




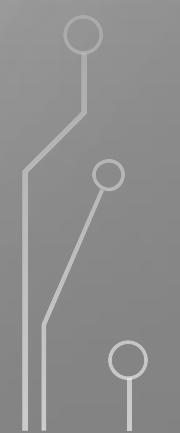
USB HID E CUSTOM HID

REALIZZAZIONE DI UN MOUSE E DI UNA TASTIERA CON STM32F4



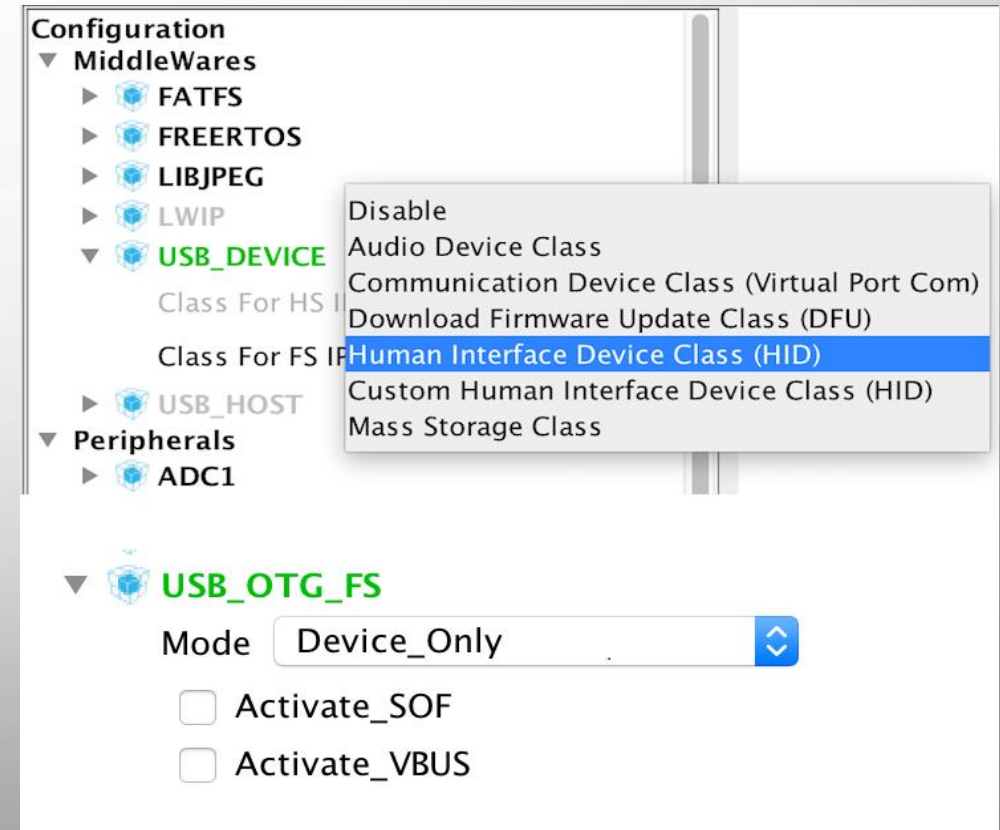
INTRODUZIONE



- In questo tutorial andremo a presentare come creare un USB HID e un USB Custom HID per realizzare un mouse e una keyboard.
 - Abbiamo bisogno di:
 - Due board STM32F4
 - AC6-Tools
 - STM32CubeMX
- 
- 

CREAZIONE PROGETTO CON CUBE MX

- Avviare CUBE MX e impostare come board del progetto la STM32F4.
- Fare un clean dei pin.
- Peripherals:
 - USB_OTG_FS (Mode: Device Only);
 - RCC (HSE: Crystal/Ceramic Resonator).
- Configuration → Middlewares:
 - USB_DEVICE: Human Interface Device Class HID



CREAZIONE PROGETTO CON CUBE MX (2)

- Lasciare al CUBE la configurazione automatica del clock.
- Nelle impostazioni di progetto, impostare il nome, la directory destinazione e selezionare come toolchain SW4STM32.
- Generare il codice ed importarlo in AC6

The screenshot shows the STM32CubeMX configuration window. A dropdown menu for the toolchain is open, listing the following options: EWARM, MDK-ARM V4, MDK-ARM V5, TrueSTUDIO, **✓ SW4STM32** (selected), Makefile, and Other Toolchains (GPDSC). In the background, the 'Generate Under Root' checkbox is checked. Below the toolchain menu, the 'Linker Settings' section shows 'Minimum Heap Size' set to 0x200 and 'Minimum Stack Size' set to 0x400. The 'Mcu and Firmware Package' section shows 'Mcu Reference' as STM32F407VGTx and 'Firmware Package Name and Version' as STM32Cube FW_F4 V1.16.0.



USB HID MOUSE

USB HID MOUSE - CODICE

- Nel file `usbd_hid.c` è definito il descrittore del device Mouse generato direttamente dal CUBE MX:

```
217 __ALIGN_BEGIN static uint8_t HID_MOUSE_ReportDesc[HID_MOUSE_REPORT_DESC_SIZE] __ALIGN_END =  
218 {  
219     0x05, 0x01,  
220     0x09, 0x02,  
221     0xA1, 0x01,  
222     0x09, 0x01,  
223  
224     0xA1, 0x00,  
225     0x05, 0x09,  
226     0x19, 0x01,  
227     0x29, 0x03,  
228  
229     0x15, 0x00,  
230     0x25, 0x01,  
231     0x95, 0x03,  
232     0x75, 0x01
```

USB HID MOUSE – CODICE (2)

- Definiamo due struct nel file main.c :
 - mouseHID_t: astrae il device mouse;
 - accelero_t: astrae le accelerazioni angolari lungo i tre assi.

```
typedef struct {  
    uint8_t buttons;    //!< Riporta la pressione del tasto sinistro del mouse  
    int8_t x;           //!< Riporta lo spostamento lungo l'asse x  
    int8_t y;           //!< Riporta lo spostamento lungo l'asse y  
    int8_t wheel;       //!< Riporta lo scroll del mouse  
}mouseHID_t;  
  
typedef struct{  
    int16_t asseX;       //!< Componente di accelerazione angolare lungo l'asse X  
    int16_t asseY;       //!< Componente di accelerazione angolare lungo l'asse Y  
    int16_t asseZ;       //!< Componente di accelerazione angolare lungo l'asse Z  
}accelero_t;
```

USB HID MOUSE – FUNZIONE LOOP

- Nella funzione `loop()` definiamo la logica del programma:
 - Per acquisire le componenti di accelerazione angolari lungo i tre assi utilizziamo la funzione `BSP_ACCELERO_GetXYZ(int16_t *pDataXYZ)` inclusa nella libreria `stm32f4_discovery_accelerometer.h`;
 - A causa dell'elevata sensibilità dell'accelerometro abbiamo definito un valore di soglia (pari a 64) per definire un range di valori in cui il cursore deve restare fermo;
 - I valori non filtrati vengono poi scalati di un valore di sensibilità s pari a 0.1 e assegnati alle componenti di spostamento del mouse lungo gli assi x ed y ;
 - La pressione del push button è associata al click sx del mouse;
 - Gli spostamenti acquisiti vengono visualizzati attraverso l'accensione dei led della board (ogni led è associato a una direzione);
 - Attraverso la funzione `USBD_HID_SendReport(...)` vengono inviati gli spostamenti acquisiti dal mouse e l'eventuale pressione del tasto.

USB HID MOUSE – FUNZIONE LOOP (2)

```
153 void loop(void){
154     /* Valuta la pressione del tasto blu (Button User)*/
155     if(BSP_PB_GetState(BUTTON_KEY) != GPIO_PIN_SET){
156         mouseHID.buttons=0x00;                // se il tasto blu non viene premuto, mouseHID.buttons = 0x00
157         for(int i=0; i<4;i++)                  // Reset di tutti i led
158             BSP_LED_Off(i);
159     }
160     else{
161         mouseHID.buttons=0x01;                // se il tasto blu viene premuto, mouseHID.buttons = 0x01 (click tasto sx del mouse)
162         for(int i=0;i<LEDn;i++)              // toggle su tutti i led per un debug visivo della pressione del tasto
163             BSP_LED_Toggle(i);
164     }
165
166     BSP_ACCELERO_GetXYZ((int16_t*)&accellero); // lettura delle componenti di accelerazione angolare lungo i 3 assi
167
168     value_x = (int8_t)(accellero.asseX*s);    // scaling della componente di accelerazione angolare lungo l'asse x
169     value_y = (int8_t)(accellero.asseY*s);    // scaling della componente di accelerazione angolare lungo l'asse y
170 }
```

USB HID MOUSE – FUNZIONE LOOP (3)

```
171 /* Valutazione delle soglie e aggiornamento dei campi x e y dell'oggetto mouseHID */
172 /* Accelerazione lungo l'asse y */
173 if (accelero.asseX>soglia){
174     BSP_LED_Toggle(LED5);
175     mouseHID.x = -value_x;           // opposto perchè la board è rivolta con il button user verso le dita
176 }
177 else if (accelero.asseX<-soglia){
178     BSP_LED_Toggle(LED4);
179     mouseHID.x = -value_x;           // opposto perchè la board è rivolta con il button user verso le dita
180 }
181
182 /* Accelerazione lungo l'asse x */
183 if (accelero.asseY>soglia){
184     BSP_LED_Toggle(LED3);
185     mouseHID.y = value_y;
186 }
187 else if (accelero.asseY<-soglia){
188     BSP_LED_Toggle(LED6);
189     mouseHID.y = value_y;
190 }
```

USB HID MOUSE – FUNZIONE LOOP (4)

```
191  /* Send HID Report */
192  USBD_HID_SendReport(&hUsbDeviceFS,(uint8_t*)&mouseHID,sizeof(mouseHID_t));
193
194  HAL_Delay(50);
195
196  /* Reset dei valori della struttura mouseHID*/
197  mouseHID.x = 0;
198  mouseHID.y = 0;
199  mouseHID.wheel = 0;
200 }
```



USB CUSTOM HID – KEYBOARD



USB CUSTOM HID KEYBOARD - CODICE

- Partire dalla stessa configurazione del CUBE MX generata precedentemente.
- Nel file `usbd_hid.c` sostituire `HID_MOUSE_ReportDesc` con il descrittore custom hid

```
266 /**
267  * @brief Funzione che implementa una custom human interface device configurata come tastiera.
268  */
269 __ALIGN_BEGIN static uint8_t HID_CUSTOM_ReportDesc[HID_CUSTOM_REPORT_DESC_SIZE] __ALIGN_END = {
270     // 41 bytes
271     0x05, 0x01,           // Usage Page (Generic Desktop Ctrls)
272     0x09, 0x06,           // Usage (Keyboard)
273     0xA1, 0x01,           // Collection (Application)
274     0x85, 0x01,           // Report ID (1)
275     0x05, 0x07,           // Usage Page (Kbrd/Keypad)
276     0x75, 0x01,           // Report Size (1)
277     0x95, 0x08,           // Report Count (8)
278     0x19, 0xE0,           // Usage Minimum (0xE0)
279     0x29, 0xE7,           // Usage Maximum (0xE7)
280     0x15, 0x00,           // Logical Minimum (0)
281     0x25, 0x01,           // Logical Maximum (1)
282     0x81, 0x02,           // Input (Data,Var,Abs,No Wrap,Linear,Preferred State,No Null Position)
283     0x95, 0x03,           // Report Count (3)
284     0x75, 0x08,           // Report Size (8)
285     0x15, 0x00,           // Logical Minimum (0)
286     0x25, 0x64,           // Logical Maximum (100)
287     0x05, 0x07,           // Usage Page (Kbrd/Keypad)
288     0x19, 0x00,           // Usage Minimum (0x00)
289     0x29, 0x65,           // Usage Maximum (0x65)
290     0x81, 0x00,           // Input (Data,Array,Abs,No Wrap,Linear,Preferred State,No Null Position)
291     0xC0,                // End Collection
292 };
293
```

USB CUSTOM HID KEYBOARD – CODICE (2)

- In USB_D_HID_CfgDesc, sempre nel file usbd_hid.c settare:
 - bInterfaceSubClass pari 0, la keyboard non rispetta le specifiche boot;
 - nInterfaceProtocol pari a 1 (keyboard)
- Sempre nello stesso file, tutti i riferimenti a mouse devono essere sostituiti con custom (e.g. HID_MOUSE_REPORT_DESC_SIZE diventa HID_CUSTOM_REPORT_DESC_SIZE).

```
152  /***** Descriptor of Joystick Mouse interface *****/
153  /* 09 */
154  0x09,      /*bLength: Interface Descriptor size*/
155  USB_DESC_TYPE_INTERFACE, /*bDescriptorType: Interface descriptor type*/
156  0x00,      /*bInterfaceNumber: Number of Interface*/
157  0x00,      /*bAlternateSetting: Alternate setting*/
158  0x01,      /*bNumEndpoints*/
159  0x03,      /*bInterfaceClass: HID*/
160  0x00,      /*bInterfaceSubClass : 1=BOOT, 0=no boot*/
161  0x01,      /*nInterfaceProtocol : 0=none, 1=keyboard, 2=mouse*/
162  0,         /*iInterface: Index of string descriptor*/
```

USB CUSTOM HID KEYBOARD – CODICE (3)

- Nel file `usbd_hid.h` effettuare le seguenti modifiche:
 - `HID_EPIN_SIZE` pari a 5;
 - `HID_CUSTOM_REPORT_DESC_SIZE` pari a 41, la lunghezza del nostro nuovo descrittore.

```
52 #define HID_EPIN_ADDR          0x81
53 #define HID_EPIN_SIZE          0x05
54
55 #define USB_HID_CONFIG_DESC_SIZ 34
56 #define USB_HID_DESC_SIZ       9
57 #define HID_CUSTOM_REPORT_DESC_SIZE 41
```

USB CUSTOM HID KEYBOARD – CODICE (4)

- Nel main.c andiamo a definire due struct:
 - keyboardHID_t: astrae la keyboard;
 - XYZ_t: astrae le accelerazioni angolari lungo i tre assi.

```
79 struct keyboardHID_t {  
80     uint8_t id;           //!< Id della Keyboard  
81     uint8_t modifiers;    //!< Valore di un tasto modificatore della KeyBoard  
82     uint8_t key1;         //!< Valore di un tasto della KeyBoard  
83     uint8_t key2;         //!< Valore di un tasto della KeyBoard  
84     uint8_t key3;         //!< Valore di un tasto della KeyBoard  
85 };  
86  
87 typedef struct {  
88     int16_t asse_x;        //!< Componente di accelerazione angolare lungo l'asse X  
89     int16_t asse_y;        //!< Componente di accelerazione angolare lungo l'asse Y  
90     int16_t asse_z;        //!< Componente di accelerazione angolare lungo l'asse Z  
91 }XYZ_t ;
```


USB CUSTOM HID KEYBOARD – CODICE (5)

- Nel main.c andiamo a definire i valori esadecimali associati ai tasti della keyboard

```
58 /* USB HID Usage Table */
59     /* USAGE NAME */           /*USAGE ID HEX */
60 #define USB_HID_KEY_LEFT_ARROW    0x50
61 #define USB_HID_KEY_RIGHT_ARROW   0x4F
62 #define USB_HID_KEY_DOWN_ARROW    0x51
63 #define USB_HID_KEY_UP_ARROW      0x52
64 #define USB_HID_KEY_SPACEBAR      0x2C
```

USB CUSTOM HID KEYBOARD – MAIN

- Nella funzione `main()` definiamo la logica del programma:
 - Per acquisire le componenti di accelerazione angolari lungo i tre assi utilizziamo la funzione `BSP_ACCELERO_GetXYZ(int16_t *pDataXYZ)` inclusa nella libreria `stm32f4_discovery_accelerometer.h`;
 - A causa dell'elevata sensibilità dell'accelerometro abbiamo definito un valore di soglia (pari a 64) per definire un range di valori in cui i valori non sono acquisiti;
 - I valori non filtrati vengono associati a quattro tasti differenti (nel nostro caso, le quattro frecce direzionali);
 - I valori acquisiti vengono visualizzati sui led della board (ogni led è associato a una direzione);
 - La pressione del push button è associata alla barra spaziatrice;
 - Attraverso la funzione `USBD_HID_SendReport(...)` vengono inviati i dati acquisiti dalla keyboard;
 - Ad ogni fine ciclo vengono resettati i valori acquisiti e rinviati i dati per emulare il rilascio del tasto.

USB CUSTOM HID KEYBOARD – MAIN (2)

```
140  while (1)
141  {
142      BSP_ACCELERO_GetXYZ((int16_t*)&XYZ);
143      value_x=XYZ.asse_x;
144      value_y=XYZ.asse_y;
145
146
147  /* Button User */
148      if(BSP_PB_GetState(BUTTON_KEY)!=GPIO_PIN_SET)
149          keyboardHID.key3=0;
150      else
151          keyboardHID.key3=USB_HID_KEY_SPACEBAR;
152
```

USB CUSTOM HID KEYBOARD – MAIN (3)

```
153  /* Accelerazione lungo l'asse y */
154      if (value_y>soglia){
155          BSP_LED_Toggle(LED3);
156          keyboardHID.key1 = USB_HID_KEY_UP_ARROW;
157      }
158      else if (value_y<-soglia){
159          BSP_LED_Toggle(LED6);
160          keyboardHID.key1 = USB_HID_KEY_DOWN_ARROW;
161      }
162
163  /* Accelerazione lungo l'asse x */
164      if (value_x>soglia){
165          BSP_LED_Toggle(LED5);
166          keyboardHID.key2 = USB_HID_KEY_RIGHT_ARROW;
167      }
168      else if (value_x<-soglia){
169          BSP_LED_Toggle(LED4);
170          keyboardHID.key2 = USB_HID_KEY_LEFT_ARROW;
171      }
```

USB CUSTOM HID KEYBOARD – MAIN (4)

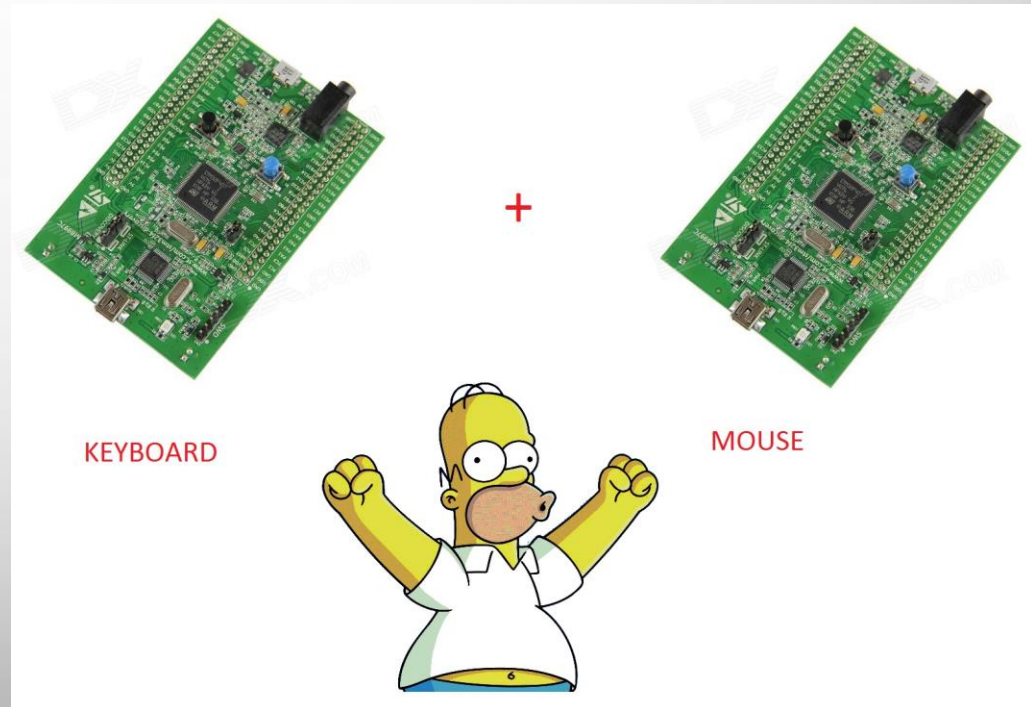
```
173  /* Invio Report */
174      USBD_HID_SendReport(&hUsbDeviceFS, (uint8_t*)&keyboardHID, sizeof(struct keyboardHID_t));
175      HAL_Delay(25);
176
177  /* Reset dei valori della struttura keyboardHID*/
178      keyboardHID.key1 = 0;
179      keyboardHID.key2 = 0;
180      keyboardHID.key3 = 0;
181
182  /* Invio Report */
183      USBD_HID_SendReport(&hUsbDeviceFS, &keyboardHID, sizeof(struct keyboardHID_t));
184
185  /* Reset Led */
186      for(int i=0; i<4;i++)
187          BSP_LED_Off(i);
188  /* Reset dei valori in attesa della prossima misurazione */
189      value_x=0;
190      value_y=0;
191      HAL_Delay(25);
192  }
193
```



APPLICAZIONE PRATICA DEI DEVICE REALIZZATI

APPLICAZIONE PRATICA

- Il mouse potrebbe essere impiegato come mirino, rotazione visuale, punta e clicca, ecc.
- La keyboard, essendo stata configurata con le frecce direzionali e la barra spaziatrice, potrebbe emulare una sorta di Wii controller con un solo tasto.



APPLICAZIONE PRATICA: FALLOUT 4

- La STM32F4 a sinistra è stata impiegata come keyboard (movimento personaggio più tasto «salta»), quella di destra come mouse (rotazione visuale con tasto «spara»).
- Di seguito il link del video realizzato della distruzione di Diamond City (si consiglia di abilitare God Mode tgm):
 - <https://m.youtube.com/watch?v=kPIZpKYVQI4>

