

Human Interface Devices (HID), ou Dispositivos de Interface Humana, são equipamentos que interagem traduzindo estímulos de entrada e saída de/para seres humanos em estímulos para/de computadores. Exemplos comuns de HIDs são teclados, mouses, telas, botões e joysticks. Para promover um protocolo generalizado destes dispositivos e permitir que eles tenham a qualidade "Plug-and-Play" (i.e. diferentes componentes de hardware com uma mesma especificação podem ser conectados sem configuração por parte do usuário), há um padrão HID publicado pelo USB Implementers Forum.

Descrição do Firmware

A biblioteca de USB disponibilizada pela ST como Middleware no software STM32CubeMX provê alguns arquivos úteis para esta aplicação:

Middleware da biblioteca:

- `usb_core.c` (cerne das funcionalidades)
- `usb_ioreq.c`, `usb_ctlreq.c` (requisições IO)
- `usb_hid.c` (trata das interações com o Host)

Arquivos de aplicação:

- `usb_conf.c` (funções LL/HAL)
- `usb_device.c` (funções de inicialização)

Destes arquivos, vale destacar o `usb_hid.c`, que contém os descritores do dispositivo USB. Esses descritores servem para que o computador (Host) saiba com qual tipo de dispositivo está lidando (e.g. mouse, teclado, etc.) e como interpretar os pacotes (reports) que está recebendo. Por default, o STM32CubeMX gera descritores para um mouse e assim será usado nesta aplicação.

Obs.: A placa de desenvolvimento usada suporta somente USB FullSpeed (12 Mbits/s), apesar de haver Middleware disponível para USB HighSpeed (480 Mbit/s).

Descritores

Dos vários descritores gerados no arquivo `usb_hid.c`, dois são particularmente interessantes de serem analisados:

USB_HID_CfgFSDesc é o descritor de configurações de USB HID e contém uma série de informações gerais sobre o dispositivo, como corrente de alimentação, número de interfaces, código do país, entre muitas outras.

HID_MOUSE_ReportDesc é um descritor dos **reports**, que são os pacotes de dados enviados pelo Device para o Host. A função deste descritor é informar ao Host como ele deve esperar que os dados do **report** cheguem, e como entendê-los assim que eles são recebidos. Aqui é que fica definido que o dispositivo é um mouse - ou teclado, ou o que seja -, quais são os valores máximos e mínimos dos campos de entrada, tamanho do report, entre outras informações.

Para gerar estes descritores, pode-se ler as definições de protocolo USB, encontrá-los gerados para uma aplicação semelhante e alterá-los, ou usar o gerador oficial chamado **HID Descriptor Tool**.

Segue abaixo, a interpretação dos valores do descritor `HID_MOUSE_ReportDesc`:

```
0x05, 0x01, // Usage Page (Generic Desktop Ctrls)
0x09, 0x02, // Usage (Mouse)
0xA1, 0x01, // Collection (Application)
0x09, 0x01, // Usage (Pointer)
0xA1, 0x00, // Collection (Physical)
0x05, 0x09, // Usage Page (Button)
0x19, 0x01, // Usage Minimum (0x01)
0x29, 0x03, // Usage Maximum (0x03)
0x15, 0x00, // Logical Minimum (0)
0x25, 0x01, // Logical Maximum (1)
0x95, 0x03, // Report Count (3)
0x75, 0x01, // Report Size (1)
0x81, 0x02, // Input
0x95, 0x01, // Report Count (1)
0x75, 0x05, // Report Size (5)
0x81, 0x01, // Input
0x05, 0x01, // Usage Page (Generic Desktop Ctrls)
0x09, 0x30, // Usage (X)
0x09, 0x31, // Usage (Y)
0x09, 0x38, // Usage (Wheel)
0x15, 0x81, // Logical Minimum (-127)
0x25, 0x7F, // Logical Maximum (127)
0x75, 0x08, // Report Size (8)
0x95, 0x03, // Report Count (3)
0x81, 0x06, // Input
0xC0,      // End Collection
```

```
0x09, 0x3C, // Usage (Motion Wakeup)
0x05, 0xFF, // Usage Page (Reserved 0xFF)
0x09, 0x01, // Usage (0x01)
0x15, 0x00, // Logical Minimum (0)
0x25, 0x01, // Logical Maximum (1)
0x75, 0x01, // Report Size (1)
0x95, 0x02, // Report Count (2)
0xB1, 0x22, // Feature
0x75, 0x06, // Report Size (6)
0x95, 0x01, // Report Count (1)
0xB1, 0x01, // Feature
0xC0,      // End Collection
```

Configuração

As modificações necessárias para o uso do HID são poucas e podem ser divididas em 3 grupos, include, criação do handle e envio dos comandos, todos os três grupos serão explicados a seguir.

Biblioteca

O código da main precisa incluir somente a biblioteca principal do HID. Ela deve ser modificada para selecionar o tipo de dispositivo que será apresentado ao computador, nos termos indicados pelo parágrafo sobre HID_MOUSE_ReportDesc.

```
#include "usb_hid.h"
```

Handle necessário

Para o uso do HID é requerido um handle que manipule os dados necessários para a comunicação USB e é declarado desta forma:

```
USB_D_HandleTypeDef hUsbDeviceFS;
```

Envio de dados

O código desta etapa deve ser utilizado todas as vezes que for enviar dados para o computador. Os dados precisam ser passados para a função em um bloco de bytes, em nosso exemplo utilizamos a estrutura abaixo

para o envio de dados de um mouse. Nessas condições o payload tem 4 bytes e é organizado da seguinte forma:

```
struct mouseHID_t {
    uint8_t buttons;
    int8_t x;
    int8_t y;
    int8_t wheel;
};
```

A única coisa que nos resta é enviar os dados armazenados por meio da função. Como a função foi implementada esperando como entrada um ponteiro para int8_t, fazemos um cast e então chamamos a função.

```
uint8_t * buff = (uint8_t *) &mouseHID;
USB_D_SendReport(&hUsbDeviceFS, buff,
                sizeof(struct mouseHID_t));
```

Descrição do Hardware

O STM32F401 possui periférico específico para comunicação via USB e para o seu funcionamento somente é necessário interligar um conector USB aos pinos disponibilizados pelo hardware para esta função.

A placa utilizada é a Nucleo-F401RE, para a utilização como HID ela precisa ser alimentada pela mesma conexão que será utilizada e este processo deve ser feito colocando o jumper PWR na posição E5V. Lembrando que para o processo de gravação é necessário que o jumper esteja na posição U5V.

O cabo utilizado é um cabo USB simples com conector dupont fêmea.

O shield apesar do shield não ser necessário para a utilização do HID ele é essencial para o exemplo disponibilizado junto a essa Application note. Ele se chama Joystick Shield V1.A estão descritas na tabela a seguir e devem ser utilizadas na função pull-up.

Setup de hardware

Para utilizar o HID é necessário simplesmente conectar os fios do cabo USB na placa seguindo a tabela a

seguir e encaixar o Joystick Shield V1.A nos conectores compatíveis com o Arduino.





USB	MCU	Cor do fio
+5V	E5V	
D-	PA11	
D+	PA12	
GND	GND	

Tabela 1: Conexões do USB.

O shield tem todas as conexões utilizadas fixas, mas vale a pena evidenciá-las para a melhor compreensão do funcionamento do código. As conexões estão dispostas na tabela 2:

Função	MCU
A	PA10
B	PB3
C	PB5
D	PB5
E	PB10
F	PA8
K	PA9
X	PA0
Y	PA1

Tabela 2: Conexões do shield.

Reproduzindo o Exemplo

O exemplo foi projetado para a utilização do Joystick Shield V1.A como HID reproduzindo o comportamento de um mouse. O joystick, propriamente dito, controla o movimento do mouse, os botões A e C executam a rolagem para cima e para baixo, os botões D, F e K o clique primário e os botões B e E o clique secundário.

STM32CubeMX

Para reproduzir o exemplo, crie um projeto em branco para a placa STM32F401RE Nucleo64 no STM32CubeMX. Quando perguntado se deseja inicializar os periféricos no modo default, selecione "Não".

Encontre na aba "Pinout & Configuration" a configuração

System Core → *RCC* → *High Speed Clock (HSE)* e selecione a opção **BYPASS Clock Source** como na imagem 2.

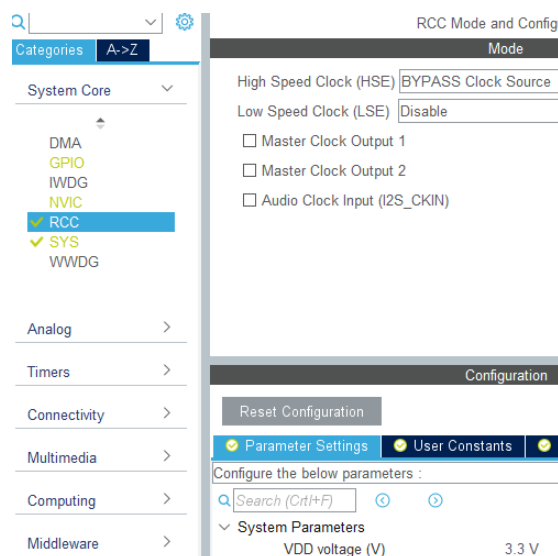


Figura 2: Configurar RCC para BYPASS Clock Source.

Depois disso, configure os pinos descritos na tabela 2 como GPIO no modo input com pull-up ativado.

Em seguida, ative as configurações *Analog* → *ADC1* → *IN0* e *IN1* como na figura 3.

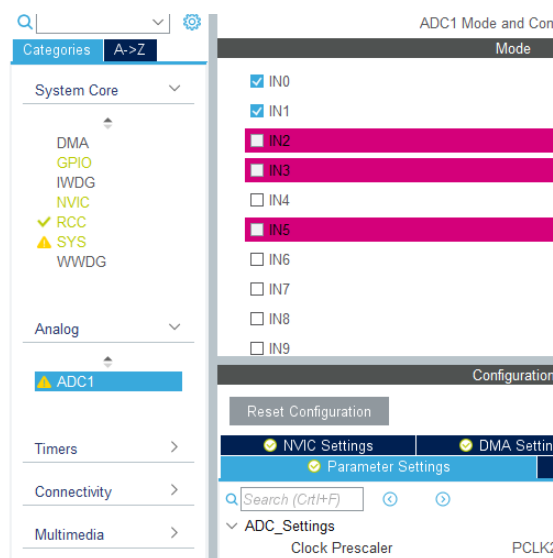


Figura 3: Ativar ADC para as portas IN0 e IN1.

Configure o USB em *Connectivity* → *USB_OTG_FS* → *Mode* para **Device only**, como na figura 4.

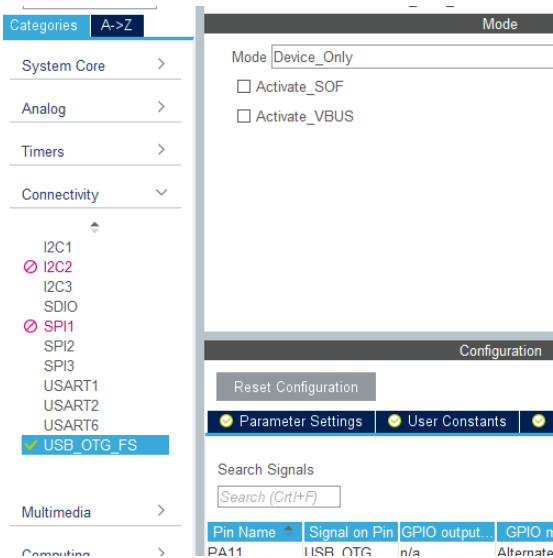


Figura 4: Configurar USB para o modo Device Only.

Na configuração :

Middleware → *USB_DEVICE* → *Class For FS IP*
Selecione a opção **Human Interface Device Class (HID)**, como na figura 5.

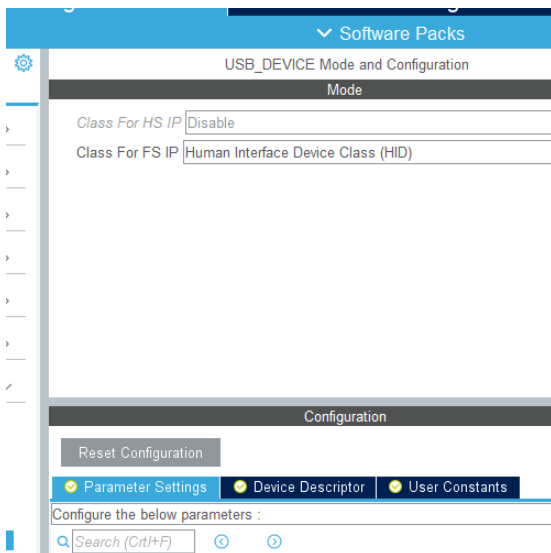


Figura 5: Configurar USB para o modo HID.

Na aba Clock Configuration aparecerá a mensagem de erro abaixo (figura 6). Selecione "Yes" e

após alguns instantes a árvore de clock deve parecer com a figura 6.

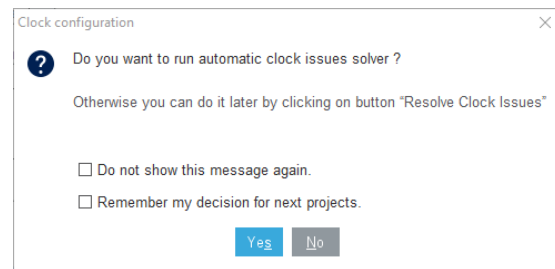


Figura 6: Problema com o clock.

Por fim, configure o nome e local do projeto e gere o código com a toolchain de preferência. Neste projeto foi usada a **SW4STM32**.

SW4STM32

A IDE usada neste projeto foi a **SW4STM32**. Para finalizar o projeto basta configurar os arquivos gerados pelo **STM32CubeMX** segundo a seção **Descrição do Firmware** deste documento e gravar na placa de desenvolvimento.

Executando o Exemplo

A gravação do exemplo pode ser feita a partir da gravação direta do .elf disponibilizado ou também pela compilação própria do código acessível pelo repositório ou pelo arquivo gerado.

IMPORTANTE! O jumper **JP5** (PWR) define a origem da alimentação da placa. Para gravar, o jumper deve estar na posição **U5V** (alimentação pelo STLink) e o USB-Dupont deve estar desconectado da energia. Após a gravação o jumper deve ser colocado na posição necessária para o HID, isto é, ligando o pino central ao **E5V** (alimentação externa pelo USB-Dupont) e o cabo do STLink deve ser desconectado da energia.

Tabela 3: Histórico de revisões.

Revisão	Autores	Modificações
0.1	Luis Lindgren e João Falcão	Lançamento inicial da Application Note